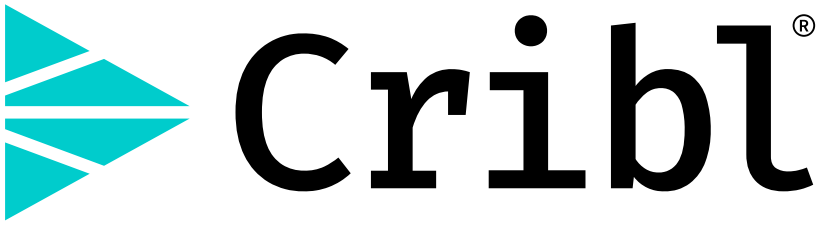


Cribl Stream Documentation Manual

Version: v4.17

Generated: 2026-03-11 07:55:49

Copyright (c) 2017- 2026 Cribl, Inc. All rights reserved.

1. About Cribl Stream 16
1.1. Cribl Stream Basic Concepts 18
1.2. Log in to Cribl 23
1.3. QuickConnect 25
1.4. Tutorials 31
1.4.1. Cribl Stream Tour 32
1.4.2. Get Started with Cribl Stream 51
1.4.3. Distributed Quick Start 71
1.5. Onboard Faster With Packs 97
1.5.1. Get Pre-Built Pack Content 98
1.5.2. Use Packs to Get Started With REST Collection 102
1.6. Process Data with AI Assistance 106
1.6.1. Create Pipelines With Cribl Copilot Editor 107
1.6.2. Convert Data into a Custom Schema 114
2. Set Up Cribl Stream 123
2.1. Reference Architectures 124
2.2. Cribl.Cloud vs. Self-Hosted 125
2.3. Learn About Connected Environments 130
2.3.1. How to Connect On-Prem Leaders to Cribl.Cloud 132
2.3.2. Simplify Billing with Universal Subscription 136
2.3.3. Data Payloads for Connected Environments 138
2.3.4. Send Data from On-Prem to Cribl.Cloud 143
2.4. Deployment Planning 146
2.4.1. Sizing and Scaling 149
2.4.2. OS Tuning for Large Deployments 161
2.4.3. Architectural Considerations 168
2.4.4. Ports 178
2.5. About Cribl.Cloud 183
2.5.1. Register Cribl.Cloud Organization 185
2.5.2. Manage Cribl.Cloud Organization 187
2.5.3. Cribl.Cloud Enterprise 189
2.5.4. Manage Cribl.Cloud Worker Groups 192
2.5.5. Learn About Connected Environments 197
2.5.5.1. How to Connect On-Prem Leaders to Cribl.Cloud 199
2.5.5.2. Simplify Billing with Universal Subscription 203
2.5.5.3. Data Payloads for Connected Environments 205

2.6. On-Prem Deployment 210
2.6.1. OS and System Requirements 211
2.6.2. Single-Instance/Basic Deployment 215
2.6.3. Run Cribl Stream 217
2.6.4. Configure Cribl Stream for Non-Root User 225
2.6.5. Distributed Deployment 227
2.6.5.1. Set Up Leader and Worker Nodes 234
2.6.5.2. Manage Worker Groups 239
2.6.5.3. Manage Workers 242
2.6.5.4. Map Workers to Worker Groups 249
2.6.5.5. Manage Config Bundles 251
2.6.5.6. Deploy Stream Configurations 255
2.6.5.7. Persistent Socket Connections 257
2.6.6. Splunk App Deployment 258
2.6.7. Bootstrap Workers from Leader 261
2.6.8. Leader High Availability/Failover 271
2.6.8.1. High Availability Requirements 273
2.6.8.2. Configure Standby Leader Nodes 277
2.6.8.3. Manage High Availability Deployment 280
2.6.9. Converting a Single Instance to Distributed Deployment 282
2.7. Orchestrated Deployment 284
2.7.1. Kubernetes/Helm Deployment 285
2.7.2. Kubernetes Worker Deployment 300
2.7.3. Considerations for Cribl Stream on Kubernetes 308
2.8. Connect Nodes to Leader Through Cribl Outpost 312
2.8.1. Set Up Cribl Outpost 314
2.8.2. Manage Cribl Outpost 317
2.9. Set Up Version Control 320
2.10. Docker Deployment 321
3. Administration 327
3.1. FinOps Center 328
3.2. Licensing 335
3.3. Configuration Management 342
3.3.1. Commit and Deploy Changes 343
3.3.2. Connect to External Git Repositories 348
3.3.3. Configure Worker Group Settings 363
3.3.4. GitOps 372
3.4. Workspaces 386

3.4.1. Configure Workspaces 391
3.5. Projects 394
3.5.1. Configure Projects 398
3.5.2. Configure Subscriptions 403
3.5.3. Add Users to Projects 406
3.6. Notifications 409
3.6.1. Email Notifications 416
3.6.2. Notification Targets 421
3.6.2.1. Webhook Notification Targets 423
3.6.2.2. PagerDuty Notification Targets 430
3.6.2.3. Slack Notification Targets 432
3.6.2.4. AWS SNS Notification Targets 434
3.6.2.5. Email Notification Targets 438
3.7. Customize the Interface 441
3.7.1. Custom Banners 442
3.7.2. Custom Login Page 443
3.8. Scripts 444
3.9. Upgrade Cribl Stream 446
3.9.1. Upgrade Outpost 461
3.10. Uninstall Cribl Stream 462
4. Secure Your Deployment 464
4.1. Secure Your Cribl.Cloud Deployment 465
4.2. Secure Your On-Prem/Hybrid Deployment 468
4.3. Secure Leader/Worker Nodes Communication 475
4.3.1. How to Secure the Auth Token for the Leader Node 480
4.3.2. TLS Defaults and System-wide Settings 483
4.3.3. Securing Cribl.Cloud with TLS and Mutual TLS 487
4.3.4. Configure mTLS Authentication On-Prem 490
4.3.5. Secure Leader Using an Existing TLS/SSL Certificate and Key 494
4.3.6. Configure TLS for API and UI Access 496
4.3.7. Secure Sources and Destinations with Certificates 498
4.3.8. Configure System Proxy 501
4.3.9. Configure SOCKS Proxy 507
4.4. Manage Secrets and Keys 512
4.4.1. Create and Manage Encryption Keys 513
4.4.2. Create and Manage Secrets in Cribl Stream 515
4.4.3. Configure KMS Providers 518
4.5. Compliance 526

4.5.1. FIPS Mode 527
4.5.2. Running Cribl Stream on a Hardened OS 534
4.5.3. SELinux (Enforcing Mode) Configuration 537
5. Monitor Health and Metrics 543
5.1. Internal Metrics 564
5.2. Internal Logs 576
6. Work with Data 582
6.1. Ingest and Inspect Data 583
6.1.1. Onboard Data 584
6.1.2. Use Datagens to Simulate Live Data 587
6.1.3. Create and Share Data Samples 592
6.1.4. Data Sample Use Cases 605
6.1.4.1. Ingest-Time Sampling 606
6.1.4.2. Sample Logs 608
6.1.4.3. Access Logs: Apache, ELB, CDN, S3 614
6.1.4.4. Firewall Logs: VPC Flow Logs, Cisco ASA 617
6.2. Event Data Structure and Flow 620
6.2.1. Event Model 621
6.2.2. Event Processing Order 624
6.2.3. Event Breakers 626
6.2.4. Event Breaker Types 635
6.2.4.1. Azure Virtual Network (VNet) Flow Event Breaker 636
6.2.4.2. CSV Event Breaker 640
6.2.4.3. File Header Event Breaker 642
6.2.4.4. JSON Array Event Breaker 645
6.2.4.5. JSON New Line Delimited Event Breaker 655
6.2.4.6. Regex Event Breaker 657
6.2.4.7. Timestamp Event Breaker 659
6.3. Routes 661
6.4. Pipelines 667
6.4.1. Validate Pipeline Logic Using Data Preview 675
6.5. Manage Metrics and High Cardinality 684
6.6. Enrich Data Using Lookups 694
6.6.1. About Lookups 695
6.6.2. Configure Lookups 702
6.6.3. Lookup Geographic Data from IP Addresses 708
6.6.4. Lookups as Filters for Masks 712
6.6.5. Lookups and Regex Magic 715

6.7. Transforming Data 725
6.7.1. Build Custom Logic to Route and Process Your Data 726
6.7.2. Code Function Examples 731
6.7.3. Ingest-Time Fields 742
6.7.4. Masking and Obfuscation 744
6.7.5. Encryption of Data in Motion 747
6.7.6. Reducing Windows XML Events 749
6.7.7. Regex Filtering 760
7. Functions 761
7.1. Aggregations 766
7.2. Aggregate Metrics 776
7.3. Auto Timestamp 785
7.4. CEF Serializer 790
7.5. Chain 792
7.6. Clone 794
7.7. Code 796
7.8. Comment 799
7.9. DNS Lookup 801
7.10. Drop 806
7.11. Drop Dimensions 807
7.12. Dynamic Sampling 812
7.13. Eval 815
7.14. Event Breaker Function 820
7.15. Flatten 823
7.16. Fold Keys 826
7.17. GeoIP 830
7.18. Grok 833
7.19. Guard 836
7.20. JSON Unroll 840
7.21. Lookup 843
7.22. Mask 849
7.23. Numerify 857
7.24. OTLP Logs 859
7.25. OTLP Metrics 868
7.26. OTLP Traces 885
7.27. Parser 891
7.28. Publish Metrics 903
7.29. Redis 909

7.30. Regex Extract 921
7.31. Regex Filter 925
7.32. Rename 926
7.33. Rollup Metrics 929
7.34. Sampling 931
7.35. Serialize 933
7.36. SNMP Trap Serialize 936
7.37. Suppress 939
7.38. Tee 942
7.39. Unroll 945
7.40. XML Unroll 947
7.41. Prometheus Publisher (deprecated) 951
7.42. Reverse DNS (deprecated) 953
7.43. Trim Timestamp (deprecated) 955
8. Integrations 958
8.1. Sources Overview 959
8.1.1. Collector Sources 972
8.1.1.1. Azure Blob Storage Collector 979
8.1.1.2. Cribl Lake Collector 988
8.1.1.3. Database Collector 991
8.1.1.4. File System/NFS Collector 1002
8.1.1.5. Google Cloud Storage Collector 1007
8.1.1.6. Health Check Collector 1014
8.1.1.7. REST / API Endpoint Collector 1028
8.1.1.8. S3 Collector 1059
8.1.1.9. Script Collector 1070
8.1.1.10. Splunk Search Collector 1076
8.1.1.11. Manage Collector State in Database and REST API Collectors 1084
8.1.1.12. Schedule and Run Collector Jobs 1087
8.1.1.13. Job Limits 1097
8.1.1.14. Using Collectors 1100
8.1.1.14.1. Configure a Database Connection 1101
8.1.1.14.2. Using S3 Storage and Replay 1108
8.1.1.14.3. Using REST/API Collectors 1115
8.1.1.14.4. Generate HMAC Functions for REST Collector Requests 1146
8.1.1.14.5. Lacework API Collection 1149
8.1.1.14.6. Microsoft Graph API Collection 1156
8.1.1.14.7. ServiceNow API Collection 1162

8.1.1.14.8. Creating a Custom Collector 1168
8.1.2. Amazon 1193
8.1.2.1. Amazon Data Firehose Source 1194
8.1.2.2. Amazon Kinesis Data Streams Source 1203
8.1.2.3. Amazon S3 Source 1211
8.1.2.4. Amazon Security Lake Source 1225
8.1.2.5. Amazon SQS Source 1234
8.1.3. Azure 1240
8.1.3.1. Azure Blob Storage Source 1241
8.1.3.2. Azure Event Hubs Source 1254
8.1.4. Google Cloud 1263
8.1.4.1. Google Cloud Pub/Sub Source 1264
8.1.5. Kafka 1269
8.1.5.1. Kafka Source 1270
8.1.5.2. Confluent Cloud Source 1281
8.1.5.3. Amazon MSK Source 1292
8.1.6. Office 365 1302
8.1.6.1. Office 365 Activity Source 1303
8.1.6.2. Office 365 Message Trace Source 1313
8.1.6.3. Microsoft Graph Source 1322
8.1.6.4. Office 365 Services Source 1330
8.1.7. Prometheus 1337
8.1.7.1. Prometheus Scraper Source 1338
8.1.7.2. Prometheus Remote Write Source in Cribl Stream 1346
8.1.7.3. Grafana Source 1355
8.1.7.4. Loki Source 1364
8.1.8. Splunk 1372
8.1.8.1. Splunk HEC Source in Cribl Stream 1373
8.1.8.2. Splunk Search Source 1389
8.1.8.3. Splunk TCP Source in Cribl Stream 1395
8.1.9. Internal 1405
8.1.9.1. Cribl Internal Source 1406
8.1.9.2. Cribl HTTP Source in Cribl Stream 1411
8.1.9.3. Cribl TCP Source 1419
8.1.10. System 1426
8.1.10.1. Datagen Source 1427
8.1.10.2. Exec Source 1430
8.1.10.3. File Monitor Source 1437

8.1.10.4. Journal Files Source 1448
8.1.10.5. System Metrics Source 1452
8.1.10.6. System State Source 1459
8.1.11. AppScope Source 1471
8.1.12. Cloudflare Source 1481
8.1.13. CrowdStrike FDR Source 1493
8.1.14. Datadog Agent Source in Cribl Stream 1500
8.1.15. Elasticsearch API Source in Cribl Stream 1512
8.1.16. HTTP/S (Bulk API) Source 1523
8.1.17. Raw HTTP/S Source in Cribl Stream 1535
8.1.18. Metrics Source 1545
8.1.19. Model Driven Telemetry Source 1553
8.1.20. NetFlow & IPFIX Source 1559
8.1.21. OpenAI Source 1573
8.1.22. OpenTelemetry (OTel) Source in Cribl Stream 1581
8.1.23. SNMP Trap Source 1592
8.1.24. Syslog Source in Cribl Stream 1599
8.1.25. TCP JSON Source in Cribl Stream 1612
8.1.26. TCP (Raw) Source in Cribl Stream 1620
8.1.27. UDP (Raw) Source 1628
8.1.28. Windows Event Forwarder Source in Cribl Stream 1635
8.1.28.1. Client Certificate Authentication for Windows Event Forwarder 1652
8.1.28.2. Kerberos Authentication for Windows Event Forwarder 1656
8.1.29. Wiz 1661
8.1.29.1. Wiz API 1662
8.1.29.2. Wiz Webhook in Cribl Stream 1670
8.1.30. Zscaler Cloud NSS Source 1680
8.2. Destinations 1691
8.2.1. Managing Destinations 1699
8.2.2. Amazon 1704
8.2.2.1. Amazon CloudWatch Logs Destination 1705
8.2.2.2. Amazon Kinesis Data Streams Destination 1712
8.2.2.3. Amazon S3 Compatible Stores Destination 1721
8.2.2.4. Amazon SQS Destination 1732
8.2.3. Azure 1739
8.2.3.1. Azure Blob Storage Destination 1740
8.2.3.2. Azure Data Explorer Destination 1748
8.2.3.3. Azure Event Hubs Destination 1762

8.2.3.4. Azure Monitor Logs Destination 1771
8.2.3.5. Microsoft Sentinel Destination 1780
8.2.4. Data Lakes 1792
8.2.4.1. Cribl Lake Destination 1793
8.2.4.2. Amazon S3 Destination 1797
8.2.4.3. Amazon Security Lake Destination 1803
8.2.5. Elastic 1811
8.2.5.1. Elasticsearch Destination 1812
8.2.5.2. Elastic Cloud Destination 1822
8.2.6. Google Cloud 1830
8.2.6.1. Google Security Operations (SecOps) Destination 1831
8.2.6.2. Google Cloud Chronicle API Destination 1841
8.2.6.3. Google Cloud Logging Destination 1850
8.2.6.4. Google Cloud Pub/Sub Destination 1857
8.2.6.5. Google Cloud Storage Destination 1864
8.2.7. Kafka 1871
8.2.7.1. Kafka Destination 1872
8.2.7.2. Confluent Cloud Destination 1883
8.2.7.3. Amazon MSK Destination 1894
8.2.8. Metrics 1905
8.2.8.1. Graphite Destination 1906
8.2.8.2. StatsD Destination 1911
8.2.8.3. StatsD Extended Destination 1916
8.2.9. New Relic Ingest 1921
8.2.9.1. New Relic Events Destination 1922
8.2.9.2. New Relic Logs & Metrics Destination 1930
8.2.10. Prometheus 1939
8.2.10.1. Prometheus Destination 1940
8.2.10.2. Grafana Cloud Destination 1948
8.2.10.3. Loki Destination 1958
8.2.11. Splunk 1967
8.2.11.1. Splunk HEC Destination 1968
8.2.11.2. Splunk Single Instance Destination 1980
8.2.11.3. Splunk Load Balanced Destination 1988
8.2.12. Internal 2000
8.2.12.1. Cribl HTTP Destination 2001
8.2.12.2. Cribl TCP Destination 2012
8.2.12.3. Cribl Search Destination 2021

8.2.12.4. Default Destination 2029
8.2.12.5. DevNull Destination 2030
8.2.12.6. Output Router Destination 2031
8.2.13. CrowdStrike Falcon LogScale Destination 2034
8.2.14. ClickHouse Destination 2042
8.2.15. Cloudflare R2 Destination 2052
8.2.16. Cortex XSIAM Destination 2059
8.2.17. CrowdStrike Falcon Next-Gen SIEM Destination 2068
8.2.18. Databricks Destination 2077
8.2.19. Datadog Destination 2083
8.2.20. Dynatrace HTTP Destination 2093
8.2.21. Dynatrace OTLP Destination 2102
8.2.22. Exabeam Security Operations Platform Destination 2111
8.2.23. Microsoft Fabric Real-Time Intelligence 2117
8.2.24. Filesystem/NFS Destination 2123
8.2.25. Honeycomb Destination 2129
8.2.26. InfluxDB Destination 2136
8.2.27. MinIO Destination 2144
8.2.28. NetFlow Destination 2153
8.2.29. OpenTelemetry (OTel) Destination 2157
8.2.30. SentinelOne AI SIEM Destination 2167
8.2.31. SentinelOne DataSet Destination 2176
8.2.32. ServiceNow Cloud Observability 2185
8.2.33. SignalFx Destination 2195
8.2.34. SNMP Trap Destination 2203
8.2.35. Sumo Logic Destination 2205
8.2.36. Syslog Destination 2215
8.2.37. TCP JSON Destination 2229
8.2.38. Wavefront Destination 2237
8.2.39. Webhook Destination 2244
8.2.40. Wiz Defend 2262
8.3. Manage Backpressure 2270
8.3.1. About Persistent Queues 2271
8.3.2. Optimize Source Persistent Queues 2276
8.3.3. Optimize Destination Persistent Queues 2287
8.3.4. About Destination Backpressure Triggers 2300
8.4. About Load Balancing 2306
8.5. Configure Host Metadata 2309

8.6. Cribl Edge to Cribl Stream 2311
8.7. Transfer Data Between Workspaces or Environments 2315
8.8. Integrating with Other Services 2318
8.8.1. Amazon 2320
8.8.1.1. AWS Cross-Account Data Collection 2321
8.8.1.2. Integrate Cribl Stream with AWS Security Hub 2327
8.8.1.3. Amazon S3 Better Practices 2332
8.8.1.4. Amazon Security Lake Integration 2357
8.8.1.5. Collecting Logs from Amazon ECS Containers 2366
8.8.2. Azure 2368
8.8.2.1. Prepare the Azure Workspace for Cribl Integrations 2369
8.8.2.2. Azure Event Hubs Integrations 2376
8.8.2.3. Microsoft Sentinel SIEM Integration 2381
8.8.3. Splunk 2384
8.8.3.1. Splunk Cloud Platform and BYOL Integrations 2385
8.8.3.2. Switch Cribl Stream Destinations from S2S to Splunk HEC 2386
8.8.3.3. Splunk to Elasticsearch 2390
8.8.3.4. Splunk to Exabeam 2397
8.8.3.5. Splunk Stream to Cribl Stream 2405
8.8.3.6. Decryption of Data in Splunk 2409
8.8.3.7. Stream-to-Splunk Encryption 2412
8.8.4. Syslog 2417
8.8.4.1. Syslog Best Practices 2418
8.8.4.2. Syslog TLS to Cribl.Cloud (Palo Alto Example) 2433
8.8.5. Configure Upstream Logging Agents 2438
8.8.6. BigPanda/Webhook Integration 2450
8.8.7. Gigamon to Cribl Stream 2454
8.8.8. System Metrics to Grafana 2457
8.8.9. Kafka Authentication with Kerberos 2462
8.8.10. Moogsoft/Webhook Integration 2464
8.8.11. Nightfall Integration 2470
8.8.12. Managing QRadar Licenses 2474
8.8.13. Slack/Webhook Integration 2478
8.8.14. Zscaler NSS Virtual Machine 2480
8.8.15. Tanium to Cribl Stream 2487
9. Packs 2494
9.1. Pack-Based Configuration Management 2508
9.1.1. Plan a Pack-Based Strategy 2512

9.1.2. Create a Pack 2520
9.1.3. Version Control Packs in External Git Repos 2528
9.2. Packs Publication Standards 2532
10. Reference 2543
10.1. Knowledge Objects 2544
10.1.1. Variables Library 2545
10.1.2. Regexes 2552
10.1.3. JSON Schemas 2556
10.1.4. Parquet Schemas 2559
10.1.5. AppScope Configs 2568
10.2. Cribl Expressions 2576
10.2.1. C.Crypto - Data Encryption and Decryption 2577
10.2.2. C.Decode and C.Encode - Encoding and Decoding 2580
10.2.3. C.Lookup - Inline Lookup Methods 2588
10.2.4. C.Mask - Data Masking Methods 2592
10.2.5. C.Net - Network Methods 2603
10.2.6. C.Text - Text Methods 2610
10.2.7. C.Time - Time Methods 2614
10.2.8. Miscellaneous Expression Methods 2621
10.3. CLI Reference 2629
10.3.1. auth 2633
10.3.2. boot-start 2638
10.3.3. cloud-workspace 2640
10.3.4. decrypt 2642
10.3.5. diag 2643
10.3.6. encrypt 2647
10.3.7. git 2648
10.3.8. help 2651
10.3.9. keys 2653
10.3.10. limits 2655
10.3.11. mode-master 2656
10.3.12. mode-single 2658
10.3.13. mode-edge 2659
10.3.14. mode-worker 2660
10.3.15. mode-managed-edge 2662
10.3.16. mode-outpost 2664
10.3.17. nc 2666
10.3.18. node 2668

10.3.19. pack 2670
10.3.20. parquet 2674
10.3.21. pipe 2678
10.3.22. pq 2680
10.3.23. reload 2684
10.3.24. restart 2685
10.3.25. scope 2686
10.3.26. start 2687
10.3.27. status 2688
10.3.28. stop 2689
10.3.29. vars 2690
10.3.30. version 2693
10.4. Environment Variables 2694
10.5. Configuration Files 2702
10.5.1. breakers.yml 2708
10.5.2. certificates.yml 2712
10.5.3. cribl.yml 2713
10.5.4. groups.yml 2724
10.5.5. inputs.yml 2727
10.5.6. instance.yml 2843
10.5.7. iometrics.yml 2848
10.5.8. jobs.yml 2849
10.5.9. job-limits.yml 2877
10.5.10. leader.yml 2880
10.5.11. limits.yml 2883
10.5.12. logger.yml 2887
10.5.13. mappings.yml and fleet-mappings.yml 2888
10.5.14. messages.yml 2889
10.5.15. notifications.yml 2890
10.5.16. outpost.yml 2895
10.5.17. outputs.yml 2898
10.5.18. parsers.yml 3013
10.5.19. persistent-queue.yml 3016
10.5.20. regexes.yml 3017
10.5.21. samples.yml 3018
10.5.22. schemas.yml 3021
10.5.23. scripts.yml 3022
10.5.24. service.yml 3023

10.5.25. vars.yml 3026
11. Troubleshoot Cribl Stream 3027
11.1. Known Issues 3028
11.2. Diagnose Issues 3029
11.3. Common Errors and Warnings 3037
11.4. Git Push Errors 3044
11.5. Git Remote Repos & Trusted CAs 3046
11.6. Sample Logs for Login Scenarios 3049
12. CrowdStream 3052
13. Third-Party Credits 3068

1. About Cribl Stream

What Is Cribl Stream?

Cribl Stream helps you process machine data - logs, instrumentation data, application data, metrics, etc. - in real time, and deliver them to your analysis platform of choice. It allows you to:

- Add context to your data, by enriching it with information from external data sources.
- Help secure your data, by redacting, obfuscating, or encrypting sensitive fields.
- Optimize your data, per your performance and cost requirements.

Sources, Cribl Stream, Destinations

Cribl Stream ships in a single, no-dependencies package. It provides a refreshing and modern interface for working with and transforming your data. It scales with - and works inline with - your existing infrastructure, and is transparent to your applications.

Conceptual Overview of Cribl Stream

This two-minute video provides a conceptual overview of Cribl Stream capabilities.

Who Is Cribl Stream For?

Cribl Stream is built for administrators, managers, and users of operational/DevOps and security intelligence products and services.

How Can I Get Started with Cribl Stream?

You can quickly create a Cribl.Cloud account to get Cribl Stream running on infrastructure that Cribl manages on your behalf. Or you can deploy Cribl Stream software on your own infrastructure, Kubernetes Pods, or containers. For differences between these options, see these topics:

- Why Use Cloud Deployment?
- Cribl Cloud vs. Self-Hosted

1.1. Cribl Stream Basic Concepts

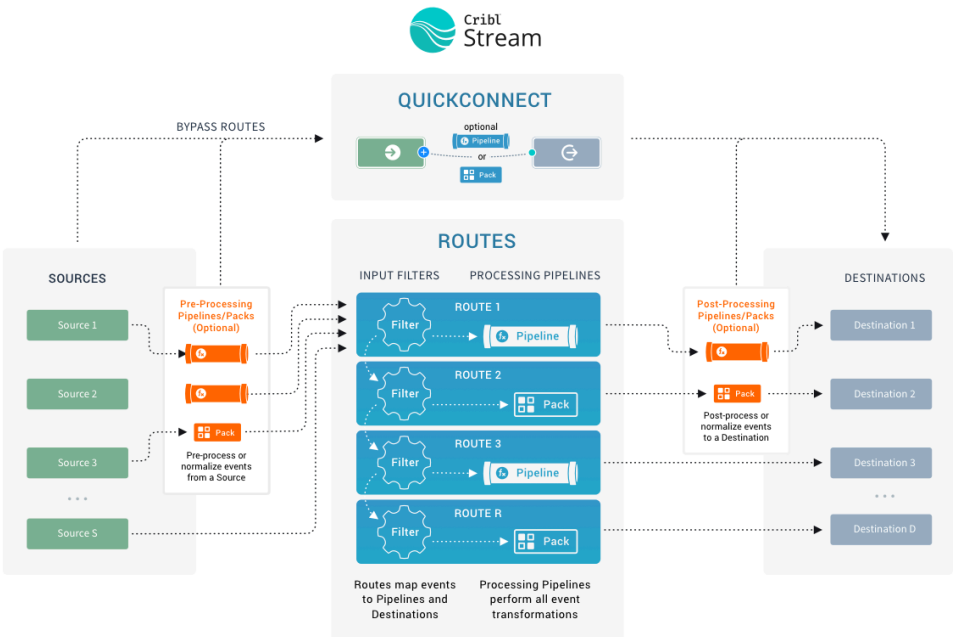
Notable features and concepts to get a fundamental understanding of Cribl Stream

---

As we describe features and concepts, it helps to have a mental model of Cribl Stream as a system that receives events from various sources, processes them, and then sends them to one or more destinations.

Sources, Cribl Stream, Destinations

Let's zoom in on the center of the above diagram, to take a closer look at the processing and transformation options that Cribl Stream provides internally. The basic interface concepts to work with are Sources, which collect data; and Routes, which manage data flowing through Pipelines, which consist of Functions.



Routes, Pipelines, Functions

Sources

Sources are configurations that enable Cribl Stream to receive data from remote senders (such as Splunk, TCP, Syslog) or to collect data from remote file stores or the local machine.

Destinations

Destinations are downstream receivers that receive data from Cribl Stream after data processing and transformation. Destinations are where Cribl Stream delivers your data after processing and transformation. Cribl Stream can send data to many different Destinations, including Splunk, Kafka, Kinesis, InfluxDB, Snowflake, Databricks, TCP JSON, and others.

Routes

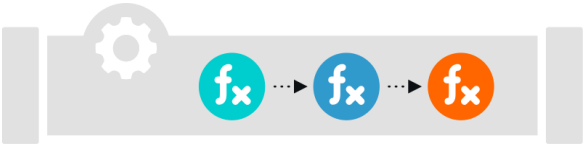
Routes evaluate incoming events against **filter** expressions to find the appropriate Pipeline to send them to. Routes are **evaluated in order**. Each Route can be associated **with only one** Pipeline and one output (configured as a Cribl Stream **Destination**).

By default, the **Final** toggle is on when each Route is created, so a Route-Pipeline-Destination set will consume events that match its filter, and that's that.

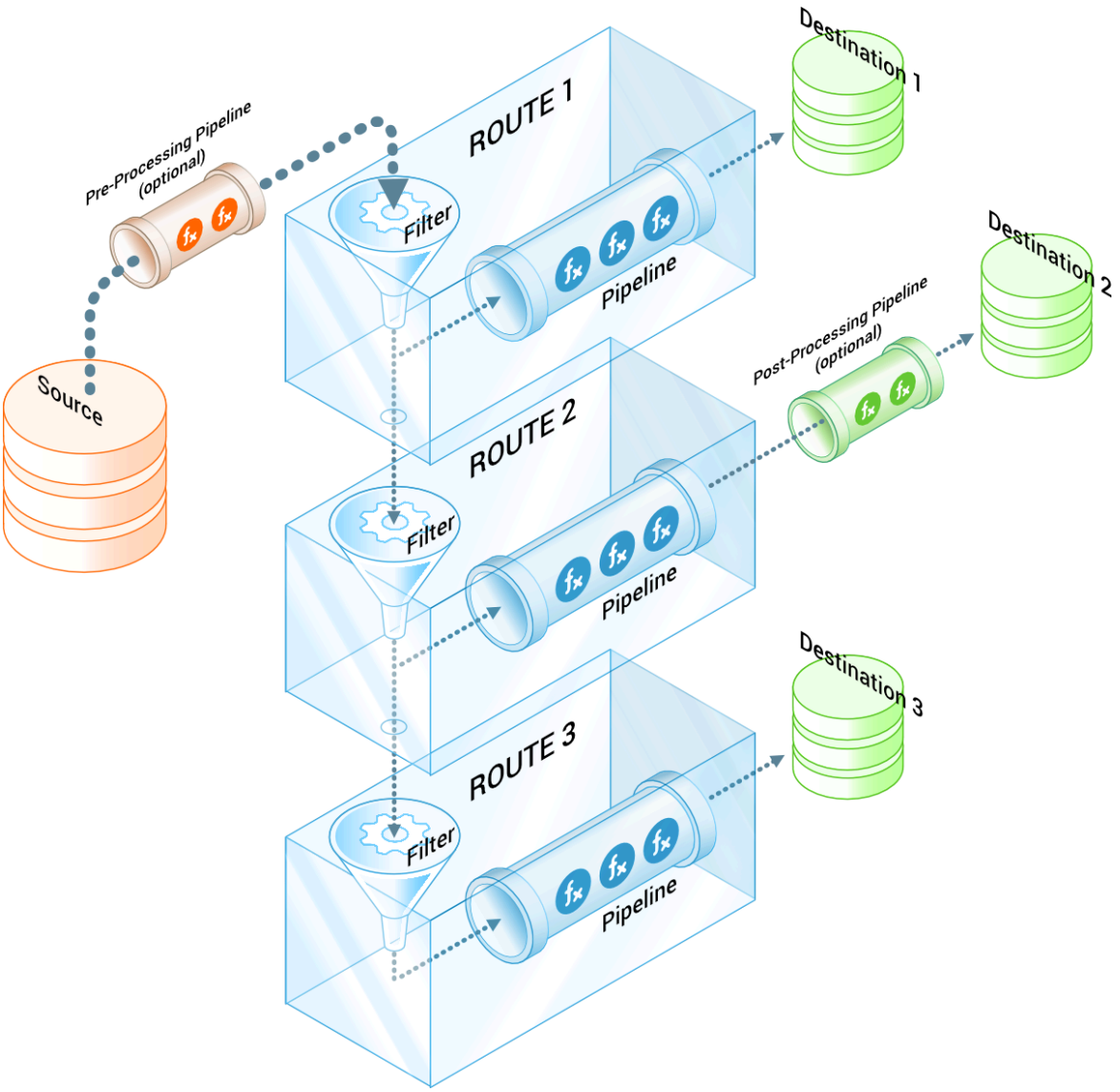
However, if you toggle **Final** off, one or more event **clones** will be sent down the associated Pipeline, while the original event continues down Cribl Stream's Routing table to be evaluated against other configured Routes. This is very useful in cases where the same set of events needs to be processed in multiple ways, and delivered to different destinations. For more details, see Routes.

Pipelines

A series of Functions is called a Pipeline, and the order in which you specify the Functions determines their execution order. Events are delivered to the beginning of a Pipeline by a Route, and as they're processed by a Function, the events are passed to the next Function down the line.



Pipelines attached to Routes are called **processing Pipelines**. You can optionally attach **pre-processing Pipelines** to individual Cribl Stream Sources, and attach **post-processing Pipelines** to Cribl Stream Destinations. All Pipelines are configured through the same UI - these three designations are determined only by a Pipeline's placement in Cribl Stream's data flow.

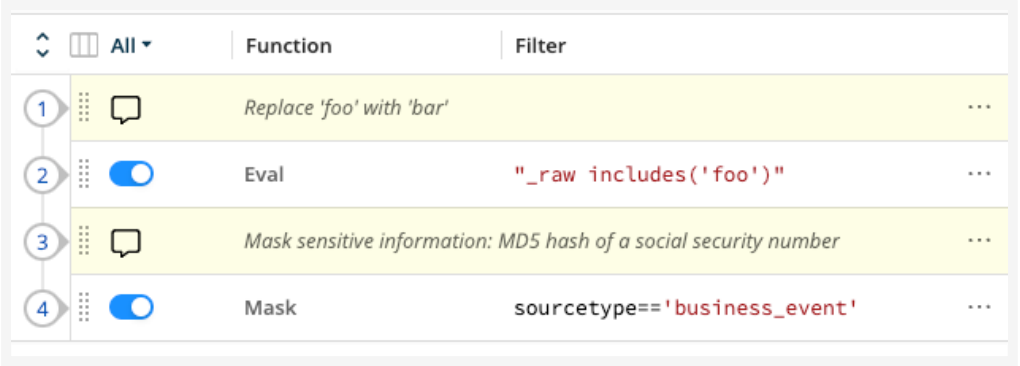


Pipelines categorized by position

Events only move forward - toward the end of a Pipeline, and eventually out of the system. For more details, see Pipelines.

Functions

At its core, a **Function** is a piece of code that executes on an event, and that encapsulates the smallest amount of processing that can happen to that event. For instance, a very simple Function can be one that replaces the term `foo` with `bar` on each event. Another one can hash or encrypt `bar`. Yet another function can add a field - say, `dc=jfk-42` - to any event with `source=*us-nyc-application.log`.



Functions stacked in a Pipeline

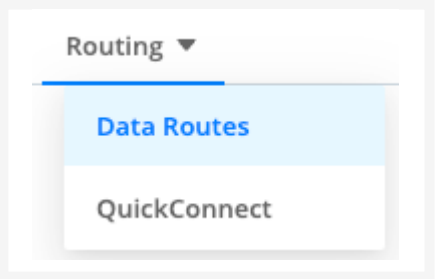
Functions process each event that passes through them. To help improve performance, Functions can optionally be configured with filters, to limit their processing scope to matching events only. For more details, see Functions.

QuickConnect

QuickConnect is a visual interface for setting up data flow through your Cribl Stream deployment. You can quickly drag and drop connections between Sources and Destinations, optionally including or excluding Pipelines or Packs.

The only major constraint is that QuickConnect completely bypasses Routes. So QuickConnect configurations have no Routing table, and no conditional cloning, cascading, or routing of data - every QuickConnect connection is parallel and independent.

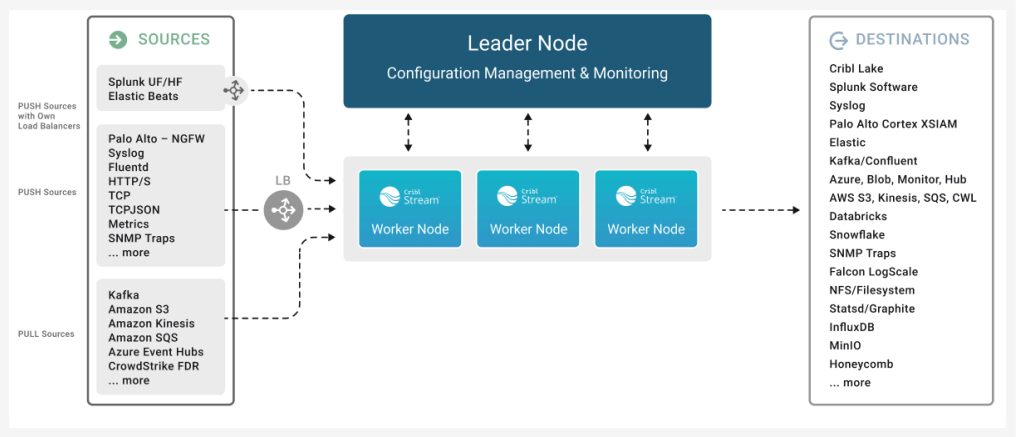
Use the **Routing** menu to toggle between the **Data Routes** and **QuickConnect** interfaces. (The **Routing** menu appears at the top of the **Worker Groups** page in a Distributed deployment, and at the top of the Stream Home page in a Single-instance deployment.)



Select Data Routes to use the Routing table, or QuickConnect to bypass Routes

A Scalable Model

You can scale Cribl Stream up to meet enterprise needs in a distributed deployment. Here, multiple Cribl Stream Workers (instances) share the processing load. But as you can see in the preview schematic below, even complex deployments follow the same basic model outlined above.



Distributed deployment architecture

Shared-Nothing Architecture

1.2. Log In To Cribl

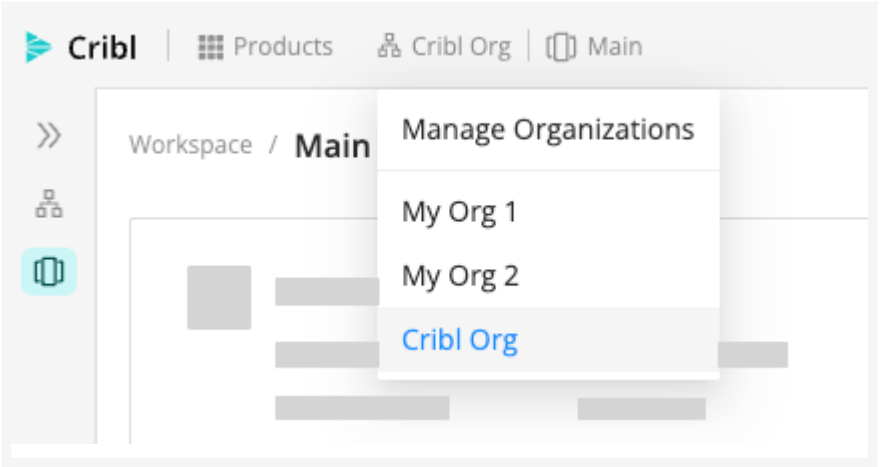
When you first log in to a Cribl instance, you will be required to configure your account and authenticate.

Depending on how your Organization is configured, you may need to either authenticate via your SSO provider (such as Okta, Entra ID, or Ping Identity), or create a password for use with your email address.

When creating a new username and password, you will be prompted for account information, such as name, company, or phone number.

Switch Between Organizations

When you are a member of multiple Organizations, you can switch between them on the top bar. Select your current Organization's name on the bar to open a drop-down with all Organizations you have access to.



Switching between Organizations

Organizations with Different Authentication Methods

If you switch between Organizations that use different authentication methods (SSO versus username/password), you will be asked to re-authenticate with the appropriate credentials.

If the main Organization uses SSO, and you are switching to an Organization that uses username/password authentication, you will need to authenticate with a password. If you are accepting an invitation to the latter Organization, you will be prompted to create a password and go through a verification process.

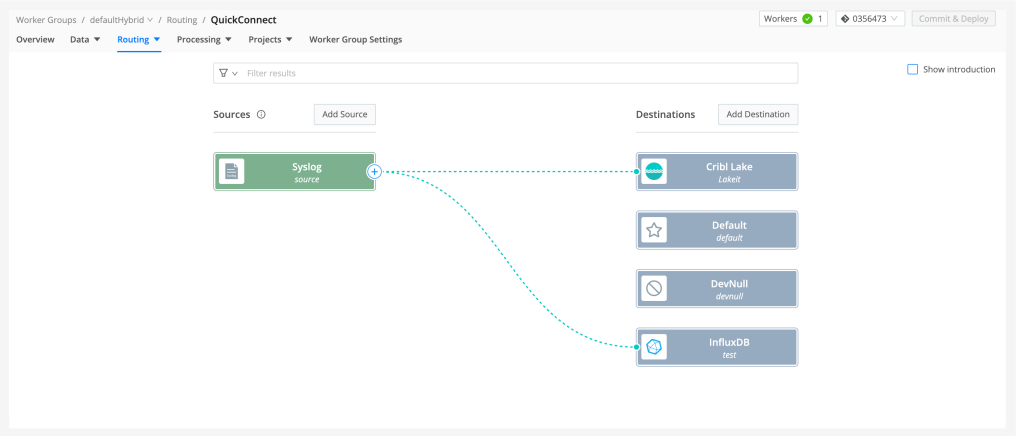
If the main Organization uses username/password authentication, and you are switching to an SSO-based Organization, you will need to authenticate via SSO.

1.3. QuickConnect

The QuickConnect visual rapid-development UI enables you to visually connect Cribl Stream inputs (Sources) to outputs (Destinations) through simple drag-and-drop.

You can insert Pipelines or Packs into the connections, to take advantage of Cribl Stream's full range of data-transformation Functions. Or you can omit these processing stages entirely, to send incoming data directly to Destinations - with minimal configuration fuss.

QuickConnect was designed as a simple, quick way to prototype, test, and send data flowing through Cribl Stream. But it's also entirely suitable for configuring a production deployment, if your needs are restricted to sending data through parallel paths, and you don't need to use Routes. (See QuickConnect Versus Routes to help you clarify that choice.)

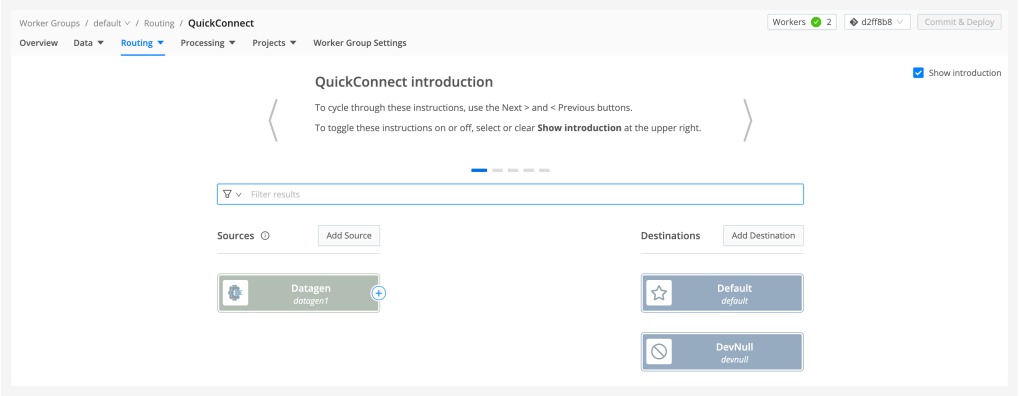


QuickConnect UI

You also have the option to start with simple direct connections, and later add Pipelines and/or Packs processing power at up to three stages of each connection.

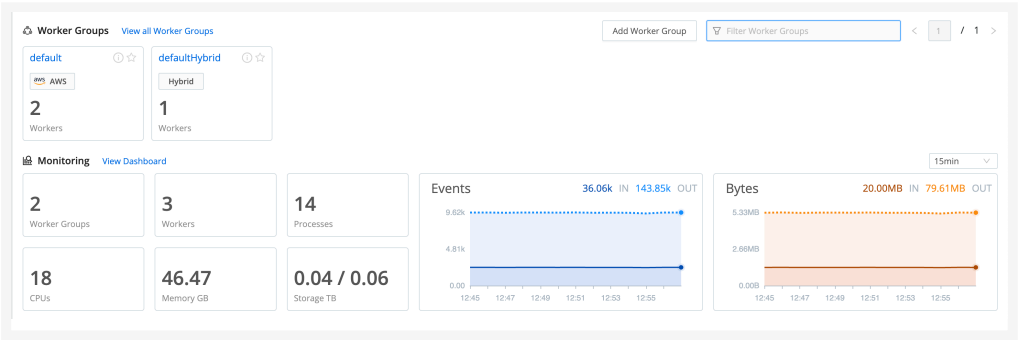
Initiate QuickConnect

When you display the home page of a Cribl Stream single instance or Worker Group, you'll see tiles that prompt you to choose between the **QuickConnect** versus **Route** configuration. Select the left tile to start using QuickConnect.



QuickConnect versus Routing

To display this landing page in a Distributed deployment, you must first select a Worker Group. In the sidebar, select **Worker Groups**, and then select the `default` or a different Worker Group. Or, select a Worker Group's tile on the Distributed landing page, as shown here.



Selecting a Worker Group

You can switch to the Routes configuration at any time from the **Worker Groups** submenu by selecting **Routing** > **Data Routes**. To toggle back, select **Routing** > **QuickConnect**.

QuickConnect Versus Routes

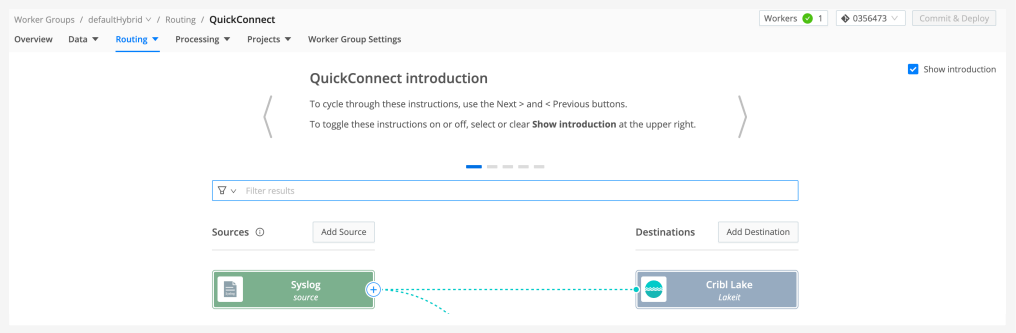
QuickConnect provides access to most of Cribl Stream's configuration options, with a few restrictions (in exchange for the simplified visual interface):

- QuickConnect completely bypasses Routes - which is why every QuickConnect path is independent and parallel. There is no Route-level data filtering, and you forego the Cribl Stream Routing table's options to clone and cascade data across Pipelines.
- Most Sources are available to fully configure. However, you'll need to use **Routing** > **Data Routes** to configure Collector Sources.
- Each supported Source's config can be created in either QuickConnect or Routes, and will reside only in that context. But you can switch a config between the two contexts, or you can create an identical config on the opposite side.

- All Destinations are available to fully configure.
- Everything you configure in QuickConnect exists separately from everything you configure in Data Routes. (But remember that you can use the **Routing** menu to toggle between contexts, and then replicate a QuickConnect configuration on the Data Routes side, or vice versa.)

Use QuickConnect

QuickConnect is designed to be nearly self-documenting, with **Introduction** help text available at the top of the UI. Here's a little extra help on help, keyed to the orange, numbered callouts in the screenshot:



QuickConnect Introduction/help carousel

1. Use the **Show introduction** check box at the upper right to toggle the help text on and off.
2. Use the dashed buttons to advance or rewind the help text's carousel (whose position is independent of your connection state).

Enable Sources and Destinations

When you add Sources and Destinations, their configuration options will open in a drawer. Fill in a unique **Input ID**, and any other required fields marked by an asterisk (*). Then select **Save**.

Switch Contexts

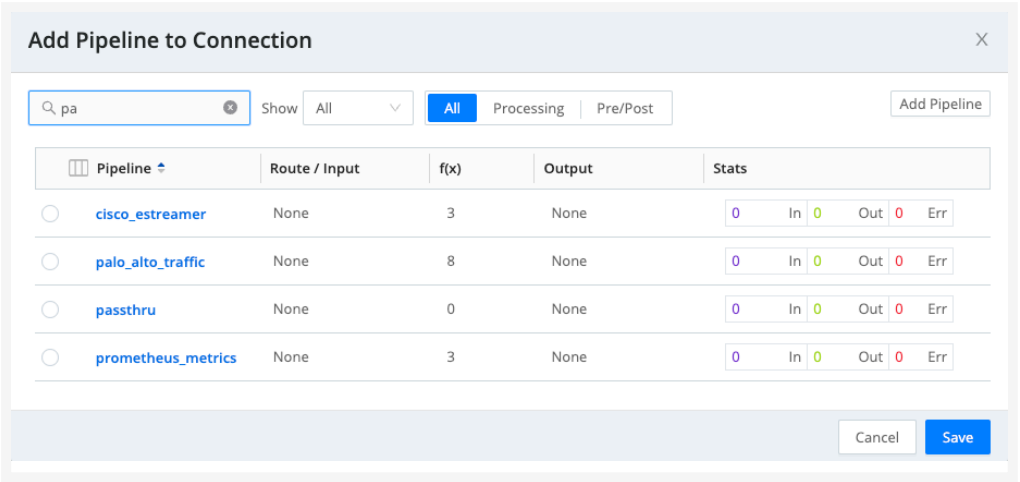
If you switch an **existing** Source to QuickConnect - for example, one of the preconfigured Sources that ship with Cribl Stream - you'll see a confirmation message. This is due to QuickConnect's independence from the Routes interface - you need to confirm your choice to move the existing configuration from the Data Routes world to the QuickConnect world.

If you switch an existing Source config to QuickConnect, its data will no longer flow through its configured Route.

You can also go the opposite way: move a Source originally configured in QuickConnect to the Routing interface. In the Source's drawer or config modal, select the **Connected Destinations** left tab, and then select the resulting tab's **Send to Routes** button. Here again, you'll need to confirm your choice before proceeding.

Select a Pipeline or Pack

When you add or modify a connection line, the **Connection Configuration** modal will prompt you to select a **Passthru**, **Pipeline**, or **Pack** connection. Selecting either **Pipeline** or **Pack** will open an **Add…** modal like this:



Adding a Pipeline

Here, you're using the radio buttons at left to select among already-configured Pipelines or Packs. **Resist the temptation to select a bold blue Pipeline or Pack name.** (Doing so will open the Pipeline's or Pack's config in a new modal, which might offer more configuration than you bargained for.)

Instead, select the radio button to the left of the Pipeline or Pack you want to select. (Or select some black text around the center of your desired Pipeline's/Pack's row, which will fill in its radio button.) Then click **Save** to confirm your choice and close the modal.

If you've attached Tags to your Pipelines or Packs, you can use the filter field to search against them. Use the format: **TAGS/tag_name**.

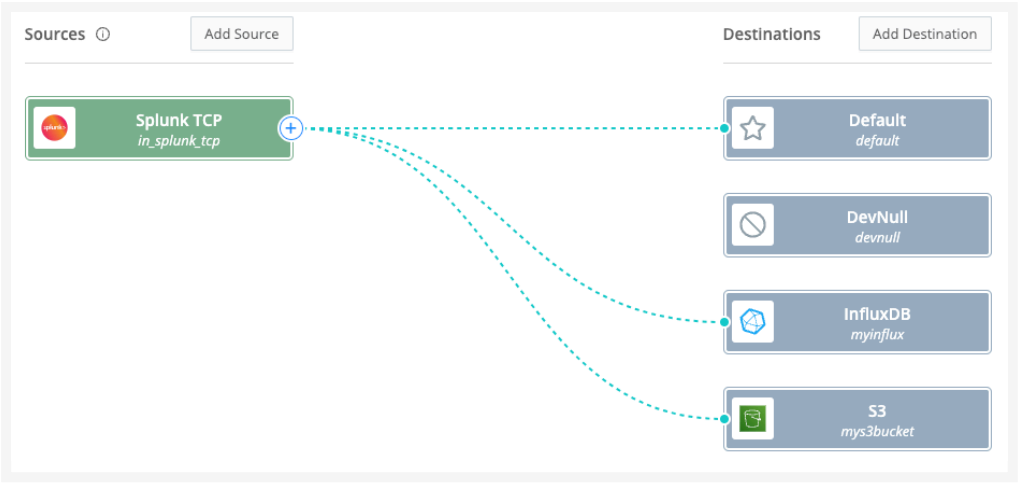
Edit a Pipeline

If you want to modify a Pipeline, this is where you do want to select its blue link in the **Add Pipeline to Connection** modal. For the options available here, see Pipelines.

Multiple Connections

You can drag multiple connection lines between a given Source/Destination pair. (You might do this, for example, to configure parallel Pipelines to handle different data types.)

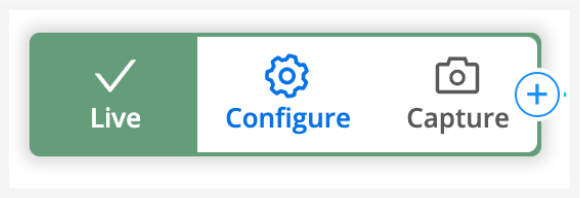
Also, you can connect one Source to multiple Destinations, or multiple Sources to one Destination. Let's isolate examples of both, from the screen capture we started with:



Multiple connections out, multiple connections in

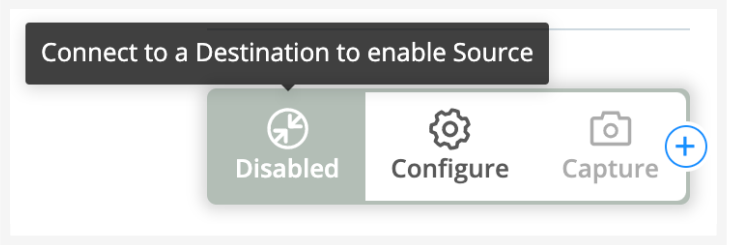
Source/Destination Tile Hover Options

Once you've configured a Source or Destination, hovering over its tile will display options like these:



Tile on-hover options

The leftmost button is an indicator of the Source's/Destination's state. Live means healthy; disabled, error, or warning states will also appear here. Note that a Source will appear as **Disabled** until you connect it to a Destination.



A Source not connected to a Destination

Selecting the middle **Configure** button reopens the Source's Destination's configuration drawer, where you can address any problems or simply update the configuration.

The right **Capture** button captures a sample of data flowing through the Source or Destination, in a drawered version of the Source's/Destinations's config modal > Live tab. (Not to be confused with a healthy tile's left **Live** indicator.)

Group Connections

Once you've configured two or more copies of the same Source or Destination type - for example, to support different configurations of one integration or protocol - QuickConnect stacks their tiles together to keep the display clean.

2 similar Destinations, stacked

The number at the stack's upper right shows how many tiles it contains.

Hovering over a stack doesn't change its options - until you select it. This will expand all the tiles in the group, so

1.4. Tutorials

Follow these tutorials to get a one-stop overview of the interface and data flow functionalities of Cribl Stream.

- Cribl Stream Tour covers QuickConnect and Data Routing, focusing on Cribl.Cloud.
- Get Started with Cribl Stream presents a more complex walkthrough of configuring a data flow, based on an on-prem deployment.
- Distributed Quick Start guides you through setting up a Distributed Deployment of Cribl Stream.

1.4.1. Cribl Stream Tour

This is a quick visual overview of configuring primary Cribl Stream components. It covers both the QuickConnect graphical UI and the more flexible Data Routing alternative, where you can conditionally branch the same incoming data to specific processing logic and destinations.

This tour is an ideal way to quickly familiarize yourself with Cribl Stream's UI and its general data-flow options. You can walk through the steps below in a free Cribl.Cloud environment, without installing any software.

While this tour is not a prerequisite to the adjacent Getting Started Guide and Distributed Quick Start tutorials, it will accelerate your later progress through them. The adjacent tutorials focus on on-prem deployments, processing Pipelines configuration, and more-complex routing options.

Accessing Cribl Stream

The quickest way to start your progress through this tour is by signing up for a free Cribl.Cloud account. Cribl will manage all infrastructure for you, and you can upgrade your deployment to a flexible paid plan whenever you want more capacity. Log in and you'll be ready to go.

If you prefer to deploy Cribl Stream on your own infrastructure, see Installing on Linux. Then register your deployed software. Registration will unlock a free license with similar capabilities to a Cribl.Cloud free plan. Customize your password (if desired) and you'll be ready to go.

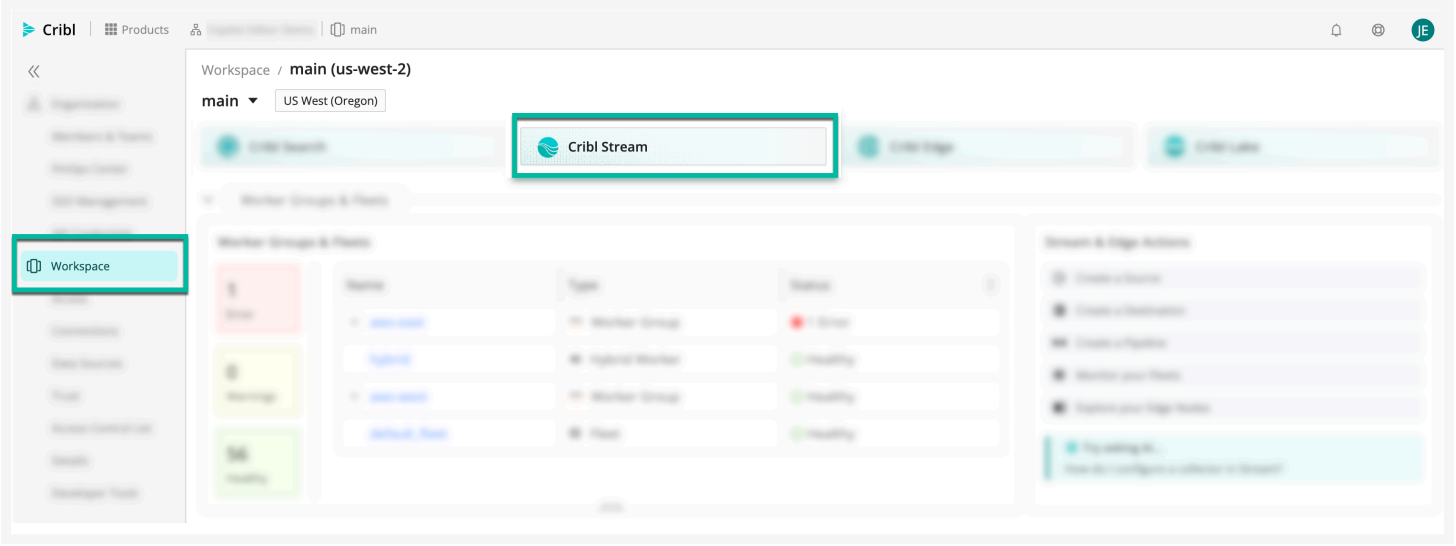
Setting Up the Tour on Cribl.Cloud

These first steps apply only to a Cribl.Cloud Organization. If you've deployed Cribl Stream as a single instance on your own Linux infrastructure, skip ahead to Simple or Complex Routing?.

In this tour, we'll use a Cribl.Cloud environment, which defaults to one Worker Group with one or two Workers. If you're running a single instance on your own Linux infrastructure, you can ignore the steps about deploying Groups.

Depending on how and when you take this tour - your environment, your plan (from Free to Enterprise), and the date - some Cribl Stream pages might look slightly different from some screen captures in this tour.

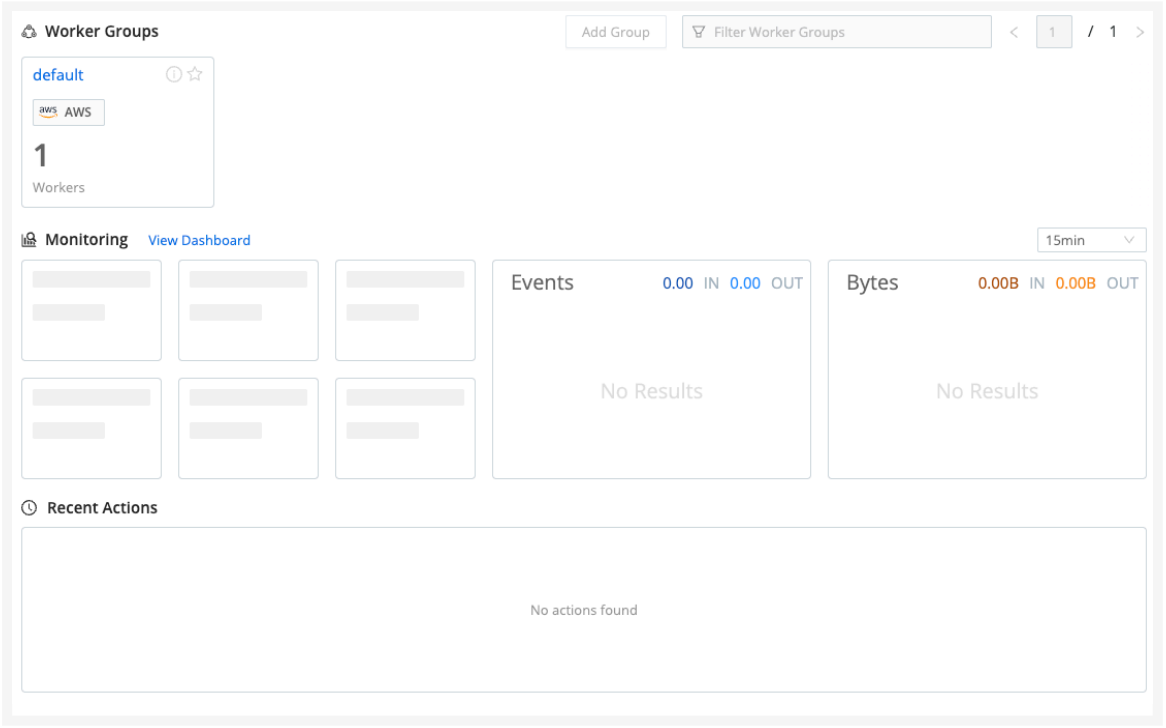
1. From your Workspace, select **Cribl Stream**.



Navigating to Cribl Stream

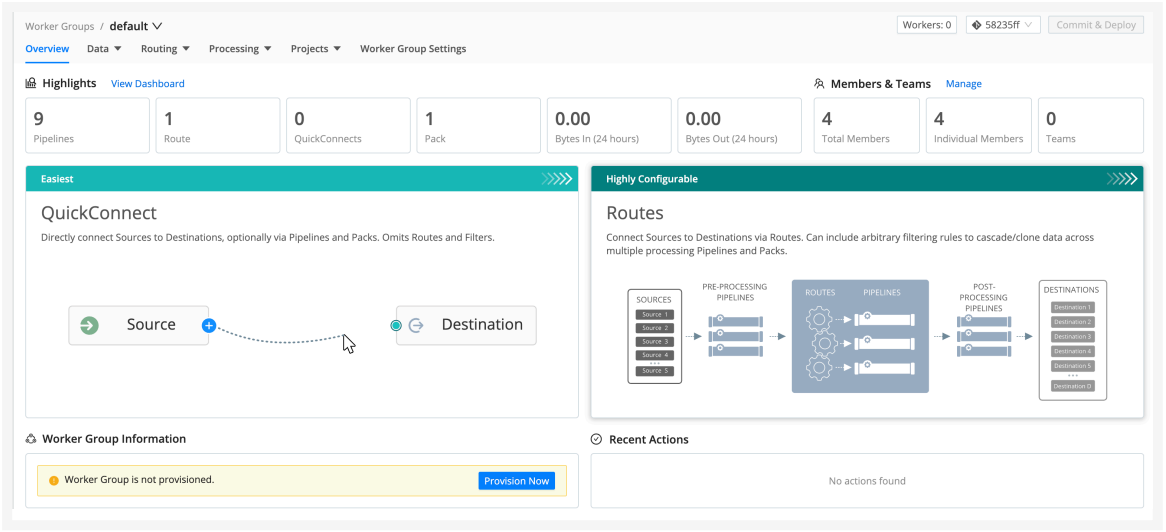
2. On the **Worker Groups** page, select the `default` Group.

A Worker Group is a collection of processing instances that share the same configuration. For details, see the Distributed Deployment topic.



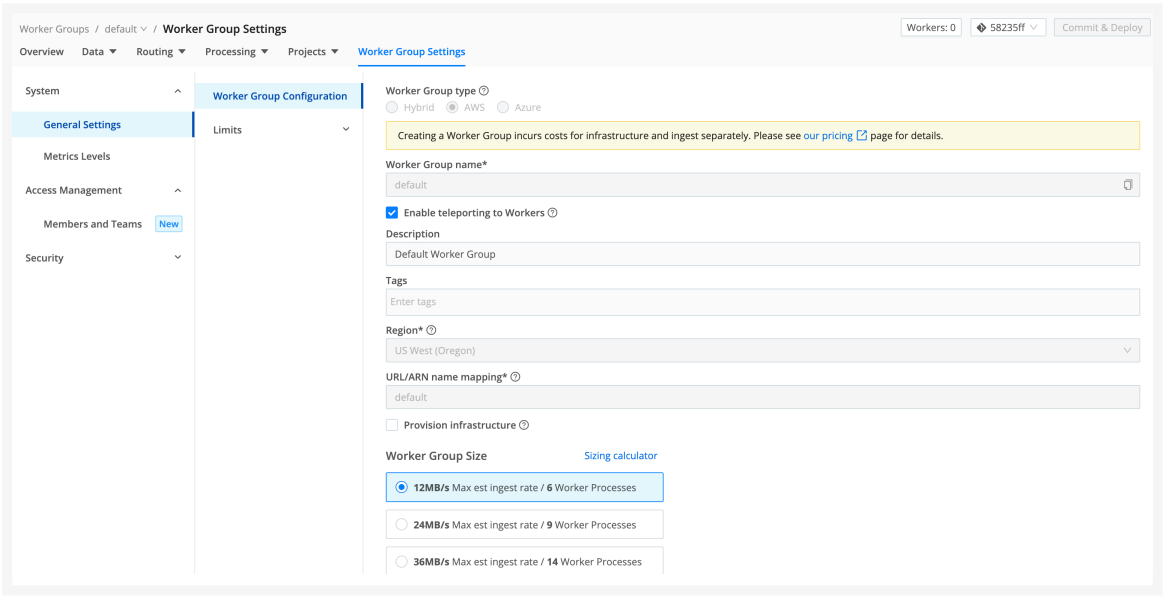
Selecting the `default` Worker Group

3. If you see a **Worker Group is not provisioned** warning, select the **Provision Now** button to enable the data flow we'll configure below.



`default` Group's initial state

4. As shown below, select both the **Enable teleporting to Workers** and the **Provision infrastructure** check boxes. Leave other defaults selected, and select **Save**.



Enabling the `default` Group

When you see a **Settings saved** confirmation banner, you're ready to commit and deploy your newly provisioned Worker Group.

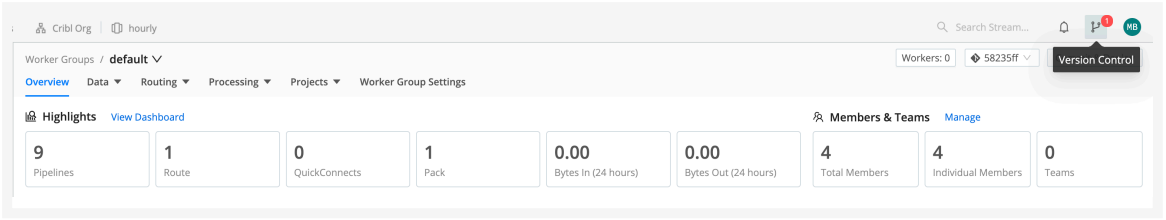
Committing and Deploying Config Changes

After any significant configuration change, you'll be prompted to use Cribl Stream's built-in version control (Git) features to deploy (or redeploy) your Workers and commit your configuration changes. Your first prompt is to deploy your newly provisioned Worker Group:

1. Select the blue **Deploy** button at the upper right (shown above with a red highlight indicator).

Leave the default commit message intact and click **Deploy**. This will make your newly provisioned Group ready to process data.

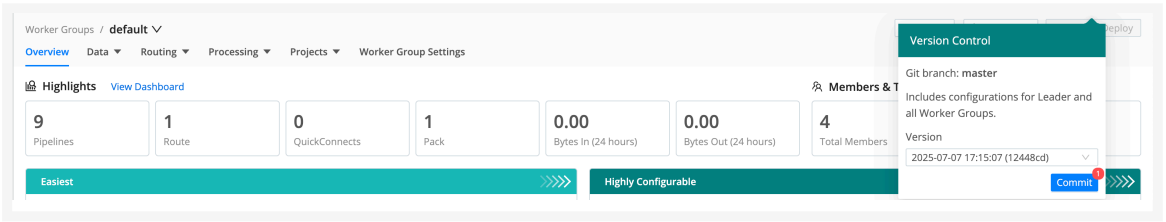
2. Next select the red-highlighted Version Control button at the upper right.



Version Control button

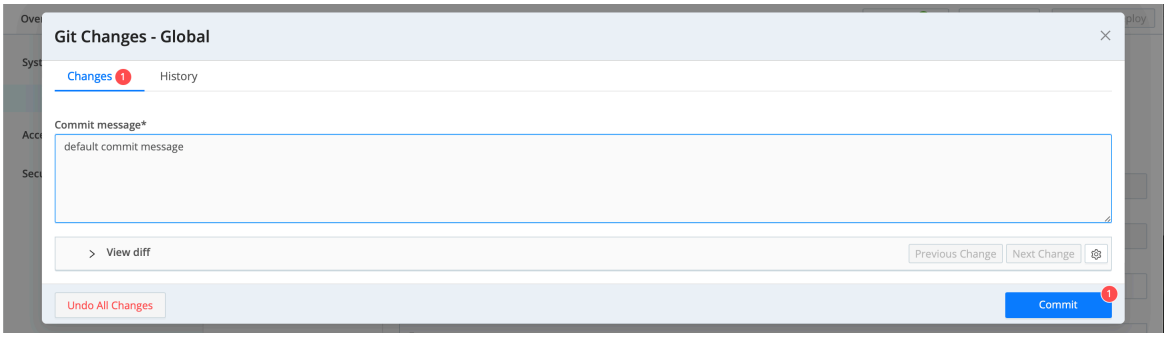
3. In the resulting drop-down, select the **Commit** button.

   (We'll explain all Version Control details later in this topic.)



Committing a configuration change

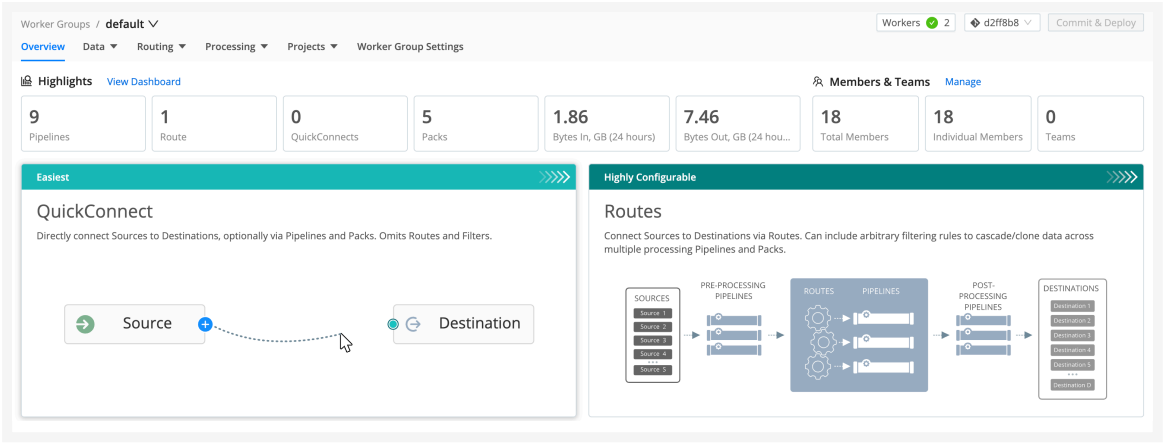
4. In the **Git Changes - Global** modal, accept all defaults, and again select **Commit** to close the modal.



Confirming a commit

5. If you don't already see a display like that shown below, select **Stream Home** to restore the **Worker Groups** page, then again select the `default` Group. You're now set up for the rest of this tour.

   To begin configuring this Group, you now have a two-up choice of **QuickConnect** or **Routes** tiles. How to choose? See the next section.



`default` Group enabled and ready to flow some data

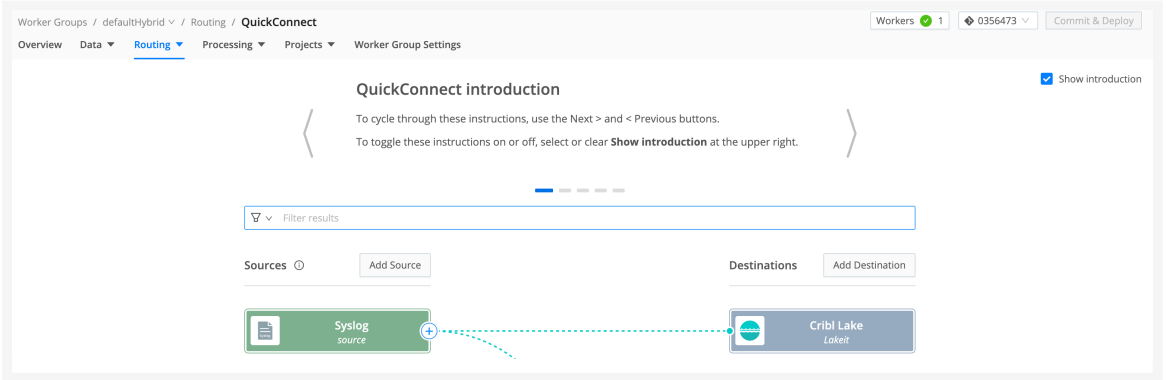
For More Information

- Cribl.Cloud Launch Guide

Simple or Complex Routing?

Your first choice is how simply or intricately you want to configure data flow from the first data Source you set up. Cribl Stream offers two user interfaces:

- QuickConnect (the left tile shown above) provides a simple visual UI. You drag and drop connections from Sources on the left to Destinations on the right. Each connection can include as much or as little intermediate processing as you like. To remove a connection, just drag it off the Destination (you'll get a confirmation modal). The only practical limitation here is that data flow through every Source -> Destination connection must be parallel and independent.
- Data Routing (the right tile shown above) provides fine-grained control, via a more traditional UI that you configure across separate modals and pages. You build a Routing table with conditional processing. This can cascade and clone data across multiple processing Routes, according to the arbitrary filtering rules and sequence that you define.



Configuring data flow in QuickConnect

So the trade-off between these two options is ease of use versus flexibility. These two options are separate, but not mutually exclusive: You can set up simultaneous data flows through QuickConnect and Data Routes.

You can also move Sources, and their connections to Destinations, between the two UIs. So one obvious option is to configure your first data flow in QuickConnect, and then move to the Routing UI as your familiarity and needs expand.

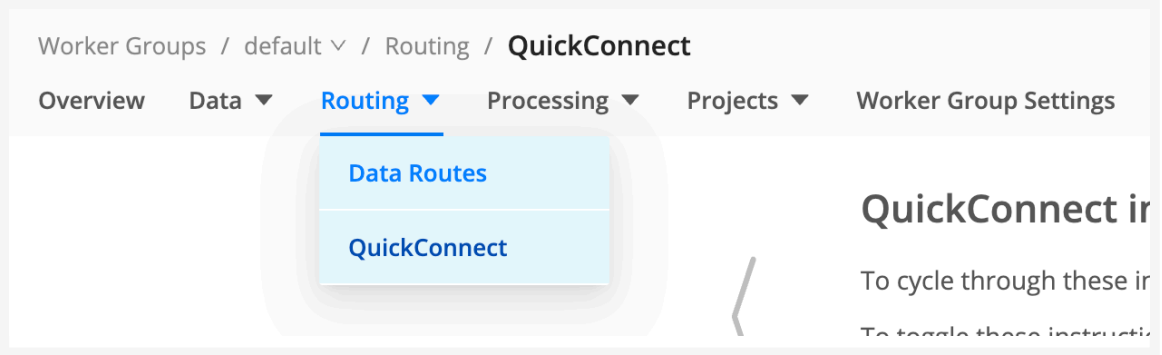
Each configured Source instance can be assigned only to QuickConnect or Routing, not both. To switch it to the opposite environment, you'll need to move it there, or duplicate its configuration there.

Navigate Between User Interfaces

To work in QuickConnect, select the left tile shown above, or select **Routing** > **QuickConnect**.

To work in the Routing UI, select the right tile shown above, or select **Routing** > **Data Routes**.

You can always switch between the two environments from a Worker Group, by using the **Routing** submenu.



Traversing between QuickConnect and Data Routes

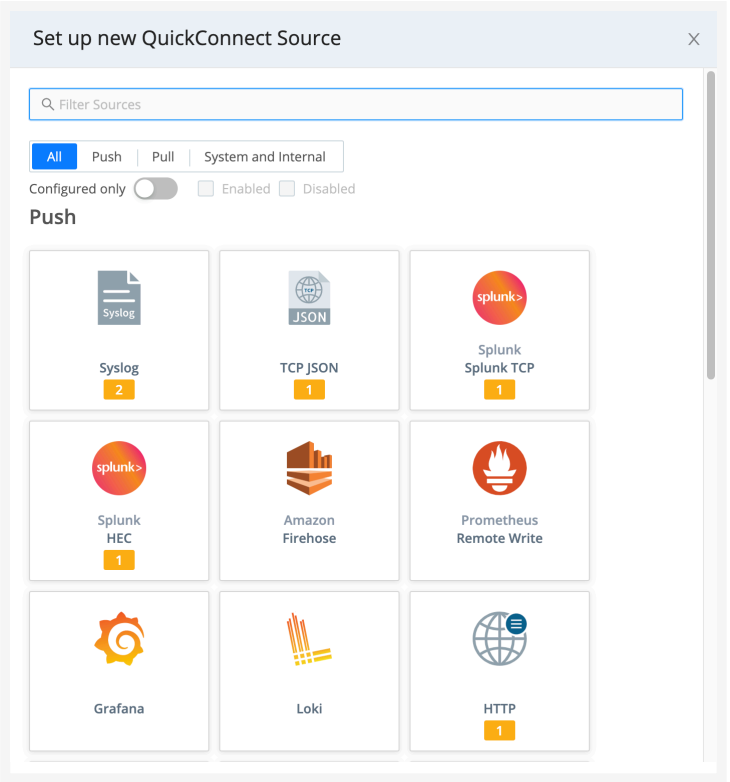
Configure a Source

A Source is a model of an upstream service sending data into the system. In this model, you configure authentication and/or other needed parameters.

You can configure multiple instances of a given Source type, each with different parameters. You can also attach processing that's specific to each instance. Cribl Stream provides the Datagen Source type to generate sample inbound events for development and testing.

Using QuickConnect

In QuickConnect, select the **Add Source** button on the left (Sources) side. In the left drawer, search for and select your desired Source type to begin configuring a new instance of that Source.



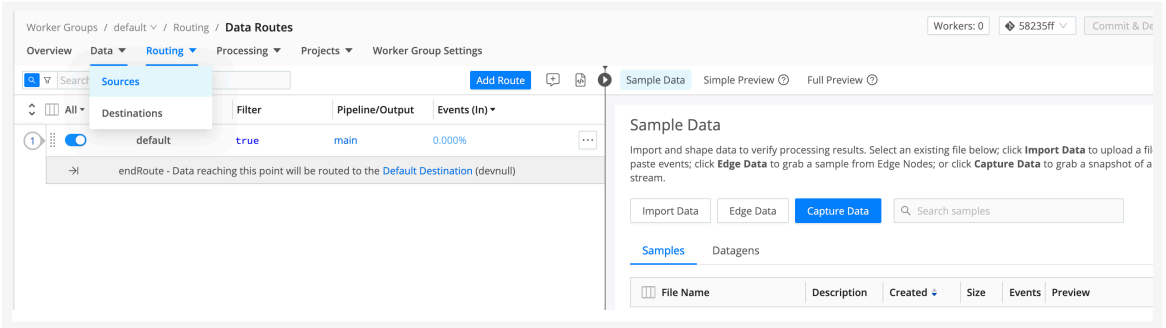
Selecting a Source through the QuickConnect UI

To configure (or reconfigure) a Source that's already present in QuickConnect, hover over its tile and select the **Configure** button.

For multiple Sources of the same type, tiles will be consolidated in a stack until you click the stack to display the hidden tiles.

Using Data Routes

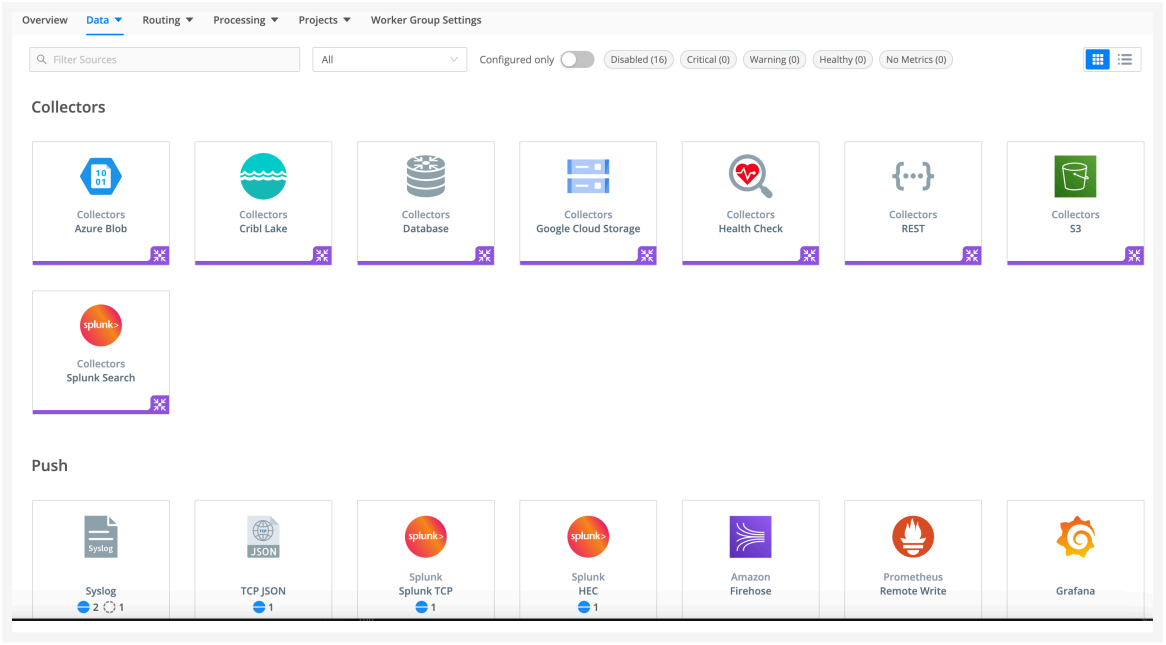
In the Routing UI, first select **Data** > **Sources**.



Selecting a Source through the Routing UI

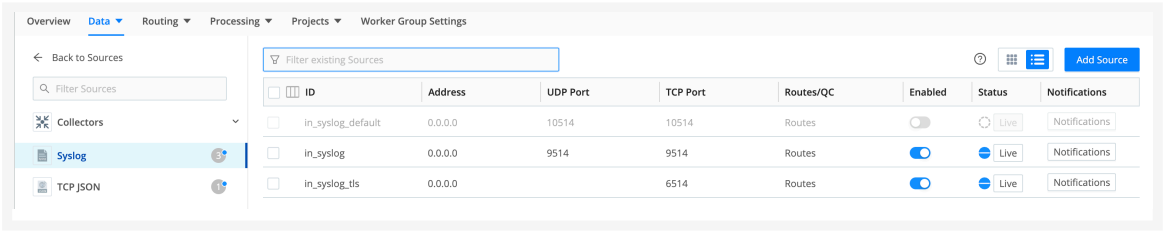
On the **Sources** page, search or filter for your desired Source type, then select its tile to begin configuring a new instance of that Source.

(After you click through to a Source, the same set of peer Sources will be available in the left nav.)



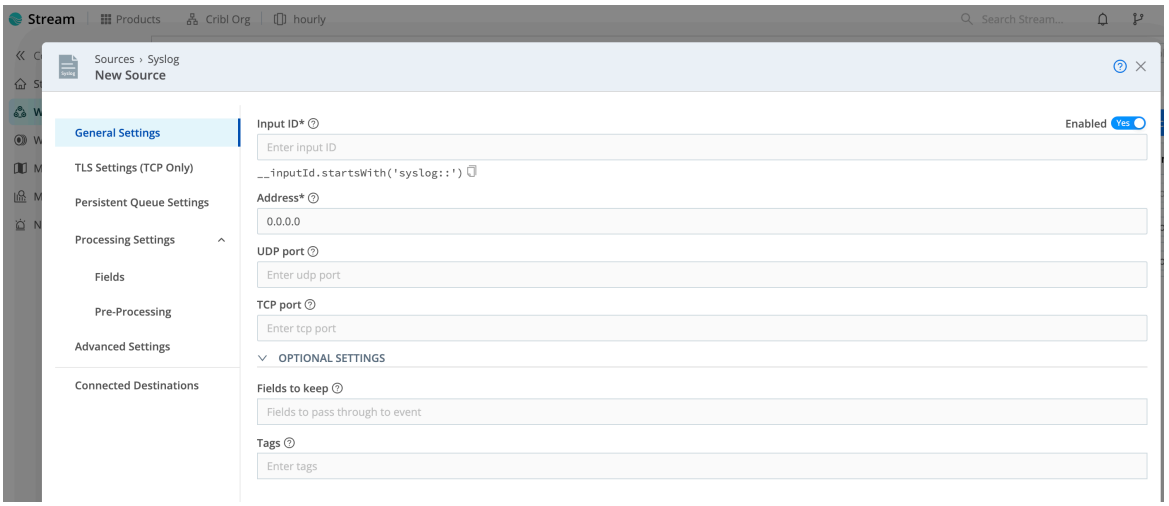
Sources page

Select an existing Source's row here to access its configuration, or select **Add Source** to configure a new instance of that Source.



Syslog Sources UI

You then configure (or reconfigure) the Source in a modal like this:



Syslog config modal

For More Information

- Sources
- Using Datagens
- QuickConnect
- Routes

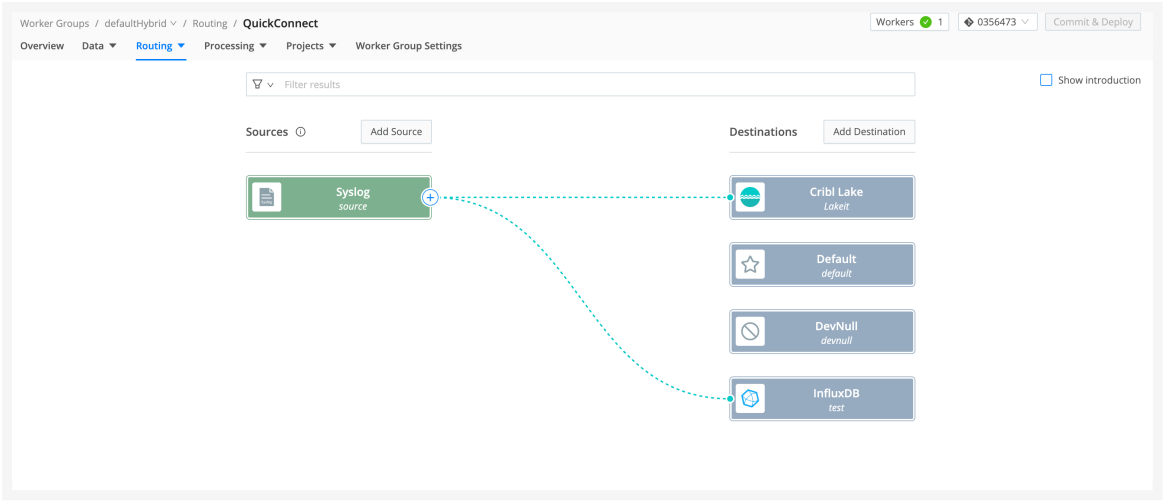
Configure a Destination

A Cribl Stream Destination is a model of a downstream receiver of events, enabling you to configure authentication and/or other needed parameters.

As with Sources, you can configure multiple instances of a given Destination type, each with different parameters. You can attach processing that's specific to each instance.

Using QuickConnect

To connect a configured Source to a Destination tile already present on the right, just drag from the Source and drop the connector onto the Destination. A pop-up will prompt you to configure the connection via a Passthru, Pipeline or Pack (explained below).

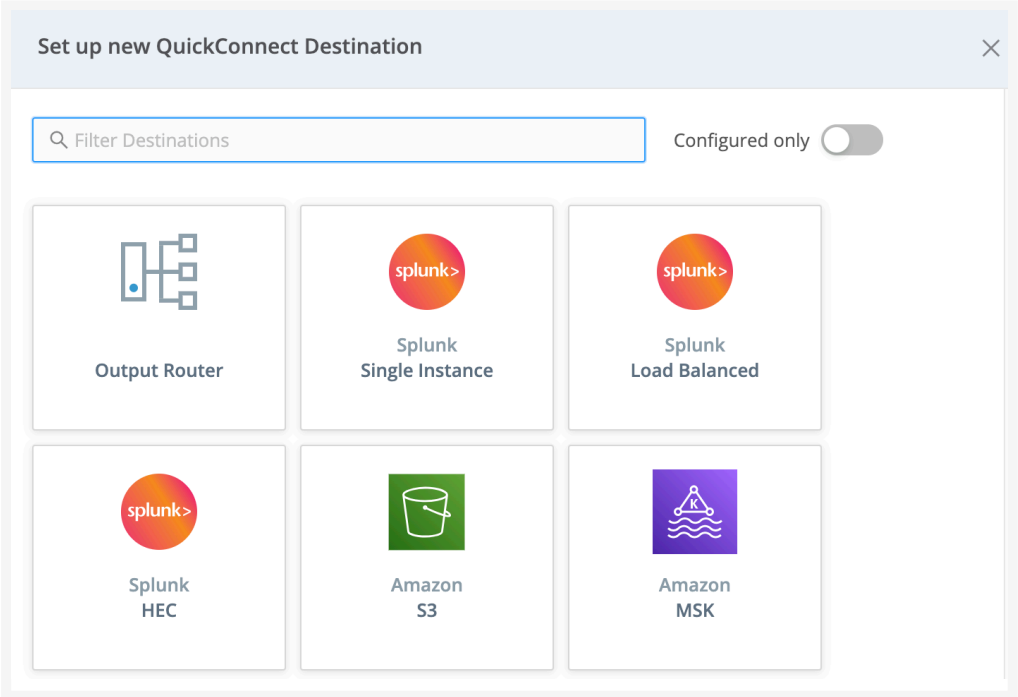


Multiple Destinations added to QuickConnect

To configure (or reconfigure) the Destination, hover over its tile and select the **Configure** button.

For multiple Destinations of the same type, tiles will be consolidated in a stack until you click the stack to display the hidden tiles.

To add a new Destination, select the **Add Destination** button on the right (Destinations) side. In the right drawer, search for and select your desired Destination type to begin configuring a new instance of that Destination.

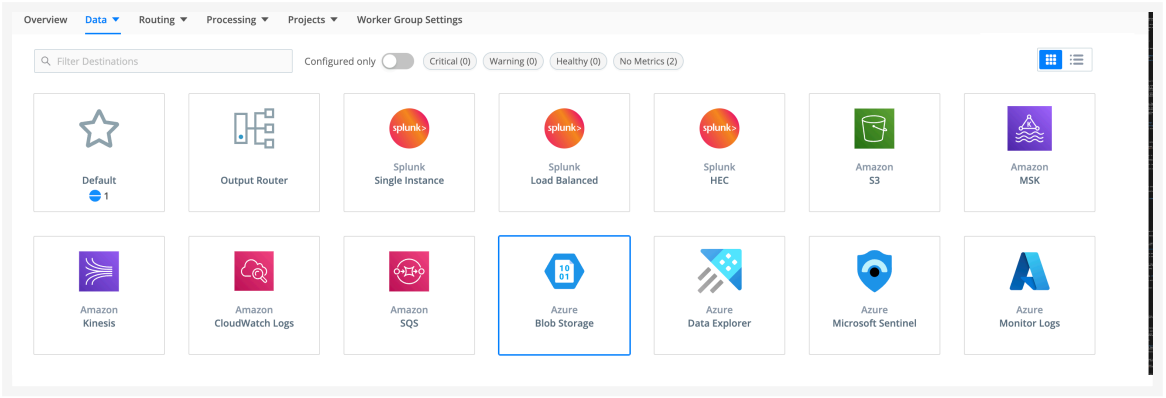


Selecting additional Destinations in QuickConnect

Using Data Routes

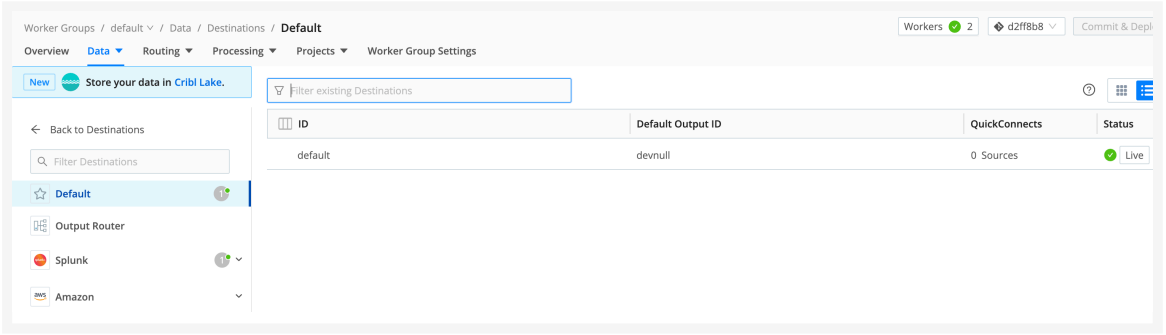
In the Routing UI, first select **Data** > **Destinations**. On the **Destinations** page, search or filter for your desired Destination type, then select its tile to begin configuring it.

(After you click through to a Destination, the same set of peer Destinations will be available in the left nav.)



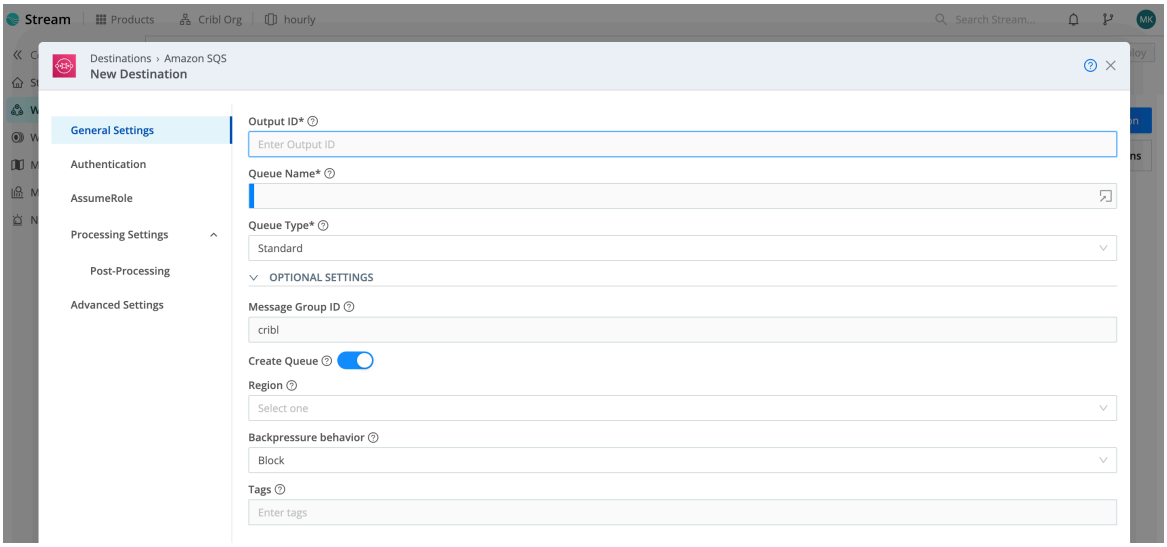
Destinations page

Select an existing Destination's row here to access its configuration, or select **Add Destination** to configure a new instance of that Destination.



Destinations UI

You then configure (or reconfigure) the Destination in a modal like this:



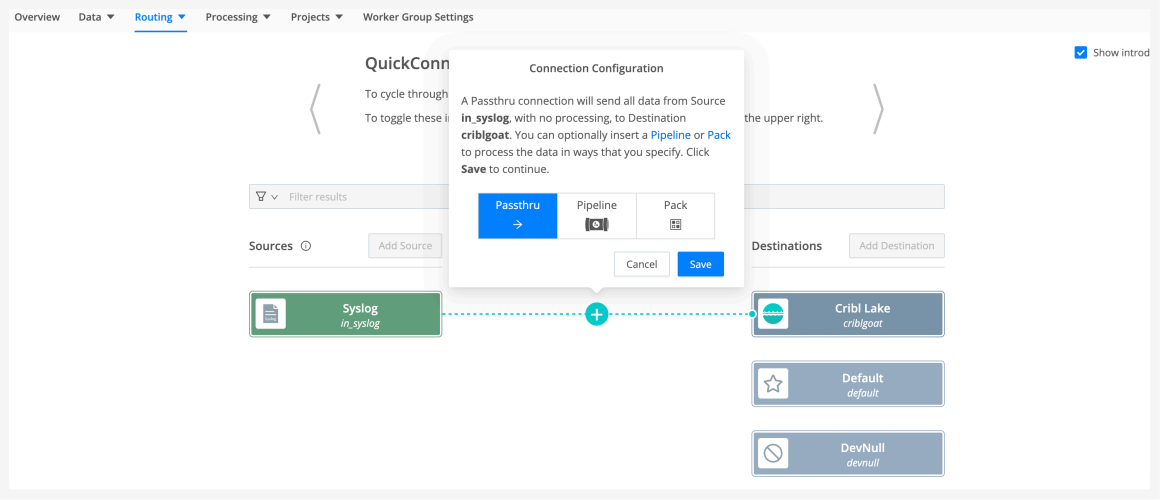
Destination config modal

For More Information

- Destinations

Connect: Passthru, Pipeline, or Pack

In QuickConnect, as soon as you drag and drop a Source -> Destination connection, a pop-up dialog will prompt you to select a connection type: **Passthru**, **Pipeline**, or **Pack**. (In Data Routes, you'll make the same choices elsewhere.)



Selecting a connection type in QuickConnect

What are these options?

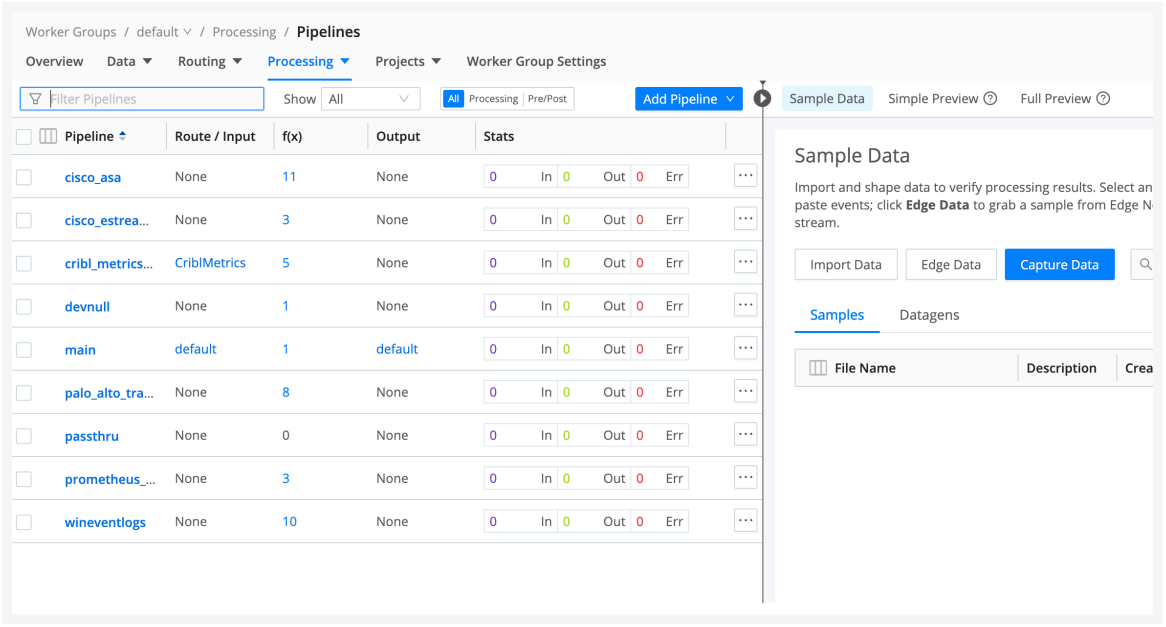
- A **Passthru** establishes a simple direct connection between a Source -> Destination pair.
- A Pipeline hosts a stack of Functions to process your data, in the order you arrange them.
- A Pack is a microcosm of a whole Cribl Stream configuration, designed to share configs between Worker Groups or deployments. A Pack can encapsulate its own Pipelines, Routes, Knowledge libraries, and sample data (but not data Sources or Destinations). A Pack can snap in wherever you can use a Pipeline.

So let's unpack Pipelines, where you'll define most of your event processing.

Pipeline Options

When you first create a new Pipeline, it's empty, making it equivalent to **Passthru**. Cribl Stream provides a set of out-of-the-box Pipelines for specific purposes. As you can see below, one of these is a **passthru** Pipeline. This similarly contains no Functions - making it the Data Routes UI's equivalent of QuickConnect's **Passthru** button.

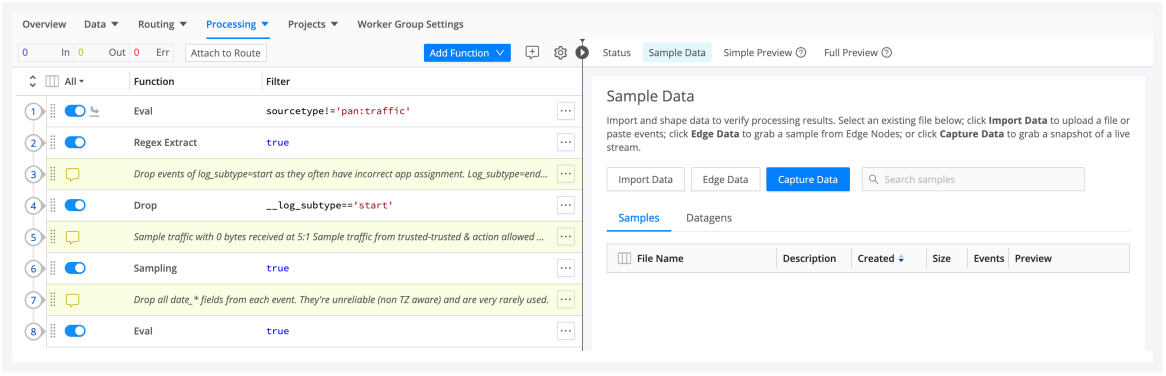
For these reasons, it's easy to start with an empty Pipeline, and then add Functions as you need them.



Pipelines Shipped with Cribl Stream

A Pipeline, as you populate it, is a stack of processing Functions and optional comments. Use the accordions on the left to expand or collapse each Function. Use the check boxes on the left to enable/disable each Function - preserving structure as you develop and debug the processing flow.

Use the right **Sample Data** pane to preview how the Pipeline's current state transforms a set of events. (Controls here enable you to capture live data, save it to sample or Datagen files, and reload those files.)



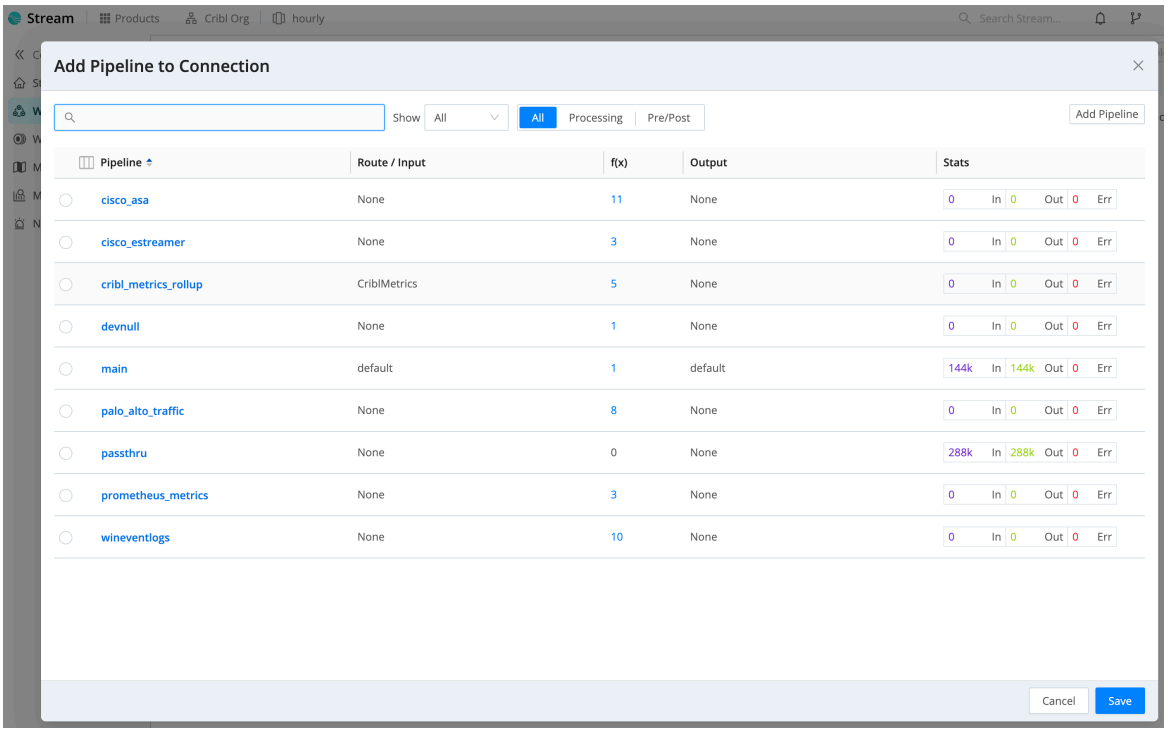
A Pipeline's Functions stack

Although you'll configure most data shaping in processing Pipelines between Sources and Destinations, you can also attach a pre-processing Pipeline to any Source instance, and a post-processing Pipeline to any Destination instance. Use these options when you need to "condition" the specific data type handled by a particular inbound or outbound integration.

Connect via QuickConnect

To select a connection type in QuickConnect, select the appropriate button in the pop-up dialog shown above. To accept the default **Passthru** option, just select **Save**. Your Source and Destination will now be connected.

If you select **Pipeline**, you'll see an **Add Pipeline to Connection** modal. Here, you can select an existing Pipeline, or select **Add Pipeline** to define a new one. Then select **Save** here to establish the connection.



Add Pipeline to Connection modal

Selecting **Pack** opens an **Add Pack to Connection** modal that works exactly the same way.

Whichever option you choose here, your connection between this Source and Destination is now complete.

To get data flowing, jump ahead to commit and deploy your new configuration.

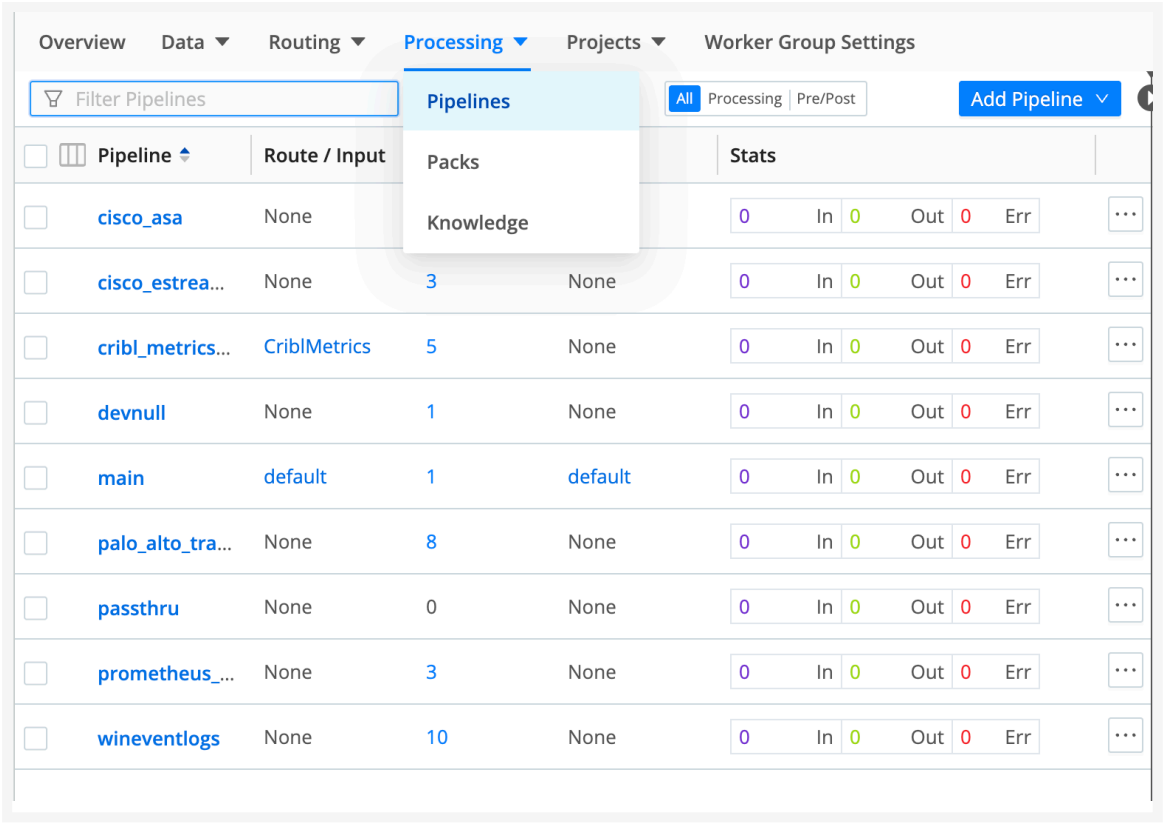
Connect via Data Routes

The Data Routes UI enables you to finely configure multiple, interdependent connections between Sources and Destinations. Events clone and cascade across a Routing table, based on the filtering rules and sequence you define.

In exchange for this flexibility, you sacrifice QuickConnect's one-stop, snap-together UI metaphor. After configuring your Sources and Destinations - which you've already done above - you need to connect them by separately setting up at least one Pipeline and one Route.

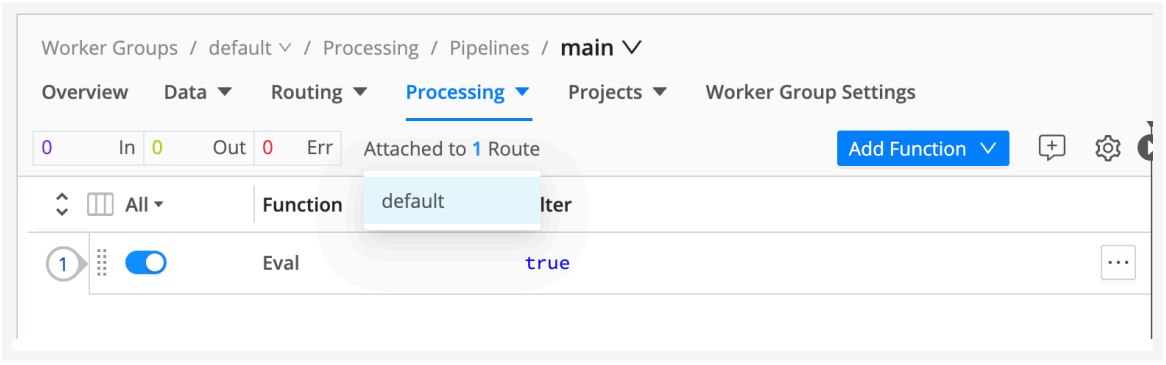
Pipelines Access

From a Worker Group, select **Processing** > **Pipelines**. In addition to the minimal **passthru** Pipeline identified above, other simple building blocks here include **main** (which initially contains only a simple Eval Function) and **devnull** (which contains a single Function that just drops events).



Routing to Pipelines

Out-of-the-box Pipelines are already connected to Routes. When you click into a Pipeline, you can hover to display the connected Route, as shown here. But as our final connection step, let's next look at how to connect up everything (Source, Pipeline, Destination) in a Route.



Checking Pipeline's Route connection

For More Information

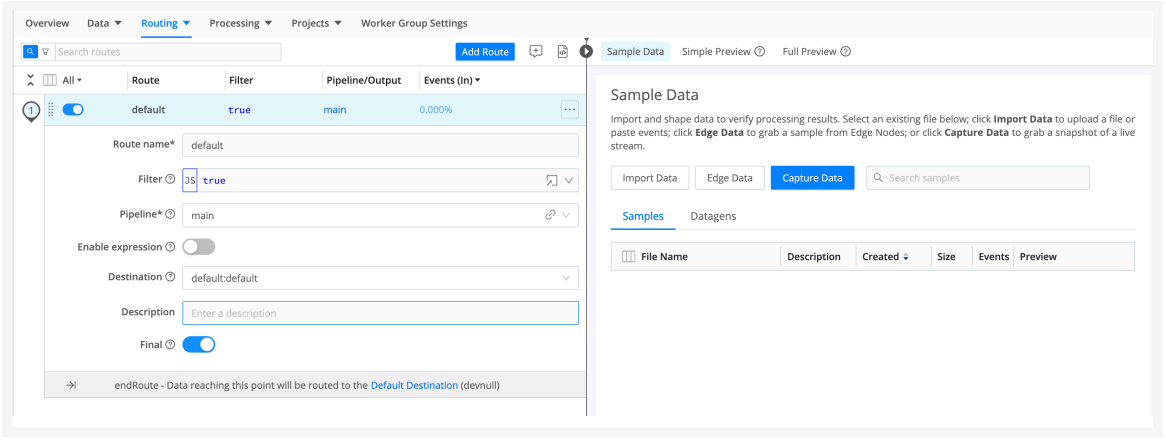
- Refine Data with Functions walks you through configuring Cribl's built-in Pipeline Functions to perform common processing tasks like parsing, masking, adding, deleting, and renaming event fields.

Configure a Route

In the Data Routes UI, a Route is where you complete a data-flow connection. Each Route connects exactly one Pipeline with exactly one Destination.

This combination can consume data from one, multiple, or all the Sources you've configured on the Data Routes side. (QuickConnect Sources are separate, but you can open those Sources' configs to move them to the Data Routes UI.) This is governed by the filtering we'll see below.

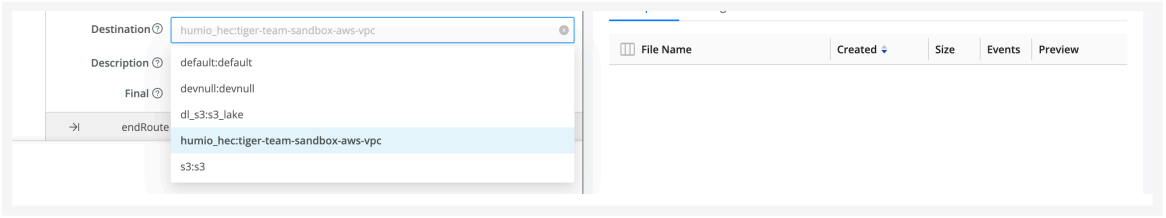
To open the Routing table, from a Worker Group, select **Routing** > **Data Routes**.



Accessing the Routing table

As you can see, Cribl Stream ships with one starter Route named `default`. It's configured as follows:

- The **Filter** expression is set to **true**. This simple, default expression makes data from **all** active Sources available to this Route. (See finer filtering options below.)
- The **Pipeline** drop-down is where you set the 1:1 Pipeline:Route connection. This starter Route's Pipeline defaults to **main**, but you can select any one other configured Pipeline.
- The **Destination** drop-down defines the single Destination for this Pipeline:Route connection. If this isn't already set to your chosen Destination, you'll most likely want to select that here. (Note that each option displayed on this list corresponds to the unique, arbitrary **Output ID** set on one **instance** of a Destination type. So some names might initially be unfamiliar.)



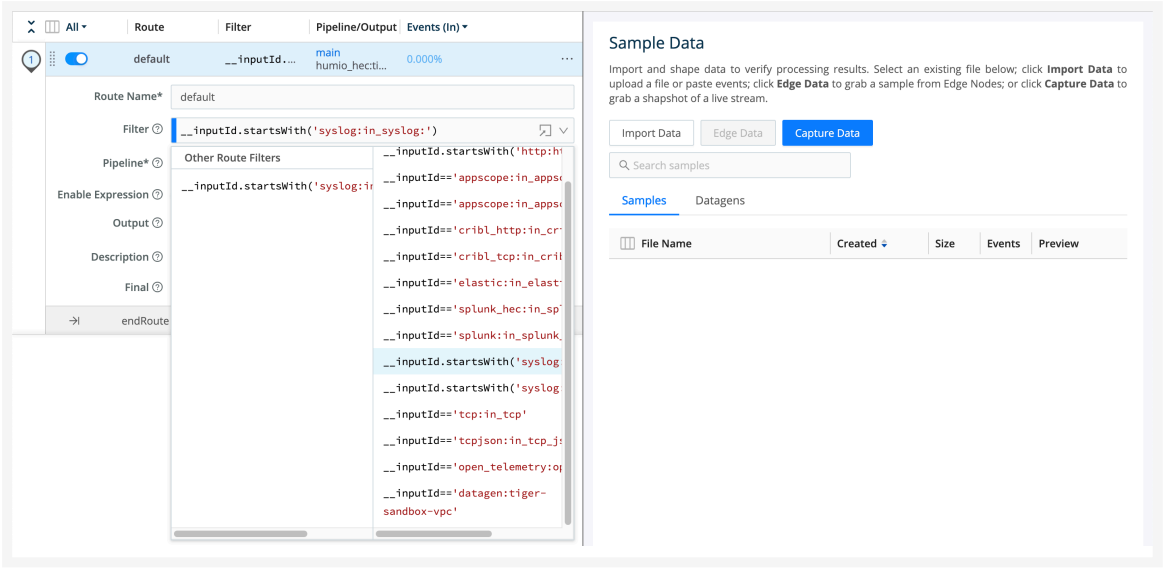
Changing a Route's Destination

To get data flowing, jump ahead to commit and deploy your new configuration.

Filter a Route's Input Data

When you want a Route to ingest data from only one Source, or a subset of Sources, you expand its **Filter** expression beyond the default **true** catch-all.

Filter is itself a drop-down. Open it to select among typeahead suggestions for several common inbound data types.



Tuning a Route's Filter expression

You can select one of these expressions and then modify it, using the same `__inputId` syntax. In the example expression below, note that the identifier after the `:` is - as with the **Destination** options we saw above - the arbitrary name of one Syslog Source instance:

```
__inputId.startsWith('syslog:in_syslog:')
```

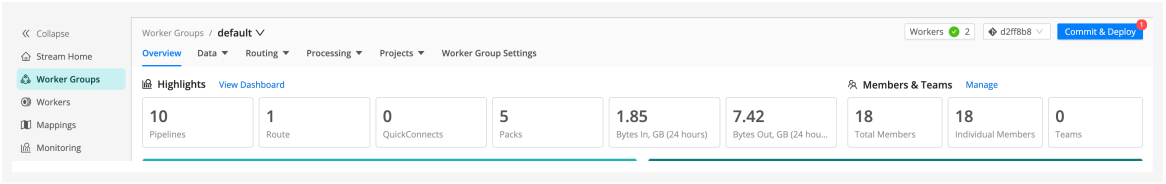
For More Information

- Build Custom Logic to Route and Process Your Data: Filter Expressions

Commit/Deploy and Monitor

To get data flowing as you expect, you'll always need to use Cribl Stream's built-in version-control (Git) integration to commit your configuration changes and deploy them to your event-processing Worker instances. If you started this tour on Cribl.Cloud, you first used this during initial setup in Committing and Deploying Config Changes above.

After any significant configuration change, you'll be prompted with Group-level **Commit/Deploy** buttons like this at the upper right:



Commit & Deploy prompt

If you're using an on-prem version of Cribl Stream, different version-control settings might display **Commit** and **Deploy** as separate buttons:.

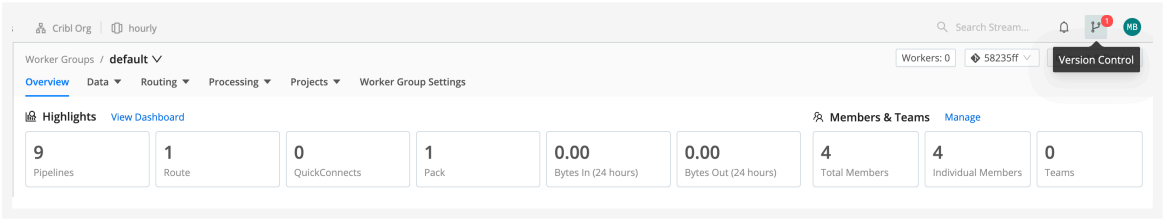
Selecting either flavor of the **Commit** button will display the confirmation modal you saw above. Just be sure to select the appropriate buttons to both commit **and** deploy your changes.

Confirming Data Flow

Data should now start flowing through the Workers. (You can check this by selecting the **Monitoring** link. This takes you to read-only dashboards. If necessary, select **Stream Home** and your Group when you want to return to configuration options.)

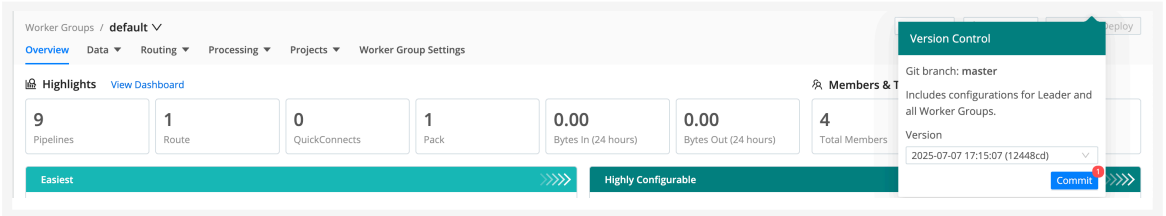
Product-Wide (Leader) Commit

As you saw during initial setup: In return for clearing all the red-dot prompts, your reward will be one more red dot, on the top nav's product-wide Version Control button. This is because Cribl Stream sees the preceding deploy to Workers as a change to record (commit) on the **Leader's** configuration.



Leader commit prompt

This top-level commit isn't necessary to enable data flow, but it backs up and safeguards your overall configuration. Select the button to display one final confirmation prompt:



Confirming a commit on the Leader

Select **Commit** here, confirm the commit and your configuration is complete!

1.4.2. Get Started With Cribl Stream

Welcome to Cribl Stream! This tutorial is a hands-on guide to the core features and common use cases of Cribl Stream. You will set up a basic deployment using mock data from a mobile phone company, giving you a real-world example of how Cribl Stream helps you work with and transform data efficiently.

In this tutorial, you will learn how to:

- Add a Source, Destination, Pipeline, and Route to get data flowing in and out of Cribl Stream.
- Capture and filter sample data.
- Use Functions to:
    - Mask sensitive data.
    - Parse raw data, promoting it to key-value fields.
    - Add, remove, or rename data fields.
    - Drop unnecessary data events.

This tutorial takes approximately 1 hour to complete. However, you don't need to finish it in one sitting. Your progress will persist in Cribl Stream between sessions.

Before You Start: Set Up a Cribl.Cloud Organization

The easiest way to complete this tutorial is on a Cribl.Cloud Organization, which is fully managed on your behalf by Cribl and includes Cribl Stream. If you already have one set up, you can use that Organization. Otherwise, to set up a free Cribl.Cloud Organization and register it in the Cribl.Cloud portal. See Register a Cribl.Cloud Portal for additional information.

If you prefer to download Cribl Stream software and deploy it on infrastructure that you manage, see Installing on Linux.

Step 1: Add a Source to Get Data into Cribl Stream

In this step, you will begin working with the basic building blocks of Cribl Stream to get data flowing in to your deployment. You will add a Source to your Cribl Stream deployment and verify data is coming into Cribl Stream by viewing the Monitoring page. See Cribl Stream Basic Concepts for more information about Sources and other useful concepts.

A Source receives data from an upstream sender. Cribl Stream can receive data from a variety of senders, such as syslog (UDP/TCP), Splunk, and Amazon S3. For this tutorial, you will enable a Cribl Stream built-in datagen (data generator) that generates a stream of realistic sample log data from a mock mobile phone company.

To add a Source:

1. On the top bar, select **Products**. Under **Stream**, select **Worker Groups**. Then, select a Worker Group. If you're working with a new deployment of Cribl Stream, you will only see one Worker Group named `default`. Select that Worker Group or another Worker Group.

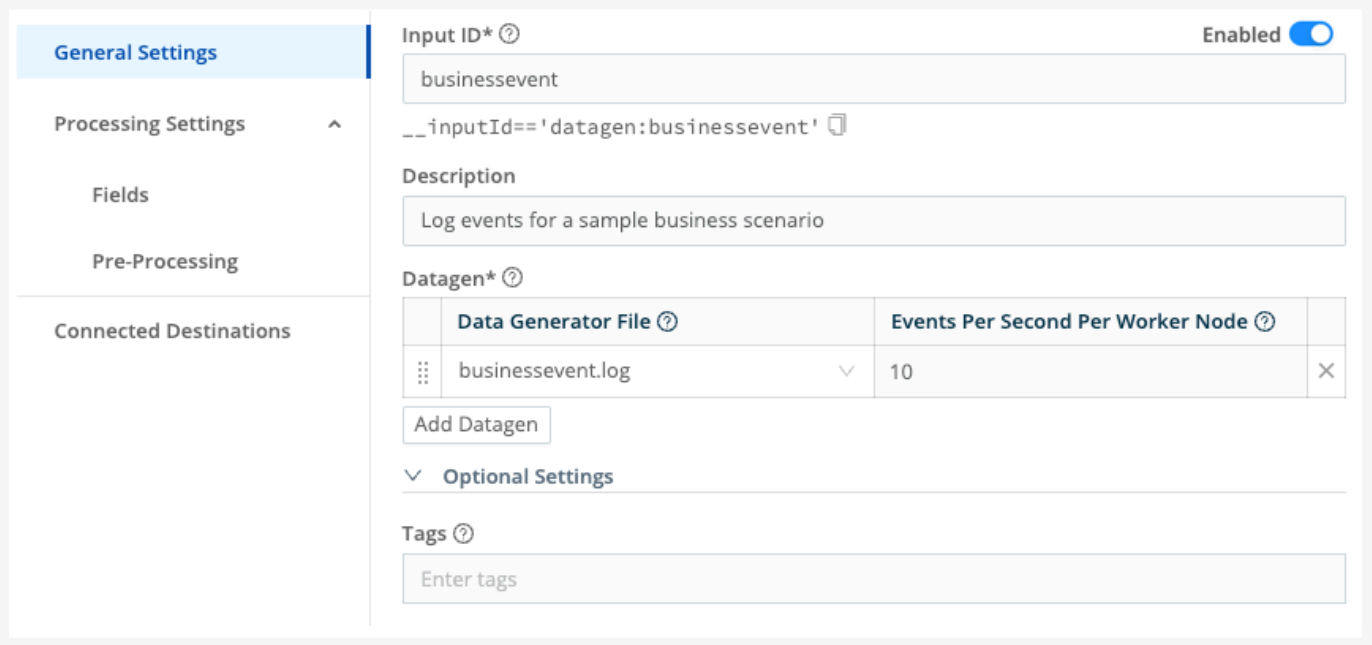
   A Worker Node is a Cribl Stream instance that is managed by a Leader Node. Worker Nodes are responsible for processing data and handling networking for a cluster. A Worker Group is a collection of Worker Nodes that share the same configuration. See Distributed Deployment for more information.

   If you see a **Worker Group is not provisioned** warning, select the **Provision Now** button to create Workers before proceeding.

2. On the **Worker Groups** submenu, select **Data** > **Sources**.
3. From the **Data Sources** page's tiles or left menu, select **Datagen**. Select **Add Source** to configure a new datagen source.

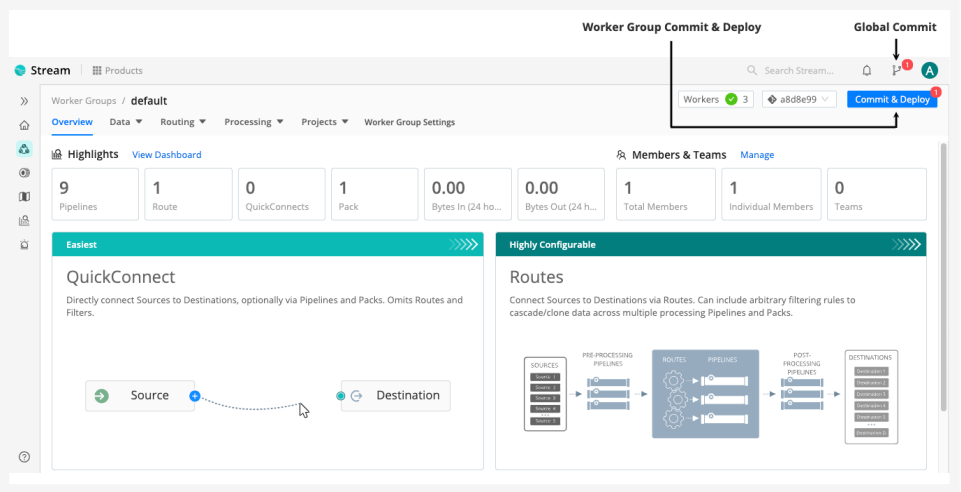
   You can use the search box to jump to the **Datagen** tile.

4. In the **Input ID** field, name this Source `businessevent`.
5. In the **Description** field, enter `Log events for a sample business scenario`.
6. In the **Data Generator File** menu, select `businessevent.log`.



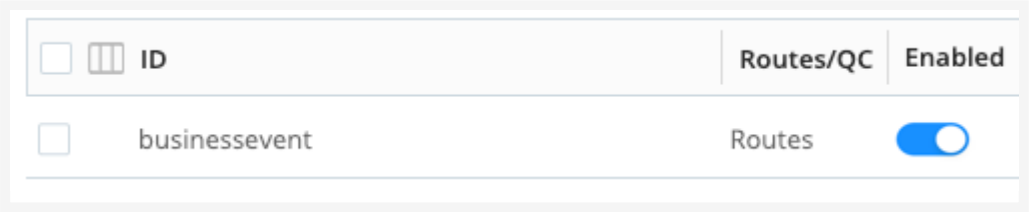
Configure a datagen Source

7. Save your changes. In the Global Commit menu, select **Commit**. Write a commit message, such as `Add datagen Source`. Then, select **Commit & Deploy** to deploy the changes to your Worker Group.



Commit and Deploy controls in the user interface

8. Verify that the datagen Source is ready to generate sample data by confirming that the **Enabled** column next to the `businessevent` toggle is toggled on.



Verify datagen Source

Now that sample data is flowing into your Cribl Stream deployment, you will learn how to add a Destination to receive the data in the next step.

Step 2: Add a Destination to Send Data to a Receiver

In this step, you will add a Destination to your Cribl Stream deployment and verify data is coming in and out of Cribl Stream by viewing the Monitoring page.

A Destination is a downstream receiver that receives data from Cribl Stream after data processing and transformation. Cribl Stream can send data to many different receivers, such as Cribl Lake, Cribl TCP, Cribl HTTP, Kafka, Kinesis, TCP JSON, Splunk, and others. For this tutorial, you will use Cribl Stream's built-in DevNull Destination, which simply discards data events. For the purposes of this tutorial, you will use the DevNull Destination because it simulates a real output, providing a quick method for testing initial Cribl Stream setups.

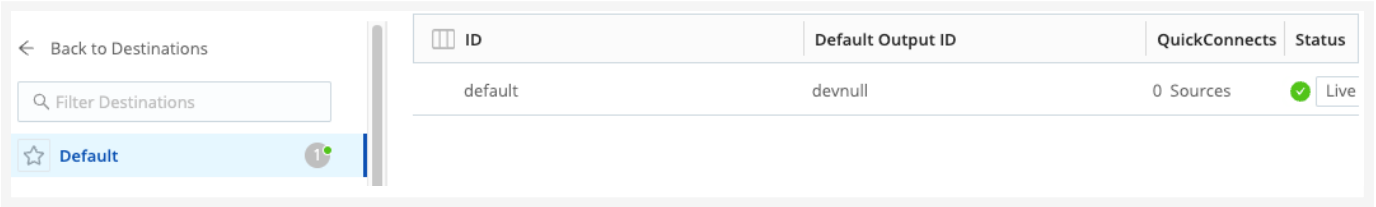
You will also use the Monitoring page to verify that data is flowing through Cribl Stream. The Monitoring page provides real-time data dashboards you can use to monitor performance and resource utilization. See Monitoring for more information about additional uses.

To add a Destination:

1. On the **Worker Groups** submenu, select **Data** > **Destinations**.
2. From the list of Destinations, select **DevNull**.

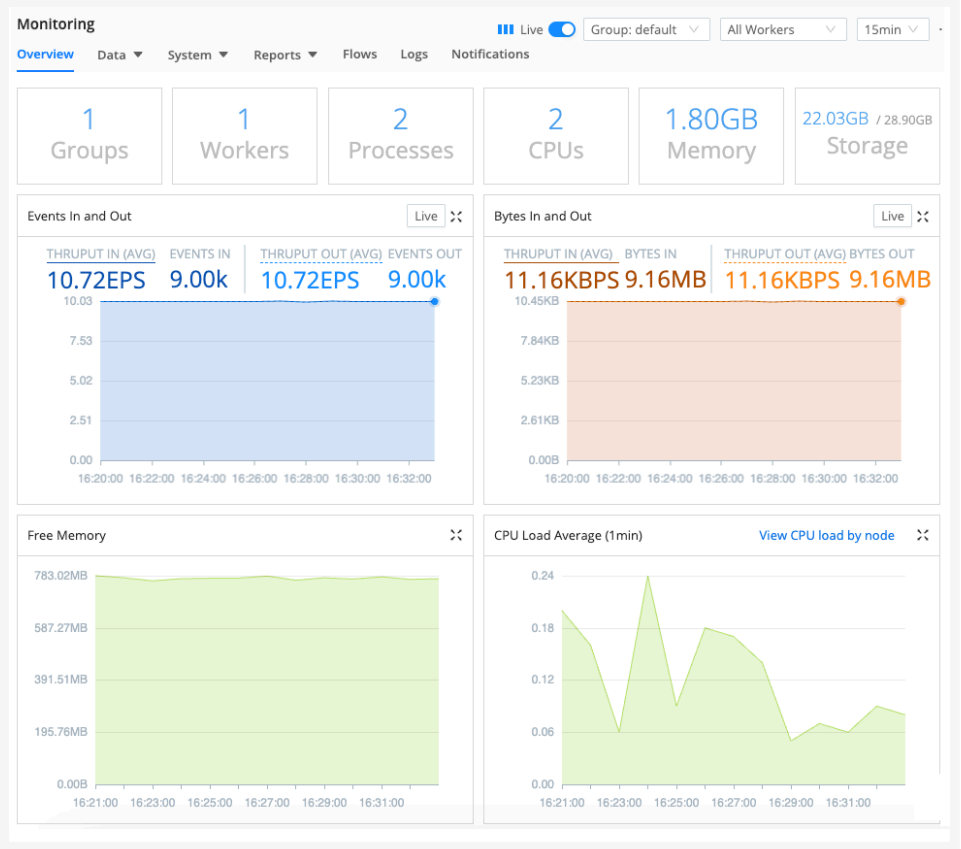
You can use the search box to jump to the **DevNull** tile.

3. On the **devnull** row, look for the **Live** indicator under **Status**. This confirms that the **DevNull** Destination is ready to accept events.
4. From the list of Destinations, select the **Default** Destination at the top. Verify that **devnull** appears in the **Default Output ID** column.



Verify DevNull Destination

5. In the Global Commit menu, select **Commit**. Write a commit message, such as `Add DevNull Destination`. Then, select **Commit & Deploy** to deploy the changes to your Worker Group.
6. Now that you've set up data flow on both sides, check the data dashboards on the Monitoring page. In the sidebar, select **Monitoring** to open a summary dashboard of the data that is flowing in and out.



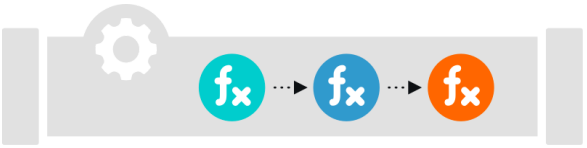
Monitoring dashboard

7. Explore some of the other dashboards by selecting some the other submenus on the Monitoring page.

Now that you have data flowing in and out of Cribl Stream, the rest of the tutorial will show you how to manage that data with Cribl Stream.

Step 3: Add a Pipeline

In this step, you will add a Pipeline to your data flow. A Pipeline contains a logical sequence of Functions, which process data events and perform various operations. Data flows from a Source into a Pipeline, where each Function processes the events in the order they appear. Once processing is complete, the data flows out of the Pipeline to a Destination.



Pipelines contain a logical sequence of Functions

Functions are pieces of JavaScript code that Cribl Stream invokes on each event that passes through a Pipeline. Each Function performs some sort of processing action on data events. This tutorial will explore Functions in more detail later. For now, the focus in this step is on creating a simple Pipeline to get you started.

To add a Pipeline:

1. On the top bar, select **Products**. Under **Stream**, select **Worker Groups**. Select a Worker Group. If you're working with a new deployment of Cribl Stream, you will only see one Worker Group named `default`. Select that Worker Group or the same Worker Group you used earlier in the tutorial.
2. On the **Worker Groups** submenu, select **Processing** > **Pipelines**.

The Pipelines page has a two-pane view, with a **Pipelines** list on the left pane and **Sample Data** controls on the right.

3. In the **Pipelines** pane, select **Add Pipeline**, then select **Add Pipeline** again.
4. In the **ID** field, enter a unique identifier. For this tutorial, use `transform-data` or your own identifier.
5. Save the changes. In the Global Commit menu, select **Commit**. Write a commit message, such as `Add transform-data Pipeline`. Then, select **Commit & Deploy** to deploy the changes to your Worker Group.

Now that you've set up your Pipeline, you will learn how to capture and filter sample data in the next step. You can use sample data to test and validate your Pipeline as you build it.

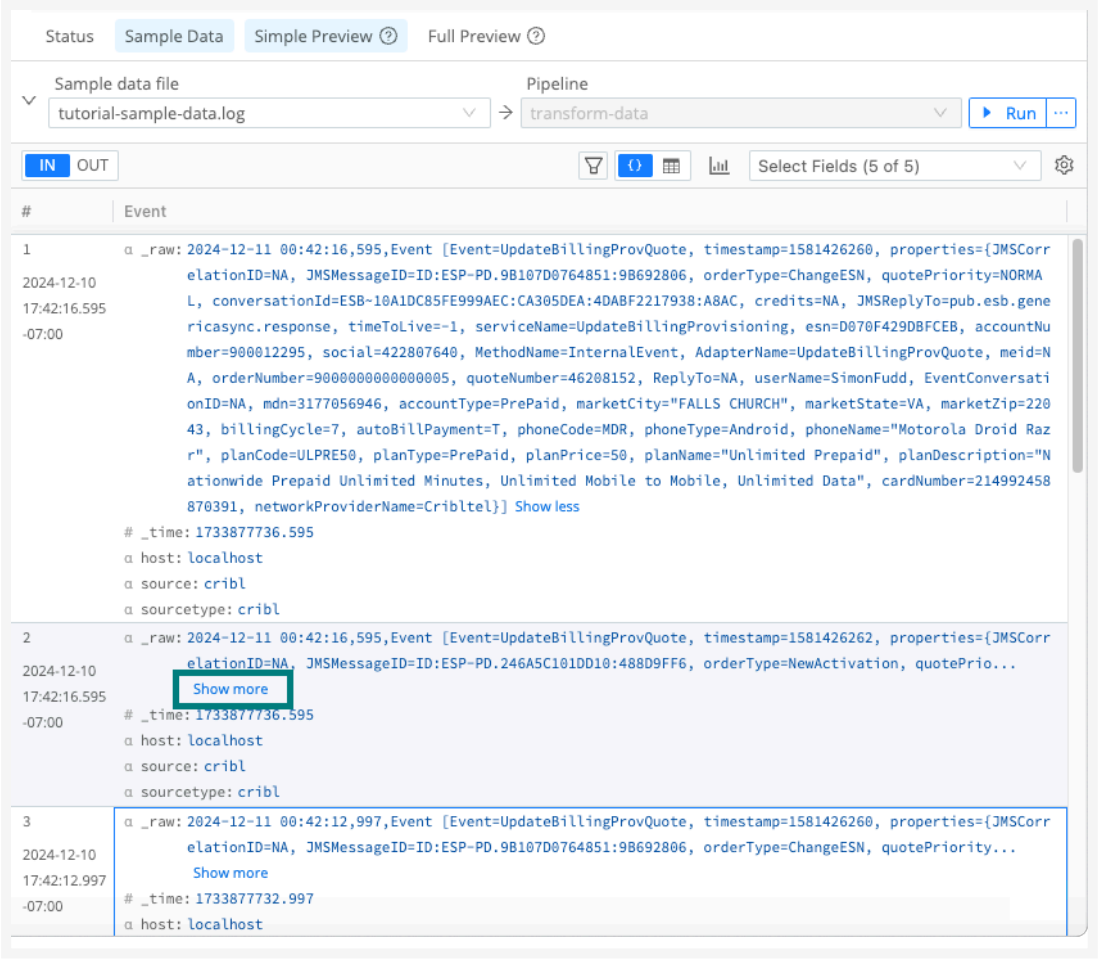
Step 4: Capture and Filter Sample Data to Test the Pipeline

In this step, you will learn how to capture sample data from the Pipeline you added in the previous step. The Data Preview tool is useful for previewing and testing data transformations as you build them. It gives you confidence that your Pipeline configurations results in the intended output. The Data Preview allows you to get data from multiple places, such as inbound streams, uploaded files, and from your computer's clipboard. See Data Preview for more information.

You will also learn how to further filter this sample data if needed. For this part of the tutorial, you will filter a key-value pair field called `marketState`. This field contains a value that indicates the U.S. state associated with the data event. The values are state abbreviations, such as `FL` for Florida, `CA` for California, and so forth. In this tutorial, you will filter the captured data to only show events that have a `marketState` value of `TX` (for Texas). Note that the filter you apply in this step only applies to this specific data capture and has no permanent impact on the data in the Pipeline.

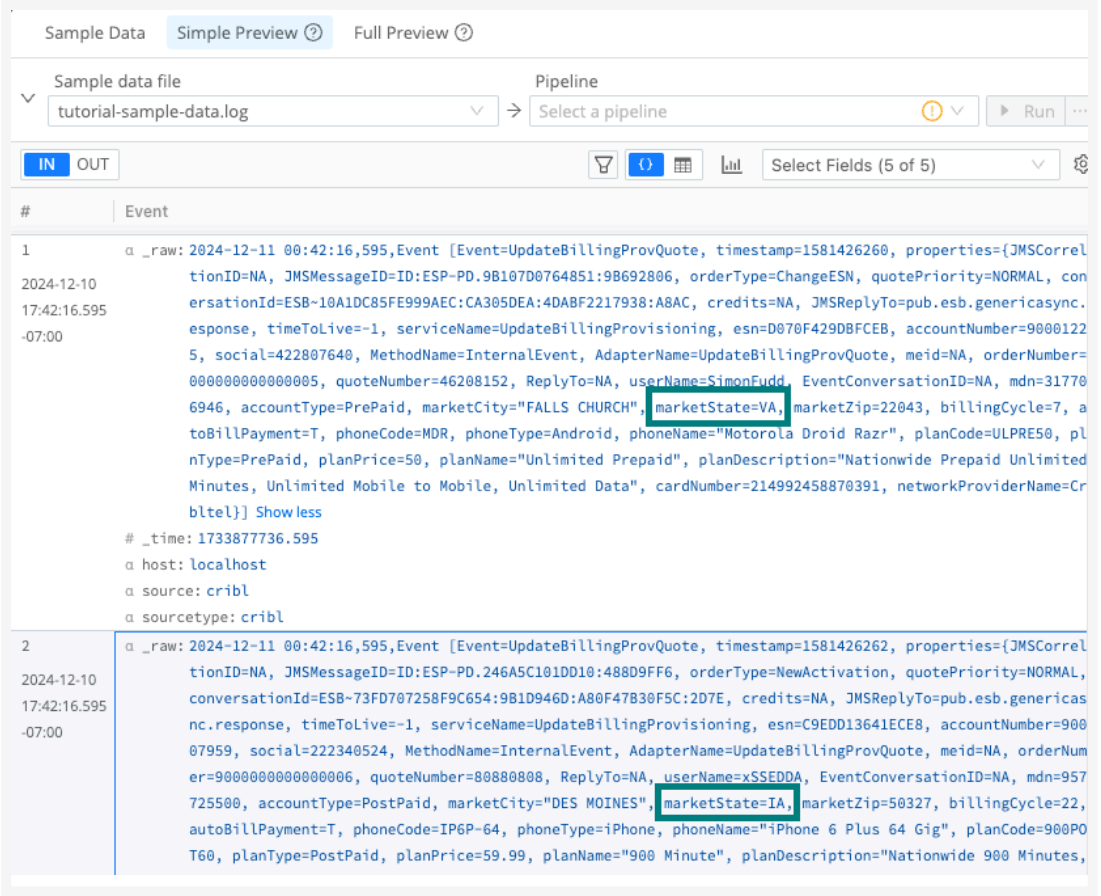
To capture and filter data:

1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Select the **Sample Data** tab. Then, select **Capture Data**. Select **Capture**. Accept the default settings and select **Start**.
2. When the modal finishes populating with events, select **Save as Sample File**. In the sample file settings, change the generated file name to a name you'll recognize, such as `tutorial-sample-data.log`.
3. Save your changes and verify that you now see a preview of the captured events. Select the **Show more** link to expand one or more events.



Select “Show more” to expand sample data

4. In the Data Preview, find at least one entry with a `marketState` key-value pair and take note the typical values, which will be the abbreviation for a U.S. state. You may need to select the **Show more** link to expand events to see this key-value pair.



Example marketState key-value pairs

5. Select the **Sample Data** tab and select **Capture Data** to capture a new data set.
6. In the **Filter Expression** field, replace the default text with this regular expression (regex): `_raw.match(/marketState=TX/)`.

As an experiment, try typing in the expression rather than copying it. As you type, notice how Cribl Stream provides auto-complete suggestions to help make a well-formed JavaScript expression. It will also alert you if your expression is invalid.

7. Select **Capture**, then **Start**.

You might notice that with this filter applied, it takes longer for Cribl Stream to capture 10 matching events. Applying filters can impact the time it takes to capture matching events.

8. Verify that the sample data only includes data events matching the `marketState=TX` filter.

Filtered marketState value

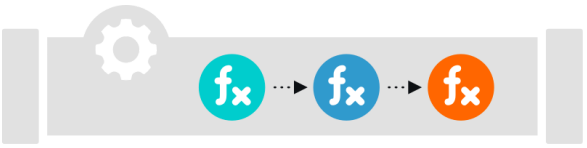
9. You can save this sample data if desired. Otherwise, select **Cancel** to discard this filtered data.

As you build Pipelines, generate, or upload sample data and check the Data Preview when working with Pipelines to validate your changes.

In the next steps, you will learn how to process the original sample data by building out the Pipeline Functions. For the rest of the tutorial, you will use the original, unfiltered capture to verify the impacts of your changes.

Step 5: Add a Mask Function to Redact Sensitive Data

For the next several steps in the tutorial, you will get hands-on experience using Functions to process data events. Functions are pieces of JavaScript code that Cribl Stream invokes on each event that passes through a Pipeline. Each Function performs some sort of processing action on data events.



Pipelines contain a logical sequence of Functions

You can use Functions to perform a variety of different processing operations on your data, such as:

- Reduce the data, trimming it to remove unnecessary fields.
- Enrich the data to add additional fields.
- Normalize data, perhaps transforming it from one data structure to another.
- Redact sensitive data.
- Replay data or generate sample data.
- Transform and transmit data.

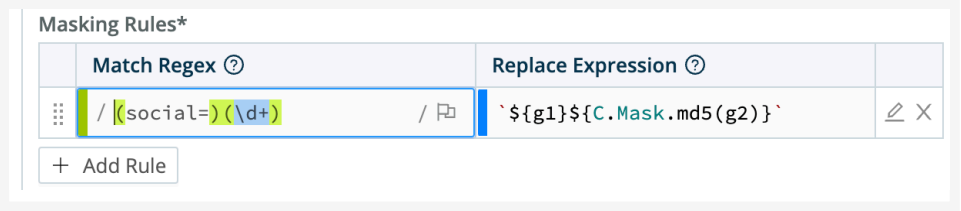
The order that Cribl Stream applies the Function to a data event depends on the order in which the Function appears in the Pipeline. You can optionally add filters to Functions to limit their processing scope to matching events only.

In this step, you will use the Mask Function to redact sensitive material from the data events. The data generated by the datagen Source includes a field for fictitious Social Security numbers, which is personally identifiable information (PII) data. The Mask Function can convert sensitive numbers into an encoded MD5 hash or another value. See Mask Function for more information.

To add a Mask Function to your Pipeline:

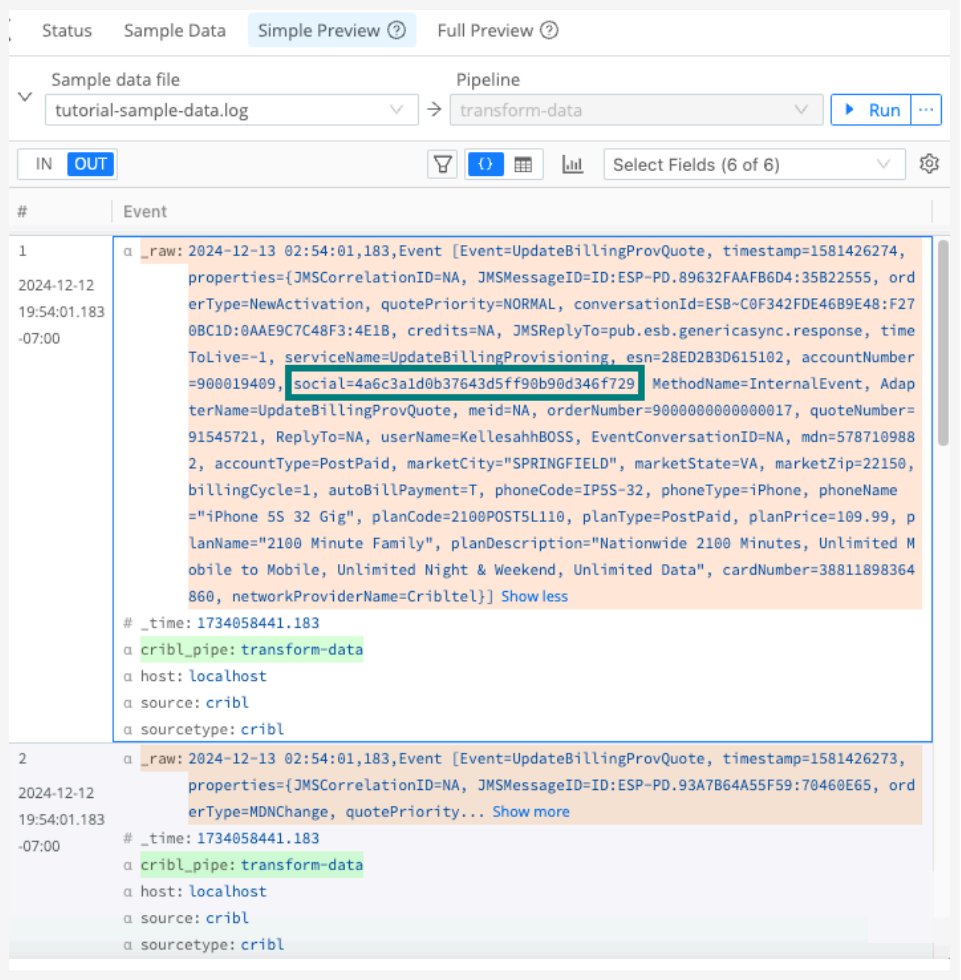
1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Then, select **Add Function**. Search for `Mask` and select it.
2. Under the **Masking Rules** field, in the **Match Regex** field, add this regular expression (regex): `(social=)(\d+)`.

   This regular expression looks for the string `social=` followed by any digits.
3. In the **Regular Expression** field, paste the following hash function, including the backticks: `` `${g1}${C.Mask.md5(g2)}` ``.
4. Leave the **Apply to Fields** set to the default setting: `_raw`.



Enter a Masking Rule to hash Social Security numbers

5. Save your changes. Then, confirm that the Data Preview pane for the `tutorial-sample-data.log` has switched from the **IN** tab to the **OUT** tab. This tab shows you the outbound effect of the Pipeline you just saved. The `_raw` field for each event has changed color, indicating that it has undergone some redactions.
6. Select the **Show more** link to expand at least one event. Verify that the `social` values displays a hash.



Mask Function and hashed result

Having successfully redacted sensitive data, you can continue exploring Functions more in the next step.

Step 6: Add a Parser Function to Extract Events

The datagen Source generates a lot of raw, unstructured data events. These data events are more useful when converted into key-value pairs and promoted to the field level.

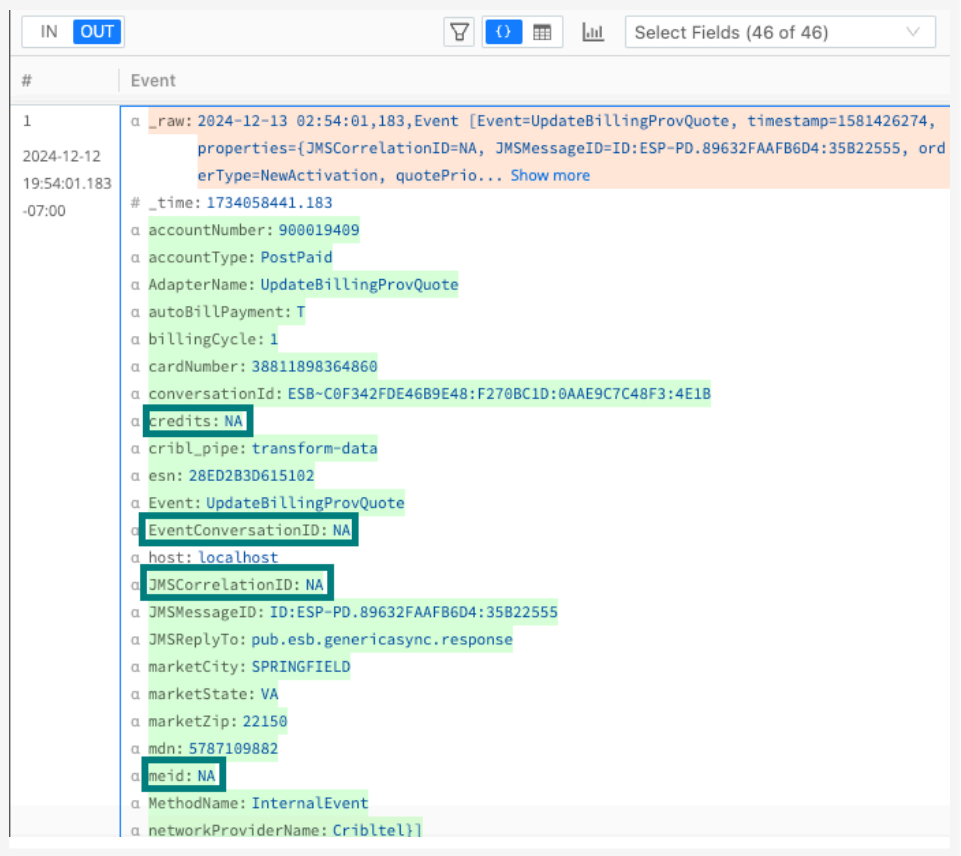
Raw, unstructured data lacks context and organization, making it harder to extract value without significant preprocessing. Rich, structured data is more useful than raw data because it is better organized, enriched, and formatted to make it easier to analyze, process, and derive insights.

In this step, you will use the Parser Function to convert raw data into key-value pairs and promote them to the field level for better data processing and analysis. You can also use the Parser Function to filter fields with certain values. You will use this Function to ignore fields containing the value `NA` (not applicable). See Parser Function for more information.

To add a Parser Function to your Pipeline:

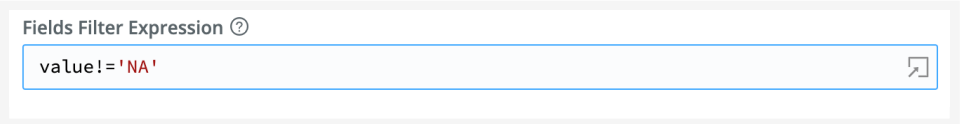
1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Then, select **Add Function**. Search for `Parser` and select it.

2. Leave the **Operation Mode** set to its `Extract` default.
3. Set the **Type** to `Key=Value Pairs`.
4. Leave the **Source Field** set to its `_raw` default.
5. Select **Save**.
6. Confirm that the Data Preview pane now displays many more fields parsed from `_raw`. While in this view, notice also that many fields have `NA` (not applicable) values.



N/A Fields in Parsed Data

7. You can enhance the **Parser** Function to ignore fields with these values. In the Parser Function **Fields Filter Expression** field, enter this negation expression: `value!='NA'`. Select **Save**.



Filter the Parser Function to ignore fields with 'NA' values

8. Confirm that the Data Preview pane now no longer shows any fields with the value `NA`, such as `credits`, `EventConversationID`, and `ReplyTo`. The remaining fields should display meaningful values.

If you didn't see the fields change, toggle the Parser Function off in the Pipeline. Then, select **Save** and watch the Data Preview pane change again. By toggling Functions on and off, you can test

and troubleshoot each Function's effect. Remember to toggle it on again before moving on to the next step in the tutorial.

Now that you've added the Parser Function to your Pipeline, your log data is better-organized and less bloated. Continue making additional data improvements in the next step.

Step 7: Add a Eval Function to Enrich Data and Remove Fields

In this step, you will use the Eval Function to enrich and streamline your data. The Eval Function adds or removes fields from events. See Eval Function for more information.

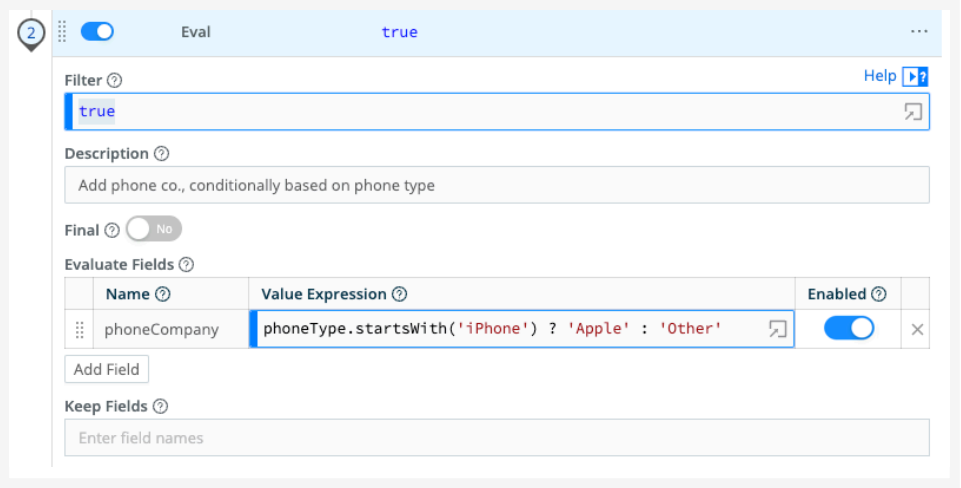
First, you will use this Function add a new field called `phoneCompany`, which will assign a value of `Apple` or `Other` based on the contents of the `phoneType` field.

Next, you will use the Eval Function to remove the `_raw` field, which is no longer needed since you promoted its contents to key-value fields in the previous step. Removing this large, redundant field results in cleaner events and reduces the load on downstream resources.

To add an Eval Function to your Pipeline:

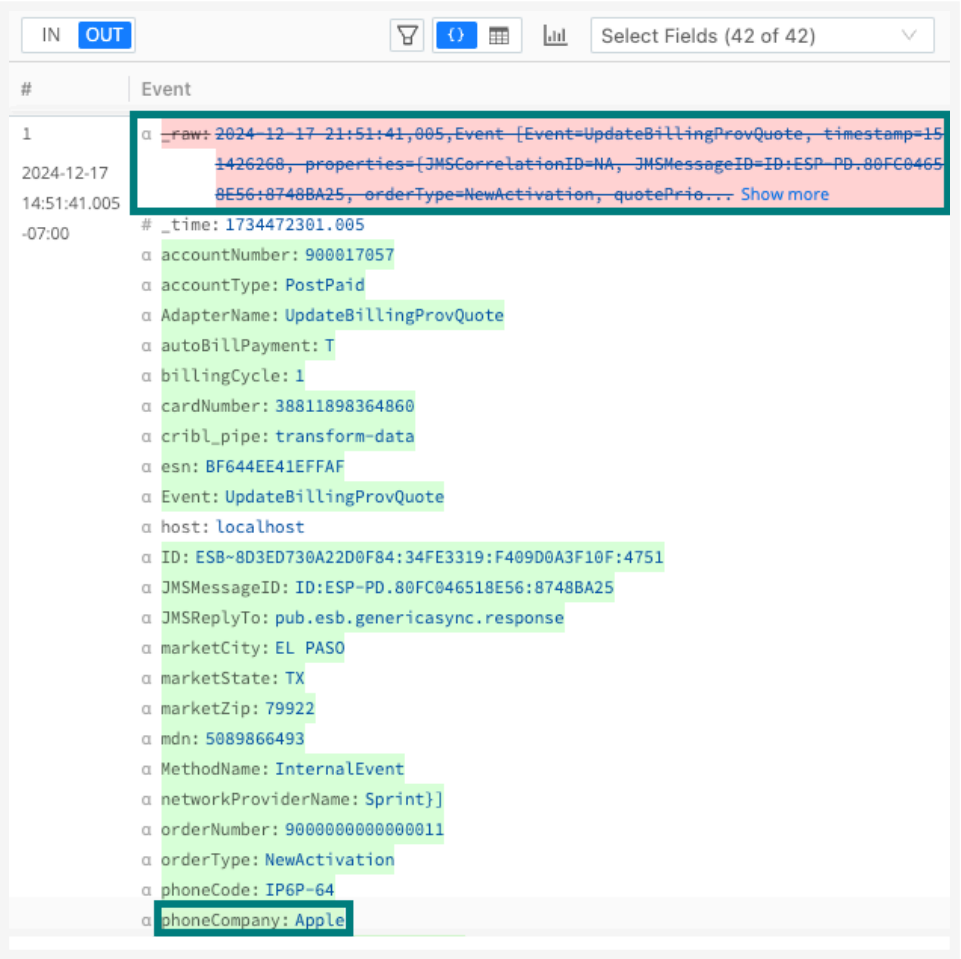
1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Then, select **Add Function**. Search for `Eval`, then select it.
2. Under **Evaluate Fields**, select **Add Field** to add a key-value field to the data event that identifies the phone company the phone is associated with.
3. In the **Name** field, type `phoneCompany`.
4. In the **Value Expression** field, paste this function: `phoneType.startsWith('iPhone') ? 'Apple' : 'Other'`.

This JavaScript expression checks if the string `phoneType` starts with `iPhone`. If it evaluates to true, it sets the return value to `Apple`. If not, it sets it to `Other`. Note the use of the operators `?` and `:` operators and how strings are enclosed in quotation marks.



Adding a field to enrich data

5. Save the changes. In the Data Preview pane, confirm that data events now contain a `phoneCompany` field that matches its `phoneType`.
6. In the Eval Function in the **Remove Fields** option, type `_raw`.
7. Save the changes and confirm that the Data Preview now shows the `_raw` field removed from each event.



The added phoneCompany field and the removed _raw field

The Eval Function reduced and enriched the data. In the next step, you will learn how to rename fields.

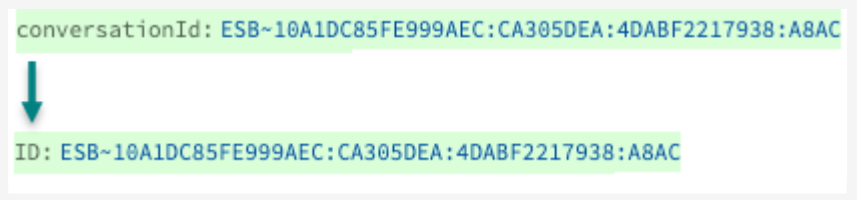
Step 8: Add a Rename Function to Rename Fields

In this step, you will use the Rename Function to change the name of a field. See the Rename Function for more information.

Changing field names can make those fields more user-friendly and also easier to read. If character counts are a concern, this Function can help reduce field characters. You will rename the `conversationId` field name to the simpler `ID` instead.

To add a Rename Function to your Pipeline:

1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Then, select **Add Function**. Search for `Rename`, then select it.
2. Under **Rename fields**, select **Add Field** to add a key-value field to the data event.
3. In the **Current name** field, type `conversationId`.
4. In the **New name** field, type `ID`.
5. Save the changes. In the Data Preview pane, confirm that the `conversationId` field has changed to the `ID` instead.



The conversationID field renamed to ID

Your Pipeline is nearly finished. In the next step, you will add the last Function.

Step 9: Add a Drop Function to Remove Unneeded Events

At this point, you have already refined our data substantially. In this step, you will add one last Function to entirely remove events that aren't of interest for a downstream service. For this tutorial, assume that you don't need data events for customers who use the prepaid monthly phone service.

You will use the Drop Function to remove any data events for customers with prepaid monthly plans. See Drop Function for more information. You will drop any data events for customers on the prepaid monthly plan by filtering for any events that do not have an `accountType` value of `PostPaid`.

To add a Drop Function to your Pipeline:

1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Then, select **Add Function**. Search for `Drop`, then select it.
2. In the **Filter** field, replace the default value with this JavaSscript expression: `accountType!='PostPaid'`.

This JavaScript expression filters for events that do not have an `accountType` value of `PostPaid` to find customers who use the prepaid monthly phone service.

3. Select **Save**.
4. In the Data Preview pane, confirm that events with the `accountType` of `PrePaid` are now struck out and faded, which indicates that Cribl Stream will drop them before forwarding the data to a downstream receiver. Verify also that events with the `accountType` of `PostPaid` remain.

As a reminder, you can turn any Function off by toggling it off in the Pipeline. Then, select **Save** and watch the Data Preview pane change again. By toggling Functions on and off, you can test and troubleshoot each Function's effect. Remember to toggle it on again before moving on to the next step in the tutorial.

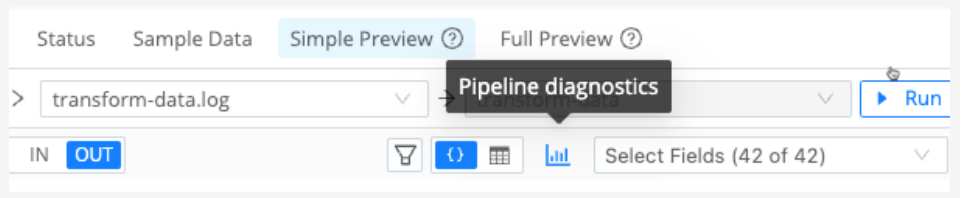
That wraps up the portion of the tutorial that transforms the data. Before moving on, you will learn how to measure the impact of the changes on your outgoing data.

Step 10: Use Pipeline Diagnostics to Measure the Impact of the Pipeline

In the previous steps, you transformed, enriched, and reduced your data. Now, you will learn how to use the Pipeline Diagnostics tool to view detailed statistics that show the impact of each Function in your Pipeline. Depending on your goals, you could use this tool to performance tune your Pipeline to get the desired results.

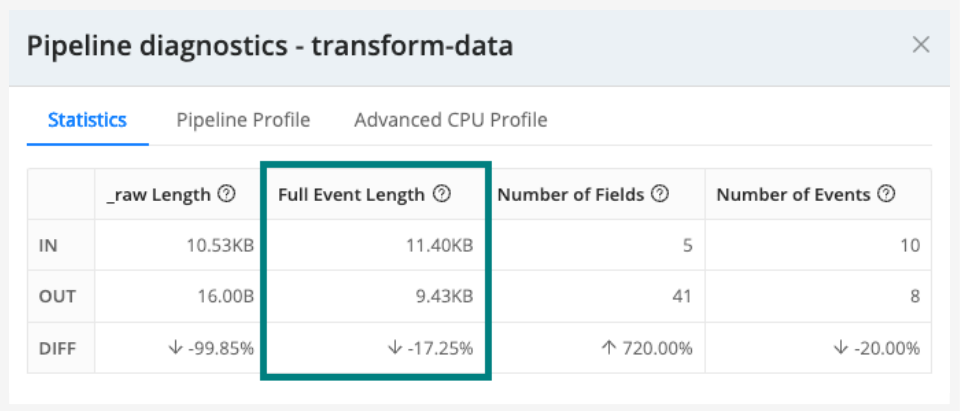
To measure the impacts of your Pipeline:

1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Then, select **Pipeline Diagnostics** button.



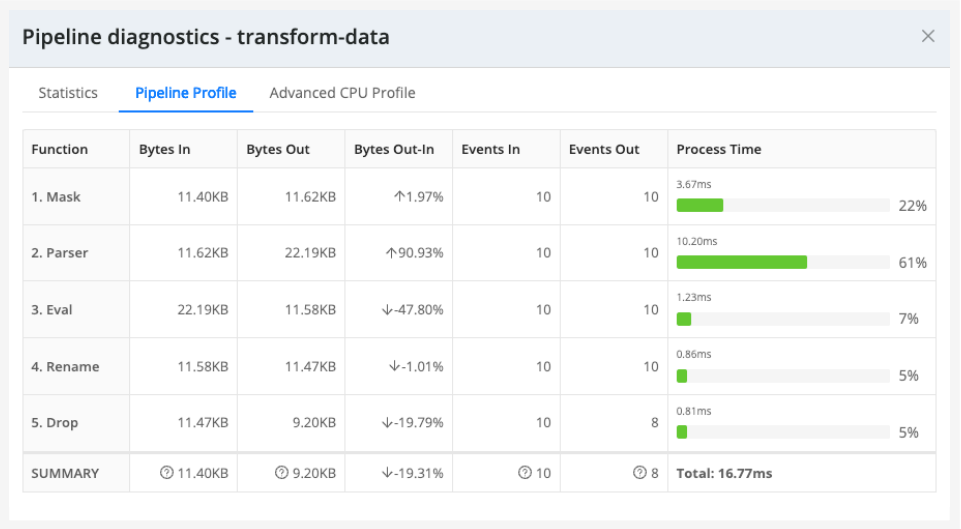
Pipeline Diagnostics

2. Confirm the impact of the Pipeline by looking at the **Full Event Length** statistic. You should see a substantial percentage reduction in the data volume.



Data reduction quantified

3. Select the **Pipeline Profile** tab to view the impacts of each Function on the data.



Impact of each Pipeline Function

You can use the Pipeline Diagnostic tool to optimize data flow and right-size the bytes in and out of the Pipeline to get your data to its desired volume threshold.

Step 11: Add and Attach a Route

Up to this point, you have built a complete, functional Pipeline. However, you have only seen its effects on a static data sample that you captured in an earlier step. To get dynamic data flowing through a Pipeline, you need to filter that data in by adding and attaching a Route.

Routes provide a way to connect processing Pipelines to Destinations that send your transformed data out to specific downstream receivers. You can maintain Pipelines and Routes separately, but each Route attaches one Pipeline to one Destination.

To add and attach a Route:

1. On the Pipelines page, select the `transform-data` Pipeline from earlier in the tutorial. Then, select **Attach to Route**. Select `Add Route`.

2. Enter a unique, meaningful **Route Name**, such as `tutorial-demo`.
3. Leave the **Filter** field set to its `true` default, allowing it to deliver all events.

A Route delivers events to a Pipeline and acts as the first stage of filtering. In production, you typically configure Routes to filter events based on characteristics such as `source`, `sourcetype`, `index`, `host`, `_time`, or others. The **Filter** field supports JavaScript expressions, including AND (`&&`) and OR (`||`) operators for precise filtering.

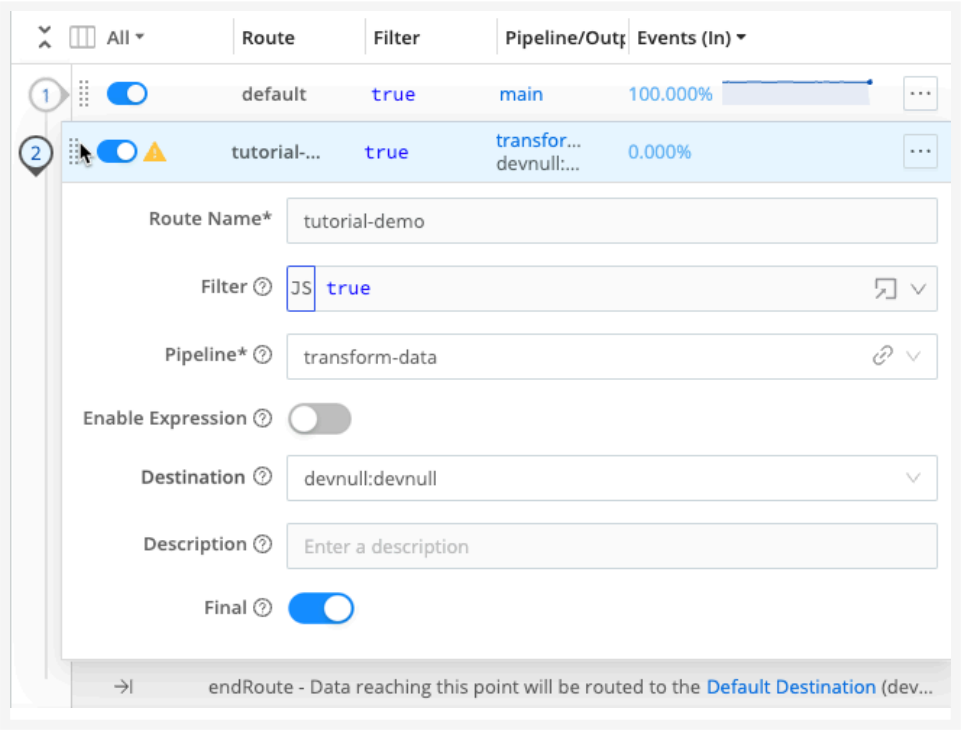
4. Set the **Pipeline** drop-down to the `transform-data` Pipeline.
5. Leave **Enable Expression** toggled off (default).

Toggle **Enable Expression** on to change the **Destination** field to a **Destination Expression** field where you can enter a JavaScript expression for your **Destination** name.

6. Set **Destination** to either `devnull` or `default`.

In this tutorial, this setting doesn't matter because the `default` points to `devnull`. However, in production, you should set this carefully.

7. Leave the **Description** empty and leave **Final** toggled on.
8. With the Route configured, you now need to move the Route so that it processes events first. Select the drag handle for this Route and drag it above the `default` Route.



1.4.3. Distributed Quick Start

This tutorial builds on our Cribl Stream Tour and Getting Started Guide by walking you through a Distributed deployment - Cribl Stream's typical deployment type for production.

For concepts and deeper details underlying the techniques presented here, see Distributed Deployment.

Requirements

To exercise these distributed features, you'll need the following prerequisites.

Licenses and Instances

This tutorial is tiered to accommodate different Cribl Stream license types:

- With a Cribl Stream Free, One, or Standard license, you'll install three Cribl Stream instances on one or multiple physical or virtual machines - one Leader Node, and two Worker Nodes.
- To do the **optional** section on adding and managing multiple Worker Groups, you'll need an Enterprise or Sales Trial license/plan.

Basic Setup

Basic building blocks are identical to the Getting Started Guide. Please refer to the following sections of that tutorial for details, as needed:

- (System) Requirements
- Download and Install Cribl Stream
- Run Cribl Stream

To set up the multiple instances you'll need, choose among the options below in Three Instances, Pick Any Medium. But first, let's lay out the division of labor in a single Worker Group.

Distributed Deployment (Identical Workers)

A Distributed deployment enables Cribl Stream to scale out to handle higher data volumes, load-balancing with failover, and parallel data processing based on conditional mapping and routing.

A single Leader Node manages multiple Worker Nodes. The Leader distributes and updates configuration on the Workers, handles version control, and monitors the Workers' health and activity metrics. The Workers do all

the data processing.

Here, we'll show how this works by configuring a Leader and two Workers. This configuration is compact, and can be demonstrated without an Enterprise (or other paid) license/plan. But you can extrapolate the same technique to setting up enterprise-scale deployments of hundreds of Workers, to handle petabytes of data.

Cribl Stream Free, One, and Standard licenses support only a single Worker Group (named `default`), so in this example, all Workers share identical configuration. With an Enterprise license/plan, you can organize Workers into multiple Groups, with varying configurations to handle scenarios like on-prem versus cloud tech stacks, or data centers in different geographic locations. For details, see Worker Groups - What Are They and Why You Should Care.

Three Instances, Pick Any Medium

You'll need to deploy three Cribl Stream instances - with SSH access - on one or more physical machines, virtual machines, or containers. If you haven't already provisioned this infrastructure, you have several alternatives, listed in the following subsections:

- Docker Containers
- Curl
- Amazon Lightsail
- AWS/EC2 (CloudFormation Optional)
- Kubernetes/Helm

Pick whichever approach will make it easiest for you to get the infrastructure launched - based on familiarity, preference, or availability.

We Don't Need No Stinkin' Permissions Errors!

Cribl's Docker containers come with Cribl Stream preinstalled. If you select any other option, be sure to both install and run Cribl Stream as the same Linux user. (For details on creating a new user - addressing both systemd and initd distro's - see Enabling Start on Boot.)

Docker Containers

You can use the Docker Compose (`docker-compose.yml`) file below to easily stand up a Cribl Stream Distributed deployment of a Leader and multiple Workers on one machine.

Before you use the Docker Compose file

1. Open the `docker-compose.yml` file in a text editor.

2. Replace both instances of `INSERT_TOKEN` with a unique, secure token value.

3. Save the updated `docker-compose.yml` file.

docker-compose.yml

```
version: '3.8'
services:
  leader:
    image: cribl/cribl:latest
    environment:
      - CRIBL_DIST_MODE=leader
      - CRIBL_DIST_LEADER_URL=tcp://INSERT_TOKEN@0.0.0.0:4200
      - CRIBL_VOLUME_DIR=/opt/cribl/config-volume
    ports:
      - "19000:9000"
    volumes:
      - "~/cribl-config:/opt/cribl/config-volume"
  workers:
    image: cribl/cribl:latest
    depends_on:
      - leader
    environment:
      - CRIBL_DIST_MODE=worker
      - CRIBL_DIST_LEADER_URL=tcp://INSERT_TOKEN@leader:4200
    ports:
      - 9000
```

Copy

When you run the `docker compose` command, the `~/cribl-config` local directory is created automatically for use as a persistent configuration store for Cribl Stream.

If you prefer to use ephemeral storage, you can delete line 8 (the `CRIBL_VOLUME_DIR` definition) and lines 11-12 (the `volumes` configuration) before running the `docker compose` command. But this will make it hard to stop and restart the same infrastructure, if you want to do the tutorial in chunks.

To deploy a Leader Node, plus (for example) two Workers already configured and wired up to the Leader, use this command:

```
docker compose up -d --scale workers=2
```

Copy

To deploy a different number of Workers, just change the `workers=2` value. By default, the above command pulls the freshest stable image (tagged `cribl/cribl:latest`) from Cribl's Docker Hub. It defaults to the following URLs and ports:

- Leader URL: `http://localhost:19000`
- Worker URLs: `http://localhost:<automatically-assigned-host-ports>`

If you're running the container itself on a virtual machine, replace `localhost` with the VM's IP address. The automatic assignment of available host-OS ports to the Workers prevents port collisions. **Within** the Docker container, these ports will forward over TCP to port 9000. To see the ports assigned on the OS, enter:

```
docker ps
```

You should see results like these:

```
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS
a3de9ea8f46f   cribl/cribl:latest    "/sbin/entrypoint.sh..."   12 seconds ago   Up 10
40aa687baefc   cribl/cribl:latest    "/sbin/entrypoint.sh..."   12 seconds ago   Up 10
df362a65f7d1   cribl/cribl:latest    "/sbin/entrypoint.sh..."   13 seconds ago   Up 11
```

Copy

The **PORTS** column shows the host-OS ports on the left, forwarding to the container-internal ports on the right. You can use the `docker_workers_N` ports if you want to log directly into Workers. In the above example:

- Worker1 URL: `http://localhost:63411`
- Worker2 URL: `http://localhost:63410`

If your Leader is crashing with two Workers, make sure you are allocating enough memory to Docker.

Once your three instances are running, proceed to Configure Leader Instance.

Curl

Use our Download page's `curl` command to directly install Cribl Stream onto your chosen infrastructure.

For x64 processors, use: `curl -Lso - $(curl https://cdn.cribl.io/dl/latest-x64) | tar zxv`

For ARM64 processors: `curl -Lso - $(curl https://cdn.cribl.io/dl/latest-arm64) | tar zxv`

Once you've configured the first Cribl Stream instance as a Leader, you can bootstrap Workers from the Leader. Or you can create Workers (with tags) from the Leader, using a `curl` command of this form:

`curl 'http://<leader-ip-or-hostname>:9000/init/install-worker.sh?token=<token>&tag=<tag1>&tag=<tag2>'`

For example: `curl 'http://localhost:9000/init/install-worker.sh?token=<token>&tag=dev&tag=test''`

Once your three instances are running, proceed to Configure Leader Instance.

Amazon Lightsail

Amazon Lightsail provides a quick, simple way to deploy Cribl Stream to AWS instances. Amazon's Get Started with Linux/Unix-based Instances in Amazon Lightsail tutorial walks you through setting up your instances and connecting via SSH.

- The free (first month) tier is all you need for this tutorial.
- As the "platform," Cribl recommends selecting **Amazon Linux 2 (default CentOS)**.
- Lightsail doesn't support IAM roles assigned to instances, or advanced load balancing, but it's adequate for this tutorial, which is not a production deployment.

Once your three instances are running, proceed to Configure Leader Instance.

AWS/EC2 (CloudFormation Optional)

You can deploy Cribl Stream's AWS EC2 instances using Cribl's CloudFormation template, see our AWS/EC2 Quick Start Guide on GitHub. Follow the **Cribl Stream Distributed** instructions. (You'll be responsible for the costs of your AWS infrastructure.)

If you prefer to deploy your own EC2 instances, the free tier is fine for this tutorial. Cribl recommends selecting **Amazon Linux 2 (default CentOS)** AMIs. Relevant instructions are linked below.

1. See any of these AWS Docs for EC2 deployment:
    - Tutorial: Getting Started with Amazon EC2 Linux Instances
    - Launching an Instance Using the Old Launch Instance Wizard
    - How Can I Create and Connect to an Amazon Linux EC2 Instance?
2. See these AWS instructions for SSH access:
    - Connect to Your Linux Instance Using an SSH Client

Once your three instances are running, proceed to Configure Leader Instance.

Kubernetes/Helm

Use Cribl's Helm charts to deploy Kubernetes pods, then proceed to the next section:

- Kubernetes Leader Deployment
- Kubernetes Worker Deployment

Configure Leader Instance

Once you've set up your Leader and Worker instances via your chosen approach above, you're ready to configure these instances and their communication.

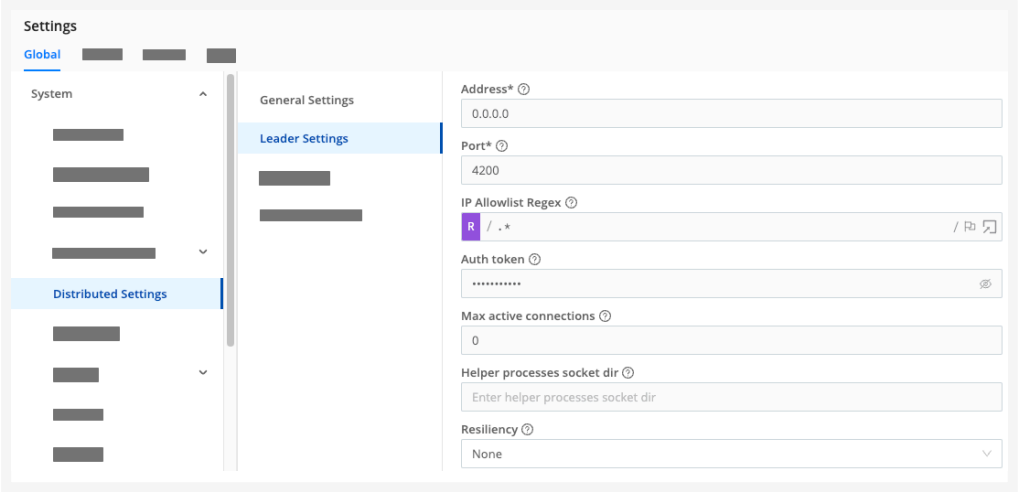
If you used an installation option like Docker Containers above, it has already preconfigured all three instances for you. This section and the next Configure Workers section will show you how to verify, and/or modify, these preset configurations.

If you want to jump ahead, your next required configuration step is Add a Source, further down.

Configure the first instance in Leader mode, and open port 4200.

1. Log into the Leader. Depending on the deployment method you chose, this will be at `http://<localhost-or-IP-address>:9000` or `http://<localhost-or-IP-address>:19000`. Use the default `admin/admin` credentials.
2. Complete the registration form.
3. In the sidebar, select **Settings**, then **Global**.
4. Select **Distributed Settings**, then **General Settings**.
5. In **Mode**, choose `Leader`.
6. Select **Leader Settings**, and make sure the **Port** is set to `4200`. (This port must be open for Workers to communicate with the Leader.)
7. Select **Save** to restart the Cribl server in Leader mode.

Keep the Leader's tab open, so that we can verify connectivity with the Workers after configuring them (in the next section).



Distributed > Leader Settings

8. Optional but recommended: In the sidebar, select **Worker Groups**. Then enable **Teleport** for your Group(s). This way, you will be able to click through from the Leader's **Workers** page to view and manage each Worker's UI, distinguished by a purple border. (This option handles authentication for you, so you don't need to manage or enter the Workers' credentials.)

Configure Workers

Next, configure the two other instances as Workers that report to the Leader to receive processing instructions. We'll configure one instance through the UI, and (optionally) bootstrap the other from the Leader.

Configure Worker via UI

1. Log into the first Worker instance. Depending on the deployment method you chose, this will be at `http://<localhost-or-IP-address>:9000` or at: `http://<localhost-or-IP-address>:<automatically-assigned-host-port>`. Use the default `admin/admin` credentials.
2. Complete the registration form, if displayed.
3. In the sidebar, select **Settings**, then **Global**.
4. Select **Distributed Settings**, then **General Settings**.
5. In **Mode**, choose `Stream: Managed Worker (managed by Leader)`.
6. Leave other settings on this tab unchanged, including **Default Group**: `default`. (This is the literal name of the single Worker Group available with free licenses.)
7. Select **Leader Settings**, and make sure the **Address** matches the domain of the Leader you previously configured.
8. Still in **Leader Settings**, display the **Auth token** value. Optionally, you can change this from the default.
9. Leave other settings unchanged, and select **Save** to restart this instance as a managed Worker.

Bootstrap Worker from Leader

You can (if you choose) configure the second Worker instance using exactly the same procedure you used just above. But here, we'll offer you a simplified version of Cribl Stream's Bootstrapping Workers from Leader procedure for downloading the config from the Leader to the second Worker:

First, switch to the terminal/console of the instance you've reserved for this second Worker.

Next, if you didn't change the default **Auth token** value when you previously configured the Leader, run this command as root user:

```
curl http://<leader-hostname-or-IP>:9000/init/install-worker.sh | sh -
```

Copy

If you cannot run as root, insert `sudo` as you pipe to the shell:

```
curl http://<leader-hostname-or-IP>:9000/init/install-worker.sh | sudo sh -
```

Copy

To instead pipe to a bash shell:

```
curl http://<leader-hostname-or-IP>:9000/init/install-worker.sh | [sudo] bash -
```

Copy

If you substituted a custom **Auth token** value on the Leader, enter:

```
curl http://<leader-hostname-or-IP>:9000/init/install-worker.sh?token=<your-custom-to
```

Copy

Or, for bash:

```
curl http://<leader-hostname-or-IP>:9000/init/install-worker.sh?token=<your-custom-to
```

Copy

The bootstrap script will install Cribl Stream into `/opt/cribl` on the target instance.

Verify Workers' Communication with the Leader

With both Workers configured, next make sure they're visible to the Leader.

1. Return to the Leader instance you previously configured.
2. In the sidebar, select **Workers**.
3. You should now see both Workers, mapped to the `default` **Group**.

If one or both Workers are missing, repeat the preceding Configure Worker via UI and/or Bootstrap Worker from Leader procedures until the Workers show up.

Otherwise, if both Workers are present, we can now configure a few more resources to get data flowing through this distributed setup.

If you're interested in details about the communication among Cribl Stream instances, see How Do Workers and Leader Work Together.

Once you've configured a Worker to point to the Leader Node, the Leader will assign the Worker a new, random admin password. This secures each Worker from unintended access. If you need to reconfigure the Worker later, you can either:

- Enable the Leader's **Teleport** option, as recommended above; or
- Reset the password on the Worker Group - see Set Worker Passwords.

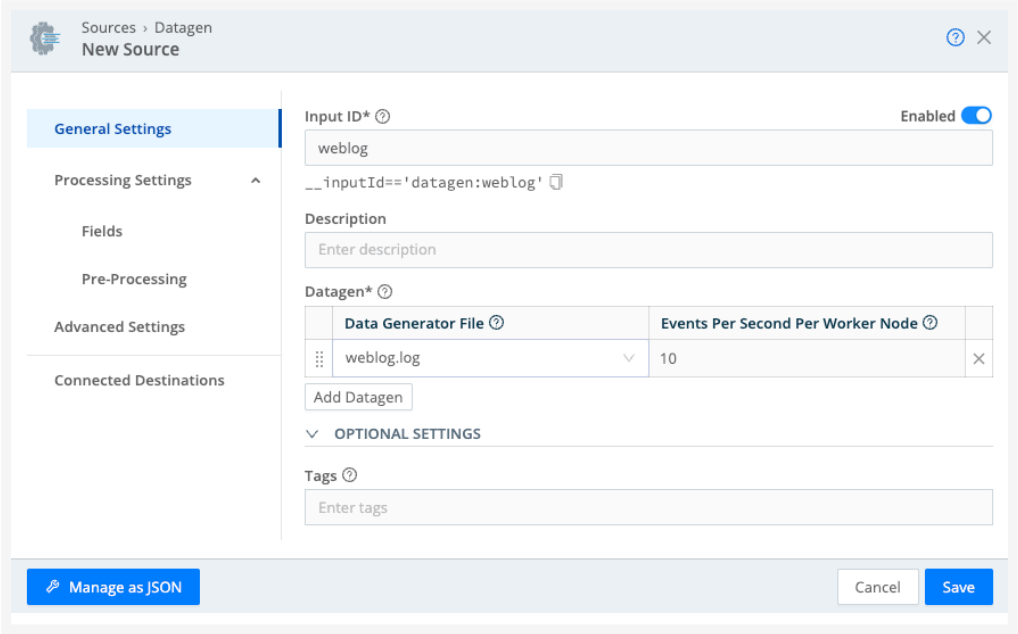
Add a Source

To minimize dependencies, this section walks you through enabling a Cribl Stream built-in Datagen Source to get some (fake) events flowing into your Workers. (This is the same approach used in our single-instance Getting Started Guide tutorial, but using a different datagen.)

If you prefer to configure Cribl Stream to receive events from a real data input that you've already set up, see our Sources topic for a link to the appropriate instructions.

1. In the Leader's sidebar, select **Worker Groups**.
2. Select the Group name. (Working with a free license, we're implicitly configuring this Source on the `default` Group.)
3. On the **Worker Groups** submenu, select **Data**, then **Sources**.
4. Select the **Datagen** tile. (You can use the search box.)
5. Select **Add Source** to open a **New Source** modal.
6. In the **Input ID** field, name this Source `weblog` (or any unique name).

7. In the **Data Generator File** drop-down, select `weblog.log`. This generates simulated log events for a website.
8. Keep the **Events per Second per Worker Node** at the default `10` EPS for now.
9. Select **Save**. In the **Enabled** column, the toggle toggled on indicates that your Datagen Source has started generating sample data.
10. Select **Commit** at the upper right, enter a commit message, and select **Commit & Deploy**. This uses Cribl Stream's `git` integration to record the `default` Group's new configuration and deploy it to your Workers.



Source (Datagen) configuration

If you'd like to verify that the Datagen is sending events, wait about a minute for them to start accumulating. Then, on the **Datagen** Sources page, select **Live** in the **Status** column for your configured Source. On the resulting **Live Data** tab, you should see events arrive within the default 10-second capture.

Add a Destination

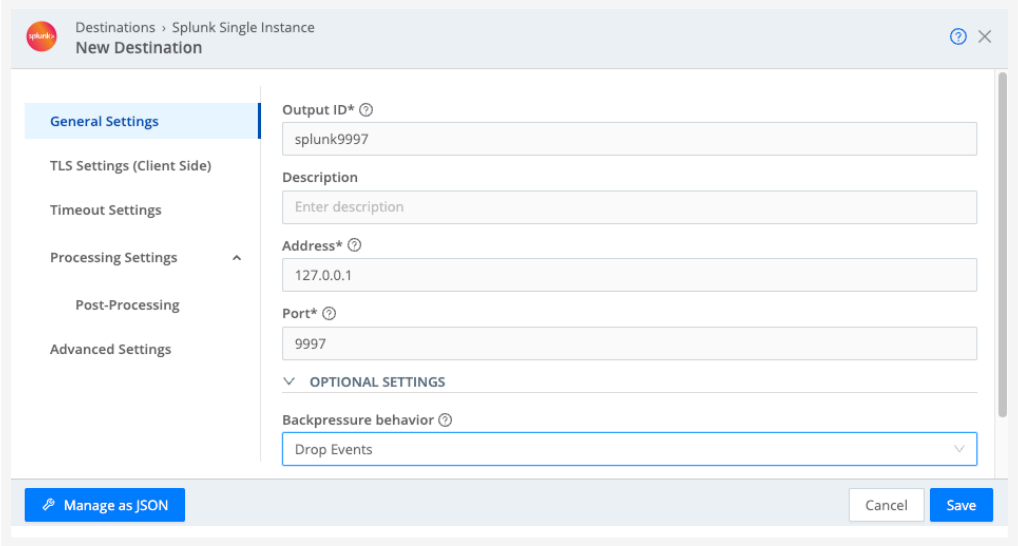
On the output side, choose any of these options:

- To configure a realistic output, but with no real dependencies and no license requirement, follow the Simulated Splunk Destination instructions just below.
- For a simpler option, jump to the Internal Destination instructions further down.
- To send data to a real receiver that you've already set up, see our Destinations topic for a link to the appropriate instructions.

Simulated Splunk Destination

Here, you'll go through the steps of configuring a typical Splunk Destination, but you'll use netcat to spoof the receiver on port 9997.

1. In the Cribl Stream Leader's UI, configure a Splunk Single Instance Destination, following these instructions. For this simulated output:
    - Set the **Address** to `127.0.0.1` (that is, localhost).
    - Do not precede this IP address with any `http://` or `https://`
    - Leave the **Port** at the default `9997`.
    - Set the **Backpressure behavior** to `Drop Events`.



A glorious spoofed Splunk Destination config

2. In the **Version Control** menu at the upper right, select **Commit**, enter a commit message, and confirm the commit.
3. Select **Deploy** to deploy this new configuration to your Workers.
4. In the first Worker instance's terminal/console, shell in, then enter `cd /opt/cribl/bin` to access Cribl Stream's CLI.
5. Enter `nc -h` to check whether netcat is installed on this Linux instance.

If the command fails, follow your Linux distro's steps for installing netcat. (For example, for Ubuntu instructions, see Docker Notes below.)

6. Enter `./cribl nc -l -p 9997` to have netcat listen on port 9997, but simply discard data.

Docker Notes

If you're using a container like Docker, before shelling in at step 4 above, you'll need to first open a shell inside that container: `docker exec -it <CONTAINER ID> /bin/bash`

In the above Docker Containers deployment example, you'd want to open the shell on the `docker_workers_1` container, whose `<CONTAINER ID>` was `a3de9ea8f46f`.

Cribl's Docker containers currently run Ubuntu 20.04.6. You can install netcat with this sequence of commands:

```
apt update apt install netcat
```

The data you'll now see displayed in the terminal will be gibberish, because of Splunk's proprietary data format.

If data isn't flowing, you might need to restart Workers. You can do this through Cribl Stream's UI. With Docker containers, use `docker compose down`, followed by `docker compose up`.

You can streamline future Commit and Deploy steps by entering a **Default commit message**, and by collapsing actions to a combined **Commit and Deploy** button. Both options are available at **Settings** > **Global** > **System** > **Git Settings** > **General**.

Internal Destination

As an alternative to the Splunk instructions above, you can configure Cribl Stream's built-in DevNull Destination to capture events and discard them. (This is the same Destination used in our single-instance Getting Started Guide tutorial.)

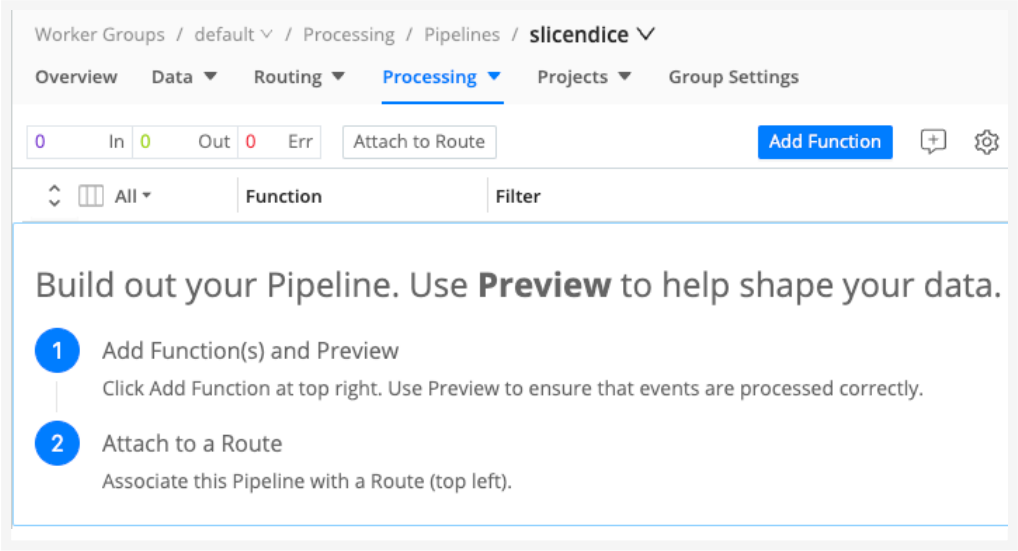
1. In the sidebar, select **Worker Groups**.
2. Select the Group name. (Working with a free license, we're implicitly configuring this Source on the `default` Group.)
3. On the **Worker Groups** submenu, select **Data**, then **Destinations**.
4. Select the **DevNull** tile. (You can use the search box.)
5. On the resulting **devnull** row, look for the **Live** indicator under **Status**. This confirms that the **DevNull** Destination is ready to accept events.
6. In the Destinations list on the left, select the **Default** Destination at the top.
7. Verify that **Default Output ID** points to the **devnull** Destination we just examined.
8. Select **Commit** at the upper right, enter a commit message, and select **Commit & Deploy**.

Add a Pipeline

To complete this Distributed deployment with realistic infrastructure, let's set up a Pipeline and Route.

If you've already done the Getting Started Guide, you've already created a `slicendice` Pipeline. In the following steps, skip ahead to adding an Eval Function.

1. From the **Worker Group** submenu, select **Processing**, then **Pipelines**.
2. Select **Add Pipeline**, then **Add Pipeline** again.
3. In the new Pipeline's **ID** field, enter a unique identifier (for example, `slicendice`).
4. Optionally, enter a **Description** of this Pipeline's purpose.
5. Select **Save**.

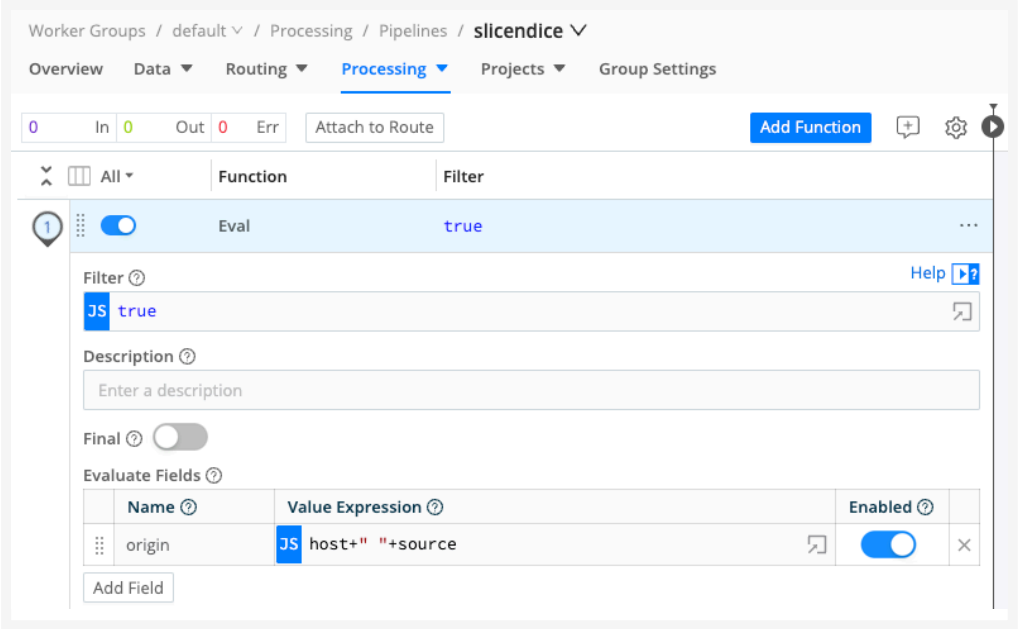


Pipeline saved

Add an Eval Function

Now add an Eval Function to the Pipeline:

1. At the top of the **Pipelines** page, select **Add Function**.
2. Search for the `Eval` Function, and select it.
3. In the new Function's **Evaluate Fields** section, select **Add Field**.
4. In the **Name** column, name the field `origin`.
5. In the **Value Expression** column, enter: `host+" "+source`
   This new `origin` field will concatenate the host and source fields from incoming events.



Adding Eval Function to Pipeline

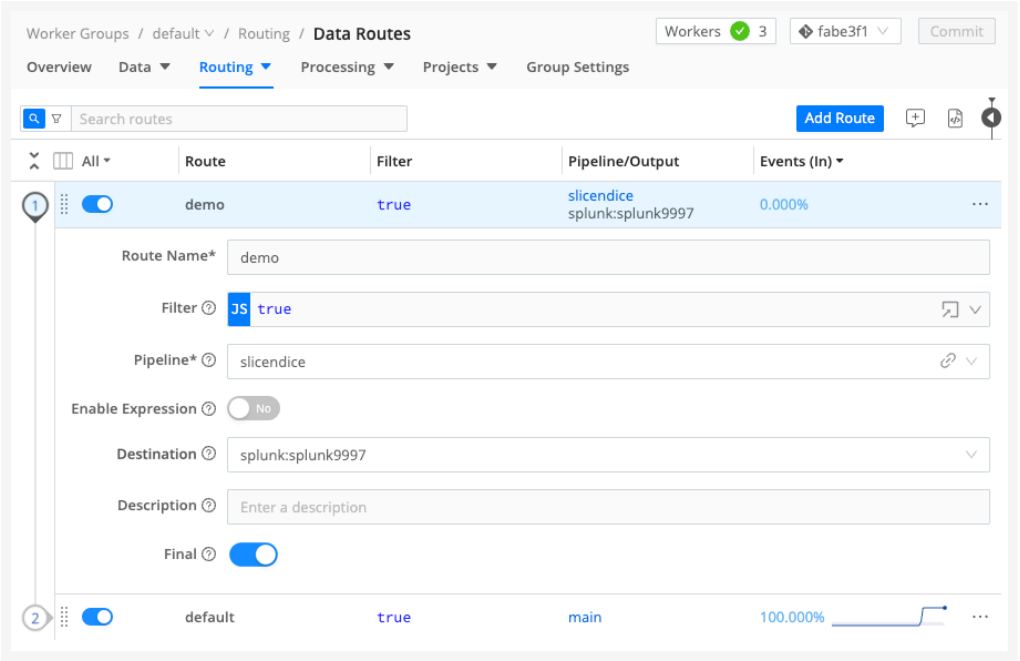
6. Select **Save** to store the Function's configuration.
7. **Commit** and **Deploy** the new Pipeline configuration.
8. Optionally, open the right pane and select **Capture Data** to verify throughput.

Add a Route

If you've already done the Getting Started Guide, you've already created a `demo` Route, attached to the `slicendice` Pipeline. In the following steps, just modify the Route to send data to the new Destination you configured above.

1. At the top of the **Pipelines** page, select **Attach to Route**.
2. Select **Add Route**.
3. Enter a unique **Route Name**, like `demo`.
4. Leave the **Filter** field set to its `true` default, allowing it to deliver all events.
5. Set the **Pipeline** drop-down to our configured `slicendice` Pipeline.
6. Set the **Destination** drop-down to the Destination you configured above. If you boldly chose the Simulated Splunk Destination, this will be named something like `splunk:splunk9997`.
7. You can leave the **Description** empty, and leave **Final** toggled on.
8. Grab the new Route by its left handle, and drag it to the top of the Routing table, so that our new Route will process events first. You should see something like the screenshot below.
9. Select **Save** to save the new Route configuration.
10. **Commit** and **Deploy** your changes.

11. Still assuming you configured a simulated Splunk output, look at the terminal where you're running netcat. You should now see events arriving.

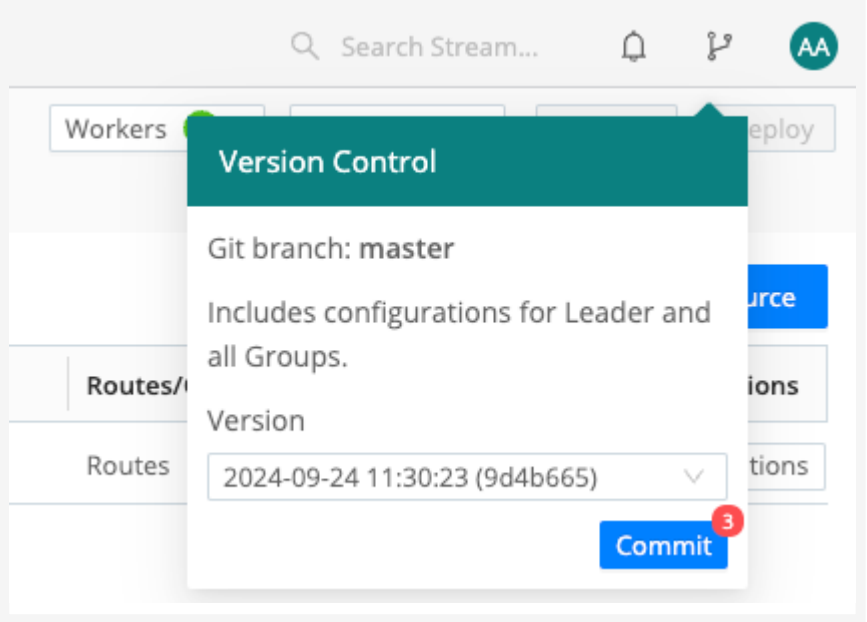


Route attached to Pipeline and Destination

Managing Workers (Scaling)

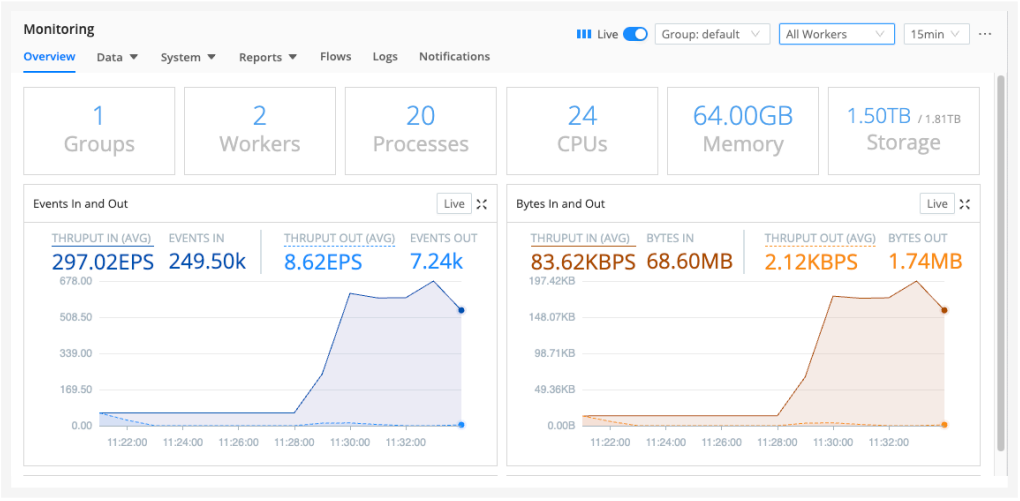
With all our infrastructure in place, let's look at how a Cribl Stream Distributed deployment scales up to balance the incoming event load among multiple Workers.

1. In the Leader's sidebar, select **Worker Groups**.
2. Select the Group name. (Working with a free license, we're implicitly configuring this Source on the `default` Group.)
3. On the **Worker Groups** submenu, select **Data**, then **Sources**.
4. Find the **Datagen** Source you configured earlier and select it.
5. Change the **Events per Second per Worker Node** from the default `10` EPS to a high number, like `200` EPS.
6. Select **Save**, then **Commit** and **Deploy** this higher event load.
7. From the **Version Control** menu at the top tight, select **Commit** to commit the Leader's newest config version. In the resulting modal, select **Commit** again to confirm.
   This uses Cribl Stream's `git` integration to save a global configuration point for your whole deployment, which you can roll back to.



Committing Leader's config

8. In the sidebar, select **Monitoring**.
9. Watch as the following changes unspool over the next few minutes:
    - The **CPU Load Average (1 min)** will spike as the higher event volume floods the system.
    - The **Workers** indicator at th top will drop to `0` as the Workers restart with the new configuration you've deployed to them, and then rebound to `2`.
    - If you use the **All Workers** drop-down at upper right to toggle between your two individual Workers, you should see that Cribl Stream is balancing roughly equal event loads among them.



10. To confirm that both Workers are `alive`: In the sidebar, select **Workers**.
11. To wrap up, repeat this procedure's first six steps to set the Datagen's rate back down to `10` EPS, and then save, commit, and deploy all changes.

What Have We Done?

This completes your setup of a basic Distributed deployment, using a free license, and configuring a single Worker Group of two identically configured Worker Processes. (You can extrapolate these same techniques you've just mastered to spin up virtually any number of Workers in the same way.)

To see how you can set up multiple Worker Groups - with separate configurations optimized for separate data flows - continue to the next section, noting its licensing prerequisites.

If you're deferring or skipping that option, jump ahead to Cleaning Up.

Add and Manage Multiple Worker Groups (Optional)

To add and manage more than one Worker Group - everything in this optional section - you'll need an Enterprise or Sales Trial license for your on-prem Cribl Stream deployment, or an Enterprise (or equivalent) plan for your Cribl.Cloud Organization. For details on adding multiple Worker Groups on Cribl.Cloud, see Cribl.Cloud Worker Groups.

See Licensing for how to acquire and install one of the above license types. Install the license on your Leader instance, and then commit this as a Leader config change (top nav's **Global Config** dialog), before you proceed.

Here, we'll build on the infrastructure we've created so far to:

- Configure Mapping Rules.
- Verify how Cribl Stream balances large data volumes among Worker Processes.
- Add a second Worker Group, data Source, and Destination.
- Add a second Pipeline and attach it to its own Route.
- Reconfigure Mapping Rules to send each Source's data through a separate Group.

To keep this Quick Start tutorial focused on techniques, rather than on configuring lots of infrastructure, we'll assign just one Worker to each Worker Group - one of the two Workers we launched above. But in production, you'll be able to apply the same principles to setting up any number of Worker Groups, with any number of Workers.

Multiple-Group Setup

With an Enterprise or Sales Trial license/plan, Cribl Stream's UI adds some extra features.

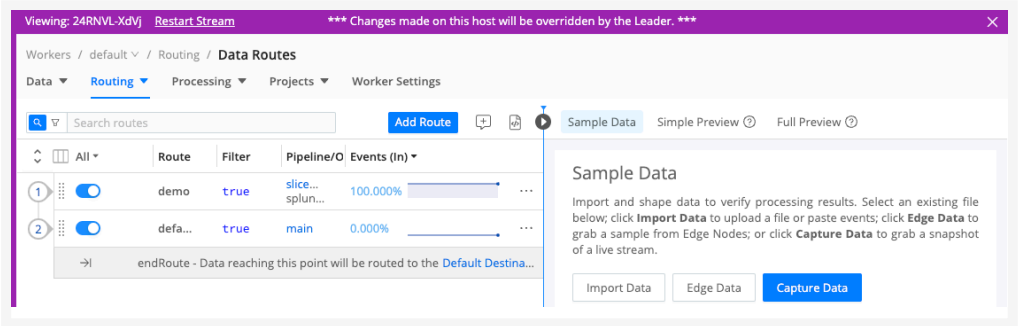
The **Worker Groups** page now supports multiple Groups. (For now, you still see only the single `default` Group - but we'll change that a few sections down.)

On Cribl.Cloud, the **Worker Groups** page adds extra columns. You'll see each Group's Worker type (customer-managed / hybrid versus Cribl-managed / Cribl.Cloud) and, for Cloud Groups, the

> infrastructure provider (AWS or Azure), anticipated ingress rate and Provisioned status. For details, see Cribl.Cloud Worker Groups.

Note that if you've enabled Worker UI access, you can click directly through to each of your Workers. This feature will come in handy just below. To try it out:

1. Either select **Workers** in the top right of the **Worker Groups** page, or select **Workers** in the sidebar.
2. Select the desired Worker's link in the **GUID** column.
3. A purple header indicates that you're viewing a Worker's UI.
4. To return to the Leader, select any option in the sidebar.



Leader's remote view of Worker's UI

Check/Restart Workers

For the remaining steps, we want to make sure both our Workers (configured earlier) are up. Cribl Stream's top header should indicate `2 WORKERS`. You can verify that they're `alive` by selecting **Workers** in the sidebar, as shown earlier. If so, proceed to Map Groups by Config.

If either or both Workers are down, restart them:

1. Make sure you've enabled Worker UI access. (It's time!)
2. Click through to a dormant Worker's UI.
3. When you see that Worker's UI (purple header), select **Restart Stream**.
4. Confirm your choice, and wait for a confirmation message to appear for a few seconds.
5. Return to the Leader's UI.
6. If the other Worker is down, repeat the above steps to restart it as well.

Map Groups by Config

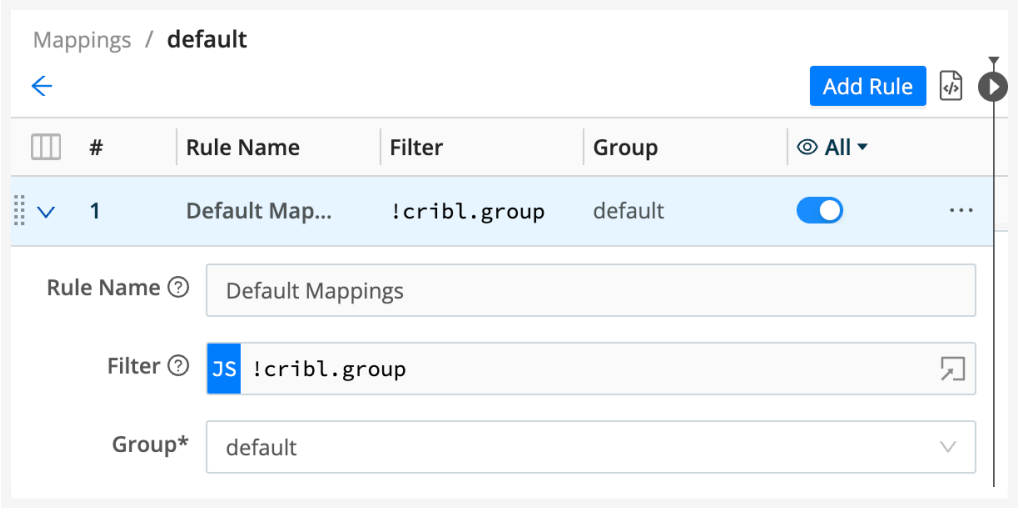
With both Workers confirmed up, let's look at how Cribl Stream has automatically mapped all these existing workers to the `default` Group.

1. In the sidebar, select **Mappings**.

2. You now see a default Mapping Ruleset, also literally named `default`. Select it.

A Cribl Stream Leader can have multiple Mapping Rulesets configured, but only one can be active at a time.

3. This `default` Ruleset contains one initial Rule, literally named `Default Mappings`. Expand its accordion as shown below.
   (The `default` Ruleset and Rule naming are separate from the `default` Group's naming. All of these out-of-the-box starting configurations have been named...literally.)



Default Mapping Rule

Below the **Rule Name**, a Mapping Rule has two functional fields:

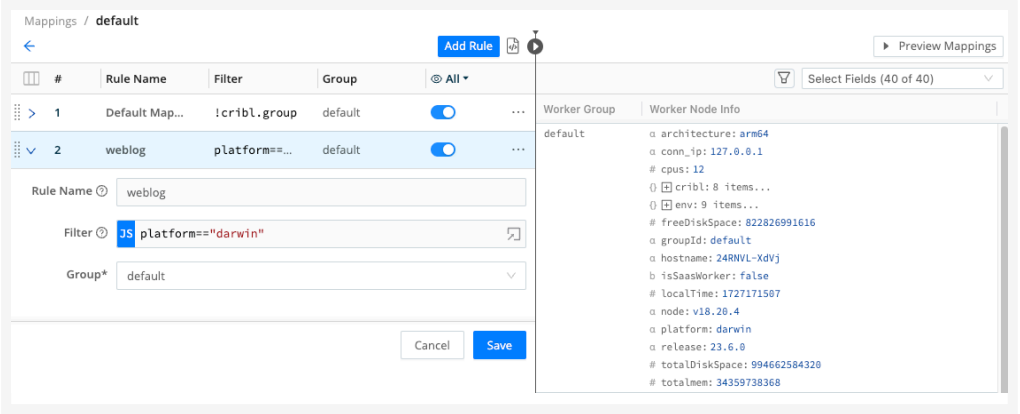
- **Filter**: `!cribl.group` - this value expression specifies "Not already assigned to a Worker Group."
- **Group**: `default` - this value specifies "Assign to the `default` Group."

So this is a catch-all rule. By following it, the Leader has assigned all (both) registered Workers to the `default` Group.

Let's make a more-specific rule, mapping a specific Worker (by hostname) to this Group, which receives events from our `weblog` Datagen Source.

Add Mapping Rules

1. Select **Add Rule**. Then configure the new Rule as shown below:
   - **Rule Name**: `weblog` - for simplicity.
   - **Filter**: `platform=="<this-worker's-platform>"` - get this value from the right Preview pane. Look for the `platform` field's value. In the example shown below, we got `"darwin"`.
   - **Group**: `default` - this is the only option available.



A specific Mapping Rule by Filter condition

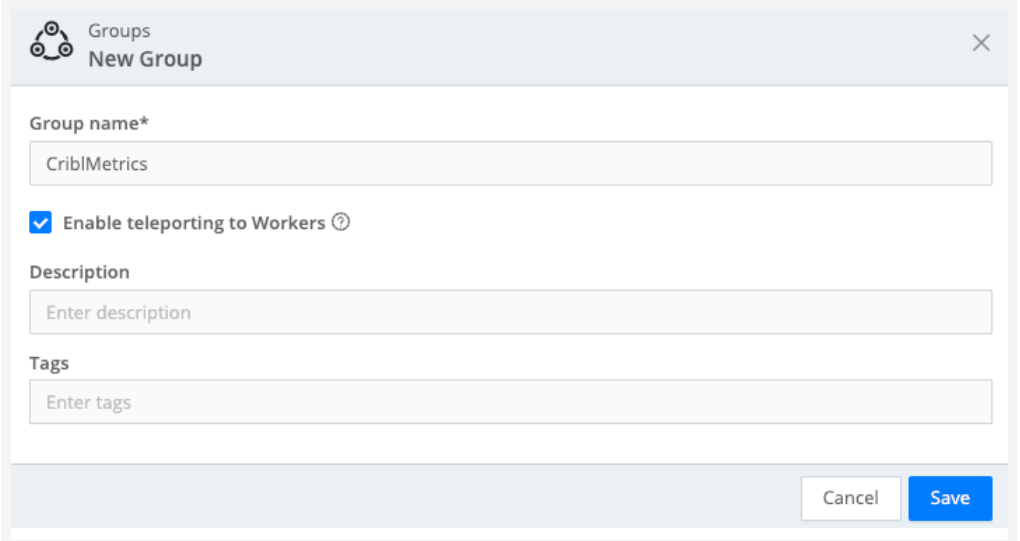
2. Select **Save** to add this Rule.
3. Confirm the warning that changes will take effect immediately.
4. From the **Version Control** menu, select **Commit** to commit the Leader's new config.

For more Mapping Rules/Rulesets details and examples, see Distributed Deployment.

Next, we'll add a second Worker Group; add a second Source (relaying Cribl Stream's internal metrics); and then add another Mapping Rule, to map our second Worker to the new Group.

Add Another Worker Group

1. In the Leaders' sidebar, select **Worker Groups**.
2. Select **Add Group**.
3. Name the new Group `CriblMetrics` to match its purpose.
4. Select **Enable teleporting to Workers**.

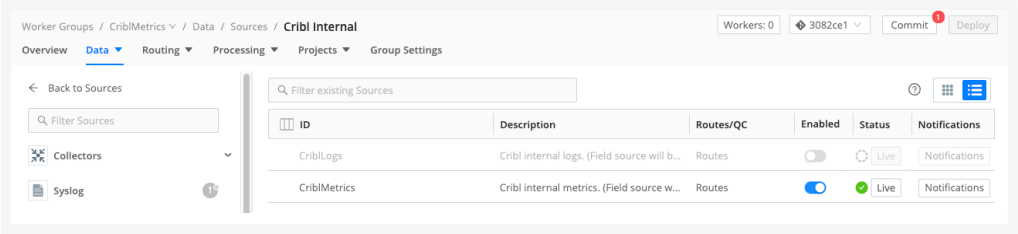


5. Save the Group.
6. Select **Deploy** on the new Group's row, and confirm your choice. The new Group should deploy immediately.
7. From the **Version Control** menu, select **Commit** to commit the new config on the Leader.

Add Another Source

On the new Group, we'll now enable a **Cribl Internal: Metrics** Source, representing a second data type.

1. In the sidebar, select **Worker Groups** , and then select the new `CriblMetrics` Group.
2. On the **Worker Groups** submenu, select **Data**, then **Sources**.
3. Select the **Cribl Internal** tile.
4. Enable the `CriblMetrics` Source and confirm.



CriblMetrics Source enabled

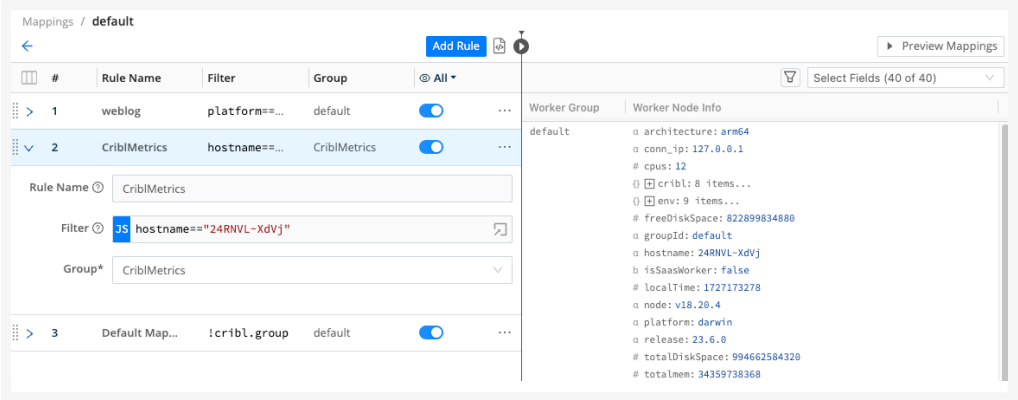
5. Select **Commit** to commit the new config on the Leader and Group.
6. From the **Version Control** menu, select **Commit** to deploy this new configuration to your Workers.

Map Workers to Groups

Now let's map this new group to the CriblMetrics Source's incoming events:

1. In the sidebar, select **Mappings**.
2. Select the `default` Mapping Ruleset to open it.
3. Select **Add Rule**. Then configure the new Rule as shown below:
    - **Rule Name**: `CriblMetrics` - for simplicity.
    - **Filter**: `hostname=="<this-worker's-hostname>"` - get this value from the right Preview pane. In the second Worker down, look for the `hostname` field's value. In the example shown below, we got `"2ca68fec7de0"`.
    - **Group**: `CriblMetrics` - this is the Group we want to map.

4. Move the `Default Mappings` rule to the bottom of the Ruleset, reflecting its catch-all function.
5. Select **Save** to store the new configuration.
6. Confirm the warning that changes will take effect immediately.

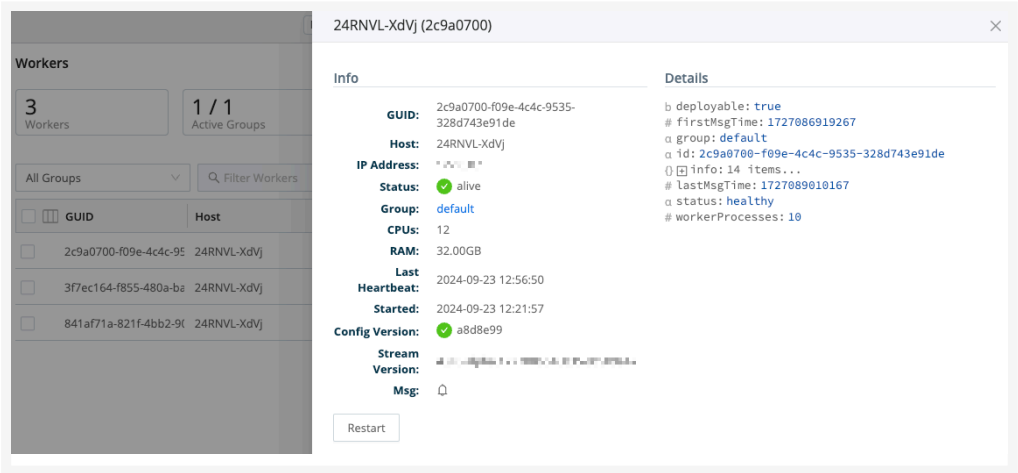


Adding a Rule to map the CriblMetrics Worker to its own Group

7. From the **Version Control** menu, select **Commit** to commit the Leader's new config.

We now have two data Sources and two Worker Groups - one each for (Web) logs versus (Cribl Internal) metrics - along with two Mapping Rules to map data accordingly. To confirm the Workers' assignment to the two Groups, select **Workers** in the sidebar.

To confirm further details about the Workers, in the sidebar, select **Workers**, and on the resulting **Workers** page, click anywhere on the Worker Node's row to reveal more details:



Worker Node Details

Configure Metrics Output

With incoming metrics now mapped to our second Worker Group, we next need to configure this Group's output. Here, we'll rely on a metrics-oriented Pipeline and a Destination that ship with Cribl Stream, and create a new Route to connect everything up.

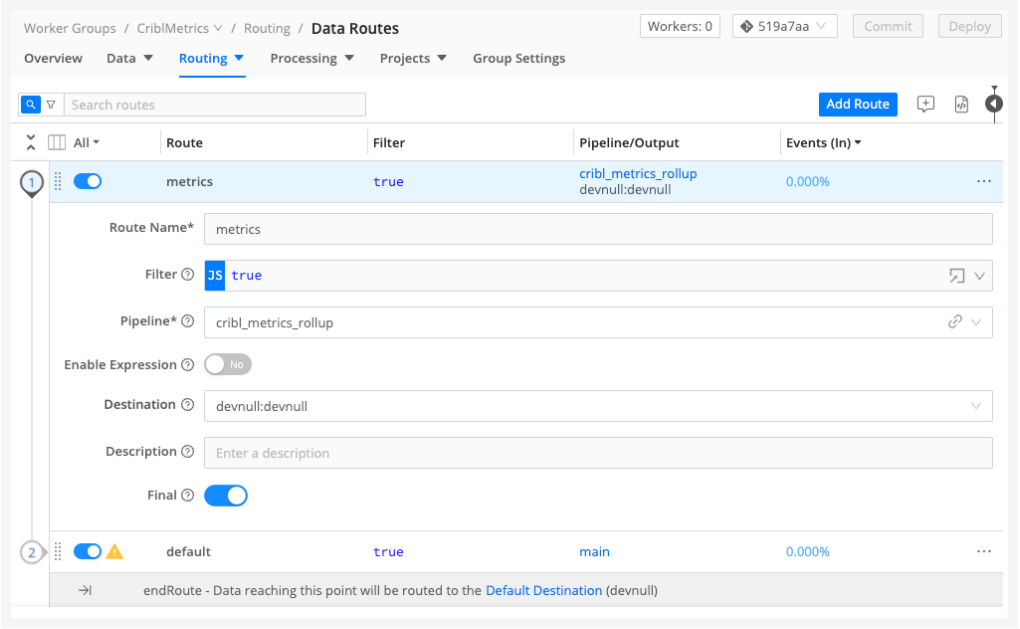
Examine the Metrics Pipeline

1. In the sidebar, select **Worker Groups** , and then select the `CriblMetrics` Group.
2. On the **Worker Groups** submenu, select **Processing**, then **Pipelines**.
3. Find the `cribl_metrics_rollup` Pipeline, and click it to expand it.
4. Expand this Pipeline's Functions (including Comments) to see its configuration. It's preconfigured with a Rollup Metrics Function to aggregate metrics to a 30-second time window. Next is an Eval Function that filters for Cribl (Cribl Stream) internal metrics and tags them on outgoing events with a new field.

Add Another Route

We'll connect this existing Pipeline to a new Route:

1. At the top of the `cribl_metrics_rollup` Pipeline page, select **Attach to Route**.
2. Select **Add Route**.
3. Enter a unique **Route Name**, like `metrics`.
4. Leave the **Filter** field set to its `true` default, allowing it to deliver all events.
5. Make sure the **Pipeline** drop-down is set to `cribl_metrics_rollup`.
6. As the **Destination**, select our old friend `devnull:devnull`.
   This is Cribl Stream's preconfigured Destination that simulates a downstream service while simply dropping events.
7. You can leave the **Description** empty, and leave **Final** toggled on.
8. Grab the new Route by its handle, and drag it to the top of the Routing table, so that our new Route will process events first. You should see something like the screenshot below.
9. Select **Save** to save the new Route configuration.
10. **Commit** and **Deploy** your changes.



Cribl metrics Route, Pipeline, and Destination wired up

Verify the Multi-Group Deployment

From the sparkline on the Route you just configured, you can already see that metrics data is flowing all the way "out" of Cribl Stream - simulated here by the DevNull Destination.

To verify the whole configuration you've created, in the sidebar, select **Monitoring**. On the **Monitoring** page, use the **All Worker Groups** drop-down (upper right) to toggle between the two Worker Groups to see the division of labor:

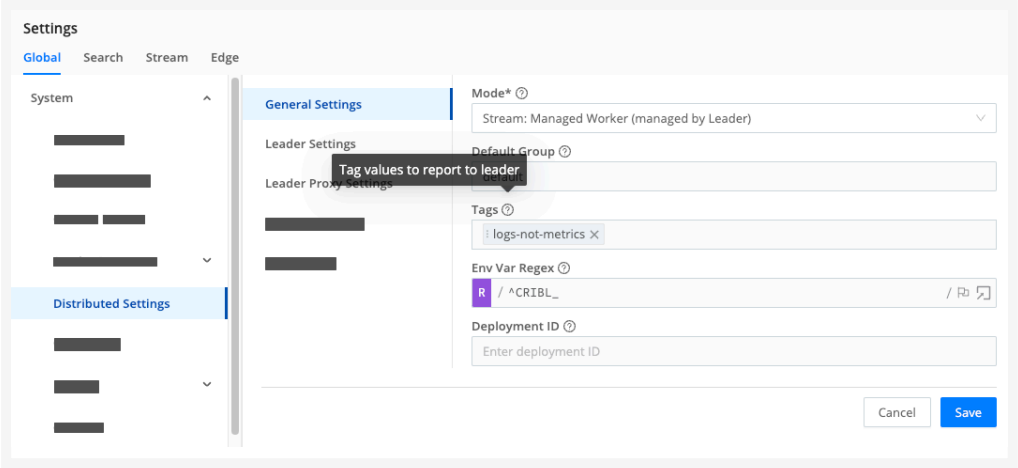
- Group `default` (the out-of-the-box Group we configured first) handles the `weblog.log` data.
- Group `CriblMetrics` handles the metrics data.

All the Distributed Things - Review

Before a final section where you can tear down your infrastructure, here's a recap of the simple (but expandable) distributed model we've created, with some ideas for expanding it:

- A Distributed deployment enables Cribl Stream to scale out to higher data volumes. This load-balancing occurs even with a single Worker Group, in which all Workers share the same configuration.
- By adding multiple Worker Groups, you can partition Worker Nodes (and their data flows) by different configurations. In this demonstration, we simply mapped Workers to Groups by the Workers' `hostname`. But you can map by a range of arbitrary criteria to meet your production needs.
- For example: Different Groups can be managed by different teams. You can filter on DNS rules to send relevant data to the relevant team's Group.
- Different Groups can also maintain configurations for different regions' data privacy or data retention requirements.

- You can also Map workers arbitrarily using Tags and other options.



Setting arbitrary Tags on a managed Worker

More About Worker Groups, Nodes, and Processes - Definitions

Cribl Stream refers to "Workers" at several levels. Now that you've been initiated into building distributed deployments, here's a guide to the fine points of distinguishing these levels:

- A **Worker Group** holds one or multiple **Worker Nodes**.
- Each Worker Node functions like an independent Cribl Stream Single-instance deployment. Worker Nodes don't coordinate directly with each other - they're coordinated only through communication to/from the Leader Node.
- Each Worker Node contains a configured number of **Worker Processes**. Unlike the above grouping abstractions, this is the level that actually processes data. To load-balance among Worker Processes, Cribl Stream's API Process round-robins incoming connections to them.

1.5. Onboard Faster With Packs

As you get started with Cribl Stream, consider using Packs to help streamline your onboarding process and get more value out of your Cribl deployment quickly.

Packs are self-contained, portable bundles of Cribl objects with pre-built logic that solves typical or specific data processing challenges. They can include configurations or example logic for a variety of objects, such as Routes, Pipelines, and Knowledge objects.

On the Cribl Packs Dispensary, Cribl officially publishes and maintains a variety of validate Packs that you might use to solve a particular challenge or process data from a variety of vendors, including many that do not yet have full Cribl Stream integrations. The Dispensary also hosts validated Packs submitted from the larger Cribl developer and user community.

Search and find for Packs that match your specific use case. Then download and import these Packs to:

- **Reduce manual effort for common vendors and log formats:** Find Packs for validated solutions created by Cribl employees and community members to skip the time-consuming process of writing complex logic for well-known log formats. Instead, import a Pack to begin ingesting, parsing, and enriching logs for common vendors quickly.
- **Learn Cribl best practices:** Use configurations created by experts from Cribl and the community to learn how to build an effective solution for common data processing challenges.
- **Get value from Cribl more quickly:** Move from initial source connection to usable, parsed data more quickly by using pre-programmed logic in Packs that you can adapt for your specific needs.
- **Simplify complex collections:** Use REST Packs to streamline configurations for the REST Collector. These officially supported Packs eliminate manual effort by providing validated authentication, pagination, and event-breaking settings tailored to specific upstream services that do not yet have a dedicated Source or Collector.
- **Perform common data processing tasks:** Use Packs for use cases such as normalizing timestamps, flattening JSON objects, or masking sensitive PII.

This page describes how to use Packs to accelerate the onboarding process with Cribl Stream. However, you can also use Packs for advanced environment templating as your Cribl Stream deployment becomes more complex and needs to scale. See Pack-Based Configuration Management for more information about this advanced configuration management use case.

1.5.1. Get Pre-Built Pack Content

Packs created by Cribl and the Cribl Community let you speed up configuring your data flows by providing ready-made solutions for many common integrations and use cases.

On the Cribl Packs Dispensary, Cribl officially publishes and maintains a variety of validate Packs that you might use to solve a particular challenge or process data from a variety of vendors, including many that do not yet have full Cribl Stream integrations. The Dispensary also hosts validated Packs submitted from the larger Cribl developer and user community.

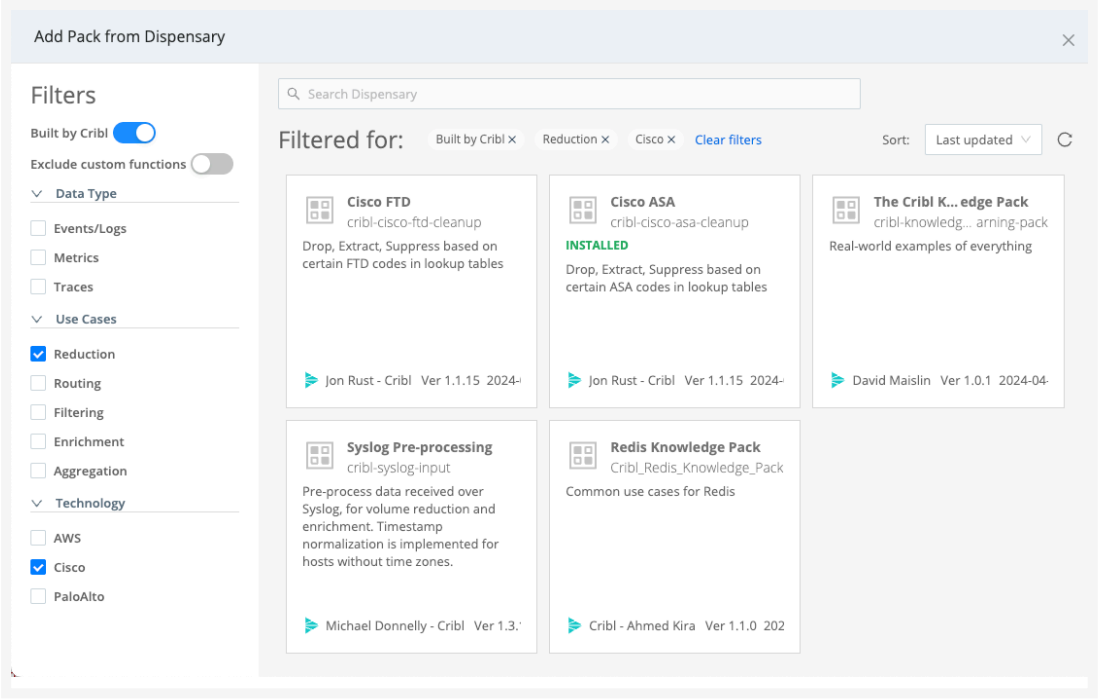
Packs created by Cribl and the Cribl Community let you speed up configuring your data flows by providing ready-made solutions for many common integrations and use cases. These Packs are hosted on the Cribl Pack Dispensary. You can download and customize each Pack as needed.

This page describes how to use Packs to accelerate the onboarding process with Cribl Stream. However, you can also use Packs for advanced environment templating as your Cribl Stream deployment becomes more complex and needs to scale. See Packs for more information about this advanced configuration management use case.

Add a Pack From the Dispensary

To import a Pack from the Dispensary directly into your deployment:

1. Navigate to the **Packs** page.
2. Select **Add Pack** at the upper right and select **Add from Dispensary**.
3. The Packs Dispensary will open in a drawer.

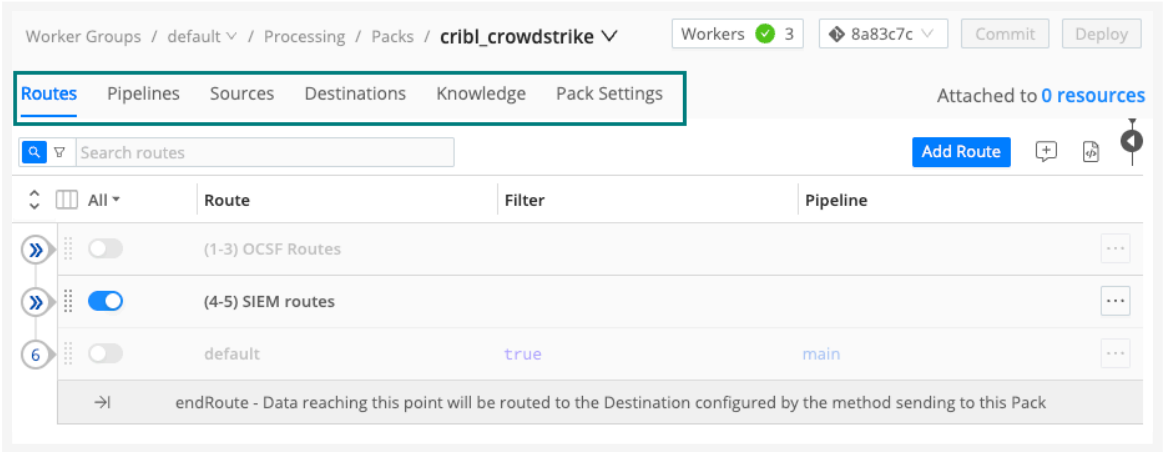


Cribl Packs Dispensary

4. Using the drawer controls, browse or search for the Pack(s) you want. (You can use the check boxes at the left to filter by data type, use case, and technology.)
5. Select any Pack tile to display its details page with its README. This will typically outline the Pack's purpose, compatibility, requirements, and installation.
6. To proceed, select **Add Pack** on the Pack details page.
7. If the Pack you are importing binds variables to any fields, you will see a **Configure Variables** button in the notification about successful import.
   Select it to go to the **Variables** page, where you can configure the values for the variables for your deployment.

Explore and Customize the Pack

Click inside the new Pack. In the Pack submenu you can see all the components that the Pack contains: Routes, Pipelines, Sources, Destinations, and Knowledge Objects.



Pack menu showing Routes, Pipelines, Sources, Destinations, and Knowledge Objects

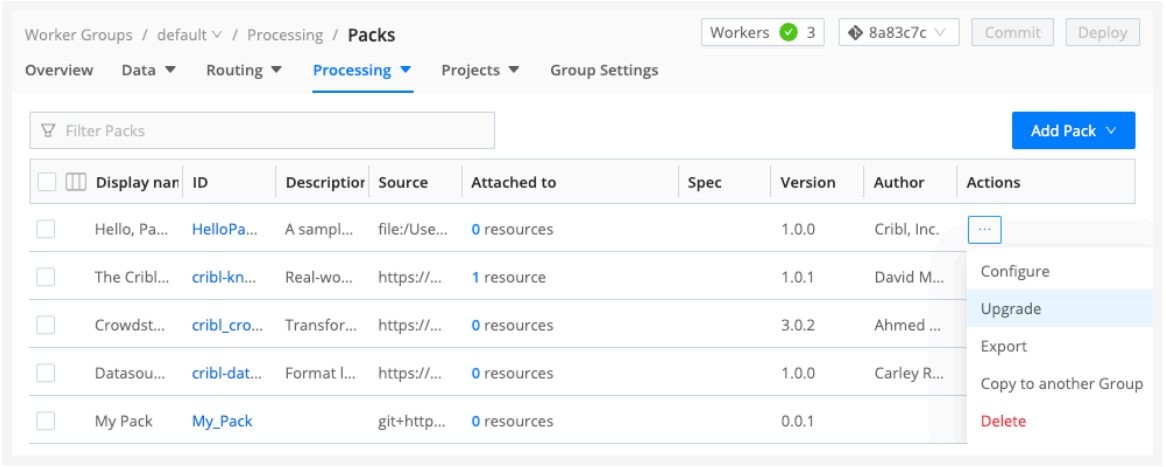
At this point you can edit any elements in the Pack to customize it to your needs. Do it the same way you would when working directly in any Worker Group.

Remember that all elements inside the Pack, including Knowledge Objects, are scoped to that Pack. This means you cannot use, for example, a Knowledge Object defined inside a Pack outside it.

Upgrade a Pack from the Dispensary

When a new version of a Pack is published to the Dispensary, you can upgrade your local Pack to get all the latest improvements.

If you didn't introduce local changes to the Pack, you can upgrade the Pack directly from the UI. On the **Packs** page, open the Options ••• menu for your Pack, and then select **Upgrade**.



Upgrading an existing Pack

If you've modified the Pack, Cribl Stream will not overwrite it during upgrading, to prevent deletion of your locally created resources. To reconcile your own changes with improvements coming in from a new Pack version:

1. Import the updated Pack as a separate Pack with a new name.
   Give it a name that includes the version number (for example, `cribl-syslog-input-120`).
2. Do a side-by-side comparison of the previous and new versions of the Pack.
   Copy configurations from your modified Pack into the upgraded Pack.
   Review all comments in the upgraded Pack to understand changes introduced in the new version.

1.5.2. Use Packs To Get Started With REST Collection

Cribl Stream adds new Sources and Collectors regularly, but for data sources that do not yet have a dedicated Source, Cribl Stream provides the REST Collector. The REST Collector can collect data from a wide variety of upstream services, including Okta.

Because the REST Collector is flexible and versatile, integrating a new API often requires manual effort to configure, validate, and troubleshoot the unique REST API details. To eliminate this manual effort, the Cribl Dispensary provides pre-configured REST Packs for common upstream services to help speed up the configuration process. These Packs are officially supported, validated, and version-controlled in the Cribl Packs GitHub repository.

The REST Packs provide optimized, custom-made configuration settings to help you build a REST Collector for specific upstream services. The Packs:

- Ensure you can collect all the relevant log data from that data source, not just basic events.
- Provide the initial configuration to handle common data operations such as event breaking and field extraction, ensuring immediate data usability.
- Provide predefined pagination and authentication configurations that are specific that service.
- In some cases, help normalize data structures for specific Destinations.

This guide explains the end-to-end workflow for importing and using the REST Packs using the Okta REST Collector IO Pack as an example.

About the Okta REST Collector IO Pack

The Okta REST Collector IO Pack provides a pre-configured REST Collector and the necessary logic to handle JSON data collected from the Okta System Log API endpoint. It is built as a complete Source and Destination solution (which is why it has the `IO` prefix). This means that data collection and delivery happen entirely within the context of the Pack, eliminating the need to connect it to globally defined Sources and Destinations.

This Pack provides configurations to parse the JSON and normalize the timestamp from the proper field within each event. It also includes configurations for two types of data reduction:

- **Drop, sample, or suppress events based on `eventType`:** You can target individual events by `eventType` using the included lookup file Knowledge object.
- **Remove nested `null` value fields:** The included Pipeline contains a Function to automatically identify and remove fields where the value is explicitly `null`. This significantly reduces log volume and minimizes processing overhead for downstream Destinations.

The Pack includes pre-built Pipeline configurations for three common normalization formats:

- Normalized JSON
- OCSF (for downstream consumption by Amazon Security Lake)
- Splunk

When you install this Pack, it includes:

Element	Description	Location in the Pack
Okta API REST Collector	Pre-configured to work with Okta System Log API endpoint	**Sources** tab
Okta Pipeline	Contains three example Functions for event breaking, timestamp normalization, and data reduction	**Pipeline** tab
Route filter for Okta events	Filters any data events to the Okta Pipeline	**Routes** tab
Okta lookup file	A lookup file that can be set to drop, sample, or suppress events by `eventType`	**Knowledge** > **Lookups**

See the Okta REST Collector IO README for more information.

How to Use the Okta REST Collector IO Pack

This workflow involves these phases:

1. Install the Okta Pack from the Cribl Dispensary.
2. Customize the Pack configurations for your environment.
3. Customize the Pack variables.
4. Connect the Pack to a Destination.

Before you start, ensure you have the necessary credentials and details for your Okta deployment.

Install the Okta Pack From the Cribl Dispensary

You can install the Okta Pack from inside the target Worker Group in Cribl Stream:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Processing**, then **Packs.** Select **Add Pack**, then **Add from Dispensary**.

3. Search for `Okta Rest Collector IO` and select it from the list to show the Pack details. Select **Add Pack**.

A confirmation message appears in the UI when the Pack finishes installing. When you close this drawer, you will see the Packs list page. Confirm that the Okta REST Collector IO Pack now appears in this list.

Customize the Pack Configurations for Your Environment

In this phase, you set the values that are specific to your Okta deployment, including your Okta domain name and API token:

1. Select the **Okta Rest Collector IO** from the Packs list page. This opens the **Routes** tab inside of the Pack context.
2. Within the Pack context, select the **Sources** tab. Under **Collectors**, select **REST**. On the list of configured REST Collectors, select **in_okta_api** (Okta-API) to open its configuration window.

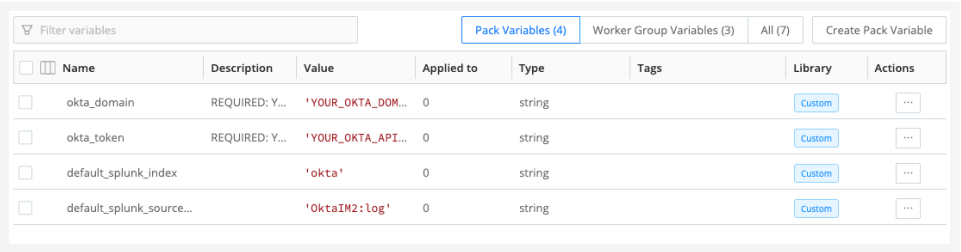
If you see tiles in the **Sources** tab instead of the list view, select **REST** to see all REST Collectors.

3. In the **General** tab, replace these placeholder values with the correct values for your Okta account:
    - **Collect URL:** Add the URL for your Okta domain. See the Okta documentation about how to find your domain for more information.
    - **Collect Headers Value:** Add your Okta API bearer token, also known as your SSWS key. See the Okta documentation about how to create an API token for more information.
4. Save your changes, then commit and deploy the changes to your Worker Group.

Customize the Pack Variables

In this phase, you set the values for the Pack variables that are specific to this Worker Group environment:

1. From within the Pack context, select **Knowledge** from the top bar. Then, select **Variables**.
2. Select the **Pack Variables** tab to see which variables come bundled with this Pack.



Available variables for the Okta Rest Collector IO Pack

3. To set their values, select each variable to open their configuration window. In the **Value** field, assign the appropriate setting for your current environment.

To validate your custom changes, try collecting some sample data to ensure the Collector is working as expected. See Data Preview for more information.

Connect the Pack to a Destination

When you import the Okta REST Collector IO Pack, it sets the Destination as the default Destination. To change this default:

1. Optionally, you may need to first add a new Destination inside the Pack context and then update the **Route** table.
2. Within the Pack context, select the **Routes** tab.
3. In the **cribl_okta_rest_api** filter, select a configured Destination from the **Destination** menu.

Verify that data is flowing from the Okta API to your target Destination. Also consider exploring the Okta Pipeline to customize the Functions based on your data needs.

1.6. Process Data With AI Assistance

Cribl Copilot Editor is an interactive, AI-powered assistant that helps you build data Pipelines in Cribl Stream using plain-language prompts. It is useful for anyone who wants help building Pipelines in Cribl Stream, especially if you're just getting started or working through a complex use case.

You might find Copilot Editor especially helpful if:

- **You are new to Cribl Stream and want a guided way to learn how it works.** Copilot Editor acts as an interactive learning tool. Copilot Editor is especially helpful when you're not sure how to start building a Pipeline, or when you're stuck writing complex Regex or JavaScript expressions. It can generate example Pipelines based on your goals, suggest alternative approaches, and help you explore solutions you might not have considered. Even if you don't use the generated Pipeline, it's a great way to learn how different Functions work together and how to build your own Pipelines.
- **You've inherited a Cribl deployment and need to make changes quickly.** If you're a system administrator or operator who needs to stand up a deployment fast or maintain an existing one, Copilot Editor can help you make quick progress without needing to dive deep into configuration syntax right away.
- **You want to experiment with AI to speed up Pipeline building.** Whether you're curious about AI or looking for a faster way to translate your intent into working logic, Copilot Editor lets you test ideas and see results with minimal setup.

You're always in control of the final result and can review, refine, or copy its suggestions into the Pipelines you build manually.

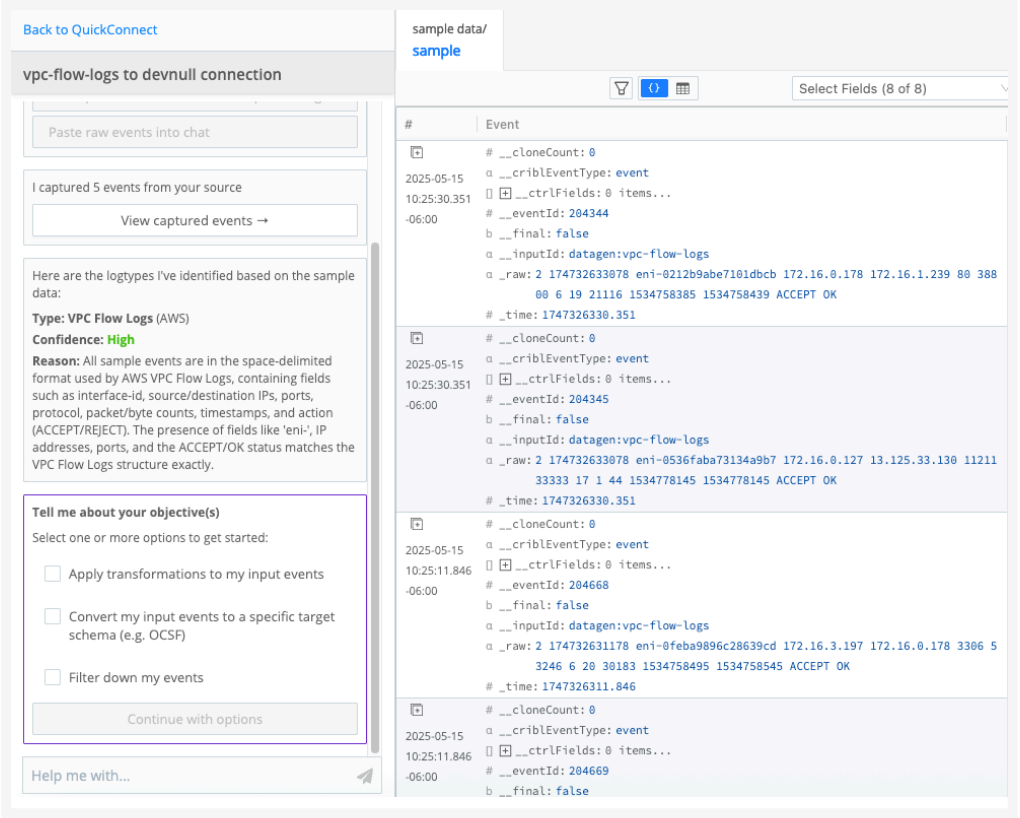
1.6.1. Create Pipelines With Cribl Copilot Editor

Cribl Copilot Editor is an interactive, AI-powered assistant that helps you build data Pipelines in Cribl Stream using plain-language prompts. Instead of building a Pipeline manually, you can describe what you want to do with your data and Copilot Editor will generate a working Pipeline for you.

Copilot Editor gives you an easy way to explore what Cribl Stream can do, especially if you're new to the platform or want help tackling more complex use cases. You stay in control: Copilot Editor's suggestions are editable, and you can review, modify, or view outputs from the suggestions before using them in production.

Here are a few common ways you can use Copilot Editor:

- **Structure raw data to match a standard schema or convert from one schema to another:** The data for many Destinations follow specific schemas. If events are improperly formatted, some Destinations may drop data or store them in ways that make the data difficult to find and use. With Cribl Stream, you can convert your raw data into normalized schemas, using Copilot to do the heavy lifting around configuring Functions. Copilot Editor is optimized for transforming data into the OCSF, UDM, and ECS schemas. You can also upload your own schema objects to convert to other schemas. See Convert Data into a Custom Schema for information about using custom schemas or schemas that aren't yet officially supported.
- **Apply common Functions to transform or enrich your data:** Copilot Editor can apply Cribl Stream's most frequently used Functions based on your intent. You don't need to know Regex or JavaScript syntax or which Functions to use up front. Just describe your goals and Copilot Editor builds the appropriate logic for you.
- **Filter and route data dynamically:** Use plain-language prompts to define filtering or routing logic without writing Regex or JavaScript expressions manually. Copilot Editor builds the first draft of the filtering and control logic. You can use its suggestions as-is or fine-tune as needed.



Copilot Editor

How to Use Copilot Editor

This section explains the overall workflow for using Copilot Editor to build Pipelines.

Before You Begin

Before you can start using Copilot Editor, you must first enable Cribl Copilot for your Organization:

1. From **Settings**, select the **Global** tab, and then select **AI Settings**.
2. Read the privacy policy and accept the terms and conditions.

See Enable or Disable Cribl Copilot for information about how to enable or disable Cribl Copilot programmatically.

Launch Copilot Editor

You can launch Copilot Editor in three ways:

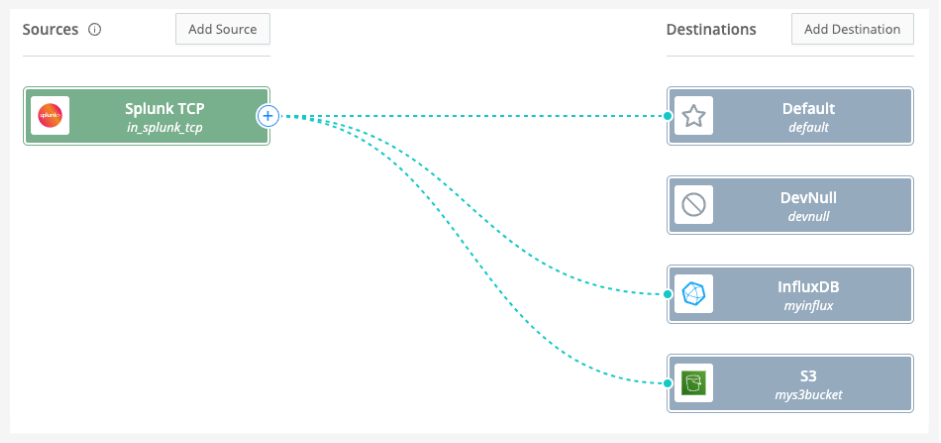
- Through QuickConnect.
- Through the Pipelines page.
- Inside a Pipeline through the **Add Function** menu.

Select a tab for more detailed instructions about each method for launching Copilot Editor:

QuickConnect | Pipeline page | Add Function menu

To launch Copilot Editor using QuickConnect, first set up your QuickConnect environment:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Select the **QuickConnect** tile.
3. Select **Add Source** to add at least one Source and **Add Destination** to add at least one Destination. Configure them as needed. Any previously configured Sources and Destinations automatically appear in QuickConnect.
4. Connect a Source to a Destination, but do not create a Pipeline yet.



Connect a Source to a Destination

5. Select a connection to open the **Connection Configuration** modal. Select **Build with Copilot Editor** to launch the interactive Copilot Editor modal.

You can also select **Build with Copilot Editor** at the top of the **QuickConnect** page to launch.

Build a Pipeline with Copilot Editor

The interactive flow of Copilot Editor follows a few general phases:

1. **Capture sample data:** When you first launch Copilot Editor, you see a chat window that prompts you to capture sample data. It asks you to choose a method for capturing sample data, such as collecting a live sample or by uploading sample data for that Source.

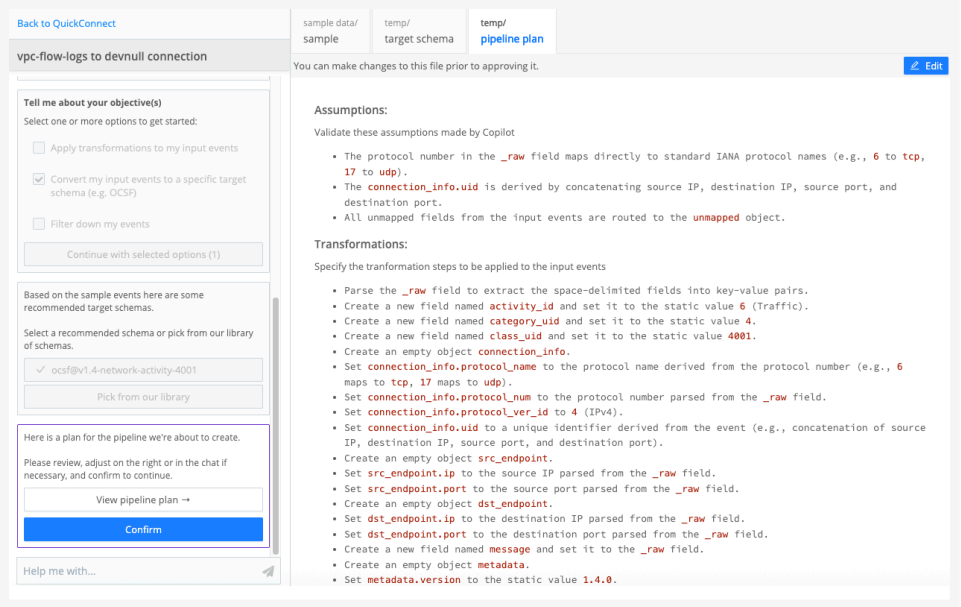
Why does it need sample data?

Copilot Editor uses this data to help you build, preview, test, and validate any data transformations you ask it to create in your Pipeline. It helps you compare the inbound and

outbound data to ensure that any Pipeline and Function configurations created with Copilot Editor produce the expected output, giving you confidence it is processing your data as intended.

2. **Choose an objective:** After analyzing your sample data, Copilot Editor displays your data in the preview window, similar to how data is presented in Data Preview in a standard Pipeline. The chat window will summarize the data and present you with a set of options for transforming your data. Select any objectives you would like to apply to your data. Copilot Editor will prompt you if it needs additional information about your objectives.
3. **View and edit the Pipeline plan:** Based on your selection, Copilot Editor generates a suggested Pipeline plan. It outlines:
    - Assumptions it made when generating the plan.
    - Transformations that it recommends based on your objectives.
    - Fields to keep and drop.

If needed, select **Edit** to further customize the proposed plan before confirming. For example, you could rename any fields as needed.



Copilot Editor Pipeline Plan

4. **Confirm the Pipeline plan and build the Pipeline:** After you confirm the plan, Copilot Editor builds a suggested Pipeline and displays a preview. Expand the Functions to explore the settings it applied. You can continue to edit the Pipeline at this stage as needed. You can also use natural language prompts in the chat window to request changes.
5. **Apply the Pipeline:** After you apply the Pipeline, select **Back to QuickConnect** at the top of the chat window to return to QuickConnect. Select **Save Connection** to add the Pipeline to your deployment.

Take note of the Pipeline ID displayed in the chat window so that you can find this Pipeline later.

6. **Confirm the Pipeline is present:** Confirm that the Pipeline ID you built with Copilot Editor appears in the Pipeline page or in QuickConnect. To find it in QuickConnect, select the connection that you used to launch Copilot Editor or its Pipeline icon, then select **Pipeline** to view the list of Pipelines for this connection.

Edit or Update a Pipeline With Copilot Editor

Any time you run a Copilot Editor, it eventually asks you to choose an objective for that session. You can select **Edit an Existing Pipeline** as the objective. It then presents you with a list of Pipelines to select and edit.

With that in mind, you can start a Pipeline editing session with Copilot Editor from a few different entry points:

- Start a new Copilot Editor session, then select the **Edit an Existing Pipeline** as the objective.
- Open an existing Pipeline, select **Add Function** , then **Edit with Copilot Editor**. Select the **Edit an Existing Pipeline** as the objective.

You can edit any existing Pipeline with Copilot Editor, not just Pipelines that were previously generated with Copilot Editor.

Guidelines and Better Practices

When working with Copilot Editor, keep these guidelines in mind:

- **Always review and validate Pipeline suggestions:** As with any AI tool, you should always check and confirm that the suggested changes match your expectations. Trust, but verify.
- **Copilot Editor may take different paths to meet an objective:** AI tools do not always solve problems in the exact same way every time, which sometimes takes some getting used to. This behavior is by design, but just be aware that the results may vary.

Use effective prompts to ensure the best results from the Copilot Editor. Some general guidelines:

- **Be specific:** To receive high-quality results, be clear and explicit with your requests. Provide exact names of fields that you want to reference or create.
- **Provide examples:** For instance, if you want to mask a specific piece of text, describe the target text in detail and explain what it should be masked to (for example, `Mask all IPv4 IP addresses to X.X.X.X`).
- **Ask questions:** Use the chat window to answer general questions about Pipelines. For example, you could ask the chatbot, `What Pipeline Functions can I use to redact sensitive information like IP addresses?`

Feature Support

Cribl does not make any warranty regarding the availability, uptime, performance, or accuracy of Cribl Copilot, including PII Detection and Prevention. Please read the Supplemental Terms for Cribl Copilot for more information.

Requirements

Cribl Copilot Editor always requires internet access.

Supported Deployment Types

Cribl Copilot Editor is available:

- For Cribl.Cloud or hybrid deployments.
- For Cribl Stream 4.12.0 and newer.

Supported Schemas

Cribl Copilot Editor specializes in transforming data into these schemas:

Schema	Supported Version(s)
OCSF Open Cybersecurity Schema Framework	• 1.6.0 • 1.5.0 • 1.4.0
UDM Unified Data Model	• 6.0
ECS Elastic Common Schema	• 9.0.0 (base version only)

Supported Functions and Integrations

Copilot Editor can build Pipelines using these Functions:

- Aggregations
- Code
- Comment
- Drop

- Eval
- Flatten
- Lookup
- Mask
- Numerify
- Regex Extract
- Regex Filter
- Rename
- Serialize

1.6.2. Convert Data Into A Custom Schema

Copilot Editor is an interactive AI assistant that helps you build Pipelines more easily in Cribl Stream. It specializes in designing Pipelines that transform your data to match specific schemas, whether they are industry-standard or custom-defined.

With Copilot Editor, you can:

- Structure raw data into a standard schema, such as OCSF, UDM, and ECS.
- Convert data from one standard schema to another.
- Transform data into a custom schema that you define.

Copilot Editor supports many schemas out of the box. See Supported Schemas for a complete list.

You can also define your own custom schema or adapt a standard schema for your needs. Copilot Editor can use your custom schema definition to help you generate a Pipeline that transforms your data to your specifications.

See Create Pipelines With Cribl Copilot Editor for a broader introduction, including how to launch it and build your first Pipeline.

Overview of the Custom Schema Workflow

The workflow for building a custom schema has five phases:

1. Get an example of the target schema: Start by identifying a sample event that represents your desired output format. If you already have a fully defined target schema, you can skip the first two steps and upload it directly to Cribl Stream.
2. Convert the example into JSON: Follow best practices to convert your example into a data schema using valid JSON.
3. Upload the schema: Add the JSON object to your Knowledge library so that Copilot Editor can reference that file.
4. Build a Pipeline to transform your data schema: Use Copilot Editor to create a Pipeline that can transform your data into the target schema.
5. Validate the schema output: Validate the data outputs to ensure you get the intended results. Then, make adjustments as needed.

Get an Example of the Target Schema

To build a custom data schema, start by identifying what you want your output data to look like. This step is optional if you already have a complete target schema ready to upload to Cribl Stream.

Find or create a sample event that closely matches your desired output. Choose an example that represents the type of event you want to map and reflects the ideal structure for your schema.

This guide uses the following sample event as a reference:

```
{
    "timestamp": "2025-07-08T14:30:00Z",
    "level": "INFO",
    "message": "User creation successful",
    "user": {
      "id": 12345,
      "username": "johndoe"
    },
    "session": {
      "session_id": "abcde12345",
      "device": "Chrome on Mac OS"
    }
  }
```

Copy

Convert the Example into JSON

Next, convert your sample event into a data schema using valid JSON object format. You'll upload this schema to Cribl Stream in a later step, allowing Copilot Editor to reference it when building your Pipeline.

Copilot Editor requires all schemas to be structured as valid JSON objects so it can parse and validate them correctly.

You can ensure that Copilot Editor can easily consume and successfully process your data schema by the best practices outlined in the following sections. The following example shows a well-formed JSON schema that follows best practices and serves as a reference in these sections:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "my-custom-schema-user-creation",
  "framework": "custom",
  "description": "A custom schema used to map user creation events.",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "timestamp": {
      "type": "string",
      "description": "The timestamp of the event in RFC 3339 format."
    },
    "level": {
      "type": "string",
      "enum": ["INFO", "WARNING", "ERROR", "DEBUG", "TRACE", "CRITICAL", "ALE
      "description": "A description of the type of event."
    },
    "message": {
      "type": "string",
      "description": "A description of the event."
    },
    "user": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer",
          "description": "The ID of the user created."
        },
        "username": {
          "type": "string",
          "description": "The username of the user created."
        }
      },
      "required": ["id"]
    },
    "session": {
      "type": "object",
      "properties": {
        "session_id": {
          "type": "string",
          "description": "The ID of the session."
        },
        "device": {
          "type": "string",
          "description": "The device of the session."
        }
      },
      "required": ["session_id"]
    },
    "additional": {
      "type": "object",
      "properties": {},
      "additionalProperties": true,
      "description": "Important log data that cannot be adequately represente
```

```
55          }
56        },
57        "required": ["timestamp", "level", "user"]
58      }
```

Copy

Define the Field Names, Types, and Descriptions

Copilot Editor can map your log data more accurately if you provide meaningful context when defining each field in your schema.

For every field in your JSON schema, always include:

- The field name name of the field.
- The field's data type (such as a string, integer, or boolean).
- A brief description of the field's purpose or contents.

Here's an example illustrating these best practices, taken from the reference schema (lines 18-21):

```
1     "message": {
2       "type": "string",
3       "description": "A description of the event."
4     },
```

Copy

In this example:

- Line 1 indicates the field name: `message`.
- Line 2 defines the data type: `string`.
- Line 3 provides a description: `A description of the event.`

List Allowed Values for Restricted Fields

If a field should only accept a specific set of values, use an enum to define the allowed options. This helps Copilot Editor understand the valid inputs for that field and ensures consistent schema mapping.

For example, see this section of the reference JSON schema (lines 13-17):

```
1     "level": {
2       "type": "string",
3       "enum": ["INFO", "WARNING", "ERROR", "DEBUG", "TRACE", "CRITICAL", "ALERT",
4       "description": "A description of the type of event."
5     },
```

Copy

In this example, line 3 defines the allowed values for the level field using an `enum` array.

Mark Required Fields

Clearly define which fields are required so Copilot Editor knows which values it must always include in the generated mapping. Use a required array inside each object to mark fields that are mandatory within that scope, and add a top-level required array to indicate which fields are required across the entire schema.

For example, see this portion from of the reference schema (lines 36-58):

```
 1         "session": {
 2           "type": "object",
 3           "properties": {
 4             "session_id": {
 5               "type": "string",
 6               "description": "The ID of the session."
 7             },
 8             "device": {
 9               "type": "string",
10               "description": "The device of the session."
11             }
12           },
13           "required": ["session_id"]
14         },
15         "additional": {
16           "type": "object",
17           "properties": {},
18           "additionalProperties": true,
19           "description": "Important log data that cannot be adequately represente
20         }
21       },
22       "required": ["timestamp", "level", "user"]
23     }
```

Copy

In this example:

- Line 13 includes a required array that marks `session_id` as mandatory: `"required": ["session_id"]`.
- Line 22 is the top-level array of required fields, indicating which fields are mandatory across the entire object: `"required": ["timestamp", "level", "user"]`.

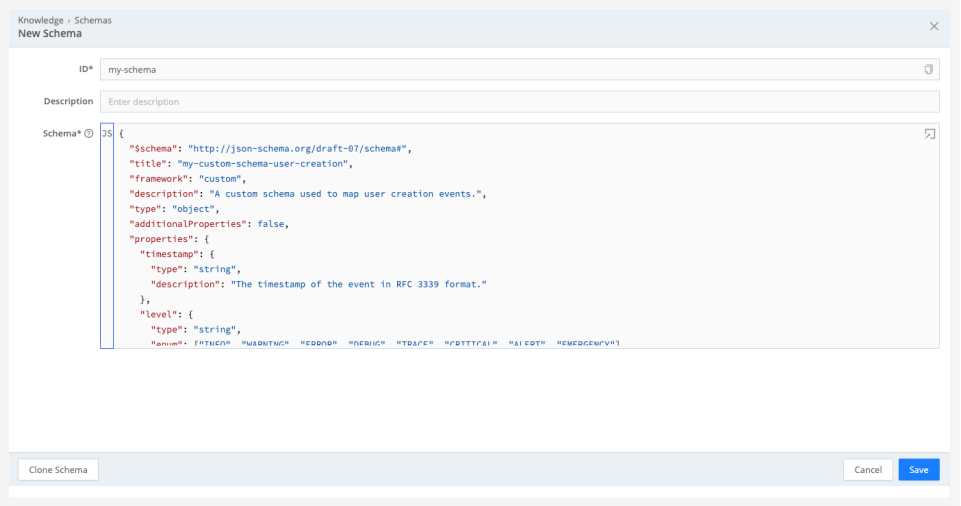
Add an Unmapped Fields Section

Consider adding an additional or unmapped field as a catch-all for any data that doesn't fit cleanly into your defined schema. This lets you retain important information from the original event without forcing it into a specific predefined field.

Upload the Schema

Once you have your schema defined in a JSON object, you can upload it to the Knowledge library so that Copilot Editor can reference that file:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a **Worker Group**.
2. Navigate to **Processing**, then **Knowledge**. Select **Schemas** from the side menu.
3. Select **Add Schema**. In the **New Schema** modal, configure the following settings:
    - **ID:** Enter a unique name for the schema.
    - **Description:** Optionally, add a short description of the schema's contents or purpose.
    - **Schema:** Use the built-in editor to paste the JSON file into the window.



Add a schema to the Knowledge library

4. Select **Save**. Cribl Stream checks your input to ensure it is valid JSON and flags any errors you need to correct before attempting to save again. Select the error icon for more information about the errors that need to be fixed.

5. Confirm that your schema file appears in the list of schemas. Then,commit and deploy the changes to your Worker Groups.

Consider using Cribl Copilot to help you write commit messages. It analyzes the diff and recommends a message that describes the update.

Build a Pipeline to Transform Your Data Schema

Once you have uploaded your schema to the Knowledge library, you can then reference it when building a Pipeline with Copilot Editor:

1. Check that you have enabled Copilot Editor and then launch Copilot Editor.
2. Follow the chat prompts to capture or import sample data. The chat window may also prompt you to select a data source.
3. When the chat window prompts you to choose an objective, select **Convert my input events to a specific**

2. Set Up Cribl Stream

Cribl Stream offers different types of deployments that you can choose based on your needs:

- Cribl.Cloud, including a hybrid option
- On-prem deployment, including:
    - Single-instance deployment
    - Distributed deployment
    - Orchestrated deployment
    - Docker deployment

To prepare for a deployment and gather decision-making information, consult Deployment Planning.

2.1. Reference Architectures

Reference architectures present possible Cribl Stream deployment scenarios for various use cases. Please see the assumptions and disclaimers accompanying each topic:

- Cribl Reference Architecture, Full-Suite
- Syslog to Cribl Stream Reference Architecture

2.2. Cribl.Cloud Vs. Self-Hosted

A Cribl.Cloud deployment differs in several ways from a customer-managed (on-prem) deployment of Cribl Suite software on your own infrastructure. Keep in mind these differences as you choose how to launch Cribl Stream. Also consider these differences as you navigate the product's UI, in-app help (including tooltips), and documentation.

Cloud-Only Features

Certain Cribl products and features are available **only** on Cribl.Cloud:

- Cribl Search, an application for searching, exploring, and analyzing machine data in place and at API endpoints.
- Cribl Lake, a data lake solution for long-term, full-fidelity data storage.
- Workspaces, an option for isolating parallel Cribl Leaders with separate access controls and other configurations.

For details about how all Cribl products interoperate to manage data, see Cribl Reference Architecture, Full-Suite.

Simplified Administration

Cribl.Cloud has been designed with options to accommodate everyone - from first-time evaluators to Enterprise customers who manage a worldwide network of private-cloud, public-cloud, and/or data-center deployments.

Cribl.Cloud's Free offering is designed to help you launch Cribl Stream - and to start processing data - as quickly and easily as possible. Upgrading to a paid Standard or Enterprise plan provides expanded deployment and configuration options.

For a comparison of features in Free, Standard, and Enterprise Cribl.Cloud plans, see Pricing.

Simplified Distributed Architecture

Cribl.Cloud is preconfigured as a Distributed deployment. A Free or Standard plan allows only a single Worker Group, and some Distributed Settings cannot be configured.

With an Enterprise plan, Cribl always provides at least two Workers, and will scale up further Workers as needed to meet your peak load. With an Enterprise plan, you also have the option to configure additional hybrid Workers and Worker Groups.

Git Preconfigured

Without an Enterprise plan, the **Settings** > **Global** > **System** > **Git Settings** section is omitted. However, a local `git` client is preconfigured in your Cribl.Cloud Organization. On the top nav, use the **Version Control** button (with a branched symbol) to commit/push changes to `git`. Select **Deploy** to deploy your committed changes. Cribl.Cloud does not support Git remote repos.

Automatic Restarts and Upgrades

Without an Enterprise plan, the **Settings** > **Controls** link are omitted. Cribl handles Leader and Worker Group restarts automatically on your behalf.

Simplified Access Management and Security

In Cribl.Cloud, you can manage access control for your Organization by selecting **Organization** in the sidebar and then selecting **Members & Teams**. The options on this tab will vary depending on your plan.

If you have a Cribl.Cloud Enterprise plan, you can use the Key Management Service (KMS), which maintains the keys Cribl Stream uses to encrypt secrets on Worker Groups and Workers. Go to **Settings** > **Security** > **KMS** to configure KMS.

If you add an Enterprise Plan, cloud and hybrid Leaders support Local and Google SSO authentication, along with OpenID Connect (OIDC) and SAML federated authentication. Cribl.Cloud does not currently support LDAP.

Permission- and Role-based access control (RBAC) is simplified in Cribl.Cloud. For details, see Permissions.

Security Features Comparison: Self-Hosted vs. Cribl.Cloud

The following table outlines the security responsibilities for self-hosted and Cribl.Cloud deployments, highlighting key differences in deployment security, data protection, threat detection, and access control.

Feature	Self-Hosted	Cribl.Cloud
Deployment Security	Customer responsible for securing the deployment environment (network isolation, physical access control, user access management).	Cribl manages the security of the cloud infrastructure.
Configuration Security	Customer responsible for securing configuration files and tokens.	Cribl manages the security of configuration files and tokens.
Git Configuration Security	Customer responsible for securing the Git repository.	Cribl manages the security of the Git repository.

Feature	Self-Hosted	Cribl.Cloud
Data at Rest	Customer responsible for encryption and key management.	Encrypted at rest by using industry standard encryption (e.g., AES-256) for storage services.
Data in Motion	Customer configures encryption (TLS) for data in transit.	Preconfigured TLS for some Sources. Can be further configured.
Threat Detection & Response	Customer responsibility. Use Cribl Stream security features (limited) and external tools.	Cribl bolsters its security posture with internal security teams and an external MSSP, ensuring comprehensive protection for its production and corporate environments through threat detection, workload scanning, and vulnerability management.
Patch Management	Customer responsible for applying security patches to Cribl Stream and underlying infrastructure.	Cribl manages patching of Cribl.Cloud (Cribl-managed) infrastructure.
Access Control	Customer configures user Roles and Permissions (RBAC).	Simplified RBAC with Enterprise plans offering advanced options (SSO, KMS).
Authentication	Customer configures authentication methods.	Local and SAML/OIDC IDP with Enterprise plans (additional options).
Key Management	Customer manages encryption keys.	Enterprise plans offer Key Management Service (KMS) for key storage.
Compliance	Customer responsible for adhering to compliance standards.	SOC 2 Type II compliant and GDPR-compliant (Cribl.Cloud).

Transparent Licensing

The Cribl.Cloud sidebar does not display a **Settings** > **Global** > **Licensing** link, nor does the **Monitoring** > **System** submenu include **Licensing**. Your plan is managed by your Cribl.Cloud Organization, and you can check credits and usage history in the FinOps Center.

Other Simplified Settings

These features are available only on customer-managed hybrid Cribl Stream Workers:

- Script Collector
- Staging directory support in File-based Destinations
- Tee Function

- File System Collector
- Filesystem Destination

These features are available only in on-prem deployments:

- **Settings** > **Global** > **Scripts** (if enabled - Cribl.Cloud does not support configuring or running shell scripts).

These features are unavailable on Cribl-managed Stream Workers in Cribl.Cloud:

- System State Source
- AppScope Source's **Filter Settings**

Configuring persistent queues in a Cribl.Cloud deployment requires an Enterprise plan. On customer-managed (hybrid or on-prem) Worker Groups, you can freely define the **Queue size limit**, based on the disk space you provision.

However, on Cribl-managed Worker Groups, each Source or Destination's queue is allocated a maximum of 1 GB disk space per Worker Process. (Given this automatic configuration, Cribl-managed Sources and Destinations expose only limited PQ controls.)

Available Ports and TLS Configurations

To get data into Cribl.Cloud, your Cribl.Cloud Organization provides several Sources and ports already enabled for you, plus 11 additional TCP ports (`20000-20010`) that you can use to add and configure more Cribl Stream Sources.

TLS Details

TLS encryption is pre-enabled for you on several Sources, also indicated on the your Workspace's **Data Sources** tab.

Cribl HTTP and Cribl TCP Sources/Destinations

Use the Cribl HTTP Destination and Source, and/or the Cribl TCP Destination and Source, to relay data between Workers that are either connected to the same Leader or different Leaders. For more information about transferring data across Organizations, go to Transfer Data Between Workspaces or Environments.

This traffic does not count against your ingestion quota, so this routing prevents double-billing. (For related details, see Exemptions from License Quotas.)

Simplified Source, Collector, and Destination Configuration

Several commonly used Sources are preconfigured for you within Cribl.Cloud's UI, and are ready to use.

The Exec Source is unavailable on Cribl-managed Workers in Cribl.Cloud, but is available on hybrid Workers.

The Cribl Internal Source's CriblLogs option, when used on Cribl-managed Worker Groups in Cribl.Cloud, contains only logs related to Sources and Destinations.

2.3. Learn About Connected Environments

If you have one or more on-prem Cribl Stream environments and Cribl.Cloud Organization(s), you can configure **Connected Environments**.

Connected Environments enables you to:

- **Take advantage of Cribl.Cloud.** You can use Connected Environments to connect and on-prem Leader to Cribl.Cloud in order to use Cribl.Cloud infrastructure management capabilities, the credit-based billing model, as well as Cribl Lake and Cribl Search solutions. For example, you have on-prem environments but you want to use the credit billing model, and you want to send Cribl metric data to Cribl Lake/Search. From there, you can build dashboards to monitor Cribl throughput and performance.
- **Send data from an on-prem deployment to a Cribl.Cloud deployment.** If you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from on-prem Worker Nodes to Cribl.Cloud Worker Nodes and transfer data without incurring any billing costs. See Send Data from On-Prem to Cribl.Cloud for implementation details.

For full access to Connected Environments, upgrade Cribl Stream to 4.8.2 or newer.

Accomplish this by creating a connection between the on-prem Leader and the Cribl.Cloud Leader. Then, use the **Connections** page in your Cribl.Cloud account to manage credit consumption for all of your on-prem environments from one Cribl.Cloud interface. This feature is called Universal Subscription.

This section provides instructions for connecting on-prem Leaders to Cribl.Cloud.

Before You Begin Connecting On-Prem Leaders

There are a few tasks you must complete before you can start connecting your on-prem Leader to Cribl.Cloud:

1. Create a Cribl.Cloud account.
   To set up connected environments, you must have an Enterprise Cribl.Cloud account.
2. Decide how you want to connect to the Leader.
   There are two methods you can use to connect an on-prem Leader to a Cribl.Cloud Leader:
   - From the Cribl.Cloud account in **Connections** > **Connected Environments**.
   - From the on-prem Leader in **Distributed Settings** > **Cloud Connection**.
3. Purchase a Cribl.Cloud Enterprise Plan with credits.

Credits serve as the virtual transaction currency in Cribl's product suite. Each Cribl product has a predefined usage rate, and as you use the products, credits are deducted from your initial pool.

Configuration Considerations

Take these considerations into account for your on-prem Leader, allowlists, security groups, and Workspaces:

- You must be on Cribl Stream version 4.8.2 or newer.
- **Ports**: Make sure that port `4200` is open on the Leader, with TCP protocol. See Leader Ports for more information.
- **Proxy**: Connected Environments doesn't support proxied connections.
- **TLS**: The connection between the Cribl.Cloud Leader and the on-prem Leader must be encrypted using TLS. When you configure the `CRIBL_CLOUD_WORKSPACE_URL` environment variable, it will start with `tls://`, which enforces TLS.
- **Static IPs**: For IP address allowlist or security group rules, you can find the static IPs of your Cribl.Cloud Leader in **Workspace** > **Access** > **Leader NLB IPs**. You can also use the `nslookup` command to obtain static IPs.
- **Workspaces**: Workspaces are separate infrastructure. For allowlists and security groups, ensure that you add each IP address individually as each Workspace has a unique IP.

2.3.1. How To Connect On-Prem Leaders To Cribl.Cloud

To connect existing on-prem Cribl Stream Distributed deployments to Cribl.Cloud, select an on-prem Leader that is running Cribl Stream 4.8.2 or newer, then follow the steps.

If you have multiple environments, such as staging and production environments when using GitOps, you must connect each environment individually.

1. On the top bar, select **Products**, and then select **Cribl**.
2. In the sidebar, select **Workspace**, and then select **Connections**.
3. Select **Configure New Environment** and fill out the following fields:

Field	Description	Example
Environment name	A name for your environment. This name will display on the **Connected Environments** page. You can search for and filter environments by name.	Goat Farm East Production
Workspace hostname/ID	Hostname of your Cribl.Cloud Workspace. This field is read-only.	`example-goat-farm-prod.cribl.cloud`
Workspace port number	Port number that the Workspace Leader uses to communicate with the on-prem Leader.	`4200`
Workspace token	The auth token of the Cribl.Cloud Leader. This field is read-only and is automatically populated.	`a12bcDEFgHiJKLmnOpQRstuvw3x4z56`

Script type	The format of the script to connect to the on-prem Leader:
	◦ Environment variable
		Environment variable:
		`CRIBL_DEPLOYMENT="Goat Farm West Product`
		`CRIBL_CLOUD_WORKSPACE_URL="tls://a12bcDE`
		CLI:

Field	Description	Example

	◦ CLI	`./cribl cloud-workspace`
		`-d “Goat Farm West Production”`
		`-H example-goat-farm-prod.cribl.cloud`
		`-p 4200`
		`-u a12bcDEFgHiJKLmnOpQRstuvw3x4z56`

Use a Command-Line Script

If you’re using the CLI option, follow these steps, in order:

1. Select **Copy Script**.
2. Open a command-line window in the Leader environment that you want to connect to Cribl.Cloud.
3. Navigate to the `$CRIBL_HOME/bin` subdirectory.
4. Paste the script results onto the command line, and run the script.

Use an Environment Variable

The **Environment Variable** script populates with the details you enter when you’re configuring a new environment. For example:

```
CRIBL_DEPLOYMENT="On-prem Stream"
CRIBL_CLOUD_WORKSPACE_URL="tls://a12bcDEFGHIjKLMnopPqr123@cribl-cloud-account-0a1bcde
```

Copy

If you’re using the environment variable, copy its output, and set it when - or before - you run `cribl start`. How you set this variable depends on your deployment.

- For example, if your Cribl Stream deployment is running in a container, set the environment variable in the container via run call or configuration.
- If you’re running your own server, you can set environment variables for the `cribl` user, or you can declare the variable along with the `start` command. For details, see our CLI Reference.

High Availability Leader Configuration

Before you configure a connected environment with a High Availability (HA) Leader, ensure you are comfortable setting up and managing Leader HA. See [Leader High Availability/Failover](deploy-add-second-leader for more information.

When you set up the primary Leader to connect to Cribl.Cloud in an HA environment, set the `CRIBL_DEPLOYMENT` and `CRIBL_CLOUD_WORKSPACE_URL` environment variables. You can do this in the Cribl.Cloud UI or using CLI.

For example:

```
Environment=CRIBL_DEPLOYMENT=aws_leader_prod
Environment=CRIBL_CLOUD_WORKSPACE_URL=tls://large-goat-farm.cribl.cloud:4200
```

Copy

The failover Leader will inherit the configuration from the filesystem changes made by the primary Leader.

Enable the On-Prem Leader Connection to Cribl.Cloud

You can connect a Leader Node to an existing Enterprise Cribl.Cloud Organization.

1. In the sidebar, select **Settings**.
2. On the **Global** tab, select **Distributed Settings**.
3. Select **Cloud Connection**.
4. Toggle **Join Cribl.Cloud Workspace** on.
5. Fill in the required fields:

Field	Description
Connection URL	The **Cribl.Cloud URL** to connect to this Leader. 1. In your Cribl.Cloud Organization, locate the URL field value on the sidebar **Workspace** > **Access** tab. 2. Copy the URL and paste it into the **Connection URL** field, using the format: `main-random-goat-a12bcdef.cribl.cloud`.
Connection port	The port for the Cribl.Cloud Leader. Default is `4200`.
Connection auth token	The authentication token for the Cribl.Cloud Leader. You'll need to get this from the Cribl.Cloud environment: 1. Navigate to **Workspace** > **Connections** and select **Configure New Environment**. 2. Copy the value in the **Workspace token** field. 3. In the on-prem environment, paste the Workspace token from Cribl.Cloud.

6. Select **Save**.

7. In the Cribl Subscription Services Agreement modal, select **Agree** to accept the agreement.
8. In the confirmation modal, select **Yes** to save changes and restart.

Once you connect an on-prem Leader to a Cribl.Cloud Organization, the license type changes to a Cribl.Cloud billing account. This means the **Licensing** page will not be accessible. To return to using an on-prem license, toggle **Join Cribl.Cloud Workspace** off, select **Save**, and select **Yes** in the confirmation modal to save changes and restart.

2.3.2. Simplify Billing With Universal Subscription

This feature requires an Enterprise Plan.

If you have multiple instances of Cribl Stream on-prem, you're likely managing multiple licenses. Instead of letting license management run your life, you can take advantage of the Universal Subscription feature. Universal Subscription lets you manage an existing on-prem license with Cribl.Cloud credits.

To get started, connect your on-prem Leader to your Cribl.Cloud Leader to send billing data payloads. Once your environments are connected, you can use the FinOps Center to get a unified, all-product view of your Cribl Stream usage and spending.

This replaces an on-prem license, and gives you the flexibility to maintain your on-prem deployments without managing multiple licenses and tracking spend across environments.

An on-prem Cribl Stream environment consists of at least one Leader Node (the control plane) and one or more Worker Groups.

Connecting your on-prem Leader license to Cribl.Cloud billing allows you to:

- Monitor credit consumption for each of your on-prem Cribl Stream environments from one user interface.
- Create a billing- and usage-based connection between the Cribl Stream on-prem Leader and the Cribl.Cloud Leader, which receives metrics from the on-prem Leader. For information about the data payloads, visit Data Payloads for Connected Environments.

Configure Connected Environments

For the setup procedure, go to How to Connect On-Prem Leaders to Cribl.Cloud, then come back here to explore how to view credit and usage data for your on-prem Connected Environments.

Full access to Connected Environments requires you to upgrade Cribl Stream to 4.8.2 or newer.

Explore Credit Consumption for Connected Environments

Once you set up a Connected Environment, you can view the incoming usage data and credit consumption for your on-prem and Cloud deployments of Cribl Stream. You must be an Organization Owner to view this data.

For more information on Cribl.Cloud billing, go to the FinOps Center page.

View Usage Data

1. In the sidebar, select **Organization**, then **FinOps Center**.
2. Select **Usage Overview**.
3. Go to **Per Product Consumption** and expand either Cribl Edge or Cribl Stream.
4. Find **Connected < Product > GBs In**.

View Credit Consumption

1. In the sidebar, select **Organization**, then **FinOps Center**.
2. On the **FinOps Center** submenu, select **Invoices**.
3. Select a month to expand the cost items. Select **Stream** or **Edge**.
4. View **Connected < Product > GBs In** usage and cost.

Here, you will see each on-prem environment that is connected to this Cribl.Cloud Organization, and the credits each environment is consuming.

Limitations of the Universal Subscription Feature

Connected Environments has some usage limitations:

- The Connected Environments feature is available only on Enterprise Cribl.Cloud plans. For Enterprise plan details, see Pricing.
- Single-instance deployments of Cribl Stream aren't eligible for a Universal Subscription.
- The Distributed deployment on-prem Leader must be running Cribl Stream 4.8.2 or newer.

2.3.3. Data Payloads For Connected Environments

In a Connected Environment setup, on-premises Leaders connect to a Cribl.Cloud Leader in the same way that Worker Nodes connect to a Leader. The Cribl.Cloud Leader sends down a license, and the on-premises Leader sends up a heartbeat that includes telemetry, which encompasses metrics and other metadata.

Why Is the Telemetry Important?

The telemetry sent from an on-premises Leader to a Cribl.Cloud Leader is encapsulated in an event. This event serves as a heartbeat, confirming that the connected environment is active and functioning properly. It includes key metadata such as the system's status, timestamp, configuration details, and metrics. Cribl.Cloud uses this data to:

- Monitor connectivity between the on-premises Leader and the cloud.
- Track system health by evaluating fields like status, edgeNodes, and metrics.
- Ensure compliance with licensing by verifying the config.featuresRev field.
- Provide insights into data flow, such as input and output metrics.

How Often Is the Heartbeat Sent?

The heartbeat interval is determined by the **Task heartbeat period** field, which specifies the number of seconds between each event. In the provided example below, the heartbeat is sent every `10` seconds. This interval is configurable, meaning administrators can adjust it to optimize performance based on their needs. For details, see Task Manifest and Buffering Limits in Worker Group Settings.

What Happens if the Heartbeat Is Missed?

If a heartbeat event is not received within the expected interval, the following actions may occur:

- **Retries** - The on-premises Leader may attempt to resend the heartbeat after a short delay.
- **Alerting** - Cribl.Cloud can generate an alert if the heartbeat is missing for an extended period, indicating a potential issue.
- **Failure detection** - A prolonged absence of heartbeats may be interpreted as a disconnection or system failure, prompting further investigation or automated recovery steps.

On-Premises Leader to Cribl.Cloud Leader Payload

This section describes the structure of the event payload sent from an on-premises Leader to a Cribl.Cloud Leader in a Connected Environment.

Sample Event

Here is a sample event that Cribl.Cloud received from a connected environment.

```
{
  "__criblEventType": "event",
  "type": "info",
  "_time": 1737577024.567,
  "status": "healthy",
  "localTime": 1737577024,
  "config.hbPeriodSeconds": 10,
  "config.featuresRev": "YbK7TIGe4jp2rdujnFiJqq0OBV67r2FKGvrHr4C6I5g=",
  "edgeNodes": 50,
  "__socketAddr": 192.168.1.100:443,
  "__srcIpPort": 10.0.0.5:52314,
  "metrics": [
    [
      0,
      0,
      "total.in_bytes",
      233,
      1737578280038,
      1737578340038,
      [
        "ci",
        "tcp:in_tcp",
        "input",
        "tcp:in_tcp",
        "input_type",
        "tcp",
        "__worker_group",
        "default",
        "__dist_mode",
        "worker",
        "__worker_node",
        "46731aec-d220-4cb0-9971-0dd0a7907d54",
        "__product",
        "stream"
      ],
      null,
      "float32"
    ]
  ],
  "__raw": "{\"type\":\"info\",\"_time\":173757...}"
}
```

Copy

Event Payload Fields

The following table describes the important fields in the event payload.

Field	Usage	Example
__criblEventType	Type of event	event (always)
type	Type of event	info (always)
_time	Timestamp of the event in Unix time (seconds)	1737577024.567

2.3.4. Send Data From On-Prem To Cribl.Cloud

You can use Connected Environments to send data from an on-prem deployment to a Cribl.Cloud deployment. For example, if you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from a on-prem Worker Nodes to their counterpart Cribl.Cloud Worker Nodes.

You can transfer data from on-prem Worker Nodes to the Cribl.Cloud Worker Nodes without incurring any data transfer costs.

Overview of the Data Migration Workflow

The workflow for migrating data from on-prem Worker Nodes to Cribl.Cloud Worker Nodes has these phases:

1. Create a Connected Environment. This process generates the information you need to set up the data flow between the on-prem Worker Nodes and the Cribl.Cloud Worker Nodes.
2. On-Prem Worker Nodes: Configure HTTP or TCP Destination. Add an auth token on the on-prem Worker Nodes to configure the first half of the connection between the on-prem Destination and the Cribl.Cloud Source.
3. Cribl.Cloud Worker Nodes: Configure HTTP or TCP Source. Add a matching auth token on the Cribl.Cloud Worker Nodes to complete the connection.
4. Validate the Data Flow. Confirm that the data is properly flowing from the on-prem Worker Nodes to the Cribl.Cloud Worker Nodes.

Create a Connected Environment

To begin the process of migrating data from on-prem to Cribl.Cloud, you need to first set up a Connected Environment on the on-Prem Leader. This process generates the information you need to set up the data flow between on-prem Worker Nodes and the Cribl.Cloud Worker Nodes.

To set up a Connected Environment, follow the instructions on How to Connect On-Prem Leaders to Cribl.Cloud. These instructions also explain how to:

- Optionally set up environment variables at the time you create the Connected Environment.
- Use a command-line script to set up a Connected Environment if needed.

Ensure that you also follow the instructions to Enable the On-Prem Leader Connection to Cribl.Cloud.

On-Prem Worker Nodes: Configure HTTP or TCP Destination

On the on-prem deployment, select a Worker Node and configure a new or existing Cribl HTTP or Cribl TCP Destination. These Destinations enable internal data transfer between Cribl nodes to prevent data transfer billing costs.

To configure an HTTP or TCP Destination:

1. Follow the general configuration instructions on the Cribl HTTP or Cribl TCP Destination documentation.
2. In the **Auth Token** section of the configuration modal, select **Auth Token** to set up a shared secret between this Destination and the Cribl.Cloud Source:
    - **Token secret (text secret):** Select the name of the secret that you have created previously or create a new secret using the **Create** button. See Create and Manage Secrets for specific instructions about creating secrets.
    - **Enable token:** Toggle the token on to enable it.
    - **Description:** Optionally, enter a description summarizing the purpose of the secret.

If you don't see the **Auth Token** section of the Destination, it means you have not set up a Connected Environment for your on-prem Leader.

3. Commit and deploy the changes.

You can add multiple tokens if you need to rotate auth tokens.

Cribl.Cloud Worker Nodes: Configure HTTP or TCP Source

On the Cribl.Cloud deployment, select a Worker Node and configure a new or existing Cribl HTTP or Cribl TCP Source. These Sources enable internal data transfer between Cribl nodes to prevent data transfer billing costs.

To configure an HTTP or TCP Source, follow the same steps listed in the previous section about Configuring HTTP or TCP Destination for the on-prem Worker Nodes.

When you select or create the secret from the **Token secret (text secret)** menu, this secret must match the value of the secret used on the on-prem Cribl HTTP or Cribl TCP Destination.

Commit and deploy after making these configuration changes.

Validate the Data Flow

As a final step, confirm that the data is properly flowing from the on-prem Worker Nodes to the Cribl.Cloud Worker Nodes. A few methods for confirming data flow include:

- Capture sample data on your Cribl HTTP or TCP Source on your Cribl.Cloud Worker Nodes and verify it looks correct. See Add Sample Data for specific instructions.

- In the configuration modal for your Cribl HTTP or TCP Source on your Cribl.Cloud Worker Nodes, select the **Charts** tab to view data events coming in to verify you see activity. You can also compare this to the outgoing data on the Cribl HTTP or TCP Destination on your on-prem Worker Nodes.

2.4. Deployment Planning

There are at least **three** key factors that will determine the type of Cribl Stream deployment in your environment:

- Amount of Incoming Data: This is defined as the amount of data planned to be ingested per unit of time. E.g., how many MB/s or GB/day?
- Amount of Data Processing: This is defined as the amount of processing that will happen on incoming data. E.g., are there a lot of transformations, regex extractions, parsing functions, field obfuscations, etc.?
- Routing and/or Cloning: Is most data going to a single destination, or is it being cloned and routed to multiple places? This is important because destination-specific serialization tends to be relatively expensive.

These factors are covered in detail in Sizing and Scaling, and in our Architectural Considerations introduction to reference architectures.

Type of Deployment

- Use Cribl.Cloud to quickly launch a Cribl-managed deployment of the combined Cribl applications suite (Stream, Edge, and Search). With this option, Cribl assumes responsibility for provisioning and managing all infrastructure, on your behalf.
- Use Single-Instance/Basic Deployment when incoming data volume is low, and/or amount of processing is light.
- Use Distributed Deployment to accommodate increased load. (See Sizing and Scaling for detailed guidance. See Bootstrap Workers from Leader to streamline Workers' deployment via scripting.)

OS and System Requirements

Leader and Worker Nodes should have sufficient CPU, RAM, network, and storage capacity to handle your **specific** workload. It's very important to test this before deploying to production. For details, see OS and System Requirements.

For large on-prem deployments handling high traffic, review the recommended OS-level tuning detailed in OS Tuning for Large Deployments.

Network Requirement (DPI/IPS Exclusion)

Deep Packet Inspection (DPI) and Intrusion Prevention Systems (IPS) on network devices situated between your Cribl Worker Groups can cause high resource utilization and connection instability in all

> environments (both Cribl.Cloud and on-prem). Explicitly exclude all Cribl data streams from DPI/IPS inspection for stable operations and optimal performance.

Cluster Installation/Configuration Checklist

This section compiles basic checkpoints for successfully launching a distributed cluster.

1. Provision Hardware

- 1 Leader Node (see specs/requirements in OS and System Requirements above).
- 4 Worker Nodes (see specs/requirements in OS and System Requirements above).
- Acquire an evaluation (Sales Trial) License from the Cribl Sales Team.

2. Configure Leader Node

- Install `git` if not present (for example, `yum install git`).
- Open the necessary ports.
- Download, Install, and Launch Cribl.
- Enable Start at Boot.
- Configure as a Leader.
- Confirm Worker Processes Settings at `-2` (via **Settings** > **Global** > **System** > **Manage Processes**).
- Install License.

3. Configure Worker Nodes

- Enable GUI Access. Administrators will need to connect to the TCP:9000 port on each Node.
- Download, Install, and Launch Cribl.
- Enable Start at Boot.
- Configure as a Worker. (Give each Worker the (arbitrary) tag `POV`.)
- Confirm Worker Processes Settings at `-2` (via **Settings** > **Global** > **System** > **Manage Processes**).
- Install License.

4. Map Workers to Groups

- On the Leader Node, create a Worker Group.
    - Name the Worker Group (arbitrarily) `POV`.

- On the Leader Node, confirm that workers are connecting.
    - In the sidebar, select **Workers**.
- Map Workers to `dev` Worker Groups.
    - Use the Filter: `cribl.tags.includes('POV')`.

5. Other

If you will be using Cribl Stream's GeoIP enrichment feature, install the MaxMind database onto the Cribl Stream Leader and all Worker Nodes.

2.4.1. Sizing And Scaling

You can scale a Cribl Stream installation **up** within a single instance and/or scale **out** across multiple instances. Scaling allows for:

- Increased data volumes of any size.
- Increased processing complexity.
- Increased deployment availability.
- Increased number of destinations.

We offer specific examples below, and introduce reference architectures in Architectural Considerations.

Scale Up

You can scale a Cribl Stream deployment to utilize as many resources on hosts as you require. You control this differently on Cribl-managed versus customer-managed Worker Groups.

Optimize Cribl-Managed Worker Nodes in Cribl.Cloud

1. Navigate to **Group Settings** > **Group Configuration** in the desired Worker Group.
2. Adjust the Sizing Calculator to determine the appropriate size for your Worker Group.

Optimize a Single-Instance Deployment

To govern resource allocation in an on-prem Single-instance deployment, navigate to **Settings** > **System** > **Service Processes** > **Processes**.

Optimize a Distributed Deployment or Hybrid Group

With an on-prem Distributed deployment or hybrid group, you allocate resources per Worker Group. From Stream Home, navigate to **Worker Groups** > `groupName` > **Group Settings** > **Worker Processes**.

In on-prem deployments, you can adjust the following Worker Process settings:

Process count: Indicates the number of Worker Processes to spawn.

- Positive numbers specify an absolute number of Workers. Negative numbers specify a number of Workers relative to the number of CPUs in the system, for example: {`<number of CPUs available>` minus `<this setting>`}.
- The default is `-2`.

- Cribl Stream will correct for an excessive + or - offset, or a `0` entry, or an empty field. Here, it will guarantee at least the **Minimum process count** set below, but no more than the host's number of CPUs available.

Minimum process count: Indicates the minimum number of Worker Processes to spawn. Overrides the **Process count**'s lowest result.

- A `0` entry is interpreted as "default," which here yields `2` Processes.

Memory (MB): Amount of heap memory available to each Worker Process, in MB.

- Defaults to `2048` (see Estimating Memory Requirements).
- Heap memory is allocated dynamically from the OS as it is requested by the Process up to the amount dictated by this setting.
- The **Memory (MB)** setting does not govern external (`ext`) memory.

If a Process needs more than the allocated amount of heap memory, it will encounter an `Out of memory` error, which is logged to `$CRIBL_HOME/log/cribl_stderr.log`. This error indicates either:

- A memory leak.
- The memory setting is too low for the workload being performed by the Process (and possibly, by extension, that the total memory provisioned for the host is insufficient for all Processes).

Cribl support can help you troubleshoot if you experience this error.

For changes in any of the above controls to take effect, you must select **Save** and then restart the Cribl Stream server via **Settings** > **Global** > **System** > **Controls** > **Restart**. In a distributed deployment, also deploy your changes to the Groups.

For example, assuming a Cribl Stream system running on Intel or AMD processors with 6 physical cores hyperthreaded (12 vCPUs):

- If **Process count** is set to `4`, then the system will spawn exactly 4 processes.
- If **Process count** is set to `-2`, then the system will spawn 10 processes (12-2).

Choose a Process Count

The **Process count** setting determines the number of usable Worker Processes. Use the following guidelines to select the optimal configuration for your environment, depending on your CPU type and whether features like the internal Syslog Load Balancer are enabled.

CPU Type/Environment	Process Count Setting	Description
x86 (with Hyperthreading)	`-2` (default)	Reserves headroom for system/API overhead.
Non-Hyperthreaded CPUs (such as, ARM Graviton, AWS c7a)	`-1`	Reserves one physical core for overhead while maximizing usable Worker Processes. Each Worker Process uses up to one physical core.
Containers/Kubernetes	Positive value (such as `4`) and/or use `CRIBL_MAX_WORKERS`	Negative values are computed relative to the vCPUs visible to the container (or to `CRIBL_K8S_CPU_LIMIT` when set).
Syslog Source LB Enabled	Reduce effective Process Count by `1`	Reserves an additional core specifically for the load balancer process. This applies to any scenario where the Syslog Source's internal TCP load balancer is active.

Allocate a Maximum Number of Worker Processes

To provision a container with a set number of Worker Processes, use the `CRIBL_MAX_WORKERS` environment variable.

Worker processes handle data ingestion and processing tasks. Improper configuration can lead to:

- **Unstable container deployments**: Incorrect Worker Process settings might cause your containers to malfunction or become unstable.
- **Outages**: Unforeseen resource consumption due to excessive number of Worker Processes can lead to Cribl crashing and service outages.

A couple notes about the `CRIBL_MAX_WORKERS` environment variable:

- The value you designate represents the **maximum** number of Worker Processes for the container. If there are fewer actual CPUs than designated in the variable, the container will provision with the number of Worker Processes that match the actual number of CPUs.
- Several different factors determine the final number of Worker Processes. Cribl uses the lowest number of all of these:
    - The Cribl license Worker Process limit
    - The number of actual Worker Processes (with **Process count** factored in)
    - The Worker count configured in `cribl.yml`
    - `CRIBL_MAX_WORKERS` (if configured)

  For example:
    - License limit: 7

- Number of actual Worker Processes (OS based): 4, calculated as number of actual Worker Processes (`6`) and **Process count** (`-2`)
- Worker Processes in `cribl.yml`: 5
- `CRIBL_MAX_WORKERS`: 3

There are three final assigned Worker Processes, because 3 is the lowest number of all the values above.

Example configurations

Docker

```
docker run -e CRIBL_MAX_WORKERS=2
```

Helm

```
helm install --set env.CRIBL_MAX_WORKERS=2
```

The `values.yml` file

```
env:
  CRIBL_MAX_WORKERS: 2
```

Copy

By default, Cribl sets the Worker Process count to `-2` (`Worker Processes (CPU cores) = total CPU cores - 2`). If you don't configure `CRIBL_MAX_WORKERS` and directly launch a container, it inherits the default behavior. This can lead to inefficient resource usage, particularly on machines with many CPU cores.

Scale Out

When data volume, processing needs, or other requirements exceed what a single instance can sustain, a Cribl Stream deployment can span multiple Nodes. This is known as a distributed deployment. Here, you can centrally configure and manage Nodes' operation using one administrator instance, called the Leader.

It's important to understand that Worker Processes operate in parallel, i.e., independently of each other. This means that:

1. Data coming in on a single connection will be handled by a single Worker Process. **To get the full benefits of multiple Worker Processes, data should come in over multiple connections.**
   For example, it's better to have five connections to TCP 514, each bringing in 200GB/day, than one at 1 TB/day.

2. Each Worker Process will maintain and manage its own outputs. For example, if an instance with two Worker Processes is configured with a Splunk output, then the Splunk destination will see two inbound connections.
   For further details, see Shared-Nothing Architecture.

For connection concurrency limits, see Number of Connections.

Capacity and Performance Considerations

As with most data processing applications, Cribl Stream's expected resource utilization will be proportional to the type of processing that is occurring. For instance, a Function that adds a static field on an event will likely perform faster than one that applies a regex to find and replace a string. Currently:

- A Worker Process will utilize up to 1 physical core (encompassing either 1 or 2 vCPUs, depending on the processor type.
- Processing performance is proportional to CPU clock speed.
- All processing happens in-memory.
- Processing does not require significant disk allocation.

Throughout these guidelines, we assume that **1 physical core** is equivalent to:

- 2 virtual/hyperthreaded CPUs (vCPUs) on Intel/Xeon or AMD processors.
- 1 vCPU on Graviton2/ARM64 processors.

The general sizing guideline from Cribl is to plan for a maximum of 200 GB/day in+out per vCPU when hyperthreading is enabled (2 vCPUs per physical core on Intel/AMD). At its simplest, this can equate to 100 GB/day in, 100 GB/day out when sending full-fidelity data to a single Destination.

Examples

Estimating Number of Cores

In estimating core requirements - per processor type, it's important to distinguish among vCPUs versus physical cores.

Intel/Xeon/AMD Processors with Hyperthreading Enabled

To estimate the number of cores needed: Sum your expected input and output volume, then divide by 400GB per day per physical core.

- Example 1: 100GB IN -> 100GB out to each of 3 destinations = 400GB total = 1 physical core (or 2 vCPUs).

- Example 2: 3TB IN -> 1TB out = 4TB total = 10 physical cores (or 20 VCPUs).
- Example 3: 4 TB IN -> full 4TB to Destination A, plus 2 TB to Destination B = 10TB total = 25 physical cores (or 50 vCPUs).

Graviton2/ARM64 Processors

Here, 1 physical core = 1 vCPU, but overall throughput is ~20% higher than a corresponding Intel or AMD vCPU. So, to estimate the number of cores needed: Sum your expected input and output volume, then divide by 480GB per day per vCPU.

- Example 1: 100GB IN -> 100GB OUT to each of 3 destinations = 400GB total = 1 physical core.
- Example 2: 3TB IN -> 1TB OUT = 4TB total = 8 physical cores.
- Example 3: 4 TB IN -> full 4TB OUT to Destination A, plus 2 TB OUT to Destination B = 10TB total = 21 physical cores.

Remember to also account for an additional core per Worker Node for OS/system overhead.

Estimating Number of Nodes

When sizing the number of Stream Worker Nodes, there are two other factors to consider:

- Number of Nodes sized for peak workloads;
- Number of Nodes offline.

This will allow you to create a robust Cribl Stream environment that can handle peak workloads even during rolling restarts and maintenance downtime.

General guidance includes:

- On each Node, Cribl recommends at least 4 ARM vCPUs, or at least 8 x86 vCPUs (i.e., 4 cores with hyperthreading). Below this theshold, the OS overhead from reserved threads claims an excessive percentage of your capacity.
- On each Node, Cribl recommends no more than 48 ARM vCPUs, or 48 x86 vCPUs (i.e., 24 cores with hyperthreading). This allocation handles disk I/O requirements when persistent queueing engages.
- Plan to successfully handle peak workloads even when 20% of your Worker Nodes are down. This allows for OS patching, and for conducting rolling Stream upgrades and restarts.
- For customers in the 5-20 TB range, size for 4-8 Worker Nodes per Worker Group.

Sizing Example for a Deployment

Step 1: Calculate the total inbound + outbound volume for a given deployment.

In this example, your deployment has 6TB/day streaming into Worker Nodes for a given site, and this data is being routed to both cheap mass storage (S3) and to your system of analysis - 10 TB/day going out.

Stream Workers must have enough CPUs to handle a total of 16 TB per day IN+OUT.

Step 2: From the available server configurations, size the correct number of Nodes for your environment.

You are considering an AWS Compute-optimized Graviton EC2 instance, with either 16-vCPU or 8-vCPU options. We'll walk you through how to calculate based on each option.

Using Graviton 16-vCPU instances

- Each Node would have 15 vCPUs available to Cribl Stream, each at 480 GB per day per vCPU.
- 15 vCPUs per Node at 480 GB per day per CPU = 7200 GB/day per Node.
- 7200 GB/day divided by 1024 GB/TB = 7 TB/day.
- To handle the workload of 16 TB/day, you need 3 Nodes. Make sure you always round up.
- 20% of 3 Nodes ≈ 1 Node. Make sure you always round up. To provide for redundancy during patching, maintenance, or restarts, you need one additional server.

With 4x 16-vCPU Nodes, you have 28 TB/day capacity, and can support 21 TB/day with one Node down.

Using Graviton 8-vCPU instances

- Each Node has 8 vCPUs (physical cores). For Graviton, we reserve 1 vCPU for OS and API overhead.
- Each Node would have 7 vCPUs available to Stream, each at 480 GB per day per vCPU.
- 7 vCPUs per Node at 480 GB per day per CPU = 3360 GB/day per Node.
- 3360 GB/day divided by 1024 GB/TB = 3.3 TB/day.
- To handle the workload of 16 TB/day, you need 5 Nodes. Make sure you always round up.
- 20% of 5 Nodes = 1 Node. To provide for redundancy during patching, maintenance, or restarts, you need one additional server.

With 6x8-vCPU Nodes, you have 19.8 TB/day capacity, and can support 16.5 TB/day with one Node down.

If you are optimizing for price, AWS pricing is linear based on the total number of cores. So six 8-vCPU systems are roughly 75% the cost of four 16-vCPU systems.

If you are optimizing for potential future expansion, the 4x16-vCPU systems would offer ideal additional capacity.

Estimating Memory Requirements

It can be difficult to determine how much heap memory you need. In general, we recommend that you start with the default heap memory allocation of 2048 MB (2 GB) per Worker Process, and add more memory as you

find that you're hitting limits.

Heap memory use is consumed per component, per Worker Process, as follows:

- Lookups are loaded into heap memory. Large lookups require extra heap memory allocation (see Choose a Lookup File Storage Mode).
- Stateful Functions (Aggregations and Suppress) consume heap memory proportional to the rate of data throughput.
- The Aggregations Function's consumption of heap memory further increases with the number of **Group by** fields.
- The Suppress Function's consumption of heap memory further increases with the cardinality of events matching the **Key expression**. A higher rate of distinct event values will consume more heap memory.

Since version 4.13, due to an upgrade of the Node.js version used by Cribl Stream, baseline memory usage may increase on hosts that are not memory-constrained, because the new Node.js version is more conservative with memory release.

External Memory

External memory is allocated to in-memory buffers to hold data to be delivered to downstream services. Downstream services can consume as much external memory as they need, up to the limit that the operating system has available. External memory is not subject to the heap memory limit that you can configure in Cribl.

Destinations are the main consumers of external memory. It can be difficult to determine how much external memory you need, although file-based Destinations require more in-memory buffers and therefore more external memory. In a load-balanced configuration, the amount of external memory that a Destination consumes increases with the number of configured endpoints. For example, Splunk Load Balanced (LB) Destinations may use up to 4 MB of memory for each indexer, which means that a single Splunk LB Destination with 300 indexers could require 1.2 GB of extended memory per process.

Recommended AWS, Azure, and GCP Instance Types

Meet sizing requirements with multiples of the following instances. In all cases, reserve at least 5GB disk storage per instance, and more if you enable persistent queuing.

AWS

For Amazon EC2 compute optimized instances. For other options, see the AWS documentation.

Architecture	Minimum	Recommended
Intel	• c5d.2xlarge (4 physical cores, 8vCPUs) • c5.2xlarge (4 physical cores, 8vCPUs)	• c5d.4xlarge or higher (8 physical cores, 16vCPUs) • c5.4xlarge or higher (8 physical cores, 16vCPUs)
Graviton2/ARM64	• c6g.2xlarge (8 physical cores, 8vCPUs) • c6gd.2xlarge (8 physical cores, 8vCPUs)	• c6g.4xlarge or higher (16 physical cores, 16vCPUs) • c6gd.4xlarge or higher (16 physical cores, 16vCPUs)

Azure

Architecture	Minimum	Recommended
Intel (Fsv2 sizes series)	Standard_F8s_v2 (4 physical cores, 8vCPUs)	Standard_F16s_v2 or higher (8 physical cores, 16vCPUs)
Arm (Dplsv5 sizes series)	Standard_D8pls_v5 (8 physical cores, 8vCPUs)	Standard_D16pls_v5 or higher (16 physical cores, 16vCPUs)

GCP

For compute-optimized machine types.

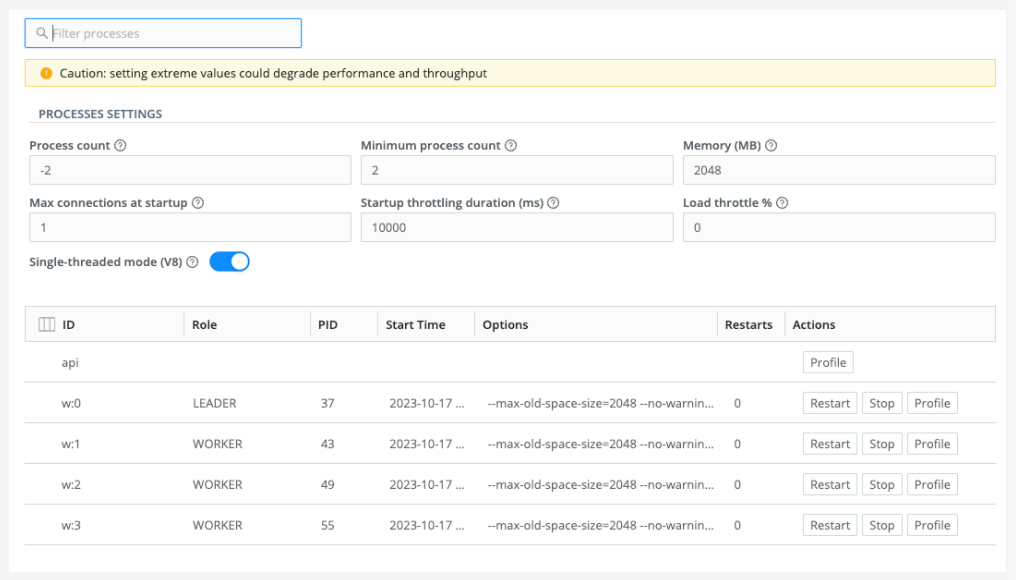
Architecture	Minimum	Recommended
Intel	• c2-standard-8 (4 physical cores, 8vCPUs) • n2-standard-8 (4 physical cores, 8vCPUs)	• c2-standard-16 or higher (8 physical cores, 16vCPUs) • n2-standard-16 or higher (8 physical cores, 16vCPUs)

Measuring CPU or API Load

You can profile CPU and API usage on individual Worker Processes.

Single-Instance Deployment

Go to **Settings** > **System** > **Service Processes** > **Processes**, and select **Profile** on the desired row.

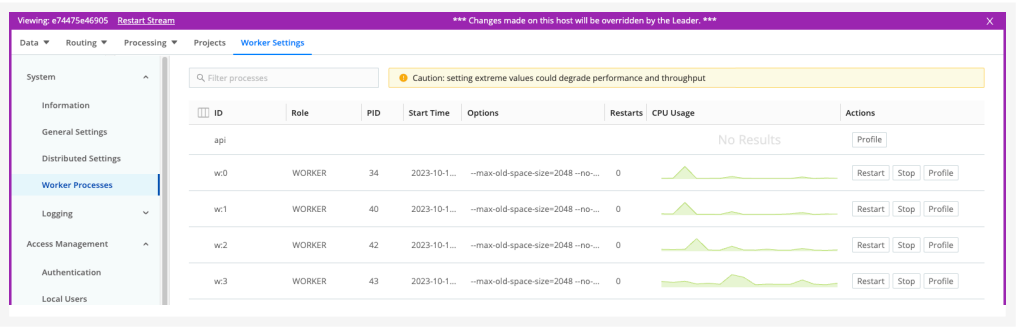


Worker CPU profiling (single-instance)

Distributed Deployment

This requires a few more steps:

1. Enable UI access to Workers if you haven't already.
2. In the sidebar, select **Workers**.
3. Select the **GUID** link of the Worker Node you want to profile.
4. Select **Worker Settings** from that Worker Node's submenu.
5. Select **System** > **Worker Processes**, and then select **Profile** on the desired Worker Process. To profile the API usage, click **Profile** on the **api** row.



Worker CPU profiling (distributed)

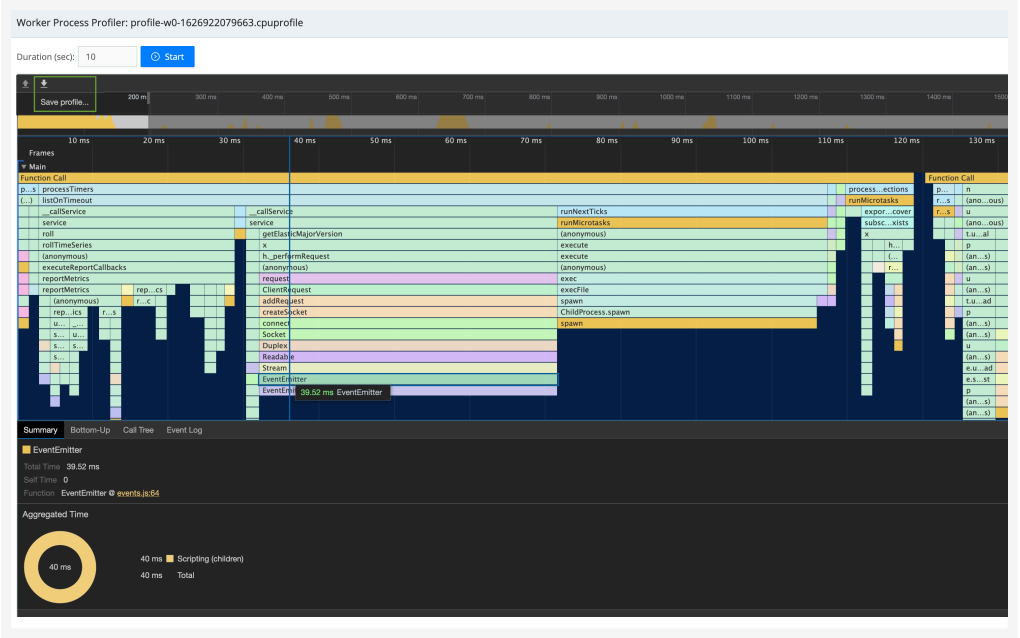
Generating a CPU or API Profile

In either a single-instance or distributed deployment, you will now see a **Worker Process Profiler** modal.

The default **Duration (sec)** of `10` seconds is typically enough to profile continuous issues, but you might need to adjust this - up to several minutes - to profile intermittent issues. (Longer durations can dramatically

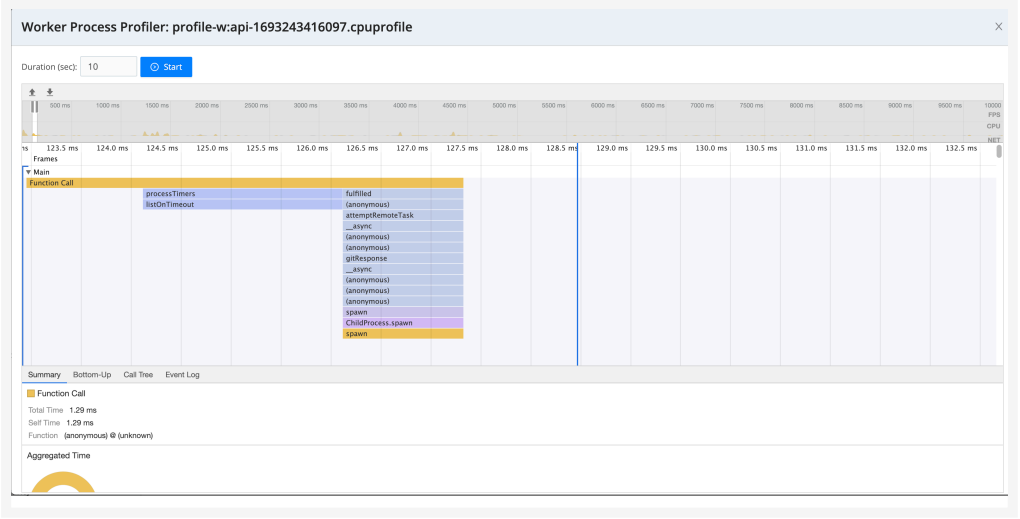
increase the lag before Cribl Stream formats and displays the profile data.)

Click **Start** to begin profiling. After the duration you've chosen (watch the progress bar), plus a lag to generate the display, you'll see a profile something like this:



Worker CPU profile

Profiling the API process can be a helpful troubleshooting tool if the Leader is slow.



API profile

Below the graph, tabs enable you to select among **Summary**, **Bottom-Up**, **Call Tree**, and **Event Log** table views.

2.4.2. OS Tuning For Large Deployments

Large on-prem deployments that handle high connection volumes or data rates may require operating system (OS) tuning to maintain optimal performance. To prevent performance issues on high-load Worker Nodes, you might need to tune your OS.

Review and apply the following OS-level configurations to address common bottlenecks like connection tracking table exhaustion and buffer overruns, ensuring stable operation for your Cribl environment.

The parameters described in this section are Linux kernel and operating system settings, not Cribl Stream configuration options. They directly affect how the OS handles network connections and system resources for high-throughput Cribl Stream workloads.

Connection Tracking (Netfilter)

Applies to: Worker Nodes (handling high volumes of inbound network connections.)

In deployments where you expect a high number of inbound connections, Linux netfilter connection tracking (`conntrack`) could become a bottleneck. Connection tracking is enabled by default on most Linux distributions and maintains a table of active network flows. When this table fills up, the kernel drops packets, which can cause connection failures and data loss.

When to Tune

Monitor your system logs (`/var/log/messages` or `journalctl \-k`) for messages indicating conntrack table exhaustion:

```
nf_conntrack: table full, dropping packet
```

If you observe these messages, you should increase the `conntrack` table size and potentially adjust related parameters.

Key `Conntrack` Parameters

These `sysctl` parameters govern connection tracking behavior:

- `net.netfilter.nf_conntrack_max`: Maximum number of flows tracked concurrently. By default, it derives from the buckets value: `nf_conntrack_max= nf_conntrack_buckets × 4.`
- `net.netfilter.nf_conntrack_buckets`: Hash buckets for the `conntrack` table. If not set at module load, the kernel computes a default from total system memory (roughly `total memory in bytes / 16384`) with floor and ceiling limits applied.

For the full list of netfilter `conntrack sysctls` and their defaults, see Netfilter Conntrack Sysfs variables.

Check Current Values and Live Load

Use the following commands to inspect your current conntrack configuration and active connection count:

```
# Current max entries
sysctl net.netfilter.nf_conntrack_max

# Current active tracked flows (count)
sysctl net.netfilter.nf_conntrack_count

# Alternatively via procfs
cat /proc/sys/net/netfilter/nf_conntrack_max
cat /proc/sys/net/netfilter/nf_conntrack_count
```

Copy

Compare the count against the max. If the count approaches or equals the max during normal operation, you need to increase capacity.

Method 1 (Recommended): Increase the `conntrack` Table Size

This is the standard and generally recommended approach to eliminate `conntrack` bottlenecks by simply giving the table more capacity.

To ensure stability, set the maximum number of tracked flows high enough to maintain adequate margin above your peak observed load. For very large deployments handling hundreds of thousands of concurrent connections, you may need values of 4 million or higher.

```
# For example, to set approximately 2 million tracked connections (adjust per sizing)
echo "net.netfilter.nf_conntrack_max=2097152" >> /etc/sysctl.conf
sysctl -p
```

Copy

Adjust the value based on your deployment's actual connection load.

If you substantially increase `nf_conntrack_max`, review `nf_conntrack_buckets` for balance. By default, the kernel derives buckets from available memory, and max defaults to approximately 4 times the bucket count. Maintaining this ratio helps ensure efficient hash table performance.

Method 2 (Extreme Case): Disable Connection Tracking

In extremely high-throughput, purpose-built environments where Cribl Stream is the only service and network overhead is a top concern, disabling conntrack entirely can be considered an aggressive measure. While this is often the simplest and most effective measure for some on-prem deployments, approach it with caution due to its side effects.

How to Disable conntrack (Persistent)

To prevent the module from loading at boot:

```
echo "install nf_conntrack /bin/true" >> /etc/modprobe.d/disable-conntrack.conf
```

Copy

Disabling conntrack is **NOT generally recommended** and carries significant consequences:

- Breaks stateful firewalls: Most modern firewall rules (iptables, nftables) rely on conntrack for stateful filtering. Disabling it will break stateful firewalls, requiring a complete rewrite of your rules to be stateless.
- Breaks NAT/Kubernetes: Network Address Translation (NAT) and IP masquerading are heavily dependent on conntrack. Disabling it will break networking for container technologies like Kubernetes and Docker.

Kubernetes Deployments

If you run Cribl Stream on Kubernetes, **do not disable netfilter or conntrack**. Kubernetes networking (including kube-proxy and network policies) depends on conntrack for proper operation. Tune conntrack parameters as described in Method 1 instead.

File Descriptor Limits

Applies to: Leader and Worker Nodes

The Cribl Stream systemd unit sets `LimitNOFILE=65536` by default, which aligns with our minimum recommendation. This limit applies per process (not systemwide or per user), meaning the API and each Worker Process can each open up to `65,536` file handles. This is generally more than sufficient for high-volume workloads, including those involving:

- File-based Destinations (S3, Filesystem, etc.), especially with partition expressions that possess a high cardinality. For details, see Amazon S3 Better Practices.
- Large numbers of concurrent inbound and outbound connections

For more information on running Cribl Stream as a service, see Run Cribl Stream.

Diagnosing `Too Many Open Files` Error

The default value is typically sufficient for normal operation. Increasing the limit is generally not the correct fix; it usually only postpones the necessity of identifying and resolving the root cause of the process hitting the resource ceiling.

Start your investigation by determining which process is affected. The `cribl.log` file will reflect a `too many open files` error, referencing one of two error codes:

- `EMFILE (Too many open files)`: This means the per-process limit has been reached. This is the common case that the LimitNOFILE setting addresses.
- `ENFILE (File table overflow)`: This means the systemwide (kernel-level) limit for all open files has been reached. The fix for this (increasing the `fs.file-max sysctl`) is outside the scope of this section, as it's rarely encountered.

The scenario that leads to the common `EMFILE` error is usually the API Process being overwhelmed due to Worker Processes being too busy. The API Process always receives the connection attempt first, then hands the file descriptor off to a Worker Process.

The `EMFILE` condition typically occurs when a large number of clients (on the order of thousands) establish connections that are not closed have excessively long timeouts. Since Worker Processes are often unable to accept the handoff of these file descriptors from the API, the API Process accumulates thousands of connections waiting to be serviced. This accumulation of active or stuck connections rapidly exhausts the file descriptor limit for the API Process. The proper fix, therefore, is to address why the Worker Processes are too busy.

Start by analyzing your Pipelines using the Pipeline Profiling feature to identify bottlenecks and reduce Worker load. For details, see Pipeline Profiling.

How to Increase the Per-Process Open File Limit (if Necessary)

If, after investigation, you've confirmed that the workload legitimately requires more file handles (such as, in extremely large, high-volume, or unique deployments), you can increase this per process limit via a systemd override:

```
# Create or edit the systemd override
sudo systemctl edit cribl
```

Copy

Add the following (example) content:

```
[Service]
LimitNOFILE=262144
```

Copy

Save the file and restart the Cribl Stream service for the change to take effect:

```
sudo systemctl restart cribl
```

Copy

You can verify the current limit for running processes by checking `/proc/<pid\>/` limits for any Cribl Stream process.

```
# For exampalefor PID 7284
cat /proc/7284/limits
```

Copy

UDP Receive Buffers

Applies to: Worker and Edge Nodes (Nodes hosting UDP input Sources).

For high-rate UDP ingest (such as Syslog, NetFlow, or IPFIX), the default kernel UDP receive buffers may be insufficient, leading to packet drops before data reaches Cribl Stream. Increasing these buffers can significantly reduce packet loss.

When to Tune

Cribl recommends tuning UDP buffers to accommodate short-lived but large data spikes. Tuning UDP buffers is recommended to accommodate short-lived but large data spikes.

Before tuning the buffer size, you should confirm that the data is being properly distributed across all the Worker Nodes in the Group. If the data is improperly load-balanced and only hitting a subset of your Nodes, you must address the distribution issue first. Fixing the load balancing might resolve the packet drop problem entirely, making buffer adjustments unnecessary.

For permanent solutions to consistently high data volume, scaling your environment (adding more worker processes or nodes) is the correct approach.

Monitor for UDP packet drops using:

```
# On modern operating systems, check for packet receive errors using ss
ss -ua
# Alternatively, if ss is unavailable
netstat -su
```

Copy

Look for increases in `packet receive errors` or `RcvbufErrors` under the UDP section.

Method 1: Kernel-Level Tuning (System-Wide OS Configuration)

This method applies the buffer increase to **all** UDP sockets on the system.

Increase the kernel's UDP receive buffer sizes:

```
# Set to 25 MB (adjust based on your volume)
echo "net.core.rmem_max=26214400" >> /etc/sysctl.conf
echo "net.core.rmem_default=26214400" >> /etc/sysctl.conf
# Apply the changes immediately without rebooting:
sysctl -p
```

Copy

Method 2: Per-Input via Cribl Stream Configuration

We generally recommend configuring the buffer size within Cribl Stream (on the specific UDP Source) for two main benefits:

- **Portability:** The setting becomes part of your Cribl Stream configuration, ensuring it is automatically applied to newly deployed Nodes without requiring manual OS tuning on each one.
- **Scope:** The larger buffer is applied only to the specific UDP sockets used by Stream Sources that have this setting defined, avoiding the general application to all system sockets.

Start with `25` MB and adjust based on your data rates and observed drop rates.

For detailed UDP tuning guidance specific to Cribl Stream, see:

- Syslog Source - UDP Tuning
- UDP (Raw) Source

SELinux Configuration

Applies to: Leader, Worker, and Edge Nodes.

In hardened environments running SELinux in enforcing mode, ensure proper SELinux contexts are set on Cribl directories to allow Stream to operate correctly. Use SELinux audit logs to troubleshoot any denials and adjust policies as needed.

For detailed guidance on running Cribl Stream with SELinux, see SELinux Configuration.

2.4.3. Architectural Considerations

This guide introduces Cribl Stream reference architectures by outlining some considerations to keep in mind when architecting your deployment. It covers different potential architectures, with each approach's advantages, limitations, and alternatives.

Architecting a deployment is always about finding the balance between cost versus risk. Once you're familiar with the contents of this guide, you'll be able to consciously make decisions about where your deployment should sit on that spectrum.

See our reference architectures for different use cases. You can replay several Webinar presentations of this material on Cribl's YouTube channel.

"It Depends"

This guide is not intended to replace Sizing and Scaling, but rather to complement it by outlining the consequences of various architectural decisions. (As needed, please either read Sizing and Scaling first, or refer to it to look up concepts.)

General Sizing Considerations

Some basic sizing variables are your processors, expected data throughput, and requirements for memory, disk space, connections volume, and the Stream Leader's own needs.

Processors

Your primary consideration in sizing processors is your data throughput capacity, which depends on the number of Worker Processes.

Overall guideline: Allocate 1 physical core for each 400GB/day of IN+OUT throughput.

This can vary based on processor speeds or types. (For example, Graviton ARM processors' throughput is 20% higher than x86-64 processors' reference 400GB/day.) But in this guide's calculations, we'll assume 400GB/day IN+OUT per processor, and dig into what that really means. Please adjust your own calculations as needed.

Throughput

Where we reference total throughput capacity below: Because Cribl Stream can send events to multiple Destinations and filter out unneeded events, IN won't always be the same as OUT. For example, 400GB IN+OUT

could mean 200GB IN + 200GB OUT, but it could also mean 300GB IN + 100GB OUT, or any combination of IN+OUT that adds up to 400GB/day.

Memory

Memory requirements also play into architecture considerations. Depending on what type of processing you'll be doing, those requirements might vary. You might know your memory requirements in advance. Or you might monitor memory usage and increase it as your processing complexity increases. The default of 2GB per Worker Process is a good starting point.

Since version 4.13, due to an upgrade of the Node.js version used by Cribl Stream, baseline memory usage may increase on hosts that are not memory-constrained, because the new Node.js version is more conservative with memory release.

Disk Space

The base recommendation, 5GB of free disk space, applies to the volume where Cribl Stream is installed. Beyond that, if you choose to use Persistent Queues, you will need to calculate additional disk space - based on the velocity of your data, and your desired duration before reaching a full queue. Normally, you must define the maximum disk space a queue can use, per Worker Process, for each Source or Destination. However, if you use a separate volume for Persistent Queues, you can remove those limits, and allow the queues to take up the whole volume if needed.

Object-store Destinations use a staging directory to write out files before sending them out. If you're using one of these Destinations, please review our Amazon S3 Better Practices for sizing and architecture considerations.

In both cases, using fast disks is important, to prevent the disks from becoming bottlenecks.

Number of Connections

Beyond throughput, you have to effectively manage the connection overhead. A single connection limits the workload to one Worker Process. Consider configuring the maximum number of concurrent connections (depending on your Source) to distribute the data and boost the total bytes per process the system handles. When dealing with high-volume streams, use frequent connection rotation (low reuse).

Worker Processes enforce specific concurrent limits on incoming Sources:

- TCP Sources (such as Splunk, Syslog, raw TCP): The limit is usually `1,000` active connections per Worker Process (WP). Reaching this limit can cause connection failures. You can increase the limit, but be aware that more concurrent activity directly increases the demand on CPU and memory.
- HTTP-family Sources (such as Cribl HTTP, Splunk HEC, Windows Event Forwarding, OTel HTTP): The limit is usually `256` active requests per WP. For HTTP traffic, the number of active requests being processed at

any moment is more critical than the total number of open connections. Reaching the limit will result in an `HTTP 503 Server is busy` response.

When you see rejected requests (such as `HTTP 503` errors or debug logs showing `Dropping request`), it indicates you might need to increase the active limit.

Scaling decisions should be based on multiple factors:

- Worker Process limits: The concurrent limits on incoming Sources.
- Bytes-Per-Second (BPS): The objective metric for the workload.
- Connection churn: How often connections are started and stopped (low reuse).
- Data complexity: The impact of your processing Pipeline on resource demand.
- Client limits: The number of parallel connections supported by the sending system (e.g., a single-connection limit on a firewall). Load-balancing features may be necessary to overcome single-connection limits on a client.

Increasing a limit allows more data to be ingested concurrently, which places immediate, higher demands on CPU and memory. You must ensure your infrastructure has sufficient excess capacity to handle the resulting processing load.

The ideal limit depends on your goals and resources. To prevent all rejections (if infrastructure allows), set the limit to `0` (unlimited).

You do not need to worry about "port exhaustion" (running out of available local ports) because many incoming connections can share a single listening port. The system manages these connections efficiently using internal file limits. Monitor these limits and memory usage when you increase the connection cap.

Monitor memory carefully when using TCP Destinations that load balance to many unique targets. The issue isn't a single Destination configuration, but the total number of target host connections maintained by each Worker Process. Each connection consumes `2` to `4` MB of memory. To manage high memory consumption in these scenarios, set a connection limit (non-zero value) on the load-balancing Destination. You must determine the specific limit based on your available resources.

Leader Node

All the above discussion concerns Worker Nodes, because they are the ones responsible for processing the data. The Leader Node requirements should not need to expand beyond these initial recommendations, unless the Leader needs to manage Edge Fleets, in which case reference Cribl Edge docs.

Finding the Balance Between Risk and Cost

Beyond your basic requirements, you should of course plan contingency capacity to support resource demands that exceed ideal circumstances.

Plan for Failure

Given the wrong circumstances, failure will happen. Your job, when architecting a deployment, is to examine the various failure scenarios and decide what level of resources you are willing to allocate to prevent failure. This is analogous to deciding how many nines of reliability you're willing to pay for.

Throughput Variability

The 400GB/day IN+OUT guidance mentioned above assumes that your data flow is relatively flat. This means you're expecting that every second, each Worker Process is handling about ~4.74MB of data IN+OUT. However, in reality, the load varies based on time of day or other system activities.

For example, you might experience higher load during business hours. Or you might have an hourly upstream process that causes spikes of activity. Also, depending on the nature of your organization, you might have certain days, or planned events, when you expect users' increased load on your systems. Then there are unplanned events, such as outages or DDoS attacks, that might lead to decreased capacity or increased demands.

Another factor to consider, if you're using Persistent Queues, is that flushing those queues will increase the total throughput demand - including on the downstream systems that need to be able to handle the burst. Unfortunately, there are too many variables to have a generic recommendation here, but a starting point of ~10% headroom is a good idea. As we discuss in the conclusion, you'll need to test these assumptions.

Now that we're digging deeper into throughput, you should start thinking about where to set a threshold, beyond which you know your system is likely to fail. If you have data flowing into a system of analysis that can let you look at your current ingestion rate, this can be a good start.

You want to look at the busiest times, and drill down far enough to see what your historical maximum per-second throughput has been. Past performance is not guarantee of future results, but it will at least give you an idea of what to expect.

If you have new Sources, or if you don't have a good way of estimating current throughput, you'll have to make some assumptions about future behavior. In the end, take whatever you come up with for expected MB/sec throughput, add whatever safety margin you feel is appropriate, and multiply back out by 84 (24*60*60/1024) to get GB/day throughput.

Exceeding Capacity

As you are deciding how much safety margin to plan into your environment, it also helps to think through what happens when you exceed capacity. Depending on the types of Sources you have, an overloaded deployment might have different failure modes. This ranges from the possibility of immediate data loss in the case of UDP, to simple delays in the case of Pull Sources. Some Push Sources, such as Cribl Edge or Splunk Universal Forwarders, have the capacity to pause or buffer the data flow for some duration.

Separate Data by Worker Groups, Routes, or Both

If you adopt a distributed deployment with multiple Worker Groups, we recommend separating your data volume by Worker Groups, Routes, or a combination of both, depending on your business needs. Common business reasons for sorting by Worker Groups include data isolation by business group, data privacy, and cost savings through geo-isolation. Also, consider which method of separation will be easier to understand and manage as your organization grows.

Connecting Cribl Stream Groups/Instances

When sending data between Workers (whether they report to the same or to different Leaders), use the Cribl HTTP Source and Destination pair, or the Cribl TCP Source and Destination pair. Ingest data usage is only charged to the receiving Organization.

While Cribl Stream supports multiple Sources/Destinations for sending and receiving, TCP JSON is ideal because it supports TLS, has solid compression, and is overall well-formatted to maintain data's structure between sender and receiver.

Single-Instance Deployment

The simplest type of deployment is single-instance. Let's look at the case of a 4 core (8 vCPUs), 16GB RAM, single-instance deployment:

Resource	Total	Used by WPs	Used by system/API
Cores (vCPUs)	4 (8)	3 (6)	1 (2)
RAM	16GB	12GB	4GB

With the default assumption of 400GB/day IN+OUT, and sending to a single Destination, the above configuration can process an average of 1200GB/day.

Let's make a few more assumptions:

- Peak usage during the day is 2x average.
- Persistent Queues for 4 hours of data.
- We want to withstand a 2x burst over typical usage from the Source.
- Our Source is syslog UDP*, so going over capacity means lost data.

(* We're assuming that the load is distributed evenly among all Worker Processes. Further down, we'll talk about why we're making that assumption, and the dangers of "pinning" Worker Processes.)

We're working backwards for this one, but it's a good exercise. Taking the 1200GB/day capacity, dividing it by 2 for peak load, and then dividing by 2 again for burst tolerance, we come up with 300GB/day adjusted capacity.

So with this deployment, we can safely say we can handle a 300GB/day stream of syslog UDP data.

Because we want the ability to keep 4 hours of data in a Persistent Queue, we want to use the peak-load number for that. Our upfront assumption is that peak load is 2x normal load, which is 600GB/day, or 25GB/hour. 100GB of disk space should be sufficient, unless we expect the 2x bursts to last a significant duration. If we want to withstand a 1-hour burst during peak times, 125GB of disk space would be needed.

Where Single-Instance Fails

The obvious practical limitation here is that all data is flowing through a single system. This means that if you bring your single-instance deployment down for patching, or experience an unexpected outage, you'll be losing data. This might be acceptable for test purposes, but in production, there are very few scenarios in which you **won't** lose data with a single-instance deployment:

- All your Sources are Pull Sources.
- Your Push Sources can pause the data flow when Stream Workers are not available.

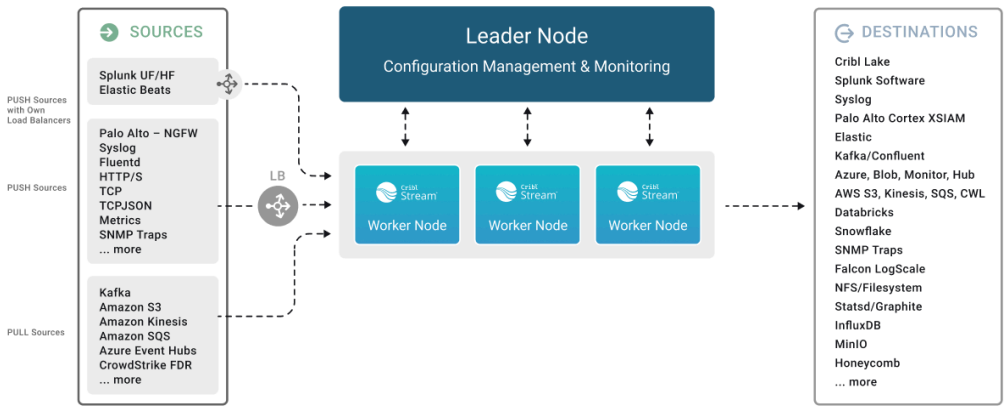
A distributed deployment, with multiple Worker Nodes, is a more robust solution that doesn't have these limitations. So next, let's look at a simple distributed example.

Distributed Deployment

The simplest distributed deployment involves a Leader Node and one or more Worker Nodes. A single Worker Node suffers from the same limitations as the single-instance deployment, so in this section we'll look at scenarios with multiple Worker Nodes.

You might be sizing a Worker Group when working with either a Cribl.Cloud Leader or an on-prem Leader. For an on-prem Leader, use our recommended system requirements.

The main thing to consider with multiple Workers is that you'll need a load balancer for all Push Sources that don't natively support balancing across multiple Worker Nodes. You can see this added load balancer called out (as **LB**) in our Distributed Deployment Architecture diagram:



Distributed deployment architecture

How Many Worker Nodes?

The next consideration is, how many Worker Nodes do you need? Having multiple Worker Nodes allows you to continue processing data while one Node is down for maintenance - so your sizing is determined by the remaining Worker Nodes. Here, you have a matrix of adjusting scaling vertically vs. horizontally, so let's look at the math:

Effectively, 2 Worker Nodes means you lose 50% capacity when one is down. With 3 Nodes, you lose 33%. With 6 Nodes, you lose 17%. You will lose 2 vCPUs in overhead on each Node, but at larger scales, the benefits outweigh the costs.

Let's look at an example using the naive 400GB/day per core capacity (i.e., 200GB/day per vCPU or Worker Process). We'll pick approximately comparable desired throughput of 5.6-6TB/day, and see how the math works out.

2 Worker Nodes

32 vCPUs x2 Nodes = 64 vCPUs (- 2 vCPUs/Node for overhead) = 60 WPs

1 Node downs during patching: 30 WPs*0.2=6 TB capacity

6 TB/64 vCPUs = 0.09 TB/vCPU

3 Worker Nodes

16 vCPUs x3 Nodes = 48 vCPUs (-2 vCPUs/Node for overhead) = 42 WPs

1 Node downs during patching: 28 WPs*0.2=5.6 TB capacity

5.6 TB/48 vCPUs = 0.12 TB/vCPU

6 Worker Nodes

8 vCPUs x6 = 48 vCPUs (-2 vCPUs/Node for overhead) = 36 WPs

1 Node downs during patching: 30 WPs*0.2 TB/WP = 6 TB capacity

6TB/40 vCPUs = 0.15 TB/vCPU

As we can see, once your required throughput is high enough to make the 2 vCPU/Node tax negligible, a larger number of Worker Nodes is advantageous. You can do the math for your specific use case - remembering to adjust for expected peak usage and desired spike tolerance.

That said, having **3 Worker Nodes is usually the recommended minimum**, since this provides resilience against unplanned outages even when one Worker is down for planned maintenance.

Where a Single Worker Group Can Fail

Even provisioning an "adequate" number of Worker Nodes, in a single Worker Group, might not be fully resilient if you encounter connection pinning, data bursts, or other unexpected resource demands.

Pinning

In Single-Instance Deployment above, we included the caveat of "assuming that the load is distributed evenly among all Worker Processes." In reality, you need to validate this assumption. In some situations, you'll need to take additional measures to assure that it's true, or work around cases when it is not.

Pinning is a situation where a single TCP or UDP Source is sending a large amount of data over a single connection, causing that connection to be "pinned" to a single Worker Process. If the throughput is high enough, and the load balancer is unable to break up that connection, the affected Worker Processes might be overloaded and unable to keep up.

In a Distributed Deployment, to mitigate pinning, the Syslog Source provides a TCP Load Balancing option, as do the TCP JSON and Cribl TCP Sources.

On TCP Sources and deployments where these options are unavailable, you can minimize pinning by having the sender establish separate connections. If that's not possible, try isolating a Worker Group that receives this input from all other processing responsibilities - this will maximize the throughput that this Group can handle.

Burst Loads Affecting Push Sources

Most of the time, your Push Sources will generate a reasonably predictable, consistent load. Using the sizing considerations outlined above, you should be able to come up with a Worker Group sized appropriately for your desired risk-versus-cost balance.

However, this balance can be upset when you have a surge of data that makes your system temporarily busier than you accounted for. The greatest impact would be felt by Push Sources that have no way to handle backpressure, such as UDP Sources.

A prime example of data burst is an ad-hoc run of a large Collection job. With multiple Worker Groups, you can separate Push versus Pull Sources, and can even use an Auto Scaling group group for large Collection jobs if desired.

Unexpected Resource Hogs

Our throughput estimates are based on certain assumptions about the average level of complexity of operations performed on the data. As you're creating your Pipelines, at some point you might decide that you require additional resources to meet your data-processing needs. Certain combinations of Function, logic, and throughput will consume extra resources.

When expanded resource demands happen in a predictable, controlled manner, all is well. However, we've seen cases where an inefficient regular expression, or a memory-hungry custom Collection script, will cause a whole Worker Group to get overloaded. This leads to crashes, or to performance poor enough to cause data

loss. Pipeline profiling helps with some of this. But using separate Worker Groups is an option to provide additional isolation.

Multiple Worker Groups

At the beginning of Distributed Deployment, we considered the limitations of a single Worker Group, and examined some strictly sizing-related reasons for multiple Worker Groups. Here, let's consider some other reasons you might decide to take advantage of multiple Worker Groups.

In some cases, it makes sense to create separate Worker Groups, each close to their data. In other cases, multiple Groups might be necessary for your architecture based on other needs. Here are a few examples, which might not be directly applicable to your scenario, but which should help you realize when multiple Groups might benefit your deployment:

- Keeping Workers close to their data to optimize syslog over UDP.
- Controlling egress costs, by reducing and compressing data before it leaves your cloud provider's region.
- Separating Push Sources versus Pull/Collector Sources.
- Geographical restrictions on data location, such as GDPR.
- Reducing or increasing the number of Workers connecting with a Source or a Destination.
- Centralize certain processing that is dependent on external systems, such as Redis lookups.
- Reducing the frequency of Stream config deployments, in cases where a deployment might disrupt data flow - such as triggering rebalancing among Kafka-based Sources.

Conclusion

As you can see by now, our "it depends" subtitle applies. But we hope that reading this guide has familiarized you with key aspects of what "it" depends on. There is no single "best practice" for sizing, because your own best architecture will depend on considering all options, including your risk-versus-cost balance, to define your own solution.

Equipped with the contents of this guide, you should be able to make better decisions about the potential risks and mitigations appropriate for your environment. After initial deployment, you can monitor utilization trends in the different resources we covered. As your needs evolve, so can your architecture. You'll make assumptions, and when the potential for failure is beyond your tolerance, you can adjust accordingly.

Some final considerations: Is it worth stress-testing your environment, or a downscaled version of it? What's the likelihood of everything going wrong at the same time - your throughput volume spiking to the highest acceptable level, while your Destinations are unavailable and your Persistent Queues are filling up? It depends.

To firm up your own architecture, consult Cribl support, Community Slack, or the Cribl Solutions Engineering or Professional Services experts dedicated to your deployment.

2.4.4. Ports

Cribl Stream requires certain ports to be open, and additional ports are needed if you intend to use specific integrations or options to work.

Leader

In a Distributed deployment, the following ports must be open on the Leader Node. Ensure that the Leader is reachable on those ports from **all** Workers.

In High Availability Leader proxy mode, UI/API traffic on port `9000` can be routed to either Leader; the standby transparently proxies requests to the elected primary Leader. For details, see Active-Active (Proxy) Mode for HA Leaders.

Default Port	Protocol	Purpose	Direction
`9000`	HTTP/S	Cribl Stream UI.	In
`9000`	HTTP/S	Bootstrapping Worker Nodes from Leader (on-prem).	In
`443`	HTTP/S	Bootstrapping Worker Nodes from Leader (Cribl.Cloud).	In
`4200`	TCP	Heartbeat/Metrics/Leader requests/notifications to clients (for example: live captures, teleporting, status updates, config bundle notifications, and so on).	In
`4200`	HTTP/S	Software upgrade (via path, not CDN).	In

If you want to use proxy for communication between Leader and Worker Nodes, use SOCKS proxy instead of HTTP/S. You need to use SOCKS proxy, because HTTP/S proxies typically don't support raw TCP sockets that Leader-Worker communication uses.

Outpost

The following port is used by Cribl Outpost.

Default Port	Protocol	Purpose	Direction
`9000`	TCP	Cribl Outpost UI. Login is disabled, but the Health endpoint is available.	In
`4200`	TCP	Communication between the Outpost and the Leader.	Out

Workers

The following ports are used by Worker Nodes.

Default Port	Protocol	Purpose	Direction
`9000`	TCP	Cribl Stream UI.	In
`9000`	HTTP/S	Communication with the Leader for bootstrapping (on-prem).	Out
`443`	HTTP/S	Communication with the Leader for bootstrapping (hybrid deployment), and with `https://cdn.cribl.io` to download configurations from CDN.	Out
`4200`	TCP	Heartbeat/Metrics/Leader requests/notifications to clients (for example: live captures, teleporting, status updates, config bundle notifications, and so on).	Out
`4200`	HTTP/S	Config bundle downloads from the Leader.	Out

Other Ports

This section lists port allocations for specific transport protocols and other special purposes.

Common Ports

Default Port	Protocol	Purpose	Direction
53	UDP	DNS lookups.	Out
389	TCP	LDAP Auth (non-TLS).	Out
`443`	HTTP/S	OIDC Auth (TLS); and Cribl Lake Destination on hybrid Worker Groups that you manage.	Out

Default Port	Protocol	Purpose	Direction
636	TCP	LDAP Auth (TLS).	Out
9002	TCP	Browser access to the identity server when using Personal Identity Verification (PIV) authentication.	In

Integrations and Apps

Integrations with specific services, via Sources and Destinations or apps, might require opening dedicated ports on Worker Nodes.

The defaults are listed below. When configuring most Sources or Destinations, you can choose a different port. However, on hybrid Worker Groups that you manage, the Cribl Lake Destination is hard-coded to send outbound HTTP/S traffic through port `443`.

Default Port	Protocol	Purpose	Direction
162	UDP	SNMP Trap collection (non-TLS). The preconfigured SNMP Trap Source listens on port `9162`.	In
162	UDP	SNMP Trap Destination (non-TLS).	Out
443	HTTP/S	Collection from and output to multiple HTTPS-based Sources and Destinations.	In/Out
4317	TCP	Collection from OpenTelemetry.	In
5986	HTTP/S	Windows Event Forwarder Source.	In
8081	TCP	Kafka Schema Registry.	Out
8088	TCP	Splunk HEC input and output.	In/Out
8089	TCP	Splunk Search.	In
8125	TCP/UDP	Output to StatsD, StatsD Extended, and Graphite (non-TLS).	Out
9090	TCP	Collection/discovery from Prometheus Scraper.	Out
9092	TCP	Collection from Confluent Cloud or Kafka, used when no port is provided.	Out
9092	TCP	Output to Confluent Cloud or Kafka, used when no port is provided.	Out

Default Port	Protocol	Purpose	Direction
`9093`	TCP	Output to Azure Event Hubs.	Out
`9200`	HTTP/S	Elasticsearch API Source.	In
`9514`	TCP/UDP	Syslog Source.	In
`9997`	TCP	Splunk TCP Source.	Out
`10000`		Splunk to Cribl Stream data port (Cribl App for Splunk).	In/Out
`10060`	TCP	TCP (Raw) data.	In
`10070`	TCP	TCP JSON data.	In
`10080`	TCP	Collection from HTTP JSON Sources.	In
`10200`	HTTP/S	Cribl HTTP Destination.	In
`10300`	TCP	Cribl TCP Destination.	In
`10420`		| `criblstream` Splunk search command to Cribl Stream (Cribl App for Splunk).	In/Out

Cribl.Cloud Ports

Cribl.Cloud provides a set of ports linked to Sources enabled by default for your Workspace. To view them:

1. From your Cribl.Cloud Organization's top bar, select **Products**.
2. Then from the sidebar, select **Cribl** > **Workspace**, and then **Data Sources**.

Additionally, Cribl.Cloud makes the `20000 - 20010` port range available for configuring other Sources.

Available Ports	Protocol	Purpose	Direction
`20000 - 20010`	TCP	Additional Sources in Cribl.Cloud.	In

No other ports can be opened for Cribl-managed Worker Groups in Cribl.Cloud.

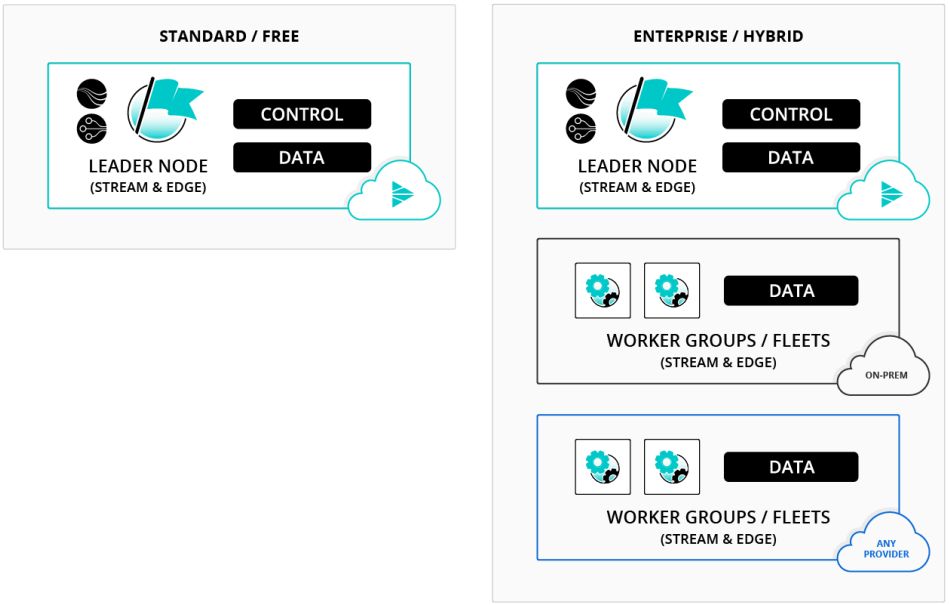
Cribl Copilot Port

2.5. About Cribl.Cloud

In a Cribl.Cloud deployment, Cribl assumes responsibility for hosting and managing your Cribl Stream infrastructure.

Cribl.Cloud comes in both free and paid versions. A free Cribl.Cloud account allows up to 1 TB/day of data throughput (data ingress + egress), but does not have the option to have multiple Worker Groups. By upgrading to a Cribl.Cloud Enterprise plan, you can implement a hybrid deployment of any complexity.

In hybrid deployments, the Leader resides in Cribl.Cloud, and manages Stream Workers and Edge Nodes regardless of their location. Your data processing configuration can consist of any combination of Workers and Edge Nodes: Cribl-managed, in private cloud instances that you manage, or in your data centers.



Standard/free versus Enterprise/hybrid deployment

For a comparison of features an capabilities of Cribl.Cloud vs an on-prem deployment, see Cribl.Cloud vs. Self-Hosted.

For an overview of additional features available on Enterprise plans, see Pricing.

If you're new to Cribl Stream, please see Cribl Stream Basic Concepts and the Getting Started Guide for help onboarding. Cribl.Cloud always runs in Distributed mode - for details about this, see Simplified Distributed Architecture.

Cloud Pricing

2.5.1. Register Cribl.Cloud Organization

To get started with Cribl.Cloud, sign up on the Cribl.Cloud portal, to create your Cribl.Cloud Organization.

Your Organization will have a dedicated portal with its own network and access boundaries, isolating your Cribl resources from other users. For more information, see Cribl.Cloud Data Isolation.

By default, your instance of Cribl.Cloud will be on the **free** Cribl.Cloud plan, which includes certain throughput and administration limits. When you need more capacity or features, you can upgrade to a paid or Enterprise plan. To upgrade, select **Go Enterprise** in your Cribl.Cloud portal.

You can find the Cribl.Cloud Suite on the AWS Marketplace. When you're ready to upgrade, you can use your Enterprise Discount Program (EDP) credits to run Cribl products, billed directly through your AWS account. There is no need for a separate procurement process.

Note that each user can register only one active Organization on the free tier. For details, see notes on Member Permissions.

To sign up for a free Cribl.Cloud instance and register your Organization:

1. Go to https://cribl.cloud/signup/ and create an account using your work email address.
2. Follow the instructions to set up your Organization, select its region for your Cribl.Cloud deployment, and to provide some information about your goals in using Cribl.

When you complete the registration steps, you will see your new Cribl.Cloud deployment home page. The home page suggests possible actions you can take based on the answers you provided during registration.

From there, consider completing the Getting Started Guide tutorial for a walk-through of the main features and common use cases for Cribl Stream.

Cribl.Cloud Organization Regions

The Organization region defines where the Leader Node resides.

Once you select a region when registering your Organization, you cannot change it. You can, however, create Cribl-managed Stream Worker Groups in different AWS and Azure regions.

In Cribl Search, the Organization region is the physical location where searches are executed. In Cribl Lake, the region defines the location where data is stored.

Troubleshooting Resources

If you get stuck during registration, Cribl University has a troubleshooting guide on How to Log Into Cribl.Cloud walks you through the registration and login workflow. To follow the direct course link, first log into your Cribl University account. Once logged in, check out other useful short courses:

- Troubleshooting Criblets
- Advanced Troubleshooting

2.5.2. Manage Cribl.Cloud Organization

The Cribl.Cloud portal is the place where you get an overview of and manage your organization, configure network settings, and control user access and billing information.

Switch Organizations

When you own or are a Member of multiple Cribl.Cloud Organizations, you can switch between the Organizations:

1. In the top bar, hover over the name of your Organization.
2. In the drop-down, select the Organization you want to switch to.

Alternatively, in the drop-down you can select **Manage Organizations**. You will move to an Organization landing page, displayed to an Organization's Owners and Admins, where you can view details of all your Organizations, switch to any of them, add a new one or delete one, if you have proper permissions.

Configure Organization Details

To customize your Organization information, in the sidebar, select **Organization**.

You can modify the following information:

Field name	Description
Organization Name	A "friendly" name for your Organization. Upon signing in, Members will see this alias above the Organization ID on the Select Organization page.
Description	Add further details about your Organization.
Opt in to beta features	If displayed, this check box enables access to new options that Cribl has not yet made generally available. As with all beta features, expect some instability in exchange for advancing to the cutting edge of your Cribl.Cloud.

Select **Save** to immediately apply your changes.

You can check additional information about your Organization, including the AWS Cloud region in which it's located, the software version, and license, by selecting **Organization Details** at the bottom of the sidebar.

Ingress Ports

Your Cribl.Cloud Organization's preconfigured and user-configurable ports for inbound data are set at the Workspace level. For details, see Cribl.Cloud Ports.

Get Support

In the top bar, select the **Support** icon to access a set of links with additional resources about Cribl Stream, including:

- Sandboxes (free, interactive tutorials on fully hosted integrations)
- Cribl University
- Concept/demo videos

Refer to Cribl Status for information about Cribl.Cloud availability, including current and historical uptime data, and incidents, split by component and AWS Region.

You can also subscribe there to incident notifications via email, text, Slack, or RSS.

Manage Personal Settings

Use the **Account** menu in the top right of the Cribl.Cloud portal to configure your personal settings.

Select **My profile** to update your personal information. Here, you can also reset the configured biometric authentication. Do this, if you want to re-configure the authentication, or if you don't want to be prompted to log in through fingerprint or face recognition.

2.5.3. Cribl.Cloud Enterprise

With a Cribl.Cloud Enterprise plan, you have these options and workflow flexibility:

- Configuring and managing multiple Worker Groups and Fleets.
- Notifications within Cribl apps, to PagerDuty, and/or to other services via webhook.
- Fine-grained, role-based control of Member Permissions on your Cribl.Cloud Organization and on individual product resources.
- Single sign-on/SSO authentication from external identity providers.
- The hybrid deployment option, described just below.

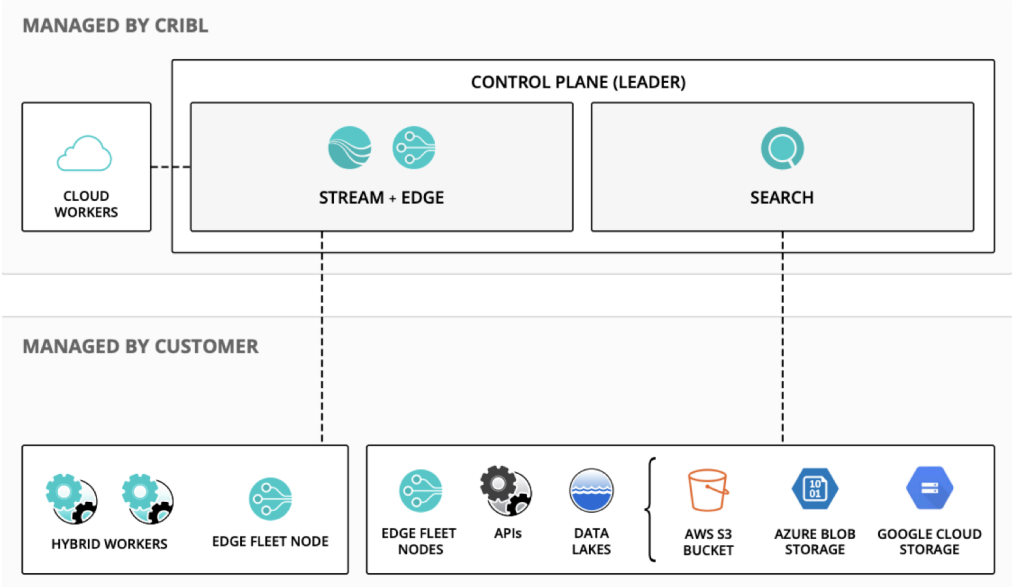
With an Enterprise Plan, the Leader resides in Cribl.Cloud, and controls a flexible mix of Cribl-managed and/or customer-managed Worker Groups. Cribl manages the Leader's high availability on your behalf.

For other Enterprise features, see our Pricing page.

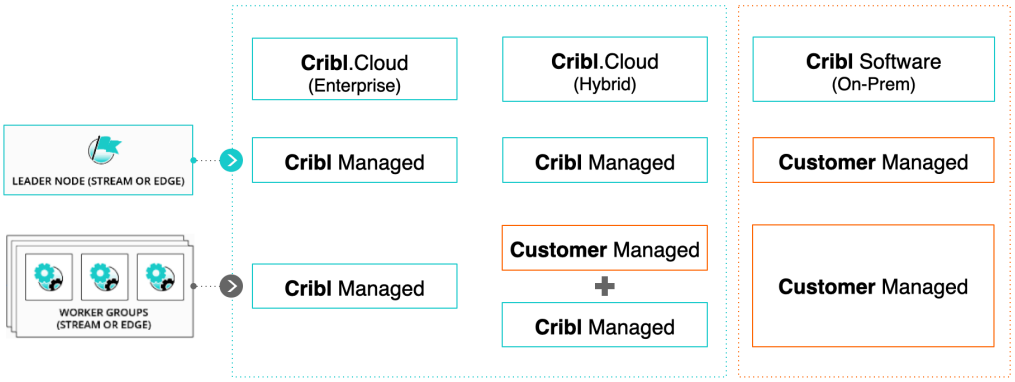
Hybrid Deployment

The diagrams below show the comparative flexibility of a hybrid Cribl.Cloud deployment. The Leader (control plane) resides in Cribl.Cloud, while the Workers that process the data can be in any combination of the following environments:

- In Cribl.Cloud, managed by Cribl.
- In public or private cloud instances that you manage.
- On-prem in your data centers.



Enterprise hybrid deployment, with control plane and Cribl-managed Workers in Cribl.Cloud



Enterprise hybrid deployment, with only control plane in Cribl.Cloud

As the footprint of your operations grows or changes, this flexibility makes it easy to reconfigure Cribl Stream in tandem. You can rapidly expand Cribl Stream observability into new cloud regions - and replace monitored hardware data centers with cloud instances - all while maintaining one centralized point of control.

You can also add Workers or Edge Nodes, and reassign them to different Worker Groups, by easily auto-generating Stream or Edge command-line scripts within Cribl Stream's UI.

Hybrid Requirements

A hybrid deployment imposes these configuration requirements:

- Hybrid Workers (meaning, Workers that you deploy on-prem, or in cloud instances that you yourself manage) must be assigned to a different Worker Group than the Cribl-managed `default` Group - which can contain its own Worker Nodes.
- All Worker Nodes' hosts must allow outbound communication to the Cribl.Cloud Leader's port 4200 at `https://main-<Organization-name>.cribl.cloud:4200`, to enable configuration and workload management by the Leader.
- On all Worker Nodes' hosts, firewalls must allow outbound communication on port 443 to the Leader and to `https://cdn.cribl.io`. This port is also used to bootstrap hybrid Workers from the Leader.
- All Worker Nodes require connectivity to `https://cdn.cribl.io/telemetry/`. For details on testing this connectivity, on the metadata transmitted to Cribl, and on how we use that data, see Telemetry Data.
- If this traffic must go through a proxy, see System Proxy Configuration for configuration details.
- To verify your Leader's region and public URL, open the Access Details modal.

Note that you are responsible for data encryption and other security measures on Worker Node instances that you manage.

Adding (Bootstrapping) Workers

To add Workers to your hybrid Cribl.Cloud deployment, Cribl recommends that you use the script outlined in Bootstrap Workers from Leader. Hosts for the new Workers must open the same ports (4200 and 443) listed in Hybrid Requirements.

You have three options for generating the script, outlined in these subsections of the Bootstrap topic linked above:

- Auto-generate it from the Leader's UI.
- Make a `GET` API request to the Leader.
- curl the same API request.

In Cribl Edge, you access all these bootstrap options via the Manage Edge Nodes page's **Add/Update Edge Node** control.

Hybrid Cribl HTTP/TCP Configuration

If you use the Cribl HTTP Destination and Source pair, or the Cribl TCP Destination and Source pair, to relay data between Worker Nodes connected to the same Leader, configuring hybrid Workers demands particular care.

The Worker Nodes that host each pair's Destination and Source must specify exactly the same Leader Address. Otherwise, token verification will fail - breaking the connection, and preventing data flow. In hybrid Cribl.Cloud deployments, the Leader's Address format is `main-<your-Org-ID>.cribl.cloud`. When configuring a hybrid Worker, use that format in the **Address** field.

To configure hybrid Workers:

1. Log directly into their UI, then select **Settings** > **Global** > **Distributed Settings**. Make sure the **Mode** is set to **Stream: Managed Worker (managed by Leader)** or **Edge: Managed Edge (managed by Leader)** (which might require a restart).
2. Then select **Leader Settings**, and ensure a consistent entry in the **Address** field.

2.5.4. Manage Cribl.Cloud Worker Groups

Using Cribl.Cloud's in-app controls, you can provision Cribl-managed Worker Nodes in Cribl.Cloud to match the throughput capacity you need. With certain plan/license tiers, you have the option to add and configure multiple Cribl-managed and/or customer-managed (hybrid) Groups.

Cloud Worker Group Providers

> Cribl.Cloud Government Worker Groups are supported exclusively on AWS US East/West Regions. Microsoft Azure is not supported.

Cribl-managed Worker Groups in Cribl.Cloud can be based in one of two providers: AWS or Azure.

Select the provider based on, for example, where your primary Sources and Destinations are located. This will help you reduce transit costs and reduce latency.

The supported regions are:

AWS Region name	AWS Region	Azure Region Name	Azure Region
Asia Pacific (Singapore)	`ap-southeast-1`	Southeast Asia	
Australia East	`ap-southeast-2`	Australia East	`australiaeast`
Canada Central	`ca-central-1`	Canada Central	`canadacentral`
Germany West Central	`eu-central-1`	Germany West Central	`germanywestcentral`
Europe (Zurich)	`eu-central-2`	Switzerland North	`switzerlandnorth`
UK South	`eu-west-2`	UK South	`uksouth`
US East (N. Virginia)	`us-east-1`	East US	`eastus`
US East (Ohio)	`us-east-2`	East US 2	`eastus2`
US West (Oregon)	`us-west-2`	West US 2	`westus2`

Config Bundles in Cribl.Cloud

In Cribl.Cloud, configuration bundle storage and delivery are fully automated for Cribl-managed Worker Groups. Workers prioritize downloading new bundles from a Cribl-managed Amazon S3 bucket over HTTPS

(port `443`), with an automatic fallback to the Leader over HTTPS (port `4200`) if the S3 endpoint is unreachable. Hybrid Workers are also bound to this same cloud-managed design; since the distribution logic resides on the Cribl-managed Leader and no settings are exposed to change it, hybrid Workers must follow the automated delivery path. You do not need to manage S3 buckets or remote URLs; simply ensure your environment allows outbound access to the required S3 and Cribl.Cloud endpoints. For details, see Manage Config Bundles.

Network Access for Config Bundles

To prevent deployment delay and `Download error` logs, you must configure your network security policies (firewalls/ACLs) to allow outbound access to the required endpoints.

For Worker Nodes running in any environment, including other cloud providers like Azure or Google Cloud Platform (GCP), you must permit access using Fully Qualified Domain Names (FQDNs) rather than static IP addresses, as S3 and Cribl.Cloud IPs are dynamic. Make sure the following endpoints are reachable:

- **S3 Storage**: The Cribl-managed S3 bucket FQDN (for example, `*.s3.<region>.amazonaws.com`).
- **Cribl.Cloud Leader**: Your specific Cribl.Cloud Leader address (for example, `main-xxxx.cribl.cloud`).

While Worker Nodes will automatically fall back to the Leader if S3 is blocked, permitting the S3 FQDN is required for optimal performance and clean logs.

Create a Cloud Worker Group

All Cribl Organizations start out with one Worker Group. With an Enterprise plan, you can create additional Cloud Worker Groups:

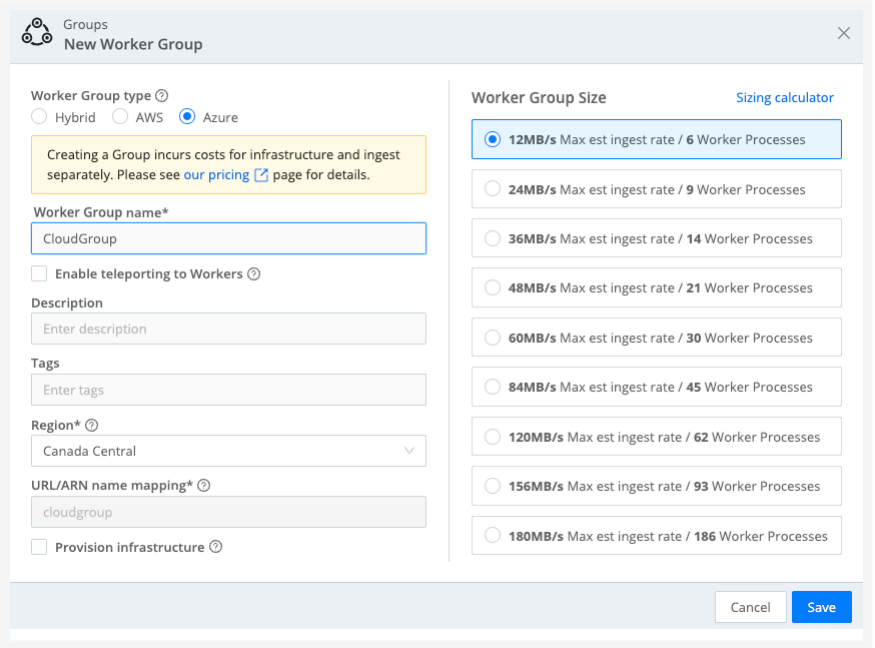
1. On the top bar, select **Products**, and then select **Stream**.
2. Select **Add Worker Group**.
3. In the **New Worker Group** modal, in **Worker Group type**, select your preferred provider: `AWS` or `Azure`.

   You can create up to 10 Cloud Worker Groups per Workspace. If you need to increase this limit, contact Cribl Support.

4. Select the Azure or AWS **Region** where you want to locate the Worker Group.
5. Enter a **Worker Group name**. Azure-based Worker Group names must not contain Microsoft reserved words, even as substrings. For a complete list of reserved words, see Resolve reserved resource name errors.
6. Select **Enable teleporting to Workers** if you want to enable authenticated access to the Worker Node's UI from the Leader.
7. Select **Provision infrastructure** if you want Cribl.Cloud to automatically create Worker Nodes.

8. The **URL/ARN name mapping** is an auto-generated, unique identifier for your Cloud Worker Groups. It follows stricter naming rules than the Group name.
9. Optionally, enter a **Description**.
10. You can also add **Tags** to help organize Worker Groups into logical categories. Then, you can search by tags on the **View all Worker Groups** interface.
11. Use the **Sizing calculator** to determine the recommended **Worker Group Size**. For details, see Using the Sizing Calculator.
12. Select the **Worker Group Size** based on the recommendation. This setting controls the number of Worker Nodes that will be provisioned, assuming a 1:2 ingest:egress ratio.
13. Confirm with **Save**.

For general guidance about capacity required for your deployment, see our Planning and Sizing topics.



Adding and sizing a new Cloud Worker Group

Azure Cloud Workers and Resources in the Same Region

If you are using an Azure-based Cribl-managed Worker Group located in the same Azure region as another resource (for example, an Azure-based Source or Destination), the two resources may not be able to communicate. This may happen if network rules on the Azure resource are enabled by setting **Public network access** to **Enabled from selected virtual networks and IP addresses** on Azure portal's **Storage Account** > **Security + networking** page.

This is due to how intra-region cloud routing occurs in Azure over the CSP backbone network leveraging resource private IPs.

You can resolve this by moving the Worker Group to another Azure region. However, this will incur additional egress fees.

Use the Sizing Calculator

The Sizing calculator is an in-built feature within Cribl Stream Cloud that aids users in determining the appropriate size for their Worker Groups, either during creation or resizing processes. This tool helps you make informed decisions based on your specific data volume and processing needs.

When creating a new Worker Group, the **Sizing calculator** section is within the Worker Group configuration modal. When resizing an existing Worker Group, navigate to the **Worker Group Settings** > **Worker Group Configuration** section for the desired Worker Group.

To generate the recommended sizing:

1. Use the slider to enter the anticipated **Daily inbound data volume** in `TB`.
2. Use the slider to enter the anticipated **Daily outbound data volume** in `TB`.
3. Select your **Processing load** based on data transformation complexity. **Low** for simple tasks like Masking or Renaming, **Medium** for functions like Rollups or Aggregations, and reserve **High** for resource-intensive operations like Regex extraction or custom code.
4. Select **Recommend Sizing**. The tool will highlight a recommended **Worker Group Size** tier. You can choose a different tier from the available options if your needs differ slightly from the recommendation.

Provision and Manage Infrastructure

If you select **Provision Infrastructure**, the infrastructure will be provisioned about 30 minutes later. You will receive an in-portal notification confirming the change.

You can change Worker Group configuration later, including modifying the **Worker Group Size**, which may change the number of provisioned Worker Nodes.

Control Costs

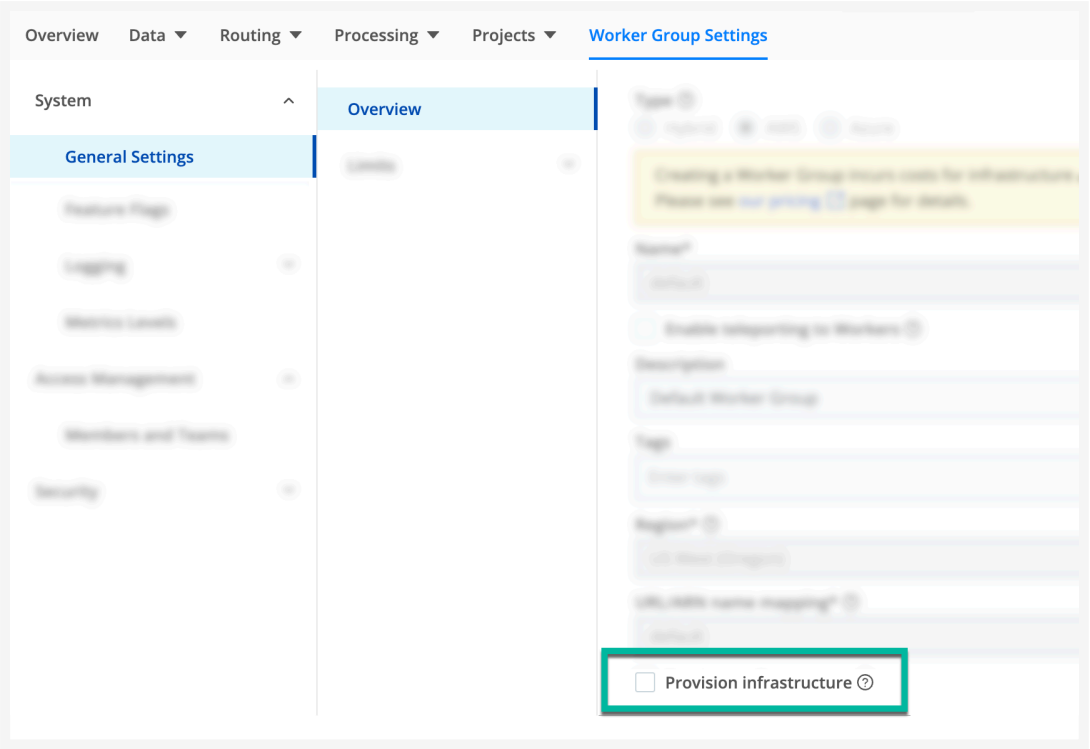
Cribl will draw down your Cribl.Cloud credits to cover the base cost of maintaining the infrastructure, as well as your normal cost for ingested data. When you create a Workspace, Cribl provisions Workers for you in the `default` Worker Group, but you can deprovision the Worker Group to control infrastructure costs.

You can do so either when creating the Worker Group, or at any later time when you are ingesting no data. With this setting, the Worker Group will go dormant, and your base infrastructure cost for this Worker Group will be `0`. In this state, the Worker Group will contain no Worker Nodes and will ingest no data.

Deprovision a Worker Group

To deprovision the Worker Group:

1. Select your Worker Group from the **Worker Groups** menu.
2. Select **Worker Group Settings**.
3. On the **General Settings** tab, select **Overview**.
4. Uncheck the **Provision infrastructure** option.
5. Select **Save** to confirm, and agree to the confirmation modal with **Deprovision**.



Deprovision a Worker Group from Worker Group Settings

When you deprovision a Worker Group, Cribl Stream retains the configuration so you can reprovision it if needed. However, any persistent queue data on the Worker Nodes will be permanently deleted.

2.5.5. Learn About Connected Environments

If you have one or more on-prem Cribl Stream environments and Cribl.Cloud Organization(s), you can configure **Connected Environments**.

Connected Environments enables you to:

- **Take advantage of Cribl.Cloud.** You can use Connected Environments to connect and on-prem Leader to Cribl.Cloud in order to use Cribl.Cloud infrastructure management capabilities, the credit-based billing model, as well as Cribl Lake and Cribl Search solutions. For example, you have on-prem environments but you want to use the credit billing model, and you want to send Cribl metric data to Cribl Lake/Search. From there, you can build dashboards to monitor Cribl throughput and performance.
- **Send data from an on-prem deployment to a Cribl.Cloud deployment.** If you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from on-prem Worker Nodes to Cribl.Cloud Worker Nodes and transfer data without incurring any billing costs. See Send Data from On-Prem to Cribl.Cloud for implementation details.

For full access to Connected Environments, upgrade Cribl Stream to 4.8.2 or newer.

Accomplish this by creating a connection between the on-prem Leader and the Cribl.Cloud Leader. Then, use the **Connections** page in your Cribl.Cloud account to manage credit consumption for all of your on-prem environments from one Cribl.Cloud interface. This feature is called Universal Subscription.

This section provides instructions for connecting on-prem Leaders to Cribl.Cloud.

Before You Begin Connecting On-Prem Leaders

There are a few tasks you must complete before you can start connecting your on-prem Leader to Cribl.Cloud:

1. Create a Cribl.Cloud account.
   To set up connected environments, you must have an Enterprise Cribl.Cloud account.
2. Decide how you want to connect to the Leader.
   There are two methods you can use to connect an on-prem Leader to a Cribl.Cloud Leader:
   - From the Cribl.Cloud account in **Connections** > **Connected Environments**.
   - From the on-prem Leader in **Distributed Settings** > **Cloud Connection**.
3. Purchase a Cribl.Cloud Enterprise Plan with credits.

Credits serve as the virtual transaction currency in Cribl's product suite. Each Cribl product has a predefined usage rate, and as you use the products, credits are deducted from your initial pool.

Configuration Considerations

Take these considerations into account for your on-prem Leader, allowlists, security groups, and Workspaces:

- You must be on Cribl Stream version 4.8.2 or newer.
- **Ports**: Make sure that port `4200` is open on the Leader, with TCP protocol. See Leader Ports for more information.
- **Proxy**: Connected Environments doesn't support proxied connections.
- **TLS**: The connection between the Cribl.Cloud Leader and the on-prem Leader must be encrypted using TLS. When you configure the `CRIBL_CLOUD_WORKSPACE_URL` environment variable, it will start with `tls://`, which enforces TLS.
- **Static IPs**: For IP address allowlist or security group rules, you can find the static IPs of your Cribl.Cloud Leader in **Workspace** > **Access** > **Leader NLB IPs**. You can also use the `nslookup` command to obtain static IPs.
- **Workspaces**: Workspaces are separate infrastructure. For allowlists and security groups, ensure that you add each IP address individually as each Workspace has a unique IP.

2.5.5.1. How To Connect On-Prem Leaders To Cribl.Cloud

To connect existing on-prem Cribl Stream Distributed deployments to Cribl.Cloud, select an on-prem Leader that is running Cribl Stream 4.8.2 or newer, then follow the steps.

If you have multiple environments, such as staging and production environments when using GitOps, you must connect each environment individually.

1. On the top bar, select **Products**, and then select **Cribl**.
2. In the sidebar, select **Workspace**, and then select **Connections**.
3. Select **Configure New Environment** and fill out the following fields:

Field	Description	Example
Environment name	A name for your environment. This name will display on the **Connected Environments** page. You can search for and filter environments by name.	Goat Farm East Production
Workspace hostname/ID	Hostname of your Cribl.Cloud Workspace. This field is read-only.	`example-goat-farm-prod.cribl.cloud`
Workspace port number	Port number that the Workspace Leader uses to communicate with the on-prem Leader.	`4200`
Workspace token	The auth token of the Cribl.Cloud Leader. This field is read-only and is automatically populated.	`a12bcDEFgHiJKLmnOpQRstuvw3x4z56`

Script type	The format of the script to connect to the on-prem Leader:
	◦ Environment variable
		Environment variable:
		`CRIBL_DEPLOYMENT="Goat Farm West Product`
		`CRIBL_CLOUD_WORKSPACE_URL="tls://a12bcDE`
		CLI:

Field	Description	Example

	◦ CLI	`./cribl cloud-workspace`
		`-d “Goat Farm West Production”`
		`-H example-goat-farm-prod.cribl.cloud`
		`-p 4200`
		`-u a12bcDEFgHiJKLmnOpQRstuvw3x4z56`

Use a Command-Line Script

If you’re using the CLI option, follow these steps, in order:

1. Select **Copy Script**.
2. Open a command-line window in the Leader environment that you want to connect to Cribl.Cloud.
3. Navigate to the `$CRIBL_HOME/bin` subdirectory.
4. Paste the script results onto the command line, and run the script.

Use an Environment Variable

The **Environment Variable** script populates with the details you enter when you’re configuring a new environment. For example:

```
CRIBL_DEPLOYMENT="On-prem Stream"
CRIBL_CLOUD_WORKSPACE_URL="tls://a12bcDEFGHIjKLMnopPqr123@cribl-cloud-account-0a1bcde
```

Copy

If you’re using the environment variable, copy its output, and set it when - or before - you run `cribl start`. How you set this variable depends on your deployment.

- For example, if your Cribl Stream deployment is running in a container, set the environment variable in the container via run call or configuration.
- If you’re running your own server, you can set environment variables for the `cribl` user, or you can declare the variable along with the `start` command. For details, see our CLI Reference.

High Availability Leader Configuration

Before you configure a connected environment with a High Availability (HA) Leader, ensure you are comfortable setting up and managing Leader HA. See [Leader High Availability/Failover](deploy-add-second-leader for more information.

When you set up the primary Leader to connect to Cribl.Cloud in an HA environment, set the `CRIBL_DEPLOYMENT` and `CRIBL_CLOUD_WORKSPACE_URL` environment variables. You can do this in the Cribl.Cloud UI or using CLI.

For example:

```
Environment=CRIBL_DEPLOYMENT=aws_leader_prod
Environment=CRIBL_CLOUD_WORKSPACE_URL=tls://large-goat-farm.cribl.cloud:4200
```

Copy

The failover Leader will inherit the configuration from the filesystem changes made by the primary Leader.

Enable the On-Prem Leader Connection to Cribl.Cloud

You can connect a Leader Node to an existing Enterprise Cribl.Cloud Organization.

1. In the sidebar, select **Settings**.
2. On the **Global** tab, select **Distributed Settings**.
3. Select **Cloud Connection**.
4. Toggle **Join Cribl.Cloud Workspace** on.
5. Fill in the required fields:

Field	Description
Connection URL	The **Cribl.Cloud URL** to connect to this Leader. 1. In your Cribl.Cloud Organization, locate the URL field value on the sidebar **Workspace** > **Access** tab. 2. Copy the URL and paste it into the **Connection URL** field, using the format: `main-random-goat-a12bcdef.cribl.cloud`.
Connection port	The port for the Cribl.Cloud Leader. Default is `4200`.
Connection auth token	The authentication token for the Cribl.Cloud Leader. You'll need to get this from the Cribl.Cloud environment: 1. Navigate to **Workspace** > **Connections** and select **Configure New Environment**. 2. Copy the value in the **Workspace token** field. 3. In the on-prem environment, paste the Workspace token from Cribl.Cloud.

6. Select **Save**.

7. In the Cribl Subscription Services Agreement modal, select **Agree** to accept the agreement.
8. In the confirmation modal, select **Yes** to save changes and restart.

Once you connect an on-prem Leader to a Cribl.Cloud Organization, the license type changes to a Cribl.Cloud billing account. This means the **Licensing** page will not be accessible. To return to using an on-prem license, toggle **Join Cribl.Cloud Workspace** off, select **Save**, and select **Yes** in the confirmation modal to save changes and restart.

2.5.5.2. Simplify Billing With Universal Subscription

This feature requires an Enterprise Plan.

If you have multiple instances of Cribl Stream on-prem, you're likely managing multiple licenses. Instead of letting license management run your life, you can take advantage of the Universal Subscription feature. Universal Subscription lets you manage an existing on-prem license with Cribl.Cloud credits.

To get started, connect your on-prem Leader to your Cribl.Cloud Leader to send billing data payloads. Once your environments are connected, you can use the FinOps Center to get a unified, all-product view of your Cribl Stream usage and spending.

This replaces an on-prem license, and gives you the flexibility to maintain your on-prem deployments without managing multiple licenses and tracking spend across environments.

An on-prem Cribl Stream environment consists of at least one Leader Node (the control plane) and one or more Worker Groups.

Connecting your on-prem Leader license to Cribl.Cloud billing allows you to:

- Monitor credit consumption for each of your on-prem Cribl Stream environments from one user interface.
- Create a billing- and usage-based connection between the Cribl Stream on-prem Leader and the Cribl.Cloud Leader, which receives metrics from the on-prem Leader. For information about the data payloads, visit Data Payloads for Connected Environments.

Configure Connected Environments

For the setup procedure, go to How to Connect On-Prem Leaders to Cribl.Cloud, then come back here to explore how to view credit and usage data for your on-prem Connected Environments.

Full access to Connected Environments requires you to upgrade Cribl Stream to 4.8.2 or newer.

Explore Credit Consumption for Connected Environments

Once you set up a Connected Environment, you can view the incoming usage data and credit consumption for your on-prem and Cloud deployments of Cribl Stream. You must be an Organization Owner to view this data.

For more information on Cribl.Cloud billing, go to the FinOps Center page.

View Usage Data

1. In the sidebar, select **Organization**, then **FinOps Center**.
2. Select **Usage Overview**.
3. Go to **Per Product Consumption** and expand either Cribl Edge or Cribl Stream.
4. Find **Connected < Product > GBs In**.

View Credit Consumption

1. In the sidebar, select **Organization**, then **FinOps Center**.
2. On the **FinOps Center** submenu, select **Invoices**.
3. Select a month to expand the cost items. Select **Stream** or **Edge**.
4. View **Connected < Product > GBs In** usage and cost.

Here, you will see each on-prem environment that is connected to this Cribl.Cloud Organization, and the credits each environment is consuming.

Limitations of the Universal Subscription Feature

Connected Environments has some usage limitations:

- The Connected Environments feature is available only on Enterprise Cribl.Cloud plans. For Enterprise plan details, see Pricing.
- Single-instance deployments of Cribl Stream aren't eligible for a Universal Subscription.
- The Distributed deployment on-prem Leader must be running Cribl Stream 4.8.2 or newer.

2.5.5.3. Data Payloads For Connected Environments

In a Connected Environment setup, on-premises Leaders connect to a Cribl.Cloud Leader in the same way that Worker Nodes connect to a Leader. The Cribl.Cloud Leader sends down a license, and the on-premises Leader sends up a heartbeat that includes telemetry, which encompasses metrics and other metadata.

Why Is the Telemetry Important?

The telemetry sent from an on-premises Leader to a Cribl.Cloud Leader is encapsulated in an event. This event serves as a heartbeat, confirming that the connected environment is active and functioning properly. It includes key metadata such as the system's status, timestamp, configuration details, and metrics. Cribl.Cloud uses this data to:

- Monitor connectivity between the on-premises Leader and the cloud.
- Track system health by evaluating fields like status, edgeNodes, and metrics.
- Ensure compliance with licensing by verifying the config.featuresRev field.
- Provide insights into data flow, such as input and output metrics.

How Often Is the Heartbeat Sent?

The heartbeat interval is determined by the **Task heartbeat period** field, which specifies the number of seconds between each event. In the provided example below, the heartbeat is sent every `10` seconds. This interval is configurable, meaning administrators can adjust it to optimize performance based on their needs. For details, see Task Manifest and Buffering Limits in Worker Group Settings.

What Happens if the Heartbeat Is Missed?

If a heartbeat event is not received within the expected interval, the following actions may occur:

- **Retries** - The on-premises Leader may attempt to resend the heartbeat after a short delay.
- **Alerting** - Cribl.Cloud can generate an alert if the heartbeat is missing for an extended period, indicating a potential issue.
- **Failure detection** - A prolonged absence of heartbeats may be interpreted as a disconnection or system failure, prompting further investigation or automated recovery steps.

On-Premises Leader to Cribl.Cloud Leader Payload

This section describes the structure of the event payload sent from an on-premises Leader to a Cribl.Cloud Leader in a Connected Environment.

Sample Event

Here is a sample event that Cribl.Cloud received from a connected environment.

```
{
  "__criblEventType": "event",
  "type": "info",
  "_time": 1737577024.567,
  "status": "healthy",
  "localTime": 1737577024,
  "config.hbPeriodSeconds": 10,
  "config.featuresRev": "YbK7TIGe4jp2rdujnFiJqq0OBV67r2FKGvrHr4C6I5g=",
  "edgeNodes": 50,
  "__socketAddr": 192.168.1.100:443,
  "__srcIpPort": 10.0.0.5:52314,
  "metrics": [
    [
      0,
      0,
      "total.in_bytes",
      233,
      1737578280038,
      1737578340038,
      [
        "ci",
        "tcp:in_tcp",
        "input",
        "tcp:in_tcp",
        "input_type",
        "tcp",
        "__worker_group",
        "default",
        "__dist_mode",
        "worker",
        "__worker_node",
        "46731aec-d220-4cb0-9971-0dd0a7907d54",
        "__product",
        "stream"
      ],
      null,
      "float32"
    ]
  ],
  "__raw": "{\"type\":\"info\",\"_time\":173757...}"
}
```

Copy

Event Payload Fields

The following table describes the important fields in the event payload.

Field	Usage	Example
__criblEventType	Type of event	event (always)
type	Type of event	info (always)
_time	Timestamp of the event in Unix time (seconds)	1737577024.567

2.6. On-Prem Deployment

Deployment guide to get you started with self-hosted Cribl Stream

---

In an on-prem (in other words, self-hosted) deployment, Cribl Stream runs on your own infrastructure.

There are at least **two** key factors that will determine the type of on-prem Cribl Stream deployment in your environment:

- Amount of Incoming Data: This is defined as the amount of data planned to be ingested per unit of time. E.g. How many MB/s or GB/day?
- Amount of Data Processing: This is defined as the amount of processing that will happen on incoming data. E.g., is most data passing through and just being routed? Or are there a lot of transformations, regex extractions, field encryptions? Is there a need for heavy re-serialization?

An alternative to self-hosting Cribl Stream is using Cribl.Cloud.

The available on-prem deployment types are:

- Single-instance deployment - When volume is low and/or amount of processing is light, you can get started with a single instance deployment.
- Distributed deployment - To accommodate increased load, we recommend scaling up and perhaps out with multiple instances.
- Splunk App deployment - If you have an existing Splunk Heavy Forwarder infrastructure that you want to use, you can deploy Cribl App for Splunk.
- Orchestrated deployment - You can deploy Cribl Stream Leader Nodes (or single instances) and Worker Nodes via Cribl's Helm charts.
- Docker deployment - You can deploy Cribl Stream instances using images from Cribl's public Docker Hub.

2.6.1. OS And System Requirements

For a successful Cribl Stream deployment, verify that your system meets the minimum hardware and software specifications outlined below.

All the quantities listed below are **minimum** requirements. Adjust them as needed based on your specific use case.

OS Requirements

You can install Cribl Stream on any common Linux distribution that fulfils the following criteria:

- 64-bit kernel >= 3.10
- glibc >= 2.17
- OpenSSL >= 3.0 (if FIPS mode is enabled)

Highly customized or rare Linux distributions might not be supported even when based on a supported kernel version.

Enforcing mode in SELinux is supported, but not required.

Tested Linux Platforms

Cribl tests the following Linux versions to confirm compatibility with the supported kernel. This list does not constitute a comprehensive record of all supported distributions and versions.

- Ubuntu 24.04
- RHEL 9, 9.3
- CentOS Stream 9
- SUSE Linux Enterprise Server 15.6
- Amazon Linux 2023

As of October 16, 2024, Cribl has deprecated support for the CentOS Linux 7 distribution, which reached its end of life on June 30, 2024.

System Requirements

The following specs are the minimum application requirements for Cribl Stream, either a Leader, a Worker Node, or a Single-instance deployment.

Given the Leader's critical role in coordinating Worker instances, we recommend deploying it on a stable and highly available infrastructure.

- 1 GHz (or faster), 64-bit CPU
- 4+ physical cores (beyond basic OS/VM requirements, 1 physical core is equivalent to 2 virtual/hyperthreaded CPUs (vCPUs).)
- 8 GB+ RAM (beyond basic OS/VM requirements)
- 5 GB free disk space.
  Cribl recommends reserving additional free disk space for systems taking advantage of Leader High Availability. See Disk Requirements for details.

See Sizing and Scaling for capacity planning details.

Cloud VM Recommendations

To fulfill the requirements using cloud-based virtual machines, see Recommended AWS, Azure, and GCP Instance Types.

Core Equivalency

We consider 1 physical core to be equivalent to:

- 2 virtual/hyperthreaded CPUs (vCPUs) on Intel/Xeon or AMD processors
- 1 (higher-throughput) vCPU on Graviton2/ARM64 processors

Memory Requirements

- Each Worker Process can consume memory up to its maximum heap size, plus Node.js overhead, and additional memory that scales with configuration (e.g., Destination type and number).
- Ensure your total host memory allocation covers the memory usage of all Worker Processes and OS requirements.
- Continuously monitor Node memory usage, especially after configuration changes.
- For additional details, see Estimating Number of Cores.

Since version 4.13, due to an upgrade of the Node.js version used by Cribl Stream, baseline memory usage may increase on hosts that are not memory-constrained, because the new Node.js version is

more conservative with memory release.

Browser Support

The Cribl Stream UI can run on Chrome, Firefox, Safari, and Microsoft Edge, on their five most-recent versions.

Additional Requirements

`git` must be available on the Leader Node. See details below.

Set Capabilities for Cribl Stream

If Cribl Stream needs to listen on low ports 1-1024, it will need privileged access. You can set Linux capabilities that grant Cribl Stream sufficient rights to perform specific privileged tasks, based on kernel privilege. To run Cribl Stream as non-root user, consider setting the following capabilities:

Set Linux Capabilities	Permissions
`CAP_NET_BIND_SERVICE`	Allows Cribl Stream to push Sources that bind to TCP/UDP port numbers below `1024`.
`CAP_DAC_READ_SEARCH`	Allows the `cribl` user to access the File Monitor Source's Manual mode feature. This capability bypasses the default Linux permissions for files and directories.
`CAP_SYS_PTRACE`	Allows the `cribl` user to scan open files for running processes, and to access the File Monitor Source's Auto-mode feature.

For details about setting these capabilities, see Persisting Overrides.

Set the `CRIBL_HOME` Environment Variable

The `CRIBL_HOME` env is available in the Cribl Stream application, but not on your terminal. If you want to use `$CRIBL_HOME`, you can:

- Assign it once, using the `export` command: `export CRIBL_HOME=/opt/cribl`
- Set it as a default, by adding it to your to your terminal profile file.

Version Control with `git`

Cribl Stream requires `git` (version 1.8.3.1 or higher) to be available locally on the host where the Leader Node will run. **Configuration changes must be committed to git before they're deployed.**

If you don't have `git` installed, see Git Documentation for details on how to get started.

The Leader Node uses `git` to:

- Manage configuration versions across Worker Groups.
- Provide users with an audit trail of all configuration changes.
- Allow users to display diffs between current and previous config versions.

Port Requirements

- Configure firewalls on Leader and Worker hosts to allow outbound HTTPS connections to `https://cdn.cribl.io` on port 443 during installation.
- Refer to the list of required ports for Distributed deployments to function, and verify that they are continously open.
- If your network requires a proxy, configure your system proxy settings as described in System Proxy Configuration. Be sure to exclude cluster communication from the proxy to avoid performance issues.
- Cribl prioritizes S3 downloads for Worker Nodes, with automatic fallback and firewall allowlisting tips for optimal performance. See, Optimized Bundle Delivery from S3 for details.

FIPS Mode Requirements

Federal Information Processing Standards (FIPS) is a set of US government standards and guidelines for information security. You can deploy Cribl Stream in **FIPS mode**. This mainly restricts the cryptographic algorithms used within Cribl Stream, and also enforces stricter password requirements.

To learn about system and password requirements, as well as instructions for running Cribl Stream in FIPS mode, refer to the FIPS Mode document.

2.6.2. Single-Instance/Basic Deployment

Getting started with Cribl Stream on a single instance

For small-volume or light processing environments - or for test or evaluation use cases - a single instance of Cribl Stream might be sufficient to serve all inputs, event processing, and outputs. This page outlines how to implement a single-instance deployment.

Architecture

Install on Linux

For system requirements, see OS and System Requirements. To launch Cribl Stream in compliance with secure Federal Information Processing Standards (FIPS), see FIPS Mode.

- Install the package on your instance of choice. Download it here.
- Ensure that required ports are available (see Network Ports).
- Un-tar in a directory of choice, e.g., in the `/opt/` directory: `tar xvzf cribl-<version>-<build>-<arch>.tgz`

To prevent issues with Cribl file operations, `/opt/cribl` and all its subdirectories must reside on the same device. Mounting separate devices within `/opt/cribl` is not recommended. For external storage needs, such as lookups or Persistent Queues (PQs), create a completely separate directory outside of `/opt/cribl`. While `/opt/cribl` is the default installation path, Cribl can be installed to other locations if necessary.

Install Cribl Stream and Cribl Edge on the Same Host

You can run an Edge Node with a Leader that is managing Cribl Stream, or an Edge Node and a Worker Node on the same host. For details, see Installing Cribl Edge and Cribl Stream on the Same Host.

Next, refer to the Run Cribl Stream topic for details on managing Cribl Stream.

2.6.3. Run Cribl Stream

Run Cribl Stream: Start, Stop, and Reload the Service

Cribl Stream Commands

To run Cribl Stream in FIPS mode, **do not** use the commands below right away; instead, first consult the FIPS Mode topic.

Go to the `$CRIBL_HOME/bin` directory, where the package was extracted (for example: `/opt/cribl/bin`). Here, you can use `./cribl` to:

- **Start**: `./cribl start`
- **Stop**: `./cribl stop`
- **Reload**: `./cribl reload`
- **Restart**: `./cribl restart`
- **Get status**: `./cribl status`
- **Switch a** distributed deployment **to single-instance mode**: `./cribl mode-single` (uses the default address:port `0.0.0.0:9000`)

Executing the `restart` or `stop` command cancels any currently running collection jobs. For other available commands, see CLI Reference.

Next, go to `http://<hostname>:9000` and log in with default credentials (`admin:admin`). You can now start configuring Cribl Stream with Sources and Destinations, or start creating Routes and Pipelines.

In the case of an API port conflict, the process will retry binding for 10 minutes before exiting.

Shutdown and Restart Sequence

Different types of configuration changes produce different behaviors in the Worker, as illustrated in the table below.

Configuration change example	How Cribl Streamresponds
Change from one Route to another	Reloads the contents of memory
Add a new Source or Destination	Restarts the Worker Process
Make a change in system settings that requires restarting the API Process	Restarts the API Process and Worker processes

The Leader initiates the configuration change by sending requests to each Worker to trigger deployment. If the change is in the third category from the table above, it will restart the Worker.

Beginning in Cribl Stream v.4.7, when a Worker receives an explicit shutdown command, it follows this sequence:

1. The API Process begins waiting for the Worker Processes to exit.
2. Each Worker Process shuts down its input Sources.
3. Each time one of its input streams ends, a given Worker Process receives a signal event from the Cribl Sream event processor directing it to flush out any stateful Pipeline Functions (such as Aggregations, Sampling, Dynamic Sampling, and Suppress).
4. The Worker Process waits to allow data to "drain" - that is, to finish flowing through the streams processing engine. This waiting period, called the **Drain timeout**, defaults to 10 seconds; see Shutdown settings. When it ends, any data not flushed out - for example, because of an error on downstream receivers - will be lost. If this happens, the `shutdown.lost_events` metric reports the number of events lost.
5. When all the Worker Processes have finished flushing and exited, the API Process reports metrics to the Leader.
6. The API Process exits, and the Worker has now shut down.

When the Leader sends a signal that causes Workers to restart, they all restart at the same time. This can result in a time interval during which no Workers are available to accept connections from upstream senders. If you encounter this problem, you can try configuring a shorter drain timeout, enabling Workers to finish restarting more quickly. This workaround, however, increases the risk of losing outbound data by allowing Worker Processes less time to flush it.

Shutdown/Restart with PQ

Enabling Persistent Queues on Destinations that support it brings both an advantage and a risk:

- The advantage is that enabling PQ generally helps ensure data delivery to your downstream systems.
- The risk is that you might see duplicate events when a Worker Process restarts.

This can happen because PQ only marks events as safe to discard once they have been handed off to the host OS to send out. So, if the Worker Process exits **before** all events have flushed, the final handful of events will not have been marked as committed and removed. Upon restart, Cribl Stream will still see them, and will resend them.

Enabling Start on Boot

Cribl Stream ships with a CLI utility that can update your system's configuration to start Cribl Stream at system boot time. The basic format to invoke this utility is:

```
[sudo] $CRIBL_HOME/bin/cribl boot-start [enable|disable] [options] [args]
```

Copy

You will need to run this command as root, or with `sudo`. For options and arguments, see the CLI Reference.

Most Linux distributions now use `systemd` to start processes at boot, while older distributions might still use `initd`. If you are not sure which service should be configured at startup, check with your Linux administrator. Then follow the corresponding procedure below.

Using systemd

To **enable** Cribl Stream to start at boot time with **systemd**, you need to run the `boot-start` command. Make sure you first create any user you want to specify to run Cribl Stream. For example, to run Cribl Stream on boot as existing user `cribl`, you'd use:

```
sudo $CRIBL_HOME/bin/cribl boot-start enable -m systemd -u cribl
```

This will install a unit file (as shown below) named `cribl.service`, and will start Cribl Stream at boot time as user `cribl`. A `-configDir` option can be used to specify where to install the unit file. If not specified, this location defaults to `/etc/systemd/system/`.

If necessary, change ownership for the Cribl Stream installation:

```
[sudo] chown -R cribl $CRIBL_HOME
```

Next, use the `enable` command to ensure that the service starts on system boot:

```
[sudo] systemctl enable cribl
```

To **disable** starting at boot time, run the following command:

```
sudo $CRIBL_HOME/bin/cribl boot-start disable
```

Other available `systemctl` commands are:

```
systemctl [start|stop|restart|status] cribl
```

Note the file's default `65536` hard limit on maximum open file descriptors (known as a `ulimit`). The minimum recommended value is `65536`. Linux tracks this per user account. You can view the current soft `ulimit` for max open file descriptors with `$ ulimit -n` while logged in as the same user running the `cribl` binary.

Installed systemd File

```
[Unit]
Description=Systemd service file for Cribl Stream.
After=network.target

[Service]
Type=forking
User=cribl
Restart=always
RestartSec=5
LimitNOFILE=65536
PIDFile=/install/path/to/cribl/pid/cribl.pid
ExecStart=/install/path/to/cribl/bin/cribl start
ExecStop=/install/path/to/cribl/bin/cribl stop
ExecReload=/install/path/to/cribl/bin/cribl reload
TimeoutSec=60

[Install]
WantedBy=multi-user.target
```

Copy

Persisting Overrides on systemd

By default, disabling and re-enabling boot start will regenerate the `cribl.service` file. To persist any overrides - such as proxy or privileged port usage - use this command:

```
systemctl edit cribl
```

This opens a text editor that prompts you to enter overrides, then saves them to a persistent file at:

```
/etc/systemd/system/cribl.service.d/override.conf
```

Using a Literal % Character

To use a literal `%` character, make sure to escape it by using a double `%%`.

For example, suppose that you want to persist the environment variable `CRIBL_DIST_WORKER_PROXY` in a systemd unit file, and the value includes a password that contains the special character `@`: `special@password`. You must encode the `@` as `%40`. However, because `%` is a special character that is used for specifiers in systemd unit files, you must also escape the `%` itself by doubling it.

In this example, `%%40` represents the literal `@` character in the password:

```
Environment=CRIBL_DIST_WORKER_PROXY=socks5://proxyuser:special%%40password@mydomain.c
```

Copy

Using Ports 1-1024 With Systemd

To listen on low ports 1-1024, Cribl Stream needs privileged access.

Do not run Cribl Stream as root!

You can enable privileged access on systemd without running as root in the following way:

1. Stop the Cribl Stream server: `systemctl stop cribl`.
2. Add the following configuration key to the `override.conf` file:

   ```
   [Service]
   AmbientCapabilities=CAP_NET_BIND_SERVICE
   ```

   Copy

   If you want to add extra capabilities, such as reading certain resources (such as, `/var/log/*`), add `CAP_DAC_READ_SEARCH` and `CAP_SYS_PTRACE` in a space-separated format as follows:

   ```
   [Service]
   AmbientCapabilities=CAP_NET_BIND_SERVICE CAP_DAC_READ_SEARCH CAP_SYS_PTRACE
   ```

   Copy

3. Either restart the host OS, or run: `systemctl daemon-reload`.
4. Start the Cribl Stream server again: `systemctl start cribl`.

To check whether a port is ready to use, you can run: `netstat -an | grep <port-number>`

Using initd

To **enable** Cribl Stream to start at boot time with **initd**, you need to run the `boot-start` command. If the user that you want to run Cribl Streams does not exist, create it prior to executing. For example, running Cribl Stream as user `cribl` on boot:

```
sudo $CRIBL_HOME/bin/cribl boot-start enable -m initd -u cribl
```

This will install an `init.d` script in `/etc/init.d/cribl.init.d`, and will start Cribl Stream at boot time as user `cribl`. A `-configDir` option can be used to specify where to install the script. If not specified, this location defaults to `/etc/init.d`.

If necessary, change ownership for the Cribl Stream installation:

```
[sudo] chown -R cribl $CRIBL_HOME
```

To **disable** starting at boot time, run the following command:

```
sudo $CRIBL_HOME/bin/cribl boot-start disable
```

To control Cribl Stream, you can use the following initd commands:

```
service cribl [start|stop|restart|status]
```

Persisting Overrides on initd

Notes on preserving required permissions across restarts and upgrades:

Using Ports 1-1024 With Initd

To listen on low ports 1-1024, Cribl Stream needs privileged access.

Do not run Cribl Stream as root!

You can enable privileged access on a Linux system with POSIX capabilities without running as root by adding the `CAP_NET_BIND_SERVICE` capability. For example:

```
# setcap cap_net_bind_service=+ep $CRIBL_HOME/bin/cribl
```

On some OS versions (such as CentOS), you must add an `-i` switch to the `setcap` command. For example:

```
# setcap -i cap_net_bind_service=+ep $CRIBL_HOME/bin/cribl
```

Upgrading Cribl Stream will remove the `CAP_NET_BIND_SERVICE` capability from the `cribl` executable, so you'll need to re-run the appropriate `setcap` command again after each upgrade.

Upon starting the Cribl Stream server, a `bind EACCES 0.0.0.0:<port>` error in the API or Worker logs (depending on the service) might indicate that `setcap` did not successfully execute.

System Proxy Configuration

For details on configuring Cribl Stream to send and receive data through proxy servers, see our System Proxy Configuration topic.

Scaling Up

A single-instance installation can be configured to scale up and utilize as many resources on the host as required. See Sizing and Scaling for details.

Anti-Virus Exceptions

If you are running anti-virus software on a Cribl Stream instance's host OS, here are general guidelines for minimizing accidental blockage of Cribl Stream's normal operation.

Your overall goals are to prevent the anti-virus software from locking any files while Cribl Stream needs to write to them, and from triggering any changes that Cribl Stream would detect as needing to be committed.

First, if Persistent Queues are enabled on any Destinations, exclude any directories that these Destinations write to. This is especially relevant if you're writing queues to any custom locations outside of `$CRIBL_HOME`.

Next, for any non-streaming Destinations that you've configured, exclude their staging paths.

Next, exclude these subdirectories of `$CRIBL_HOME`:

- `state/`
- `log/`
- `.git/` (usually only exists on Leader Nodes)
- `groups/` (on Leader Nodes)
- `local/` (on Workers or Leader)

Finally, avoid scanning any processes. Except for the queueing/staging directories already listed above, Cribl Stream runs everything in memory, so scanning process memory will slow down Cribl Stream's processing and reduce throughput.

Collecting Host Metadata

2.6.4. Configure Cribl Stream For Non-Root User

We strongly recommended to never run Cribl Stream as the root user. Privileged access might be necessary, however, if Cribl Stream needs to read certain resources (such as `/var/log/*`), or to listen on low ports 0-1023. Here are two methods to grant this access while maintaining a non-root user:

- **Using Systemd capabilities**: Set Linux capabilities that grant Cribl Stream sufficient rights to perform specific privileged tasks. For details, see Set Capabilities for Cribl Stream below.
- **Using setcap (Linux with POSIX capabilities)**: This method directly modifies the executable file to grant it specific privileges. For details, see Persisting Overrides on initd.

Set Capabilities for Cribl Stream

Capabilities are permissions that grant privileged processes sufficient rights to accomplish a specific task, based on a kernel privilege. To run Cribl Stream as non-root user, consider setting the following capabilities:

Capability	Permissions
`CAP_NET_BIND_SERVICE`	Allows Cribl Stream to push Sources that bind to TCP/UDP port numbers below `1024`.
`CAP_DAC_READ_SEARCH`	Allows the `cribl` user to access the File Monitor Source's Manual mode feature. This capability bypasses the default Linux permissions for files and directories.
`CAP_SYS_PTRACE`	Allows the `cribl` user to access the File Monitor Source's Auto-mode feature.

For details about setting these capabilities, see Persisting Overrides on systemd.

OS-Specific Options

On some OS versions (such as CentOS), you must add an `-i` switch to the `setcap` command. For example: `# setcap -i cap_net_bind_service=+ep $CRIBL_HOME/bin/cribl`

Upgrading Cribl Stream will remove the `CAP_NET_BIND_SERVICE` capability from the `cribl` executable, so you'll need to re-run the appropriate `setcap` command after each upgrade.

Changing Ownership of Cribl Stream After Startup

To change the ownership of Cribl Stream from root to a non-root user after the application has already been started, you must modify the ownership of both the primary installation directory (`$CRIBL_HOME`) and any Cribl Stream-related directories in the `/tmp` directory. If you changed the default `/tmp` directory location using

the **System** > **General Settings** > **Sockets** > **Directory** setting, then adjust the location accordingly. Use the `chown` command to change the owner of these directories to the desired non-root user. To complete the process, restart the Cribl Stream application instance to ensure the changes take effect.

2.6.5. Distributed Deployment

Cribl Stream Distributed deployment is a multi-instance deployment that is fit for large data volumes and increased processing.

In a Distributed deployment, a single Leader Node manages all Worker Nodes and is responsible for tracking and monitoring their activity metrics, keeping configurations in sync, and handling version control.

The following topics explain more high-level concepts related to Distributed deployment:

- To facilitate authoring and management of configuration settings for a particular set of Worker Nodes as a group, see Worker Groups.
- To plan for unexpected outages in Distributed deployments, see Leader High Availability/Failover.

For detailed deployment requirements, see OS and System Requirements, and for details on managing Cribl Stream, see Run Cribl Stream.

Concepts

Review the following basic concepts as a first step toward understanding how a Distributed deployment works.

Single instance: A single Cribl Stream instance, running as a standalone (not distributed) installation on one server.

Leader Node: A Cribl Stream instance that runs in **Leader** mode. You use the Leader Node to centrally monitor and author configurations for the Worker Nodes in a Distributed deployment.

Worker Node: A Cribl Stream instance that runs as a **managed Worker**. The Leader Node fully manages the configuration of each Worker Node. (By default, the Worker Nodes poll the Leader for configuration changes every 10 seconds.)

Worker Group: A collection of Worker Nodes that share the same configuration. You map Workers to a Worker Group using a Mapping Ruleset.

- Use multiple Worker Groups to make your configuration reflect organizational or geographic constraints. For example, you might have a US Worker Group, an APAC Worker Group, and an EMEA Worker Group, each with their own distinct certifications and settings.

Worker Process: A Linux process within a Single Instance, or within Worker Nodes, that handles data inputs, processing, and output. The process count is constrained by the number of physical or virtual CPUs available; for details, see Sizing and Scaling.

Mapping Ruleset: An ordered list of filters used to map Workers Nodes into Worker Groups.

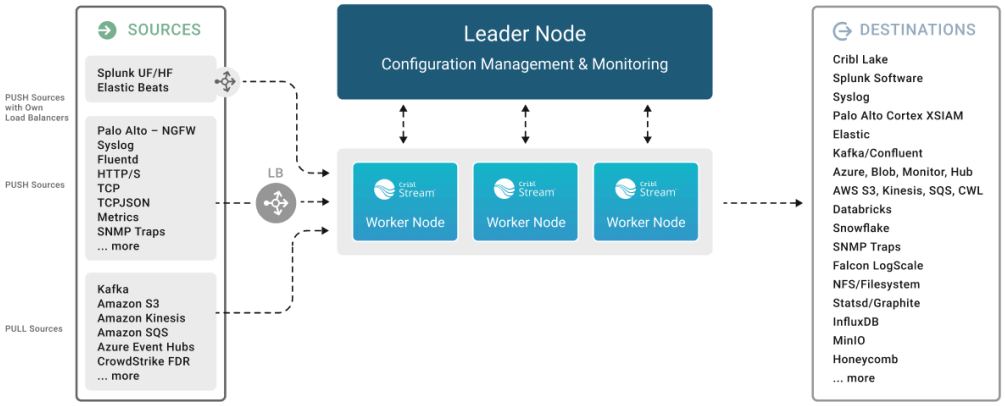
Options and Constraints

You can manually change the local configuration on a Worker Node, but changes won't persist on the filesystem. To permanently modify the configuration of a Worker Node, save and commit it, and then deploy it from the Leader Node. See Deploy Stream Configurations.

With certain license tiers, you can configure Permissions-based or Role-based access control at the Worker Group level. Non-administrator users can access Workers only within the Worker Groups on which they're authorized.

Architecture

Here's a visual overview of a distributed Cribl Stream deployment.



Distributed deployment architecture

The division of labor among components of the Leader Node and Worker Nodes in a Distributed deployment is as follows:

Leader Node

- API Process: Handles all the API interactions.
- Config Helpers: One process per Worker Group. Helps with maintaining configurations, previews, etc.

Worker Nodes

- API Process: Handles communication with the Leader Node (that is, with its API Process) and handles other API requests.
- Worker Processes: Handle all the data processing.

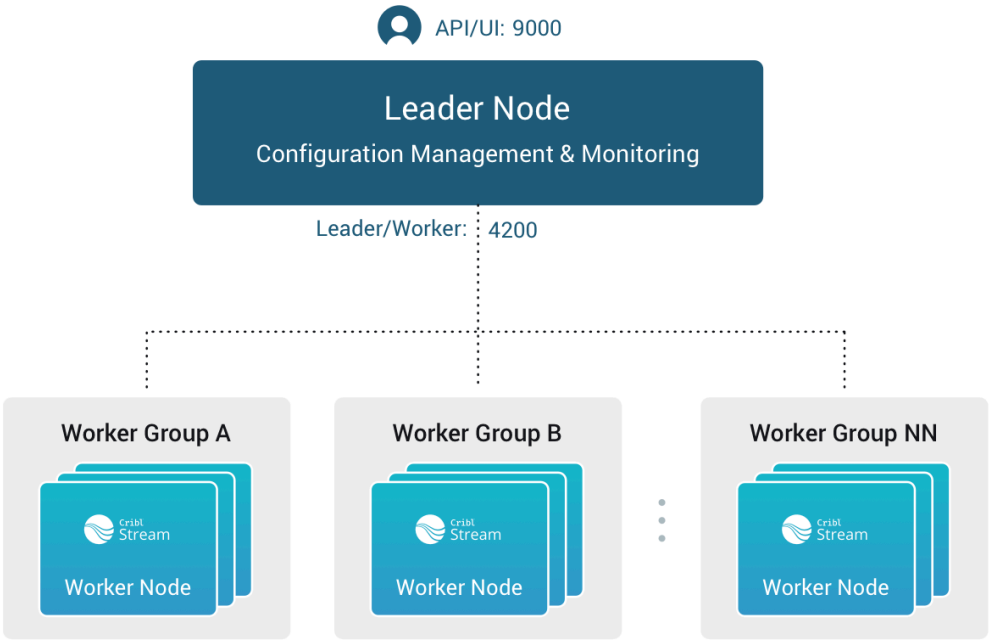
For comparison, the division of labor on a Single-instance deployment is simpler, because the Leader Node and Worker Nodes are essentially condensed into one stack:

- API Process: Handles all the API interactions.
- Worker Processes: Handle all data processing.
    - One Worker Process in a Cribl Stream deployment has a dual role: it processes data like other Worker Processes and assists the Leader Node in managing configurations. The API Process functions similarly to a Leader Node's API Process, handling API requests and responses. Other Worker Processes operate as standard Worker Nodes, primarily focused on data processing.

How Worker Nodes and the Leader Node Work Together

The Leader Node has two primary roles:

- It serves as a central location for the operational metrics of Worker Nodes. The Leader Node contains a monitoring console that includes dashboards that cover almost every operational aspect of the deployment.
- It authors, validates, deploys, and synchronizes configurations across Worker Groups.



Leader Node/Worker Nodes relationship

Network Port Requirements (Defaults)

The following are the default network port requirements.

- UI access to Leader Node: TCP 9000.
- Worker Node to Leader Node: TCP 4200. Used for Heartbeat/Metrics/Leader requests/notifications to clients (for example: live captures, teleporting, status updates, config bundle notifications, and so on).
- Worker Node to Leader Node: HTTPS 4200. Used for config bundle downloads.

Communication

Worker Nodes periodically (every 10 seconds) send a heartbeat to the Leader Node. This heartbeat includes information about the Worker Nodes and a set of current system metrics. The heartbeat payload includes facts - such as hostname, IP address, GUID, tags, environment variables, current software/configuration version, etc. - that the Leader Node tracks with the connection. If a Worker Node fails to successfully send two consecutive heartbeats, it is removed from the Workers page until the next time Leader Node receives a heartbeat message from it.

When a Worker Node checks in with the Leader Node:

1. The Worker Node sends heartbeat to Leader Node.
2. The Leader Node uses the Worker Node's configuration, tags (if any), and Mapping Rules to map it to a Worker Group.
3. The Worker Node gets its Worker Group's updated configuration bundle, if necessary.

Persistent Socket Connections for Leaders

Distributed deployments use Unix domain socket files on the Leader for inter-process communication between internal services. On most Linux distributions, system cleaners periodically delete files in `/tmp`. If socket files under `/tmp` are removed, certain product pages stop working until the Leader is restarted and the sockets are recreated.

Configure socket protection as part of your initial Leader installation. See Persistent Socket Connections for options to:

- Stop the host from cleaning `/tmp/cribl-*` socket files.
- Move socket files to a protected directory using the **Helper processes socket dir** setting on the Leader.

In-Flight Data Safeguarding

If a Worker Node goes down, it loses any data that it's actively processing. With streaming senders (Push Sources), Cribl Stream normally handles that data only in-memory. So, to minimize the chance of data loss, configure persistent queues on Sources and/or Destinations that support it.

Leader/Worker Dependency

If the Leader Node goes down, Worker Groups can continue autonomously receiving and processing incoming data from most Sources. They can also continue executing Collection tasks already in process.

However, if the Leader Node fails, future scheduled Collection jobs will also fail. So will data collection on the Amazon Kinesis Data Streams, Prometheus Scraper, and all Office 365 Sources, which function as Collectors.

Workers GUID on Host Image or VM

When you install and first run the software, Cribl Stream generates a GUID that it stores in a `.dat` file located in `CRIBL_HOME/local/cribl/auth`, e.g.:

```
$ cat /opt/cribl/local/cribl/auth/676f6174733432.dat
{"it":1647910738,"phf":0,"guid":"e0e48340-f961-4f50-956a-5153224b34e3","lics":["licer
```

Copy

When deploying Cribl Stream as part of a host image or a virtual machine, be sure to remove this file, so that you don't end up with duplicate GUIDs. Cribl Stream will regenerate the file on the next run.

Commit and Deploy Controls

Distributed mode includes Worker Group-specific **Commit** and **Deploy** controls. These controls function differently from the global **Commit** button because they have a different scope.

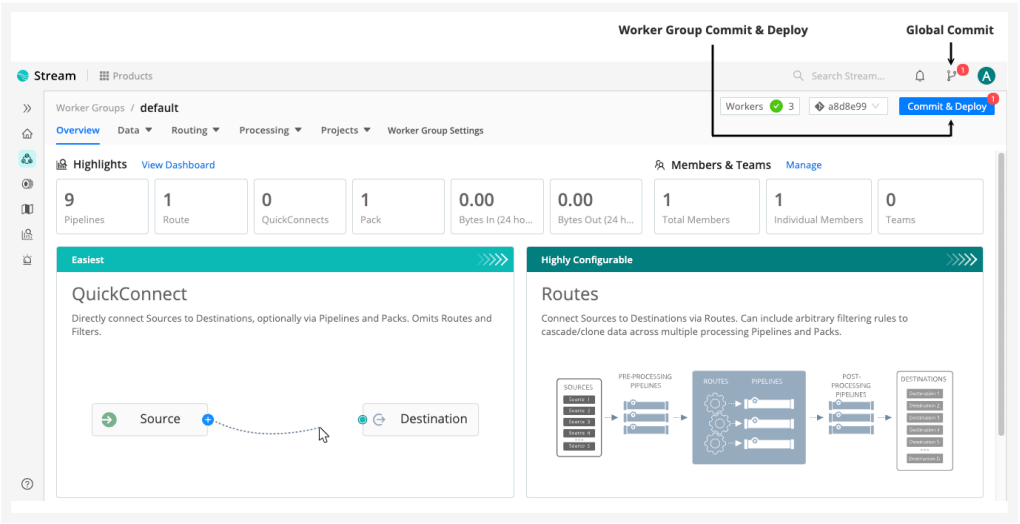
The global **Version Control** button:

- **Scope:** Commits changes for **all** Worker Groups and the Leader Node simultaneously.
- **Result:** Changes are versioned but not deployed.

The Worker Group **Commit** and **Deploy** buttons:

- **Scope:** Commits and deploys changes for only the currently selected Worker Group.
- **Result:** Allows for a single-click Commit and Deploy, making changes live for that specific group immediately.

These controls will appear slightly differently depending on whether you have changes committed but not yet deployed, and on whether you've enabled the Collapse actions setting.



Commit and Deploy controls in the user interface

For details, see Version Control.

Manage Workers and Worker Groups

To manage your Cribl Stream deployment's resources, use the Worker and Worker Group management features.

Manage Worker Groups

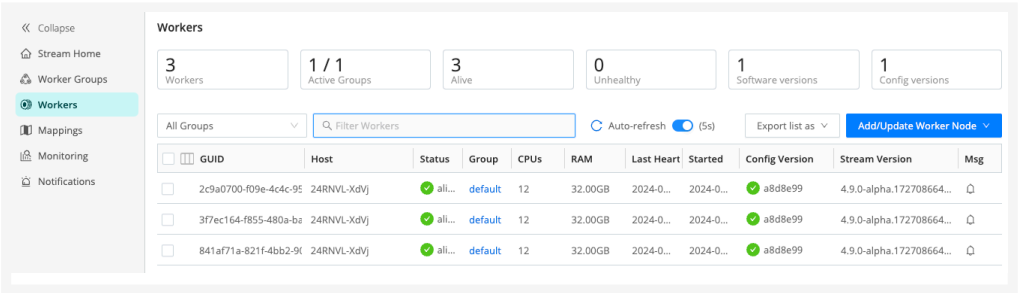
To manage Worker Groups, in the sidebar, select **Worker Groups**.

This opens the **Worker Groups** landing page, where you can view summary of information for each Worker Group and select it to view its Worker Nodes.

Manage Workers

To manage Worker Nodes, in the sidebar, select **Workers**.

The **Workers** tab provides status information for each Worker Node in the selected Worker Group. An **Auto-refresh** toggle (on by default) ensures that the **Config Version** automatically refreshes when you commit or deploy. (This option is available in Cribl Stream 4.4 and newer and appears only in Worker Groups that contain fewer than 100 Worker Nodes.)



Worker Nodes status/controls

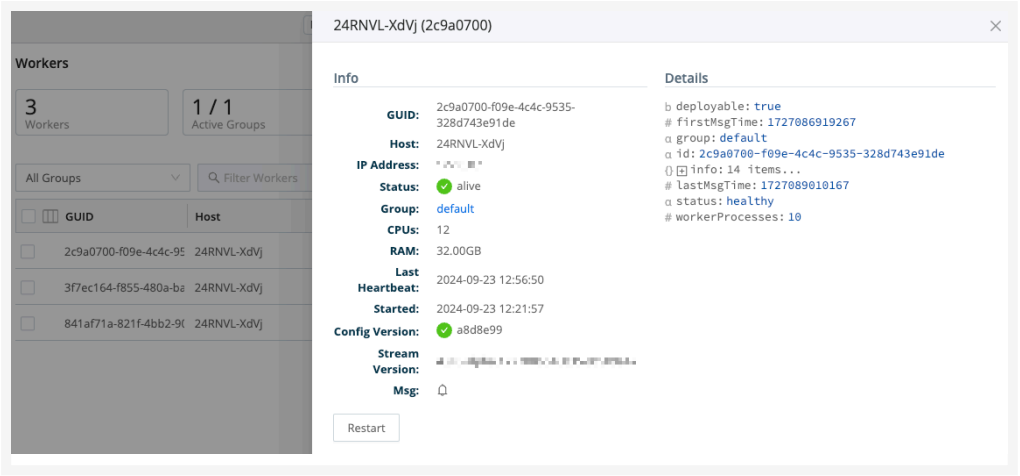
If you see unexpected results here, keep in mind that:

- Worker Nodes that miss 5 heartbeats, or whose connections have closed for more than 30 seconds, will be removed from the list.
- For a newly created Worker Group, the **Config Version** column can show an indefinitely spinning progress spinner. This happens because the Worker Nodes are polling for a config bundle that has not yet been deployed. To resolve this, select **Deploy** to force a deploy.

To export the list of Worker Nodes, select the **Export list as** drop-down. Select `JSON` or `CSV` as the export file format. If a list has any filters applied, the exported list will also be filtered. To export an entire list, remove any

applied filters.

To reveal more details about a Worker Node, click anywhere on its row.



Worker Node details

Manage Mappings

The **Mappings** page displays status and controls for the active Mapping Ruleset.

To manage your mapping rulesets, select **Mappings** in the sidebar.

Select a Mapping Ruleset ID to manage and preview its rules.

Auto-Scale Workers and Load-Balance Incoming Data

If data flows in via load balancers, make sure to register all instances. Each Cribl Stream Node exposes a health endpoint that your load balancer can check to make a data/connection routing decision.

Health Check Endpoint	Healthy Response
`curl http://<host>:<port>/api/v1/health`	`{"status":"healthy"}`

2.6.5.1. Set Up Leader And Worker Nodes

In a Distributed deployment, each Cribl Stream instance runs in a defined mode: either as a Leader Node, governing the whole deployment, or as a Worker Node managed by the Leader.

Before setting up your Leader and Worker Nodes, be sure to review Deployment Planning.

You can select the mode in which any instance operates, using:

- Environment Variables
- UI Settings
- UI Access to Workers (Teleporting)
- Add Tags to Workers in Teleport View
- CLI
- `instance.yml`

Internally, the mode for a Leader Node is called `master`, which corresponds to an earlier, outdated name for the mode. That is why you will see `master` in commands, configuration keys and environment variable names.

Set Mode with Environment Variables

You can configure the mode for Leader and Worker Nodes via environment variables.

`CRIBL_DIST_MODE` defines the mode in which the current environment is working: `worker` or `leader`. `CRIBL_DIST_LEADER_URL` defines the Leader Node URL for the deployment.

For a Worker Node, set `CRIBL_DIST_LEADER_URL` in the following way:

```
export CRIBL_DIST_LEADER_URL=tcp://${CRIBL_DIST_TOKEN:-criblmaster}@<leaderHostname>:
```

Copy

When configuring a Leader Node, use `0.0.0.0` (or another suitable binding address) for `host`:

```
export CRIBL_DIST_LEADER_URL=tcp://${CRIBL_DIST_TOKEN:-criblmaster}@0.0.0.0:4200 ./cr
```

Copy

If `CRIBL_DIST_LEADER_URL` is set, you can omit setting `CRIBL_DIST_MODE`, and the mode for the instance will default to `worker`.

Environment variables take priority over other ways of setting up Leader and Worker Nodes. If any `CRIBL_DIST_*` variable is defined in the system, you won't be able to configure Distributed mode settings via UI.

Set Mode with UI Settings

You can configure the mode for each instance in a Distributed deployment via UI.

1. In the sidebar, select **Settings**, then **Global**.
2. Select **Distributed Settings**, then **General Settings**.
3. In **Mode**, choose the mode you want the instance to run as:
    - **Leader**.
    - **Stream: Managed Worker (managed by Leader)**.
4. Select **Leader Settings**, and configure the required Leader settings:
    - **Address** and **Port** for Leader Node.
    - **Address** (for example: `criblleader.mycompany.com`) for Worker.
    - **Forward to Leader API**: Toggle on (default) to enable API requests on the distributed port and provide backward compatibility.

    If you need to customize the Leader's port (from its default `4200`), you can do so via the CLI's `mode-worker` command.
5. Optionally, customize the remaining settings.

If you configure Leader High Availability/Failover, further **Distributed Settings** changes will be locked for each Leader Node. You can still update distributed settings by modifying configuration files, as outlined on the page linked above.

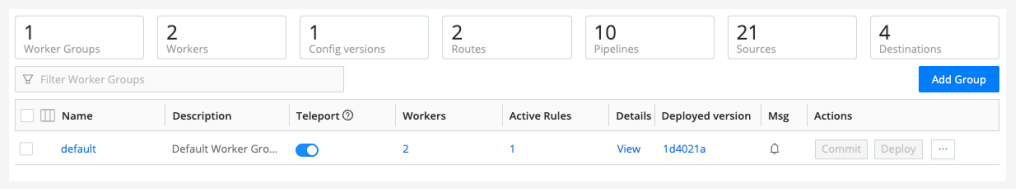
Enable UI Access to Workers (Teleporting)

Remote UI access to Workers (also known as teleporting) enables you to click through from the Leader's **Workers** page to an authenticated view of each Worker's UI.

When you make a change directly on a Worker Node using remote access (teleport), Cribl Stream does **not** propagate those changes to the Leader Node. The next commit and deploy from the Leader will **overwrite** the changes. The only exception is adding tags. Additionally, the Preview view in teleport will not reflect any local changes done using remote access (teleport).

To enable teleporting (remote UI access) to Workers from the Leader Node:

1. In the sidebar, select **Worker Groups**.
2. In the row of the Worker Group you want to enable teleporting for, enable **Teleport**.

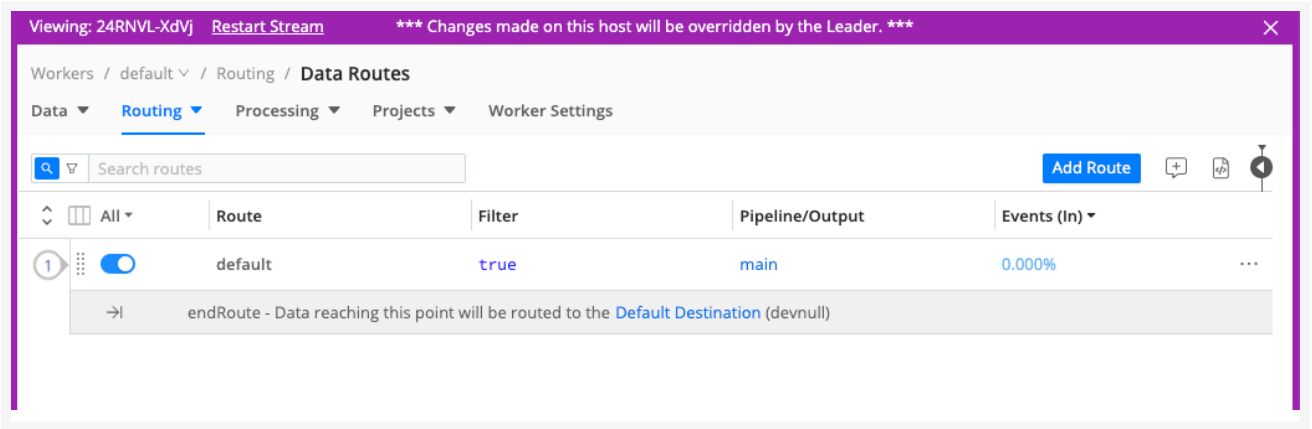


Worker Teleport setting

On Cribl.Cloud, the **Worker Groups** page adds extra columns. You'll see each Group's Worker type (customer-managed / hybrid versus Cribl-managed / Cribl.Cloud) and, for Cloud Groups, the anticipated ingress rate and Provisioned status. For details, see Managing Cribl.Cloud Worker Groups.

To access a Worker Node via teleporting:

1. In the sidebar, select **Workers**.
2. Select the link for any Worker Node you want to inspect.
   A purple border that appears around the UI indicates that you are viewing the Worker Node remotely. Select **Restart Stream** on the border to restart the Worker, or select the **X** close button to return to the **Workers** page.



Authenticated view of a Worker

An alternative way of allowing UI access to Workers is enabling the `workerRemoteAccess` configuration key in `groups.yml`.

Add Tags to Workers in Teleport View

You can add tags to a Worker Node you've accessed by teleporting. Tags help the Leader Node map a Worker Node to a Worker Group. Note that changing the tags on a Worker can cause the Leader to map the Worker into a new Group.

1. Teleport into a Worker Node.
2. On the **Workers** submenu, select **Worker Settings**, then **Distributed Settings**.
3. Enter tags for this Worker Node and select **Save**.

In Cribl.Cloud, you can only add tags to *hybrid* Worker Nodes. You must accept the confirmation modal, as changing the tags on a Worker can affect the Worker Group that the Leader maps it into.

Set Mode with CLI

You can configure the mode for an instance in a Distributed Deployment by using commands: `mode-master` and `mode-worker`.

For example, to configure a Worker Node, use command of this form:

```
./cribl mode-worker -H <master-hostname-or-IP> -p <port> [options] [args]
```

Copy

The command requires the `-H` and `-p` parameters. For other options, see the CLI Reference for mode-worker.

Here is an example command:

```
./cribl mode-worker -H 192.0.2.1 -p 4200 -u myAuthToken
```

Copy

Cribl Stream will need to restart after this command is issued.

Set Mode in `instance.yml`

You can configure each instance to be either Leader or Worker in `$CRIBL_HOME/local/_system/instance.yml`, under the `distributed` section, by setting the `mode` value:

Leader Worker

$CRIBL_HOME/local/_system/instance.yml

```
distributed:
  mode: master
  master:
    host: <IP or 0.0.0.0>
    port: 4200
    forwardToLeaderApi: true
    tls:
      disabled: true
    ipWhitelistRegex: /.&ast;/
    authToken: <auth token>
    enabledWorkerRemoteAccess: false
    compression: none
    connectionTimeout: 5000
    writeTimeout: 10000
```

Copy

Leader/Worker Mode and Auth Tokens

When you configure an instance to work as the Leader, switching from a Single-instance to a Distributed mode, Cribl Stream generates a random auth token.

See How to Secure the Auth Token for the Leader Node for information on changing your auth token and allowed characters.

Troubleshooting Resources

Cribl University offers a Troubleshooting Criblet on Attaching Workers to Leaders. (To follow the direct course link, first log into your Cribl University account - it's free!)

2.6.5.2. Manage Worker Groups

Worker Groups are collections of Worker Nodes that share the same configuration. They facilitate authoring and management of configuration settings.

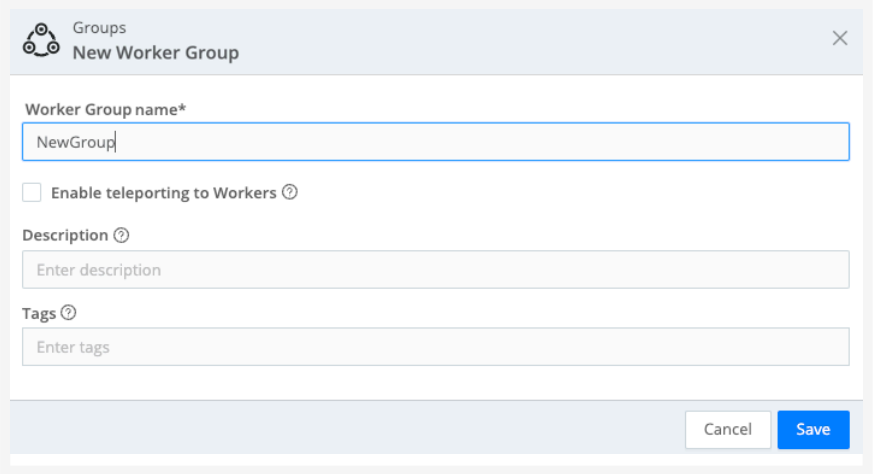
You configure data flow, including Routes, Pipelines, Sources, Destinations, per Worker Group. Each Worker Group also has separate version control.

A Cribl.Cloud deployment on a certain plan also allows you to create Cribl-managed Worker Groups in Cribl.Cloud. See Managing Cribl.Cloud Worker Groups for more details.

Create a Worker Group

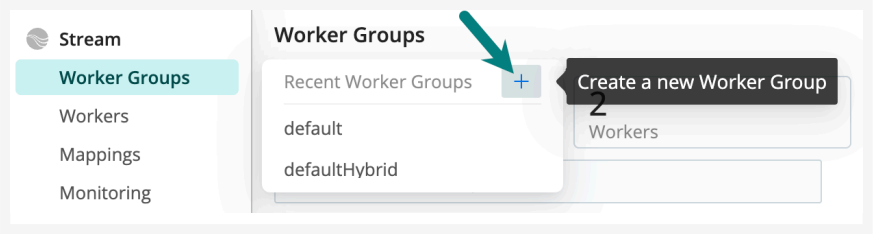
To create a new Worker Group:

1. In the sidebar, select **Worker Groups**.
2. Select **Add Worker Group**.
3. If you are on Cribl.Cloud and see the **Worker Group type** field, select **Hybrid**.
4. Enter a unique **Worker Group name**.
5. Optionally, you can **Enable teleporting to Workers** to allow UI access for individual Worker Nodes.
6. Optionally, type in a **Description** and select **Tags**.



Creating a New Group

You can also create a new Worker Group by selecting **Products** on the top bar, hovering over **Worker Groups** in the sidebar, and selecting the **+** button.



Creating a New Worker Group from **Products** Sidebar

You can also use the Cribl API to create a new Worker Group.

Configuring multiple Worker Groups, or configuring more than 10 Worker Processes, requires a Cribl Stream on-prem deployment with a certain license tier.

To explicitly set passwords for Worker Groups, see User Authentication.

Clone a Worker Group

Cloning a Worker Group copies everything configured on the original Group: Sources, Pipelines, Packs, Routes, and Destinations.

To clone a Worker Group:

1. In the sidebar, select **Worker Groups**.
2. In the **Actions** column, select ••• (Options).
3. Select **Clone**.
4. Configure the cloned group by choosing its name, and optionally description and tags.

Recover a Worker Group

If you delete a Worker Group, you can no longer access its **Version Control** menu. As a result, you also lose the option to directly recover the Worker Group.

You can, however, still recover a deleted Worker Group, using one of the following two methods:

- Create a new Worker Group with the same name.
- Revert commit via command line.

Create a New Worker Group with the Same Name

When you create a new Worker Group with the same name as a deleted one, it will restore that group with all of its configuration.

1. In the sidebar, select **Worker Groups**, then select **Add Worker Group**.
2. Set **Worker Group name** to be exactly the same as the deleted group. Fill in the remaining information and **Save**.
3. If you previously committed the deletion of the original Worker Group, open the **Version Control** menu.
4. Select a previous version of the deleted Worker Group and confirm with **Done**.

Revert Commit via Command Line

You can restore a deleted Worker Group by reverting the deletion commit in version control via command line.

1. Stop the Cribl server.
2. Using command line, look at the version control history for the Leader Node and identify the commit which deleted the Worker Group.
3. Revert to the last commit before the Group was deleted.
4. Restart the Cribl server.

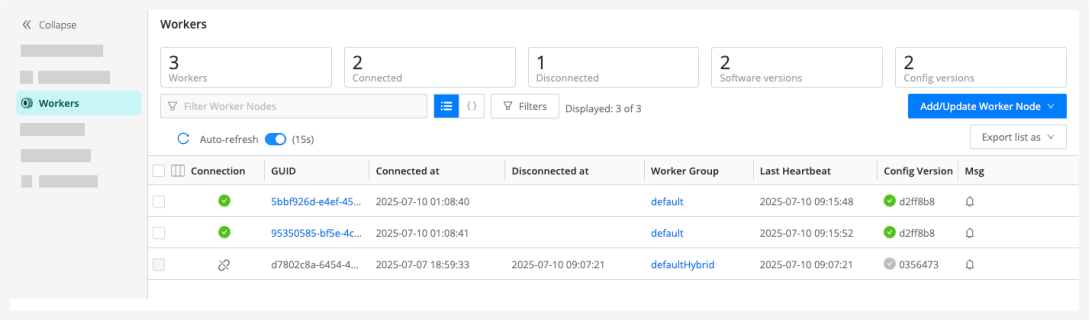
2.6.5.3. Manage Workers

Learn how to manage and filter Stream Workers

View Worker Status Information

The **Workers** page provides status information for each Worker Node in the selected Worker Group, as well as a UI for adding and updating Workers.

In Cribl Stream, select **Workers** from the sidebar.



Workers page showing a list of Worker Nodes

Monitor Worker Connectivity

You can track which Worker Nodes are connected to the Leader Node in the **Workers** page. Disconnected Workers show an icon in the **Connection** column and will have information about the time the Leader last received their heartbeat.

By default, Cribl Stream will retain information about disconnected Workers between restarts. To change this behavior on-prem, toggle **Disable Node persistence** on in **Global Settings** > **System** > **General Settings** > **Limits** > **Other**. This will also affect Cribl Edge Nodes.

Disable Node persistence is a deprecated setting that will be removed in a future version. Toggling **Disable Node persistence** will restart the Leader Node.

You can configure how long the Leader should keep track of disconnected Worker Nodes by configuring the **Time to keep disconnected Workers** Worker Group setting (**General Settings** > **Worker Group Configuration**). By default the Leader retains information about disconnected Nodes for one day. You can set it to any value between 0 (which means Nodes are removed within a few minutes) and 90 days.

Filter Workers

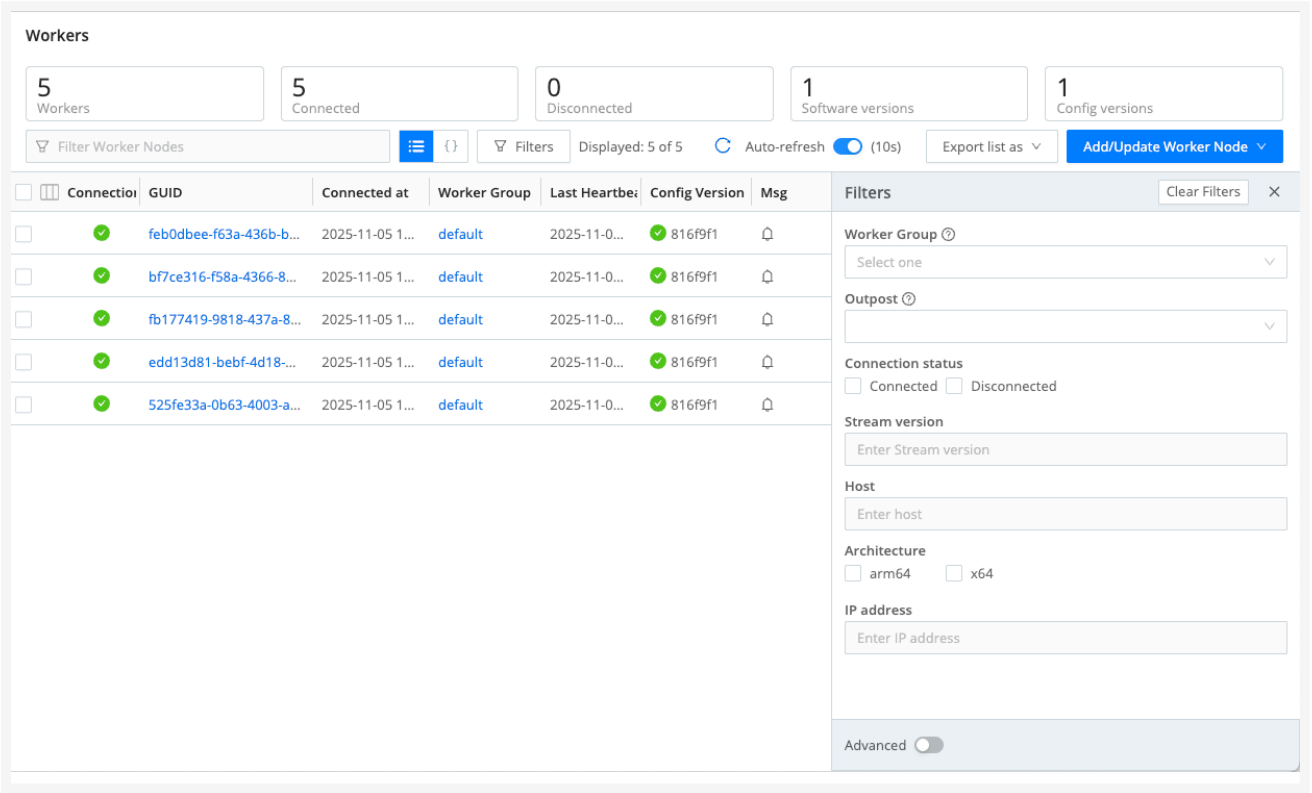
When your Cribl Stream deployment has a large number of Workers, you may want to filter your Worker list down to a manageable subset. This can help you monitor your Stream Workers and troubleshoot any issues.

The **Filters** option allows you to apply filter criteria to your Worker list, so you can quickly find Workers by connection status, IP address, version, and more.

Access Worker Filters

In Cribl Stream, you can filter Workers in two places:

- In a specific Group in the Worker Groups list. Navigate to **Worker Groups**, then select the **Workers** for the Group.
- On the list of all Workers. Navigate to **Workers** from the sidebar.



The Filters option

Filter Modes

You can choose between the default form-based mode and **Advanced** mode when you create a Worker filter. Form-based mode contains the preconfigured fields and checkboxes you see when you select **Filters**.

For a table of the options and their definitions, see Form-based Filter Options.

Advanced mode is a JavaScript input field where you can create your own filter, referencing the JSON objects that are associated with a Worker Node.

You can edit the JavaScript by typing in the **Expression** field, but doing so disables the form-based filters unless you undo the changes. To revert back to the form-based filter, select **Undo changes**.

Create a Worker Filter

The **Filters** option lets you group a list of Workers by criteria such as IP address, install type, version, and more.

To create a Worker filter:

1. Navigate to a list of Workers, either from the **Workers** column in a Group or from the **Workers** sidebar.
2. Select **Filters** to open the Filters panel.
3. Create a filter by selecting from the Worker Filter Options, or toggle **Advanced** on to write your own filter using JavaScript.
4. Once you have selected filter criteria, you can use the **Filter Worker Nodes** field to further narrow down the list of visible Workers.
5. Select **Clear filters** to undo your selections in the Filters panel. This does not clear the **Filter Worker Nodes** field.

Share a Filter

To share a filter with another Cribl user:

1. Select the **Filters** button and create a filter.
2. Select the link icon to copy the filters to your clipboard, or copy the browser URL.
3. Now you can share the link with another user.

Filters are Workspace-specific, so they will only work within the Workspace where you created them.

Form-based Filter Options

You can choose from a set of preconfigured, form-based filter options, and each has an associated JavaScript property. You'll see the JavaScript property in the **Expressions** field when you toggle **Advanced** on.

Field	Description	JavaScript property
Worker Group	Find Workers based on their Worker Group assignment. Visible when filtering from the **Workers** page.	`group=='<group>'`

Field	Description	JavaScript property
Outpost	Which Outpost the Worker is connected to.	`info.outpost.host.toLowerCase().includes(<outpost-host>)`
Connection status	Whether the Worker is currently connected to the Leader.	`(disconnected === <true|false>)`
Stream version	Enter a semantic version of Cribl Stream. For example, 4.1.0.	`info.cribl.version.toLowerCase()`
Host	The host of the Worker. You can type in any part of the host string.	`info.hostname.toLowerCase().includes(<host>)`
Architecture	The processor architecture for the Worker.	`(info.architecture === '<architecture>')`
IP address	The IP address of the Worker Node. Entering a partial value returns all Nodes with matching values. For example, entering `17` will return all Nodes that have `17` somewhere in the address.	`info.conn_ip.toLowerCase().includes('<ip-here>')`

Advanced Filter Options

You can use JavaScript expressions in the Advanced filter to surface more specific, granular search results. You can use any Worker Node properties and types in the **Expression** field.

The JavaScript expression should be in this format, in hierarchical order from parent to child:

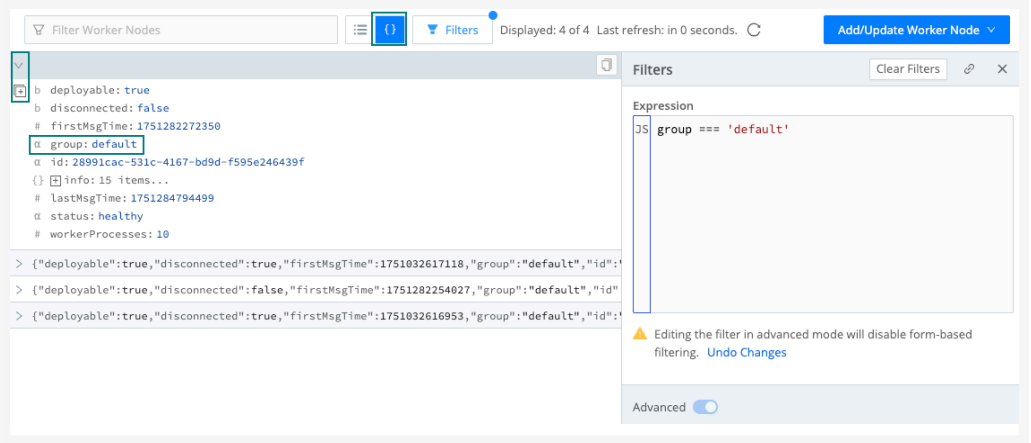
```
parent.child.grandchild === 'value'
```

For example:

`info.aws.type === 't2.medium'`.

You can also automatically populate the Advanced filter **Expression** field:

1. Select the **JSON view** button.
2. Expand the Worker Node row and select the Worker property, then any applicable child properties.
3. Type directly in the **Expression** field to fine-tune the filter.



Filter Worker Nodes in JSON view

Supported Node Properties

You can filter on these Worker Node properties using a JavaScript expression. Remember to separate the parent and child properties using a period `.`.

```
"deployable": "boolean (Whether the Node can be deployed)",
"disconnected": "boolean (Whether the Node is connected to the Leader)",
"workerProcesses": "number (Number of Worker processes for this Node"),
"id": "string",
"status": "string (Healthy)",
"group": "string (Group name)",
"info": {
  "hostname": "string",
  "platform": "string (OS Platform; linux, etc.)",
  "architecture": "string (CPU Architecture; arm64, x64 etc.)",
  "release": "string (OS Kernel release)",
  "cpus": "number (Number of CPU cores)",
  "totalmem": "number (Memory in bytes)",
  "node": "string (NodeJS version)",
  "cribl": {
    "startTime": "number (Unix time in msecs when the process started)",
    "guid": "string (unique instance ID)",
    "installType": "string",
    "tags": "string array (tags configured on the Node)",
    "version": "string (Stream version)"
  },
  "freeDiskSpace": "number (Free disk space in bytes)",
  "totalDiskSpace": "number (Total disk space in bytes)",
  "conn_ip": "string (this is the IP that Cribl Leader sees; different from actual
  "localTime": "number (local Unix time on the Node in msec)"
},
"lastMsgTime": "number (time when Node sent the last message in msec)",
"firstMsgTime": "number (time when Node sent the first message in msec)"
```

Copy

Example Advanced Filter Expressions

2.6.5.4. Map Workers To Worker Groups

Mapping Rulesets define how Worker Nodes are distributed among Worker Groups. A Ruleset consists of a list of rules that evaluate Filter expressions on the information that Worker Nodes send to the Leader Node.

Only one Mapping Ruleset can be active at any one time.

The ruleset behavior is similar to Routes, where the order matters, and the **Filter** section supports full JavaScript expressions. The ruleset matching strategy is first-match, and one Worker Node can belong to only one Worker Group.

Configuring Mapping Rulesets is possible only when you have started Cribl Stream in Leader mode.

If you define no Mapping Rulesets of your own, the built-in `default` Mapping Ruleset with a `default` rule matching all (`true`) assigns all Worker Nodes to the `default` Worker Group, unless the Worker Node itself specifies differently.

Worker Mapping Priority

Worker Nodes can be assigned to Worker Groups in two ways: via Mapping Rulesets, and by including the group in the Worker's payload. When both are defined, the assignment follows this priority:

1. Group matched by a filter in a Mapping Ruleset.
2. Group defined in the Worker Node.
3. The `default` group.

Create a Mapping Ruleset

To create a Mapping Ruleset:

1. In the sidebar, select **Mappings**.
2. Select **Add Ruleset**
3. Provide a unique **ID** for the new Ruleset and confirm with **Save**.

The Ruleset **ID** must not contain slashes (`/`).

4. Select the new Ruleset in the table to enter a rule editing screen.

You can't create custom Mapping Rulesets for Cribl Stream Workers in Cribl.Cloud.

Add Mapping Rules

Each Mapping Ruleset can have one or more rules.

1. Select **Add Rule** to start adding rules. While you build and refine rules, the Preview in the right pane will show which currently reporting and tracked Worker Nodes map to which Worker Groups.
2. For each rule, formulate a **Filter** expression that maps Worker Nodes.
3. In **Worker Group**, identify the Worker Group to which the Worker Node should be assigned.
4. **Save**, and add more rules, if needed.

Activate Mapping Ruleset

A ruleset must be activated before it can be used by the Leader Node.

To activate it, go to **Mappings** and select **Activate** on the required ruleset.

Activating a ruleset deactivates any other existing rulesets.

Mapping Rule Example

This example assumes that you want to define a rule for all hosts that satisfy the following set of conditions:

- IP address starts with `10.10.42` AND Worker Node has more than 6 CPUs, OR:
- `CRIBL_HOME` environment variable contains `w0`.

The rule should assign the Worker Node to `Group420`.

Field	Value
Rule Name	`myFirstRule`
Filter	`(conn_ip.startsWith('10.10.42.') && cpus > 6) || env.CRIBL_HOME.match('w0')`
Group	`Group420`

2.6.5.5. Manage Config Bundles

Cribl Stream uses config bundles, compressed archives containing configuration files and data essential for Worker operation.

Bundle Management in Cribl.Cloud

In Cribl.Cloud, bundle storage and delivery are fully managed. You do not need to configure Amazon S3 buckets or remote URLs for Cribl-managed Worker Groups and hybrid Worker Groups.

- **Primary path (S3 over HTTPS)**: Cribl-managed Workers first attempt to download the new config bundle from a Cribl-managed Amazon S3 bucket over HTTPS (via port `443`).
- **Automatic fallback (Leader over HTTPS)**: If the Worker cannot reach the S3 endpoint (for example, because of firewall or proxy rules), it automatically falls back to downloading the same bundle from the Leader over HTTPS via port `4200`. For details on ports, see Ports and Required Ports in Cribl.Cloud.

For details on preventing deployment delays by white-listing the required S3 and Leader endpoints, see Network Access for Config Bundles.

Bundle Management in Customer-Managed Deployments

For Workers you manage, you control bundle storage:

- **Default**: The Leader serves bundles directly from its local storage.
- **Optional S3**: You can offload bundle storage to your own S3 bucket to reduce the load on your Leader Node. For details, see Store Bundles Remotely.

Bundle Lifecycle and Retention

Regardless of the delivery method, the Leader and Worker Nodes automatically manage the lifecycle of the config bundles archives.

Leader Node Bundles

The Leader Node actively manages bundles, performing the following tasks:

- **Cleanup on startup**: Upon startup, the Leader Node clears all bundles.
- **Running state**: The Leader Node maintains a maximum of five bundles per Worker Group while running.
- **Automatic cleanup**: Creating a new bundle triggers the deletion of older ones.

Worker Node Bundles

The Worker Node pulls bundles from the Leader Node and manages them as follows:

- **Cache**: Worker Nodes cache the latest five bundles and their backups.
- **Recent file retention**: Worker Nodes retain any files created within the last ten minutes.
- **Reconfigure cleanup**: Reconfigured events initiate bundle cleanup.

Store Bundles Remotely

For customer-managed Leaders, you can optionally reduce Leader load by storing bundles in an Amazon S3. This offloads distribution, allowing Worker Nodes and Edge Nodes to pull bundles directly from S3.

Configure S3 Storage

You can configure bundling in S3 through the following ways:

- **UI**: Configure the Leader settings on the Leader Node. In the sidebar, select **Settings**. In the **Global** tab, select **System** > **Distributed Settings** > **Leader Settings**. In the **S3 Bundle Bucket URL** field, specify an S3 bucket for remote bundle storage. Format: `s3://${bucket}`.
- **YAML**: Define the S3 bucket URL in the `master.configBundles.remoteUrl` property of your `instance.yml` configuration file.
- **Environment variable**: Set the `CRIBL_DIST_LEADER_BUNDLE_URL` environment variable.

Optimized Bundle Delivery

Cribl Stream prioritizes efficient bundle downloads for Worker Nodes. Here's how:

- **Direct S3 downloads (preferred)**: When possible, Cribl prioritizes direct downloads from Amazon S3 for optimal performance.
- **Private network fallback**: Worker Nodes on private networks attempt to retrieve bundles from a Content Delivery Network (CDN). If blocked, they automatically fall back to downloading from the Leader Node.
- **Optional S3 firewall allowlist**: For Worker Nodes with internet access, consider allowlisting the regional Amazon S3 endpoint used by your bundle bucket in your firewall or proxy configuration. For example, endpoints look like `s3.eu-central-1.amazonaws.com` or `s3.us-west-2.amazonaws.com`, and dual-stack variants look like `s3.dualstack.us-west-2.amazonaws.com` or `bucket-name.s3.dualstack.us-west-2.amazonaws.com`.

S3 endpoints are regional and do not share a single wildcard hostname, so patterns such as `*.s3.amazonaws.com` are not sufficient for firewall or proxy allowlists. Refer to the AWS documentation: Using Amazon S3 dual-stack endpoints for the complete and current list of S3 regional and dual-stack endpoints.

Authenticating to S3

When storing configuration bundles in an Amazon S3 bucket, Cribl Stream needs appropriate credentials to access the bucket. There are several ways to provide these credentials, with IAM roles being the most secure and recommended approach.

IAM Roles (Recommended)

The best practice is to configure an IAM role with read access to your S3 bucket and attach this role to the EC2 instance (or other AWS service) where your Cribl Stream Leader and Worker Nodes are running. Cribl Stream will automatically use the instance's IAM role to authenticate with S3. This method avoids the need to manage AWS credentials directly within Cribl and is the most secure option.

To authenticate via IAM roles:

1. Create an IAM role with the `AmazonS3ReadOnlyAccess` policy (or a custom policy with more restrictive permissions if needed) for the S3 bucket containing your bundles.
2. Attach this IAM role to the EC2 instance(s) running your Cribl Stream Leader and Worker Nodes.
3. Provide the path to your S3 bucket using the CRIBL_DIST_LEADER_BUNDLE_URL environment variable or the S3 Bundle Bucket URL field in the **Distributed/ Leader** settings (For example, `s3://your-bucket-name/path/to/bundles`). `path/to/bundles` is optional.

No further configuration is required. Cribl Stream will automatically discover and use the instance's IAM role.

AWS Access Keys

Alternatively, you can provide AWS access keys directly. Ensure the environment in which Cribl Stream runs in contains the following variables:

```
export AWS_ACCESS_KEY_ID="YOUR_ACCESS_KEY_ID"
export AWS_SECRET_ACCESS_KEY="YOUR_SECRET_ACCESS_KEY"
export CRIBL_DIST_LEADER_BUNDLE_URL="s3://your-bucket-name/path/to/bundles"
```

Copy

Encryption and Backward Compatibility

If your Workers are running versions older than 4.9.0 and your Leader is v.4.9.0 or newer, deployments will always use Leader-hosted bundles. This happens because of how encryption works:

- Newer Leaders encrypt bundles when uploaded to S3 to enhance security and prevent the exposure of sensitive information like secret keys.
- Older Workers check the code for errors by comparing its checksum (a unique identifier).

- Since the Workers check the encrypted code, they will always find errors because the checksum changes after encryption.

You might see an error similar to the one below in your logs: `Checksum mismatch, expected=f781b2bace2e1869e840b0ee200b57378ecae985, found=a59af697c140521ea7f42af7630ac1fce88e4ff0"**`

2.6.5.6. Deploy Stream Configurations

Cribl Stream configurations are deployed after the bundle containing them is saved and committed. *Deploying* here means **propagating updated configurations to Worker Nodes.**

The typical workflow for deploying Cribl Stream configurations is the following:

1. Configure.
2. Save.
3. Commit (and optionally push).
4. Deploy.

You deploy new configurations at the Worker Group level: Locate the desired Worker Group and select **Deploy**. Worker Nodes that belong to the group will start **pulling** updated configurations on their next check-in with the Leader Node.

Can't Log In to the Worker Node as Admin User?

When a Worker Node pulls its first configuration, the admin password will be randomized, unless specifically changed. This means that users won't be able to log in on the Worker Node with default credentials. For details, see Set Worker/Edge Node Passwords.

Configuration and Lookup File Locations

Node Type	Configuration Location	Lookup File Location
Leader Node	`$CRIBL_HOME/groups/<groupName>/local/cribl/`	`$CRIBL_HOME/groups/<groupName>/`
Worker Node	`$CRIBL_HOME/local/cribl/` (after pulling configurations)	`$CRIBL_HOME/data/lookups` (after pu configurations)

When deployed via the Leader Node, lookup files are distributed to Worker Nodes as part of a configuration deployment.

If you want your lookup files to be part of the Cribl Stream configuration version control process, we recommended deploying using the Leader Node. Alternatively, you can update your lookup file on the individual Worker Nodes, which can be useful for larger lookup files (greater than 10 MB, for example), for lookup files maintained using some other mechanism, or for lookup files that are updated frequently.

For other options, see About Lookups.

Some configuration changes will require restarts, while many others require only reloads. See Configurations and Restart for details.

Restarts and reloads of each Worker Process are handled automatically by the Worker Node. Note that individual Worker Nodes might temporarily disappear from the Leader's **Workers** tab while restarting.

Worker Process Rolling Restart

During restarts that do not involve restarting the API process, Worker Processes on a Worker Node restart in a rolling fashion to minimize disruption and maintain as much processing capability as possible. **20% of running processes - with a minimum of one process - are restarted at a time.** All Worker Processes in a restarting batch must come up and report as **started** before the next batch is initiated. This rolling restart continues until all processes have restarted. If a Worker Process fails to restart, configurations will be rolled back.

If a restart does involve restarting the API process, Worker Processes on a Worker Node do not restart in a rolling fashion. All Worker Processes must stop before the API can restart, and once the API restarts, it starts up all Worker Processes.

2.6.5.7. Persistent Socket Connections

Distributed deployments use Unix domain sockets for inter-process communication (IPC) between a Leader Node's distributed processes and services. Multiple Unix domain sockets must exist (created by their respective process) on a Leader's disk. One of these domain socket files ensures that the metrics services operate correctly. On the host where the Leader Node is running, the default location for Unix domain socket files is the operating system temporary directory - for example, `/tmp`.

Many Linux distributions maintain a system cleaner service (such as systemd-tmpfiles) that removes files from the temporary directory periodically, such as every 10 days. If Cribl's sockets are removed, certain product pages (such as Monitoring) break.

You can protect the sockets by either blocking the cleaner or moving the Cribl socket files.

Block the Cleaner

You can stop the host operating system from cleaning socket files out of `/tmp/cribl-*` subdirectories. Using Amazon Linux 2 instances as an example:

1. Add a new `tmp.conf` file to `/etc/tmpfiles.d` with the line: `X /tmp/cribl-*`
2. Restart the system cleaner and reload its configuration using this command: `systemctl restart systemd-tmpfiles-clean.service`
3. Restart the Cribl server via the UI or command line.

Move the Sockets

Alternatively, you can move the Cribl socket files to a different, protected directory. The user that owns the `cribl` process must have the necessary permissions to write to this directory.

In Unix-like operating systems, the maximum length for Unix domain socket paths is 108 bytes. To minimize the path length, use a directory that is close to the root, such as `/var/tmp`.

Specify the directory in **Settings** at **Global** > **System** > **Distributed Settings** > **Leader Settings** > **Helper processes socket dir**.

2.6.6. Splunk App Deployment

Getting started with Cribl App for Splunk

Deploying Cribl App for Splunk

In an on-prem Splunk environment, you can install and configure Cribl Stream as a Splunk app (Cribl App for Splunk), on a Search Head. Note the following limitations:

- You **cannot** use Cribl App for Splunk in a Cribl Stream distributed deployment as a Leader, or as a managed Worker.
- You **cannot** install the app on Splunk Cloud or on a Splunk Heavy Forwarder. (See Single-Instance/Basic Deployment and Distributed Deployment for alternatives.)

Running on a Search Head (SH)

When running on an SH, Cribl Stream is set to **mode-searchhead**, the default mode for the app. It listens for **localhost traffic** generated by a custom command: `criblstream`. The command is used to forward search results to the Cribl Stream instance's TCP JSON input on port `10420`, but it's also capable of sending to any other Cribl Stream instance listening for TCP JSON.

Once received, data can be processed and forwarded to any of the supported Destinations. In addition, several out-of-the box saved searches are ready to run and send their results to Cribl with a single click.

Installing the Cribl App for Splunk on an SH

- Select an instance on which to install.
- Ensure that ports `10000`, `10420`, and `9000` are available. See the Requirements section for more info.
- Get the bits here, and install as a regular Splunk app.
- Restart the Splunk instance.
- Go to `https://<instance>/en-US/app/cribl` or `https://<instance>:9000`, and log in with Splunk **admin** role credentials.

Typical Use Cases for Search Head Mode

- Working with search results in a Cribl Stream pipeline.
- Sending search results to any Destination supported by Cribl Stream.

Commands and Examples

Below are some examples of how to use the `criblstream`, `criblencrypt`, and `cribldecrypt` commands available in this app.

The `cribldecrypt` command can decrypt events originating from multiple Stream Worker Groups or Edge Fleets **only** if you use the same key material on all Worker Groups and Fleets encrypting data.

Forward Search Results to a Cribl Stream Instance

The basic command for sending events to a Cribl Pipeline is `criblstream`, for example:

criblstream command

```
index="my_index" | criblstream dest=host:port
```

Copy

Search for a Value

Suppose that you want to search events for a particular value. Because events are encrypted, one approach would be to decrypt all events and then search the unencrypted events for your target value. But decrypting large numbers of events takes a long time, and is extremely inefficient.

Here's a better approach, taking advantage of the fact that the encrypted form of a value is always the same:

1. Encrypt the target value.
2. Search the events (which are still encrypted) for the encrypted form of the target value.
3. If you find events that contain the (encrypted) target value, decrypt them if desired.

You can combine all three steps in a single command:

Search the index for the encrypted target value and decrypt matching events

```
index="encrypted" \
[ | makeresults | eval encrypted_field="target_value" \
| criblencrypt field=encrypted_field keyclass= 0 \
| fields encrypted_field ] | cribldecrypt
```

Copy

The `criblencrypt` command requires a `field`, and either a `keyclass` (as shown above) or a `keyid`. For details, see Encryption.

Internal Fields

The Cribl App for Splunk adds the following internal fields to events:

- `_CRIBL_QUEUE`
- `_CRIBL_TCP_ROUTING`

The app forwards these fields to Splunk, because the `transforms.conf` file that ships with the app relies on them.

Beginning with version 4.0, only the Cribl App for Splunk adds these fields. Cribl Stream and Cribl Edge do not. (Earlier versions of Cribl Stream did add these fields, but neither Stream itself nor Splunk receivers have processed them in recent versions.)

2.6.7. Bootstrap Workers From Leader

Boot fully provisioned Workers

Cribl Stream Workers can completely provision themselves, directly from the Leader, upon initial boot. This means that a group of any number of Workers can launch and be fully functional within the cluster, in seconds.

How Does It Work?

A Cribl Stream Leader Node provides a bootstrap API endpoint, at `/init/install-worker.sh`, which returns a shell script. You can run this shell script on any supported machine (see Restrictions below), without Cribl Stream installed. This fully provisions the machine as a Worker Node.

Although you can specify the download URL when you execute the initial curl command, the Cribl Stream package is not downloaded until you generate the script via the API, and then execute it.

For additional information on using environment variables to set the hostname, see Setting the Hostname.

Requirements

All Worker Nodes' hosts must enable ongoing outbound communication to the Leader's port 4200, to enable the Leader to manage the Workers. While the bootstrap script runs, firewalls on each Worker's host must also allow outbound communication on the following ports:

- Port 443 to `https://cdn.cribl.io`.
- Port 443 to a Cribl.Cloud Leader.
- Port 9000 to an on-prem Leader.

If any of this traffic must go through a proxy, see System Proxy Configuration for configuration details. To anticipate and resolve edge cases not mentioned here, see Restrictions below.

Troubleshoot Root Access or SSL Errors

The script will install Cribl Stream into `/opt/cribl`, and will make system-level changes. For systems like Ubuntu, which don't allow direct root access, you'll need to use the `sudo` prefix when executing the script.

The script will create a user named `cribl` to install, own, and run Cribl Stream/Edge.

If you encounter errors of this form:
`ssl certificate problem: self signed certificate in certificate chain`
...add the `-k` flag to disable certificate validation.

UI Access

Cribl Stream admins can use the UI to concatenate and copy/paste the bootstrap script, automating several steps below. You can use an adjacent option to grab a script that updates a Worker's Group assignment.

To use these options, you must have the `admin` Role on a Distributed deployment's Leader Node, with a certain license tier. You must also create the Worker Groups before using the following instructions to add or reassign Workers to them.

Add New Worker Node

1. In the Cribl Stream sidebar, select **Workers**.
2. On the **Workers** page, select **Add/Update Worker Node**.
3. Select **Linux** > **Add** from the menu.
4. In the **Add Linux Worker** modal, the **Install package location** defaults to `Cribl CDN`. If desired, change this to `Download URL`. (For details about this option, see Add Download URL.)
5. As needed, correct the target **Group**, **User**, and **User Group**, as well as the **Leader or Outpost hostname/IP** (URI).
6. Copy the resulting script to your clipboard and select **Done** to close the modal.
7. Paste the script onto your Worker Node's command line and execute it.
    - As needed (see the note above), prepend `sudo` to the generated **Script** field's contents, and/or append the `-k` flag.

Update Existing Worker Node

You can also auto-generate a script that will update an existing Worker's Group assignment, and/or assign the Worker to a different Leader:

1. In the Cribl Stream sidebar, select **Workers**.
2. On the **Workers** page, select **Add/Update Worker Node**.
3. Select **Linux** > **Update** from the menu.
4. In the **Update Configuration on Linux Workers** modal, select the new target **Group** for this Worker Node.

5. As needed, correct or change the **Leader or Outpost hostname/IP** and/or **Leader port number**.
6. The **Script type** drop-down defaults to the `Environment variable` option for generating the script's command. If you're not relying on environment variables, change this to `CLI`.
7. Copy the resulting script to your clipboard and select **Done** to close the modal.
8. Paste the script onto your Worker Node's command line and execute it.
    - As needed (see the note above), prepend `sudo` to the generated **Script** field's contents, and/or append the `-k` flag.

API Spec

Request Format

```
GET http://<leader hostname or IP>:9000/init/install-worker.sh
```

Query Strings

String	Required?	Description
`token`	optional	Leader Node's shared secret (`authToken`). By default, this is set to a random string. You can find this secret in the Leader Node's **Distributed Settings** section. In versions older than 4.5.0, the default token was `criblmaster`. This token is still in use in instances upgraded from an earlier version.
`group`	optional	Name of the cluster's Worker Group. If not specified, falls back to `default`.
`download_url`	optional	Provide the complete URL to a Cribl Stream installation binary. This is especially useful if the Worker Nodes don't have access to the Internet to download from cribl.io.
`tag`	optional	When used in conjunction with Mapping Rules, enables you to specify the Worker Group you want the bootstrapped Worker to join. Multiple tags should be in the form `&tag=tag1&tag=tag2`.
`user`	optional	The user to run Cribl Stream. Defatuls to `cribl`.
`user_group`	optional	The user group that owns all files. Defaults to `cribl`.
`install_dir`	optional	The file path to install Cribl Stream. Defaults to `/opt/cribl`.

Example HTTP Request

```
GET http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-be
```

Copy

As of version 3.0, Cribl Stream's former "master" application components are renamed "leader." While some legacy terminology remains within CLI commands/options, configuration keys/values, and environment variables, this document will reflect that.

Example Response

```
#!/bin/sh

### START CRIBL LEADER TEMPLATE SETTINGS ###

CRIBL_MASTER_HOST="<Master FQDN/IP>"
CRIBL_AUTH_TOKEN="<Auth token string>"
CRIBL_VERSION="<Version>"
CRIBL_GROUP="<Default group preference>"
CRIBL_MASTER_PORT="<Master heartbeat port>"
CRIBL_DOWNLOAD_URL="<download url>"

### END CRIBL MASTER TEMPLATE SETTINGS ###

# Set defaults
checkrun() { $1 --help >/dev/null 2>/dev/null; }
faildep() { [ $? -eq 127 ] && echo "$1 not found" && exit 1; }
[ -z "${CRIBL_MASTER_HOST}" ] && echo "CRIBL_MASTER_HOST not set" && exit 1
CRIBL_INSTALL_DIR="${CRIBL_INSTALL_DIR:-/opt/cribl}"
CRIBL_MASTER_PORT="${CRIBL_MASTER_PORT:-4200}"
CRIBL_AUTH_TOKEN="${CRIBL_AUTH_TOKEN:-criblmaster}"
CRIBL_GROUP="${CRIBL_GROUP:-default}"
if [ -z "${CRIBL_DOWNLOAD_URL}" ]; then
    FILE="cribl-${CRIBL_VERSION}-linux-x64.tgz"
    CRIBL_DOWNLOAD_URL="https://cdn.cribl.io/dl/$(echo ${CRIBL_VERSION} | cut -d '-'
fi
UBUNTU=0
CENTOS=0
AMAZON=0

echo "Checking dependencies"
checkrun curl && faildep curl
checkrun adduser && faildep adduser
checkrun usermod && faildep usermod
BOOTSTART=1
SYSTEMCTL=1
checkrun systemctl && [ $? -eq 127 ] && BOOTSTART=0
checkrun update-rc.d && [ $? -eq 127 ] && BOOTSTART=0

echo "Checking OS version"
lsb_release -d 2>/dev/null | grep -i ubuntu && [ $? -eq  0 ] && UBUNTU=1
cat /etc/system-release 2>/dev/null | grep -i amazon && [ $? -eq 0 ] && AMAZON=1
```

```
echo "Creating cribl user"
if [ $UBUNTU -eq 1 ]; then
    adduser cribl --home /home/cribl --gecos "Cribl Stream User" --disabled-password
fi
if  [ $CENTOS -eq 1 ] || [ $AMAZON -eq 1 ]; then
    adduser cribl -d /home/cribl -c "Cribl Stream User" -m
    usermod -aG wheel cribl
fi

echo "Installing Cribl Stream"
mkdir -p ${CRIBL_INSTALL_DIR}
curl -Lso ./cribl.tar.gz "${CRIBL_DOWNLOAD_URL}"
tar xzf ./cribl.tar.gz -C ${CRIBL_INSTALL_DIR} --strip-components=1
rm -f ./cribl.tar.gz
chown -R cribl:cribl ${CRIBL_INSTALL_DIR}

if [ $BOOTSTART -eq 1 ]; then
    echo "Setting Cribl Stream to start on boot"
    ${CRIBL_INSTALL_DIR}/bin/cribl boot-start enable -u cribl
fi

mkdir -p ${CRIBL_INSTALL_DIR}/local/_system
cat <<-EOF > ${CRIBL_INSTALL_DIR}/local/_system/instance.yml
distributed:
  mode: worker
  master:
    host: ${CRIBL_MASTER_HOST}
    port: ${CRIBL_MASTER_PORT}
    authToken: ${CRIBL_AUTH_TOKEN}
    tls:
      disabled: true
  group: ${CRIBL_GROUP}
EOF

chown -R cribl:cribl ${CRIBL_INSTALL_DIR}
if [ $BOOTSTART -eq 1 ]; then
  service cribl start
else
  ${CRIBL_INSTALL_DIR}/bin/cribl start
fi
```

Copy

curl Option

An easy way of wrapping HTTP methods is to use the `curl` command. Here is an example, which uses a `GET` operation by default, with the same URL used in the above HTTP example:

```
curl http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-b
```

Copy

Check Requirements above to avoid/resolve port or ownership issues.

Chain Script Execution

The `GET` and `curl` procedures above will only output the contents of the script that needs executing - the script will still need to be manually executed.

However, you can automate that part, too, using a command like the one shown below. This passes the script's contents to the `bash` shell to immediately execute.

```
curl http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-b
```

Copy

As noted above, on Ubuntu and similar systems, you might need to insert `sudo` before the `bash`. If you don't have a bash shell available, you can pipe the command to `| sh -` instead.

Add Download URL

By default, the script gets configured to download the Cribl Stream package from the public Cribl repository. If you want to specify a different download location in the script, you'll use the `download_url` parameter.

To successfully execute the `curl` command while also specifying the download URL, you must enclose the URL in double quotes. The reason for this is that the `&` character that joins multiple HTTP parameters is interpreted by the shell as the operator to run commands in the background. Double-quoting the URL, as shown in this example, prevents this.

```
curl "http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-
```

Copy

curl Offline Option

To bootstrap Workers in an internet-disconnected (airgapped) environment, you can separate downloading the installation package from Cribl's repository versus running the `curl` command. Here's an example:

1. Download the installation package from Cribl's download page and place it in the desired directory, such as `/tmp` or `/root`.
2. Bootstrap the Worker from the file's current directory using the following command. Replace `<CRIBL_FILE>` with the full file name of the downloaded package, such as `cribl-4.10.1-45136dbb-linux-x64.tgz`:

```
curl "https://<mycriblleader.mydomain.ext>:9000/init/install-worker.sh?group=default&
```

Copy

Tag to Assign Workers to Worker Groups

Cribl Stream uses Mapping Rulesets to map Workers to Worker Groups. When you create a Worker from a bootstrap script, you can take advantage of Mapping Rulesets to specify which Worker Group you want the newly-created Worker to join. This is done by adding tags to the download URL, in the form `&tag=tag1&tag=tag2`.

Basic Example

Suppose you have a Worker Group, `Group420`, that you want bootstrapped Workers to join.

In the active Mapping Ruleset for `Group420`, create a new rule that maps any Worker with the tag `awseast1` to the `Group420` Worker Group. You can do this with a filter:

```
cribl.tags.include('awseast1')
```

Copy

Then, in the bootstrap script, add a download URL with a tag that matches the filter you just created. For example:

```
curl "http://<logstream_leader>:9000/init/install-worker.sh?tag=awseast1&token=<token
```

Copy

When you use the script to bootstrap a new Worker, the Worker will be assigned to `Group420`.

Advanced Example

Suppose you have four Worker Groups distributed between two regions and two platforms:

	AWS	Azure
Region 1	`Group01`	`Group02`
Region 2	`Group03`	`Group04`

Using two tags (one for region and one for platform) you can represent all four possible combinations, and thus all four Worker Groups:

- `tag=aws&tag=region1` maps to `Group01`.
- `tag=azure&tag=region1` maps to `Group02`.
- `tag=aws&tag=region2` maps to `Group03`.
- `tag=azure&tag=region2` maps to `Group04`.

For each Worker Group, you'll create a Mapping Rule with a filter for the appropriate tag combination to match. For example, this filter would match `Group04`:

```
cribl.tags.includes('azure') && cribl.tags.includes('region2')
```

Copy

Then you can use the tag combination in the download URL of a bootstrap script. For example, the tags in this download URL map to `Group04`:

```
http://<logstream_leader>:9000/init/install-worker.sh?tag=azure&tag=region2&token=<to
```

Copy

Any Worker created by the script with the above download URL will be assigned to `Group04`.

Status Codes

Status Code	Reason

2.6.8. Leader High Availability/Failover

To handle unexpected outages in on-prem Distributed deployments, Cribl Stream supports configuring standby Leaders for failover. In this High Availability (HA) scenario, if the primary Leader goes down, one of the other Leaders, called standby Leaders, take its place. This ensures continuity of operation, including functions that require the Leader, such as Collectors and Collector-based Sources, which can ingest data without interruption.

Cribl.Cloud handles High Availability and configuring standby Leaders automatically, requiring no configuration or action on your part.

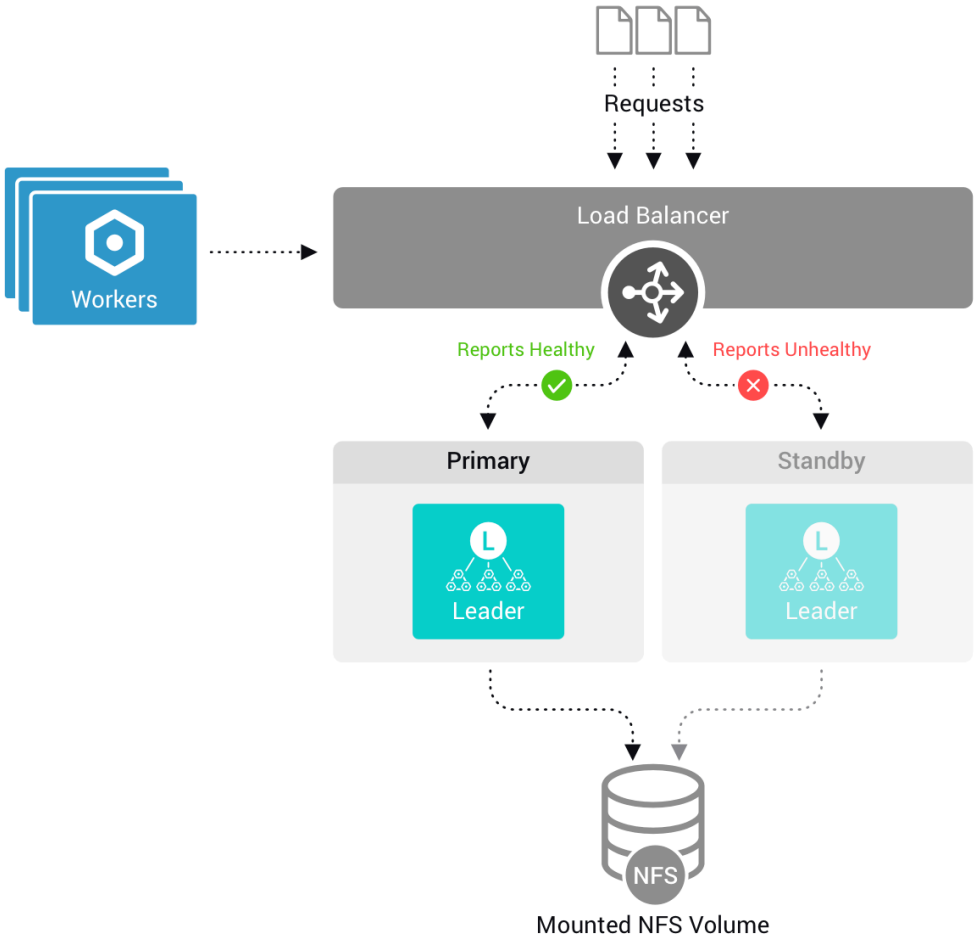
For license tiers that support configuring standby Leaders, see Cribl Pricing.

Primary and Standby Leaders

Only one Leader Node can be active at any given time, typically the primary Leader. A standby Leader (or Leaders) will become active only in the event of failover, when the primary Leader becomes unavailable.

The primary Leader stores its configurations and its git repository on the local disk. All changes to configurations and git commits are replicated to a shared failover Network File System (NFS) volume, and from it to the standby Leaders.

If the primary Leader becomes unavailable, a standby Leader takes its place. The standby then pulls the required configuration and the latest git commits from the failover volume. All Worker Nodes will connect to the new primary Leader, which retains the state and metrics of the old primary Leader



Leader High Availability/Failover Design

Leader Settings in High Availability Setups

When you first configure Leader High Availability, Cribl Stream will create a new `leader.yml` file in the local

2.6.8.1. High Availability Requirements

Before adding a standby Leader, ensure that you fulfill all requirements in this section.

Disk Space Requirements

Ensure that all Leaders, both the primary and the standby ones, have enough disk space available. You need enough space to hold the contents of the failover volume (except the `state` directory), all your configuration files, plus at least 10 GB in reserve for future use.

Leader Health Check Timeouts

Configure adequate timeout for Leader health checks to ensure the Leader has enough time to pull all the configuration and the git repository from the failover volume. Too low timeout could results in the Leader being terminated prematurely.

The recommended timeout scales with the number of Worker Groups and Fleets the Leader manages. The exact value depends on the size of the git repository, size of the `groups` directory, and the total number of configuration files. Approximate recommendations are:

- 2 minutes for smaller deployments
- 5 minutes for deployments with > 50 Worker Groups/Fleets
- 10 minutes for deployments with > 100 Worker Groups/Fleets

Git Configuration Requirements

If you have git timeout configured in `local/cribl.yml`, ensure that it is equal or greater than the system default of 10 minutes.

Auth Token Requirements

All Leaders must have matching auth tokens. If you configure a custom **Auth token**, make sure that all Leaders have that same token.

- In Cribl Stream, check that the values of **Settings** > **Global** > **Distributed Settings** > **Leader Settings** > **Auth token** match for each Leader.
- Or, from the filesystem, check that the values in instance.yml > `master` section > `authToken` values match for each Leader.

See How to Secure the Auth Token for the Leader Node for information on changing your auth token and allowed characters.

NFS Requirements

- On all Leader Nodes, use the latest version of the NFS client. **NFSv4 is required.**
- Ensure that the NFS volume has at least 100 GB available disk space.
- Ensure that the NFS volume's IOPS (Input/Output Operations per Second) is ≥ 200. (Lower IOPS values can cause excessive latency.)
- Ensure that ping/latency between the Leader Nodes and NFS is < 50 ms.
- Ensure your NFS system supports updating `mtime`.

You can validate the NFS latency using a tool like `ioping`. Navigate to the NFS mount, and enter the following command:

```
ioping .
```

Copy

For details on this particular option, see the ioping docs.

NFS Mount Option Requirements

The Leader Node will access large numbers of files whenever you use the UI or deploy configurations to Worker Nodes. When this happens, NFS's default behavior is to synchronize access time updates for those files, often across multiple availability zones and/or regions. To avoid the problematic latency that this can introduce, Cribl recommends that you add one of the following NFS mount options:

1. `relatime`: Update the access time only if it is more than 24 hours ago, or if the file is being created or modified. This allows you to track general file usage without introducing significant latency in Cribl Stream. To do the same for folders, add the `reldiratime` option.
2. `noatime`: Never update the access time. (Cribl Stream does not need access times to be updated to operate correctly.) This is the most performant option - but you will be unable to see which files are being accessed. To do the same for folders, add the `nodiratime` option.

Recommended NFS Configuration

If you are on AWS, we recommend the following:

- Use Amazon's Elastic File System (AWS EFS) for your NFS storage.

- Ensure that the user running Cribl Stream has read/write access to the mount point.
- Configure the EFS **Throughput mode** to `Enhanced` > `Elastic`.
- For details on NFS mount options, see Recommended NFS mount options.

For best performance, place your Leader Nodes in the same geographic region as the NFS storage. If the Leader and NFS are distant from each other, you might run into the following issues:

- Latency in UI and/or API access.
- Missing metrics between Leader restarts.
- Slower performance on data Collectors.

Load Balancer Requirements

Configure all Leaders behind a load balancer.

- Port `4200` must be exposed via a network load balancer.
- Port `9000` can be exposed via an application load balancer or network load balancer.

Active-Active (Proxy) Mode for HA Leaders

Both primary and standby Leaders use an Active-Active (Proxy) configuration to simplify load balancer (LB) routing. All Leaders can receive UI/API traffic on port `9000`, with the standby Leader automatically proxying any incoming requests to the elected primary. This architecture maintains crucial single-writer semantics, as only the primary Leader owns the state.

For proxy mode to work, the standby must be able to reach the primary over port `9000` to forward UI/API traffic. The control-plane connections on port `4200` are also proxied. Ensure firewall rules allow Leader-to-Leader communication on these paths.

Load Balancer Health Checks

Health checks over HTTP/HTTPS via the `/health` endpoint are only supported on port `9000`. Load balancers that support such health checks include:

- Amazon Web Services (AWS) Network Load Balancer (NLB). Suitable for TCP, UDP, and TLS traffic.
- AWS Application Load Balancer (ALB). Application-aware, suitable for HTTP/HTTPS traffic.
- HAProxy.
- NGINX Plus.

To ensure reliable load balancing for your Leader Nodes, configure health checks against the `/health` endpoint. For optimal performance and to avoid potential throttling, adhere to the following guidelines:

- **Polling frequency**: Set your load balancer to check the health endpoint every `60` seconds. This interval aligns with Cribl.Cloud's internal monitoring and prevents excessive API calls.
- **Avoid overly frequent checks**: Polling more often than every `60` seconds (for example, every `5` seconds) can overload the Leader's API Process.
- **Load balancer routing**: Configure your load balancer to direct traffic exclusively to Leader Nodes that return a `200` status code from the health check.

For detailed information on the `/health` endpoint query and response formats, refer to the Query the Health Endpoint documentation.

AWS Network Load Balancers

If you need to access the same target through a Network Load Balancer, use an IP-based target group and deactivate client IP preservation. For details, see:

- Why can an instance in a target group not reach itself via NLB?
- Why can't a target behind my Network Load Balancer connect to its own Network Load Balancer?

Source-Level Health Checks

For many HTTP-based Sources, you can enable a Source-level health check endpoint in the **Advanced Settings** tab. Load balancers can send periodic test requests to these endpoints, and a `200 OK` response indicates that the Source is healthy.

Frequent requests to Source-level health check endpoints can trigger throttling settings. In such cases, Cribl will return a `503 Service Unavailable` response, which can be misinterpreted as a service failure. A `503` response may indicate that the health checks are running too frequently instead of an actual service failure.

2.6.8.2. Configure Standby Leader Nodes

To prepare your deployment for High Availability, in the configuration for all Leaders, both primary and standby, you must specify the same failover volume - a shared Network File System (NFS) volume.

You can configure standby Leader Nodes in the following ways. These configuration options are similar to configuring the primary Leader Node:

- Using the UI
- Updating the YAML config file
- Using the Command Line
- Using Environment Variables

Remember, the `$CRIBL_VOLUME_DIR` environment variable overrides `$CRIBL_HOME`.

Use the UI

For all Leader Nodes, both primary and standby, configure the following:

1. In **Settings** > **Global** > **Distributed Settings** > **General Settings**, select **Mode**: `Leader`.
2. Next, on the **Leader Settings** left tab, select **Resiliency**: `Failover`. This exposes several additional fields.
3. In the **Failover volume** field, enter the NFS directory that the configurations and git commits will be replicated to. This directory must be outside of `$CRIBL_HOME`. One valid solution is to use `CRIBL_DIST_MASTER_FAILOVER_VOLUME=<shared_dir>`. See Using Environment Variables for more information.
4. Optionally, adjust the **Lease refresh period** setting, which determines how often the primary Leader tries to refresh its hold on the Lease file, and the **Missed refresh limit**, which determines how many Lease refresh periods elapse before standby Nodes attempt to promote themselves to primary.
   Increase these two values to avoid unnecessary failovers if your NFS has higher latency or is occasionally slow.
5. Select **Save** to restart.

When you save the **Resiliency**: `Failover` setting, further **Distributed Settings** changes via the UI will lock for both the primary and backup Leader. (This prevents errors in bootstrapping Workers due to incomplete token synchronization between the two Leaders.) However, you can still update each Leader's distributed settings by modifying its configuration files.

Use the YAML Config File

For all Leader Nodes, both primary and standby, configure the following in `$CRIBL_HOME/local/_system/instance.yml`, under the `distributed` section:

1. Set `resiliency` to `failover`.
2. Specify a volume for the NFS disk to automatically add to the Leader Failover cluster and trigger automated data migration.

$CRIBL_HOME/local/_system/instance.yml

```
distributed:
  mode: master
  master:
    host: <IP or 0.0.0.0>
    port: 4200
    resiliency: failover
    failover:
      volume: /path/to/nfs
```

Copy

Note that `instance.yml` configs are local, not on the shared NFS volume.

Use the Command Line

For all Leader Nodes, both primary and standby, use a command of this form:

```
./cribl mode-master -r failover -v /tmp/shared
```

For all options, see the CLI Reference.

Use Environment Variables

For all Leader Nodes, both primary and standby, configure environment variables listed in Environment Variables Reference: Adding Fallback Leaders.

Address HA Migration Timeouts

When enabling Leader High Availability (HA) on a production system with a lot of data, or when booting up new Leader Nodes, the process of moving data can sometimes take longer than expected. If the system runs out of time during this move, it might only copy part of the data, which can cause problems when the system restarts.

To mitigate the risk of incomplete data transfers, temporarily increase the system's transfer time prior to enabling HA. You can do this by adjusting a setting called `TimeoutSec` in the `cribl.service` configuration file. Locate the **Service** section and add or modify the `TimeoutSec` directive. For example, set it to `600` seconds (10 minutes):

```
[Service]
TimeoutSec=600
```

Copy

A successful data transfer will be confirmed by specific messages in the system's log, such as:

```
{"time":"2025-02-
17T22:01:46.295Z","cid":"api","channel":"ResiliencyConfigs","level":"info","message":"fa
migration is complete"}
```

A failed migration due to timeout might show logs ending with a system shutdown, like this:

```
{"time":"2025-02-
06T15:29:15.416Z","cid":"api","channel":"ShutdownMgr","level":"info","message":"Starting
shutdown ...","reason":"Got SIGTERM","timeout":0}
```

After the HA setup is complete, revert the `TimeoutSec` setting to its original value. Leaving the timeout extended indefinitely can lead to prolonged Leader downtime if a problem occurs during a future failover. Make sure to document the original setting before making changes to facilitate easy reversion.

2.6.8.3. Manage High Availability Deployment

Manage your High Availability deployment by monitoring Leaders, performing an upgrade, and disabling standby Leader Nodes.

Monitor Primary and Standby Leader Nodes

To view the status of your Leader Nodes:

1. On the top bar, select **Products**, and then select **Stream**.
2. In the sidebar, select **Monitoring**, then **System**, then **Leaders**.
3. In the **Leaders** table, you can see the status of all your Leader, including the roles they play (primary or standby).

This page is only accessible when you have a standby Leader configured.

Monitoring Leader Nodes

Upgrade a High Availability Deployment

Upgrading the Leader Node through the UI is not supported for Distributed deployments with a standby Leader. Instead, use the command line (CLI) to upgrade.

Follow this upgrade order:

1. Stop the standby Leader first.
2. Upgrade the standby Leader.
3. Start the standby Leader.
4. Stop the primary Leader; the upgraded standby will take over automatically through the HA failover mechanism.
5. Upgrade the now-stopped former primary Leader for your Stream or Edge deployment.
6. Start the former primary Leader. It will rejoin and assume the standby role (since the other Leader is currently active).

7. Upgrade Worker Nodes in rolling batches; do not upgrade them to a version newer than the Leader.

Disable a Standby Leader

Cribl recommends that you maintain a standby Leader to ensure continuity in your on-prem Distributed deployment. Should you decide to disable it, contact support to assist you.

2.6.9. Converting A Single Instance To Distributed Deployment

If you've configured a Cribl Stream Single-instance deployment and now want to promote it to Distributed, here are a couple of approaches to doing so, while retaining the configuration you've already created.

Simple Copy

We'll start with the simplest scenario, in which you plan to set up distributed mode with a single Worker Group. By default, this group will literally be named `default`. (We'll also explain how to extend this scenario.)

1. If you haven't already installed `git` (required for the Leader), do so as outlined here.
2. Stop the Cribl Stream server (`./cribl stop`).
3. Your single instance's configs are under Cribl Stream's `local/` subdirectory. So, copy `$CRIBL_HOME/local/cribl/*` to `$CRIBL_HOME/groups/default/local/cribl/`. This stages your configs for the `default` Worker Group.
4. Restart Cribl Stream, selecting **Distributed Mode**: `Leader`.
5. At this point, the Leader should have inherited your previous single-instance settings. Commit and deploy these settings to the `default` group, which should resume the same data processing that your single instance was executing.
6. If you want to replicate the same configs to additional Worker Groups, add those groups now on the **Worker Groups** page. Then repeat the preceding four steps, targeting the new subdirectories that have been created on the filesystem for the new groups.

Creating multiple Worker Groups requires certain license tiers. For details, see Pricing.

rsync

This alternative approach uses `rsync` to replicate your single-instance configs.

1. Use this command to rsync your single-instance configuration to each of your distributed Groups (replacing the `<group-name>` placeholder here):

```
rsync -a $cribl/local/cribl newmaster:$cribl/groups/<group-name>/local/
```

Copy

2. Restart Cribl Stream, selecting **Distributed Mode**: `Leader`.

3. Commit and deploy the new configuration.

2.7. Orchestrated Deployment

You can set up and control your Cribl Stream deployment by using orchestration tools such as Kubernetes.

See the following guides for setting up an orchestrated deployment:

- Kubernetes/Helm Deployment
- Kubernetes Worker Deployment
- Considerations for Cribl Stream on Kubernetes

2.7.1. Kubernetes/Helm Deployment

Boot fully provisioned Leader and Worker Nodes via Helm

Cribl's leader and workergroup Helm charts provide a fast way to deploy a distributed Cribl Stream environment to a Kubernetes cluster. For additional guidance, see Considerations for Cribl Stream on Kubernetes.

For up-to-date details on Prerequisites, Deploying, Upgrading, and running distributed on a Free License, see the Cribl Helm Charts repo's Readme.

K8s Leader Deployment

The rest of this page outlines how to deploy a Cribl Stream Leader Node (or Single instance) to AWS via Kubernetes, using a Cribl-provided Leader Helm chart.

This chart is a work in progress, provided as-is. Cribl expects to further develop and refine it.

Cribl recommends deploying the Leader on stable, highly available infrastructure, because of its role in coordinating all Worker instances.

Deprecation Notice

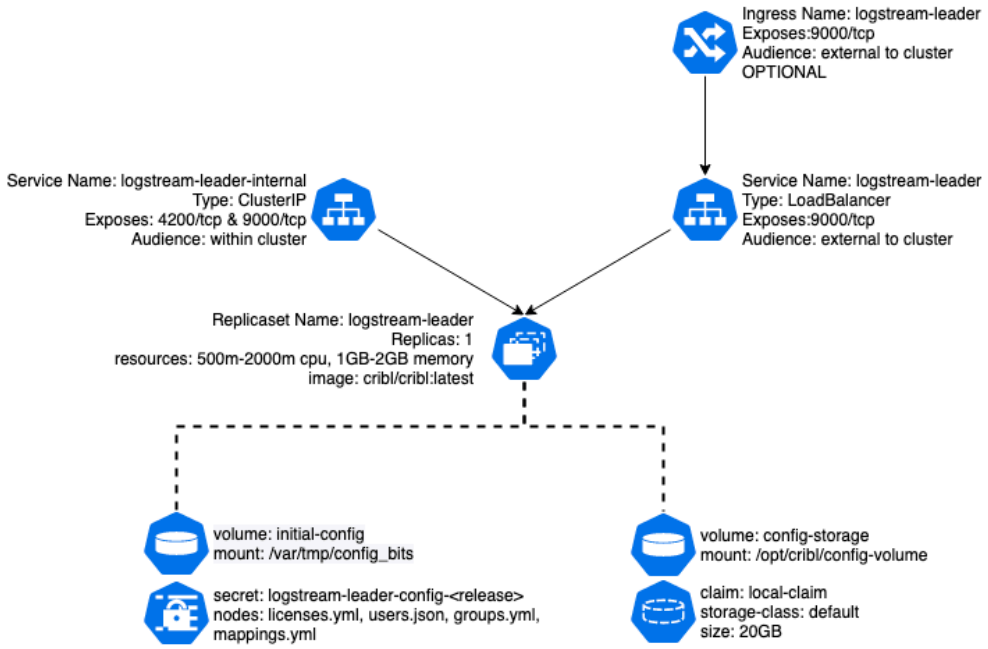
This chart replaces the `logstream-master` chart, which was deprecated as of v.2.9.9. See Migration below for instructions on migrating to this new chart to access features newly provided in this and future versions.

New Capabilities

- Supports Cribl Stream v.4.3.1.

Deployment

As built, Cribl's chart will deploy a Cribl Stream Leader server for Cribl Stream, consisting of a deployment, two services, and a number of persistent volumes.



Deployment schematic

Note that this chart creates two load-balanced services:

- The main one (named after the Helm release), which is intended as the primary service interface for users.
- The "internal" one (named `<helm-release>-internal`), which is intended for the worker-group-to-leader communication.

By default, this chart installs only a Cribl Stream Leader Node. To also deploy Cribl Stream Worker Groups/Fleets via Helm, you can use the Set Up Worker Groups/Mappings override described below.

You can also use Cribl's separate logstream-workergroup chart. For details, see Kubernetes Deployment: Worker Group/Fleet in this documentation.

AWS and Kubernetes Prerequisites

This section covers both general and specific prerequisites, with a bias toward the EKS-oriented approach that Cribl uses for its own deployments.

Set Up AWS CLI

Install the AWS CLI, version 2, following AWS' instructions.

Next, create or modify your `~/.aws/config` file to include (at least) a `[profile]` section with the following SSO (single-sign-on) details:

~/.aws/config

```
[profile <your-profile-name>]
sso_start_url = https://<your-domain>/start#/
sso_region = <your-AWS-SSO-region>
sso_account_id = <your-AWS-SSO-account-ID>
sso_role_name = <your-AWS-role-name>
region = <your-AWS-deployment-region>
```

Copy

Set Up kubectl

You will, of course, need `kubectl` set up on your local machine or VM. Follow Kubernetes' installation instructions.

Add a Cluster to Your kubeconfig File

You must modify your `~/.kube/config` file to instruct kubectl what cluster (context) to work with.

1. Run a command of this form: `aws --profile <profile-name> eks update-kubeconfig --name <cluster-name>`
   This should return a response like this: `Added new context arn:aws:eks:us-west-2:424242424242:cluster/<cluster-name> to /Users/<username>/.kube/config`
2. In the resulting `~/.kube/config` file's `args` section, as the new first child, insert the profile argument that you provided to the aws command. For example:

~/.kube/config

```
args:
- --profile=<profile-name>
- --region
[...]
```

Copy

3. Also change the `command: aws` pair to include the full path to the `aws` executable. This is usually in `/usr/local/bin`, in which case you'd insert: `command: /usr/local/bin/aws`.

This section of `~/.kube/config` should now look something like this:

~/.kube/config

```
args:
- --profile=<profile-name>
- --region
- us-west-2
- eks
- get-token
- --cluster-name
- lab
command: /usr/local/bin/aws
env:
- name: AWS_PROFILE
  value: <profile-name>
```

Copy

With these AWS and Kubernetes prerequisites completed, you're now set up to run `kubectl` commands against your cluster, as long as you have an active aws SSO login session.

Next, do the Helm setup.

Install Helm and Cribl Repo

1. You'll need Helm (preferably v.3.x) installed. Follow the instructions here.
2. Add Cribl's repo to Helm, using this command: `helm repo add cribl https://criblio.github.io/helm-charts/`

Persistent Storage

The chart requires persistent storage. It will use your default StorageClass, or (if you prefer) you can override `config.scName` with the name of a specific StorageClass to use.

Cribl has tested this chart primarily using AWS EBS storage, via the CSI EBS driver. The volumes are created as `ReadWriteOnce` claims. For details about storage classes, see Kubernetes' Storage Classes documentation.

AWS-Specific Notes

If you're running on EKS, Cribl highly recommends that you use Availability Zone-specific node groups. For details, see eksctl.io's Autoscaling documentation.

Do not allow a single node group to spans AZs. This can lead to trouble in mounting volumes, because EBS volumes are AZ-specific.

See other EKS-Specific Issues on our GitHub repo.

Configure the Chart's Values

You'll want to override some of the chart's default values. The easiest way is to copy this chart's default `values.yaml` file from our repo. save it locally, modify it, and install it in Helm:

1. Copy the **raw** contents of: https://github.com/criblio/helm-charts/blob/master/helm-chart-sources/logstream-leader/values.yaml
2. Save this as a local file, for example: `/bar/values.yaml`
3. Modify values as necessary (see Values to Override in the Helm Charts docs).
4. Install your updated values to Helm, using this command: `helm install -f /bar/values.yaml`

Match Versions

Cribl recommends that you use the same Cribl Stream version on Worker Nodes/Edge Nodes versus the Leader Node. So if, for any reason, you're not yet upgrading your Workers to the version in the Leader's default `values.yaml` > `criblImage.tag`, be sure to override that `criblImage.tag` value to match the version you're running on all Workers.

EKS-Specific Values

If you're deploying to EKS, many annotations are available for the load balancer. Set these as values for the `service.annotations` key. Internally, we typically use the annotations for logging to S3, like this:

values.yaml [excerpt]

```
service.beta.kubernetes.io/aws-load-balancer-access-log-enabled: "true"
service.beta.kubernetes.io/aws-load-balancer-access-log-emit-interval: "5"
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name: "<bucket
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix: "ELB"
```

Copy

For an exhaustive list of annotations you can use with AWS's Elastic Load Balancers, see the Kubernetes Service documentation.

Configure a Service Account

You can configure the Leader deployment to use a Kubernetes ServiceAccount for accessing cloud resources. This approach proves especially beneficial when you need to grant IAM roles directly to the Leader pod. A common use case is leveraging AWS KMS as the KMS provider without the need for access key IDs and secrets.

To configure a ServiceAccount for the Leader chart, you can modify the following options within your `values.yaml` file or by using the `--set` parameters during Helm deployment:

Value	Default	Description
`serviceAccount.create`	`false`	If set to `true`, the chart will create a new ServiceAccount specifically for the Leader Pod.
`serviceAccount.name`	`undefined`	Name of the ServiceAccount to use: • If `create: true` and this value is unset, the name defaults to `<release>-leader`. • If `create: true` and this value is set, the service account will use the provided name. • If `create: false`, this value **must** match the name of an existing ServiceAccount.
`serviceAccount.annotations`	`{}`	A key/value map of annotations to be added to the newly created ServiceAccount.

When `serviceAccount.create` is set to `false` and a `name` is provided, the chart assumes you are referencing an existing, pre-configured ServiceAccount. Conversely, if `serviceAccount.create` is `true`, the chart will generate a new ServiceAccount, either with a default name or the name you specify. When creating a service account, you can include annotations to declare the specific IAM roles that the service account should have permission to access.

Example configuration for creating a ServiceAccount with IAM role annotations:

```
serviceAccount:
  create: true
  name: "cribl-leader-sa"
  annotations:
    [eks.amazonaws.com/role-arn](https://eks.amazonaws.com/role-arn): "arn:aws:iam::1
```

Copy

Basic Chart Installation

With the above prerequisites and configuration completed, you're ready to install our chart to deploy a Cribl Stream Leader Node. Here are some example commands:

- To install the chart with the release name `logstream-leader`:
  `helm install logstream-leader cribl/logstream-leader`
- To install the chart using the storage class `ebs-sc`:
  `helm install logstream-leader cribl/logstream-leader --set config.scName='ebs-sc'`

Post-Install/Post-Upgrade

Cribl Stream will not automatically deploy changes to the Worker Nodes/Edge Nodes. You'll need to commit and deploy changes to all of your Worker Groups/Fleets.

Change the Configuration

If you don't override its default values, this Helm chart effectively creates a Single-instance deployment of Cribl Stream, using the standard container image. You can later configure Distributed mode, licensing, user passwords, etc., all from the Cribl Stream UI. However, you also have the option to change these configuration details upfront, by installing with value overrides. Here are some common examples.

Apply a License

If you have a Standard or Enterprise license, you can use the `config.license` parameter to add it as an override to your install:

```
helm install logstream-leader cribl/logstream-leader --set config.license="<long encoded license string redacted>"
```

Run Distributed on a Free License

If you do not specify a license with `config.license`, and you want to run Distributed, you'll need to go to Cribl Stream's **Settings** > **Licensing** page and accept the Free license. (The Free license allows one Worker Group/Fleet.)

If your Helm configuration includes the `config.groups` option, the Cribl Stream Leader Node will be configured as a distributed Leader. If you omit that option, it will be configured as a single instance. (You can later use Cribl Stream's **Settings** > **Distributed Settings** page to select **Mode:** `Leader`.)

Set the Admin Password

Normally, when you first install Cribl Stream and log into the UI, it prompts you to change the default admin password. You can skip the password-change challenge by setting your admin password via the `config.adminPassword` parameter:

```
helm install logstream-leader cribl/logstream-leader --set config.adminPassword="<new password>"
```

Set Up Worker Groups/Mappings

As mentioned above, the chart's default is to install a vanilla deployment of Cribl Stream. If you are deploying as a Leader, you can use the `config.groups` parameter to define the Worker Groups/Fleets you want created and mapped. Each group in the list you provide will be created as a Worker Group/Fleet, with a Mapping Rule to seek a tag with that Worker Group's name in it:

```
helm install logstream-leader cribl/logstream-leader --set config.groups=
{group1,group2,group3}
```

The example above will create three Worker Groups/Fleets/Fleets - `group1`, `group2`, and `group3` - and a Mapping Rule for each.

Migrate from the logstream-master Chart

Here is how to migrate from the deprecated `logstream-master` chart to `logstream-leader`.

Export Your Configuration

You'll need to "export" your data from the existing `logstream-master` pod. And first, you'll need to get the current pod's name, as well as its namespace. The easiest way to do this is to run `kubectl get pods -A` and then look pods that start with the release name you used when you ran `helm install`. For example, if you installed with the following command:

```
helm install ls-master cribl/logstream-master
```

...you'd look for a pod name that started with `ls-master`.

Once you've identified your pod and namespace, you can export your configuration using a combination of `kubectl` and `tar`:

```
kubectl exec <pod name> -n <namespace> -- bash -c "cd /opt/cribl/config-volume; tar c
```

Copy

This command executes the tar based back up of the config-volume, and outputs it to a local tar file (`cribl_backup.tar`).

"Re-Hydrate" the Backup on the logstream-leader Chart

Exploding the tarball onto the new persistent volume is a one-time event. Once the config volume is restored, you'll make changes to the config via the Cribl Stream UI or API. Either approach will change the config on disk, which you wouldn't want to overwrite the next time the pod restarts. You can manually re-hydrate the backup by installing the logstream-leader chart, and then running the following command:

```
cat cribl_backup.tar| kubectl -n <namespace> exec --stdin <pod name> -- bash -c "cd /
```

Copy

This will restore the data into the config volume (which is mounted as `/opt/cribl/config-volume`). If you want to double-check that, run:

```
kubectl -n <namespace> exec <pod name> -- bash -c "ls -alR /opt/cribl/config-volume"
```

Copy

After this, you want to **delete** the active pod, allowing the new one to come up with the restored configuration. To do this, you'd run the following `kubectl` command:

```
kubectl -n <namespace> delete <pod name>
```

Copy

This will cause the pod to exit, but the deployment will replace it with a new pod which will use the same config persistent volume.

Reconfigure the Worker Groups/Fleets

Now that you've got a new working leader chart, you need to tell the workers to connect to the new leader instead of to the old `logstream-master` instance. This is a simple `helm upgrade` operation. You'll need to use the same command string that you used to install, changing the word `install` to `upgrade`. But change the value of `config.host` to the new service that was created for the logstream-leader install. (You can change `config.host` either via the `--set` option or in the `values.yml` file.) For example, if you ran the `logstream-leader` install with the release name `ls-lead`, like this:

`helm install ls-lead -f <values file> cribl/logstream-leader`

...you'd run `kubectl get service -n <namespace> | grep ls-lead` to get the two services that it created, and you'll want the name of the one that ends in `-internal`. In this case, that name would be `ls-lead-leader-internal`.

Assume that for your workergroup install, you used a release name of `ls-wg1`, and a values file named `my-values.yml` with the following contents:

```
config:
  host: logstream-master-internal
  group: kubernetes
  token: <token>
  rejectSelfSignedCerts: 0
```

Copy

...then you'd replace the `host` value in this file with `ls-lead-leader-internal`, and then run:

```
helm upgrade ls-wg1 -f my-values.yml -n <namespace>
```

The upgrade **should** replace all the existing workergroup pods with newly reconfigured ones. However, if you notice any workergroup pods with an AGE value indicating that it was started before the upgrade command, simply kill those pods, and they will re-spawn with the new configuration.

Preload Configuration

The `extraConfigmapMounts` and `extraSecretMounts` options enable you to preload configuration files into the leader chart, via ConfigMaps and Secrets that you've created in your Kubernetes environment. However, because ConfigMaps and Secret mounts are read-only, you can't simply mount them into the configuration tree.

Therefore, you must mount them to a location outside of the `/opt/cribl` tree, and then copy the files into the tree at startup. This copying can be accomplished using environment variables, as we'll see below.

Both ConfigMaps and Secret mounts can be made writable, but the K8s documentation recommends against this.

Configuration Locations

The chart creates a single configuration volume claim, `config-storage`, which gets mounted as `/opt/cribl/config-volume`. All Worker Group/Fleet configuration lives under the `groups/` subdirectory. If you have a Worker Group/Fleet named `datacenter_a`, its configuration will live in `/opt/cribl/config-volume/groups/datacenter_a`. See Configuration Files section for details on file locations.

Use Environment Variables to Copy Files

The cribl container's `entrypoint.sh` file looks for up to 30 environment variables assumed to be shell-script snippets to execute before Cribl Stream startup (`CRIBL_BEFORE_START_CMD_[1-30]`). It also looks for up to 30 environment variables to execute after Cribl Stream startup (`CRIBL_AFTER_START_CMD_[1-30]`).

The variables in each set need to be in order, and cannot skip a number. (The `entrypoint.sh` script breaks the loop the first time it doesn't find an env var, so if you have `CRIBL_BEFORE_START_CMD_1` skipping to `CRIBL_BEFORE_START_CMD_3`, then `CRIBL_BEFORE_START_CMD_3` will not be executed.)

The chart uses this capability to inject the license and to set up groups. We'll use this same capability to copy our config files into place. So if you've provided the `config.license` and `config.groups` variables (occupying the first two slots), you'll need to start with `CRIBL_BEFORE_START_CMD_3`. In the examples below, we'll start with `CRIBL_BEFORE_START_CMD_3`, assuming that a `config.license` and `config.groups` have been set.

Figure Out Which Variable to Use

The easiest way to figure out which environment variable you need to use is to deploy the chart with all the options you plan to use (that is, to use the `helm install` command with options that you plan to use for your deployment). Then check the pod definition for `CRIBL_*` environment variables. For example, if you used the following install command:

```
% helm install lsms -f ../leader-values.yaml -n logstream-ht cribl/logstream-leader
```

Copy

You can now get the pod's name:

```
% kubectl get pods -n logstream-ht
NAME                                      READY   STATUS    RESTARTS   AGE
lsms-leader-659bfccdd6-xsz67              1/1     Running   0          52m
```

Copy

And then you can use `kubectl describe` to get the relevant environment variables:

```
% kubectl describe  pod/lsms-leader-659bfccdd6-xsz67 -n logstream-ht  | egrep "CRIBL_
CRIBL_BEFORE_START_CMD_1:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/licenses.yml ]
CRIBL_BEFORE_START_CMD_2:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/mappings.yml ]
CRIBL_AFTER_START_CMD_1:       [ ! -f $CRIBL_VOLUME_DIR/users_imported ] && sleep 20
```

Copy

From that, you can tell that we already have a `CRIBL_BEFORE_START_CMD_1` and `CRIBL_BEFORE_START_CMD_2`, so our next logical variable should be `CRIBL_BEFORE_START_CMD_3`.

Preload Scenario

Here's a preload scenario that includes a sample ConfigMap, `extraConfigmapMounts`, copy command, and copy-once flag.

The ConfigMap

Let's say we want to preconfigure a collector job in the `group1` Worker Group. The job will be called `InfrastructureLogs`, and it will read ELB logs from an S3 bucket. First, we'll need a `jobs.yml` file, like this:

```
InfrastructureLogs:
  type: collection
  ttl: 4h
  removeFields: []
  resumeOnBoot: false
  schedule: {}
  collector:
    conf:
      signatureVersion: v4
      enableAssumeRole: true
      recurse: true
      maxBatchSize: 10
      bucket: <my infrastructure logs bucket>
      path: /ELB/AWSLogs/${aws_acct_id}/elasticloadbalancing/${aws_region}/${_time:%Y
      region: us-west-2
      assumeRoleArn: arn:aws:iam::<accountid>:role/LogReadAssume
    destructive: false
    type: s3
  input:
    type: collection
    staleChannelFlushMs: 10000
    sendToRoutes: false
    preprocess:
      disabled: true
    throttleRatePerSec: "0"
    breakerRulesets:
      - AWS Ruleset
    pipeline: devnull
    output: devnull
```

Copy

We'll need this loaded into a ConfigMap object, so we'd run kubectl to create a ConfigMap from the directory where our `jobs.yml` file resides:

```
kubectl create configmap job-config --from-file <containing directory> -n <deployment namespace>
```

So if that file is in a directory called `./config-dir`, and we're deploying the leader chart into the `logstream` namespace, we'd create it like this:

```
kubectl create configmap job-config --from-file ./config-dir -n logstream
```

extraConfigmapMounts Config

In our `values.yaml` file, we need to specify the ConfigMap and where to mount it:

```
extraConfigmapMounts:
  - name: job-config
    configMap: job-config
    mountPath: /var/tmp/job-config
```

Copy

This example will mount the files in the ConfigMap into the pod's `/var/tmp/job-config` directory.

Copy the Config Files

You could simply define, in the `values.yaml` file (or via `--set`):

```
env:
  CRIBL_BEFORE_START_CMD_3: "cp /var/tmp/job-config /opt/cribl/config-volume/groups/g
```

Copy

However, there are two potential problems with that:

1. There is no guarantee that the destination directory tree will be there. (The first time a pod spins up, it won't be.)
2. If the pod has crashed and spun up anew, blindly copying will overwrite any changes previously made. This is rarely desirable behavior.

File Copying Pattern

Since we might want to copy multiple configuration files in one shot, it makes sense to use some sort of "flag file" to ensure that we copy the files only once. The script snippet to copy the `jobs.yaml` file looks like this, formatted for readability:

```
FLAG_FILE=/opt/cribl/config-volume/job-flag
if [ ! -e $FLAG_FILE ]; then
  mkdir -p /opt/cribl/config-volume/groups/group1/local/cribl # ensure the directory
  cp /var/tmp/job-config/jobs.yml /opt/cribl/config-volume/groups/group1/local/cribl
  touch $FLAG_FILE
fi
```

Copy

This looks to see if the file `/opt/cribl/config-volume/job-flag` exists, and if it doesn't, creates the directory tree, copies the config file(s), and then creates the job flag file. However, we need to format it a little differently to easily encompass it in the `env` variable:

```
env:
  CRIBL_BEFORE_START_CMD_3: "FLAG_FILE=/opt/cribl/config-volume/job-flag; if [ ! -e $
```

Copy

Once you run `helm install` with this in the `values.yaml` file, you can do `kubectl exec` on the pod to execute a shell:

```
kubectl exec -it <pod name> -- bash
```

...and then look at `/opt/cribl/config-volume/groups/group1/local/cribl/jobs.yml` to verify that it is in place.

Uninstall the Infrastructure

To spin down deployed pods, use the helm uninstall command - where `<release-name>` is the namespace you assigned when you installed the chart:

```
helm uninstall <release-name>
```

You can append the `--dry-run` flag to verify which releases will be uninstalled before actually uninstalling them:

```
helm uninstall <release-name> --dry-run
```

Known Issues

- Cribl's current architecture supports **only** TCP ports in Worker Groups'/Fleets' `service` > `ports` configuration. This restriction might be removed in future versions.
- The upgrade process from versions older than 2.4.0 creates an `initContainer`, which will run prior to any instance of the Cribl Stream pod. Because the coalescence operation will not overwrite existing data, this is not a functional problem. But depending on your persistent-volume setup, the `initContainer`'s precedence might cause pod restarts to take additional time while waiting for the volume claims to release. The only upgrade path that will have this issue is v.2.3.* -> v.2.4.0. In the next iteration, we'll remove the `initContainer` from the upgrade path.

2.7.2. Kubernetes Worker Deployment

Boot a fully provisioned Worker Group via Helm.

This page outlines how to deploy a Cribl Stream Worker Group to AWS via Kubernetes, using a Cribl-provided Helm chart. For additional guidance, see Considerations for Cribl Stream on Kubernetes.

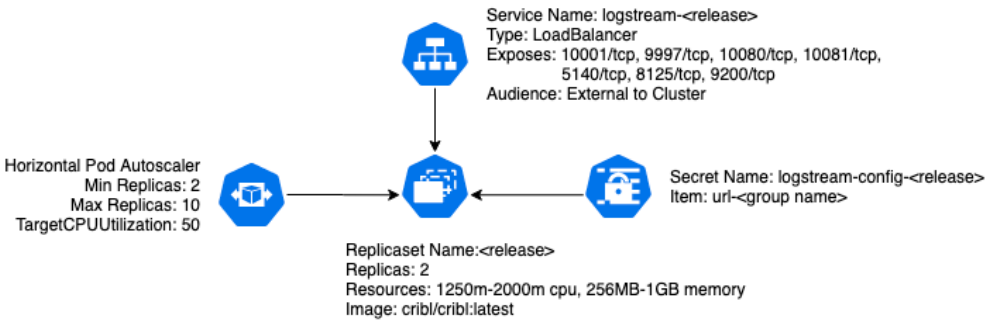
This chart will deploy only a Cribl Stream Worker Group, whose functioning depends on the presence of a Cribl Stream Leader Node. To deploy the Leader, see Kubernetes Leader Deployment.

New Capabilities

- Supports Cribl Stream v.4.3.1.

Deployment

As built, Cribl's chart will deploy a simple Worker Group for Cribl Stream, consisting of a deployment, a service, a horizontal pod autoscaler configuration, and a secret used for configuration.



Deployment schematic

AWS and Kubernetes Prerequisites

This section covers both general and specific prerequisites, with a bias toward the EKS-oriented approach that Cribl uses for its own deployments.

Set Up AWS CLI

Install the AWS CLI, version 2, following AWS' instructions.

Next, create or modify your `~/.aws/config` file to include (at least) a `[profile]` section with the following SSO (single-sign-on) details:

~/.aws/config

```
[profile <your-profile-name>]
sso_start_url = https://<your-domain>/start#/
sso_region = <your-AWS-SSO-region>
sso_account_id = <your-AWS-SSO-account-ID>
sso_role_name = <your-AWS-role-name>
region = <your-AWS-deployment-region>
```

Copy

Set Up `kubectl`

You will, of course, need `kubectl` set up on your local machine or VM. Follow Kubernetes' installation instructions.

Add a Cluster to Your `kubeconfig` File

You must modify your `~/.kube/config` file to instruct `kubectl` what cluster (context) to work with.

1. Run a command using this format: `aws --profile <profile-name> eks update-kubeconfig --name <cluster-name>`
   This should return a response like this: `Added new context arn:aws:eks:us-west-2:424242424242:cluster/<cluster-name> to /Users/<username>/.kube/config`
2. In the resulting `~/.kube/config` file's `args` section, as the new first child, insert the profile argument that you provided to the `aws` command. For example:

/.kube/config

```
args:
- --profile=<profile-name>
- --region
[...]
```

Copy

3. Also change the `command: aws` pair to include the full path to the `aws` executable. This is usually in `/usr/local/bin`, in which case you'd insert: `command: /usr/local/bin/aws`.

This section of `~/.kube/config` should now look something like this:

~/.kube/config

```
args:
- --profile=<profile-name>
- --region
- us-west-2
- eks
- get-token
- --cluster-name
- lab
command: /usr/local/bin/aws
env:
- name: AWS_PROFILE
  value: <profile-name>
```

Copy

With these AWS and Kubernetes prerequisites completed, you're now set up to run `kubectl` commands against your cluster, as long as you have an active aws SSO login session.

Next, install Helm and the Cribl repository.

Install Helm and Cribl Repo

1. You'll need Helm (preferably v.3.x) installed. Follow the instructions here.
2. Add Cribl's repo to Helm, using this command: `helm repo add cribl https://criblio.github.io/helm-charts/`
3. Display the default values available to configure Cribl's `logstream-workergroup` chart: `helm show values cribl/logstream-workergroup`

Configure the Chart's Values

You'll want to override some of the values you've just displayed. The easiest way is to copy this chart's default `values.yaml` file from our repo. save it locally, modify it, and install it in Helm:

1. Copy the **raw** contents of: https://github.com/criblio/helm-charts/blob/master/helm-chart-sources/logstream-workergroup/values.yaml
2. Save this as a local file, for example: `/foo/values.yaml`
3. Modify values as necessary (see Values to Override in the Cribl Helm Charts docs). To see the full scope of values available, run: `helm show values cribl/logstream-workergroup`.
4. Install your updated values to Helm, using this command: `helm install -f /foo/values.yaml`

Match Versions

Cribl recommends that you use the same Cribl Stream version on Leader Nodes versus Worker Group/Fleet Nodes. So, if you're not yet upgrading your Leader to the version in the current `values.yaml` > `criblImage.tag`, be sure to override that `criblImage.tag` value to match the version you're running on the Leader.

Set Worker Processes Statically

For all container-based Worker Groups, Cribl recommends that you directly specify the Worker **Process count** using a positive integer (like `+3`). This convention prevents the overprovisioning of Worker Processes that can occur with dynamic values (like the default `-2` setting).

To access **Process count** in Stream, from a Worker group select **Group Settings** > **System** > **Worker Processes** (Distributed deployments) or **Settings** > **System** > **Service Processes** (Single-instance deployments).

Set a Maximum Number of Worker Processes

To provision a Kubernetes container with a specific number of Worker Processes, use the `CRIBL_MAX_WORKERS` environment variable. Setting a maximum amount of Worker Processes on a Kubernetes node helps to prevent overloading the node with more CPUs than it can handle.

Use `CRIBL_MAX_WORKERS` to set an in-product limit on Worker Processes, regardless of the underlying environment. This prevents Cribl Stream from spawning more Worker Processes than specified, even if the Kubernetes node has available resources.

See Sizing and Scaling for more information on this environment variable and example configurations.

Kubernetes CPU Limit Configuration

For Cribl Stream Worker Groups, the Helm chart ensures that the number of Worker Processes is determined by the CPU resources allocated to the pod, rather than the physical CPU count on the host.

To achieve this, it sets the `CRIBL_K8S_CPU_LIMIT` environment variable to the `resources.limits.cpu` setting on the pod. Cribl Stream then uses the value as a CPU limit to determine how many Worker Processes to start.

We recommend maintaining the default configuration of the `CRIBL_K8S_CPU_LIMIT` environment variable even though you can statically set the number of Worker Processes. This prevents performance problems like resource contention and pod instability from exceeding CPU limits.

Install the Chart

With the above prerequisites and configuration completed, you're ready to install our chart to deploy a Cribl Stream Worker Group/Fleet. Here are some example commands:

- To install the chart with the release name `logstream-wg`:
  `helm install logstream-wg cribl/logstream-workergroup`
- To install the chart using the Cribl Stream Leader `logstream.lab.cribl.io`:
  `helm install logstream-wg cribl/logstream-workergroup --set config.host='logstream.lab.cribl.io`
- To install the chart using the Cribl Stream Leader `logstream.lab.cribl.io` in the namespace `cribl-helm`:
  `helm install logstream-wg cribl/logstream-workergroup --set config.host='logstream.lab.cribl.io' -n cribl-helm`

Upgrade

You upgrade using the `helm upgrade` command. But it's important to ensure that your Helm repository cache is up to date, so first issue this command:

```
helm repo update
```

Copy

After this step, invoke:

```
helm upgrade <release> -n <namespace> cribl/logstream-workergroup
```

Copy

For the example above, where the release is `logstream-wg` and is installed in the `cribl-helm` namespace, the command would be:

```
helm upgrade logstream-wg -n cribl-helm cribl/logstream-workergroup
```

Copy

This Helm chart's upgrade is idempotent, so you can use the upgrade mechanism to upgrade the chart, but you can also use it to change its configuration (as outlined in Change the Configuration).

Optional: Kubernetes API Access

Versions 2.4.0+ include access mechanisms for Worker Groups to access the Kubernetes API. The `values.yaml` file provides three relevant options:

- `rbac.create` - Enables the creation of a Service Account, Cluster Role, and Role Binding (which binds the first two together) for the release.
- `rbac.resources` - Specifies the Kubernetes API resources that will be available to the release.
- `rbac.verbs` - Specifies the API verbs that will be available to the release.
- `rbac.extraRules` - Additional rulesets for the cluster role.

For more information on the verbs and resources available, see Kubernetes' Using RBAC Authorization documentation.

Change the Configuration

Once you've installed a release, you can get its `values.yaml` file by using the `helm get values` command. For example, assuming a release name of `logstream-wg`, you could use this command:

```
helm get values logstream-wg -o yaml > values.yaml
```

This will retrieve a local `values.yaml` file containing the values in the running release, including any values that you overrode when you installed the release.

You can now make changes to this local `values.yaml` file, and then use the `helm upgrade` operation to "upgrade" the release with the new configuration.

For example, assume you wanted to add an additional TCP-based syslog port, listening on port 5141, to the existing `logstream-wg` release. In the `values.yaml` file's `service` > `ports` section, you'd add the three key-value pairs shown below:

values.yaml (excerpt)

```
service:
  [...]

  ports:
  [...]
  - name: syslog
    port: 5141
    protocol: TCP
```

Copy

Then you'd run:

```
helm upgrade logstream-wg cribl/logstream-workergroup -f values.yaml
```

Remember, if you installed in a namespace, you need to include the `-n <namespace>` option to any `helm` command. You'll still have to create the source in your Cribl Stream Leader, and commit and deploy it to your Worker Group/Fleet.

Use Persistent Storage for Persistent Queueing

The `extraVolumeMounts` option makes it feasible to use persistent volumes for Cribl Stream persistent queueing. However, Cribl does not recommend this combination - there is variability in persistent-storage implementations, and this variability can lead to problems in scaling Worker Groups/Fleets. However, if you choose to implement persistent volumes for queueing, please consider these suggestions:

1. Use Elastic Block Store (EBS block) storage for the `CRIBL\_VOLUME\_DIR`. EBS storage is resilient and has minimal risk of data loss or corruption. While shared storage solutions like Elastic File System (EFS) can be flexible, they may not provide the necessary performance for persistent queuing. EBS volumes, while less flexible, offer better performance and reliability.
2. Understand your Kubernetes networking topology, and how that topology interacts with your persistent-storage driver. (For example, if you're on AWS, ensure that your volumes are available in all Availability Zones that your nodes might run in.)
3. Monitor the Worker Group/Fleet pods for volume issues. The faster you can see such issues and react, the more likely that you'll be able to resolve them.

Load Balancing on Self-Managed Kubernetes

The Cribl Helm chart currently specifies the service type as `LoadBalancer`. This service type is designed to work with cloud providers that offer managed load balancers.

If you are deploying Cribl on a self-managed Kubernetes environment, your Kubernetes cluster will need to provide a load balancer to route and distribute data to Workers. You must modify the Helm chart to use a different service type if a load balancer is not provided by your Kubernetes environment.

If you are deploying Cribl on a self-managed Kubernetes environment, you have the following options:

- **Use an external load balancer**: If your Kubernetes environment provides a load balancer, the Helm chart can use the `LoadBalancer` service type.
- **Use an `Ingress controller`**: The Helm chart includes an optional configuration for using an `Ingress controller.` This feature is disabled by default. When enabled, the Helm chart will create an Ingress resource, allowing you to route traffic to your Cribl services through your chosen Ingress controller (for example, NGINX Ingress Controller, Traefik).

For details, see Services, Load Balancing, and Networking.

Modifying the Helm chart to use a different service type requires careful consideration of your network architecture and infrastructure.

Uninstall the Infrastructure

To spin down deployed pods, use the helm uninstall command - where `<release-name>` is the namespace you assigned when you installed the chart:

```
helm uninstall <release-name>
```

You can append the `--dry-run` flag to verify which releases will be uninstalled before actually uninstalling them:

```
helm uninstall <release-name> --dry-run
```

Notes on This Example

- If you installed in a namespace, you'll need to include the `-n <namespace>` option in any `helm` command.
- In the above syslog example, you'd still need to configure a corresponding syslog Source in your Cribl Stream Leader, and then commit and deploy it to your Worker Group(s)/Fleet(s).

Known Issues

- The chart currently supports **only** TCP ports in `service > ports` for Worker Groups/Fleets. This limitation might be removed in future versions.
- See EKS-specific issues on our GitHub repo.

2.7.3. Considerations For Cribl Stream On Kubernetes

Kubernetes can be a powerful platform for deploying Cribl Stream, offering scalability and flexibility. However, there are specific use cases where Kubernetes might not be the optimal choice. This guide will explore the factors to consider when deciding whether Kubernetes is the right fit for your Cribl Stream deployment. We'll also discuss workarounds and best practices to ensure a smooth integration.

Understanding the Challenges of Dynamic Workloads and HP Autoscaling

To maximize the benefits of Cribl Stream's integration with Kubernetes, it's essential to understand its optimal use cases and potential limitations.

Optimal Use Cases

Cribl Stream, when deployed on Kubernetes, excels at handling pull-based Sources, where data is readily available for collection. This is because Kubernetes's containerized environment and dynamic scaling capabilities align well with data Sources where collection is on demand or scheduled.

By actively pulling data from Sources like Collector Sources, Cribl Stream running on Kubernetes ensures timely data ingestion, and minimizes the risk of data loss. This proactive approach also enables Cribl Stream to handle a large number of Sources efficiently, making it ideal for environments with many log files or syslog servers.

Less-Ideal Use Cases

Cribl Stream Workers running on Kubernetes can effectively handle data streams from push-based Sources, provided they are provisioned appropriately with fixed resources and sufficient overhead to accommodate average and peak data volumes. However, challenges can arise when dealing with senders that experience frequent, significant fluctuations in data volume, especially when using auto-scaling with scale-in policies.

Kubernetes environments, with their dynamic scaling capabilities, can be powerful tools to accommodate these fluctuations. However, the ephemeral nature of Kubernetes pods, particularly when managed by Horizontal Pod Autoscaler (HPA), can introduce complexities. HPA automatically scales the number of pods based on defined metrics, such as CPU utilization. While this is beneficial for resource optimization, it can lead to unexpected behavior in certain scenarios.

When a sudden spike in data volume occurs, HPA may scale up by creating new pods to handle the increased load. However, if the traffic spike is short-lived, these newly created pods may be terminated prematurely to conserve resources. This can disrupt ongoing data processing tasks, leading to data loss or processing delays. Additionally, the constant creation and destruction of pods can introduce latency and overhead, impacting overall system performance.

Consider the following when deploying Cribl Stream within Kubernetes environments.

General Kubernetes Best Practices

When managing large Cribl deployments within Kubernetes, it's essential to be aware of potential limitations inherent to large Kubernetes clusters. To ensure optimal performance and stability, consider the best practices outlined in the Kubernetes documentation. These guidelines address critical aspects like resource management, network configuration, and monitoring, which become increasingly important as your Kubernetes environment scales.

Tuning Cribl for Kubernetes

Cribl Stream leverages Worker Processes to efficiently handle data ingestion and processing tasks. By configuring the maximum number of Worker Processes per pod, you can optimize resource allocation and prevent overloading.

Kubernetes containers use resource limits that map to shares of the CPU and Memory. Although a Kubernetes container has visibility of the entire host CPU, it can only use the resources allocated to the pod based on the configured limits. For details, see Resource Management for Pods and Containers.

To ensure optimal performance and avoid overloading your Kubernetes pods, consider the following:

- **Set the Maximum Worker Processes:**
  Use the `--set env.CRIBL_MAX_WORKERS=NUMBER_WORKER_PROCESSES` configuration setting to specify the maximum number of Worker Processes per pod. While Cribl cannot directly determine the number of CPUs within a container, this setting allows you to increase the maximum number of Worker Processes, helping to optimize resource allocation and avoid excessive load. This setting helps balance resource utilization and prevent excessive load on individual pods.
- **Determine the Optimal Worker Process Count:**
  - **Self-Hosted Environments:** Calculate the available CPU cores on your host nodes and set the Worker Process count accordingly. To avoid overloading your nodes, it's recommended to deploy a limited number of Cribl Stream pods per node. A 1:1 mapping, where each node hosts a single Cribl Stream pod, is often a good starting point. For details, see Sizing and Scaling.
  - **Cloud Environments:** Refer to your cloud provider's instance type specifications or leverage auto-scaling mechanisms like HPA to dynamically adjust the number of pods based on demand.

Security Context Considerations for OpenShift and Non-Root Environments

To make sure your deployments run smoothly on OpenShift or non-root environments, you'll need to configure the security context through the `values.yaml` file. The `values.yaml` file allows you to override default settings and customize your deployment without altering the core Helm chart templates.

To add or update the `values.yaml` file:

1. Locate the `values.yaml` file.
2. Modify the `securityContext` section.
3. Customize the capabilities to match your environment's needs:

```
securityContext:
  capabilities:
    add:
      - CAP_NET_BIND_SERVICE
      - CAP_SYS_PTRACE
      - CAP_DAC_READ_SEARCH
      - CAP_NET_ADMIN
```

Copy

4. Install your updated values to Helm, using this command: `helm install -f /bar/values.yaml`. For a description of the capabilities, see Set Capabilities for Cribl Stream.

Persistent Queue Considerations for Kubernetes Deployments

Persistent queuing in Cribl Stream on Kubernetes requires careful consideration of autoscaling and storage performance.

Autoscaling

To ensure data durability and prevent unexpected data loss, it's crucial to carefully manage autoscaling when using persistent queues in Cribl Stream on Kubernetes.

- **Avoid Automatic Scale-In:** Disable automatic scale-in or limit it to manual intervention. This prevents unexpected pod terminations, which can lead to data loss.
- **`StatefulSet` Deployment:** Use the `StatefulSet` deployment mode of the Worker Group chart. `StatefulSet` ensures that pods are assigned to specific volumes, maintaining data consistency.

Storage Performance

To ensure optimal performance and data integrity, use high-performance storage for persistent queues in Cribl Stream on Kubernetes.

High-Performance Storage: Use Elastic Block Store (EBS block) storage for the `CRIBL_VOLUME_DIR`. EBS storage is resilient and has minimal risk of data loss or corruption. EBS volumes can be moved to a new Kubernetes host (in the same Availability Zone (AZ)) in the event of a host failure or termination.

Avoid Shared Storage: While shared storage solutions like Elastic File System (EFS) can be flexible, they may not provide the necessary performance for persistent queuing. EBS volumes, while less flexible, offer better

performance and reliability.

Additional Considerations

- **Volume Mount Configuration:** Ensure `CRIBL_VOLUME_DIR` is set, and set to the mount point of the PVC defined in the volumeClaimTemplates configuration. Enabling `CRIBL_VOLUME_DIR` ensures that the worker GUID is consistent on each startup.
- **Monitoring:** Continuously monitor PQ usage, including storage utilization and performance metrics, to identify potential bottlenecks.
- **`ExtraVolumeMounts`:** While the Kubernetes `extraVolumeMounts` option allows for persistent volume usage, Cribl doesn't recommend it due to potential issues arising from variations in persistent storage implementations. Additionally, `extraVolumeMounts` is a requirement for non-root Kubernetes environments like Anthos and OpenShift, which can make the setup more complex.

2.8. Connect Nodes To Leader Through Cribl Outpost

Simplify Node-to-Leader communication with Cribl Outpost.

Cribl Outpost simplifies management of deployments by acting as a relay between individual Edge Nodes or Stream Workers and the Leader Node.

Once you establish an Outpost and configure it to accept traffic from Edge Nodes or Stream Workers, control plane communication between those Nodes and the Leader will pass through the Outpost.

Cribl Outpost is ideal in situations where you can't establish direct communication between Edge Nodes or Workers and the Leader, for example due to restrictions in opening up firewall ports.

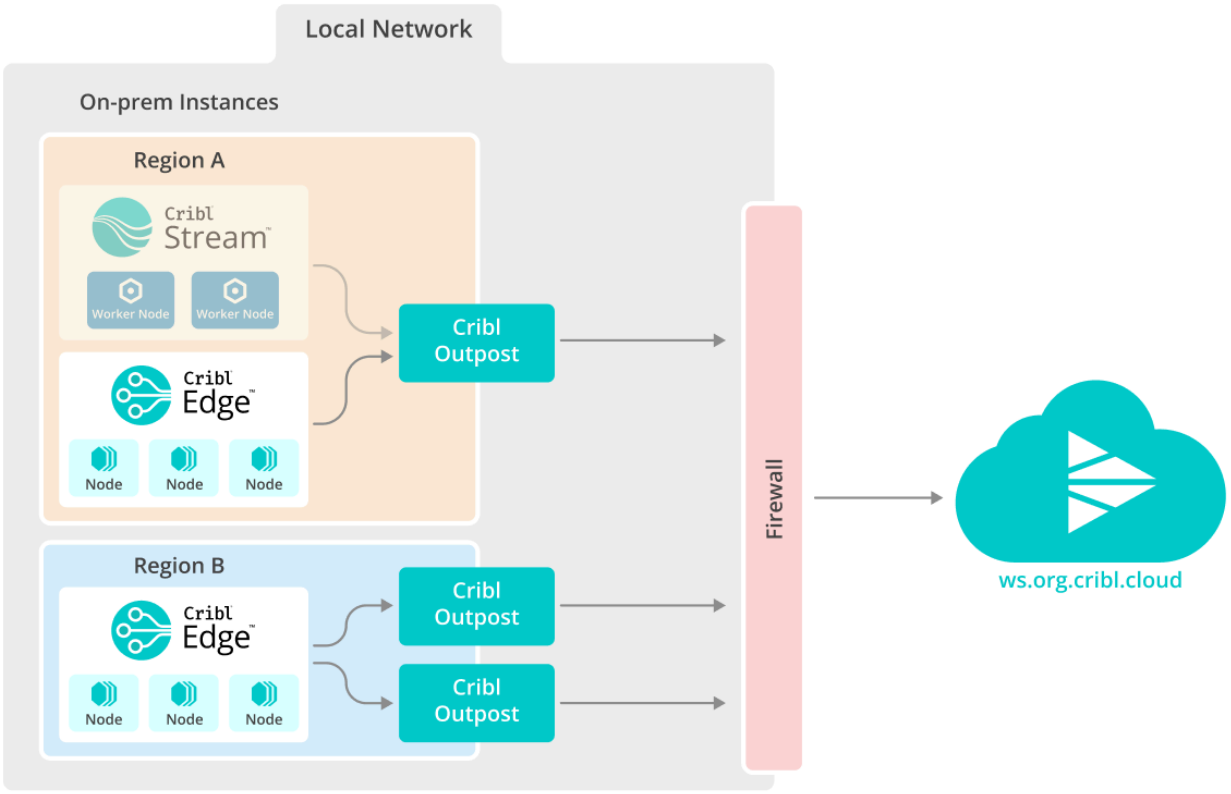
Using an Outpost limits the need to set up third-party proxies and simplifies maintenance by reducing the complexity of firewall setups.

Cribl Outpost requires an Enterprise plan or Enterprise license. For details, see Pricing.

If you connect an Outpost to a Leader that does not have a proper plan or license, the Outpost node will close the listener for communication incoming from Edge Nodes or Workers, but will be shown as healthy on the Leader.

How Outpost Works

An Outpost is a special type of Node that rests between Edge Nodes or Stream Workers and the Leader.



Overview of how Outpost connects Leader with Edge Nodes and Stream Workers

2.8.1. Set Up Cribl Outpost

Create a Cribl Outpost to relay communication between Worker Nodes and the Leader.

Prerequisites

Cribl Outpost requires an Enterprise plan or Enterprise license. For details, see Pricing.

You can install Cribl Outpost on Linux only. Do not run an Outpost Node on the same host as a Stream Worker to avoid both Nodes competing for resources.

Create an Outpost

There are two ways of adding a new Outpost Node:

- from a binary file
- using the bootstrap script provided in the UI

Add an Outpost from a Binary File

To add a new Outpost Node using a downloaded binary file:

1. Go to the Cribl Download page and set the **Software** drop-down to `Cribl Suite (Edge and Stream)` for your processor architecture.
2. Select **Download now** to get the `.tgz` archive.
3. Un-tar the install package in your directory of choice:

```
cd /opt/
tar xvzf cribl-<version>-<build>-<arch>.tgz
```

Copy

4. Set the installation mode to `mode-outpost` to connect the Outpost to the designated Leader and allow Stream Workers to connect to it in turn:

```
/opt/cribl/bin/cribl mode-outpost /
   -H <leader-hostname-or-IP> /
   -p <port> /
   -u <token> /
   -S 1
```

Copy

`<token>` must match the Leader token, to ensure a connection with the Leader.
In Cribl.Cloud, get the token for your Workspace from the Worker Node modal (**Add/Update Worker Node**) from the **Auth token** field.
In an on-prem deployment, get the token from **Settings** > **Global** > **System** > **Distributed Settings** > **Leader Settings** > **Auth token**.

5. Configure the installation as a service and start it:

```
sudo /opt/cribl/bin/cribl boot-start enable -m systemd -u cribl
sudo systemctl restart cribl-outpost.service
```

Copy

At this point your Cribl Outpost is ready for use and you can start configuring Worker Nodes to connect to it.

Add an Outpost with a Bootstrap Script

To add a new Outpost Node using a CLI bootstrap cript:

1. From the sidebar, select **Outpost**, then select an Outpost Group.
2. Select **Add/Update Outpost Node**, then select **Add**.
3. Fill and modify the deployment option fields:
   - The **Install package location** defaults to `Cribl CDN`. If desired, change this to `Download URL`.
   - As needed, correct the target **Outpost Group**, as well as the **Leader hostname/IP** (URI).
   - As needed, correct the **User** and **User Group** to run Cribl as.
   - As needed, correct the **Installation directory**.
4. Optionally, add **Tags** that you can use for filtering and grouping Outposts. Use a tab or hard return between (arbitrary) tag names.
5. Copy the resulting script to your clipboard.

Finally, to add the Outpost, paste the script onto the command line where you want to set up the node, and execute it.

If your script successfully added the Outpost, you will see it in the **Outposts** list for your selected group.

Connect Nodes to an Outpost

Worker Nodes see Outpost in the same way as a Leader Node. This means that, to connect a Node to an Outpost, you only need to provide the Outpost listener URL that you defined with the `-O` parameter, and the Leader auth token.

The Outpost doesn't have a separate auth token. When configuring Node connection, use the auth token of the Leader.

Configure the connection either from Settings (**Global** > **Distributed Settings** > **Leader Settings**), or by setting the `CRIBL_DIST_LEADER_URL` environment variable:

```
CRIBL_DIST_LEADER_URL=tcp://<leader-token>@<outpost-hostname>:<outpost-port>
```

Copy

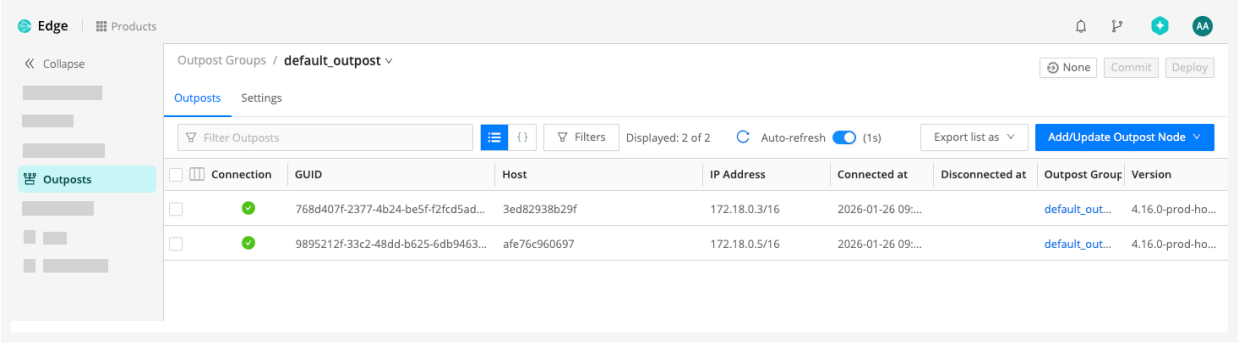
2.8.2. Manage Cribl Outpost

Control, diagnose, and upgrade your Cribl Outposts.

View Outposts

The **Outposts** page displays all Outposts configured in your deployment, including those that are currently disconnected.

1. On the top bar, select **Products**, and then select **Stream**.
2. In the sidebar, select **Outposts**. You will see a list of all Outpost Groups.
3. Select any group to see its Outposts, including disconnected ones.



List of Outposts

4. Select any row in the table to see detailed information about the Outpost.

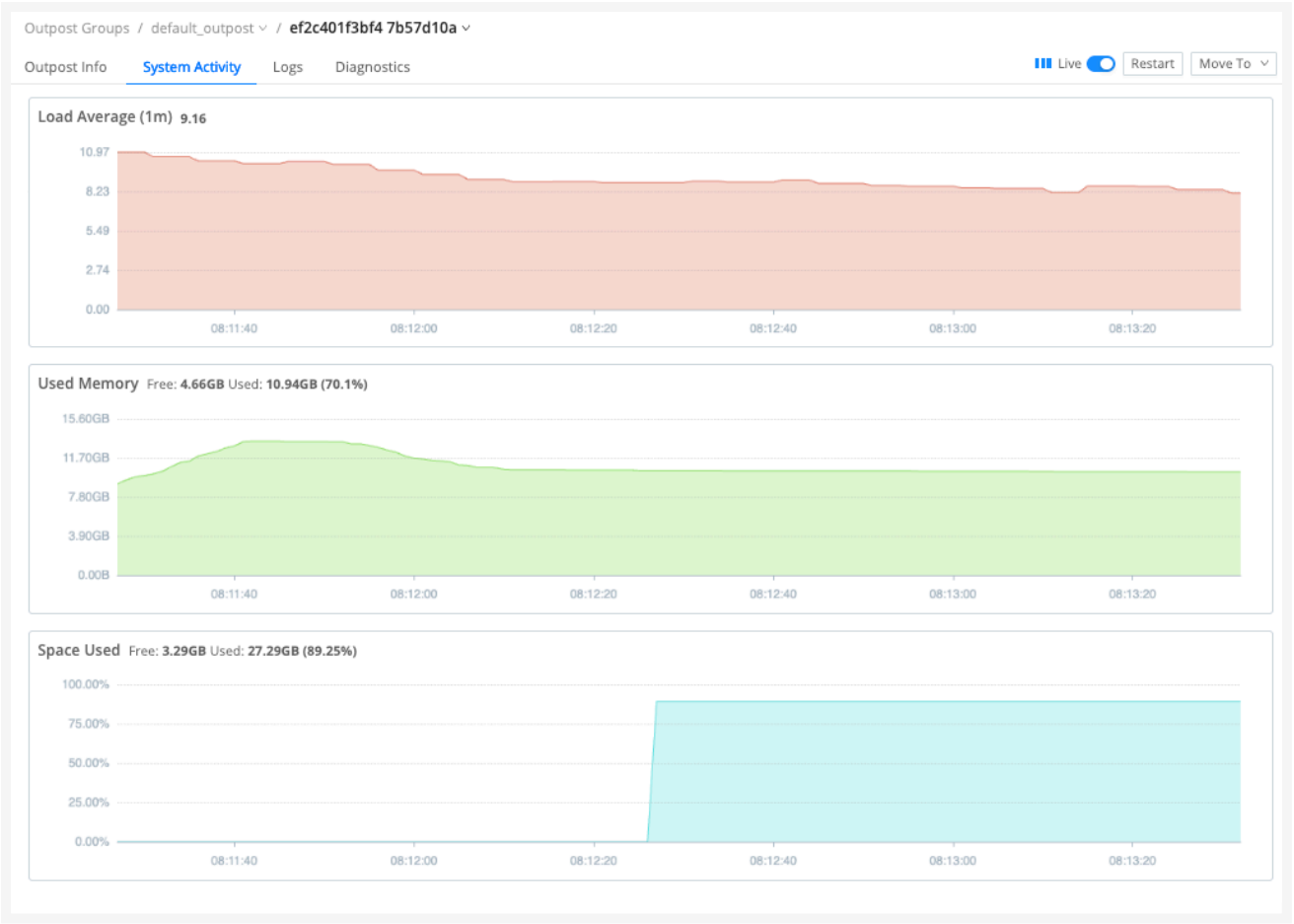
Alternatively, select **List View** above the Outpost Group table to view all Outposts in all groups.

To export the filtered list of Outposts in JSON or CSV format, select **Export list as** above the table.

Check Outpost Activity

To view the data flow through an Outpost Node, select an Outpost Group and an individual Outpost, then select the **System Activity** tab. This tab displays charts for:

- Load Average (1m)
- Amount of memory used
- Amount of disk space used



Outpost System Activity charts

View Outpost Logs

To view Outpost logs, consult the **Logs** tab in the Outpost screen. For more information about Outpost logs, see Internal Logs.

Create Outpost Diagnostic Bundle

In the outpost screen you also create a diagnostic bundle to share with Cribl Support as help in diagnosing any issues. To do so, on the **Diagnostic** tab, select **Create Diagnostic Bundle**. To learn more about diagnostic bundles, see Diagnose Issues.

You can also create diagnostic bundles by using the `diag` command.

Restart an Outpost

Required Permission: **Admin/Owner**

To restart an Outpost:

1. On the top bar, select **Products**, and then select **Stream**.
2. In the sidebar, select **Outposts**, then select one of the Outpost Groups.
3. Select the check box in the row of the Outpost or Outposts you want to restart and select **Restart Selected**.
4. Confirm with **Restart**.

Manage Outpost Groups

Outpost Groups let you manage configurations for Outposts in bulk. The **Settings** tab in Outpost Group view lets you configure preferences for the group. The settings available are a subset of the settings for a Worker Group (in Cribl Stream) or an Edge Fleet (in Cribl Edge).

Outposts created in versions older than 4.16.0 will always stay in the `default_outpost` group. To move those Outposts between groups, first upgrade them to version 4.16.0 or newer.

Outpost Connectivity

You can configure how long the Leader should keep track of disconnected Outposts by configuring the **Time to keep disconnected Outposts** Outpost Group setting. By default the Leader retains information about disconnected Outposts for one day. You can set it to any value between 0 (which means Outposts are removed within a few minutes) and 90 days.

Outpost Upgrades

Defining a target version for an Outpost Group lets you upgrade all Outposts from that group in bulk. See Upgrade Outpost for more information.

Move Outpost to Another Group

You can move an Outpost to another group by selecting its check box and selecting **Move To** below the Outpost table.

Moving an Outpost to another group can temporarily disconnect it from the Leader. If the new group has a target software version selected, the Outpost will upgrade to that version on move.

Moving an Outpost via the **Move To** button will be impossible for Outposts that have any `DIST_` environment variables set.

2.9. Set Up Version Control

As part of the Cribl Stream installation process for on-prem deployments, verify that Git is installed on the hosts running your deployment. Git is a required component for various reasons:

- It is a dependency for many Cribl Stream features.
- The Monitoring > Licensing dashboard will display configuration change markers only if you have `git` installed.
- All configuration changes must be committed before they are deployed. The Leader notifies Workers that a new configuration is available, and Workers pull the new configuration from the Leader Node.

See Commit and Deploy Changes for more information about how to deploy changes with the UI. You can also use the Cribl API to automate commit and deploy commands. Read Commit and Deploy Changes in the API documentation for more information.

Requirements

Cribl Stream requires Git **1.8.3.1.** or later.

For distributed deployments, `git` **must** be installed and available locally on the host running the Leader Node.

Verify Git

To verify that `git` is available and running the required version, run:

```
git --version
```

Copy

Install Git

To install Git, see the Git Installation steps in the official Git documentation.

2.10. Docker Deployment

You can use the following `docker-compose.yml` to stand up a Cribl Stream distributed deployment of a Leader and one or more Workers:

docker-compose.yml

```
services:
  leader:
    image: cribl/cribl:latest
    environment:
      - CRIBL_DIST_MODE=leader
      - CRIBL_DIST_LEADER_URL=tcp://${CRIBL_DIST_TOKEN:-criblmaster}@0.0.0.0:4200
      - CRIBL_VOLUME_DIR=/opt/cribl/config-volume
    ports:
      - "19000:9000"
    volumes:
      - "~/cribl-config:/opt/cribl/config-volume"
  workers:
    image: cribl/cribl:latest
    depends_on:
      - leader
    environment:
      - CRIBL_DIST_MODE=worker
      - CRIBL_DIST_LEADER_URL=tcp://${CRIBL_DIST_TOKEN:-criblmaster}@leader:4200
    ports:
      - 9000
```

Copy

When you run the `docker compose` command, the `~/cribl-config` local directory is created automatically for use as a persistent configuration store for Cribl Stream.

If you prefer to use ephemeral storage, simply delete line 8 (the `CRIBL_VOLUME_DIR` definition) and lines 11-12 (the `volumes` configuration) before running the `docker-compose` command.

Cribl recommends deploying the Leader on stable, highly available infrastructure, because of its role in coordinating all Worker instances.

Running as a Non-Root User

You can run the Cribl image as a non-root user within a container using Docker's `--user` option or the user setting in Compose. You can specify any desired `UID` (User ID).

To ensure correct functionality, you must set the `CRIBL_VOLUME_DIR` environment variable to a location the non-root user can write to. This is because the software requires write permissions to directories like `/opt/cribl/state`, `/opt/cribl/log`, and `/opt/cribl/local`. Failure to set `CRIBL_VOLUME_DIR` will result in a warning and potential functional issues.

If you mount a persistent volume, setting `CRIBL_VOLUME_DIR` to the mount point is strongly recommended for optimal performance and reliability. While `/tmp/cribl_data` might be used, a dedicated persistent volume ensures data integrity and prevents potential data loss or corruption.

We recommend specifying `CRIBL_VOLUME_DIR` whenever running as a non-root user to avoid unexpected behavior.

See example below:

docker-compose.yml

```
services:
  master:
    image: cribl/cribl:latest
    user: 999:999
    environment:
      - CRIBL_DIST_MODE=leader
      - CRIBL_DIST_LEADER_URL=tcp://${CRIBL_DIST_TOKEN:-criblmaster}@0.0.0.0:4200
      - CRIBL_VOLUME_DIR=/opt/cribl/config-volume
    ports:
      - "19000:9000"
    volumes:
      - "~/cribl-config:/opt/cribl/config-volume"
  workers:
    image: cribl/cribl:latest
    user: 999:999
    depends_on:
      - leader
    environment:
      - CRIBL_DIST_MODE=worker
      - CRIBL_DIST_LEADER_URL=tcp://${CRIBL_DIST_TOKEN:-criblmaster}@leader:4200
      - CRIBL_VOLUME_DIR=/tmp/
    ports:
      - 9000
```

Copy

Deploying Hybrid Workers

With a Leader on Cribl.Cloud, encryption is enabled by default. Set the hybrid Worker's `CRIBL_DIST_LEADER_URL` environment variable to begin with the `tls://` protocol. For example:

```
CRIBL_DIST_LEADER_URL=tls://<token>@main-<Organization-name>.cribl.cloud:4200
```

Copy

When you include an auth token in `CRIBL_DIST_LEADER_URL` to deploy managed Cribl Stream or Edge Nodes, it can include only alphanumeric characters or the underscore (`_`). Do not include `<`, `>`, `"`, `` ` ``, `\r`, `\n`, `\t`, `{`, `}`, `|`, `\`, `^`, or `'`.

Selecting the Number of Workers

To deploy a Leader Node, plus (e.g.) two Workers already configured and wired up to the Leader, use this command:

```
docker compose up -d --scale workers=2
```

Copy

To deploy a different number of Workers, just change the `workers=2` value.

If you are deploying the Leader and Workers on the same machine or VM, and the Leader is crashing with two workers, make sure you are allocating enough memory to Docker.

Default Image, OS, and Port Assignments

By default, the above command pulls the freshest stable image (tagged `cribl/cribl:latest`) from Cribl's Docker Hub. Our Docker images are currently built on Ubuntu 20.04.6.

Launching Docker containers with the above `docker-compose.yml` file will default to the following URLs and ports:

- Leader URL: `http://localhost:19000`
- Worker URLs: `http://localhost:<automatically-assigned-host-ports>`

With virtual machines, the VMs' IP addresses would replace `localhost`. The automatic assignment of available host-OS ports to the Workers prevents port collisions. **Within** the Docker container, these ports will forward over TCP to port 9000. To see the ports assigned on the OS, enter:

```
docker ps
```

You should see results like these:

```
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS
a3de9ea8f46f   cribl/cribl:latest    "/sbin/entrypoint.sh..."   12 seconds ago   Up 10
40aa687baefc   cribl/cribl:latest    "/sbin/entrypoint.sh..."   12 seconds ago   Up 10
df362a65f7d1   cribl/cribl:latest    "/sbin/entrypoint.sh..."   13 seconds ago   Up 11
```

Copy

The host-OS ports are shown on the left, forwarding to the container-internal ports on the right. You can use the `docker_workers_N` ports if you want to log directly into Workers. In the above example:

- Worker1 URL: `http://localhost:63411`

- Worker2 URL: `http://localhost:63410`

Adding Port, Forwarding, and Protocol Assignments

To specify additional ports in a distributed deployment, add them to the `docker-compose.yml` file's `workers > ports` section:

docker-compose.yml (modified)

```
version: '3.8'
services:
  [...]
  workers:
    image: cribl/cribl:latest
    depends_on:
      - leader
    environment:
      - CRIBL_DIST_MODE=worker
      - CRIBL_DIST_LEADER_URL=tcp://${CRIBL_DIST_TOKEN:-criblmaster}@leader:4200
    ports:
      - 9000
      - <Additional ports go here>
```

Copy

You can specify port forwarding, and TCP versus UDP protocol, in the same keys:

Specifying protocols when adding ports

```
    ports:
      - 9000
      - "10060/tcp"
      - "10070/tcp"
```

Copy

Updating the Docker Image

Cribl recommends that you always use our latest stable container image wherever possible. This will provide bug fixes and security patches for any vulnerabilities that Cribl has discovered when scanning the base image OS, dependencies, and our own software.

Note that the sample `docker-compose.yml` provided above automatically specifies the latest stable image at:

```
image: cribl/cribl:latest
```

Copy

You can explicitly pull the latest stable image with this CLI command:

```
docker pull cribl/cribl:latest
```

Copy

Updating the Packaged OS

Cribl strongly recommends that you monitor and patch vulnerabilities in the packaged OS. The base OS might have been updated with fixes for new vulnerabilities discovered after Cribl published its container images. This is especially important if you choose to keep an earlier Cribl image in production.

3. Administration

Consult the following topics to learn how to manage and configure your Cribl Stream deployment.

- FinOps Center
- Licensing
- Configuration Management
- Workspaces
- Projects
- Notifications
- Customize the Interface
- Scripts
- Upgrade Cribl Stream
- Uninstall Cribl Stream

3.1. FinOps Center

The Cribl FinOps Center is designed to simplify the process of tracking your product usage and credit consumption. You can also download finalized invoices and export them to external spend analysis products.

With the FinOps Center, you can:

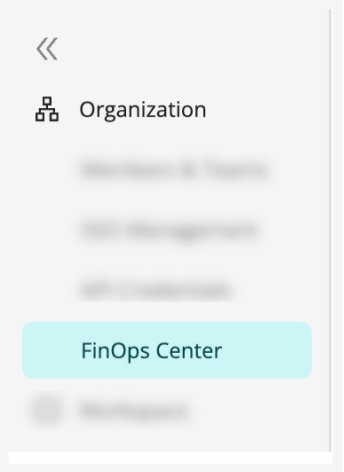
- Explore credit usage across all Cribl products, including credit refunds and rollovers.
- Narrow down usage to a specific time frame using the custom time picker.
- View contract renewal date against usage.
- Get answers to questions you have about your bill.
- Easily comprehend both your total usage and cost for all the Cribl products you own, including Connected Environments and hybrid Worker Groups.

For more information about Cribl.Cloud billing and pricing, see our Cribl Pricing page.

Access Billing Information

The **FinOps Center** page is the single source of truth for your Cribl.Cloud Organization billing and usage data.

1. Log in to Cribl.Cloud.
2. In the sidebar, select **Organization**, then **FinOps Center**.

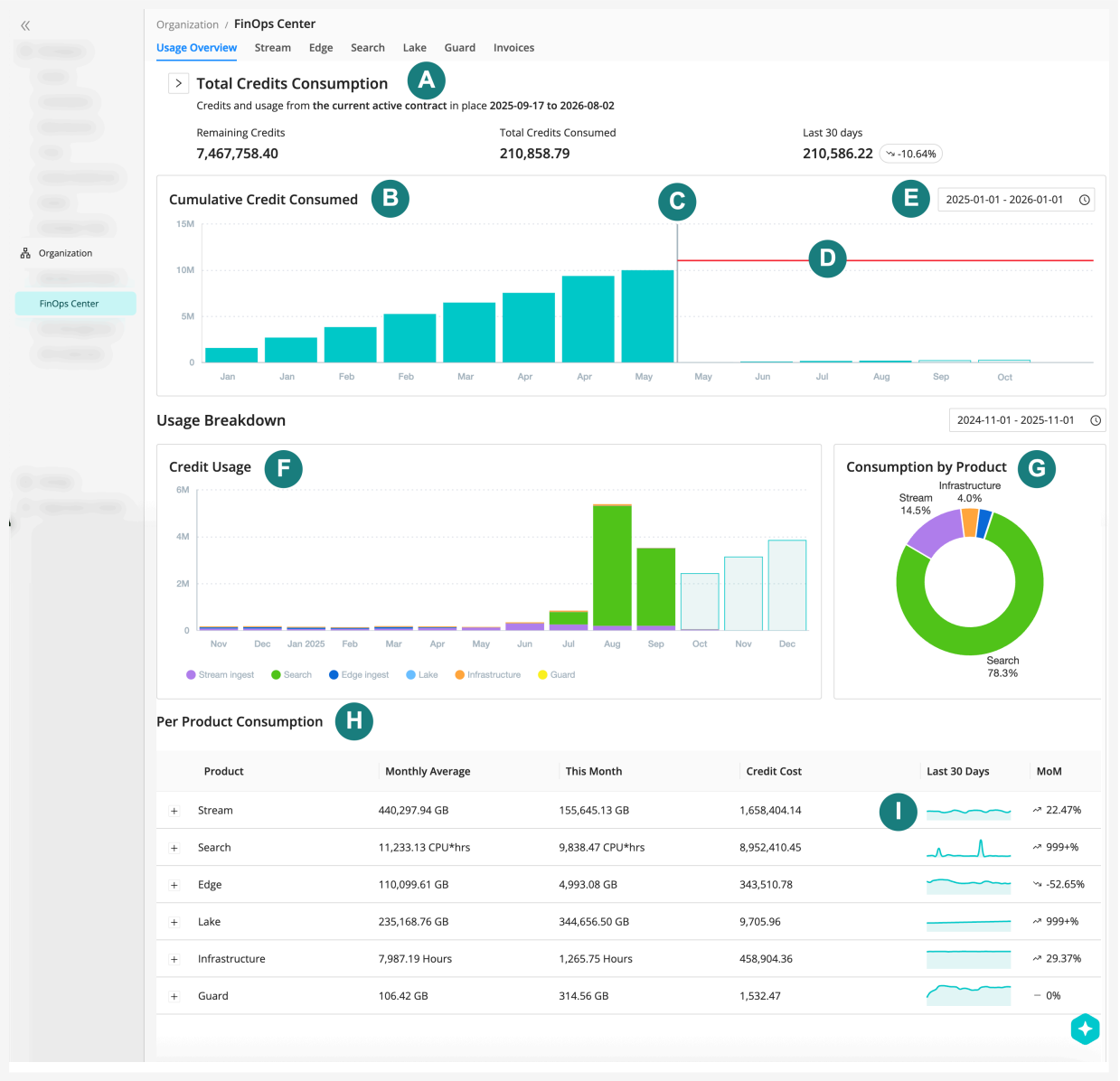


Organization sidebar menu

If you don't see **Organization** in the sidebar, select the Organization icon from the top navigation bar and choose your Organization.

Explore the FinOps Center

Select the **Usage Overview** tab for a holistic view.



Explore the Cribl FinOps Center

Callout	Area	Description
A	**Total credits consumption**	View an overview of your credits for the current active contract. Expand the arrow to reveal a credit ledger, including any credits that rolled over from your last contract period.
B	**Cumulative credit consumed**	View a column chart containing your cumulative credit consumption across all products.

Callout	Area	Description
C	**Contract renewal indicator**	The vertical grey line indicates the point at which your Cribl contract was renewed. Because credits are consumed on a per-contract basis, you might see a drop in consumed credits directly after the renewal indicator, even if it is mid-month.
D	**Total acquired credits**	The horizontal red line indicates total acquired credits in relation to consumed credits, including refunds and rollovers.
E	**Date range picker**	Use the date range selector to view usage and cumulative credit consumption in a specific date and time period.
F	**Credit usage**	View a column chart containing credit usage by product. Hover over each column to view the number of credits consumed for the time period indicated on the x-axis.
G	**Consumption by product**	View a donut chart of your credit consumption by product for the selected date range, expressed as a percentage of your total credit allotment for the active contract period.
H	**Per product consumption**	Find detailed, per-product credit consumption information in this section. Expand a product name for the single product usage view.
I	**Sparklines**	View usage changes at-a-glance with the per-product sparkline chart and month-over-month trend.

View Product-Specific Usage and Credit Consumption

To further understand how your Organization is using Cribl and spending credits, you can select a product or offering from the top menu bar.

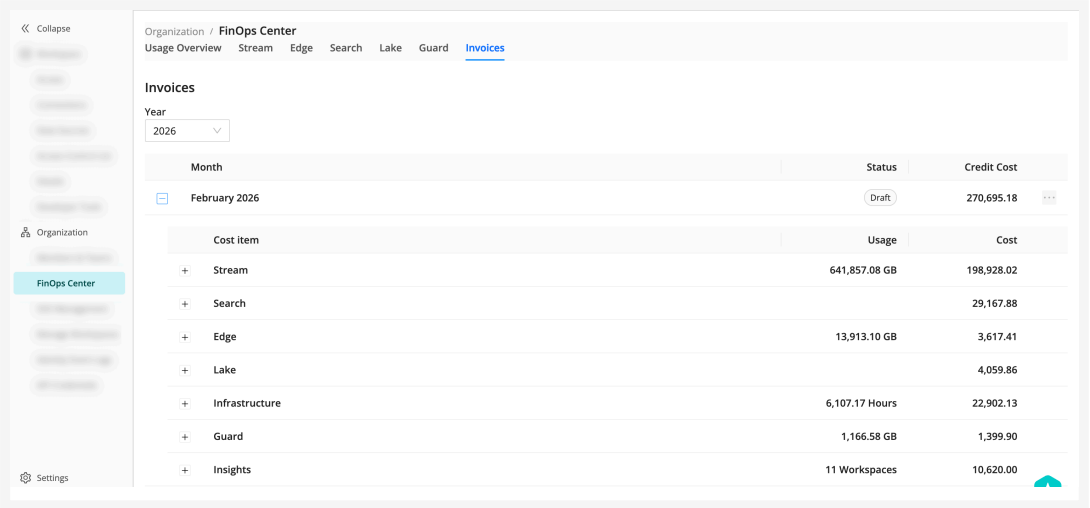
Stream | Edge | Search | Lake | Guard

On the Stream tab, you'll find these helpful billing metrics:

- The total credit consumption by Cribl Stream, including the current month and total monthly average.
- Credit consumption and 3-month consumption forecasting based on the current credit consumption in this Organization.
- Credit consumption by environment type, including hybrid, cloud, and Connected Environments.
- Cribl Stream usage in GB of data ingested, visualized by environment type.
- Cribl Stream usage per Workspace in GB of data ingested.
- The 10 Worker Groups with the most data usage in GB.

Understand Your Monthly Invoice

Your invoice offers details about the usage and costs you have incurred in the current and previous months.



Monthly invoice information

To view your monthly Cribl invoice, make sure you're logged in to your Cribl.Cloud account and you've selected your Organization.

1. From the sidebar, select **FinOps Center**.
2. The **Invoices** tab is where you'll find your monthly invoices.
    - Each invoice is ordered by month, most recent to oldest.
    - The invoice status will be **Draft** for the current month until your billing period ends, as defined in your contract.
    - You can download **Final** invoices as JSON or CSV using the Actions ⋯ menu.
3. To view the details of each monthly invoice, expand the cost item.

Here, you'll see the breakdown of credits consumed by product and infrastructure:

Product	Cost item
Cribl Stream	• Cloud Worker Group Net Data Ingest • Hybrid Workers GBs Received • Connected Stream GB In (on-prem Leaders that are passing usage data to Cribl.Cloud)
Cribl Edge	• Edge Node GBs Ingress • Connected Edge GBs In

Product	Cost item
Cribl Search	• Search Subscription (flat-rate pricing) • Search Total Compute (CPU x Hours)
Cribl Lake	Cribl Lake Total Storage (dataset size volume per GB on an assumed 30 day data retention schedule)
Infrastructure	Cribl.Cloud Worker Group infrastructure usage in hours, per MB/s.
Cribl Guard	Cribl Guard GB scanned. See Cribl Guard documentation for more information.
Cribl Insights	Insights rolling retention by Workspace. See Insights Pricing for more details.
Other	Lakehouse and Federated engine usage and cost: • Lakehouse engine storage in GB. • Federated engine usage by size, in hours. See the Engines documentation for more details.

Download Invoices

You can download your monthly Cribl.Cloud invoices as CSV or JSON files. Then, use them in external tools to help with budget planning and analytics or store them for historical purposes.

You can only download invoices with a **Final** status. **Draft** invoices can't be downloaded.

To download an invoice:

1. In the sidebar, select **Organization**, then **FinOps Center**.
2. Select the **Invoices** tab.
3. Hover over Actions ⋯ for the invoice you want to download.
4. Select **Download CSV** or **Download JSON**.

Frequently Asked Questions

How Is Usage Calculated?

For Cribl Stream and Cribl Edge, your total spend equates to data ingest plus the infrastructure resources needed for processing your data. We charge on ingress (data in), but you never pay for egress (data out), so you

can send your processed data to as many receivers as you'd like.

For Cribl Lake, usage equates to the amount of compressed data stored. And for Cribl Search, usage equates to the amount of compute resources used to deliver search results. We charge in compute hours, and most searches take less than a second.

How Is Cost Calculated?

In Cribl.Cloud, one credit equals one US dollar. Each of our four products (Cribl Stream, Cribl Edge, Cribl Lake, Cribl Search) consume credits at a different rate. For a detailed explanation, see our Cribl Pricing Guide or contact Cribl Sales.

Here's a quick summary of Cribl.Cloud credit consumption:

Product	General Calculation
Stream	Ingest pricing (infrastructure pricing based on region): • 0.32 Credits/GB per Cloud Worker • 0.26 Credits/GB per Hybrid Worker
Edge	Ingest pricing: • 0.21 Credits/GB per Edge Node
Search	Compute pricing: • 1 Credit per CPU hour If a Search is entirely satisfied by one or more Lakehouses, you are not billed Search CPU hours.
Lake	Storage pricing based on 30-day retention: • .05 Credits/GB if Cloud Worker managed by Cribl • .02 Credits/GB if self-managed • Lakehouse is billed on how long the Lakehouse has been running. The price per month depends on the Lakehouse capacity you purchased.
Lakehouse	• Lakehouse billing begins as soon as Cribl provisions the Lakehouse, shortly after you select **Save**. • You're billed based on how long the Lakehouse has run. • Price per month depends on the Lakehouse capacity you purchased. For more details about Lakehouse provisioning, see our Lakehouse docs.

Product	General Calculation
Guard	• Cribl Guard billing begins as soon as the Cribl Guard Function starts ingesting data in the Pipeline and scanning for sensitive information. • You're billed credits per GB scanned. • You can start accruing charges immediately after enabling **Protect** on a Destination from the Cribl Guard homepage. See Configure Cribl Guard for more details.
Insights	Pricing is based on a rolling data retention period. For more details, visit Insights Pricing.

Where Can I View Connected Environment Consumption On My Bill?

You can view Connected Environment consumption and usage in a few places.

First, make sure you're logged in to your Cribl.Cloud account.

View credit cost for Connected Environments:

1. Navigate to **Organization**, then **FinOps Center**.
2. In the **Month** column, expand a finalized invoice, then select **Stream** or **Edge**.
3. If you have Connected Environments cost, you'll see it listed here as **Connected Stream GBs In** or **Connected Edge GBs In**.

View usage for Connected Environments:

1. Navigate to **Organization**, then **FinOps Center**.
2. Select **Stream** or **Edge** in the top menu.
3. Scroll to **Stream Usage (GB)** or **Edge Usage Per Workspace (GB)**.

3.2. Licensing

Every Cribl Stream download package comes with a Free license that allows for processing of up to 1 TB/day. This license requires sending anonymized Telemetry Data to Cribl.

Enterprise, Standard, and Sales Trial licenses are entitled to a defined, per-license daily ingestion quota. These licenses do not require sending telemetry metadata.

> This page applies only to on-prem deployments. For basic information about Cribl.Cloud Enterprise and Standard plans, see Cloud Pricing.

Manage On-Prem Licenses

Check Licenses

To check your license type, restrictions, and/or expiration date:

1. In the sidebar, select **Settings**.
2. On the **Global** tab, select **Licensing** to see these details.

Add or Renew Licenses

On the Leader (or in a Single-instance deployment), after submitting your newly untarred Cribl Stream software registration page, you can add and manage licenses at **Settings** > **Global** > **Licensing**. Select **Add License** to paste in a license key provided to you by Cribl.

This applies to Cribl Stream Standard and Enterprise licenses, which must be renewed annually. Free licenses are already onboard the download package, need not be added or managed here, and do not expire.

License Expiration

When an Enterprise or Standard license expires, Cribl Stream attempts to fall back to a Sales Trial or Free license; if there is no valid fallback license, Cribl Stream will backpressure and block all incoming data.

- You can remove an expired paid license to fall back to the Sales Trial or Free type.
- An expired Sales Trial license will fall back to Free, but you must remove the Sales Trial license and restart the Leader.

If you already added a new paid license, but still see Sales Trial or Free after a previous license expires, delete the expired license or restart the Leader to force the updated license state to take effect.

To resolve expiration and ensure the correct license is active:

1. In the sidebar, select **Settings**.
2. On the Global tab, select **Licensing**.
3. If an expired license is present, select **Delete license**.
4. Restart the Leader to apply the current license immediately.

Upon expiration of a paid license, if there is no valid fallback license, Cribl Stream will backpressure and block all incoming data.

Licenses in Airgapped Deployments

If your environment does not allow internet connectivity, you'll need to bypass the registration web form when you deploy or upgrade Cribl Stream.

To do so:

1. On the Leader (or single instance) host, navigate to `$CRIBL_HOME/local/cribl`.
2. Create a file named `licenses.yml`.
3. Paste in your license key in this format, then save the result:

```
licenses:
- <your-license-key-here>
```

Copy

Notification of Renewal and Expiration

The UI will prompt you to renew your paid license by displaying banner and drawer notifications, starting 30 days before license expiration. When you see these prompts, verify your license expiration date and contact Cribl Sales at sales@cribl.io to renew.

Beyond these default notifications, you can configure custom License-Expiration Notifications. You can send these to PagerDuty, or to other external services via webhook.

License Quotas

Track Data Ingestion

To track your data ingestion volume over the last 30 days:

1. In the sidebar, select **Monitoring**.
2. Select **System** and then **Licensing**. This shows you the dashboard with your license consumption.

For higher-fidelity metrics, you can enable the Cribl Internal Source CriblMetrics option to forward metrics to your metrics Destination of choice, and run a report on `cribl.total.in_bytes`.

Exemptions from License Quotas

The following data is exempt from license quotas and Cribl Credit usage:

- Data generated by Cribl Internal Sources does not count against your ingestion quota.
- Edge Node events that you send to the DevNull Destination or drop in any Pipeline do not count against your ingestion quota. This license reduction is calculated based on average event size and may not represent the exact number of bytes you drop.

If you are connecting Workers/Edge Nodes in a Cribl Stream Cloud hybrid deployment, you are granted additional exemptions to prevent double billing (in Cribl Edge, these exemptions also apply to customer-managed deployments):

- **Cribl Stream (internal data transfers):** Data transferred between Cribl Stream Workers using Cribl HTTP and Cribl TCP Sources does not count against your ingestion quota.
- **Cribl Edge to Cribl Stream:** Data that Cribl Edge sends to Cribl Stream from the Cribl HTTP and Cribl TCP Destinations counts against quota only in Cribl Edge.
- **Cribl Search to Cribl Stream:** When you target a Cribl-managed Stream Worker Group within the same Workspace over Cribl HTTP, you incur no additional ingest charges. To forward data Cribl Search to Cribl Stream, you use the `send` operator. You only consume credits for the CPU time that the `send` operator uses in Cribl Search. See Cribl Search: Send for more information about this operator.
- **Datagens:** Data generated by Datagens does not count against your ingestion quota.

License Quotas with Compressed S3 Buckets

If you pull data from compressed S3 buckets, the license quota is applied to the uncompressed size of the file objects.

Enforce License Quotas

If your data throughput exceeds your license quota:

- Free and Standard licenses enforce data ingestion quotas through limits on the number of Worker Groups/Fleets and Worker/Edge Node Processes.
- Enterprise license keys turn off all enforcement.

- When an Enterprise or Standard license expires, Cribl Stream will attempt to fall back to a trial or free license, or - only if that fails - will block incoming data. For details, see Combining Licenses.

License Types

Cribl offers several Cribl Stream license types including Enterprise, Standard, and Free. For a current list of available license types and a detailed comparison of what's included in each, see Cribl Pricing.

Combine Licenses

Multiple licenses can coexist on one instance. When two or more Standard or Enterprise licenses are present, the effective ingestion quota for the deployment will be equal to the sum of their individual ingestion quotas.

When multiple *license types* coexist, the effective license is determined by the following priority order:

1. Enterprise or Standard licenses that have not expired.
2. Sales Trial licenses.
3. Free licenses.

Licensing in Distributed Deployments

With licenses that limit the number of Worker Processes/Edge Nodes, Cribl Stream will attempt to balance or rebalance Worker Processes (threads) as evenly as possible across all licensed Worker Nodes.

You configure licensing only on the Leader Node. (See Managing Licenses - Adding and Renewing.) The Leader will push license information down to Worker Groups as part of the heartbeat (Stream, Edge). There is no need to configure or store licenses directly on Worker Nodes.

Temporary License Restriction on Worker Node Startup

When you first start a Worker Node, it will use the default license. As a result, you might see a log entry warning about a limit on Worker Processes:

```
{
  "time": "2023-02-10T15:11:48.609Z",
  "cid": "api",
  "channel": "cribl",
  "level": "warn",
  "message": "number of worker processes has been limited by the license",
  "requested": 14,
  "licenseLimit": 10,
  "source": "cribl.log"
}
```

Copy

You can ignore this warning message. The license won't limit Worker Processes once the Leader passes the production license to the Worker.

Convert an On-Prem License to a Cribl.Cloud Billing Plan

You can convert on-prem Cribl Stream licenses to a Cribl.Cloud billing plan. This lets you keep your on-prem deployments of Cribl Stream, but manage all of the environment licenses in one interface.

For more details and setup steps, visit Connecting On-Prem Leaders to Cribl.Cloud Billing.

Telemetry Data

A **Free** license requires sharing of telemetry **metadata** with Cribl. Cribl uses this metadata to help us understand how to improve the product and prioritize new features.

Telemetry payloads are sent from all Cribl Stream Nodes (Leader and Workers), to an endpoint located at `https://cdn.cribl.io/telemetry/`.

Test the Telemetry Endpoint Connectivity

To manually test connectivity to the telemetry endpoint, especially if you need to configure a proxy, you can use the following command:

```
curl https://cdn.cribl.io/telemetry/
```

Copy

Expected response:

```
cribl /// living the stream!
```

If you get a 302 response code, check whether you've omitted the trailing `/` from the URL.

Disable Telemetry

With an Enterprise or Standard license, can disable telemetry sharing from on-prem Cribl Stream. With a Free license, disabling telemetry will cause Cribl Stream to block inbound traffic within 24 hours.

If you would like an exception to disable telemetry in order to deploy in your environment, please contact Cribl Sales at sales@cribl.io, and we will work with you to issue licenses on a case-by-case basis.

Once you have received a license that removes the telemetry requirement, you can disable telemetry in Cribl Stream:

1. Navigate to **Settings** > **Global** > **System** > **General Settings** > **Upgrade & Share Settings**.
2. Toggle **Share telemetry with Cribl** off.

Metadata Shared Through Telemetry

Your Cribl Stream instance shares metadata with Cribl per interval (roughly, every minute).

Details sent for Nodes in any mode:

- Version
- OS Distribution/Version
- Kernel Version
- Instance GUID
- License ID
- Earliest, Latest Time
- Number of Events In and Out, overall and by Source type and Destination type
- Number of Bytes In and Out, overall and by Source type and Destination type
- Number of Open, Closed, Active Connections
- Number of Routes
- Number of Pipelines
- Runtime Environment Indicators (AWS and Kubernetes)

Additional details sent for Leader Nodes (when Cribl isn't sending managed Node data):

- Runtime Environment
- OS Distribution / Version

- Version
- Kernel Version

How We Use Telemetry Data

With telemetry enabled, Cribl software sends usage data directly to Cribl. We securely store and encrypt this data on Cribl-managed servers, with restricted access.

Cribl fully anonymizes and aggregates this usage data in our systems of analysis. There, we use the aggregated metrics to improve Cribl products and services by analyzing circumstances around disruptions, opportunities for ingest efficiencies, and ways to optimize performance. Access is restricted to only those Cribl employees and contractors who require anonymized data to perform their jobs.

Cribl collects License IDs only to ensure that all data is being sent by a Cribl product. These IDs cannot be used to personally identify any user of Cribl software or Cribl.Cloud services.

For further details, see the Cribl Privacy Policy.

Troubleshooting Resources

3.3. Configuration Management

Configuration management is the practice of maintaining your deployment's settings in a known, consistent, and versioned state. For Cribl Stream, this means ensuring that you have a clear strategy for developing, maintaining, versioning, and deploying your configuration content at scale.

There are three basic categories of configuration content that you typically need to manage in your Cribl deployment:

- **Data Flow Integrations:** The configurations for Cribl Stream integrations (Sources, Collectors, and Destinations) that connect Cribl to your upstream data sources and downstream data destinations.
- **Processing Logic:** The configurations for Routes, Pipelines, and Functions contain the rules and logic that filter and transform your data.
- **Supporting Knowledge Objects:** These are the reusable assets that apply context and structure to your raw data, including Event Breakers, lookup files, HMAC Functions, and more.

See Reference Architecture: Configuration Management for a general introduction to configuration management goals and an overview of configuration management strategies and approaches.

3.3.1. Commit And Deploy Changes

In Cribl Stream, you use the commit and deploy workflow to push configuration changes from the Leader Node (the control plane) to your Worker Groups (the data plane). See The Cribl Three-Plane Model for an explanation of the Cribl architectural model if needed.

The commit and deploy workflow ensures that changes are versioned, auditable, and synchronized. It also allows you to revert and roll back changes as needed.

Only users with **Admin** or **Editor** permissions have the ability to view and commit configuration changes. See Permissions for more information about roles and abilities.

The Commit and Deploy Lifecycle

The Leader Node acts as the central configuration management hub and is the single authority for all configurations. When you deploy changes from Cribl Stream, it uses a local Git repository to track and version control all configuration changes for that Leader.

The lifecycle of a configuration change follows a specific order of operations:

1. **Version Control:** Every time you commit in the UI, Cribl Stream performs a Git commit to the local repository on the Leader. This captures a point-in-time snapshot of your settings, including the user who made the change and a description of the update.
2. **Worker Notification:** When you deploy, the Leader increments the configuration version and notifies all Workers in the targeted Worker Group that a new version is ready.
3. **Pull Mechanism:** Workers do not have configurations pushed to them. Instead, they pull the latest bundle from the Leader over the management channel. This ensures that even Workers with intermittent connectivity eventually synchronize to the source of truth: the Leader.

See How Worker Nodes and the Leader Node Work Together for a more detailed technical explanation of the architecture.

Global Commit vs. Worker Group Commit

Distributed mode includes Worker Group-specific **Commit** and **Deploy** controls. These controls function differently from the global **Commit** button because they have a different scope.

The global **Version Control** button:

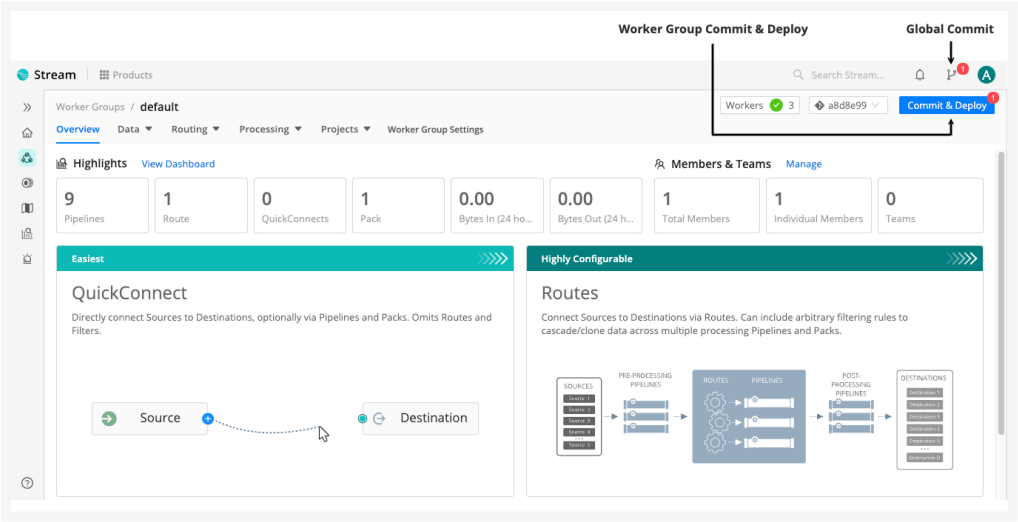
- **Scope:** Commits changes for **all** Worker Groups and the Leader Node simultaneously.

- **Result:** Changes are versioned but not deployed.

The Worker Group **Commit** and **Deploy** buttons:

- **Scope:** Commits and deploys changes for only the currently selected Worker Group.
- **Result:** Allows for a single-click Commit and Deploy, making changes live for that specific group immediately.

These controls will appear slightly differently depending on whether you have changes committed but not yet deployed, and on whether you've enabled the Collapse actions setting.



Commit and Deploy controls in the user interface

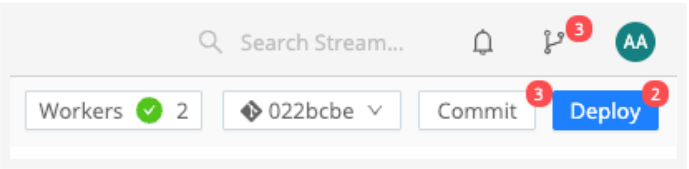
Commit and Deploy Changes

You make configuration changes directly through the Cribl UI or using direct API calls to the Leader Node. The Leader distributes the configuration bundle to Worker Groups upon a manual commit and deploy.

You can also use the Cribl API to automate commit and deploy commands. Read Commit and Deploy Changes in the API documentation for more information.

To commit and deploy configuration changes in the UI:

1. Check the top bar to determine if you have any pending changes to commit. Pending commits have a red dot indicator.

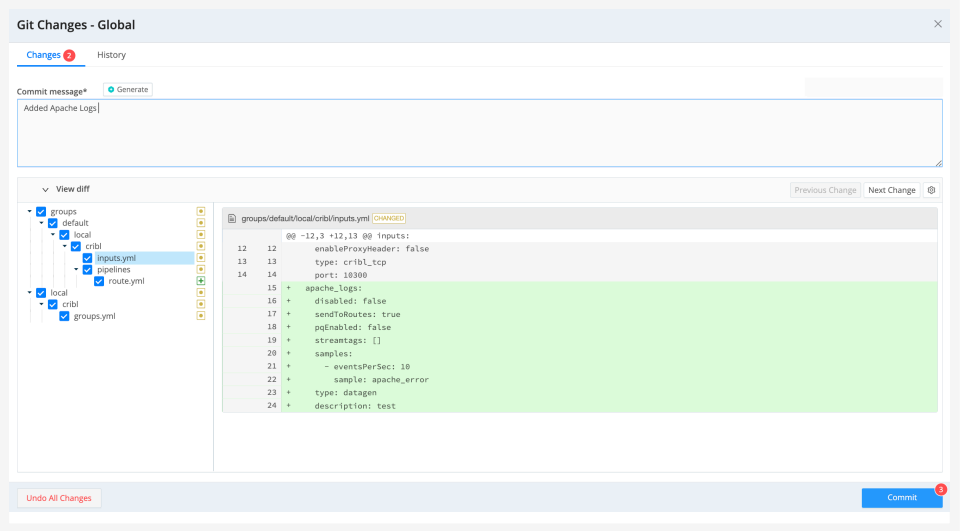


Changes pending commit

2. Select **Commit** to commit pending changes.

Ensure you select the correct button for your intended scope: either the specific Worker Group or the Global (all Groups and Leader) level. See Global Commit vs. Worker Group Commit for an explanation of the differences in scope and which buttons to use.

3. In the **Git Changes** modal, verify the configuration changes. Optionally, clear the check boxes for individual files to exclude them from the commit. If needed, select **Undo All Changes** to reverse the changes instead of committing them.



Review the pending commit

4. Enter a commit message, or select **Generate** to have Cribl Copilot create a message for you, if enabled.
5. Commit your changes, depending on your scope:

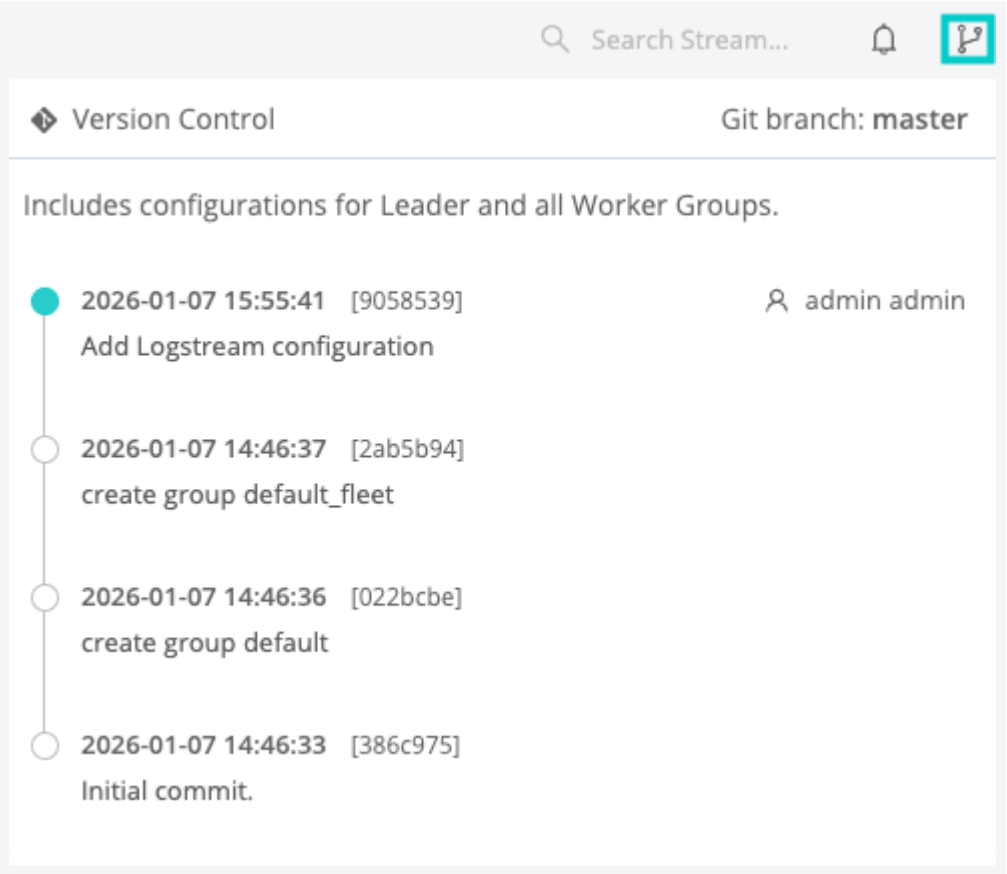
Button	Scope	Result
Commit	Global: Commits changes for **all** Worker Groups and the Leader Node simultaneously.	Changes are versioned but not deployed.
Commit Only	Worker Group: Commits configurations for only the currently selected Worker Group.	Changes are versioned but not deployed.
Commit & Deploy	Worker Group: Commits configurations for only the currently selected Worker Group.	Commits and deploys configuration for only the currently selected Worker Group.

If the commit was successful, you will see a **Commit successful** confirmation message.

View and Revert Committed Changes

To view committed changes in the UI:

- **Worker Groups:** Select **Commit** from the top bar and select the **History** tab in the **Git Changes** modal to cycle through the most recent commits.
- **Global:** Select the global **Version Control** icon from the top bar to see a list the most recent changes. Select a change to view it in the **Git History** modal.



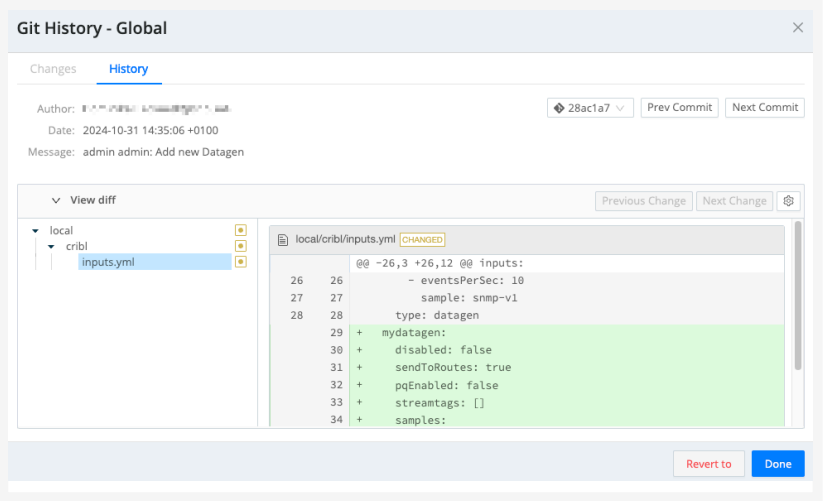
View commits using the Version Control icon in the top bar

Revert Commits in the UI

Reverting a commit in the UI proceeds slightly differently for Single-instance and for Distributed deployments. In a Distributed deployment, you revert commits for each Worker Group separately.

To revert a commit in the UI:

1. If you are on a Distributed deployment, in the sidebar, select **Worker Groups**, then select the Worker Group you want to revert the commit for.
2. Select the commit you want to revert.
3. In the resulting **Git History** modal, verify the diffs in the configuration changes and select **Revert to**.



Diff preview before reverting a commit

4. Confirm the revert. This will cause Cribl Stream to restart.

Revert Commits Using the CLI

To revert a commit using the CLI:

1. Make sure you have committed all your recent configuration changes.
2. Stop Cribl Stream.
3. In your terminal, navigate to your `$CRIBL_HOME` location (typically, `/opt/cribl`).
4. Run `git log` to get a list of most recent commits and find the hash of the commit you want to revert.
5. Run `git revert <commit-hash>` to revert this commit.
   If you want to revert more than one commit, list all their hashes as arguments for `git revert`.

Running `git revert` will create one or more new "Revert" commits that record the change in your git history.

3.3.2. Connect To External Git Repositories

In on-prem deployments, Cribl supports Git remote repositories so you can implement version control for all configuration changes. Connecting to Git is not required.

This feature requires an Enterprise Cribl license. For details, see Pricing.

You can configure a Leader Node with Git remote push capabilities through the Cribl Stream CLI, or through the Cribl Stream user interface. From Settings, select the **Global** tab, then select **System** > **Git Settings**.

To create a repo, see these tutorials:

- Setting Up a Repository (CLI instructions, host-agnostic, from Atlassian).
- Creating a New Repository (specific to the GitHub Web UI).
- Create a Repo (longer GitHub-specific tutorial, also covers committing changes).

Remote Formats Supported

Remote URI schema patterns should match this regex: `(?:git|ssh|ftps?|file|https?|git@[-\w.]+):(\/\/)?(.*?)(\.git\/?)?$`

You can find a list of supported formats in this external article: What are the supported git url formats?

For example:

- GitHub or other providers: `<protocol>://git@example.com/<username>/<reponame>.git`
- Local Git servers: `git://<host.xyz>:<port>/<user>/path/to/repo.git`

Secure Local Configuration and Remote Repos

Make a local backup of your whole `$CRIBL_HOME` directory before you set a remote repo. This safeguards your Cribl Stream configuration against being overwritten by a misconfigured remote.

Create each new remote as an empty repo, with no README file and no commit history. (See the following GitHub and GitLab examples.)

Some files that are used by Cribl Stream (both Leader and Worker Groups) contain sensitive keys; examples are `cribl.secret` and `...auth/ssh/git.key`. These will be pushed to the remote repo

as part of the entire directory structure under version control. Ensure that this repo is secured appropriately.

Connect to a Remote Repo over HTTPS (Recommended)

Cribl recommends connecting to a remote repo over HTTPS. The following examples show a token-based and a password-based HTTPS connection to GitHub.

Example: Connect to GitHub over HTTPS with a Personal Access Token

1. Create a new GitHub repository.
   For best results, create a new empty repo, with no README file and no commit history. (This will prevent `git push` errors.) Note the username and email associated with your login on the repo provider.
2. Create a personal access token with **repo** scope.
3. Copy the token to your clipboard.
4. In Cribl Stream, from Settings, select the **Global** tab, then select **System** > **Git Settings**.
5. In the **Remote** tab, select **Authentication type**: **None**.
6. Fill in the **Remote URL** field with your username and repo name. Use the format below:
   `https://<username>:<token>@github.com/<owner>/<repository>.git`

For additional details, see the GitHub Creating a Personal Access Token tutorial.

For GitHub repos specifically, use only personal access tokens in the **Remote URL** field. GitHub announced its end of support for plaintext passwords as of August 13, 2021.

Example: Connect to GitHub over HTTPS with Username and Password

1. Create a new GitHub repository.
   For best results, create a new **empty** repo, with no README file and no commit history. (This will prevent `git push` errors.) Note the username and email associated with your login on the repo provider.
2. In Cribl Stream, from Settings, select the **Global** tab, then select **System** > **Git Settings**.
3. In the **Remote** tab, select **Authentication type**: **Basic**.
4. Enter your credentials in the **User** and **Password** fields.

Connect to a Remote with SSH

You can set up SSH keys from the CLI, or upload keys via the UI. If you have a passphrase set, this functionality is available only through the CLI - see Encryption: Configuring Keys with the CLI. The example below outlines

the UI steps.

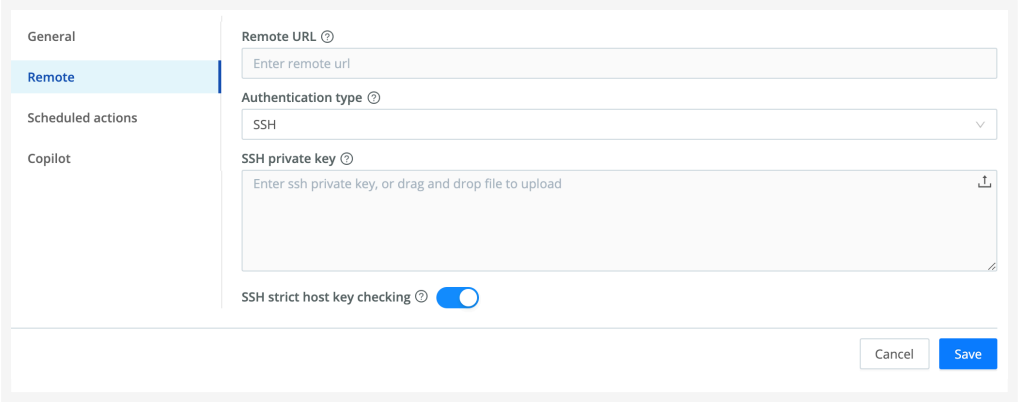
Example: Connect to GitHub with SSH

1. Create a new GitHub repository.
   For best results, create a new **empty** repo, with no readme file and no commit history. (This will prevent `git push` errors.) Note the user name and email associated with your login to the repo provider.
2. Add an SSH public key to your GitHub account.
3. In Cribl Stream, from Settings, select the **Global** tab, then select **System** > **Git Settings**.
4. In the **Remote** tab, fill in the **Remote [repo] URL**. In the generic example below, replace `<username>` with your username on the repo provider:
   - **Remote URL**: `<protocol>://[user@]server[:port][/path/to/repo.git]` (For example: `ssh://user@server/project/path.git`)
   - For GitHub specifically, the URL/protocol format must be: `ssh://user@server/project/path.git`
   - A specific (fictitious) GitHub example: `ssh://myuser@git.example.com/repos/myproject.git`

   > The Remote Host (or SSH Remote) field currently requires a full SSH URL, including the `ssh://` prefix. While the `[user@]server:project.git` (Git-style) format is more common, it is not yet supported. Translate Git-style URLs to SSH-style: for example, `user@server:project/path.git` becomes `ssh://user@server/project/path.git`.
5. Paste in the **SSH private key**.

   > The key will paste in with an appended newline below it. Do not delete this newline before you save the Remote Settings, or else you will trigger an error of the form: `fatal: Could not read from remote repository.`
6. As the user running Cribl Stream, run this command to add the GitHub keys to `known_hosts`:
   `ssh-keyscan -H github.com >> ~/.ssh/known_hosts`

For additional details, see the GitHub Connecting to GitHub with SSH tutorial.



Cribl Stream Git settings

GitLab Notes

For repos hosted on GitLab, we recommend that you:

- Create a new empty (blank) project, with no README file and no commit history. (Clear the **Initialize repository with a README** check box.)

  This will prevent `git push` errors by creating a project with no branch, not even `main`. When you later push the Cribl Stream `master` branch, that will create a corresponding `master` branch on the GitLab remote with no conflicts.

- Create a GitLab project access token for authentication. See the GitLab documentation, which also covers project bot conventions.
- With project bots, the first token's username is set to `project_{project_id}_bot`. The password is the alphanumeric token.
- Create the token with `write_repository` scope.
- Specify a remote URL in HTTPS format. For example: `https://localgitlab.<yourdomain.ext>/<yourusername>/cribl.git`
- If you do see a `main` branch in the Cribl Stream UI, do not select it. Leave the branch at `master`, then use the Stream UI's **Commit** and **Push** commands. This will create the desired `master` branch on GitLab's remote.

The GitLab **Repository Settings** > **Push Rules** section includes these two settings of interest:

- As needed, enable **Check whether author is a GitLab user**.
- Understand the consequences of of enabling **Prevent committing secrets to Git**. This blocks commits of `.pem` and `.key` files. If you have certificates or SSH keys configured, this will break commits from Cribl Stream, throwing only a generic `API Error` in the UI. Check your `git` CLI client for more-specific diagnostics.

Additional Git Settings

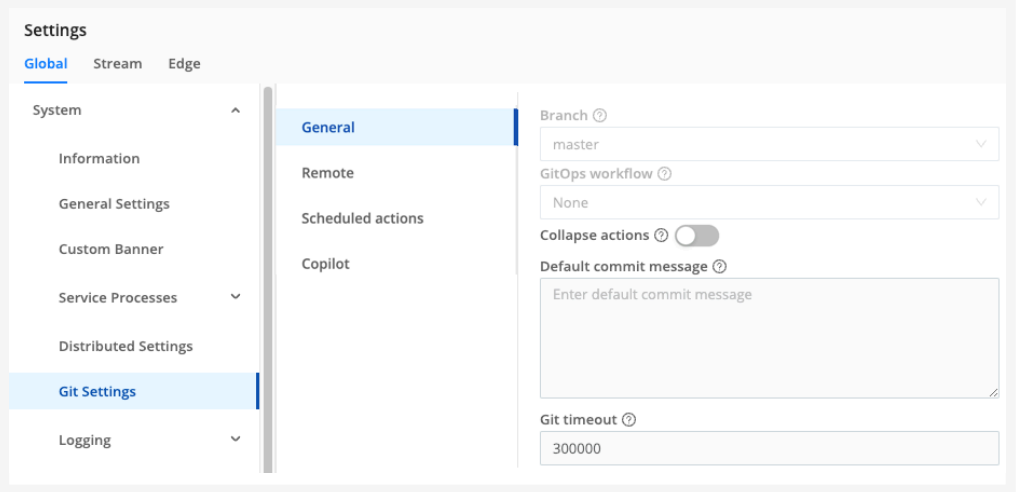
These settings are on-prem only. Cribl.Cloud users will only see the default commit message.

From Settings, select the **Global** tab, then select **System** > **Git Settings**. Select the **General** tab to modify the following additional Git configurations, all of which are optional:

- **Branch**, **GitOps workflow**: These drop-downs are available to configure GitOps.

  See the GitOps documentation for details and precautions.

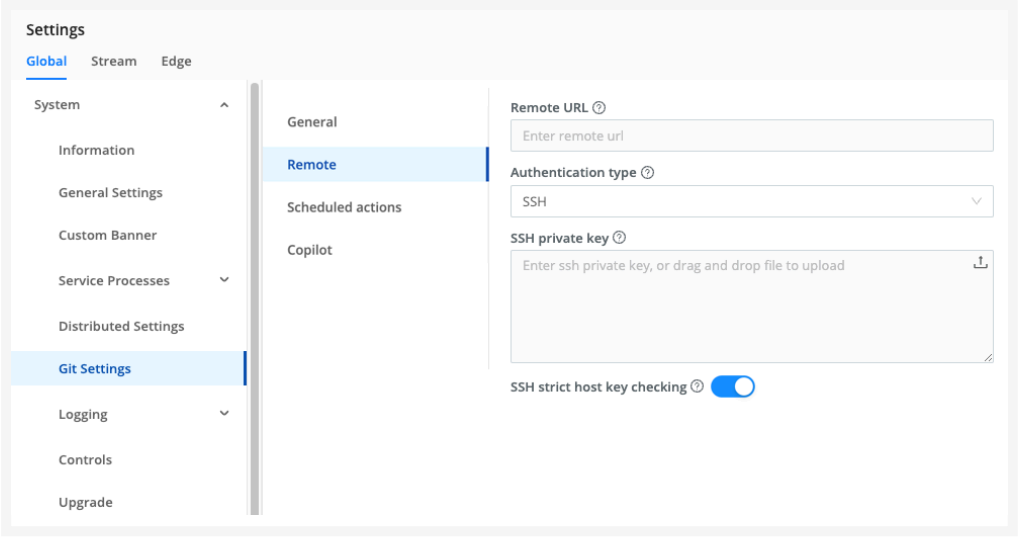
- **Collapse actions**: Combine multiple Git buttons to one, to reduce repetitive clicks. See Collapse Actions below.
- **Default commit message**: Enter a placeholder message to apply to all commits from Cribl Stream. This also reduces clicks.
- **Git timeout**: Maximum time (in milliseconds) to wait for Git processes to complete before killing them. If a Leader instance hangs, try lowering this value from the default `600000` (600 seconds = 10 minutes). However, avoid very low values (like `1000` ms), which might time out even when Git processes are working properly. Enter `0` to wait indefinitely.



Git Authentication Type settings

On the **Git Settings** > **Remote** tab, you can change the **Authentication Type** from its **SSH** default to **Basic** authentication. This displays two additional fields:

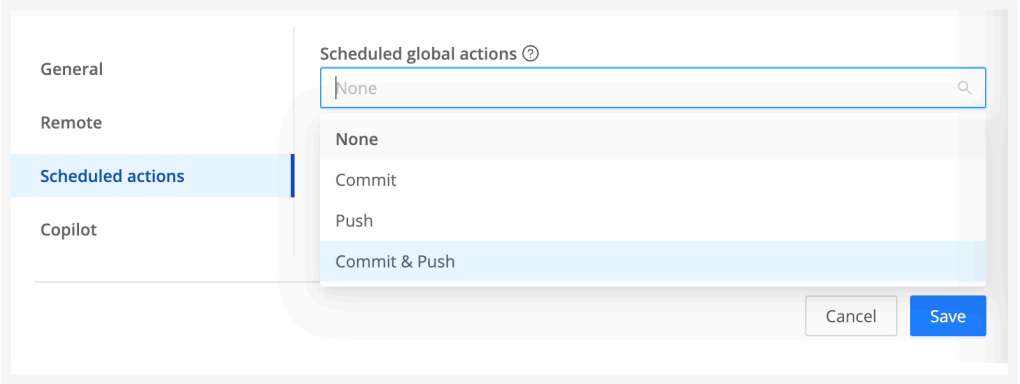
- **User**: Username on the repo.
- **Password**: Authentication password (for example, a GitHub personal access token).



Git Authentication Type settings

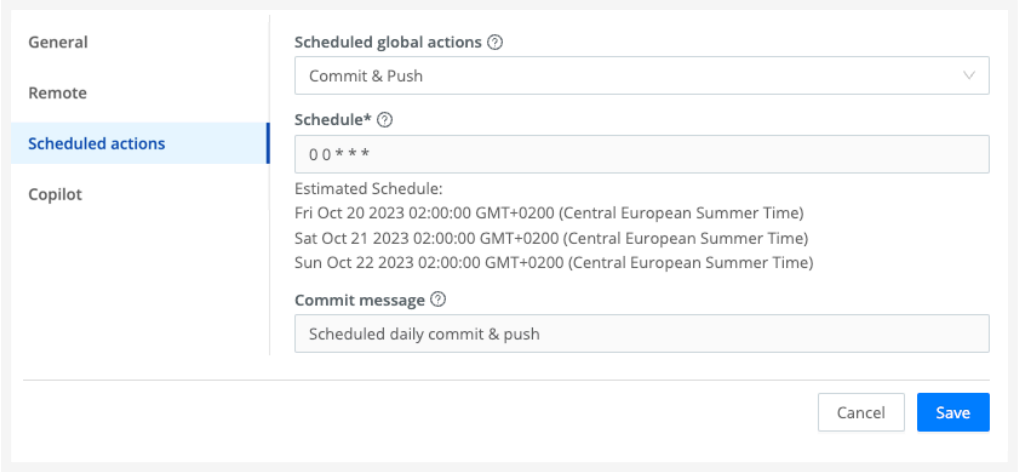
GitHub (specifically) does not support Basic authentication.

On the **Git Settings** > **Scheduled Actions** tab, you can schedule a **Commit**, **Push**, or **Commit & Push** action to occur on a predefined interval.



Git Scheduled Actions selection

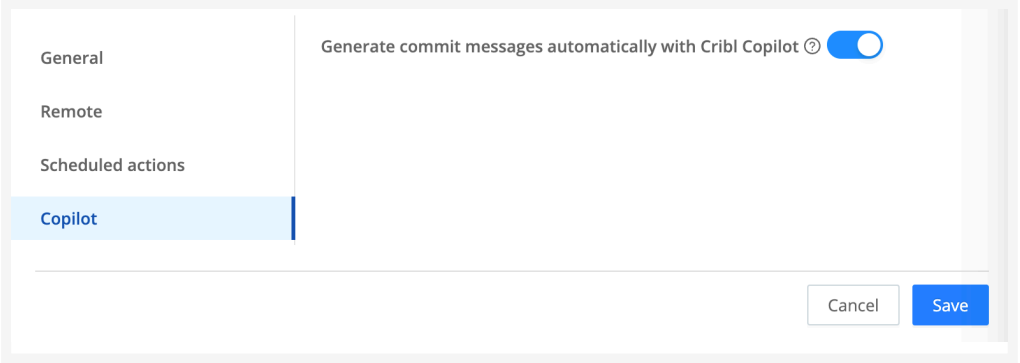
For the selected action type, you can define a cron schedule, and a commit message distinct from the **General** tab's **Default commit message**. Then select **Save**.



Saving a Git Scheduled Action

You can schedule only one type of action. To swap to a different type, select it from the **Scheduled global actions** drop-down, and resave. To turn off scheduled Git commands, select **None** from the drop-down, and resave.

On the **Git Settings** > **Copilot** tab, you can toggle the **Generate commit messages automatically with Cribl Copilot** option on to have Cribl Copilot auto-generate every commit message for you. Cribl Copilot-generated messages override any default commit message you may have set.



Enabling Copilot-generated commit messages

Push Configs to a Remote Repo

Once you've configured a remote, a **Git Push** button appears in the **Global Config** dialog. You can use this to copy committed configuration changes to your remote.

The branch indicator will normally read `master`, as shown above, unless you have enabled GitOps with an appropriate license.

Collapse Actions

If you enabled the **Git Settings** > **Collapse Actions** option, you will be offered a combined **Commit & Push** button in the **Global Config** dialog.

At the Group level, enabling **Collapse Actions** adds a combined **Commit & Deploy** option to the modal you see after clicking **Commit** for the Group config.

Enabling **Collapse Actions** with a remote repo simplifies the **Git Changes** confirmation dialog. It substitutes just a commit **Message** text box, with **OK** and **Cancel** buttons - omitting the diff view. Don't enable this option if you prefer to inspect configuration changes before committing.

Back Up Configs Out of Band

Once you've configured your remote repo, changes to all Cribl Stream config files under the `$CRIBL_HOME` directory will normally be backed up to your remote whenever you select the above **Git Push** or **Commit & Push** UI buttons. The exceptions are:

- Any configuration files you've chosen to store outside `$CRIBL_HOME`.
- Any configuration files residing in paths you've added to `.gitignore`.

To back up these files to your remote, you'll need to either:

- Change the above paths/settings, or
- Use an external `git` client to manually push them.

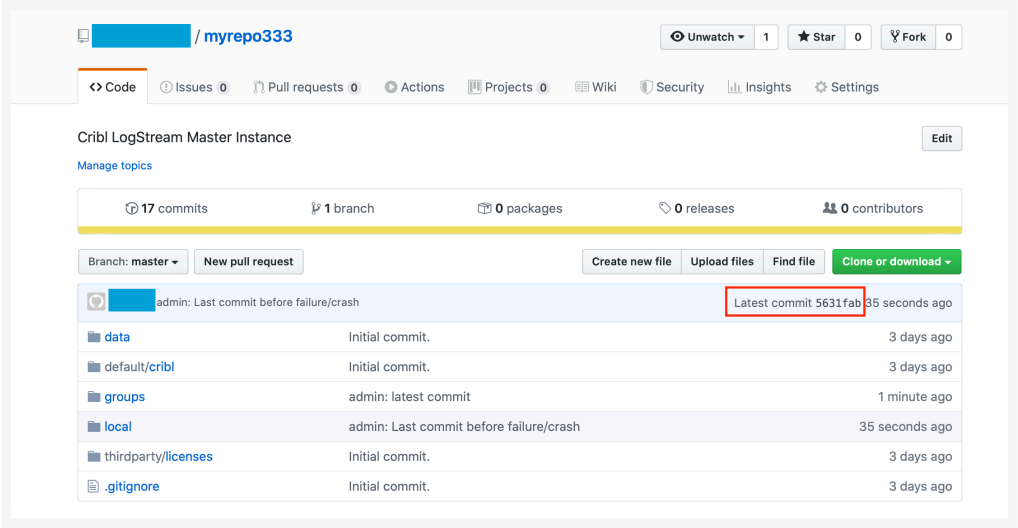
Troubleshooting Push Errors

To resolve errors commonly encountered when pushing to a remote repo, see:

- Git Push Errors
- Git Remote Repos & Trusted CAs

Restore Leader from a Remote Repo

If a remote repo is configured and has the latest known good Leader configuration, this section outlines the general steps to restore the config from that repo.



Restoring from remote repo

Let's assume that either the entire `$CRIBL_HOME` directory of the Leader is corrupted, or you're replicating the remote repo's config onto a fresh instance. Let's also assume that the remote repo has the form: `git@github.com:<username>/<reponame>.git`.

1. **Important**: In a directory of choice, untar the **same Cribl Stream version** that you're trying to restore, but do not start it.

Make sure you download the **matching** Cribl Stream version. The Cribl Download page foregrounds current version of Cribl Stream, but provides a link to this archive of prior releases.

2. Change directory into `$CRIBL_HOME` and initialize `git`:

```
# git init
```

Copy

3. If you are using SSH key authentication, specify the key using the following command:

```
GIT_SSH_COMMAND='ssh -i </path/to/github/repo>.key -o IdentitiesOnly=yes'
```

Copy

As an alternative to executing `GIT_SSH_COMMAND` on the fly, you can set your key in `$CRIBL_HOME/.git/config` with: `git config core.sshCommand "ssh -i /path/to/key"`

Or set it globally with: `git config --global core.sshCommand "ssh -i /path/to/key"`

4. Ensure that you have proper access to the remote repo:

```
# git ls-remote git@github.com:<username>/<reponame>.git
56331fabb4822eaec4ca0ffd008d6e9974c1e419f        HEAD
5631fabb4822eaec4ca0ffd008d6e9974c1e419f         refs/heads/master
```

Copy

5. Next, add/configure the remote:

```
# git remote add origin git@github.com:<username>/<reponame>.git
```

Copy

6. Now set up your local branch to exactly match the remote branch:

```
# git fetch origin
# git reset --hard origin/master
```

Copy

7. Finally, to confirm that the commits match, run this command while in `$CRIBL_HOME`:

```
# git show --abbrev-commit
```

Copy

You should see output indicating that `HEAD` points to both `master` and `origin/master`, as in this example:

```
commit 5631fab (HEAD -> master, origin/master)
Author: First Last <email@example.com>
Date:   Fri Jan 31 10:16:07 2020 -0500

    admin: Last commit before failure/crash
......
```

Copy

Step 6 above pulls in all the latest configs from the remote repo. Step 7 confirms the local repo matches the remote. You should now be able to start the Leader as normal. Once up and running, Workers should start

checking in after about 60 seconds.

Verify cribl.secret

The `cribl.secret` file - located at `$CRIBL_HOME/local/cribl/auth/cribl.secret` - contains the secret key that is used to encrypt sensitive settings on configuration files (for example, AWS Secret Access Key). Make sure this file is properly restored on the new Leader, because it is required to make encrypted conf file settings usable again.

.gitignore File

A `.gitignore` file specifies files that `git` should ignore when tracking changes. Each line specifies a pattern, which should match a file path to be ignored. Cribl Stream ships with a `.gitignore` file containing a number of patterns/rules, under a section of the file labeled `CRIBL SECTION`.

.gitignore

```
# Do NOT REMOVE CRIBL and CUSTOM header lines!
# DO NOT REMOVE rules under the CRIBL section as they may be reintroduced on update.
# You can ONLY comment out rules in the CRIBL section.
# You can add new rules in the CUSTOM section.
### CRIBL SECTION -- DO NOT REMOVE ###
default/ui
default/data/ui
/bin/
/lib/
/log/
/pid/
data/uploads
/diag/
**.tmp
/state/
**/local/cribl/messages.yml
**/local/edge/messages.yml
**/default/cribl/lakes.yml
**/local/cribl/lakes.yml
/thirdparty/
*.cpuprofile
**/auth/ssh/git.key
ha-version.txt
### CUSTOM SECTION -- DO NOT REMOVE ###

<User-defined patterns/rules go here>
```

Copy

Worker Deployment: How to Handle Customer-Defined `.gitignore` Exclusions

The `bundler` object in `cribl.yml` allows you to deploy files to GitOps that are usually excluded via `.gitignore`, like `cribl.secret` and `secrets.yml`. This facilitates loading custom secrets per environment and Group.

Files listed in the `bundler` object are deployed, but they are never committed to GitOps. You must implement your own logic (outside GitOps) to place the correct secret files for each environment and Group prior to deployment to ensure that the correct secrets are present. This approach safeguards sensitive information while leveraging the `bundler` functionality within your GitOps workflow.

To leverage this functionality for deployment-critical files, configure the Leader's `cribl.yml` file located at `$CRIBL_HOME/local/cribl/cribl.yml` using the `bundler.bundleGitIgnoredPatterns` setting. This ensures essential files reach Workers, even if they are not tracked with version control due to customer-specific `.gitignore` rules. It is not necessary to restart the Leader after editing the Leader's `cribl.yml` file.

Example: Use `.gitignore` with `bundler`

To exclude all Cribl secrets from Git, in `.gitignore`, enter `local/cribl/auth/cribl.secret` to exclude the Leader's `cribl.secret` file as well as the glob pattern `**/local/cribl/auth/cribl.secret` to exclude all `cribl.secret` files located in the `local/cribl/auth` subdirectory of any Group within the `groups` directory, regardless of the `groupName`.

```
# Exclude Leader's cribl.secret
local/cribl/auth/cribl.secret

# Exclude cribl.secret for all Worker Groups
**/local/cribl/auth/cribl.secret
```

Copy

To exclude only a specific group's `cribl.secret` file, use `groups/{groupName}/local/cribl/auth/cribl.secret` to `.gitignore` instead of `**/local/cribl/auth/cribl.secret`.

In `bundler.bundleGitIgnoredPatterns`, add `local/cribl/auth/cribl.secret` to ensure that the `cribl.secret` file is included in the config bundle during deployment without being included in version control. Patterns in `bundler.bundleGitIgnoredPatterns` are evaluated with a base directory of `$CRIBL_HOME/groups/<group>`. The bundler does not require the Leader's `cribl.secret`.

```
bundler:
  bundleGitIgnoredPatterns:
```

3.3.3. Configure Worker Group Settings

Required Permission: **Admin**

Worker Group Settings provide a granular way to manage the behavior of the data plane (the Worker Groups that handle data processing). These settings allow you to configure features and behaviors at the Worker Group level, ensuring that you can tailor performance, security, and data flow to the specific needs of different Pipelines or environments.

You can configure the following settings at the Worker Group level:

- Teleporting
- Throttling
- Upgrading
- Security

To access and configure Worker Group settings:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Select the **Worker Group Settings** tab.

Not all settings are available for Cribl.Cloud deployments.

General Settings

Overview

You can configure the following options on this tab:

Enable teleporting to Workers:Toggle on to enable authenticated access to Workers' UI from the Leader Teleporting. Toggle off to disable.

Description: Optionally, add or edit the Worker Group description.

Tags: Use tags to organize Worker Groups into logical categories. Then, you can search by tags on the **View all Groups** interface. This search filters the list of Worker Groups, showing only those with the tag you entered.

Time to keep disconnected Nodes: Configure how long the Worker Group should retain information about a Worker Node that disconnects from the Leader. Format examples: 8h, 5d, 1w (format defaults to days if you only specify a number). Default: 1 day. This setting is only available for customer-managed (hybrid or on-prem) Worker Groups.

API Server Settings

General

You can set the following options for the API server.

Host: The hostname or IP address you want to bind the API server to. Defaults to `0.0.0.0` for Leader, standalone Stream Workers, and Worker Groups.

You can override the value with the `CRIBL_API_HOST` environment variable.

Port: API port to listen to.

Defaults to `9000` for Leader, standalone Stream Workers, and Worker Groups.

You can override the value with the `CRIBL_API_PORT` environment variable.

TLS

For information on TLS options, see the documentation for any Source or Destination that supports TLS.

Advanced

Set the following advanced options for the API server:

- **Retry count**: The number of times to retry binding to the API port. Default is `120`.
- **Retry period**: The period between consecutive retries for API port binding, in seconds. Default is `5`.
- **URL base path**: The URL base path from which to serve all assets. Setting a URL base path may be useful when operating behind a proxy server.
- **Listen on port**: Toggle on (default) to expose the API service to the network on the configured API port.
- **Local UI access**: Toggle on (default) to allow direct browser access to the UI for Worker Nodes.
- **Enable API cache**: Toggle on to enable browser caching of frequent API requests. Default is toggled on. Toggling off can slow the the UI response time.

Default TLS Settings

See the System-wide TLS Settings Including Ciphers.

Limits

The **Limits** tab provides access to controls for metrics, storage, metadata, jobs, the Redis cache and connections to it, and CPU settings.

Metrics

See Controlling Metrics Volume.

Storage Limits

You can configure the following options for storage. In **General Settings**, open **Limits** and then select **Storage**:

Sample size limit: Maximum file size, in binary units (`KB`, `MB`), for sample data files. Maximum: `3 MB`. Default: `256 KB`.

Min free disk space: The minimum amount of disk space on the host before various features take measures to prevent disk usage (KB, MB, etc.). Default: `5 GB`. To avoid the "Unable to write to the filesystem, disk space low" error, ensure that the available disk space remains above this threshold by monitoring disk usage and configuring alerts.

Worker Process PQ size limit: Highest accepted value for the **Queue size limit** option used in individual Sources' and Destinations' persistent queues. Default: `1 TB`. Consult Cribl Support before increasing beyond this value.

Metadata Limits

Event metadata sources: List of event metadata sources to enable. No sources are enabled by default.

Job Limits

Disable jobs/tasks: Cribl Edge only. See Fleet Settings for more information.

Concurrent job limit: The total number of jobs that can run concurrently. Defaults to `10`.

Concurrent system job limit: The total number of **system** jobs that can run concurrently. Defaults to `10`. Minimum `1`.

Concurrent scheduled job limit: The total number of **scheduled** jobs that can run concurrently. This limit is set as an offset relative to the

Concurrent job limit. Defaults to `-2`.

Skipped jobs indicate that a Group's **Concurrent job limit** has been reached or exceeded. Increase this limit to reduce the number of skippable jobs. For resource-intensive jobs, this might require deploying

more Worker Nodes.

Concurrent task limit: The total number of tasks that a Worker Process can run concurrently. Defaults to `2`. Minimum `1`.

Concurrent system task limit: The number of system tasks that a Worker Process can run concurrently. Defaults to `1`. Minimum `1`.

Task usage percentage limit: Value, between `0` and `1`, representing the percentage of total tasks on a Worker Process that any single job may consume. Defaults to `0.5` (50%).

Task poll timeout: The number of milliseconds that a Worker's task handler will wait to receive a task, before retrying a request for a task. Defaults to `60000` (60 seconds). Minimum `10000` (10 seconds).x

Artifact reaper period: Interval on which Cribl Stream attempts to reap jobs' stale disk artifacts. Defaults to `30m`.

Finished job artifacts limit: Maximum number of finished job artifacts to keep on disk. Defaults to `100`. Minimum `0`.

Finished task artifacts limit: Maximum number of finished task artifacts to keep on disk, per job, on each Worker Node. Defaults to `500`. Minimum `0`.

Manifest flush period: The rate (in milliseconds) at which a job's task manifest should be refreshed. Defaults to `100` ms. Minimum `100`, maximum `10000`.

Manifest buffer size limit: The maximum number of tasks that the task manifest can hold in memory before flushing to disk. Defaults to `1000`. Minimum `100`, maximum `10000`.

Manifest reader buffer size: The number of bytes that the task manifest reader should pull from disk. Defaults to `4kb`.

Job dispatching: The method by which tasks are assigned to Worker Processes. Defaults to `Least In-Flight Tasks`, to optimize available capacity. `Round Robin` is also available.

Job timeout: Maximum time a job is allowed to run. Defaults to `0`, for unlimited time. Units are seconds if not specified. Sample entries: `30`, `45s`, `15m`.

Task heartbeat period: The heartbeat period (in seconds) for tasks to report back to the Leader/API. Defaults to `60` seconds. Minimum `60`.

Redis

Cache

Key TTL in seconds: Maximum time to live of a key in the cache (seconds). `0` indicates no limit. Defaults to `10 minutes`.

Key limit: Maximum number of keys to retain in the cache. `0` indicates no limit. Defaults to `0`.

Cache size limit (bytes): Maximum number of bytes to retain in the cache. `0` indicates no limit. Defaults to `0`.

Service period (seconds): Frequency of cache limit enforcement. Defaults to every `30 seconds`.

Server assisted: Default is toggled off. When toggled on, the following control appears.

Client tracking mechanism: Mechanism for invalidation message delivery. In default mode, the server remembers which keys a client has requested and only sends invalidations for those, using more Redis server memory. In broadcast mode, it sends all invalidations, requiring more processing by Cribl Stream.

Connections

Reuse Redis connections: Toggle on if you want Cribl Stream to try to reuse Redis connections when multiple Redis Functions (or references to them) are present. When enabled, displays the following additional control:

- **Connection limit**: The maximum number of identical connections allowed before Cribl Stream tries to reuse connections. Defaults to `0`, meaning unlimited connections are allowed (equivalent to leaving **Reuse Redis connections** toggled off). Setting a non-zero integer value forces Cribl Stream to try to reuse connections for each individual Worker Process (**not** to reuse connections **among** Worker Processes).

To understand why and when to employ these controls, see Reusing Redis Connections.

Lookups

Lookup file max size: Maximum allowed disk space for persisting single lookup data file.

Defines the maximum allowed disk space for a single lookup file. If you're using very large lookup files, increase this limit. However, a larger limit can consume more disk space and memory on the Worker Nodes, potentially impacting performance.

Lookup files disk limit: Specifies the total maximum disk space available for all lookup data and their associated metadata files. This is an aggregate limit, meaning it applies to the sum of all lookup files managed by that node. This limit is crucial for preventing a runaway process from filling up a Worker Node's disk with lookup files.

Other

CPU profile TTL: The time-to-live for collected CPU profiles.

Default managed node heartbeat period: How many seconds a managed Worker will wait to send back a heartbeat to the Cribl control plane.

Config bundle download timeout: How many seconds a Cribl Stream Worker will wait for a successful Leader connection before canceling a download of a new configuration bundle. This timeout helps prevent Workers from hanging indefinitely when there are network issues or other delays during the download process. A `0` value means wait indefinitely, which could cause Workers to hang.

Minimum reconnect interval: The minimum time to wait before attempting to reconnect to the Leader after being disconnected or when the Leader is not available. The interval is doubled with each failed reconnect attempt until the maximum limit is reached.

Maximum reconnect interval: The maximum time to wait before attempting to reconnect to the Leader after being disconnected or when the Leader is not available.

Random reconnect interval: Upper limit on the random time added to each reconnect interval. Set this to a value other than zero to randomly spread out reconnection attempts. This can prevent the Leader being overwhelmed by reconnection attempts by a large number of Workers.

Rate limit bytes per second: The maximum number of bytes per second that a Worker Node can ingest. If `0` or not configured, no rate limiting is applied.

Proxy Settings

Use proxy env vars: Honors the `HTTP_PROXY/HTTPS_PROXY` environment variables. Defaults to toggled on.

Cribl prioritizes environment variables for proxy settings in this order: Process, User, and System.

If your Cribl service is managed by a service manager other than systemd (such as upstart or init), the `Use proxy env vars` toggle might not behave as expected because Cribl might prioritize environment variables set by the service manager instead of using the proxy settings you intended.

Sockets

Directory: Holds sockets for inter-process communication (IPC), such as communications between a load-balancing process and a Worker Process. Defaults to `/tmp` (your system's temp directory).

Shutdown Settings

Drain timeout (sec): Determines how long a Cribl server will wait for writes to complete before the server shuts down on individual Worker Processes. If you notice that Workers are under-ingesting available data upon shutdown or restart, increase the `10`-second default. Acceptable range of values: minimum `1` second, maximum `600` seconds (10 minutes).

SNI

Disable SNI-based connection routing: This setting affects how the Cribl Stream control plane routes connections.

Do not change this setting without consulting Support first.

Support

Feature Flag Overrides: Cribl thoroughly tests all releases in Cribl.Cloud environments to ensure stability and reliability. However, new features can sometimes behave unpredictably in customer-managed data environments. The scale of some environments and the complexity of many possible configurations make it difficult to test for every possible scenario, occasionally leading to unforeseen issues.

To ensure the stability of your production environment, Cribl Stream includes a safety mechanism that makes it possible to disable specific new features if they are causing unexpected issues. This is a customer-first tool designed to minimize impact and avoid the need to rollback upgrades or wait for a full patch release.

If you suspect that a new feature is impacting your environment, please contact the Support team for assistance. This process ensures that Cribl can quickly stabilize your environment while also collecting the necessary information to improve Cribl Stream for everyone.

Do not change this setting without consulting Support first.

Worker Processes

Process count: Indicates the number of Worker Processes to spawn.

- Positive numbers specify an absolute number of Workers. Negative numbers specify a number of Workers relative to the number of CPUs in the system, for example: {`<number of CPUs available>` minus `<this setting>`}.
- The default is `-2`.
- Cribl Stream will correct for an excessive + or - offset, or a `0` entry, or an empty field. Here, it will guarantee at least the **Minimum process count** set below, but no more than the host's number of CPUs available.

Minimum process count: Indicates the minimum number of Worker Processes to spawn. Overrides the **Process count**'s lowest result.

- A `0` entry is interpreted as "default," which here yields `2` Processes.

Memory (MB): Amount of heap memory available to each Worker Process, in MB.

- Defaults to `2048` (see Estimating Memory Requirements).
- Heap memory is allocated dynamically from the OS as it is requested by the Process up to the amount dictated by this setting.

- The **Memory (MB)** setting does not govern external (`ext`) memory.

If a Process needs more than the allocated amount of heap memory, it will encounter an `Out of memory` error, which is logged to `$CRIBL_HOME/log/cribl_stderr.log`. This error indicates either:

- A memory leak.
- The memory setting is too low for the workload being performed by the Process (and possibly, by extension, that the total memory provisioned for the host is insufficient for all Processes).

Cribl support can help you troubleshoot if you experience this error.

The following controls are also available on this tab to optimize Worker Processes' throughput on startup.

Max connections at startup: Maximum number of connections accepted at Worker Process startup. Defaults to `1`. Enter a negative integer for unlimited connections.

Startup throttling duration (ms): Maximum time (in milliseconds) to continue throttling connections after Worker Process startup. Defaults to `10000` ms (10 sec.) Enter `0` to disable throttling.

Load throttle %: Sets a threshold to prevent overwhelming Workers. If 90% of a Worker Process' CPU utilization readings exceed this threshold over one minute, the process will reject new connections until the CPU load stabilizes. Another process that is below the threshold will accept the connection the next time it is established. Defaults to `0%` (no throttling). Enter a percentage between `1-100` to enable throttling.

You can configure the CPU saturation threshold, but the 90% sampling trigger is not configurable. Also, `_raw stats > cpuPerc` values might diverge from your **Load throttle %** threshold. This is because `cpuPerc` is sampled and averaged once per minute, whereas the **Load throttle %** is evaluated every second, with a rolling 1-minute lookback sample. (These intervals are also not configurable.)

Enable heap snapshots: Default is toggled off. Toggle on for Cribl Stream to automatically create memory snapshots for Worker Processes when they approach or exceed memory limits. Only the two most recent heap snapshots are retained. Older snapshots are automatically deleted. This behavior cannot be modified.

The **Enable heap snapshots** is available for hybrid or on-prem deployments only. This setting should only be enabled if recommended by support or if you are experiencing out-of-memory issues. Be aware that enabling this setting may impact system performance and storage usage.

Other Settings

This page's remaining options work essentially the same way as their counterparts in **Settings** > **Global**. Use the following links for details about: logging levels/redactions, access management, security, scripts, and

3.3.4. GitOps

With Cribl Stream's GitOps features, you can manage Cribl Stream configuration with standard version-control systems and CI/CD flow. You can separate development from production configurations, and thus, safely build and continuously deploy your observability pipelines. The production environment will be read-only, and thus strongly protected against unauthorized or unintended changes. This intentionally restrictive approach is a good fit for some but not all organizations.

The GitOps workflow described here assumes that your development, staging, and production environments are identical or nearly identical. If, on the other hand, your development environment differs significantly from production (in its branching structure, for example, or if users have wide latitude for experimentation), you should consider solutions other than GitOps.

GitOps requires a Distributed Cribl Stream deployment with a certain license tier.

Cribl.Cloud does not support GitOps.

How to Use This Doc

This doc has three sections.

- Work through all three sections, in order, once.
- Once you have done that, you are finished with the first two sections, Preparing Your Git Repo and Preparing Cribl Stream.
- Next, incorporate the HTTP request from the third section (GitOps Workflow) into your CI/CD tool infrastructure.
- From then on, your normal production routine will be to just repeat the GitOps Workflow (that is, the third section) again and again.

You'll perform setup steps in the Cribl Stream UI, in your Git management system, and at the command line. Screenshots are in light mode for Cribl Stream, and in dark mode for GitHub examples.

Environment and branch names have been chosen to help you keep track of what you're doing - if you decide to use different ones, exercise care. We've chosen names that always make it clear whether we're talking about environments or about branches:

- Development and Production will be your Cribl Stream environments.
- `dev` and `prod` will be the branches in your Git repo.

Procedures begin with instructions that are agnostic to the Git management system you want to use, whether that is free GitHub, GitHub Enterprise, GitLab, Bitbucket, or a self-hosted Git server. The detailed instructions that follow are for use with free GitHub. If you're using one of the other Git management systems, you'll need to substitute appropriate instructions for the ones that begin "To do this in GitHub …"

Check out Cribl's free GitOps sandbox for a "deep dive" and to work through a practical example.

If you encounter either of the following situations, see the linked appendix:

- You need to upgrade your GitOps-enabled Cribl Stream deployment.
- You are not using systemd, and need to override the defaults using environment variables.

Preparing Your Git Repo

The procedure in this section only needs to be done once. When you've completed it, you can prepare Cribl Stream for GitOps.

Creating SSH Public and Private Keys

You'll use SSH to connect to your Git management system; Cribl does **not** recommend using basic authentication for GitOps.

At the command line, generate public and private SSH keys, specifying Ed25519 as the key type:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

Copy

Verify that the `~/.ssh/` directory contains both of the newly-generated keys:

- `id_ed25519` (the private key that Cribl Stream will use).
- `id_ed25519.pub` (the public key that your Git management system will use).

Copy the public SSH key to your clipboard:

```
cat ~/.ssh/id_ed25519.pub
```

Copy

Associate the public key with your Git management account.

To do this in GitHub:

1. If you do not have a GitHub account, create one.
2. Sign in to your GitHub account.
3. On the SSH and GPG keys page, click **New SSH key**.
4. Paste the key in the **Key** field.
5. Click **Add SSH key**.

Creating a Private Git Repo

You need to create a dedicated, private Git repo - your Development environment will write to it, and your Production environment will sync to it. Create a private repo named `cribl`.

To do this in GitHub:

1. In GitHub, select **Create new...** on the top bar, and choose **New repository**.
2. Name the repo `cribl`.
3. Change the repo from **Public** to **Private**.
4. Select **Create repository**. The new repo's landing page will open.

Preparing Cribl Stream

The procedure in this section only needs to be done once. When you've completed it, you can move on to testing and using GitOps.

Setting Up Your Environments

In your distributed Cribl Stream deployment, set up two separate environments, one named `Development` and the other named `Production`. Cribl strongly recommends that you use systemd to manage the `cribl` service.

Complete the following steps in **both** environments:

- Create one Cribl Leader and two Cribl Workers.
- On the Leader, install a Cribl Enterprise License.
- On the Leader, install `git` using the following command (or the equivalent for your OS): `sudo yum install git -y`

Then, in the Production environment:

- On the Leader, install `jq` - the GitOps workflow instructions below use `jq`.
    - Run this command (or the equivalent for your OS): `sudo yum install jq -y`

- Note the Production Leader's IP address for use later on.

All Leader Nodes should be on the same version to avoid backward compatibility errors.

Configuring Your Production Leader

First, you'll ensure that your Production Leader and your Git repo can communicate; then, you'll put your Production Leader into read-only mode.

Creating the `gitops` User

1. Create a new Local User called `gitops`.
2. On the Production Leader's local filesystem, temporarily record the password for the `gitops` User. This password well be referred to as `GITOPS_USER_PASSWORD` in later steps.
3. Associate the default `gitops` Role with the `gitops` User. (See Adding Roles.)
    - This Role allows only to `POST` to the `/version/sync` endpoint of the Cribl API (in the API Reference, see **versioning**).

Adding a Secret to Git

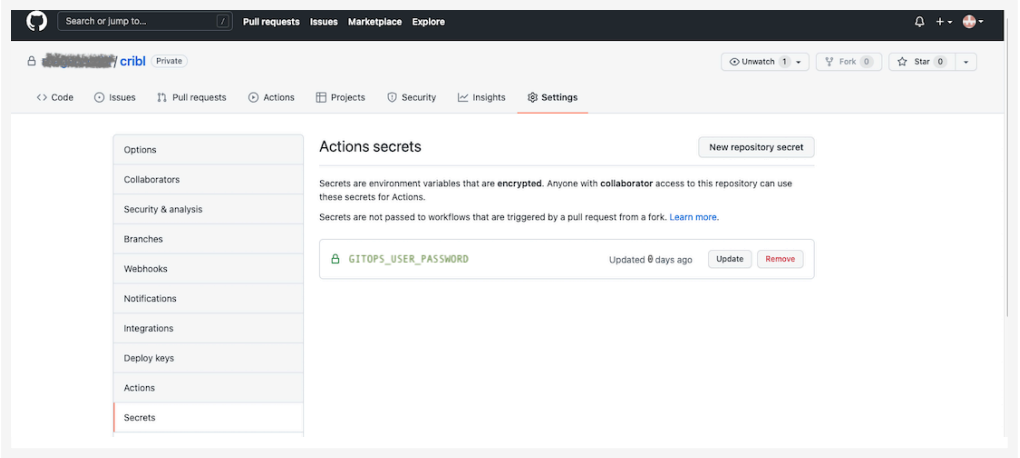
Your Git repo needs to be able to communicate with your Production Leader. This requires that you store a secret in your Git management system. We'll use `GITOPS_USER_PASSWORD` as the secret.

From the Production Leader, upload the `GITOPS_USER_PASSWORD` to the Git management system (or perform an equivalent operation) now.

To do this in GitHub:

1. In the GitHub site for your repo, navigate to **Settings** > **Secrets and variables** > **Actions**.
2. Click **New repository secret** and follow the prompts.

When done, you should see a `GITOPS_USER_PASSWORD` row under **Actions secrets**:



The GitOps user password has been uploaded to GitHub

Connecting Your Production Environment to Your Git Repo

In Cribl Stream, in the UI for the Production Leader, select **Settings** in the sidebar, then select **Global** > **Git Settings**.

In the **Remote** tab:

1. In **Remote URL**, enter the URL for your private Git repo.
    - On GitHub, the **Remote URL** will look like this: `git@github.com:<your_GitHub_username>/cribl.git`
2. Make sure that in the **Authentication type** drop-down, `SSH` (the default) is selected.
3. In **SSH private key**, copy and paste the private key (`~/.ssh/id_ed25519`).

The key will paste in with an appended newline below it. Do not delete this newline before you save the Remote Settings, or else you will trigger an error of the form: `fatal: Could not read from remote repository.`

4. Toggle **SSH strict host key checking** off. This will allow the remote repository to accept your host key information. If you miss this step, you will trigger an error.
5. Click **Save**.
6. Toggle **SSH strict host key checking** on again.
7. Click **Save** (again!).
8. **Commit** and **Push**.

Check your private Git repo - in the `master` branch, you should now see your commit.

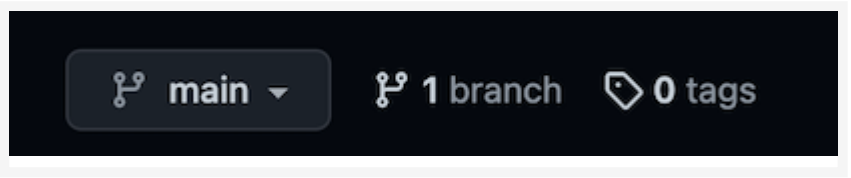
Setting Up the `prod` Branch

In this section, we'll set up the Git `prod` branch; later on, we'll set up `dev`.

Recall that your Git repo has one branch, named `master`. Now, rename `master` to `prod`.

To do this in GitHub:

1. In your GitHub repo's landing page (for example, `https://github.com/<your_Git_username>/cribl`), select the **1 branch** tab. (That's what the branches tab will be named, assuming that the sole branch is `master`).



The branches tab

2. In the `master` branch's row, select ••• at far right, then select **Rename branch**, and rename `master` to `prod`.

Now, configure your Cribl Stream instance to be using the `prod` branch:

1. In Cribl Stream, select **Settings** > **Global** > **Git Settings**.
2. In the **General** tab, set **Branch** to `prod`.
3. Save.

Making the Production Leader Read-Only

In a terminal on the Production Leader, become root:

```
sudo su -
```

Copy

Add `systemctl` overrides to set environment variables that will (a) make the Production Leader read-only upon startup, and (b) set the branch to `prod`.

1. Use `systemctl` to open the `override.conf` file in an editor:

```
systemctl edit cribl
```

Copy

2. When prompted, add the following overrides:

```
[Service]
Environment=CRIBL_GIT_OPS=push
Environment=CRIBL_GIT_BRANCH=prod
```

Copy

3. Restart:

```
systemctl daemon-reload && systemctl restart cribl
```

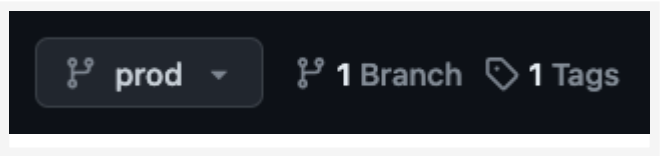
Copy

Creating the `dev` Branch

Recall that your Git repo has one branch, named `prod`. Now, create a new branch named `dev`, **from your existing** `prod` branch.

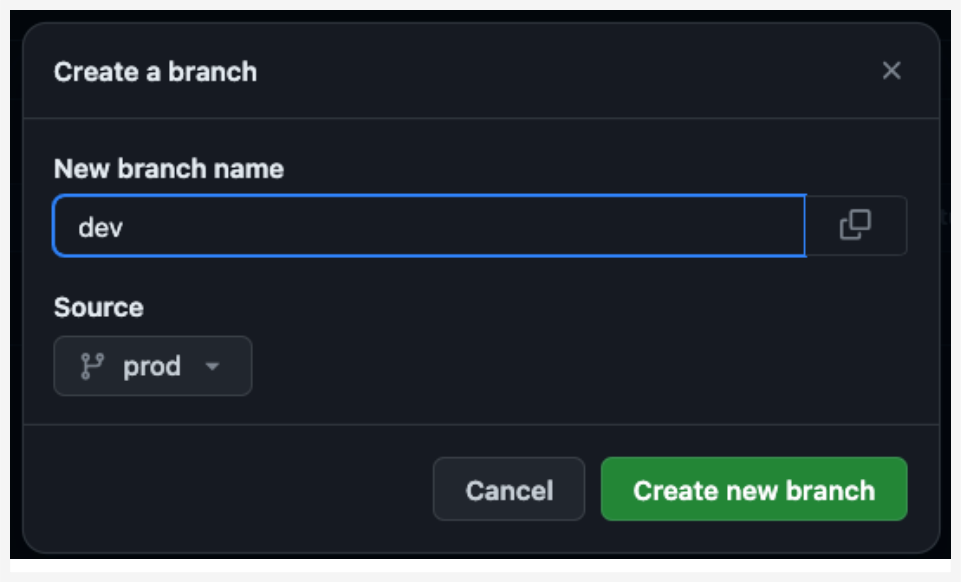
To do this in GitHub:

1. In your GitHub repo's landing page (for example, `https://github.com/<your_Git_username>/cribl`), select the **1 branch** tab. (That's what the branches tab will be named, assuming that the sole branch is `prod`).



The branches tab

2. Click **New branch** to open the **Create a branch** modal.
3. Make sure that **Source** is set to `prod`.
4. In the **New branch name** field, enter `dev`.



Creating the `dev` branch

5. Click **Create new branch** to close the modal.
6. Back in the **Branches** tab, click **Overview** to refresh the tab - you should now see both the `prod` and `dev` branches listed.

Configuring Your Development Leader

First, you'll point your Development Leader to the `dev` branch in the remote repo. Then, you'll ensure that one or more users exist that can push configs to that repo.

Pointing Your Development Leader to Its Branch

In Cribl Stream, in your Development Leader's UI, select global **Settings** (lower left) > **Git Settings**.

In the **Remote** tab:

1. In **Remote URL**, enter the URL for your private Git repo - and if the URL includes a Git username, this should be `gitops`, the one you created above.
    - On GitHub, the **Remote URL** will look like this: `git@github.com:<your_GitHub_username>/cribl.git`
2. Make sure that in the **Authentication type** drop-down, `SSH` (the default) is selected.
3. In **SSH private key**, copy and paste the private key (`~/.ssh/id_ed25519`).

The key will paste in with an appended newline below it. Do not delete this newline before you save the Remote Settings, or else you will trigger an error of the form: `fatal: Could not read from remote repository.`

4. Toggle **SSH strict host key checking** off. This will allow the remote repository to accept your host key information. The remote host's information is stored in `~/.ssh/known_hosts`. If you miss this step, you will trigger an error.
5. Click **Save**.
6. Toggle **SSH strict host key checking** on again.
7. Click **Save** (again!), but do not **Commit** or **Push** yet.

In the **General** tab:

1. Set **Branch** to `dev`.
2. Leave **GitOps Workflow** set to `None`.
3. Click **Save**, but do not **Commit** or **Push** yet.

Restart the Development Leader. When it comes back up, it will be able to write to your Git repo.

Setting Up Users to Push to the Remote Repo

You will need one or more users capable of pushing configs to the remote repo. There is more than one way to achieve this. Your choice will depend on the needs of your organization.

- Any user with the `admin` Role already has ability to push to the remote repo. If this is your preferred approach, skip to the Pointing Your Development Leader to Its Branch.
- If you prefer to allow one or more **regular** users to push to the remote repo, **without** having the power of the `admin` Role, you can create such users by completing the procedure below.

First, edit the `roles.yml` file to make a new `push_to_remote` Role. (We're using the command line here because you cannot perform this operation in the UI.)

1. SSH into the Leader Node.
2. `cd` into your `$CRIBL_HOME/local/cribl` directory.
3. Open or create the `roles.yml` file in a text editor.
4. Without changing the existing text in the file, paste in the following snippet:

```
push_to_remote:
  policy:
    - POST /version/sync
    - POST /version/commit
    - POST /version/push
    - GET /system/settings/git-settings
  description: Members with this role can push configs to a remote repo.
```

Copy

5. Restart Cribl Stream.

Next, back in the UI, create a new `config_pusher` User.

1. In global Settings (lower left) > **Access Management** > **Local Users**, click **Add User** to open the User creation modal.
2. Name the new Local User `config_pusher`.
3. In addition to the usual configuration steps, use the **Roles** drop-down to associate the `push_to_remote` Role with the `config_pusher` User.
4. Click **Save**.
5. **Commit** and **Push**.

Verify That Your New User Can Push to the Remote Repo

1. Log out, then log back in as `config_pusher`.
2. The **Version Control** drop-down should now display a **Push** button.

Later, when you are ready to push new configs to the remote repo, click this new **Push** button.

GitOps Workflow

If you've completed the above preparatory sections, your GitOps workflow is ready go. Here's an overview, with links to detailed procedures.

- Work in the Development Environment: All your commits and testing happen in the Development environment first; you push changes to the `dev` branch in your Git repo.
- Merge Changes from dev to prod: When you are satisfied that your changes are ready for production, you create a pull request that brings the `prod` branch up-to-date with `dev`. This is all in your Git management system, and changes nothing in your Cribl Stream Production environment.
- Sync Your Production Leader to Git: For your changes to take effect in production, you will send an HTTP request to the `/version/sync` endpoint of the Cribl API, on the Production Leader. Once you are in production, it will be your CI/CD infrastructure - not you - that sends the request. This will cause the Production Leader to pull in the latest changes from your Git repo's `prod` branch - without violating the Production Leader's read-only status.

The procedures include a harmless test task so that you can validate your setup and acquire a feel for the workflow.

Working in the Development Environment

In this step, you'll make changes in the Development environment, and subsequently propagate them to Production. To test this with a minimal unit of work, start by adding a Route.

In Cribl Stream, in the Development Leader's UI, navigate to **Routing** > **Data Routes**.

1. Click **Add Route**.
2. Configure the new Route as follows:

Name	Filter	Pipeline	Output
`test-gitops`	`true`	`devnull`	`devnull:devnull`

3. Click **Commit**.
4. In the **Git Changes** modal, add a commit message and click **Commit & Deploy**.
5. Finally, push the changes to your remote repo via the **Global Config** dialog, whose appearance will vary depending on how you've configured **Git Settings** > **Collapse Actions**. Make sure the dialog says that the **Git branch** you're pushing to is `dev`, as configured above.

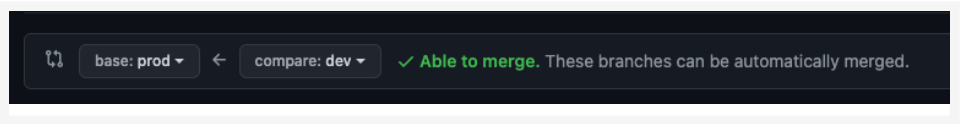
Merging Changes from `dev` to `prod`

In your Git management system's `dev` branch, you should see the new `route.yml` appearing in the `/groups/<group_name>/local/cribl/` directory.

Merge the changes from the `dev` branch to `prod` now.

To do this in GitHub:

1. Click **Compare & pull request**. Be sure that the `base` drop-down says `prod` and the `compare` drop-down says `dev`.



Comparing `dev` to `prod`

2. Add some comments.
3. Click **Create pull request**. The UI should say that you want to `merge 1 commit into prod from dev`.
4. If GitHub detects conflicts, fix them.
5. Have the appropriate person on your team review the pull request.
6. You (or the appropriate person) should now click **Merge pull request**, then **Confirm merge**.

If the merge is successful, GitHub will display a confirmation message. Your committed changes have now been merged from `dev` to `prod`.

Syncing Your Production Leader to Git

Your changes so far have all been done in the Development environment. Meanwhile, you have committed to the `dev` branch, and pushed to the `prod` branch **in your Git management system** - but your Production **environment** has not changed.

Since, for safety's sake, you made the Production environment read-only, you can neither write to it nor update it via a UI. Instead, the following must happen:

- You will send an HTTP request to the `/version/sync` endpoint of the Cribl API, on the Production Leader (in the API Reference, see **versioning**). Once you are in production, it will be your CI/CD infrastructure - not you - that sends the request.
- This will cause the Production Leader to pull in the latest changes from your Git repo's `prod` branch - without violating the Production Leader's read-only status.

Below you'll see a script you can use to send the HTTP request.

- This script is generic, and will require you to adapt it to work with the Git management system you're using.
- Once you run the script, you should see the changes you made in your Development environment (in this example, the new Route) in the Production environment, too.

Syncing to Git: Generic Script

Substitute your Production Leader's IP address and the `gitops` User's password for the placeholders.

```
#!/bin/bash

# Authenticate and Save the Token.
TOKEN=$(curl http://<Production_Leader_IP_address>:9000/api/v1/auth/login \
     -H "Content-Type: application/json" \
     -d '{"username":"gitops","password":"$GITOPS_USER_PASSWORD"}' 2>/dev/null | \
     jq -r .token)

# Set up the Authentication Header
export AUTH_HEAD="Authorization: Bearer $TOKEN"

# Make the notification call
curl -X POST "http://<Production_Leader_IP_address>:9000/api/v1/version/sync" -H "acc
```

Copy

Making the HTTP Request from CI/CD

Now that you have run one of the above scripts to prove that your GitOps workflow works, you should incorporate the script's HTTP request into your CI/CD setup. Once that is done, you're in production with GitOps! To develop, test, and push changes to production, just keep running through the whole GitOps workflow.

Appendix: Upgrading

Upgrading a Cribl Stream GitOps deployment is essentially a variation on the day-to-day GitOps workflow described in the previous section, with the key difference that you take the Production environment out of, then put it back into, read-only mode.

Once you have begun the upgrade, **do not** make any changes unrelated to the upgrade process, in either the Development or Production environment, until the upgrade process is complete. For example, do not start creating any new Routes or Pipelines.

First, in your Development environment:

1. Validate that Cribl Stream is working as expected, and that the latest changes have been pushed to the `dev` branch in your Git repo.
2. Stop the Cribl Stream Leader and back up `$CRIBL_HOME` (the Cribl install directory).
3. Use the UI to upgrade the Leader and Workers.
4. Create a PR to merge the `dev` branch into `prod`. Merge the PR.

Next, in your Production environment:

1. Stop the Cribl Stream Leader and back up `$CRIBL_HOME` (the Cribl install directory).
2. Take your Production environment **out of read-only mode** according to the instructions below.
3. Use the UI to upgrade the Leader and Workers - but **do not** commit changes after the upgrade.
4. Put the Leader **back into read-only mode** according to the instructions above.
5. Sync the Production environment to the `prod` branch in your Git repo.

Finally, confirm that all Workers in both environments have the latest configuration.

Taking the Production Leader Out of Read-Only Mode

In a terminal on the Production Leader, become root:

```
sudo su -
```

Copy

Comment out the `systemctl` overrides that set environment variables that (a) had made the Production Leader read-only upon startup, and (b) had set the branch to `prod`.

1. Use `systemctl` to open the `override.conf` file in an editor:

```
systemctl edit cribl
```

3.4. Workspaces

Workspaces offer a multi-tenancy capability, enabling you to create multiple isolated instances in your Cribl.Cloud Organization. This lets you strengthen your security, and fulfill compliance and isolation requirements.

Each Workspace offers a dedicated virtual private cloud (VPC) that acts as a separate environment within your Organization. Workspaces do not share product configurations, resources, or data flows, and each has a separate Leader Node. They do share a layer of access control (Organization-level Permissions and SSO configuration), as well as API credentials, licenses, and billing.

A centralized interface allows you to manage all your Workspaces and control access for Members and Teams to individual Workspaces.

Multiple Workspaces require a Cribl.Cloud Enterprise plan. For details, see Pricing.

An example use case scenario for creating multiple Workspaces is setting up separate environments for different business units in an enterprise. Data management, security, development, and any other units can have their own federated Workspaces, with completely separate members and permissions lists. This way, they can work in isolation without risk of interference and with increased focus.

Limitations

You can create up to 5 Workspaces per Organization.

If you need more Workspaces in your Organization (up to 50), contact Cribl Support to discuss extension options.

Multiple Workspaces require a Cribl.Cloud Enterprise plan. For details, see Pricing.

Manage Workspaces

To list the Workspaces in the Organization, in the sidebar, select **Workspace** or **Organization** and then **Manage Workspaces**. The list includes the name, ID, description, Cribl Insights retention status, tags, Cribl version number, and current state for each Workspace in the Organization.

Select a Workspace name or the ellipsis (**. . .**) in the **Actions** column to edit the Workspace name, description, and tags or delete the Workspace.

View Workspace Details

Workspaces provide a set of information that you can use when configuring data flow through their Worker Nodes.

View Access Details

To view detailed access information of your Workspace, in the sidebar, select **Workspace** and then **Access**.

This page presents a summary of information about your Cribl.Cloud Organization, as well as the **Cribl.Cloud URL** for the current Workspace.

Cribl.Cloud URL

Cribl.Cloud URL is a static address associated with the load balancer that is in front of the Leader. Customer-managed (hybrid) Worker Nodes will connect to this address on port 4200, while the Leader UI is served from this address on port 443.

You can use this URL for certain API calls and certain Collection operations coordinated by the Leader.

Static External IPs for the Leader

The **Leader NLB IPs** field in **Workspace** > **Access** lists the IPs for the Leader Network Load Balancers associated with the Workspace.

Typically, only 2 of the 3 addresses will be active (returned by DNS) at any time, while the inactive address is swapped in during infrastructure maintenance events.

You can add all those IPs to your firewall allowlist to ensure that the load balancers are accessible across your hybrid deployment.

Stream Worker Group Details

Stream Worker Group Details provides information about Cribl-managed Worker Groups in Cribl.Cloud, if any are provisioned.

If no Worker Group is provisioned, you can select **Provision Now** to provision infrastructure for the Group. After a lag, the Group will be ready to process data, and this modal's remaining fields will populate.

For a provisioned Worker Group, you see the following information:

Field	Description
Trust	AWS IAM role ARN for Cribl-managed Workers in this Worker Group. Use this ARN as the principal in AWS trust policies when configuring cross-account access.

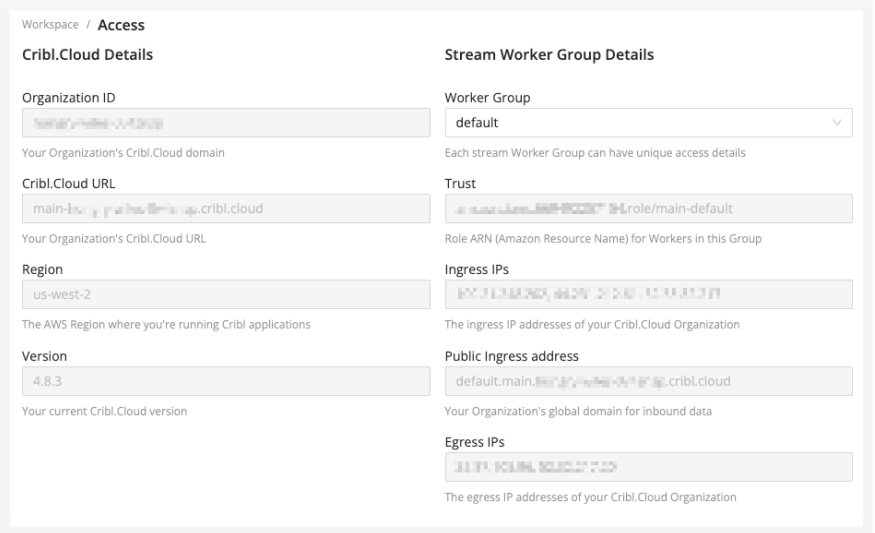
Field	Description
Ingress IPs	The IPv4 addresses of Worker Groups' load balancers, also used when receiving data from Push Sources. These addresses will remain constant, so you can build firewall rules around them. Three Ingress IPs are provided for each Worker Group, one on each Availability Zone, assuming the Group has at least three Workers. In a Worker Group of fewer than three Workers, one of the IPs will be inactive.
Public Ingress address	Each Group's domain for inbound data. This address prepends the Group name to the Organization's global domain name. It does not append ports per data type - you can obtain these from the Data Sources tab.
Egress IPs	Your Cribl.Cloud Organization's current public IP addresses. These addresses are Group-specific and also dynamic: Cribl will occasionally update them when we need to rescale core infrastructure.

Public Ingress address and **Egress IPs** are used for both outbound connections from the Workers and Pull Sources.

Because individual **Ingress IPs** can become inactive, Cribl recommends that you send your data to the **Public Ingress address** instead of directly to IPs.

Egress IPs are not static.

If you need a static egress IP, contact Cribl Support.



Access Details modal for a provisioned Group

Configuring Stream Groups (beyond the `default` Group) requires a certain plan. For details, see Pricing. For details about creating and provisioning Groups, see Cribl.Cloud Worker Groups.

View Data Sources

To view detailed access information about a Workspace's Data Sources, in the sidebar, select **Workspace** and then **Data Sources**.

The **Data Sources** page lists ports, protocols, and data ingestion inputs that are open and available to use, including pre-enabled Sources. Use the **Group** drop-down to filter these details per Cribl-managed Worker Group in Cribl.Cloud. For details, see Available Ports and TLS Configurations.

For each existing Source listed here, Cribl recommends using the preconfigured endpoint and port to send data into Cribl Stream.

Get ARN

To get the Worker role ARN for a Workspace, in the sidebar, select **Workspace**. In the **Stream Worker Group Details** section, select a Worker Group and copy the value of the **Trust** field.

You can use this **Worker ARN** as the principal in the trust policy for an AWS account's IAM role that Cribl Workers will assume. Attaching this trust relationship enables the `AssumeRole` action, providing cross-account access. For usage details, see the AWS Cross-Account Data Collection topic's **Account B Configuration** section.

This option applies only to your Cloud Workers. You cannot use this technique to enable access to customer-managed hybrid Workers.

Set Up ACL

To set up Access Control List (ACL) Rules, in the sidebar, select **Workspace** and then **Access Control List**.

ACL Rules (IPv4 CIDR ranges) let you restrict data sent to your data sources. The Rules you define here are global to all Cribl-managed Worker Groups in Cribl.Cloud in the current Workspace. You can set up a maximum of 6 ACL Rules.

The default `0.0.0.0/0` rule (modifiable) imposes no limits. End a rule with `/32` to specify a single IP address, or with `/24` to enable a whole CIDR block from `x.x.x.0` to `x.x.x.255`.

Select **Save** after adding, modifying, or removing rules. Each change takes up to 5 minutes to propagate. During this time, rule editing is temporarily disabled to prevent conflicts. A successful update proceeds silently - you will not see a confirmation message.

The **ACL** options apply only to Cribl-managed Workers. You cannot use them to set access rules on customer-managed hybrid Workers.

3.4.1. Configure Workspaces

Learn how to configure Workspaces in your Cribl.Cloud deployment.

Before You Begin

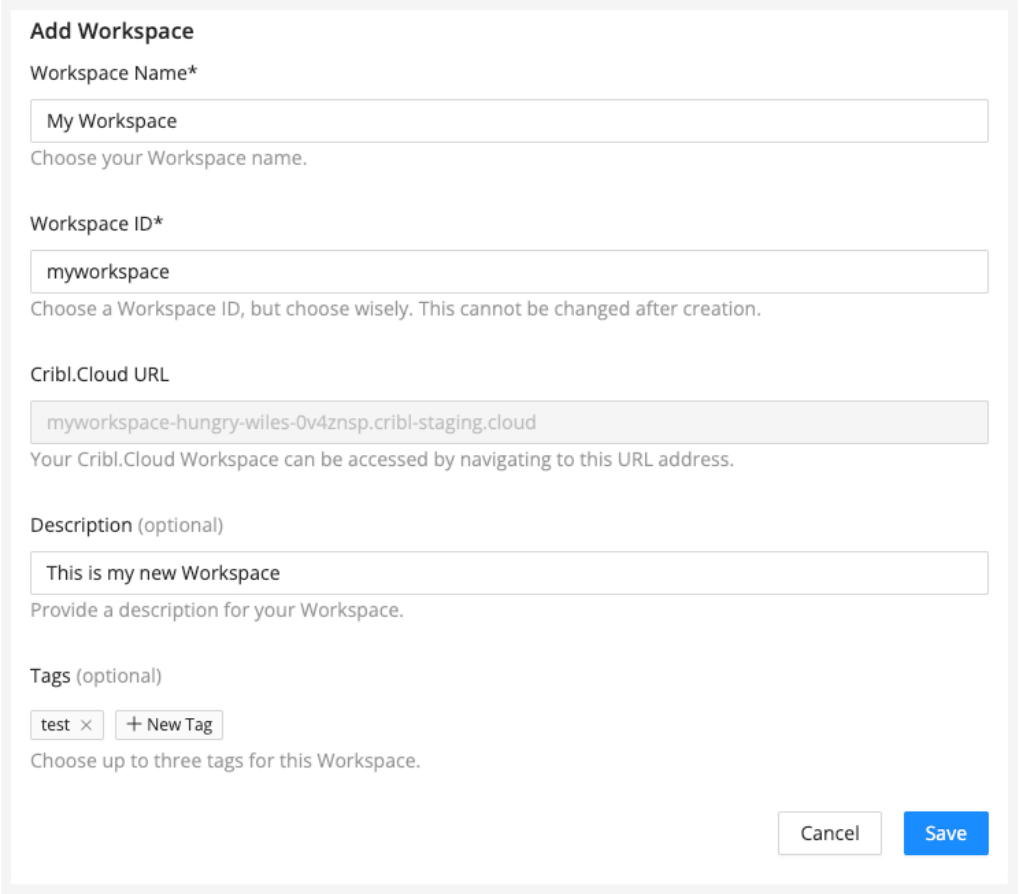
When you create a new Workspace in Cribl.Cloud, Cribl automatically provisions Cribl-managed Worker Nodes in the `default` Worker Group. **These Worker Nodes consume infrastructure credits**, but you can deprovision the `default` Worker Group from Worker Group settings. See Manage Cribl.Cloud Worker Groups for steps.

Add New Workspace

To create a new Workspace:

1. On the top bar, select your current Workspace name. This opens a modal with a list of all Workspaces you have access to.
2. Select **Add Workspace**.
3. Fill the form with the following information:

Field	Description	Notes
Workspace Name	Human-readable name for the Workspace.	
Workspace ID	Internal ID for the Workspace. This ID will be part of the Workspace's Cribl.Cloud URL.	**Cannot be changed** once the Workspace is created.
Description	Additional Workspace description.	Optional
Tags	Up to 3 tags you can use to filter Workspaces.	Optional



Creating a new Workspace

4. Confirm with **Save**.

The Workspace can take up to several minutes to spin up. While the process continues, you can navigate away and keep working with your Cribl.Cloud instance as usual.

Edit or Delete a Workspace

To edit a Workspace:

1. On the top bar, select your current Workspace name. This opens a modal with a list of all Workspaces you have access to.
2. Next to the Workspace you want to edit, select **Settings** in the **Actions** column to open the **Details** page for the Workspace.
3. On the **Details** page, you can change the name of the Workspace, description, and tags. However, you can't change the Workspace ID after it has been created.
4. Select **Save**.

For all Workspaces except `main`, you can select **Delete Workspace** on the **Details** page to delete it. You will

3.5. Projects

Projects (also called Stream Projects) diversify who can use Cribl Stream, and how:

- Projects create isolated spaces for teams and users to share and manage their data. Project editors get a streamlined visual UI.
- Cribl admins can filter users' access to specific data and Destinations - scoping the access to match what's relevant to those users' work.

This self-service approach to Cribl Stream data can benefit its immediate users, by offering them accelerated access to relevant data, with minimal configuration requirements. It can also benefit their peers.

Project editors get isolated data flows, and they can customize them to meet their needs, with no impact on other Cribl Stream users across their organization. Data can be forked into (for example) full-fidelity versus redacted views, to match different groups' authorization and needs.

You can also isolate data and permissions at a higher level by using Workspaces (available in Cribl.Cloud). Workspaces offer a separate VPC (virtual private cloud) that acts as a separate environment that contains not only Cribl Stream, but all Cribl products. See Workspaces for more details.

Breaking Change

Any Projects you created in v.4.1.x are deleted when you upgrade to v.4.2. You must re-create your Projects after you upgrade. You should also delete and re-create any Subscriptions that carry over during the upgrade.

In Cribl Stream 4.2 and newer, Projects/Subscriptions rely entirely on the fine-grained Members/Permissions access control first introduced in Stream 4.2. See Adding Users to Projects for details.

Use-Case Scenario

Imagine that a Cribl administrator wants to filter data from a shared set of firewall Sources, splitting the data between her organization's Security and Ops teams. A requirement is that each team's data should be isolated, with neither team having access to the other team's data. The Security team uses Splunk, while the Operations team uses Elasticsearch.

Projects enable this filtering and isolation. Each team gets secure access to its own sets of data, and each team's data transformations and configuration changes don't affect the other team's data or configs. Additionally, data capture and samples are isolated to each project.

Requirements

Enabling Projects requires an Enterprise plan or on-prem license. For details, see Pricing.

Projects' Subscriptions can ingest data only from Cribl Stream Sources configured as **Send to Routes**, not from QuickConnect Sources. (This is despite Projects' QuickConnect-like Project view, introduced below.)

Components

This feature relies on the following building blocks:

- Data Projects
- Subscriptions
- Members and Permissions

Data Projects

The **Data Projects** tab is where Cribl admins can add and configure Projects, share access to those Projects, and associate Subscriptions with specific Destinations.

Admins see an expanded version of the same drag-and-drop visual UI that Project editors can use to connect configured Subscription with configured Destinations.

Each project maintains its own independent processing Pipelines, data capture and sample data.

Subscriptions

Each Subscription filters a subset of a Worker Group's incoming data to expose to Project editors, along with processing options for that data.

To ensure Project metrics appear in Subscription charts, remove `project` from the **Disable field metrics** list in both Worker and Leader group settings. If `project` is present in this list, metrics for Projects will not be collected or displayed.

Members and Permissions

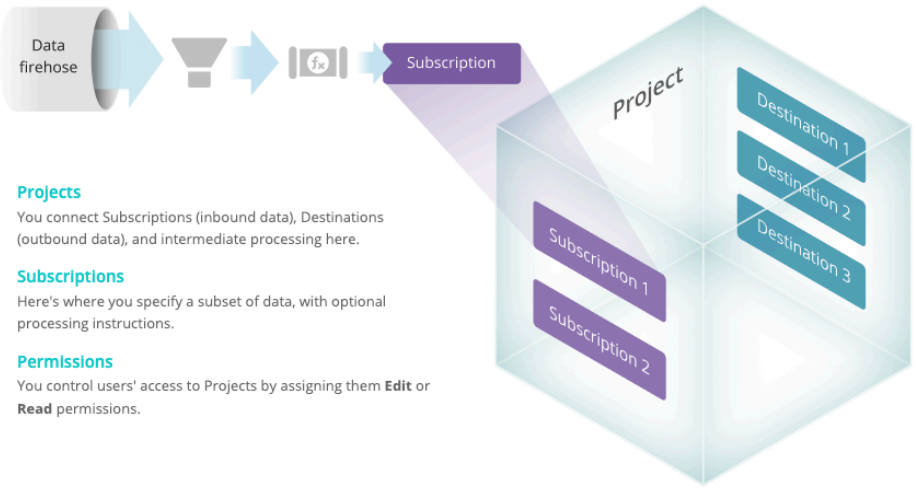
Projects work with the Permissions model for access control. Cribl Stream admins (users who hold the Stream- or Worker Group-level `Admin` or `Editor` Permission, or the legacy `stream_admin` Role) can create and configure Projects and Subscriptions. They can access resources outside individual Projects to build the Projects.

As a Stream Admin, you also control users' access to Projects and Subscriptions. For details, see Adding Users to Projects.

How the Components Fit Together

The components' names and relationship are easiest to remember in terms of the publish/subscribe model long used by Cribl Stream integrations like Kafka or Google Cloud Pub/Sub:

- A **Project** is a microcosm of a whole Cribl Stream deployment - gathering together Subscriptions' incoming data, routing/processing, Destinations, and authorized users.
- A **Subscription** specifies a subset of available data to retrieve, and how to import the data.



How Subscriptions and Projects filter incoming data

Get Started with Projects

You interact with Projects in either of two ways, depending on your Cribl Stream Permissions.

As a Project Editor

When you log into Cribl Stream and navigate to a Group, you'll find your Projects and Subscriptions already set up for you in a convenient visual UI.

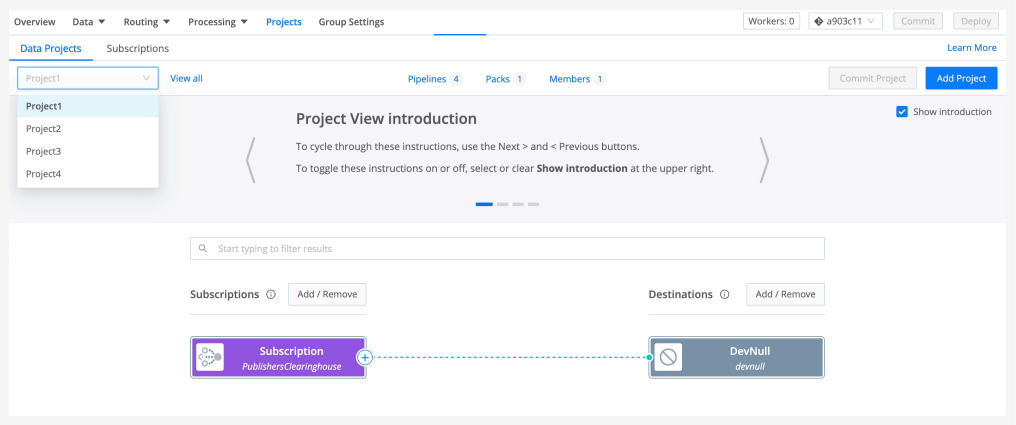
You can adjust your Projects' data flow by connecting Subscriptions to Destinations. You can insert and configure Pipelines, and can insert Packs, that are part of the Project. You can also commit your configuration changes to Git. This all works the same way as in QuickConnect.

As a Cribl Admin

To access Projects configuration options:

1. Navigate to a Worker Group. (Projects and their components are configured per Group, providing a first layer of isolation).
2. Select **Projects** in the submenu to see two menu options: **Data Projects** and **Subscriptions**.

3. See the following pages for details on configuring each of these components for Project editors.



Accessing the Project view

3.5.1. Configure Projects

Projects define the data that the Project's users can consume, where that data can be sent, and who has access to the Project. A Project associates specific Subscriptions with specific Destinations. For fine-grained access control, you can share each Project with specific Members and Teams.

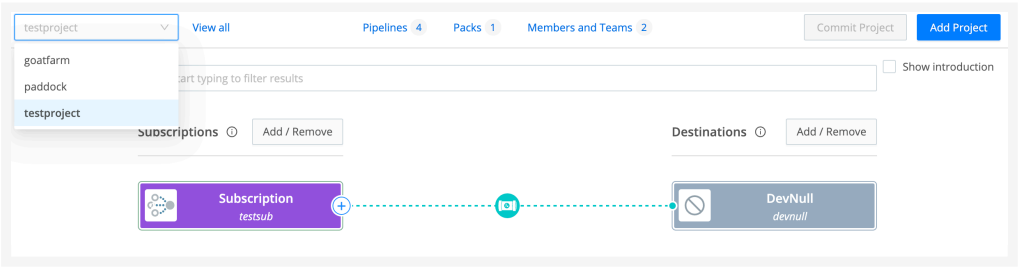
By connecting incoming data sources (Subscriptions) to specific Destinations, Projects function like a Route's Destination selection. We expand on this similarity in Subscriptions and Routes.

Access Projects

From your Worker Group's **Manage** submenu, select **Projects** to open the default **Data Projects** page.

This is the "Project view," an expanded version of the UI in which Project editors will manage connections among the resources you define here. From left to right, the options in this administrator's view are:

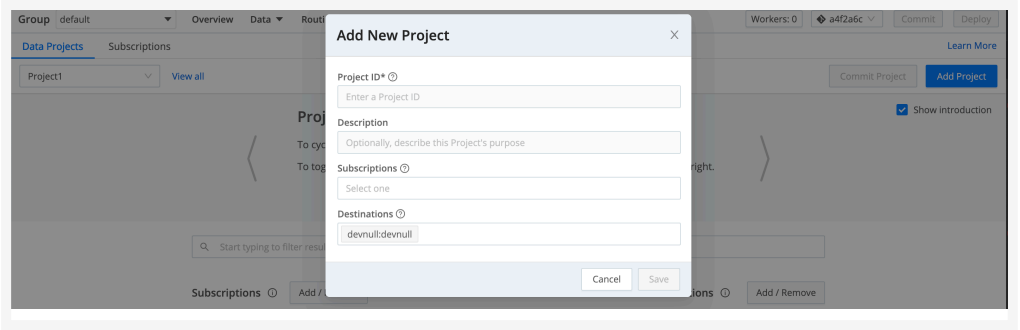
- Projects drop-down list: To bring a different Project into the Project view, select it here.
- **View all** link: Select to open the **All Projects** modal, where you can configure and share existing Projects.
- **Pipelines** link: Select to open a modal where you can manage Pipelines that are scoped to this Project.
- **Packs** link: Works the same way as the **Pipelines** link. The modal will show only Packs that are part of this Project. For Project editors, Packs will be read-only resources - Project editors can insert or remove Packs, but not modify them.
- **Members and Teams** link: Select to open a drawer where you can grant Members and/or Teams access to the Project.
- **Commit Project** button: Commit changes from this Project to Git.
- **Add Project** button: Creates a new Project. Covered just below.



Accessing the Project view

Configure a New Project

1. From the Project view shown above, select **Add Project** to open the **Add New Project** modal shown below.



Configuring a new Project

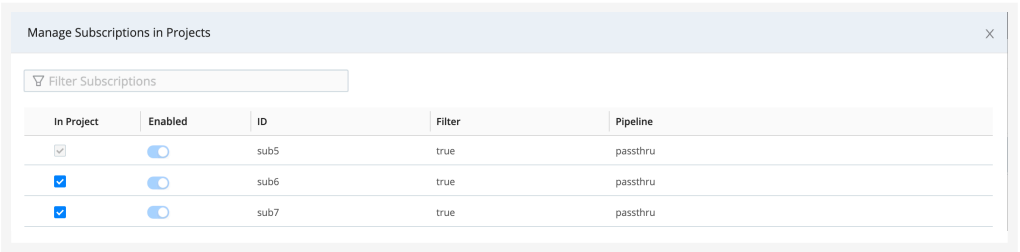
2. Assign this Project a unique **Project ID**. (This can include letters, numbers, `_`, and `-`. IDs are case-sensitive, so entering the same characters with different capitalization will create a separate ID.)
3. Optionally, add a Project **Description**.
4. Select one or more configured Subscriptions. (You can access only Subscriptions that are already configured in this Project. But you can reopen this modal later to add newly configured Subscriptions. Also, the Subscriptions UI enables you to add Subscriptions to Projects.)
5. Select one or more configured Destinations that will have access to this Project's data. (You can access only Destinations that are already configured in this Project - but you can configure and add more Destinations in the next section.)
6. Select **Save** to finish configuring your Project.
7. Commit and deploy your changes.

To view **Live Data** on your configured **Destination**, you must **Commit** and **Deploy** your changes.

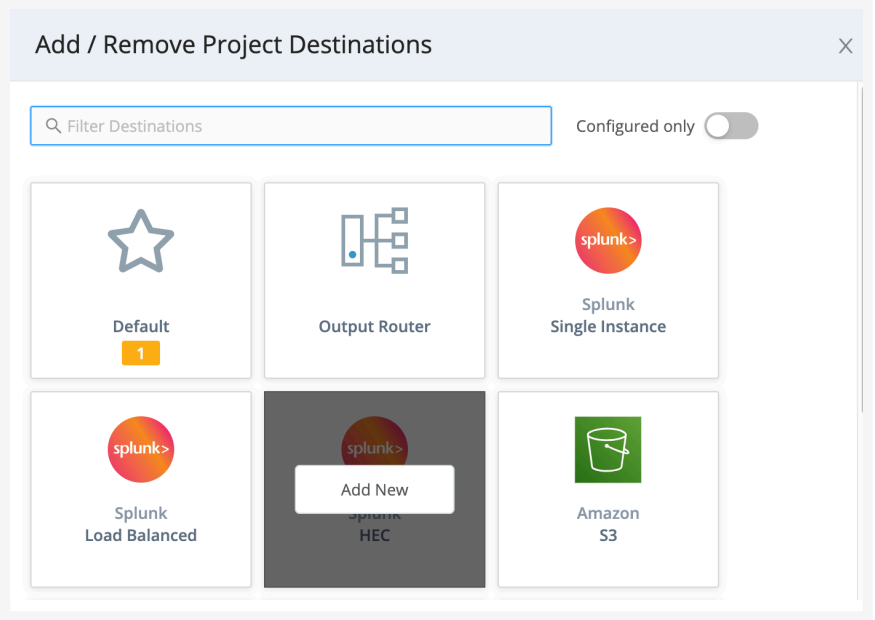
Test a Project (Manage Data Flow)

After you return to the Project view, make sure your desired Project has focus in the upper-left drop-down list. You can now use the Project view's visual UI to expand the Project and enable its data flow. The options here include:

- Connect the Project's already configured Subscriptions and Destinations, optionally adding a Pipeline or Pack to each connection.
- Select **Subscriptions** > **Add / Remove** to open a drawer where you can expose any Subscription you've already configured in the Project, remove Subscriptions from the Project view, or add new Subscriptions.



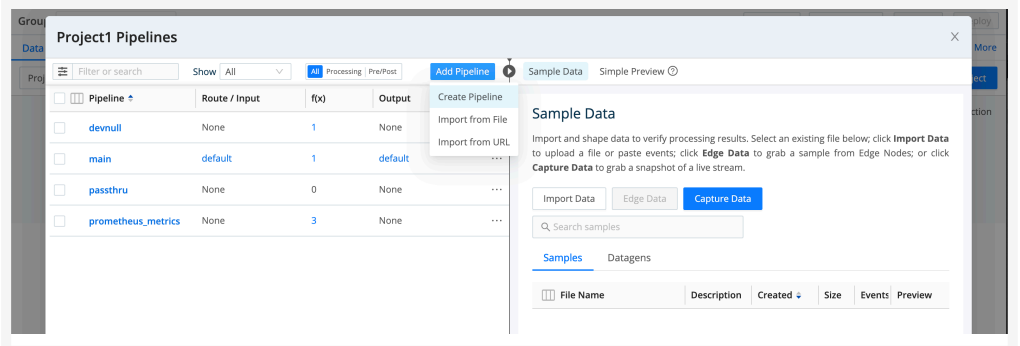
- Select **Destinations** > **Add / Remove** to open a drawer where you can expose and configure any Destination type available in your Cribl Stream deployment. (Any Destination that you add here will automatically be added to the Project's configuration modal when you next reopen it.)



As you configure and test your Project, beware of high ingestion rates that could overwhelm downstream receivers. To prevent dropped events, enable Persistent Queues on Destinations that support them.

Enable Pipelines and Packs

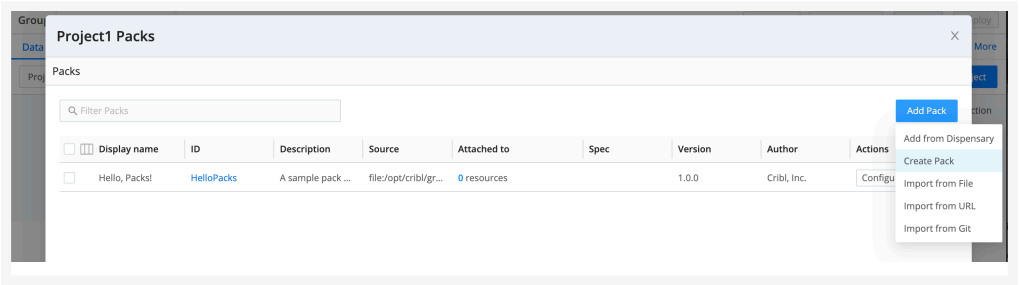
Each Project that you configure opens with a basic set of Pipelines and one default Pack. To manage these, select the upper **Pipelines** or **Packs** link, respectively. Each opens a modal similar to Cribl Stream's standard Pipelines or Packs UI, with access to standard Sample Data and Preview options on the right.



Managing Project Pipelines

To offer Project editors additional processing resources to insert on Subscription/Destination connections, select the modal's **Add Pipeline** or **Add Pack** button. You can then enable new Pipelines and Packs in the Project, with these constraints:

- You can create a new Pipeline or Pack here. It will be available only in the current Project.
- You can import an existing Pipeline or Pack. Any modifications that you make here will similarly be scoped to the current Project.
- To import an existing Pipeline from your Cribl Stream deployment, you must export it as JSON, then select the **Import from Pipeline** drop-down option shown above. (You can also **Import from URL**.)
- You can import Packs from an exported JSON file, a URL, or the Dispensary or Git options shown on the drop-down below.

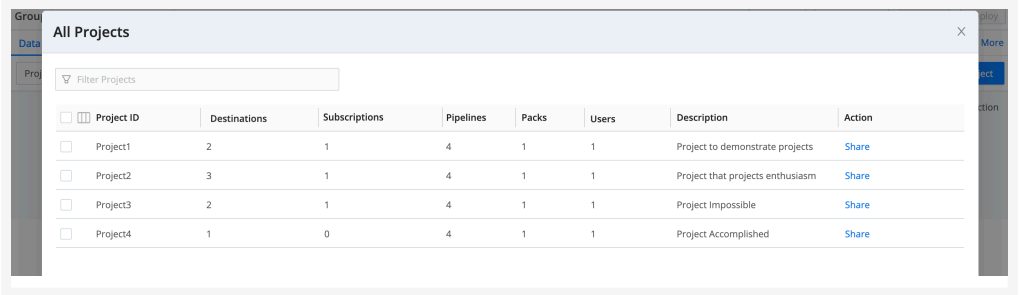


Adding Project Packs

Edit a Project

To update an existing Project, select the Project view's upper-left **View all** link. From the resulting **All Projects** modal, select the Project's row to reopen its configuration in a modal similar to the **Add New Project** modal covered above.

You can also select any row's **Share** link to open a drawer in which to manage users' access to the Project.



View all > All Projects modal

Access Project Metrics

You can access the following metrics on your Projects:

- Events and bytes in/out of Subscriptions.

- Bytes in/out of Pipelines, Packs, and Destinations.
- Events in/out/dropped through Pipelines, Packs, and Destinations;

To tap these metrics, enable Cribl Stream's internal CriblMetrics Source. This Source's defaults are different for Cribl.Cloud versus on-prem deployments.

Cribl.Cloud Leaders and Groups

For Cribl.Cloud Leaders, and Worker Groups of Cribl-managed Workers in Cribl.Cloud, Projects' metrics are always available to the CriblMetrics Source. This availability is not configurable.

On-Prem Leaders and Groups

For on-prem Leaders, and customer-managed (hybrid or on-prem) Worker Groups, Projects' metrics are disabled by default. (As with other on-prem exclusions, we provide this default to optimize performance.) However, you can easily make these metrics available to the CriblMetrics Source.

See Controlling Metrics for general instructions on modifying defaults at the Leader, Worker Group, or single-instance level. From the **Disable field metrics** list, remove the `project` metric. Then save, commit, and deploy your changes.

3.5.2. Configure Subscriptions

You use a Subscription's **Filter** expression to specify a subset of a Worker Group's overall incoming data stream to forward to Destinations. Optionally, you can also specify a Pre-processing Pipeline to condition that data before it leaves the Subscription.

You gather Subscriptions into Projects, where you match them up with Destinations and share them with authorized users.

A Project can contain multiple Subscriptions, and each Subscription can be associated with multiple Projects. Each Project's copy of the Subscription will share the same configuration, but will have independent data flow.

Subscriptions and Routes

From the UI similarity, you've probably noticed that Subscriptions look and function much like Routes. There are both similarities and differences - which we'll consider in reverse order.

Subscriptions/Routes Differences

Compared to Routes, Subscriptions present these exceptions:

- Subscriptions lack a Route's **Output** or **Destination Expression** field. (Instead, you associate Subscriptions with Projects, where Project editors can process the data and send it to Destinations.)
- Subscriptions have no **Final** flag. So a Subscription is similar to a Route with the **Final** toggle switched off.

Subscriptions do not relay backpressure upstream to data senders. (This is by design, to prevent any Subscription from affecting data flow through parallel Subscriptions, Projects, or Routes/Pipelines.) If any backpressure occurs, a Subscription will drop data.

Subscriptions/Routes Similarities

A further link to Cribl Stream's Data Routes interface is that Subscriptions can ingest data **only** from Sources configured (at the Cribl Stream product level) as **Send to Routes**. (Subscriptions and Projects cannot ingest data from QuickConnect Sources, despite the similarity of Projects' visual UI (the Project view).

This also means that Subscriptions tap exactly the same overall flow of inbound data that Data Routes do, in parallel. So there are cases where, after you've configured Subscriptions to ingest certain data, you might want to configure a Pipeline at the top of the Routing table to drop the same data. This will prevent Cribl Stream from processing the same events twice.

Usage Tips

Compared to sending the same data flow through Routes, adding Subscriptions can have a performance impact, because each Subscription creates a copy of the data that Cribl Stream needs to process. Therefore, your resource requirements could change with the number of Subscriptions you add.

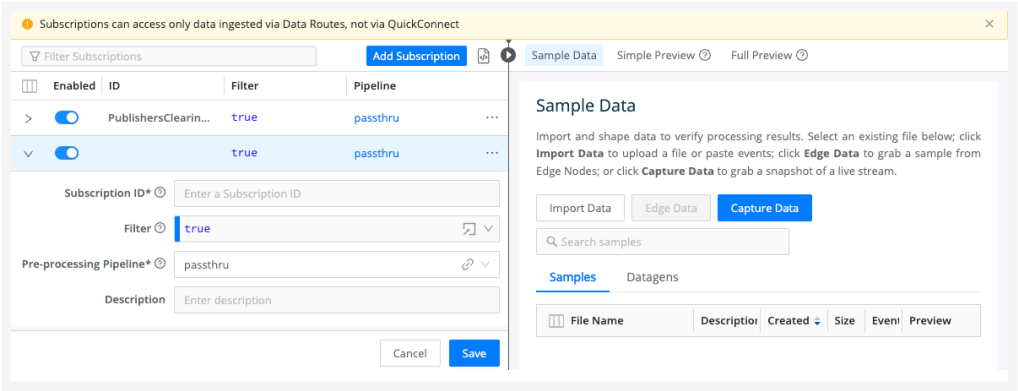
Please work with your Cribl Solutions Engineer to select an architecture that best balances your needs for performance versus segmented data/configuration access.

Cribl recommends that you configure a separate Subscription - and corresponding Pipeline(s) - for each data type you ingest. This enables you to filter by data type in the Subscription's definition, and again in individual Pipeline Functions.

Access Subscriptions

From your Worker Group's submenu, select **Projects**, and then select **Subscriptions**.

Any Subscriptions you've already configured will appear here. Expand the accordions to access each Subscription's configuration.



Configuring a new Subscription

Configure a New Subscription

1. Select **Add Subscription** to open the configuration options described below.
2. Assign this Subscription a unique **Subscription ID**. (This can include letters, numbers, `_`, and `-`. IDs are case-sensitive, so entering the same characters with different capitalization will create a separate ID.)
3. Modify the **Filter** by replacing the default `true` with a JavaScript expression that selects a subset of events to filter through this Subscription. (Your expression can specify an upstream sender, a data type, the presence of particular fields, and more. For examples, see Build Custom Logic to Route and Process Your Data: Filter Expressions.)
4. Modify (as needed) the Source's default `passthru` Pre-processing Pipeline.
5. Optionally, add a **Description** of the Subscription's purpose.

6. Optionally, assign the Subscription to one or more Projects. (You can also import Subscriptions from individual Projects' configurations, and can control Subscriptions' visibility in the Project view.)
7. Select **Save** to save the new Subscription to the Subscriptions table.
8. Commit and deploy your changes.

A Subscription must specify a Pre-processing Pipeline, even if this is the default `passthru`, which performs no processing.

Add and Preview More Subscriptions

On the **Subscriptions** page, you can select **Add Subscription** to define multiple Subscriptions. These stack on the left, like Routes in Cribl Stream's Routing table.

Note these further similarities to the Routing table:

- Each Subscription's blue title bar has a toggle that you can use to disable the Subscription from gathering data, while preserving its configuration.
- On the right, you can capture, import, and save sample data, and. You can use these samples to preview how your Subscriptions transform incoming data.

Edit a Subscription

To update an existing Subscription, just select its accordion in the Subscriptions table. This will reopen the same configuration options outlined above.

3.5.3. Add Users To Projects

As a Stream admin, you control Members' access to individual Projects by assigning Permissions to those Members, as outlined below.

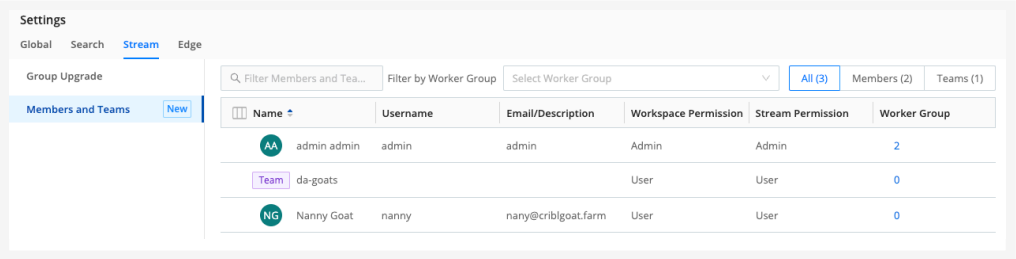
Breaking Change

In Cribl Stream 4.2 and newer, Projects and Subscriptions rely entirely on the fine-grained Members/Permissions access control first introduced in Cribl Stream 4.2. The v.4.1.x `project_user` Role and `ProjectSourceSubscribe` Policy, which provided access to all Projects, are now retired. To migrate Projects configured in v.4.1.x, you'll find those projects' users visible in Cribl Stream as Members. Assign them appropriate new Permissions, as outlined here.

Share a Project (Example)

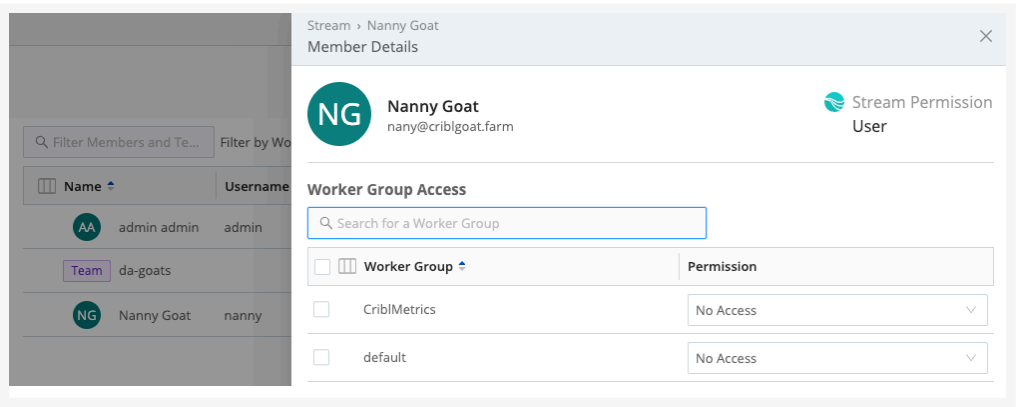
To share a Project with an eligible Cribl Stream Member or Team:

1. Select **Settings** > **Stream** > **Members and Teams**.
2. Make sure each applicable Member or Team has the `User` **Workspace Permission**, and the `User` **Stream Permission**. (Select an existing Member's row to adjust their Permissions.)



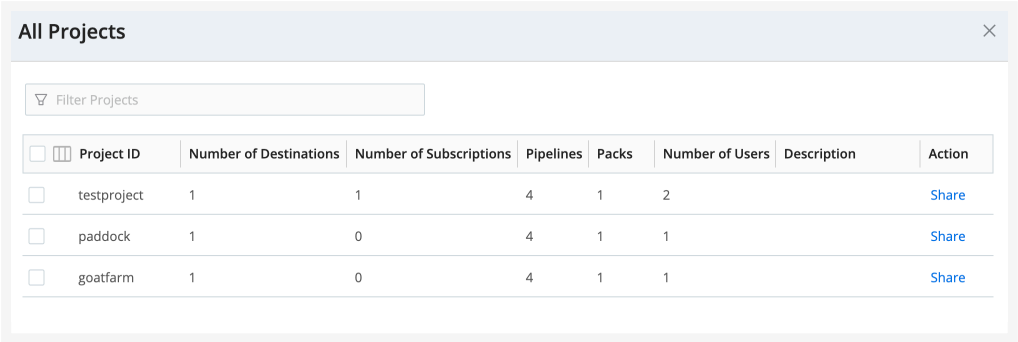
Stream Settings > Members and Teams

3. Select each applicable Member and Team's row to open their details drawer.



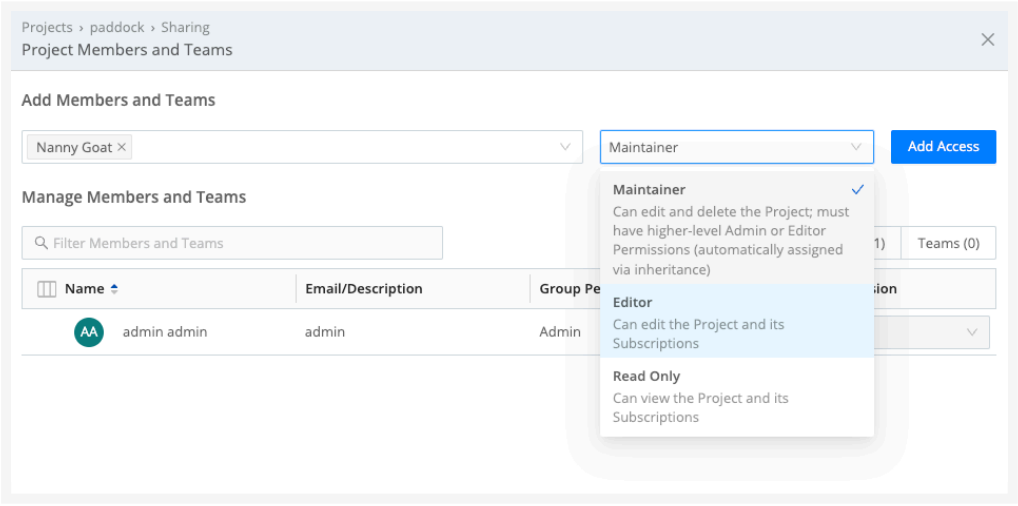
Stream Members > User Details

4. Grant the Member or Team a `User` Permission on the applicable Project's parent Worker Group. (Higher Permissions offered on this drop-down are not applicable to Project editors.)
5. In the sidebar, select **Worker Groups** and select the same Worker Group.
6. Select **Projects** > **Data Projects** > **View all**.



View all > All Projects modal

7. Select the Project's **Share** link to open the **Project Members and Teams** drawer. Here, you can see all the Group's Members and Teams with whom this Project can be (or has been) shared.
   You can also open an individual Project's **Sharing** drawer directly from the Project view. Give this Project focus in the Project view, then select the **Members and Teams** link at the top.
8. Grant each applicable Member or Team an `Editor` Permission on the Project. (The alternative `Read Only` Permission is intended for support users. It does not enable configuring connections or Pipelines.)



Project Sharing drawer - assigning `Editor` Permission

9. Select **Save** to confirm your changes, then select outside the drawer to close it.
10. Commit your changes, and deploy the Group if prompted.

These Members and Teams who now have Project `Editor` Permissions - and no higher Stream Permissions - view data filtered through the Projects UI. They also have read access to Cribl Stream Data Monitoring and Notifications.

You can reopen the same drawer to modify Members' Permission levels. To remove a Member from the Project, assign them `No Access`.

3.6. Notifications

With Notifications, you can set up alerts to notify Cribl Stream system administrators about events that require their immediate attention.

For example, you could set up a Notification to alert you when a Source stops sending data to a given Worker Group, or when the volume of data sent to a Destination from the Worker Group exceeds a threshold.

Notifications require certain plan or license tiers. Without an appropriate license, the configuration options described here will be hidden or disabled in Cribl Stream. For more information on license types, and a feature comparison, see Pricing.

How to Get Notifications

You can get notifications through:

- **Internal logs:** Cribl Stream stores these application-wide logs in the `notifications.log` file on the Leader Node, which also handles sending all Notifications. See Internal Logs for more information about logs.
- **Events in the UI:** Displays events in the Monitoring page. See Monitoring for more information about UI notifications.
- **Custom targets:** You can configure Notification targets to send alerts through email, pagers, Slack messages, and more. See Notification Targets for information about available targets and setup instructions.

Types of Notifications

Type	Available Notifications	For more info
Sources	• High Data Volume • Low Data Volume • No Data • Source Persistent Queue Usage	Add a Notification for a Source

Type	Available Notifications	For more info
Destinations	• Destination Backpressure Activated • Destination Persistent Queue Usage • Unhealthy Destinations	Add a Notification for a Destination
Licenses	• Pending expiration of license	Add a Notification about License Expiration

You can also configure Cribl Stream to relay Notifications generated by other Cribl products. For an example, see the Cribl Search topic Notifications via Cribl Stream.

Notification Permissions

Notifications work with Cribl Stream role-based access control (RBAC). For users with non-administrative permissions, their assigned Roles and Policies determine the Worker Groups on which they can view Notification messages, configure Notifications, and configure targets.

Add a Notification for a Source

To create alerts for a Source or Collector:

1. Navigate to your Worker Group. Then select **Data** > **Sources**.
2. Select the Source type to display the table of its configured Sources. For example, select **Syslog** to set up notifications for a Syslog Source.
3. Select the **Notifications** button in the row for the Source you want to configure.
4. In the **Notification** modal, configure the following settings under **General**:
    - **ID**: Provide a unique ID for the Notification in this section.
    - **Enabled**: Toggle on (default) if you want Cribl Stream to enable the Notification. Toggle off to turn off the Notification.
5. In the **Configuration** section in the **When...** field, select the condition that will trigger the Notification and then configure the settings for that trigger condition. See Source Settings for more information about the trigger condition settings. The available Source Notification trigger conditions:
    - **High Data Volume**: Incoming data is above the data volume threshold for your configured time window.
    - **Low Data Volume**: Incoming data is below the data volume threshold for your configured time window.
    - **No Data Received**: The Source or Collector ingests zero data over your configured time window.

- **Persistent Queue Usage**: The Source persistent queue accumulates files past the threshold percentage of storage capacity. See Optimize Source Persistent Queues for more information about working with Source persistent queues to manage backpressure.

6. In the **Configuration** section, set the **Only notify on start and resolution** toggle. When toggled on, Cribl Stream will send a Notification when the triggering condition begins and then a second Notification to report when the condition has ended.
7. Configure the **Send Notification to** setting to set the Notification target(s). If you don't have a Notification target, the **Messages** icon will indicate you have a new Notification. See Notification Targets for more information about available targets and adding new targets.
   - Select **Add Target** to add an existing target.
   - Select **Create Target** to add a new target.
8. Save the Notification.

Your Notification will now appear on the **Events** tab within the configuration window for this Source. To view all Notification events and targets, select **Notifications** in the sidebar.

Consider testing the condition to determine if you configured the Notification properly by deliberately triggering the Notification trigger.

Source Notification Settings

The function of the **Data volume** and **Time window** settings change depending on which Notification trigger you selected:

Notification Trigger	Settings
High Data Volume	Triggers an alert if the total data ingested within the configured **Time window** exceeds the specified **Data volume** threshold.
Low Data Volume	Triggers an alert if the total data ingested within the configured **Time window** falls below the specified **Data volume** threshold.
No Data Received	Triggers an alert if a Source or Collector fails to ingest any data for the duration of the configured **Time window**.
Persistent Queue Usage	Triggers an alert when persistent queue disk usage exceeds the specified **Usage threshold**. The **Time window** determines how often this condition is checked.

Source name: This field locks to the Source for this Notification and cannot be edited.

Time window: This field's value sets the threshold period before the Notification will trigger. The default `60s` will generate a Notification when the Source has reported the trigger condition over the past 60 seconds. To enter alternative numeric values, append units of `s` for seconds, `m` for minutes, `h` for hours, and so forth.

See Source Notifications Best Practices for information about avoiding false notifications when configuring the **Time window** setting for Collectors that run on a schedule.

Data volume: Enter the threshold above which a Notification will trigger. Accepts numerals with units like `KB`, `MB`, and so forth. For example: `4GB`. If you want the unit to be bytes, enter the numeral only, without a unit designator.

Usage threshold: The percentage of full or used storage disk capacity. Cribl Stream generates a `Persistent Queue usage has surpassed <threshold>%` Notification when the Source persistent queue storage disk capacity is past the specified **Usage threshold** percentage. This field appears only when you configure a Notification for **Persistent Queue Usage**.

Metadata: User-defined fields included in the Notification payload. All Notification types can contain metadata. To add custom metadata fields, select **Add field** and add the fields as key-value pairs:

- **Name**: Enter a name for the custom field.
- **Value**: Enter a JavaScript expression that defines this field's value, enclosed in quotes or backticks. This can evaluate to a constant if needed.

Source Notification Best Practices

To ensure your Source Notifications are reliable and easy to manage, follow these best practices:

- For Collectors that run on a schedule, always set the **No Data Received** Notification time window to a duration that is longer than the time between scheduled runs. Because there is no data flow between runs, a shorter time window will trigger false alarms.
- When cloning a Source, remember to manually recreate any Notifications you need on the new Source. Cloning a Source does **not** copy its associated Notifications.

Add a Notification for a Destination

To create alerts for a Destination:

1. Navigate to your Worker Group. Then select **Data** > **Destinations**.
2. Select the Destination type to display the table of its configured Sources. For example, select **Syslog** to set up notifications for a Syslog Destination.
3. Select the **Notifications** button in the row for the Destination you want to configure.
4. In the **Notification** modal, configure the following settings under **General**:
    - **ID**: Provide a unique ID for the Notification in this section.

- **Enabled**: Toggle on (default) if you want Cribl Stream to enable the Notification. Toggle off to turn off the Notification.

5. In the **Configuration** section in the **When...** field, select the condition that will trigger the Notification and then configure the settings for that trigger condition. See Destination Settings for more information about the trigger condition settings. The available Destination Notification trigger conditions:
   - **Destination Backpressure Activated**: The Destination has engaged its backpressure behavior, such as when a downstream Destination receiver is unreachable or slow to respond. The triggering conditions:
     - The Destination's **Backpressure behavior** is set to `Block` or `Drop`, and backpressure causes outgoing events to block or drop.
     - The Destination's **Backpressure behavior** is set to Persistent Queue and its **Queue-full behavior** is set to either `Block` or `Drop new data`. When the queue is full, it causes the Destination to block or drop outgoing events.
   - **Persistent Queue Usage**: The Destination persistent queue accumulates files past the threshold percentage of storage capacity. See Optimize Destination Persistent Queues for more information about working with Destination persistent queues to manage backpressure.
   - **Unhealthy Destination**: The health of the Destination has been in red status (as indicated on the Monitoring page) over the configured time window.
6. In the **Configuration** section, set the **Only notify on start and resolution** toggle. When toggled on, Cribl Stream will send a Notification when the triggering condition begins and then a second Notification to report when the condition has ended.
7. Configure the **Send Notification to** setting to set the Notification target(s). If you don't have a Notification target, the **Messages** icon will indicate you have a new Notification. See Notification Targets for more information about available targets and adding new targets.
   - Select **Add Target** to add an existing target.
   - Select **Create Target** to add a new target.
8. Save the Notification.

Your Notification will now appear on the **Events** tab within the configuration window for this Destination. To view all Notification events and targets, select **Notifications** in the sidebar.

Destination Notification Settings

The function of the **Time Window** settings change depending on which Notification trigger you selected:

Notification Trigger	Settings
Destination Backpressure Activated	The threshold for the Notification to trigger is: Cribl Stream detected a blocked or dropped state during ≥ 5% of the trailing **Time window**

Notification Trigger	Settings
Persistent Queue Usage	The **Time Window** determines how frequently the Notifications repeat.
Unhealthy Destination	The **Time Window** determines how frequently the Notifications repeat.

Destination name: This field locks to the Destination for this Notification.

Time window: This field's value sets the threshold period before the Notification will trigger. The default `60s` will generate a Notification when the Destination has reported the trigger condition over the past 60 seconds. To enter alternative numeric values, append units of `s` for seconds, `m` for minutes, `h` for hours, and so forth.

Usage threshold: The percentage of full or used storage disk capacity. Cribl Stream generates a `Persistent Queue usage has surpassed <threshold>%` Notification when the Destination persistent queue storage disk capacity is past the specified **Usage threshold** percentage. This field appears only when you configure a Notification for **Persistent Queue Usage**.

Metadata: User-defined fields included in the Notification payload. All Notification types can contain metadata. To add custom metadata fields, select **Add field** and add the fields as key-value pairs:

- **Name**: Enter a name for the custom field.
- **Value**: Enter a JavaScript expression that defines this field's value, enclosed in quotes or backticks. This can evaluate to a constant if needed.

Destination Notification Best Practices

When cloning a Destination, remember to manually recreate any Notifications you need on the new Destination. Cloning a Destination does **not** copy its associated Notifications.

Add a Notification About License Expiration

To create a license-expiration alert:

1. Navigate to **Settings** > **Global** > **Licensing**. Then select **Add Expiration Notification**.
2. In the **Notification** modal, configure the following settings under **General**:
    - **ID**: Provide a unique ID for the Notification in this section.
    - **Enabled**: Toggle on (default) if you want Cribl Stream to enable the Notification. Toggle off to turn off the Notification.
3. Optional: Add or create a Notification target. See Notification Targets for more information. If you don't have a Notification target, the **Messages** icon will indicate you have a new Notification.
4. Save the Notification.

To view all Notification events and targets, select **Notifications** in the sidebar. You can also reach this page by selecting **Settings** > **Global** > **Licensing**.

Consider testing the condition to determine if you configured the Notification properly by deliberately triggering the Notification trigger.

Edit Notifications

To edit a Notification:

1. Select **Notifications** in the sidebar. You can also reach this page by selecting an existing Notification in a

3.6.1. Email Notifications

If you're a Cribl Stream admin, email notifications make it easy to receive alerts about any operational issues that require your attention, such as a particular Source or Destination condition or a pending license expiration.

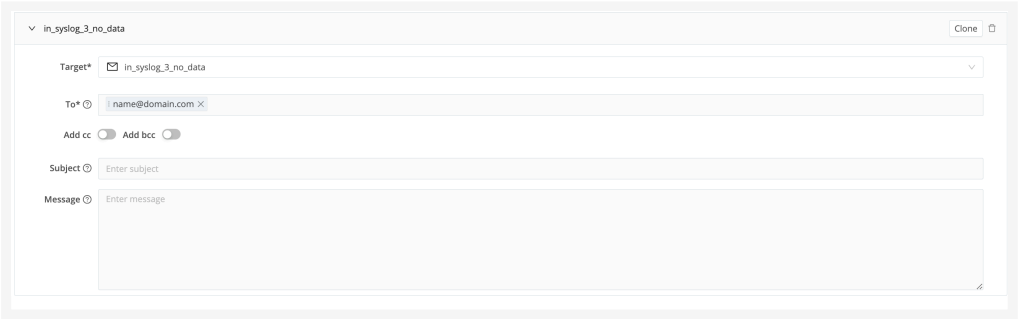
An email Notification requires two things – a configured Notification and an email Notification target. Cribl.Cloud Organizations using certain plan/license tiers also have access to a preconfigured email Notification target.

Email Notifications do not have an unsubscribe option. Recipients who do not want to receive particular email Notifications should contact their Cribl Stream admins.

Email Notifications are sent on a no-reply basis.

Configure Email Notifications

When you create a Notification for an email target, specify the recipients of the message, the subject line, and the contents of the message.



Email Notification with target selected

Target: The ID of the email target to which you want to send the Notification.

Select **Add target** to add an existing target.

Select **Create target** to create a new target. For information on configuring email Notification targets, see Email Notification Targets.

If the Notification already has a designated target, you can change the selection by selecting the drop-down.

When an email Notification target is selected, the following additional fields appear:

To: The email address of the recipient.

Add cc: When enabled, reveals a field where you can enter the addresses of additional recipients.

Add bcc: When enabled, reveals a field where you can enter the addresses of additional recipients that do not appear in the Notification email.

Cribl Stream does not limit the total number of recipients for a Notification, but your email service might set a limit.

Subject: The subject line of the email Notification. You can use variables in the subject line.

Message: The content of an email Notification. You can use variables in the body of the email message.

Email Notification Variables

Email variables are placeholders in the email template that get replaced with actual values when the email is sent. These variables can be:

- General-use variables, like `condition`, `worker_group`, or `timestamp`
- Special-purpose variables

You can use a variety of general-use and special-purpose variables in the subject or message body of an email Notification. Insert a variable name between two braces preceded by a `$`. For example: `${cribl_notification}`.

General-Use Email Variables

Variable	Description
`workspace`	Workspace name (Cribl.Cloud only).
`organization`	Organization ID (Cribl.Cloud only).
`timestamp`	Timestamp when the email is sent. For example: `2019-08-04 18:22:24 UTC`.
`cribl_notification`	User-defined notification ID.

Special-Purpose Email Variables

Variable	Usage	Notification type	Example
`_raw`	The `_raw` field of the Event triggering the Notification.	• License expiration • Cribl Search	`Your Cribl license expires in 3 days on 2019-08-04 18:22:24 UTC. Please update your license to continue processing data after 2019-08-04 18:22:24 UTC.`

Variable	Usage	Notification type	Example
`__worker_group`	Name of the Worker Group triggering the Notification.	• Destination Backpressure • High data volume • Low data volume • No data • Persistent Queue Usage • Unhealthy Destination	`default`
`timewindow`	Metric evaluation period used for Notification.	• Destination Backpressure • High data volume • Low data volume • No data • Persistent Queue Usage • Unhealthy Destination	`60s`
`status`	Status of the Notification Status of backpressure engagement.	• Destination Backpressure • High data volume • Low data volume • No data • Persistent Queue Usage • Unhealthy Destination	`OK`, `ALARM` `DISENGAGED`, `BLOCKING` or `DROPPING`
`input`	Type and name of a Source or Collector.	• High data volume • Low data volume • No data	`syslog:in_syslog_1`
`output`	Type and name of a Destination.	• Destination Backpressure • Persistent Queue Usage • Unhealthy Destination	`webhook:out_webhook_1`

Variable	Usage	Notification type	Example
`bytes`	Quantity of data triggering the specified Notification.	• High data volume • Low data volume • No data	`0`
`dataVolume`	Data volume which triggered the Notification. Accepts numerals with units like KB, MB, and so on.	• High data volume • Low data volume	`1G`
`health`	Health metric of the specified Source or Destination.	• Unhealthy Destination	`0`
`queue_usage`	Percentage of capacity set in the **Usage threshold** field.	• Persistent Queue Usage	`90`
`usage`	The actual usage, in percent.	• Persistent Queue Usage	`50`
`usageThreshold`	The usage threshold, in percent.	• Persistent Queue Usage	`50`

Troubleshoot Email Notifications

The section details the troubleshooting steps you can take if an email Notification fails to reach its intended recipient.

Test the Email Notification Target

An email Notification can fail if the target is misconfigured. You can test your email Notification target by following this procedure:

1. Open the target (in **Notifications** > **Targets**).
2. Select **Test Target**.
3. Add one or more email addresses to the **Test Target** modal and select **Send Test Email**.

4. Check the designated inbox to verify receipt of the test message. If it does not arrive in the designated inbox, review the target configuration.

Check Notification Service Logs

A failed email Notification leaves a log entry. You can examine logs by following this procedure:

1. Select **Monitoring** > **Logs**.
2. In the drop-down on the left, select **Leader** > **Notifications Service**.
3. Examine any logs that have errors.

Check Notifications

Cribl Stream stores Notifications. You can check them by following this procedure:

1. Select **Monitoring** > **Notifications**.
2. In the search field, search for **cribl_notification** fields that correspond to the Notification ID of the failed

3.6.2. Notification Targets

A Notification target allows you to receive alerts using your preferred channel or alert delivery method. Every Notification requires one or more targets.

Available target types include:

- Webhook
- PagerDuty integration
- Slack
- AWS SNS
- Email
- System Messages (default)

When you create a new Notification, its default target type will be **System Messages**. When an event occurs in Cribl Stream to trigger a Notification condition, you will see an indicator in the **Messages** icon. Select the icon to open the **Messages** drawer and view the Notification details.

Add a Notification Target

To add a new Notification target:

1. In the Cribl Stream sidebar, select **Notifications**, and then select the **Targets** tab.
2. Select **Add Target** to open the **New Target** modal.
3. Give this target a unique **Target ID**.
4. Select the desired **Target type**.

Then configure the target as described in the appropriate page:

- Webhook
- PagerDuty
- Slack
- AWS SNS
- Email

When you create a Notification target in Cribl Stream, Edge, or Search, it will also be available to you in the other products. For example, if you create a Notification target in Cribl Stream, you can also access it in Cribl Edge and Cribl Search.

Notifications require certain plan/license tiers. Without an appropriate license, target configuration options will be hidden or disabled in Cribl Stream's UI.

Manage Notification Targets

You can manage targets by selecting **Notifications** > **Targets**.

Select any existing Notification target to open a modal where you can manage the target, using buttons at the bottom of the page.

Select **Delete Target** to delete an existing Notification target. You can also select multiple targets for deletion, using the check boxes on the left side of the modal.

Select **Clone Target** to copy an existing target. A modal will open so you can modify the configuration of the target you started with.

To edit any target's definition in a JSON text editor, select **Manage as JSON** at the bottom of the **Notifications** > **Targets** modal. You can directly edit multiple values, and you can use the **Import** and **Export** buttons to copy and modify existing target configurations as `.json` files.

Some targets have additional controls for target configuration. See the documentation for the target of interest for more information.

3.6.2.1. Webhook Notification Targets

With this option, you can send Cribl Stream Notifications to an arbitrary webhook. Select **Webhook** in the **New Target** modal to expose multiple left tabs, with the following configuration options:

General Settings

Target ID: Enter a unique ID used to identify the target. This will show in the **Target ID** column of the **Targets** tab. You can't change it later, so make sure you like it.

Configuration

The added options that appear on this first left tab are:

URL: The endpoint that should receive Cribl Stream Notification events.

To proxy outbound HTTP/S requests, see System Proxy Configuration.

Method: Select the appropriate HTTP verb for requests: `POST` (the default), `PUT`, or `PATCH`.

Format: Specifies how to format Notification events before sending them to the endpoint. Select one of the following:

- `NDJSON` (newline-delimited JSON, the default).
- `JSON Array`.
- `Custom`, which exposes these additional fields:
    - **Source expression**: JavaScript expression whose evaluation shapes the event that Cribl Stream sends to the endpoint - for example: `notification=${_raw}`. For other fields you can use, see Expression Fields. If empty, Cribl Stream will send the full Notification event as stringified JSON.
    - **Drop when null**: Toggle on if you want to drop events where the above **Source expression** evaluates to `null`.
    - **Event delimiter**: Delimiter string to insert between individual events. Defaults to newline character (`\n`).
    - **Content type**: Defaults to `application/x-ndjson`. You can substitute a different content type for requests sent to the endpoint. This entry will be overridden by any content types set in this modal's **Advanced Settings** tab > **Extra HTTP Headers** section.
    - **Batch expression**: Expression specifying how to format the payload for each batch. Defines a wrapper object in which to include the formatted events, such as a JSON document. This enables

requests to APIs that require such objects. To reference the events to send, use the `${events}` variable. An example expression to send the batch inside a JSON object would be: `{"items" : [${events}] }`.

Expression Fields

When building the Source expression, you can use the following fields:

Fields Common to All Notification Types

- `starttime`: Beginning of the time bucket where this condition was reported. All Notifications have this field.
- `endtime`: End of the time bucket where this condition was reported. All Notifications have this field.
- `_time`: Timestamp when this Notification was created. All Notifications have this.
- `cribl_host`: Hostname of the (physical or virtual) machine on which this Notification was created. All Destination and Source Notifications have this.
- `cribl_notification`: Configured name/ID of this Notification. All Destination and Source Notifications have this.
- `origin_metadata`: Object containing metadata about the Notification's origin, with the following fields for all Destination Notifications:
  - `type`: "output".
  - `id`: ID of the affected Destination.
  - `subType`: Destination's type (where applicable).
- `origin_metadata`: Object containing metadata about the Notification's origin, with the following fields for all Source Notifications:
  - `type`: "input".
  - `id`: ID of the affected Source.
  - `subType`: Source's type (where applicable).

Unhealthy Destination

- `health`: Numeric value where `0`=green, `1`=yellow, `2`=red.
- `output`: Output ID of the affected Destination.
- `_raw`: "Destination `${output}` [in group `${__worker_group}`] is unhealthy".
- `_metric`: "health.outputs".

Destination Persistent Queue Usage

- `output`: Output ID of the affected Destination.
- `timeWindow`: The interval at which Notifications will be repeated.
- `usageThreshold`: The minimum persistent queue utilization (stated as a percentage) that will trigger Notifications.
- `_metric`: "system.pq_used".
- `usage`: The persistent queue utilization (stated as a percentage) that triggered the Notification.

Destination Backpressure Activated

- `backpressure_type`: `1` for Block, `2` for Drop.
- `output`: Output ID of the affected Destination.
- `_raw`: "Backpressure ([dropping|blocking]) is engaged for destination `${output}` [in group `${__worker_group}`]".
- `_metric`: "backpressure.outputs".

Source High Data Volume

- `health`: Numeric value where `0`=green, `1`=yellow, `2`=red.
- `bytes`: Number of bytes received in the time bucket.
- `input`: Input ID of the affected Source.
- `_raw`: "Source `${input}` [in group `${__worker_group}`] traffic volume greater than `${dataVolume}` in `${timeWindow}`".
- `_metric`: "total.in_bytes".

Source Low Data Volume

- `health`: Numeric value where `0`=green, `1`=yellow, `2`=red.
- `bytes`: Number of bytes received in the time bucket.
- `input`: Input ID of the affected Source.
- `_raw`: "Source `${input}` [in group `${__worker_group}`] traffic volume less than `${dataVolume}` in `${timeWindow}`".
- `_metric`: "total.in_bytes".

Source No Data Received

- `health`: Numeric value where `0`=green, `1`=yellow, `2`=red.
- `_time`: Timestamp when this Notification was created.
- `input`: Input ID of the affected Source.
- `_raw`: "Source `${input}` [in group `${__worker_group}`] had no data for `${timeWindow}`".
- `_metric`: "total.in_bytes"

License Expiration

- `severity`: One of "warn" or "fatal".
- `title`: One of: "License expiring soon, data will stop flowing." Or: "License has expired. Data flow has been stopped.".
- `text`: One of: "License will expire on `${expirationDate}`, no external inputs will be read" after that time. Please contact sales@cribl.io to renew your license." Or: "License has expired."

Authentication

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Basic**: In the resulting **Username** and **Password** fields, enter HTTP Basic authentication credentials.

Post-Processing Settings

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Whether to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. When enabled, any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. When disabled, all `Retry-After` headers are ignored.

Settings for failed HTTP requests: When you want to automatically retry requests that receive particular HTTP response status codes, use these settings to list those response codes.

For any HTTP response status codes that are not explicitly configured for retries, Cribl Stream applies the following rules:

Status Code	Action
Greater than or equal to `400` and less than or equal to `500`.	Drop the request.
Greater than `500`.	Retry the request.

Upon receiving a response code that's on the list, Cribl Stream first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream continues retrying the request without increasing the time between retries any further.

By default, this Destination has no response codes configured for automatic retries. For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: When you want to automatically retry requests that have timed out, toggle this control on to display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).

- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates not authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle this off if you want Cribl Stream to close the connection immediately after sending a request.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB, but you can set this limit to as high as 500 MB (512000 KB).

High values can cause high memory usage per Worker Node, especially if a dynamically constructed URL causes this target to send events to more than one URL. The actual request body size might exceed the specified value because the target adds bytes when it writes to the downstream receiver. Therefore, we recommend that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass to all events as additional HTTP headers. Values will be sent encrypted. You can also add headers dynamically on a per-event basis in the `__headers` field; headers added by this method take precedence over headers defined in the **Extra HTTP headers** table.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Used to declare headers that are safe to log as plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

3.6.2.2. PagerDuty Notification Targets

This option sends Cribl Stream Notifications to PagerDuty, a real-time incident response platform, using Cribl Stream's native integration with the PagerDuty API. Select Select **PagerDuty** in the **New Target** modal to expose the following additional options on the modal's (single) **General Settings** left tab:

General Settings

Target ID: Enter a unique ID used to identify the target. This will show in the **Target ID** column of the **Targets** tab. It can't be changed later, so make sure you like it.

Configuration

Routing key: Enter your 32-character Integration key on a PagerDuty service or global ruleset.

Group: Optionally, specify a PagerDuty default group to assign to Cribl Stream Notifications.

Class: Optionally, specify a PagerDuty default class to assign to Cribl Stream Notifications.

Component: Optionally, a PagerDuty default component value to assign to Cribl Stream Notification. (This field is prefilled with `logstream`.)

Severity: Set the default message severity for events sent to PagerDuty. Defaults to `info`; you can instead select `error`, `warning`, or `critical`. (Will be overridden by the `__severity` value, if set.)

Retries

Honor Retry-After header: Whether to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. When enabled, any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. When disabled, all `Retry-After` headers are ignored.

Settings for failed HTTP requests: When you want to automatically retry requests that receive particular HTTP response status codes, use these settings to list those response codes.

For any HTTP response status codes that are not explicitly configured for retries, Cribl Stream applies the following rules:

Status Code	Action
Greater than or equal to `400` and less than or equal to `500`.	Drop the request.

Status Code	Action
Greater than `500`.	Retry the request.

Upon receiving a response code that's on the list, Cribl Stream first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream continues retrying the request without increasing the time between retries any further.

By default, this Destination has no response codes configured for automatic retries. For each response code you want to add to the list, click **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: When you want to automatically retry requests that have timed out, toggle this control on to display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

3.6.2.3. Slack Notification Targets

You can send Notifications to a Slack channel, using Slack's Incoming Webhooks.

You can also use a Webhook Destination to send specific event data to Slack for a team's attention. For more information, see our Slack/Webhook integration topic.

First, in your Slack workspace, use a Slack app to enable incoming webhooks.

Then, in the **New Target** modal, click **Slack** to expose the following additional options on the modal's (single) **General Settings** left tab:

General Settings

Target ID: Enter a unique ID used to identify the target. This will show in the **Target ID** column of the **Targets** tab. You can't change it later, so make sure you like it.

Configuration

Webhook URL: Add the full URL of your Slack Incoming Webhook. For example: `https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX`.

Retries

Honor Retry-After header: When toggled on and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds.

Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).

When toggled off (default), Cribl Stream ignores all `retry-after` headers.

Settings for failed HTTP requests: Automatically retries after unsuccessful response status codes, such as `429` (Too Many Requests) or `503` (Service Unavailable). Clicking **Add Setting** reveals a table where you can add retry parameters for individual failed HTTP requests. These include:

- **HTTP status code**: The individual status code for which the retry parameters apply.
- **Pre-backoff interval (ms)**: How long, in milliseconds, Cribl Stream should wait before initiating backoff. The maximum interval is 600,000 ms (10 minutes).

- **Backoff multiplier**: Base for exponential backoff. A value of `2` (default) means Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so forth.
- **Backoff limit (ms)**: The maximum backoff interval, in milliseconds, Cribl Stream should apply. Default (and minimum) is 10,000 ms (10 seconds); maximum is 180,000 ms (180 seconds).

Retry timed-out HTTP requests: When toggled on, Cribl Stream to automatically retries HTTP requests that have timed out. Default is toggled off. Enabling this option exposes the following settings:

- **Pre-backoff interval (ms)**: How long, in milliseconds, Cribl Stream should wait before initiating backoff. Maximum interval is `600,000 ms` (10 minutes).
- **Backoff multiplier**: Base for exponential backoff. A value of `2` (default) means Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so forth.
- **Backoff limit (ms)**: The maximum backoff interval, in milliseconds, Cribl Stream should apply. Default (and minimum) is 10,000 ms (10 seconds); maximum is 180,000 ms (180 seconds).

3.6.2.4. AWS SNS Notification Targets

You can send Notifications to an Amazon Simple Notification Service (SNS) topic. This gives you access to a broad array of Notification destinations, such as various AWS services, mobile push notifications, or text messages.

To add an Amazon SNS Notification target in Cribl Stream, select **Notifications** in the sidebar, and then select **Targets** > **Add Target** > **AWS SNS**.

General Settings

Target ID: Enter a unique ID used to identify the target. This will show in the **Target ID** column of the **Targets** tab. It can't be changed later, so make sure you like it.

Configuration

Destination type: Defaults to **Topic ARN**. The SMS section explains the **Phone number** option.

Default Topic ARN: The default Amazon Resource Name (ARN) of the Amazon SNS topic to which you want to send Notifications. Enter the ARN in the following format:

```
arn:aws:sns:region:account-id:MyTopic

// for example:
// arn:aws:sns:us-west-2:000000000000:my-topic
```

Copy

If you use a non-AWS URL, the format must be:

```
{url}/myQueueName

// for example:
// https://host:port/myQueueName
```

Copy

The default topic ARN must be a JavaScript expression (which can evaluate to a constant value), enclosed in quotes or backticks. Can be evaluated only at initialization time. For example, if you're referencing a Global

Variable: `` `https://host:port/myQueue-${C.vars.myVar}` ``. This value can be overridden by the Notification event `__topicArn` field.

Topic type: The type of the topic selected in AWS SNS. Can be **Standard** or **FIFO**.

Message group ID: The tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are processed in a first in, first out (FIFO) manner. Must be a JavaScript expression (which can evaluate to a constant value), enclosed in quotes or backticks. Can be evaluated only at initialization time. For example, referencing a global variable: `` `https://host:port/myQueue-${C.vars.myVar}` ``.

Region: Select the Region associated with the Amazon S3 bucket.

Phone number allowlist: A wildcard list of phone numbers that are allowed to receive SMS notifications. This is used when **Destination type** is set to **Phone number**.

Authentication

Auto: This default option uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance, local credentials, sidecar, or other source. The attached IAM role grants Cribl access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Worker Nodes not in an AWS VPC, like those running a private cloud.

The **Manual** option exposes these additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

The values for **Access key** and **Secret key** can be a constant, or a JavaScript expression (such as `${C.env.MY_VAR}`) enclosed in quotes or backticks, which allows configuration with environment variables.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The Secret option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. To store a new, reusable secret, select **Create**.

Assume Role

Enable for SNS: Toggle on to define an IAM Role to use, instead of automatically detecting one locally.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role. This is required only when assuming a role that requires this ID to delegate third-party access. For details, see AWS' documentation.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Post-Processing

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Maximum number of retries: The maximum number of retries before the output returns an error. The retries use an exponential backoff policy.

Endpoint: The SNS service endpoint. If empty, defaults to AWS' Region-specific endpoint. Otherwise, it must point to an SNS-compatible endpoint.

Signature version: Signature version to use for signing SNS requests. Defaults to `v4`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

SMS Notifications

You can use an Amazon SNS Notification target to send text messages (SMS) to a list of phone numbers. To do this, you'll need to set up a allowlist of phone numbers that are permitted to receive Notifications.

1. Select **Notifications** in the sidebar, and then select **Targets** > **Add Target**.

2. Enter a unique **Target ID**.
3. Set the **Target type** to **AWS SNS**.
4. Set **Destination type** to **Phone number**.
5. Set **Region** to the Region of the Amazon S3 bucket.
6. In **Default Phone number**, enter a comma-separated list of phone numbers that are allowed to receive Notifications. This value can be overridden by the Notification event `__phoneNumber` field. You can use `*` as the wildcard character.
   For example: `+15555550123, +15555551***`.
7. If desired, use **Phone number allowlist** to specify a wildcard list of allowed phone numbers.
8. Configure the remaining sections of the AWS SNS Notification target as described earlier.
9. Select **Save**.

Now, when you set up Notifications, you can select the new Amazon SNS target and specify any phone number that matches the configured allowlist.

3.6.2.5. Email Notification Targets

You can send notifications by email, using an SMTP server of your choice. Once you have configured the email notification target, you can specify the recipients and customize the subject line and message content. See Email Notifications for details on configuring an email Notification.

To add an email Notification target in Cribl Stream, select **Notifications** in the sidebar, and then select **Targets** > **Add Target** > **Email**.

General Settings

Target ID: Enter a unique name to identify this email Notification target.

Configuration

Address: Identify the SMTP server by its hostname or IP address.

Port: Set the SMTP port. Use port `587` for SMTP Secure (SMTPS). Use `465` when SSL/TLS is enabled. You can also use port `2525` if your email service provider supports this port as a backup when other ports are blocked by a network provider or a firewall.

In Cribl.Cloud, Organizations on an Enterprise plan can use the default email notification target - `system_email` - and avoid configuring an SMTP server. If you **do** wish to configure an SMTP server in Cribl.Cloud, use any port other than port `25`.

Encryption type: Specify the encryption type used to secure SMTP communication. Options include:

- **STARTTLS**: Select this option to start the connection as plaintext, then upgrade it to a TLS-encrypted one if the server supports it. If the server doesn't support it, the connection remains plaintext.

  STARTTLS upgrades the connection to be encrypted but does not authenticate the server. Take additional steps to prevent man-in-the-middle attacks.

- **Require STARTTLS**: Select this option to require TLS. If the server doesn't support a secure connection, the connection will be dropped.
- **TLS (SMTPS)**: Select this option to use an encrypted connection from the start without requiring a subsequent connection upgrade.
- **None**: Select this option to use a plaintext connection.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path** or by another trusted CA (such as the system's CA).

From: Identify the email address of the sender.

Authentication

Username: The authentication principal (if required).

Password: The authentication credential (if required).

Test Email Notification Targets

When you finish configuring your email Notification target, you can manage it, using the following options.

Click **Save & Test Target** to save your target and open a modal where you can enter an email to test your email Notification target. After configuration, the button text changes to **Test Target**.

Default Email Notification Target for Cribl.Cloud Enterprise Orgs

For Cribl.Cloud Enterprise orgs, a default email Notification target is available for Cribl Stream, Edge, and Search. You can use this target to route any email Notification to any valid email address.

This target appears on the **Notifications** > **Target** page as `system_email`. You cannot modify or remove this target.

The `system_email` Notification target is managed by Cribl. It will be disabled in the unlikely event of abuse. If this target is disabled for a workspace, a log entry will appear in the Notifications Service logs. Select **Monitoring** > **Logs** > **Notifications Service** to view this log.

The `system_email` email Notification target is available only for Cribl.Cloud Enterprise organizations.

Send Domain for Cribl.Cloud Email Notifications

Every Cribl.Cloud workspace and organization has a unique address for its email Notification target. Messages sent using this target will have a sender's address in the form `do-not-reply@<workspaceName>-<organizationId>.criblcloud.email`. To ensure delivery, recipients should add the `criblcloud.email` domain to their email allowlists.

3.7. Customize The Interface

You can customize the Cribl Stream interface to adapt it to your needs by adding:

- A custom login page with a selected logo and additional explanatory text.
- Custom banners to serve information, warnings, or news to users.

3.7.1. Custom Banners

Cribl admins can define a custom banner from the Leader. This will be displayed at the top of your Cribl Organization, above all Cribl apps. You can display one banner at a time.

Your banner can display warnings, requirements, neutral updates, or good news about new features or options. It can contain a mixture of text, a raw URL, or a URL with a friendly/short label. The banner does not appear unless you enable and configure it, as described here.

1. In the sidebar, select **Settings** > **Global**.
2. Select **System** > **Custom Banner**.
3. Toggle **Enable banner** on to expose the configuration options.
4. From the **Background color** dropdown, select a background color for your banner. You can also pick a custom color by selecting **+Add color** in the dropdown.
5. Optionally, toggle **Invert front color** to **Yes** to make the text on the banner white instead of black (typically, for darker custom banner colors).
6. In the **Banner message** field, enter a single-line message of up to 100 characters. The displayed banner will truncate characters that cannot fit across one line of a given viewport.
    - You can use emojis and special characters in the banner message. To open the menu of supported options, place the cursor in the **Banner message** field and press **Ctrl+Cmd+Space** (macOS) or **Windows key + semicolon** (Windows).
7. In the **Link URL** field, optionally enter a hyperlink to append to the message.
8. In the **Link display** field, optionally give your **Link URL** a text label. This can include up to 100 characters, but a long label can force-truncate the **Banner message**. A short label helps everything fit.
9. Select **Save** to display your banner.
10. Commit your changes from the Leader.

To hide the banner, toggle the **Enable banner** off, and again **Save** and commit your changes. Your banner's configuration will remain intact, ready to re-enable and/or revise when you're ready.

3.7.2. Custom Login Page

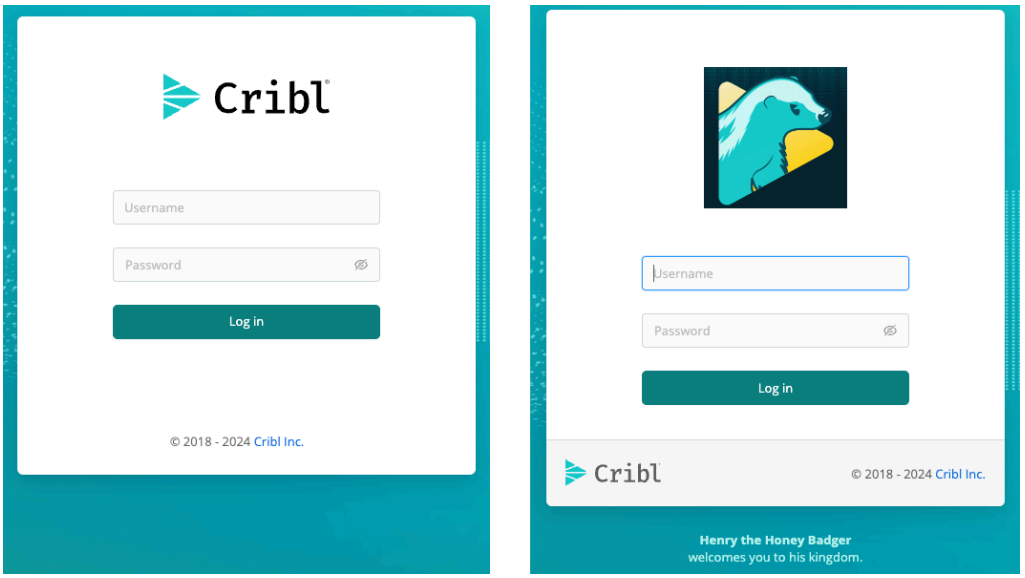
You can customize the look of the login page for all Cribl apps.

Custom login pages are available only in on-prem installations.

Configure Custom Login Page

To configure a custom login page:

1. In the Cribl Stream sidebar, select **Settings**, then **Global**.
2. Under **System**, select **General settings**.
3. Go to **Custom login page** and toggle **Enabled** on.
4. In **Logo file**, select an image file that will serve as a logo. The image must be no larger than 350x350px, under 2MB, and in .png, .jpg, or .jpeg format.
5. In **Content**, edit the provided HTML to modify the text block displayed under the login box.



Built-in and customized login pages

3.8. Scripts

Scripts are available only in on-prem deployments, not in Cribl.Cloud.

Admins can run scripts (for example, shell scripts) from within Cribl Stream by configuring and executing them at **Settings** > **Global** > **Scripts**. Scripts are typically used to call custom automation jobs or, more generally, to trigger tasks on demand. For example, you can use Scripts to run an Ansible job, or to place a call to another automation system, when Cribl Stream configs are updated.

With Great Power Comes Great Responsibility!

Scripts will allow you to execute almost anything on the system where Cribl Stream is running. Make sure you understand the impact of what you're executing before you do so!

Enable Scripts

For security reasons, using scripts is disabled by default on all new deployments. You can re-enable them by using the `limits` command (available since Cribl Stream 4.8.2).

To control script availability, run:

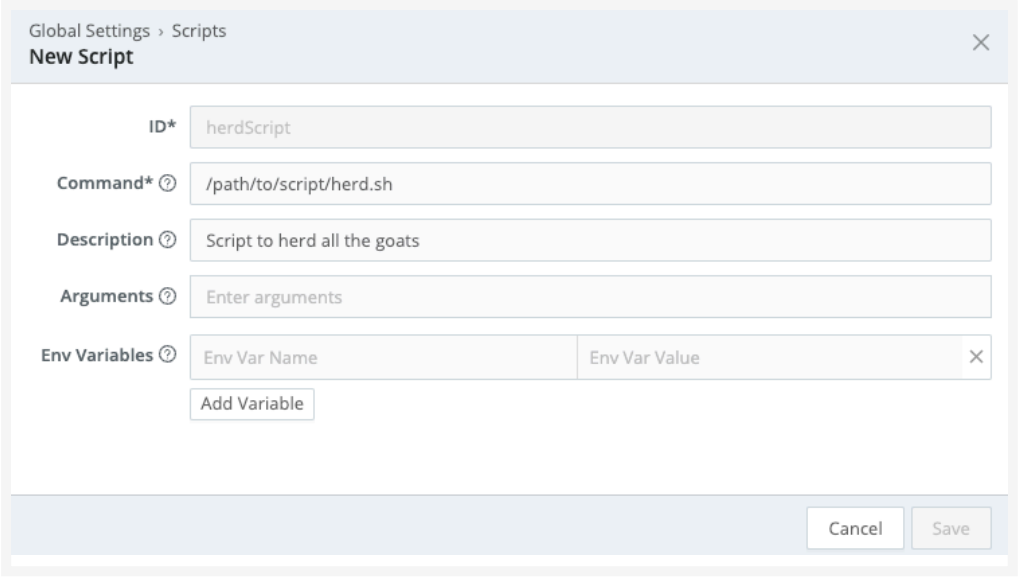
```
# Enable scripts
./cribl limits -s true
# Disable scripts
./cribl limits -s false
```

Copy

When you enable or disable scripts, you need to restart the instance for the changes to take effect.

Add Scripts

To add a new script, select **Add Script** on the scripts settings page.



Settings > Scripts page

The **New Script** modal provides the following fields:

- **ID**: Unique ID for this script.
- **Command**: Command to execute for this script.
- **Description**: Brief description about this script. Optional.
- **Arguments**: Arguments to pass when executing this script.
- **Env variables**: Extra environment variables to set when executing script.

Scripts in Distributed Deployments

- Scripts can be deployed from the Leader Node, but can be **run** only locally from each Worker Node.
- If the Script command is referencing a file (for example, `herd.sh`), that file must exist on the Cribl Stream instance. In other words, the Script management interface cannot be used to upload or manage script files.

3.9. Upgrade Cribl Stream

This page explains how you can upgrade a Cribl Stream single-instance or distributed deployment.

Considerations

Here are some considerations to look at before you upgrade.

Manual Upgrade Required for Hybrid and On-Prem Deployments

Whether your upgrade is automatic or manual depends on your deployment type:

- **For Cribl.Cloud:** Cribl Stream automatically upgrades the Leader Node in Cribl.Cloud deployments.
- **For customer-managed (hybrid) Cribl.Cloud Worker Groups, and all on-prem deployments:** Requires manual upgrades for Worker Groups and distributed instances.

Maintaining compatible versions across all Cribl Stream deployments–whether in the cloud, hybrid, or on-premise–is crucial for optimal performance and stability.

Supported Version Differences Between Leader and Workers

Cribl strongly recommends keeping Leader Nodes and Worker Groups/instances on the same version for the best experience and to ensure access to all features. Upgrading a Leader Node to a newer version may surface configuration options in the UI that will not be functional until you upgrade the connected Workers to the same version.

However, this section outlines the supported version differences between Leader Nodes and Worker Groups (or distributed instances) across all deployment types–including Cribl.Cloud, hybrid, and on-premise. Following these guidelines is essential to ensure functionality, compatibility, and full support coverage:

- **Patch Versions**: Cribl supports Leaders and Worker Groups or distributed instances on different patch versions within the same minor version. For example, a Worker Group on 4.5.0 and Leader on 4.5.1.
- **Minor Versions**: Cribl supports Worker Groups or distributed instances that are one minor version behind the Leader Node. For example, a Worker Group on 4.5.1 and a Leader is on 4.6.0.

Do not upgrade Workers to a newer version than the Leader Node.

Cribl does not support version differences greater than one minor version, which may lead to unpredictable behavior and compatibility issues. Support requests for configurations exceeding this limit will require updating to compatible versions.

Requirements for Specific Version Upgrades

- **Upgrading from a pre-GA candidate:** Cribl Stream does not support direct upgrades from any pre-GA version (such as a Cribl-provided test candidate) to a GA version. To get the GA version running, you must perform a new install.
- **Upgrading your Leader to 4.0 and newer:** Before upgrading, see Persisting Socket Connections to prepare the host to keep communications open from Worker Nodes.
- **Upgrading to 3.5.4 or newer:** All Worker Nodes will need to be on the same version as the Leader. Leaders running v.3.5.4 and newer test whether Worker Nodes are running a compatible version before deploying configs that could break Worker Nodes' data flow. The Leader will prompt you to upgrade these Nodes as needed.

  Version 3.5.4 was also a compatibility breakpoint for the Cribl HTTP Source and Destination, and for the Cribl TCP Source and Destination. When running on Cribl Stream 3.5.4 and newer, these two Sources can send data only to Nodes running v.3.5.4 and newer, and these two Destinations can receive data only from Nodes running v.3.5.4 and newer. When running on Cribl Stream 3.5.3 and older, these four integrations can similarly interoperate only with Nodes running v.3.5.3 and older.

- **Upgrade to v.4.3.x using Leader-Managed Upgrades for Worker Nodes:** If a Leader is on Cribl Stream v.4.3.x, you should upgrade all Worker Nodes to v.4.3.x **before** making any config changes or committing and deploying to the Nodes. If you have already committed and deployed to Nodes that are on a version older than 4.3.0, and the Leader is on v.4.3.x, you won't be able to upgrade the Nodes. In this case, there are three workaround options:
  - Revert and redeploy the last commit, then upgrade the Worker Nodes and deploy the most up-to-date changes; **or**
  - Upgrade the Worker Nodes via command line or script; **or**
  - Downgrade the Leader to v.4.2.2, and wait for a future release that will enable you to upgrade Worker Nodes across all Leaders and 4.x.x versions.
- **Upgrading from v.4.1 and older:** Before upgrading, encrypt TLS certificate private key files when you add or modify them. See the next section about Safeguard Unencrypted Private Key Files for Rollback for information about backing up your keys from older versions.

Safeguard Unencrypted Private Key Files for Rollback

Before upgrading from a version older than 4.1, make a backup copy of all unencrypted TLS certificate private key files. Having access to the unencrypted files is essential if you later find that you need to roll back to your previous version.

To safeguard your unencrypted private keys, make a full backup of all Cribl config files. Files in the `auth/certs` directory are particularly important, such as those in:

- `groups/default/local/cribl/auth/certs/`
- `groups/<groupname>/local/cribl/auth/certs/`
- `cribl/local/cribl/auth/certs/`

Take appropriate precautions to prevent unauthorized access to these unencrypted private key files. If you need to roll back to a version older than 4.1, see Restore Unencrypted Private Keys.

Network Connectivity Requirements

For a smooth upgrade process, Cribl Workers require reliable and timely network access to specific resources during the upgrade. Check the following:

- **Preferred Connectivity:** Workers attempt to download upgrade packages from the Cribl content delivery network (CDN) for efficient updates. A stable and responsive connection to `cdn.cribl.io` is a must for successful upgrades.
- **Alternative Connectivity (if CDN access is restricted):** If accessing `cdn.cribl.io` is not possible due to security restrictions or other reasons, Workers can fall back to downloading the upgrade package directly from the Leader.

Network Performance Considerations

When accessing the CDN at `cdn.cribl.io`, a slow or unreliable network connection will cause the upgrade to fail due to significant delays. The upgrade package size can be around 80 MB.

If you are having problems downloading the package from the CDN, check the following:

- **Ensure adequate network bandwidth and stability.** Test your network connection for reliability before upgrading.
- **Prevent RPC timeouts.** Cribl Stream versions 4.11.1 and newer have a higher RPC timeout threshold. Upgrade your Worker Nodes to v.4.11.1 or newer to help prevent future RPC timeouts.
- **Ensure access to the CDN (to download packages) is not blocked by firewall for upgrades.** See the following section about Considerations for Firewalls/Web Proxies for more information.

Service Timeout for Leader Upgrades

If your Leader is managed by a service manager (such as `systemd`, openRC, or Upstart), the default `TimeoutSec=60` might be insufficient for complex environments.

The cumulative time required to parse dozens of Worker Groups, or environments with a high density of Sources, Destinations, and Routes, can easily exceed the 60-second window. If this threshold is hit, `systemd` will end the process mid-migration, causing the upgrade to fail.

To prevent this, temporarily increase the timeout:

1. Open the service configuration (or use an override): `systemctl edit cribl.service`
2. Set a higher limit, for example: `TimeoutSec=600`
3. Reload the configuration: `systemctl daemon-reload`
4. Retry the upgrade.

Once the upgrade completes, repeat these steps to revert to the default value, `TimeoutSec=60`. This keeps your stop/restart cycles fast by allowing `systemd` to intervene quickly if the process becomes unresponsive.

For more background on timeouts during larger data migrations, see details in Addressing HA Migration Timeouts.

Considerations for Firewalls/Web Proxies

For optimal upgrade performance, configure firewalls to allow access to `cdn.cribl.io`:

- If your organization restricts access to `cdn.cribl.io`, configure firewalls to immediately deny connection requests. This allows Workers to promptly fall back to the Leader for downloading the upgrade package and avoid timeouts.
- You may encounter the "unsafe legacy renegotiation disabled" error when the Cribl Stream Leader attempts to connect to the CDN, preventing Worker Node upgrades. This error occurs if your firewall or proxy (like PAN firewalls) doesn't support the TLS Renegotiation Indication Extension (RFC 5746).
- Ensure your firewall or proxy is compliant with RFC 5746. Contact your firewall/proxy vendor for update instructions.

Disable SNI Routing for Firewalls/Web Proxies

If you're experiencing issues with proxies or firewalls between your Cribl Stream Worker Nodes and Leader Node, you can disable Server Name Indication (SNI) routing for specific Groups.

To disable SNI routing for a Group: Navigate to **Worker Group Settings** > **General Settings** > **SNI** > **Disable SNI Routing** option. **Commit** and **Deploy** and then proceed with the upgrade. You must enable this setting for every Group that is behind a web proxy or firewall.

Upgrade a Standalone/Single-Instance

To upgrade a single-instance deployment from the UI, select **Settings** > **System** > **Upgrade**.

To upgrade a single-instance deployment using the command line:

1. Stop Cribl Stream.
2. Uncompress the new version on top of the old one.

On some Linux systems, `tar` might complain with: `cribl/bin/cribl: Cannot open: File exists`. In this case, please remove the `cribl/bin/cribl` directory if it's empty, and untar again. If you have **custom functions** in `cribl/bin/cribl`, please move them under `$CRIBL_HOME/local/cribl/functions/` before untarring again.

3. Restart Cribl Stream.

Upgrade a Distributed Deployment

You can upgrade Leader and Worker Nodes with the command line interface (CLI) or the user interface (UI).

Upgrade and Automatic Rollback via the UI

When you choose to upgrade via the UI, Cribl Stream stops its own server, updates the installed package, and restarts Stream for you.

Upgrading through the UI is not supported for distributed environments with an additional Leader configured for high availability/failover. Instead, use the command line interface (CLI) to upgrade.

Here's how to access the Upgrade UI in Stream:

1. First, select **Settings** in the sidebar.
2. Select **Upgrade** depending on your deployment type:
    - In a single-instance deployment, select **System**, then **Upgrade**.
    - To upgrade the Leader Node, select **Global**, then **System** and **Upgrade**.
    - To upgrade a single Worker Group, select **Stream** then **Group Upgrade**.

You can also enable an automatic backup and rollback in case an upgrade fails.

These options will work only if all Stream instances (including Worker Processes) start at v.2.4.4 or higher. For other version-specific limitations, please see Known Issues.

Upgrade the Leader via the UI

To upgrade the Leader in a distributed deployment, go to **Settings** > **Global** > **System** > **Upgrade**.

When upgrading, you can choose to either download upgrade packages from Cribl's content delivery network (CDN), or from a filesystem location that you specify in the form of a path.

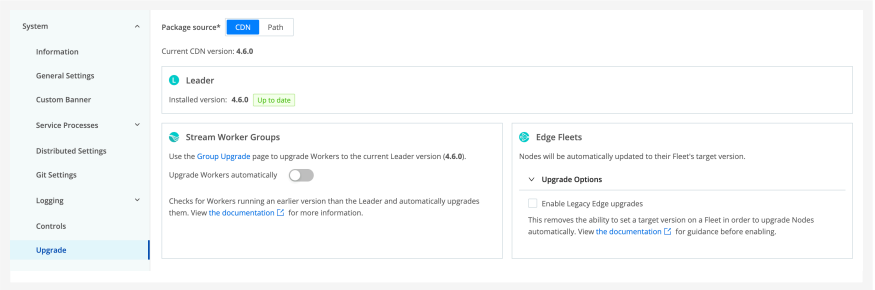
Select a Package Source

Here, you'll select between **CDN** and **Path** for your upgrade (in an on-prem, distributed deployment):

Package Source	Description	When You Select This Package Source
CDN	Downloads an installation package directly from Cribl's content delivery network.	With **CDN** selected, you can see the currently installed version and the version available on the CDN. The Leader will always upgrade to the current CDN version when you select **Upgrade**.
Path	You designate the path to the installation package in the **Path** field.	You must designate the correct paths for your Leader and Worker Groups, both version and architecture. For example, a Worker Group that contains ARM64 Worker Nodes will require an ARM64 installation package.

Use the buttons to choose the desired package source. Then, the UI will display settings and information appropriate for the package source you chose, as described in the respective Configure Stream Settings for CDN Upgrade and Configure Stream Settings for Path Upgrade sections below.

Configure Stream Settings for CDN Upgrade



Upgrade Stream using CDN

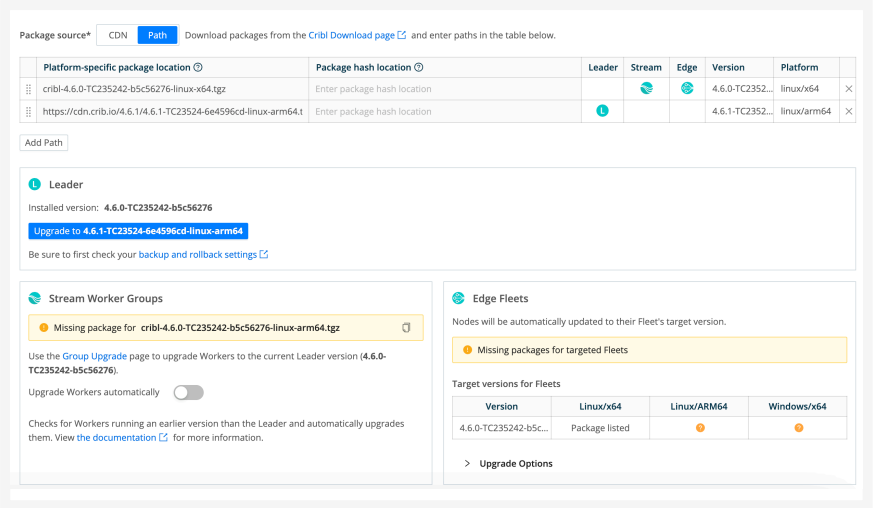
If you select **CDN** as your upgrade method, you will see the following options:

- **Current CDN version**: This lists the latest version of Cribl Stream available on the CDN.
- **Leader**: This shows the currently installed version of Cribl Stream on the Leader. If a newer version is available, you will be able to use the **Upgrade to** button.
- **Stream Worker Groups**: Provides a quick link to the Group Upgrade page if you want to upgrade Worker Groups individually.

  In Cribl.Cloud, this area will only show hybrid Worker Nodes.

  To automatically upgrade Worker Nodes to the Leader version, enable **Upgrade Workers automatically**. See Automatic Upgrades for more details.

Configure Stream Settings for Path Upgrade



Upgrade Stream using Path

With **Path** as your upgrade method, you'll see the following settings and information in the **Package Source** table:

- **Platform-Specific Package Location**: Enter or paste the path to the Cribl installation package. This can be either of the following:
  - An HTTP URL, for example `https://cdn.cribl.io/dl/4.1.0/cribl-4.1.0-6979aea9-linux-x64.tgz`.
  - A local filesystem path, for example `myfolder/directory/cribl-package.tgz`. In a distributed deployment, the Leader must have access to the local filesystem location. The Leader retrieves the file and distributes config bundles to Worker Nodes and Edge Nodes.
- **Package Hash Location**: Enter either of the following:
  - An HTTP URL, for example `https://cdn.cribl.io/dl/4.1.0/cribl-4.1.0-6979aea9-linux-x64.tgz.sha256`.
  - A local filesystem path to the hash that validates the package.

  Supports SHA-256 and MD5 formats. You can simply append `.sha256` to the contents of the **Platform-specific package location** field.

When Stream is running in FIPS mode, MD5 is not an acceptable hashing algorithm.

- **Leader**: Indicates the version the Leader will be upgraded to when an upgrade is available and you select the **Upgrade to** button.
- **Stream**: The version in this row will be used to upgrade the Stream Worker Group, because it matches the Leader's version.
- **Edge**: The version in this row will be used to upgrade Edge Fleets with a matching target version.
- **Version**: Displays the package version in this row.

- **Platform**: Displays the platform architecture for the package in this row.

Select **X** to immediately delete a row - there is no confirmation prompt.

> You can add multiple rows to this table to specify packages for different versions and platforms/architectures. To obtain the latest packages from https://cribl.io/download, use the drop-down list to specify each platform (for example, x64 versus ARM). When you stage these packages on your own servers, preserve the original file names.

Leader area: Shows you the currently installed Leader version and whether an upgrade is available. Check the backup and rollback settings before you upgrade.

Stream Worker Groups: Provides a quick link to the Group Upgrade page if you want to upgrade Worker Groups individually.

To automatically upgrade Worker Nodes to the Leader version, enable **Upgrade Workers automatically**.

The **Edge Fleets** area offers **Upgrade Options** for Edge. See Package Sources for On-Prem Upgrades for more information.

Missing Package Links

When you see this warning, it means there are Worker Groups that can't be upgraded due to a missing package path.

To add package links:

1. Select the **Copy** icon to add the link to your clipboard.
2. Paste the copied package link into a new **Platform-specific package location** field.
3. Add the appropriate path directory to the beginning of the installation package path (where your Cribl upgrade packages are located).
   For example: `myfolder/directory/cribl-package.tgz`
4. Select **Save** to add the upgrade package path and resolve the warning.

Automatic Upgrades

You can enable **Upgrade Workers automatically** in **Settings** > **Global** to check for Worker Nodes running an earlier version than the Leader, and automatically upgrade them to match the Leader.

Enable automatic upgrades has a different default state depending on the deployment type:

- In on-prem deployments, toggle **Enable automatic upgrades** off (default) to prevent the Leader from automatically upgrading out-of-date Worker Nodes. Before toggling on, upgrade the Leader itself to Cribl Stream's most recent version.

- In Cribl.Cloud deployments, toggle **Enable automatic upgrades** on. This enables Cribl to automatically upgrade Worker Nodes to Cribl Stream's newest stable version. (Cribl-managed/Cribl.Cloud and customer-managed/hybrid Worker Nodes will auto-upgrade as soon as they see a new Leader version.) If you toggle **Enable automatic upgrades** off, you will need to explicitly upgrade each Worker.

When enabled, the automatic Worker upgrade process works like this:

1. The Leader pulls packages, and checks their hashes.
    - The Leader must be able to connect to the path.
    - The Leader must also have privileges to download the files.
    - If the path is an HTTP URL, the Leader copies the file to a known location in its filesystem.
    - If the package is already hosted on the Leader Node, specify its filesystem path.
2. Worker Nodes pull packages and check their hashes.
    - Worker Nodes pull from Cribl's content delivery network (CDN) and fall back to the Leader through HTTP, not directly from the Leader's filesystem.
    - Each Worker pulls the package that is appropriate for that Worker's platform and architecture.
3. Worker Nodes install the packages.

Upgrade Worker Groups via the UI

To upgrade a Worker Group:

1. Go to **Settings**, then **Stream**.
2. Select **Group Upgrade**.
3. Select the **Upgrade** button in a row to upgrade that Worker Group.

The **Upgrade Group** modal offers two states: Basic Upgrade and Advanced Upgrade.

Upgrading Worker Nodes from the Leader Node requires a Cribl Stream Standard or Enterprise license.

Basic Upgrade Configuration

In this default **Upgrade Group** modal, you can simply upgrade the whole Group, by selecting the modal's **Upgrade** button to confirm.

When one or more Worker Nodes can be upgraded in the Group, the **Upgrade** button will be enabled.

By selecting the **Upgrade** button, you can initiate the upgrade job with a task for each Worker that can be upgraded.

Cribl Stream will check to ensure that Worker Nodes are upgraded no higher than the Leader's version. Upgrades are performed as the user that was running Cribl Stream on each machine.

The **Worker Nodes** column will display a warning icon with a tooltip if the Group contains Worker Nodes that can't be upgraded for one or more of the following reasons:

- Worker Nodes are already running the current version.
- Worker Nodes running v.2.4.4 or older are too old to upgrade.

When none of the Worker Nodes in the Group can be upgraded, the **Upgrade** button will be disabled.

Advanced Upgrade Configuration

Select the modal's gear ( ) button to expose these additional options:

Quantity %: Specify what percentage of the Group's Worker Nodes to upgrade in this operation. If you enter a value less than the default `100%`, Cribl Stream will perform a partial upgrade, keeping the remaining Worker Nodes active to process data.

Rolling upgrade: This option upgrades Worker Nodes in batches, each sized according to the value in the **Quantity %** field. When enabled, this toggle also enables the modal's two remaining controls:

- **Retry delay (ms)**: How many milliseconds to wait between upgrade attempts. Defaults to `1000` ms (1 second).
- **Retry count**: How many times to retry a failed upgrade. Defaults to `5`.

After you confirm the Worker Group upgrade, the table on the Group Upgrade page will display an additional button on this Group's row:

- **View**: Select to display the upgrade task's status in the Job Inspector modal
- select that modal's **System** tab to access details.

When you initiate an upgrade via the UI, the new package is untarred to `$CRIBL_HOME/unpack.<random-hash>.tmp`. This location inherits the permissions you've already assigned to `$CRIBL_HOME`.

Backup and Rollback

When you initiate an upgrade through the UI, Cribl Stream first stores a backup of your current stable deployment. If the upgrade fails, then by default, Stream will automatically roll back to the stored backup package. You can adjust this behavior in **Settings**.

Go to **Settings** > **Global**, then **System** and **General Settings**, then **Upgrade & Share Settings**.

Use the following controls to adjust backup and rollback behavior.

Cribl Stream can perform rollbacks only on Worker Nodes/instances that were running at least v.3.0.0 before the attempted upgrade.

Enable automatic rollback: Cribl Stream will automatically roll back an upgrade if the Cribl Stream server fails to start, or if the Worker Node fails to connect to the Leader. (Toggle off to defeat this behavior.)

Rollback timeout (ms): Time to wait, after an upgrade, before checking each Node's health to determine whether to roll back. Defaults to `30000` milliseconds (30 seconds).

Rollback condition retries: Number of times to retry the health check before performing a rollback. Defaults to `5` attempts.

Check interval (ms): Time to wait between health-check retries. Defaults to `1000` milliseconds (1 second).

Backups directory: Specify where to store backups. Defaults to `$CRIBL_HOME/state/backups`.

Backup persistence: A relative time expression specifying how long to keep backups after each upgrade. Defaults to `24h`.

Upgrade and Rollback via the CLI

For a Distributed Deployment, this is the general upgrade order:

First, upgrade the Leader Node. Then, upgrade the Worker Nodes. Lastly, commit and deploy the changes on the Leader.

For distributed environments with a second (standby) Leader configured for high availability/failover, this is the upgrade order:

1. Stop the standby Leader for your Stream or Edge deployment.
2. Upgrade the standby Leader.
3. Start the standby Leader.
4. Stop the primary Leader. The standby will take over as primary.
5. Upgrade the former primary Leader.
6. Start the former primary Leader, which will become the new standby Leader.

Upgrade the Leader Node via the CLI

1. Commit and deploy your desired previous version. (This will be your most recent checkpoint.) Optionally `git push` to your configured remote repo.

2. Stop Cribl Stream.
    - Optional but recommended: Back up the entire `$CRIBL_HOME` directory.
    - Optional: Check that the Worker Nodes are still functioning as expected. In the absence of the Leader Node, they should continue to work with their last deployed configurations.
3. Uncompress the new Cribl Stream version on top of the old one.
4. Change ownership of the `cribl` directory as needed: `[sudo] chown -R cribl:cribl $CRIBL_HOME`
5. Restart Cribl Stream and log back in.
6. Wait for all the Worker Nodes to report to the Leader, and ensure that they are correctly reporting the last committed configuration version.

Upgrade from 4.1.0

When upgrading a Leader Node from version 4.1.0 using the UI, you may encounter the following error:

```
{"status":"error","message":"Protocol \"https:\" not supported. Expected
\"http:\"","error":{"code":"ERR_INVALID_PROTOCOL"}}
```

To resolve this, configure the `no_proxy` directive to include the API's listening address on the Leader Node. To encompass all addresses, set `no_proxy` to `0.0.0.0`. This prevents internal traffic on the Leader Node from passing through the proxy server.

Here's an example `No_Proxy` configuration:

```
http_proxy=http://localhost:1234 https_proxy=http://localhost:1234
no_proxy=0.0.0.0 bin/cribl start
```

This extra step is due to changes in communication between the connection listener service and the API server via an `http` client.

Upgrade the Worker Nodes via the CLI

These are the same basic steps as when upgrading a single instance deployment, above:

1. Stop Cribl Stream on each Worker Node.
2. Uncompress the new version on top of the old one.
3. Change ownership of the `cribl` directory as needed: `[sudo] chown -R cribl:cribl $CRIBL_HOME`
4. Restart Cribl Stream.

Commit and Deploy All Changes from the Leader Node

1. Ensure that newly upgraded Worker Nodes report to the Leader with their **new** software version.

2. Commit and deploy the newly updated Leader configuration after all Worker Nodes have upgraded.

Successful upgrade confirmation

Breaking Change at v.3.2

Because of a breaking change at v.3.2, Leaders running v.3.2.x cannot upgrade (via the UI) Worker Nodes running versions older than 3.2.0. The upgrade will fail with errors of the form: `Error checking upgrade path` and `Cannot read property 'greaterThan' of undefined`.

The workaround is to upgrade these Worker Nodes via the filesystem to v.3.2.0 or higher. The error does not affect upgrades of Worker Nodes running v.3.2.0+.

Manual Rollback

Using CLI commands, it's possible to explicitly roll back an on-prem Leader, and on-prem or hybrid Worker Nodes, to an earlier release. This works much like an upgrade.

Explicit rollback might be necessary if an automatic rollback fails. Otherwise, Cribl recommends first considering other options - an upgrade, or working with Cribl Support - before a manual rollback. Rollback can encounter these complications:

- Your current configuration might take advantage of dependencies not supported in an earlier release.
- Do not manually roll back any Cribl Stream instance running in a container. Instead, locate, download, and launch a container image hosting the earlier version you want.

Rollback Outline

We assume that you are rolling back to a previously deployed version that you know to be stable in your environment. The broad steps are:

1. Ideally, link your deployment to a Git remote repo. Commit and push your Leader's configuration to that remote. The repo will provide a stable location from which to recover your config, if necessary.
2. Stop the Leader instance. (From `$CRIBL_HOME/bin/`, execute `./cribl stop`.)
3. Also create a local backup of your Leader's whole `$CRIBL_HOME` directory.
4. Obtain the installation package for the earlier release and platform you need (see the Rollback Example).
5. Uncompress the earlier version to your original deployed directory. You can do this from the command line or programmatically (see the Rollback Example).
   If installing to your existing target directory fails, try the same mitigations we list above for upgrades.
6. In a distributed deployment, Worker Nodes must not run a higher version than the Leader. Repeat steps 2, 3, and 5 above on all Worker Nodes (substituting "Worker" for "Leader").

Rollback Example

Although rollback can be partially scripted, you cannot discover older installation packages programmatically - the initial steps here require human intervention:

1. Stop and back up the Leader. (Follow steps 1-3 in the Rollback Outline.)
2. Open the **Cribl Releases** download page: https://cribl.io/download/.
3. In the **Cribl Past Releases** section, locate the older version that you want to restore.
4. Select your target platform from the adjacent drop-down menu.
5. Click the corresponding download button. The browser downloads the installation package for your target platform and saves it directly to your designated download location. The download URL for installation packages is similar to this example for v.4.1.0: `https://cdn.cribl.io/dl/4.1.0/cribl-4.1.0-6979aea9-linux-arm64.tgz`
6. Navigate to your designated download location to locate the downloaded installation package.
   - For `.tgz` files, double-click the installation package to extract the contents or run a command like `tar -zxvf <fileName>.tgz`
   - For `.msi` files, double-click the installation package to launch the installer
7. Repeat the preceding steps to adjust all Worker Nodes to a compatible version.

Splunk App Package Upgrade Steps

Follow these steps to upgrade from v.1.7, or newer, of the Cribl App for Splunk:

1. Stop Splunk.
2. Untar/unzip the new app version on top of the old one.

On some Linux systems, `tar` might complain with: `cribl/bin/cribl: Cannot open: File exists`. In this case, please remove the `cribl/bin/cribl` directory if it's empty, and untar again. If you have **custom functions** in `cribl/bin/cribl`, please move them under `$CRIBL_HOME/local/cribl/functions/` before untarring again.

3. Restart Splunk.

Upgrade from Splunk App v.1.6 (or Lower)

As of v.1.7, contrary to prior versions, Cribl's Splunk App package defaults to Search Head Mode. If you have v.1.6 or earlier deployed as a Heavy Forwarder app, upgrading requires an extra step to restore this setting:

1. Stop Splunk.
2. Untar/unzip the new app version on top of the old one.

3.9.1. Upgrade Outpost

Use Outpost Group configuration to automatically upgrade Outposts.

Required Permission: **Admin/Owner**

Outposts are upgraded in bulk by setting the desired target version for the Outpost Group they belong to.

To automatically upgrade Outposts in a group:

1. In the sidebar, select **Outposts**.
2. In the row for the Outpost Group you want to upgrade, select the button in the **Target Software Version** column.
3. In the **Target Version** modal, select a version from the **Upgrade target version** dropdown. You have two options:
    - **None** - when set to None, the Outposts in the group will never upgrade automatically.
    - **<version> (current leader version)** - when set to this option, the Outposts will be upgraded to match the software version of the currently running Leader Node.
4. Confirm with **Save**.

The next time the Outpost in the selected group connects to the Leader, it will upgrade to the selected version.

Outposts in the selected group will not upgrade to any future version until you change the target software version in the drop-down (even if you upgrade the Leader).

The **Add/Update Outpost Node** command uses the **Target Software Version** for your Outpost Group when constructing the script for upgrading your Outposts.

See Add an Outpost with a Bootstrap Script for more details about the command.

3.10. Uninstall Cribl Stream

Removing an Cribl Stream installation, whether for a clean reinstall or permanently

---

Uninstalling the Standalone Version

- (Optional) To prevent systemd from trying to start Cribl Stream at boot time, run the following command: `sudo $CRIBL_HOME/bin/cribl boot-start disable`
  If you're running both Cribl Stream and Cribl Edge on the same host, be sure to execute this command in the same mode (Stream or Edge) and the same directory in which you originally ran `cribl boot-start enable`.
- Stop Cribl Stream (stopping the main process).
- Back up necessary configurations/data.
- Delete the `cribl` user: `userdel cribl`
- Delete the user group: `groupdel cribl`
- Remove the directory where Cribl Stream is installed.
- Directly delete any local files that Cribl Stream added while running, such as the `.dat` file and other local configs.
- Remove any leftover files in the system's temporary directory. On Linux, this is typically `/tmp`.
- If you've customized the **Helper processes socket directory** settings to change the default `/tmp` directory, adjust the cleanup accordingly. For the Leader, this setting is located in the **Distributed Settings** > **Leader Settings** > **Helper processes socket dir**. For the Worker Node, it is located in **System** > **General Settings** > **Sockets** > **Directory setting**.
- Remove any related configuration files and scripts, and disable the Cribl service if using a service manager.

In a Distributed deployment, repeat the above steps for the Leader instance and all Worker instances.

Uninstalling the Splunk App Version

- Stop Splunk.
- Back up necessary configurations/data.
- Remove the Cribl App in `$SPLUNK_HOME/etc/apps`.
- Remove the Cribl module in `$SPLUNK_HOME/etc/modules/cribl` (some versions).

4. Secure Your Deployment

Secure your Cribl Stream deployment, whether hosted on Cribl.Cloud, hybrid or on-prem, by following these guides:

- Secure your Cribl.Cloud Deployment
- Cribl.Cloud Security Practices
- Secure your On-Prem/Hybrid Deployment
- Secure Leader and Communication
- Manage Secrets and Keys
- Compliance

4.1. Secure Your Cribl.Cloud Deployment

Cribl.Cloud is SOC 2 Type II-certified and GDPR-compliant, simplifying compliance with industry standards and regulatory requirements. For Cribl.Cloud security architecture details, see Cribl.Cloud Security Practices.

To secure your Cribl.Cloud deployment, consider implementing the following.

Secure Your Cribl.Cloud Account

Create a strong password during your initial setup.

To create a secure password, ensure it is at least 12 characters long and includes a combination of lowercase letters, uppercase letters, numbers, and special characters (!@#$%^&*). Follow the on-screen instructions carefully to meet these requirements.

(For Cribl.Cloud setup instructions, see Initial Cribl.Cloud Setup).

Review Your Network Settings

When you log into your Cribl.Cloud Organization, navigate to the **Workspace** link. Here you can check and manage connectivity details - Data Sources, Access Control, and Trust relationships - for your Cribl-managed Workers in Cribl.Cloud.

On the **Data Sources** tab, you can review the available ports and TLS configurations.

Note that TLS encryption is pre-enabled for you on some, but not all Sources. For more information, see TLS in Cribl.Cloud.

Configure TLS Mutual Authentication on Cribl.Cloud

Within Cribl.Cloud, enabling mutual TLS authentication for individual data sources requires a trusted Certificate Authority (CA) certificate chain. This CA certificate verifies the legitimacy of the client certificate presented by the source during connection. For details, see TLS Mutual Authentication in Cribl.Cloud.

Set Up Role-Based Access Control for Members

Cribl.Cloud offers detailed role-based access control (RBAC) to ensure appropriate permissions and security. For details on how to invite and manage the roles for Members of your Organization, see Members.

Set Up Your Preferred Authentication Method

When your Cribl.Cloud Organization uses a certain plan, you have a range of secure and convenient login options to fit your Organization's needs, including:

- Local Accounts - Create traditional usernames and passwords for direct login. For details, see Local Authentication.
- Federated Authentication (OIDC and SAML): Integrate with existing identity providers (IdPs) like Okta or Entra ID for centralized user management and single sign-on (SSO). For details, see Cribl.Cloud SSO Setup.

Data at Rest

Cribl.Cloud provides built-in encryption for data at rest, ensuring that your information remains secure within its infrastructure.

Secure Data in Motion

For an extra layer of protection, you can enrypt individual fields within events. For details, see Encryption of Data in Motion.

Currently, Cribl Stream supports decryption only when Splunk is the end system. For details, see Decryption of Data in Splunk.

For a comprehensive solution, we recommend configuring TLS. TLS encrypts the entire data stream, including all event fields, regardless of Source or Destination type. This provides a more robust security posture for your data in motion. For details, see Secure Cribl Stream Sources and Destinations with Certificates.

Configure a Key Management Service (KMS)

A Cribl.Cloud Organization on a certain plan offers a Key Management Service (KMS) that strengthens the security of your data. KMS securely manages the encryption keys used to protect sensitive information on Worker Groups and Nodes within Cribl. For details on how to activate and configure KMS, see KMS Configuration.

(Optional) Segment Your Organization with Workspaces

Segment your organization's data and user access using Workspaces. This multi-tenancy feature allows you to create isolated instances within your Cribl.Cloud Organization, addressing critical security, compliance, and data isolation requirements tailored for diverse business units.

Each Workspace boasts a dedicated virtual private cloud (VPC). This essentially creates a separate environment within your Organization for each Workspace, ensuring the complete isolation of data and users in different Workspaces.

For details, see Configuring Workspaces.

(Optional) Create Isolated Data Environments Using Stream Projects

Within Cribl Stream, Projects offer a more granular level of isolation allowing teams and users to manage and share data flows without impacting others. Projects enable you to define the data that the Project's users can consume, where that data can be sent, and who has access to the Project.

Projects are ideal for teams and users who need to share and manage specific data, with accelerated access to relevant data and minimal configuration requirements. Data can be forked into full-fidelity versus redacted views to match the authorzation requirements and needs of different teams or users.

For details, see Configuring Projects.

4.2. Secure Your On-Prem/Hybrid Deployment

This document outlines essential security considerations to safeguard your on-prem/hybrid Cribl deployment.

Secure Your Installation

Secure your Cribl Stream deployment by ensuring the following during installation.

Choose a Secure Environment

Protect your Cribl deployment by:

- **Isolating your deployment** - deploy Cribl Stream on a dedicated network segment to minimize exposure.
- **Restricting physical access** - control who can physically access the deployment hardware.

Configure a System Proxy

Enhance the security and reliability of your Cribl deployment by leveraging proxy servers.

- To configure a system proxy for outbound HTTP/S traffic: Set the HTTP_PROXY and HTTPS_PROXY environment variables before starting Cribl Stream, specifying the proxy's IP address or FQDN and optionally the port number. For details, see System Proxy Configuration.
- To configure a SOCKS proxy for secure communication between Leader and managed Nodes: Follow the steps in Configure a SOCKS proxy. This will allow you to bypass network restrictions and ensure uninterrupted data flow.

Create a Dedicated User

To enhance security, avoid running Cribl Stream with root privileges. Instead, create a dedicated user account with limited permissions.

For details, see Setting Capabilities for Cribl Stream or Running Edge as an Unprivileged User.

Change Default Admin Password

After your initial login, immediately create a strong, unique password for the administrator account. This will help prevent unauthorized access.

Note: Worker Node passwords are managed by the Leader Node.

Locate and Access Your Cribl Secrets

When you install Cribl Stream, the following security files are created:

- `cribl.secret` - This file stores a unique cryptographic key used for authentication, password encryption, and key encryption. It's essential for securing your Cribl Stream deployment when authenticating with integrated services.
- `keys.json` - This file contains cryptographic keys used for internal operations within Cribl Stream.

These files are stored in:

Single-instance deployment - both files are located in `$CRIBL_HOME/local/cribl/auth/`.

Distributed deployment - both files reside on the Leader Node, in `$CRIBL_HOME/groups/<group-name>/local/cribl/auth/`.

You can use the **Settings** > **Security** > **Secrets** section to create and update authorization tokens, username/password combinations, and API-key/secret-key combinations for reuse across the application. For details, see Secrets.

Set Your Remote Git Repo to Private

Cribl uses Git for efficient configuration management. In Distributed deployments, a local Git repository tracks configuration changes. For added security and disaster recovery, you can also connect to a remote Git repository. Here are some key considerations.

- **Private repositories** - Always ensure that your remote Git repository is private, to prevent unauthorized access to your configuration files.
- **Disaster recovery** - Remote repositories provide a valuable backup for your Cribl configuration in case of system failures or data loss.

FIPS Mode (Cribl Stream only)

Do not start Cribl Stream without FIPS mode enabled, or else you will be unable to run it in FIPS mode later. You must enable FIPS mode as described in the FIPS Mode document, after installing **but before starting** Cribl Stream.

Secure the Leader Auth Token

Cribl Stream generates a strong, random authentication token for new installations and when transitioning from Single-instance to Distributed mode.

If you need to change the default token:

- Create a unique, secure value.

- Update the token in the UI or Docker Compose file.

For details, see How to Secure the Auth Token for the Leader Node.

Set up Role-Based Access Control for Members

Cribl offers detailed role-based access control (RBAC) to ensure appropriate permissions and security. For details on how to define and manage fine-grained access control across Cribl products and resources, see Members and Permissions.

Enhance Security with an Identity Provider (IDP)

We strongly recommend using an external Identity Provider (IDP) to implement single sign-on (SSO) for Cribl. This enhances your organization's security, by centralizing user authentication and authorization.

For details, see:

- SSO with Okta and OIDC (on-prem)
- SSO with Okta and SAML
- SSO with Microsoft Entra ID (On-Prem)

Secure Your Communication

Secure your Cribl communication channels by using the following options.

- **SSL/TLS** - Encrypt Cribl access and traffic with SSL or TLS.
- **Custom headers** - Enhance security with custom HTTP headers.
- **Encryption** - Create and manage your own keys to encrypt events in real time.
- **Secrets** - Centrally manage secrets that your instances use to authenticate on integrated services.
- **KMS** - Use internal or external key management service for advanced key management.

Choose a combination that best aligns with your organization's security policies.

Secure Access to the Leader and Worker Nodes

You can secure access to the Cribl Edge API and UI by configuring Secure Sockets Layer (SSL) or Transport Layer Security (TLS). Do this on the Leader, to secure inbound communications to the Edge Nodes. You can use your own certs and private keys, or you can generate a pair with OpenSSL. For details, see Configure TLS for API and UI Access.

For TLS details, see the following:

- TLS version support and defaults

- TLS settings and Traffic Types
- System-wide TLS Settings Including Ciphers

Import an Existing Certificate and Key

You can secure Leader-to-Worker Node communications by using an existing TLS/SSL certificate and key, in `\.pem\` format. For details, see Import Certificate and Key.

Create a Self-signed Certificate

You also have the option to create a self-signed certificate and key by following the steps outlined in Configure SSL.

Connect to the Leader Securely

Configure secure communication between your Leader and Edge Nodes using either the user interface, the `instance.yml` configuration file, or environment variables. For details, see Secure Leader/Worker Nodes Communications.

Secure Access to the Worker/Edge Nodes UI

Cribl recommends that in Enterprise Distributed deployments, you prevent direct browser access to the UI for all Worker Nodes. For details, see Secure Access to Worker Node UIs.

Configure TLS Mutual Authentication (optional)

Implement client certificate exchange to enhance security by allowing only authorized clients with valid certificates to connect to the Leader. Configure this feature after setting up encrypted TLS communication. For details, see Configuring TLS Mutual Authentication.

Secure Communication Between Sources and Destinations

To authenticate Cribl Stream with external data senders and receivers, configure certificates at the Worker Group level. For details, see Secure Cribl Stream Sources and Destinations with Certificates.

Configure Custom HTTP Headers

Custom HTTP headers are metadata that can be sent along with HTTP requests and responses. They provide a way to convey extra data that is not part of the standard HTTP protocol. You can customize Cribl's security settings with custom HTTP headers. For details, see Custom HTTP Headers.

Create Encryption Keys

You can create and manage keys that Cribl Stream will use for real-time encryption of fields and patterns within events. For details, see:

- Accessing Keys.
- Add New Key.
- Get Key Bundle.
- Safeguarding Unencrypted Private Key Files for Rollback (Edge/Stream).
- Restoring Unencrypted Private Keys.

Configure a Key Management Service (KMS)

Cribl offers a built-in Key Management Service (KMS) that strengthens the security of your data. With Standard and Enterprise license tiers, you have the option to integrate an external KMS. A KMS securely manages the encryption keys used to protect sensitive information on Worker Groups and Worker Nodes within Cribl Stream.

For details, see Configure KMS.

Create Secrets

Use the Cribl secrets store to isolate and simplify the management of authentication tokens, usernames/passwords, and API keys.

- To find out more about secrets, see The cribl.secret File.
- To access secrets, see Accessing Secrets.
- To add a new secret, see Add New Secret.
- For secret types, see Secret Type.

Secure Data at Rest

You are solely responsible for implementing encryption for data at rest on systems hosting Cribl products. To secure data at rest, consider:

- **Filesystem-level encryption**: Encrypting the entire file system where Cribl data resides.
- **Hardware-level encryption**: Using self-encrypting drives (SEDs) or other hardware-based encryption mechanisms.

Secure Data in Motion

For an extra layer of protection, consider encrypting your data while it travels between Cribl Stream and your data Sources and Destinations. For details, see Data Encryption.

Currently, Cribl Stream supports decryption only when Splunk is the end system. For details, see Data Decryption.

For a comprehensive solution, we recommend configuring TLS. TLS encrypts the entire data stream, including all event fields, regardless of Source or Destination type. This provides a more robust security posture for your data in motion. For details, see Secure Cribl Stream Sources and Destinations with Certificates.

Disable the Exec Source

The Exec Source, while powerful, can pose security risks if not configured and used carefully. To mitigate these risks, you can disable the Exec source by setting the `CRIBL_NOEXEC` environment variable. This prevents the execution of arbitrary commands, reducing the attack surface. For details, see Disable the Exec Source.

Filter Sensitive Information in API Responses

Use the `sensitiveFields` configuration property to filter sensitive information in the responses to API requests to `/inputs and /outputs endpoints`. Add an array that lists the attributes you want to filter under `sensitiveFields` in the `api` section of your `cribl.yml` configuration file, as shown in this example:

Example sensitiveFields array in cribl.yml

```
api:
  host: 0.0.0.0
  port: 9000
  sensitiveFields:
    - awsApiKey
    - awsSecret
    - token
    - password
  retryCount: 120
  ...
```

Copy

In the responses for `/inputs` and `/outputs` endpoints, Cribl returns empty values (`''`) for the attributes you list in the `sensitiveFields` array. In this example, Cribl will return empty values for the attributes `awsApiKey`, `awsSecret`, `token`, and `password`.

Use caution: the `sensitiveFields` configuration property affects UI workflows and applications that rely on the responses to `/inputs` and `/outputs` endpoints. The responses will contain empty values (`''`) instead of plaintext values for the attributes that you list in the `sensitiveFields` array. When

> you change or update a Source or Destination, you must also re-enter the correct value for any attribute that is listed in the `sensitiveFields` array.

Refer to the API reference for more information about the `/inputs` and `/outputs` endpoints and the attributes that appear in their responses.

4.3. Secure Leader/Worker Nodes Communication

You can configure secure communication between your Leader and Worker Nodes. As an additional security measure, we recommend you prevent direct browser access to the UI for all Worker Nodes.

Disable Exposing the API Service for Worker Nodes

Cribl recommends that in enterprise distributed deployments, you disable exposing the API service to the network on the configured API port for all Worker Nodes.

1. Select a Worker Group.
2. Open **Group Settings** (top right).
3. Under **General Settings** > **API Server Settings**, select **Advanced**.
4. Toggle **Listen on port** off.
5. Select **Save**.

Secure Browser Access to Worker Node

Cribl recommends that in enterprise distributed deployments, you prevent direct browser access to the UI for all Worker Nodes.

On the Leader Node, start by enabling UI access to Worker Nodes (Stream, Edge). This way, admins will still be able to tunnel through from the Leader to the UI on any Worker Node. This is also a prerequisite for Connect to the Leader Securely described below.

1. Select a Worker Group.
2. Open **Group Settings** (top right).
3. Under **General Settings** > **API Server Settings**, select **Advanced**.
4. Toggle **Local UI access** off.
5. Select **Save**.

Connect to the Leader Securely

For Cribl.Cloud Leader deployments, TLS is enabled by default, so you don't need to configure it manually – the Cribl.Cloud platform handles this automatically. The TLS configurations described below are only for on-prem and self-managed deployments.

You can configure secure communication between your Leader and Worker Nodes using the UI, the `instance.yml` config file, or environment variables.

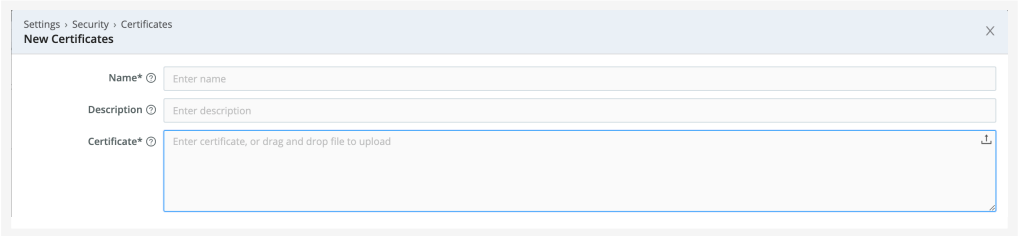
Secure via the UI

To set up secure communication via the UI, you configure first the Worker Group, then the Worker Node, then the Leader.

Worker Group Setup

For each Worker Group whose Worker Nodes you want to secure:

1. Open your TLS/SSL certificate file, and copy its contents to your clipboard. (This can be the same certificate you uploaded to the Leader, or a different certificate.)
2. In the sidebar, select **Worker Groups**, and then select the Worker Group you want to configure.
3. Select **Group Settings** or **Fleet Settings** (upper right).
4. Select **Security** > **Certificates** > **TLS**.
5. Click **Add Certificate**.
6. In the **New Certificates** modal, add the certificate's contents to the **Certificate** field.
   (You can paste the file's contents, or you can drag/drop or upload the `.pem` file.)
7. As you did on the Leader, also insert your **Private key**, and (as needed) your **Passphrase** and **CA certificate**.
8. Click **Save**.
9. Commit and Deploy the Worker Group's new configuration, including the new certificate.
10. Repeat the preceding steps on each Worker Group.



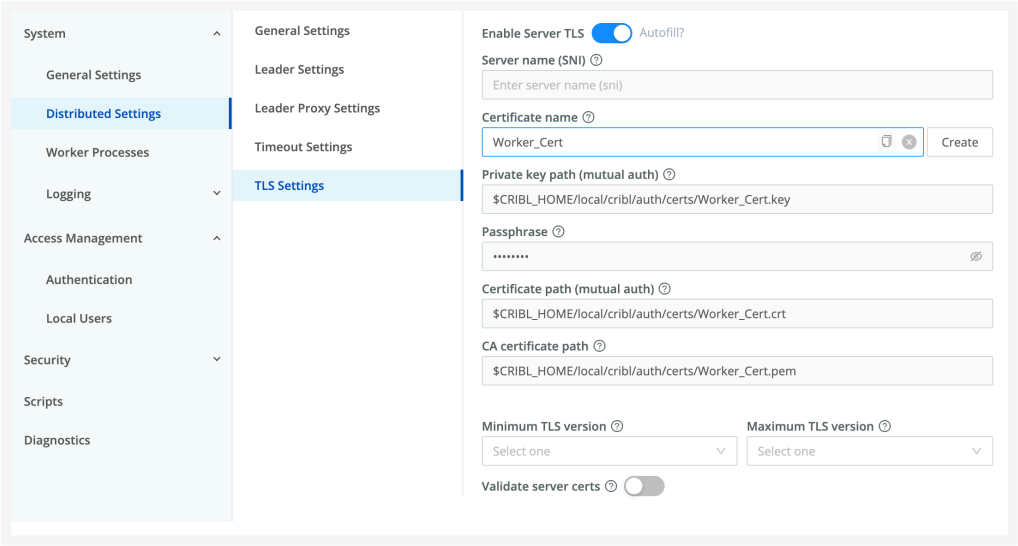
Worker Group-level certificates are configured like the Leader's certificate

Worker Node Setup

For each Worker Node that you want to secure:

1. Enable the Leader's UI access to each desired Worker Node (Stream, Edge).
2. Tunnel through from the Leader to the Worker Node's UI.

3. Navigate to this Worker Node's **Worker Settings** (upper right) > **System** > **Distributed Settings** > **TLS Settings**.
4. Toggle **Enable Server TLS** on.
   This will expose the remaining TLS settings.
5. From the **Certificate name** drop-down, select the certificate you uploaded to the parent Worker Group.
   This will prefill all the required fields. (See all deployed certificates at the left nav's **Security** > **Certificates** link.)
6. Click **Save**.
   The Worker will be unavailable during a short lag, while it restarts with the new configuration.
7. Repeat the preceding steps on each Worker Node.



Configuring TLS on Worker Node's UI, from the Leader

Leader Setup

Next, return to the Leader's UI:

1. Select **Settings** > **Global** > **System** > **Distributed Settings** > **TLS Settings**.
2. Toggle **Enable server TLS** on.
3. In the **Certificate name** drop-down, select an existing certificate. This will auto-populate the corresponding certificate fields.
4. Click **Save**.

After you've enabled TLS on the Leader, generating bootstrap scripts to add or update Worker Nodes will automatically prepend `https://` to the Leader's URL.

Secure via the YAML Config File

You can also configure the Leader's `$CRIBL_HOME/local/_system/instance.yml` file to ensure that TLS is enabled. Here's the relevant section:

```
distributed:
  mode: managed-edge
  master:
    host: <hostname>
    port: 4200
    authToken: <token>
    tls:
      disabled: false
      rejectUnauthorized: false
      requestCert: false
    resiliency: none
  group: default_fleet
```

Copy

After you've enabled TLS on the Leader, generating bootstrap scripts to add or update Worker Nodes will automatically prepend `https://` to the Leader's URL.

Secure via Environment Variables

Another way to set up secure communications between Worker Nodes and the Leader is via environment variables (Stream, Edge).

If you deploy your Worker Nodes in a container, you can enable encrypted TLS communications with the Leader by configuring the `CRIBL_DIST_LEADER_URL` with the `tls:` protocol. This will override the default setting in `instance.yml`. Here's the format:

```
CRIBL_DIST_LEADER_URL=tls://<authToken>@leader:4200
```

Secure the Leader Node

This is a best practice that enables the Leader to validate itself to clients. We can secure it using the self-signed certificate we created in Configure SSL Certificates:

1. Navigate to **Settings** > **Global** > **General Settings** > **API Server Settings** > **TLS**.
2. Toggle **Enabled** off.

3. From the **Certificate Name** drop-down, select a cert you've previously imported. This will populate the corresponding fields here.
4. Click **Save**.

After this save, you must prepend `https://` to all Cribl Stream URLs on the Leader Node. E.g., to get back to the Settings page you just configured, you'll now need to use `https://<hostname>:<port>/settings/system)`.

4.3.1. How To Secure The Auth Token For The Leader Node

For new installations, the Leader Node generates its own secure, random auth token. The Leader also creates a new token when you switch from Single-instance to the Distributed mode by configuring a Node to be a Leader Node.

In versions older than 4.5.0, the default token was `criblmaster`. This default auth token does not change automatically if you upgrade your deployment to version 4.5.0 or newer.

If you need to change your auth token (for example, you prefer to use your own token or you are still using the older default of `criblmaster`), change the token value to a unique secure value.

This applies whether you configure your Leader via the UI, or via a Docker Compose file.

High Availability (Failover) Mode

Beginning in Cribl Stream 4.7, the auth token information that is stored in the `leader.yml` file on the failover volume will take precedence over the auth token stored in the local `instance.yml` file. This way, a single token is shared between primary and any standby Leaders.

In versions older than 4.7, there is no `leader.yml` file; instead, the auth token is in the `instance.yml` file only. Whenever you change an auth token, you must update it to be identical on all Leaders. See Configuring Additional Leader Nodes in the failover topic.

Change the Auth Token Value

Changing the auth token once you have Worker Nodes deployed will break communication between the Nodes and the Leader. If you do change the auth token value, make sure that you update the auth tokens for the Leader and the Worker Nodes to the same value.

Create a unique token value with strong security. We recommend creating a string that contains at least 14 characters and includes uppercase letters, lowercase letters, and numbers.

Allowed Characters in the Leader Auth Token

Starting in version 4.6.1, these are the only allowed characters you can use in the auth token for your Leader:

- a-z: Lowercase letters from a to z.
- A-Z: Uppercase letters from A to Z.

- 0-9: Digits from 0 to 9.
- `_`: Underscore.
- `!`: Exclamation mark.
- `-`: Hyphen (dash).

In version 4.5.1 and newer, you *can't* use these special characters in the Leader auth token:

- `<`: Open angle bracket.
- `>`: Close angle bracket.
- `"`: Double quotation mark.
- `{`: Open brace.
- `}`: Close brace.
- `|`: Pipe.
- `\`: Backslash.
- `^`: Caret.
- `'`: Single quotation mark.
- `` ` ``: Backtick.
- Whitespace

One option is to generate a token value with a shell command. For example, on Linux or Mac OS, the following command generates a 32-character value:

```
head /dev/urandom | LC_ALL=C tr -dc A-Za-z0-9 | head -c32 | cut -c 1-
```

Copy

Use the CLI, UI, or environment variables to change the auth token. Manually editing the `instance.yml` file to change the auth token, although technically possible, will leave the token in plain text and prevent it from being properly encrypted. Using the following methods ensures your token is securely encrypted.

Once you have created a secure auth token value, you can change it:

- In the **Auth token** field in the UI.
- In the appropriate place in your Docker Compose file (if applicable).

- For the Leader via command line: `./cribl mode-master -u <token>`. If your Cribl Stream deployment uses High Availability (HA), you must use the command line (not the UI) with the command listed above.
- For the Worker Nodes via command line:
  - `./cribl mode-managed-edge -u <token>` for Cribl Edge.
  - `./cribl mode-worker -u <token>` for Cribl Stream.

Running either of the commands resets all the related options. This means that, in addition to the token, you must provide the command with the current values of all the other arguments.

You can find the **Auth token** setting by navigating to **Settings** > **Global** > **System** > **Distributed Settings** > **Leader Settings**.

4.3.2. TLS Defaults And System-Wide Settings

Cribl Stream prioritizes secure communication by default. This ensures the integrity and confidentiality of data transmitted between your Cribl Stream instance and various Sources and Destinations.

TLS Version Support and Defaults

The minimum supported TLS version in Cribl Stream is `TLS 1.2` by default. This default applies to all secure connections, including supported Sources and Destinations, Worker/Leader communications, and Cribl.Cloud.

Cribl.Cloud and on-prem deployments use the same TLS settings, with `TLS 1.2` being the default minimum version.

You can check whether a Source or Destination supports TLS by reviewing the information box at the beginning of its corresponding documentation page.

TLS Settings and Traffic Types

This table shows TLS client/server pairs, and encryption defaults, per traffic type.

Traffic Type	TLS Client	TLS Server	Encryption	Cert Auth	CN* Check
UI	Browser	Cribl Stream	Default disabled	Default disabled	Default disabled
API	Worker/Edge Node	Leader	Default disabled	Default disabled	Default disabled
Worker-to-Leader	Worker/Edge Node	Leader	Default disabled	Default disabled	Default disabled
Data	Any data sender	Cribl Stream (Source)	Default disabled	Default disabled	Default disabled
Data	Cribl Stream (Destination)	Any data receiver	Default disabled	Default disabled	Default disabled
Authentication	——–	——–	——–	——–	——–
Local*	Browser	Cribl Stream	Default Disabled	N/A	N/A

Traffic Type	TLS Client	TLS Server	Encryption	Cert Auth	CN* Check
LDAP*	Cribl Stream	LDAP Provider	Custom	N/A	Default Disabled
Splunk*	Cribl Stream	Splunk Search Head	Default Enabled	N/A	Default Disabled
OIDC†/Okta*	Browser and Cribl Stream	Okta	Default Enabled	N/A	Enabled (Browser)
OIDC†/Google*	Browser and Cribl Stream	Google	Default Enabled	N/A	Enabled (Browser)

** Common name*
† OpenID Connect

System-wide TLS Settings Including Ciphers

You can configure advanced, system-wide TLS settings - minimum and maximum TLS versions, default cipher lists, and ECDH curve names. Select **Settings** > **Global** > **System > General Settings > Default TLS Settings**.

In the **Default cipher list** field, you can specify one or more ciphers from the following list:

- `ECDHE-RSA-AES128-GCM-SHA256`
- `ECDHE-ECDSA-AES128-GCM-SHA256`
- `ECDHE-RSA-AES256-GCM-SHA384`
- `ECDHE-ECDSA-AES256-GCM-SHA384`
- `DHE-RSA-AES128-GCM-SHA256`
- `ECDHE-RSA-AES128-SHA256`
- `DHE-RSA-AES128-SHA256`
- `ECDHE-RSA-AES256-SHA384`
- `DHE-RSA-AES256-SHA384`
- `ECDHE-RSA-AES256-SHA256`
- `DHE-RSA-AES256-SHA256`
- `HIGH`
- `!aNULL`
- `!eNULL`
- `!EXPORT`

- `!DES`
- `!RC4`
- `!MD5`
- `!PSK`
- `!SRP`
- `!CAMELLIA`

CA Certificates and Environment Variables

For any Cribl Stream Source or Destination that supports TLS, you can configure a **CA Certificate Path** field that points to a Certificate Authority (CA) `.pem` file(s). However, you can also use environment variables to manage CAs globally. Here are some common scenarios:

- **Add a set of trusted root CAs to the list of trusted CAs that Cribl Stream trusts**. Set the `NODE_EXTRA_CA_CERTS` environment variable for each Worker Node. For example, if you are using systemd, add the following line in each Worker Node's systemd unit file (replace `/<path>/<to>/<the>/<directory>/<containing>/<certs>/ca.pem` with the path to your CA `.pem` file):

  ```
  ...
  [Service]
  Environment="NODE_EXTRA_CA_CERTS=/<path>/<to>/<the>/<directory>/<containing>/<ce
  ...
  ```

  Copy

  When configuring TLS authentication on Worker Nodes, make sure you place your certificates into a separate directory outside of $CRIBL_HOME. If you place the certificates inside $CRIBL_HOME, they'll be removed when the next config bundle is deployed from the Leader.

  For details about `NODE_EXTRA_CA_CERTS`, see the node.js documentation.

- **Configure Cribl Stream to accept all TLS certificates, regardless of their validity**. Set the `NODE_TLS_REJECT_UNAUTHORIZED` environment variable for each Worker Node. For example, if you are using systemd, add the following line in each Worker Node's systemd unit file:

```
...
[Service]
Environment="NODE_TLS_REJECT_UNAUTHORIZED=0"
...
```

Copy

`NODE_TLS_REJECT_UNAUTHORIZED=0` disables TLS certificate validation, which can decrease the security posture of your Cribl installation. For this reason, we recommend avoiding `NODE_TLS_REJECT_UNAUTHORIZED=0` in production environments. Instead, use the `NODE_EXTRA_CA_CERTS` environment variable to explicity trust the necessary certificates.

For details about `NODE_TLS_REJECT_UNAUTHORIZED`, see the node.js documentation.

As a result of a Known Issue, passphrases in TLS certificates configured for Sources or Destinations in a Cribl Edge Fleet will not be decrypted correctly when a Subfleet inherits the configuration.

Avoid using passphrases with TLS certificates if you intend to have Subfleets inherit Source or Destination configurations.

Global CA Certificates and Sources

While Cribl Stream supports using environment variables like `NODE_EXTRA_CA_CERTS` to add trusted root CAs globally, this approach is not applicable to all Sources. Some Sources, such as the Windows Events Fowarder (WEF), require a specific CA certificate configured at the Source level to function correctly.

Additionally, OS certificate stores accessed through Node.js boot flags (such as `--use-bundled-ca`, `--use-`

4.3.3. Securing Cribl.Cloud With TLS And Mutual TLS

Secure data transmission is essential for protecting sensitive information as it travels across networks. Cribl.Cloud offers two primary methods for securing data in transit:

- **Transport Layer Security (TLS)** - Encrypts data during transmission and authenticates the server.
- **Mutual TLS (mTLS)** - Adds client authentication on top of standard TLS.

For encrypting data in transit and simple server verification, Standard TLS is sufficient. Opt for mTLS when security policies mandate additional client verification.

This guide explains both security methods and walks you through configuration options in Cribl.Cloud.

TLS in Cribl.Cloud

In Cribl.Cloud, you can enable TLS encryption and use Cribl-provided certificates for certain Sources to ensure that your data is protected in transit.

Cribl-provided certificates offer the following benefits:

- **Reliable, ready-to-use security**: When you enable TLS and enter the correct environment variables for Cribl-provided certificates in the Source configuration, data in transit is always encrypted.
- **Zero maintenance**: Cribl-provided certificates are automatically managed and renewed.
- **Simplified deployment**: No need to generate or install custom certificates.

You only need to use your own private key or configure mTLS if specifically required by your security team or company policies. If you need custom certificates, see Certificate-Based Authentication.

While TLS encryption ends at individual Nodes upon data arrival in Cribl.Cloud, data at rest is protected by other robust security measures within the platform.

Cribl-provided certificates are unique to each Workspace and Organization.

Cribl.Cloud supports Cribl-provided certificates for Sources including Cribl TCP, Cribl HTTP, Syslog, HTTP, and Splunk TCP. To view a complete list of the supported Sources, along with their corresponding ports and ingress addresses:

1. Log in to your Cribl.Cloud portal.
2. On the top bar, select **Products**, and then select **Workspace** under **Cribl**.
3. In the sidebar, select **Data Sources**.

Enable TLS on Sources Using Cribl-Provided Certificates

To enable TLS using Cribl-provided certificates for a supported Source, configure the following **TLS Settings** for the Source:

- **Enabled**: Toggle on.
- **Private key path**: Enter `$CRIBL_CLOUD_KEY`. This environment variable represents the path to the default private key in Cribl.Cloud.
- **Certificate path**: Enter `$CRIBL_CLOUD_CRT`. This environment variable represents the path to the default certificate in Cribl.Cloud.
- **Minimum TLS version**: Select `TLSv1.2`.

When you finish configuring the Source, select **Save**, then **Commit** and **Deploy**.

For details on how to securely forward syslog data to your Cribl.Cloud instance, see Syslog TLS to Cribl.Cloud (Palo Alto Example).This topic demonstrates a vendor-specific configuration (Palo Alto) for secure data ingestion.

Enable mTLS Authentication on Cribl.Cloud

When standard TLS isn't sufficient and you need client authentication, you can enable mTLS.

Prerequisites for mTLS

mTLS requires:

- A CA certificate chain that can validate the client certificates.
- Client certificates issued to each client that will connect to Cribl.Cloud.
- Client systems configured to present their certificates during connection.

In Cribl.Cloud, you configure mTLS authentication separately for each Source.

This requires a CA certificate chain that can validate the client certificate used for authentication. For secure communication using mTLS authentication in Cribl.Cloud, you need to provide the root CA certificate of your client certificate. If you don't have the root CA certificate, you can also provide a CA certificate chain that includes all intermediate certificates leading back to a trusted root certificate.

Configure mTLS on a Source

You can add your CA certificate by creating a new certificate entry in Cribl.Cloud.

1. Prepare the CA certificate chain PEM file.
2. Go to the Worker Node: **Settings** > **Global** > **Security** > **Certificates** and select **Add Certificate**.

3. Populate the **Certificate** field with a valid PEM-formatted content (from your trusted CA certificate chain):

```
-----BEGIN CERTIFICATE-----
CERTIFICATE CONTENT
-----END CERTIFICATE-----
```

Copy

4. Populate the **Private key** with the key in PEM format:

```
-----BEGIN RSA PRIVATE KEY-----
HIDDEN PRIVATE KEY
-----END RSA PRIVATE KEY-----
```

Copy

5. In the **CA certificate** field, enter your PEM-formatted certificate and save. This will generate the CA certificate path.

While the **Add Certificate** form requires you to provide a Private Key and Certificate, these fields are not necessary for client certificate validation. This is a design limitation that requires you to provide some properly formatted values to proceed. You can use a self-signed certificate for this purpose.

Apply the CA Certificate to Your Source:

1. Edit the certificate again to view the certificate path and copy it.
2. Navigate to the Source where you want to enable mTLS and open the configuration modal. In the **TLS Settings** tab:
    1. Toggle **Enable TLS** to `Yes`.
    2. Set **Authenticate Client** to `Yes`.
    3. Paste the path in the **CA certificate path** field.
    4. Select **Save**, then **Commit** & **Deploy**.

4.3.4. Configure MTLS Authentication On-Prem

Once you have configured the Leader for encrypted TLS communication, you can implement client certificate exchange to enable mutual authentication (mTLS). This allows Cribl Stream to permit only explicitly authorized clients, which hold valid certificates, to connect to the Leader.

When a client certificate is presented to a Leader, two things happen:

1. The Leader validates the client certificate presented to Cribl Stream. The `Validate Client Certs` setting is optional, but highly recommended. When enabled, the Leader checks the certificate against the trust store, to see if it has been signed by a valid certificate authority (CA).

The Leader checks the list of certificates in the `CA Certificates Path` box first (if populated), then against the list of built-in system certificates.

2. The Leader checks whether the `Common Name` (CN) matches the regular expression in the configuration. Cribl Stream's default is to accept any value in the `Common Name` field. You can customize this as needed.

Within the `Common Name`, we validate against the value after the `CN=string`. If your `Common Name` is `CN=logstream.worker`, you would enter `logstream\.worker` in the `Common Name` field - including the backslash, because the value entered is a regular expression.

The Cribl.Cloud Leader connections on port `4200/tcp` do not currently support mutual TLS (mTLS).

Certificate Requirements for mTLS

mTLS provides a significantly higher security posture by requiring both the server and the client to prove their identities. For this handshake to succeed, Cribl Stream relies on the underlying TLS stack to validate the integrity and purpose of each certificate.

Before enabling mTLS, verify that your certificate infrastructure meets the standards outlined below. If a certificate is expired, uses a legacy algorithm, or cannot be traced back to a trusted root, the TLS connection will be rejected immediately to prevent unauthorized access.

General Certificate Standards

Regardless of their role in the handshake, all certificates used by the Leader, Worker/Edge Nodes, and Sources/Destinations must adhere to these baseline criteria:

- **Validity Period**: Certificates must be current. TLS handshakes will fail if the system time is outside the NotBefore or NotAfter window.

- **Modern Cryptography**: Use certificates signed with modern algorithms (such as `RSA` or `ECDSA`). Avoid legacy or deprecated hash functions like `MD4`, `MD5`, or `SHA-1`, which are often blocked by TLS.
- **Chain of Trust**: The peer validating the certificate must be able to build a complete path from the end-entity certificate through any Intermediate CAs to a trusted root. In Cribl Stream, this trust is established via the **CA Certificates Path** configuration or the host system trust store.

Server-Specific Requirements

Server certificates are presented by components that terminate TLS (such as the Leader on its management port). These certificates must prove they are authorized to act as a "server" to the connecting client:

- **Key usage**: The certificate Extended Key Usage (EKU) must include `serverAuth`.
- **Hostname Verification**: To prevent man-in-the-middle attacks, the Common Name (CN) or Subject Alternative Name (SAN) must exactly match the `hostname` or IP address that clients use to connect (for example, `leader.cribl.com`).

Client-Specific Requirements

Client certificates are presented by the component initiating the connection (such as a Worker/Edge Node connecting to a Leader). These are subject to stricter identity checks:

- **Client Authentication**: The certificate EKU must explicitly include `clientAuth`. Public CAs (such as Let's Encrypt) have increasingly disallowed this attribute in their standard issuance policies to comply with stricter industry root program requirements. For mTLS, you will likely need to use an internal PKI or a specialized certificate product that permits client-side usage.
- **Identity Constraints**: Client certificates must be leaf certificates (where **Basic Constraints** are set to `CA=false`). Root or Intermediate CA certificates should never be used as client identifiers.
- **CN Regex Matching**: Cribl Stream provides an additional layer of security by checking the certificate CN against a user-defined regular expression to ensure only specific named clients are permitted.

Limitations on mTLS Authentication

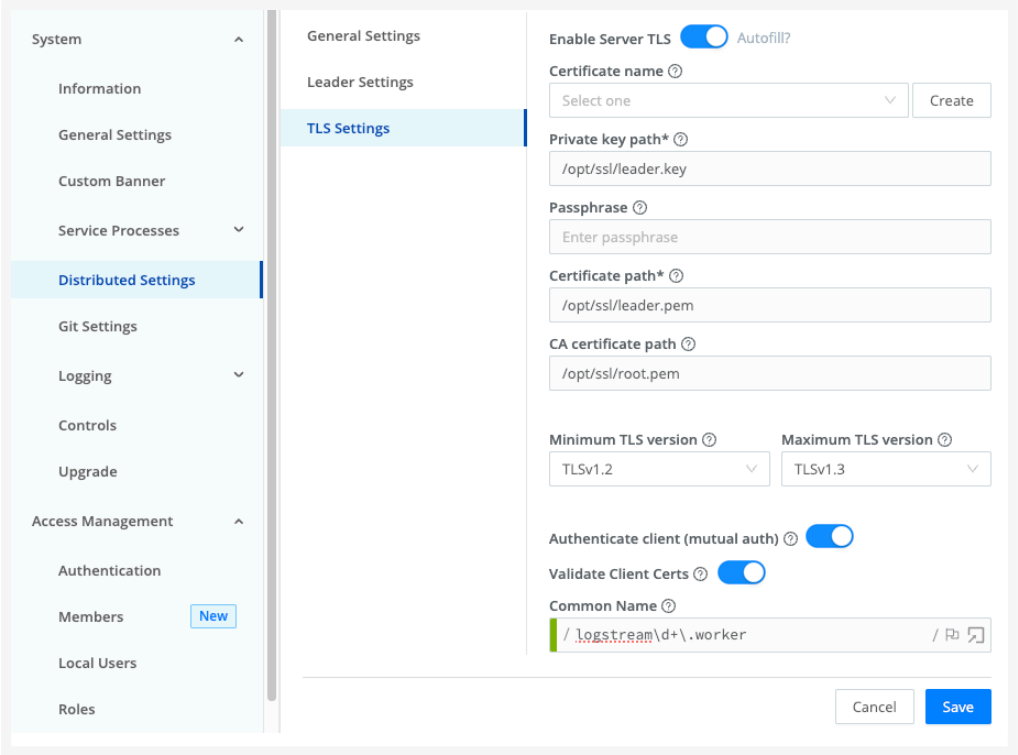
When configuring mTLS authentication on Worker Nodes, make sure you place your certificates into a separate directory outside of `$CRIBL_HOME`. If you place the certificates inside `$CRIBL_HOME`, they'll be removed when the next config bundle is deployed from the Leader.

Similarly, you can't bootstrap Worker Nodes with mutual authentication already populated. To bootstrap Worker Nodes, you supply only the shared authentication token. Certificates should be viewed as two-factor authentication; so placing the certificates in the config bundle defeats the purpose of two-factor authentication.

Configure via the UI

mTLS authentication in Cribl.Cloud is configured separately for each Source.

Below is a sample configuration on the Leader:



Configuring mutual authentication

In the above configuration, note:

- You can set both the **Minimum TLS version** and **Maximum TLS version**.
- The `Common Name` regex contains an additional check for `\d+`. This allows checking for the format: `logstream1.worker`, `logstream2.worker`, `logstream3.worker`, etc.

Using YAML Config File

To set up mTLS authentication via the Worker Node's `instance.yml` config file, you need to change some values and also add a few keys, as shown in this example:

```
distributed:
  mode: managed-edge
  master:
    host: <hostname>
    port: 4200
    authToken: <token>
    tls:
      disabled: false
      rejectUnauthorized: true # false if ignoring untrusted certs
      requestCert: true
      privKeyPath: /path/to/certs/worker.key
      certPath: /path/to/certs/worker.pem
      caPath: /path/to/certs/root.pem
    resiliency: none
  group: default_fleet
```

Copy

Using Environment Variables

You can set up mTLS authentication by configuring this environment variable, using the format shown in this example:

```
CRIBL_DIST_LEADER_URL="tls://<authToken>@leader.cribl:4200?tls.privKeyPath=/path/to/c
```

Copy

Once you've set this variable, restart the Worker Node. You should see the Worker Node successfully reconnect to the Leader.

If the Worker Node doesn't connect, check `cribl.log` on both the Worker Node and Leader for more context

4.3.5. Secure Leader Using An Existing TLS/SSL Certificate And Key

This page outlines how to protect the Leader to Worker Nodes communications, using an existing TLS/SSL certificate and key.

Cribl Stream expects certificates and keys to be formatted in privacy-enhanced mail (`.pem`) format. To generate a self-signed certificate and corresponding key, see Securing Cribl Stream.

Import Certificate and Key

To use your certificate and key to prepare secure communications between Workers/Edge Nodes and the Leader:

1. Navigate to the Leader's **Settings** > **Globals** > **Security** > **Certificates** > **New Certificates** modal.
2. Open your TLS/SSL certificate file. (The self-signed certificate example in Configure TLS for API and UI Access used the placeholder name `myCert.pem`.)
3. Copy the file's contents to your clipboard.
4. Paste the file's contents into the modal's **Certificate** field.

> You can skip the preceding three steps: Just drag/drop your `.pem` file from your filesystem into the **Certificate** field, or upload it using the button at the field's upper right.

5. Open your private key file. (The self-signed certificate example in Configure TLS for API and UI Access used the placeholder name `myKey.pem`.)
6. Copy the file's contents to your clipboard.
7. Paste the clipboard contents into the same Leader modal's **Private key** field.

> Here again, you can skip the preceding three steps by dragging/dropping or uploading the `.pem` file from your filesystem.

8. Fill in the **Name** and **Description** fields.
9. If you've uploaded a self-signed certificate, just **Save** it now.
10. If your private key is encrypted, fill in the modal's **Passphrase** with the corresponding key. (You can paste the key's contents, or you can drag/drop or upload the key file.)

11. If you're uploading a certificate signed by an external certificate authority - For example, a downloaded Splunk Cloud certificate - import the chain into the **CA certificate** field before saving the certificate. For details, see Obtain the Certificate Chain (TLS/SSL).

4.3.6. Configure TLS For API And UI Access

To safeguard inbound communications to your Worker Nodes, configure Transport Layer Security (TLS) on the Leader. This will enable secure access to the Cribl Stream API and UI.

In Cribl Stream 4.1 and newer, for enhanced security, Cribl encrypts TLS certificate private keys in configuration files when you add or modify them. Before upgrading to v.4.1.0 (or newer), back up your existing unencrypted key files - see Safeguarding Unencrypted Private Keys for Rollback. If you need to roll back to a version older than 4.1, see Restore Unencrypted Private Keys below.

You can use your own certificates and private keys, or you can generate a pair with OpenSSL, as shown here:

```
openssl req -nodes -new -x509 -newkey rsa:2048 -keyout myKey.pem -out myCert.pem -days 420
```

This example command will generate both a self-signed cert `myCert.pem` (certified for 420 days), and an unencrypted, 2048-bit RSA private key `myKey.pem`. (Change the filename arguments to modify these placeholder names.)

As indicated by these examples, Cribl Stream expects certificates and keys to be formatted in privacy-enhanced mail (`.pem`) format.

In the Cribl Stream UI, you configure the certificate and its associated private key differently depending on your deployment type:

- For Single-instance deployments, navigate to **Settings** > **Globals** > **Security** > **Certificates** (Edge, Stream).
- For Distributed deployments, navigate to a Worker Group and select **Worker Group Settings** > **Security** > **Certificates** (Edge, Stream).

Alternatively, in the `local/cribl.yml` file, you can edit the `api` section to directly set the `privKeyPath` and `certPath` attributes. For example:

cribl.yml

```
api:
  host: 0.0.0.0
  port: 9000
  disabled : false
  ssl:
    disabled: false
    privKeyPath: /path/to/myKey.pem
    certPath: /path/to/myCert.pem
...
```

Copy

See Secure Leader/Worker Nodes Communication for details about using this certificate and key to secure communications on, and among, your Leader and Worker Nodes.

Restore Unencrypted Private Keys

If you need to roll back from Cribl Stream 4.1 or later to an earlier version, follow this procedure to restore the unencrypted private key files that you backed up earlier.

1. Stop Cribl Stream 4.1 (or later) on hosts - the Leader and all Worker Nodes.
2. Untar the older Cribl Stream version (v.4.0.4, for example) onto the Leader and all Worker Nodes.
3. Manually edit each Cribl Stream config file that contains an encrypted private key. Replace each key with its prior unencrypted version.

4.3.7. Secure Sources And Destinations With Certificates

You can secure connections between Cribl Stream and external systems by using TLS (Transport Layer Security) certificates or custom HTTP headers. These extra security measures are optional, but TLS certificates or custom HTTP headers help authenticate data senders and receivers, protect data in transit, and prevent unauthorized access.

Certificate-Based Authentication

Cribl Stream stores and manages certificates at the Worker Group level. When you add a certificate to a Source or Destination, Cribl Stream creates it within the same Worker Group where you configured the integration.

For information about how certificate references work in Packs, see Use Named TLS Certificates.

Add a TLS Certificate to a Worker Group

To add a TLS certificate that will be available to all Sources and Destinations at the Worker Group level:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Group Settings**. On the side menu, expand **Security** and select **Certificates**.
3. Select **Add Certificate**. In the **New Certificate** modal:
    - **Name:** Enter a unique name to identify the certificate.
    - **Description:** Optionally, add a short description of the certificate's scope or purpose.
    - **Certificate:** Upload or paste the full contents of your certificate file. Ensure you paste the entire block, including the `BEGIN` and `END` lines:

      ```
      -----BEGIN CERTIFICATE-----
      ... (certificate data) ...
      -----END CERTIFICATE-----
      ```

      Copy

    - **Private key:** Upload or paste the full contents of your private key (`.pem`). Ensure you paste the entire block, including the `BEGIN` and `END` lines. Some private keys may use labels like `RSA PRIVATE KEY` or `EC PRIVATE KEY`. Be sure to include the correct `BEGIN` and `END` lines that match your key type.

- **Passphrase:** Optionally include the certificate's passphrase if it uses passphrase encryption.
- **CA certificate:** Optionally upload or paste a certificate signed by an external certificate authority by importing the chain. See Obtain the Certificate Chain (TLS/SSL) for more information.
- **Referenced:** This table shows which Sources and Destinations are using this certificate. If this table is empty, you haven't assigned this certificate to any components yet.

4. Select **Save**. The certificate now appears in the list of certificates for the Worker Group.

Add a TLS Certificate to a Source or Destination

To add a TLS certificate directly to an integration:

1. Open the configuration settings for your Source or Destination.
2. Select **TLS Settings (Client Side)** or **TLS Settings (Server Side)**, depending on the integration.
3. Toggle **Enabled** on.
4. Use the **Certificate name** drop-down to add a certificate:
   - **To add a new certificate:** Select the **Create** button next to the field. See Add a TLS Certificate to a Worker Group for field descriptions.
   - **To add an existing certificate:** Select the name of the certificate from the drop-down. This will auto-populate the related fields.
5. Select **Save**.

Maintain TLS Certificates

Certificate revocation can lead to unexpected service interruptions. To mitigate risks:

- Regularly review and renew certificates.
- Implement robust monitoring and alerting systems to detect potential issues.
- Ensure secure storage of certificates and private keys.
- In case of emergency, have a well-defined recovery plan to restore service quickly.

Custom HTTP Headers

You can also encode custom, security-related HTTP headers as needed. In a Distributed deployment, navigate to Worker Group Settings, then **API Server Settings** > **Advanced** > **HTTP Headers**. (In a Single-instance deployment, the **API Server Settings** are within **Global Settings** > **General Settings**.) Define HTTP headers as key-value pairs, selecting **Add Header** as needed.

Implement a Custom Content Security Policy (CSP)

Content Security Policy (CSP) helps protect Cribl Stream from Cross-Site Scripting (XSS) and other web injection attacks by controlling which resources the browser is allowed to load. You can implement CSP using the Custom HTTP Headers setting described above.

Basic Configuration

To configure a basic CSP for Cribl Stream, add the following header in your **HTTP Headers** settings:

Header Name: `Content-Security-Policy`

Header Value:

```
default-src 'self'; script-src 'self' 'unsafe-eval' 'sha256-ZaAN8+jAf3MdnSLRdmTuExUhA
```

Copy

Test your CSP policy thoroughly in a development environment before deploying to production and monitor the browser console for CSP violations after implementation.

SAML SSO Configuration

If your organization uses SAML Single Sign-On, you'll need a modified CSP policy to accommodate the dynamic JavaScript and form submissions required for SAML authentication:

Header Name: `Content-Security-Policy`

Header Value for SAML:

```
default-src 'self'; script-src 'self' 'unsafe-eval' 'unsafe-inline'; connect-src 'sel
```

Copy

Replace `YOUR_IDP_DOMAIN` with your actual Identity Provider's domain (for example, your Okta domain).

SAML configurations require additional permissions that reduce the restrictiveness of the CSP compared to the basic configuration.

4.3.8. Configure System Proxy

This document outlines how to configure Cribl Stream to use a proxy server for outbound HTTP/S requests.

There are two ways to define proxy variables in Cribl Stream:

- **Using environment variables**: This method is independent of the systemd service manager and can be used regardless of how Cribl Stream is installed or started.
- **Using systemd override files**: This method is specific to systemd and requires that Cribl Stream is started as a systemd service. Choose the method that best suits your deployment environment and preferences.

Prevent Proxy, Config, and Case Conflicts

Before you deploy Cribl Stream and configure proxies, Cribl recommends running the `printenv` shell command to check for any existing proxies in your environment. If you find existing proxies, be sure to include their environment variables in the systemd override file.

If Cribl Stream is already running on any Nodes you are proxying, restart the Cribl server after you initially configure - or make any changes to - the variables outlined on this page.

The environment variables' names can be either uppercase or lowercase. However, if you set duplicate versions of the same name, the lowercase version takes precedence. For example, if you've set both `HTTPS_PROXY` and `https_proxy`, the IP address specified in `https_proxy` will take effect.

Set Up Proxy Environment Variables (Using Environment Variables)

You can direct all outbound HTTP/S requests to go through proxy servers. You do so by setting a few environment variables before starting Cribl Stream, as follows:

Configure the `HTTP_PROXY` and `HTTPS_PROXY` environment variables, either with your proxy's IP address, or with an FQDN that resolves to that IP address.

Optionally, follow either convention with a colon and the port number to which you want to send queries. Some `HTTP_PROXY` examples:

```
$ export HTTP_PROXY=http://10.15.20.25:1234
$ export HTTP_PROXY=http://proxy.example.com:1234
```

Copy

`HTTPS_PROXY` examples:

```
$ export HTTPS_PROXY=http://10.15.20.25:5678
$ export HTTPS_PROXY=http://proxy.example.com:5678
```

Copy

In the above examples, note that an `HTTPS_PROXY` environment variable's referenced URL should generally be in `http://` format.

To set up Leader/Worker communication, use the CRIBL_DIST_WORKER_PROXY environment variable.

HTTP and/or HTTPS?

Several Cribl Stream endpoints rely on the HTTPS protocol. These include the Cribl telemetry endpoint (which must be accessed under some license types and all Cribl.Cloud plans), and the CDN used to propagate application updates and certain documentation features (API Reference and docs PDFs).

You might configure certain other Cribl Stream features (such as REST API Collectors) that require access to HTTP endpoints. For maximum flexibility, consider setting environment variables to handle both the HTTPS and HTTP protocols.

Set Up Proxy Configuration with systemd

If you are proxying outbound traffic and starting Cribl Stream using systemd, add your proxy environment variables to the systemd override file (see Persisting Overrides).

Ensure you explicitly include the protocol (`http://` or `https://`) in the variable values, as shown below:

Installed systemd File

```
[Service]
Environment=http_proxy=http://proxy.example.com:3128
Environment=https_proxy=http://proxy.example.com:3128
Environment=no_proxy=localhost,127.0.0.1,::1
```

Copy

This will prevent Cribl Stream from throwing "failed to send anonymized telemetry metadata" errors.

Using a Literal % Character

To use a literal `%` character, make sure to escape it by using a double `%%`.

For example, suppose that you want to persist the environment variable `CRIBL_DIST_WORKER_PROXY` in a systemd unit file, and the value includes a password that contains the special character `@`: `special@password`. You must encode the `@` as `%40`. However, because `%` is a special character that is used for specifiers in systemd unit files, you must also escape the `%` itself by doubling it.

In this example, `%%40` represents the literal `@` character in the password:

```
Environment=CRIBL_DIST_WORKER_PROXY=socks5://proxyuser:special%%40password@mydomain.c
```

Copy

Authenticating on Proxies

You can use HTTP Basic authentication on HTTP or HTTPS proxies. Specify the username and password in the proxy URL. For example:

```
$ export HTTP_PROXY=http://username:password@proxy.example.com:1234
$ export HTTPS_PROXY=http://username:password@proxy.example.com:5678
```

Copy

If your `username` or `password` contains special characters, Cribl Stream will try to use these fields as the proxy address. As a workaround, URL-encode these fields.

Bypassing Proxies with `NO_PROXY`

If you've set the above environment variables, you can negate them for specified (or all) hosts. Set the `NO_PROXY` environment variable to identify URLs that should bypass the proxy server, to instead be sent as direct requests. Use the following format:

```
$ export NO_PROXY="<list of hosts/domains>"
```

Cribl recommends including the Leader Node's host name in the `NO_PROXY` list.

Client Configuration and Leader Reference

The way a client references the Leader Node depends on its configuration, which originates from the Distributed Settings for that particular Node. These settings reside in the

`$CRIBL_HOME/local/_system/instance.yml` file.

When deploying a configuration update, the Leader Node informs all connected clients (Workers and Edge Nodes) within its Group or Fleet that a new configuration bundle is available. The Leader will reference itself using the same method (IP address or FQDN) that clients already use to connect to it in the distributed environment. This ensures a seamless download process for the clients, regardless of their initial connection method.

If a Worker is configured to use the Leader's IP for communication, the URL will also contain that specific IP. Consequently, the `NO_PROXY` variable on the client must match the IP address used in the client configuration and Leader URL for optimal operation.

Cribl v.4.6.0 introduces direct S3 downloads for configuration bundles, bypassing the Leader Node and scaling deployments with many Worker Nodes. However, if your S3 bucket is firewalled or requires access controls, a proxy might be needed. In such scenarios, you need to exclude the S3 bucket address from the "NO_PROXY" list for successful downloads.

`NO_PROXY` Usage

Within the `NO_PROXY` list, separate the host/domain names with commas or spaces.

Optionally, you can follow each host/domain entry with a port. If not specified, the protocol's default port is assumed.

To match subdomains, you must either list them all in full (for example, `NO_PROXY="foo.example.com,bar.example.com"`), or apply a wildcard by prefixing the domain name with a period or `*.`: `No_Proxy=".example.com"` or `No_Proxy="*.example.com"`.

To match the whole domain including its subdomains, add it both with and without wildcard to the list: `No_Proxy="example.com,.example.com"`.

To disable all proxies, use the `*` wildcard: `NO_PROXY="*"`. `NO_PROXY` with an empty list disables no proxies.

Cloud `NO_PROXY` Usage

You must include any cloud metadata endpoints (such as the AWS Instance Metadata Service) in the `NO_PROXY` list:

- AWS EC2 and Azure VM instances must include `169.254.169.254` in the list. If using IPv6 on AWS EC2, add `fd00:ec2::254` to the list.
- AWS ECS Fargate tasks must include `169.254.170.2`.
- GCP (Google Cloud Platform) VM instances must include `metadata.google.internal` and `169.254.169.254`.

Where Proxies Apply

Proxy configuration is relevant anywhere Cribl Stream initiates outbound HTTP or HTTPS connections to external services. This typically includes:

- Sources that pull data from remote HTTP/S endpoints (for example, REST APIs or cloud services).
- Destinations that send events to HTTP/S-based endpoints or services.
- Collectors that query external systems over HTTP/S.
- Notification Targets that call external webhooks or API endpoints.

Proxies are not used for Sources that listen for traffic (such as Syslog or TCP/UDP listeners), because they do not initiate outbound HTTP/S requests.

Support for proxies varies by component. For any specific Source, Destination, Collector, or Notification target, see its individual documentation page to confirm whether it supports proxies.

Testing Proxies

To initially test your proxy configuration, consider setting up a simple, free proxy server like mitmproxy (https://mitmproxy.org), and then monitoring traffic through that server. Verify that you can trace proxied requests from your Cribl Stream instance, and can validate that outgoing requests (to Destinations) are working properly.

Proxying Multiple Cribl Stream Instances in One Browser

Cribl Stream stores authentication tokens based on each http header's URI scheme, host, and port information. Within a given browser, Cribl Stream enforces a same-origin policy to isolate instances.

This means that if you want to run multiple proxied Cribl Stream instances in one browser session, you must assign them different URI schemes, hosts, and/or ports. Otherwise, logging into an extra Cribl Stream instance will expire the prior instance's session and log it out.

For example, assume that you've set up this pair of Apache proxy forward rules:

- https://web/cribla forwards to `cribl_hosta:8001/cribla`.
- https://web/criblb forwards to `cribl_hostb:8001/criblb`.

These two proxied addresses cannot be run simultaneously in the same browser session. However, this pair - which lead with separate URI schemes - could:

- https://web/cribla forwards to `cribl_hosta:8001/cribla`.
- https://web2/criblb forwards to `cribl_hostb:8001/criblb`.

Where separate instances **must** share URI formats, a workaround is to open the second instance in an incognito/private browsing window, or in a completely different browser.

4.3.9. Configure SOCKS Proxy

This document outlines how to configure Cribl Stream to use a SOCKS proxy server for outbound communication.

By using a SOCKS proxy, you can enhance security, bypass network restrictions, and route specific traffic through a controlled intermediary server.

The SOCKS proxy is exclusively for Worker Nodes to communicate with a Leader Node. The Leader Node doesn't use the proxy for its outbound communications.

Architectural Overview: SOCKS Proxy in Cribl Deployments

A SOCKS proxy acts as an intermediary for outbound network connections from a client application (in this case, your Worker Node). It allows you to centralize and control the egress points for your Cribl infrastructure. The specific architectural flow depends on whether your Cribl deployment is entirely on-prem or on Cribl.Cloud.

Management vs. Data Plane Communications

If you are using a proxy for Cribl software cluster communications, it must be a SOCKS proxy (Layer 4), not an HTTP proxy. This is because cluster communications use raw TCP connectivity on port 4200, which HTTP proxies cannot handle.

Communication Types Overview

Cribl products require distinct proxy types for different communication streams. Management communications between Leaders and Worker Nodes use a proprietary protocol over persistent TCP connections (port 4200), necessitating a **SOCKS (Layer 4) proxy** for relaying.

Configuration bundle downloads, however, use standard HTTP/HTTPS protocols. If you're using a proxy for these downloads, it needs to be an **HTTP proxy** (configured via `HTTP_PROXY`/`HTTPS_PROXY` environment variables) with access to the Leader Node's (ports 443/4200) or S3 endpoints. This differs from the SOCKS proxy needed for management communications. For details, see Manage Config Bundles.

Given these different proxy requirements, you'll often find your deployment needs both SOCKS and HTTP proxies. It really depends on your network environment and whether you choose to use proxies for specific communication types.

The table below summarizes the communication type and proxy support for each.

Communication Type	Protocol Requirements	Proxy Support	Port	Purpose
Management Plane	TCP (Layer 4)	SOCKS only HTTP proxies NOT supported	`4200`	Leader ↔ Worker Node control, configuration, and status
Config Bundle Downloads	HTTP/HTTPS (Layer 7)	SOCKS proxies NOT supported HTTP proxies only (`HTTP_PROXY`/`HTTPS_PROXY`)	`443`/`4200`	Worker Node downloading configuration bundles from Leader or S3
Data Plane	HTTP/HTTPS (Layer 7)	HTTP proxies (`HTTP_PROXY`/`HTTPS_PROXY`)	Various	Data ingestion, forwarding to destinations (S3, Splunk, etc.)

On-prem Deployment Architecture

In an on-prem deployment, all Cribl components - the Leader Node and the Worker Node - are deployed and managed within your organization's network. If you choose to use a SOCKS proxy, it will exclusively control the outbound communication from your Worker Nodes to their Leader Node.

Cribl.Cloud/Hybrid Deployment Architecture

In a Cribl.Cloud deployment, the Leader Node is hosted and managed by Cribl within the Cribl.Cloud infrastructure. Your Worker Nodes, however, are typically deployed within your own on-prem network or private cloud. A SOCKS proxy can be used to route the outbound communication from your customer-managed Worker Nodes to the Cribl-managed Leader in the cloud.

Configure SOCKS Proxy

SOCKS proxy settings are applied directly to the specific Cribl Nodes (Leader, Worker, or Edge) that need to use the proxy for their outbound communication.

There are two ways to define proxy variables in Cribl Stream.

- **Environment Variables**: Ideal for most scenarios, especially when automating deployments or managing a large number of Worker Nodes.
- **UI Configuration**: Suitable for smaller deployments or when manual configuration is preferred. However, be aware of the additional steps and potential complexities involved.

Configure via Environment Variables

This is the most straightforward and flexible method, ideal for automated deployments, containerized environments (Docker, Kubernetes), or managing a large number of Nodes.

For Worker/Edge Nodes Connecting to a Leader

Use the `CRIBL_DIST_WORKER_PROXY` on each Worker Node to configure its connection to the Leader Node.

```
CRIBL_DIST_WORKER_PROXY="socks5://<username>:<password>@<proxy_host>:<proxy_port>"
```

Copy

The default protocol is SOCKS5, which uses the format: `socks5://<username>:<password>@<host>:<port>`. Specify the SOCKS proxy host, port, and optional authentication credentials in the variable.

- `<username>:<password>`: Optional authentication credentials for your SOCKS proxy.
- `<proxy_host>`: The hostname or IP address of your SOCKS proxy server.
- `<proxy_port>`: The port on which your SOCKS proxy server listens.

You can alternatively specify SOCKS4 as the protocol with the format: `socks4://<proxyhost>:<port>.` To authenticate on a SOCKS4 proxy with username and password, use this format: `socks4://<username>:<password>@<proxyhost>:<port>`. The `<proxyhost>` can be a `hostname`, `IPv4`, or `IPv6`.

For Leader Node Connections (On-Prem Specific)

In on-prem deployments, if your Leader Node needs to route its own outbound connections (e.g., to external services like CDNs for software updates, licensing servers, or telemetry endpoints) through a web proxy, configure this using standard HTTP proxy settings. Note that Leader Nodes do not initiate connections to Worker Nodes - those connections are always initiated by the Worker Nodes to the Leader. SOCKS proxy configuration is not supported for Leader Node outbound connections to cloud services.

Where to set environment variables:

- **Systemd Service:** For Linux deployments, add `Environment="<VARIABLE_NAME\>=..."` under the `[Service]` section in your systemd override file (such as, `/etc/systemd/system/cribl.service.d/override.conf`).
- **Docker/Kubernetes:** Set as environment variables in your Dockerfile, `docker run` command, or Kubernetes Deployment/StatefulSet manifests.
- **Directly on Host:** For manual installations, you can set it in a shell script that starts Cribl, or in a profile file (for example, `.bashrc`, `.profile`) for the user running Cribl.

After setting the environment variable, restart the Cribl service on the Node for the changes to take

effect.

Config bundle downloads cannot be sent via a SOCKS proxy, but instead use an HTTP proxy that allows requests to the Leader or an S3 bucket.

Configure via the UI

Configuring a SOCKS proxy for Cribl Stream using the UI is more complex than using environment variables due to the multi-step process involved. It requires enabling direct access to the Leader, setting up the proxy on the Worker Nodes, and then configuring the SOCKS proxy settings. Additionally, using the UI might not be suitable for all deployment scenarios, especially those with large numbers of Worker Nodes or complex network configurations.

Configuring a SOCKS proxy for Cribl Stream using the UI is suitable for individual Worker Nodes or smaller deployments where direct access to each Worker Node's UI is feasible.

To configure a SOCKS proxy for Cribl Stream, follow these steps.

1. **Enable direct access**: Allow Worker Nodes to access the Leader Node directly.
    - Cribl Edge: Set up Leaders and Edge Nodes.
    - Cribl Stream: Set up Leaders and Worker Nodes.
    - Remember to enable UI Access (and teleport) to Workers or Edge Nodes as applicable.
2. **Set up Worker Nodes and SOCKS proxy**: Configure the Worker Nodes and ensure the SOCKS proxy server is running and accessible.
3. **Configure proxy details**: You can access the Worker Node via teleporting. To configure without teleporting, log in to each Worker Node directly using a local user account.
4. Select **Worker Node Settings** from the Worker Node submenu. Navigate to **Distributed Settings** > **Leader Proxy Settings** and configure the following fields:
    - **Username**: Enter the username for the SOCKS Proxy authentication.
    - **Password**: Enter the password for the SOCKS Proxy authentication.
    - **Proxy host**: Enter the proxy server host. Can be a `hostname`, `IPv4`, or `IPv6`.
    - **Proxy port**: The port on which your SOCKS proxy server listens.
    - **Protocol version**: Version of the SOCKS protocol. Defaults to `5` for `socks5`.
5. **Save the settings**: The Worker Nodes will automatically reconnect to the Leader node using the configured SOCKS proxy.
6. **Disable direct access**: Verify the SOCKS proxy configuration is verified. To ensure all traffic between Worker Nodes and Leader is routed through the SOCKS proxy, disable the direct network connections

between them. This includes removing any firewall rules or network configurations that allow direct communication.

Best Practices & Troubleshooting

For successful SOCKS proxy implementation, consider the following.

- **Firewall Rules**: Set up your firewall rules properly for a seamless deployment.
    - **Worker Node to SOCKS Proxy:** Allow outbound TCP from Cribl Worker Node to your SOCKS proxy on its configured `proxy_port`.
    - **SOCKS Proxy to Leader (TCP 4200)**: Your SOCKS proxy must have outbound access to the Leader Node on TCP port 4200. For on-Prem deployments, ensure your Leader Node accepts inbound connections from the SOCKS proxy on TCP 4200. For Cribl.Cloud deployments, allow outbound access from your SOCKS proxy to the Cribl.Cloud Leader's IPs for the Network Load Balancers (found in **Workspace** > **Access** > **Leader NLB IPs**). For details, see View Workspace Detail.
- **DNS Resolution**: The SOCKS proxy server must resolve the Leader Node's hostname (on-prem Leader or `your-workspace.cribl.cloud`).
- **Proxy Placement (On-Prem)**: Strategically place the SOCKS proxy to optimize network paths and minimize latency.
- **Egress Control (Cribl.Cloud)**: The SOCKS proxy provides a vital, controlled egress point for Cribl Nodes' communication with Cribl.Cloud, aligning with corporate security policies.
- **Authentication**: Securely manage and ensure correct credentials if your SOCKS proxy requires authentication.
- **Other Proxy Types**: SOCKS proxies (`CRIBL_DIST_WORKER_PROXY`) are for communication between the Leader and Worker Node. Other outbound traffic (for example, to external Sources/Destinations) may require standard HTTP/HTTPS proxies (`HTTP_PROXY`/`HTTPS_PROXY`). For details, see Configure System Proxy.

4.4. Manage Secrets And Keys

Encryption keys and secrets allow you to secure data inside events, as well as authorization tokens, passwords, API keys and others.

Source	Description
Create and Manage Encryption Keys	Create and Manage Encryption Keys
Create and Manage Secrets	Protecting Secrets with Centralized Store
Configure KMS	Configuring internal and external Key Management Services

4.4.1. Create And Manage Encryption Keys

You can create and manage keys that Cribl Stream will use for real-time encryption of fields and patterns within events. For details on applying the keys that you define here, see Encryption Keys.

After setting up a key, you can also reference it in Cribl Search, using the `encrypt` and `decrypt` functions in your queries.

Access Keys

- In a Single-instance deployment, select **Settings** > **Security** > **Encryption Keys**.
- In a Distributed deployment with one Worker Group, select **Settings** > **Security** > **Encryption Keys**.
- In a Distributed deployment with multiple Worker Groups, keys are managed per Worker Group. Select **Manage** > **Groups** > `<groupName>` **Group Settings** > **Security** > **Encryption Keys**.

On the resulting **Manage Encryption Keys** page, you can configure existing keys, and/or use the following options to add new keys.

Get Key Bundle

To import existing keys, click **Get Key Bundle**. You'll be prompted to supply a login and password to proceed.

Add New Key

To define a new key, click **New Key**. The resulting **New Key** modal provides the following controls:

Key ID: Cribl Stream will automatically generate this unique identifier.

Description: Optionally, enter a description summarizing the purpose of the key.

Encryption algorithm: Currently, Cribl Stream supports the `aes-256-cbc` (default) and `aes-256-gcm` algorithms.

KMS for this key: Currently, the only option supported here is `local` (the Key Management Service built into Cribl Stream).

Key class: Classes are arbitrary collections of keys that you can map to different levels of access control. For details, see Key Classes. This value defaults to `0`; you can assign more classes, as needed.

Expiration time: Optionally, assign the key an expiration date. Directly enter the date or select it from the date picker.

Use initialization vector: If enabled, Cribl Stream will seed encryption with a nonce to make the key more random and unique. Optional (and defaults to disabled) with the `aes-256-cbc` algorithm; automatically enabled (and cannot be disabled) with the `aes-256-gcm` algorithm.

Initialization vector size: Length of the initialization vector (IV), in bytes. This option is displayed only with the `aes-256-gcm` algorithm. Defaults to `12` bytes to optimize interoperability, but you can use the drop-down to set this anywhere between `12` to `16` bytes.

Modify Keys

To protect against accidental changes, a key's parameters - once saved - can be edited only through configuration files.

When you update keys by editing the `keys.json` file, you must add them back to the directories above (respectively, on a single instance or on a Distributed deployment's Leader Node).

Sync `auth/(cribl.secret|keys.json)`

> Cribl Stream performs decryption only when the data is being sent to Splunk. For details, see Decryption of Data in Splunk

To successfully decrypt data, the `decrypt` command will need access to the same keys that were used to encrypt, **in the Cribl instance where encryption happened**.

- In a Single-instance deployment, the `cribl.secret` and `keys.json` files reside in: `$CRIBL_HOME/local/cribl/auth/`.
- In a Distributed deployment, the `cribl.secret` and `keys.json` files reside on the Leader Node in: `$CRIBL_HOME/groups/<group-name>/local/cribl/auth/`.
- When using the UI, you can download these files by selecting **Get Key Bundle**.

Sync/copy these files over to their counterparts on the Search Head/decrypting side, residing in: `$SPLUNK_HOME/etc/apps/cribl/local/cribl/auth/`.

4.4.2. Create And Manage Secrets In Cribl Stream

The Cribl Stream secrets store enables you to centrally manage secrets that Cribl Stream instances use to authenticate on integrated services. Use the **Manage Secrets** page to create and update authorization tokens, username/password combinations, and API-key/secret-key combinations for reuse across the application. These defined secrets can then be programmatically accessed and used within your Cribl Stream configurations using the C.Secret Function.

To ensure they can be decoded correctly, all secrets in Cribl Stream must use `latin1` encoding.

Access Secrets

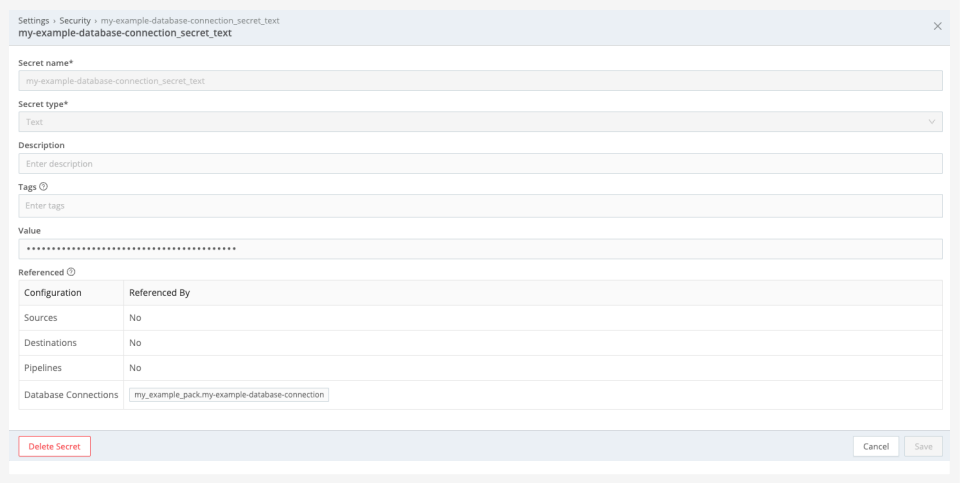
- In a Single-instance deployment, select **Settings** > **Security** > **Secrets**.
- In a Distributed deployment with one Worker Group, select **Configure** > **Settings** > **Security** > **Secrets**.
- In a Distributed deployment with multiple Worker Groups, secrets are managed on each Worker Group. Select **Worker Group** > `<Name>` > **Group Settings** > **Security** > **Secrets**.

On the **Manage Secrets** page, you can configure existing secrets, and/or click **New Secret** to define new secrets.

View Secret References

To view where a secret is used in your deployment:

1. Select a secret from the secrets list.
2. View the **Referenced** table at the end of the secret details. This table lists all Sources, Destinations, Pipelines, or Database Connections that currently reference that secret. If a secret is not referenced in one of those resources, the **Referenced By** column displays **No**.



View where secrets are referenced in your deployment

If a secret originates in a Pack, the name of the secret is prefixed with the name of the Pack. For example: `my-pack.my-secret`.

Add New Secret

The **New Secret** modal provides the following controls:

- **Secret name**: Enter an arbitrary, unique name for this secret.
- **Secret type**: Some options for this field expose additional controls, as described below.
- **Description**: Optionally, enter a description summarizing the purpose of the secret.
- **Tags**: Optionally, enter one or multiple tags related to this secret.

Secret Type

This drop-down offers the following types:

- **Text**: This default type exposes a **Value** field where you directly enter the secret.
- **API key and secret key**: Exposes **API key** and **Secret key** fields, used to retrieve the secret from a secure endpoint. This is the only secret type that Cribl Stream supports for its AWS-based Sources, Collectors, and Destinations, and its Google Cloud Storage Destination.
- **Username with password**: Exposes **Username** and **Password** fields, which you fill to retrieve the secret using Basic Authentication.

Encrypt Secrets Using the Command Line

You also have the option to create and encrypt sensitive values using the same mechanism Cribl Stream employs internally for all sensitive fields. This ensures consistency and leverages the built-in encryption

capabilities within Cribl Stream.

This approach allows you to store all your secrets securely in a password store, keeping them out of your Git repository, and adheres to strict organizational security standards that prohibit storing secrets in version

4.4.3. Configure KMS Providers

Cribl Stream's Key Management Service (KMS) maintains the keys that protect sensitive information throughout your deployment. Cribl Stream offers two options for key management:

- **Internal KMS (Stream Internal)** - The default built-in key management system.
- **External KMS Providers** - Enterprise-grade options with HashiCorp Vault and AWS KMS integration (requires Enterprise license).

The table below summarizes the key differences between the Internal vs. External KMS.

Feature	Internal KMS (Stream Internal)	External KMS Providers (HashiCorp Vault and AWS KMS)
Master Key Storage	Stores `cribl.secret` directly on the filesystem.	Stores `cribl.secret` in a separate key management system.
Master Key Location	In a file called `cribl.secret`.	Completely removed from Cribl Stream's filesystem.
License Requirement	Available by default on all license tiers.	Requires an Enterprise license/plan.
Management	Managed entirely within Cribl Stream.	Managed by the external provider (Vault or AWS).
Configuration/Secrets	Maintained through Cribl's Settings interface.	Configured through Cribl's Settings interface.
Setup Complexity	Simpler to set up.	More complex to set up.
Security	Less separation between the application and its encryption keys.	Enhanced security through separation of encryption keys from the application.
Centralized Key Management	None.	Enables centralized key management across your organization.
Availability	Available by default. Unavailable if an external KMS is enabled for the same Leader Node/Worker Group.	Unavailable if internal KMS is enabled for the same Leader Node/Worker Group.
Supported Providers	N/A	HashiCorp Vault, AWS KMS.

How Encryption Works in Cribl Stream

The Role of `cribl.secret`

At the core of the Cribl Stream encryption system is a master encryption key called `cribl.secret`. This key is used to encrypt sensitive configuration elements such as:

- Authentication credentials in Sources and Destinations (stored in `inputs.yml` and `outputs.yml`)
- API keys and tokens
- Database connection strings (stored in `database-connections.yml`)
- TLS certificates and associated private keys (stored in the `auth/certs` directory
- Other sensitive attributes throughout Cribl Stream
- Secrets stored in `secrets.yml`

KMS Provider Interaction

When using the default Cribl Stream KMS provider:

- The `cribl.secret` value is stored directly on the filesystem in a file called `cribl.secret`.
- Cribl Stream uses this key to encrypt sensitive information in various configuration files.

When you configure an external KMS provider (HashiCorp Vault or AWS KMS):

- The `cribl.secret` file is removed from the filesystem.
- Cribl Stream sets and retrieves the `cribl.secret` value via the chosen external KMS provider.
- The external KMS provider only handles the master encryption key (`cribl.secret`) itself.
- All other sensitive information (Source/Destination credentials, TLS certificates, database connection strings, etc.) remains in the Cribl Stream configuration files, encrypted using the externally-managed key.
- Some sensitive configuration elements may remain in plaintext in the configuration files, depending on the specific attribute.

Manage KMS Across Distributed Environments

In Distributed deployments:

- Each environment (Leader Node, Worker Group) can have its own KMS configuration.
- The internal KMS is unavailable for any environment where an external KMS is enabled.
- Other environments without external KMS configured will continue to use the internal KMS.
- Each environment requires its own distinct secret that cannot be shared between environments.

External KMS Providers and Worker Groups

Some guideines to keep in mind when using an external KMS provider in a Distributed deployment:

- To integrate an external KMS provider into an on-prem Distributed deployment, the Leader Node must have internet access.
- After initial license installation in Distributed mode, you may need to take the following steps to activate KMS features within Worker Groups:
    - Open **Settings** > **Global** > **Service Processes** > **Processes**.
    - In the list of processes, locate any process with a Role of `CONFIG_HELPER`.
    - Select the **Restart** button next to that process.
    - Upon restarting, the KMS will be available for use in the corresponding Worker Group.

Deployment Scenarios

The KMS configuration varies depending on your deployment type:

- In an on-prem Single-instance deployment, you configure the KMS at **Settings** > **Global** > **Security** > **KMS** > **Configure**.
- In a Distributed deployment, you configure the Leader's KMS at the same global location, while additional KMS configs for each Worker Group are available at the Worker Group's **Group Settings** > **Security** > **KMS** > **Configure** page.

Available KMS Providers

The KMS **Configure** drop-down currently provides these options:

- Stream Internal: The only option available without an Enterprise license/plan. With this option, the secrets themselves are configured and maintained in Cribl Secrets section.
- HashiCorp Vault: Stores the `cribl.secret` key securely in HashiCorp Vault, completely removing it from Cribl's local filesystem.
- AWS KMS: Stores the `cribl.secret` key directly within AWS Key Management Service, completely removing it from Cribl's local filesystem.

External KMS Providers and Worker Groups

To integrate an external KMS provider into an on-prem Distributed deployment, Cribl Stream's Leader Node must have internet access.

When you initially install a license in Distributed mode, a known bug prevents immediate use of KMS features within Worker Groups. Here is the workaround:

1. Open **Settings** > **Global** > **Service Processes** > **Processes**.
2. In the list of processes, locate any process with a Role of `CONFIG_HELPER`.
3. Select that process' **Restart** button.

Upon restarting, KMS will be available for use in the corresponding Worker Group.

Internal KMS

The **KMS provider** field defaults to `Stream Internal`. With this option, no further configuration here is required (or possible). See Secrets to configure individual secrets.

HashiCorp Vault

Cribl Stream's integration with HashiCorp Vault uses a Vault policy to define the capabilities granted to tokens used for KMS access. This policy ensures tokens have the necessary permissions to interact with Vault paths relevant to KMS functionality.

Vault Policy Permissions

This table outlines the required Vault policy permissions for different access scenarios.

Path	Capabilities	Description
`<path to vault secret>`	Create, Update, Read	Grants access to create, update, and read secrets at the specified path.
`auth/token/create`	Update	Grants permission to update authentication tokens (only required for Token authentication).
`sys/health`	Read	Allows reading system health information (only needed if enabling health checks).

`<path to vault secret>` represents the actual path to the secret within your Vault hierarchy.

Keep these capabilities in mind as you proceed with the configuration.

Setting the **KMS provider** drop-down to `HashiCorp Vault` exposes the following configuration options:

KMS Settings

Vault URL: Enter the Vault server's URL (for example, `http://localhost:8200`).

Namespace: If you are using HashiCorp Vault Enterprise namespaces, enter the desired namespace.

Authentication

Auth Provider: The method for authenticating requests to HashiCorp Vault server. Select one of `Token`, `AWS IAM`, or `AWS EC2`. Your selection determines the remaining **Authentication** options displayed.

Token-based Authentication

Token: Enter the authentication token. This token will be used only to generate child tokens for further authentication actions.

AWS IAM Authentication

In HashiCorp Vault, the term "method" can refer to `userpass`, `token`, or `aws`, among others, but the `aws` method supports two authentication types: `iam` and `ec2`. Meanwhile, in Cribl Stream, you'll see "method" used differently, for example in the **Authentication method** setting described below.

Use the **Authentication method** buttons to select one of the following:

- **Auto**: Uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance, local credentials, sidecar, or other source. The attached IAM role grants Cribl Stream Worker Nodes access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.
- **Manual**: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials directly or by reference. This is useful for Worker Nodes not in an AWS VPC, for example, those running in a private cloud. It prompts you to provide an **Access key** and a **Secret key**.

Vault AWS IAM Server ID: Value to use for the `Vault-AWS-IAM-Server-ID` header value. This should match the value configured with IAM authentication on Vault.

Vault Role: Authentication role to use in Vault.

Custom Auth Path: If you enabled authentication in HashiCorp Vault with a custom path, enter that path again here.

For example:

- If you enabled authentication with the HashiCorp Vault command `vault auth enable -path /my-auth aws` instead of `vault auth enable aws` you would set a custom path of `my-auth`. Subsequently, when you perform actions using the `vault write` command, you'd specify an auth type with the `auth_type=ec2` or `auth_type=iam` options.

Assume Role

This section is displayed for all AWS IAM authentication methods.

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for Vault Auth: Toggle on if you want to use your Assume Role credentials to access Vault authentication.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming the role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

AWS EC2 Authentication

Vault Role: Enter the authentication role to use in Vault.

Custom auth path: If you enabled authentication in HashiCorp Vault with a custom path, enter that path again here. For example:

- You could have used the HashiCorp Vault command `vault auth enable -path /my-auth aws` to enable authentication with a custom path of `my-auth`. Subsequently, when you perform actions using the `vault write` command, you'd specify an auth type with the `auth_type=ec2` or `auth_type=iam` options.

Secret Engine

Mount: Mount point of the Vault secrets engine to use. (Currently, only the KVv2 engine is supported.) Defaults to `secret`.

Secret Path: Enter the path on which the Cribl Stream secret should be stored. For example, `<somePath>/cribl-secret`.

In a Distributed deployment, the Leader, and each Worker Group, require a distinct secret. This location cannot be shared between them.

Advanced

Enable Health Check: Whether to perform a health check before migrating secrets data. Defaults to toggled on.

Health check endpoint: Configurable endpoint to use for validating system health. Defaults to `/v1/sys/health`.

AWS KMS

Setting the **KMS provider** drop-down to `AWS KMS` exposes the following configuration options:

Authentication

Authentication method: Select an AWS authentication method.

- **Auto**: This default option uses the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, or the attached IAM role. Works only when running on AWS.
- **Manual**: You must select this option when not running on AWS.

The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to `env.AWS_ACCESS_KEY_ID`, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to `env.AWS_SECRET_ACCESS_KEY`, or to the metadata endpoint for IAM credentials.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for KMS: Toggle on if you want to use Assume Role credentials to access the AWS KMS.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role. This is required only when assuming a role that requires this ID in order to delegate third-party access. For details, see AWS' documentation.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Service Configuration

KMS Key ARN: Enter the Amazon Resource Name (ARN) of the AWS KMS Key to use for encryption. This entry is required.

When you configure your IAM account/role in AWS, grant access to the following permissions on the KMS key that will be used:

- `kms:Encrypt`
- `kms:Decrypt`

Then use that account for authentication.

4.5. Compliance

Cribl offers compliance options with different security standards, including:

- FIPS
- STIG
- SELinux

4.5.1. FIPS Mode

Federal Information Processing Standards FIPS represent a set of US government standards and guidelines for information security. You can deploy Cribl Stream in **FIPS mode**. This mainly restricts the cryptographic algorithms used within Cribl Stream and also enforces FIPS compliant password requirements.

> Do not start Cribl Stream without FIPS mode enabled, or else you will be unable to run it in FIPS mode later. You must enable FIPS mode as described in this section, after installing **but before starting** Cribl Stream.

Requirements

This section describes the system and password requirements for FIPS mode.

FIPS Mode System Requirements

To run Cribl Stream in FIPS mode, your environment must satisfy the following requirements.

Operating Systems

Must support FIPS 140-2 or FIPS 140-3. Several Linux distributions meet this standard. See NIST's list of tested configurations here.

> While RHEL 9 can be configured to run in FIPS mode at the operating system level, this **does not automatically enable FIPS mode within Cribl Stream.** You must still follow the procedures outlined in this document to explicitly enable Cribl Stream's FIPS mode. Cribl Stream relies on its own configuration to ensure FIPS compliance, independent of the OS-level FIPS setting.

OpenSSL

The operating system environment must have a FIPS validated version of OpenSSL installed.

- **For Cribl Stream versions 4.5.0 through 4.8.1**: The FIPS provider version must be 3.0.8 or 3.0.9, and the system-wide OpenSSL version must also be 3.0.8 or version 3.0.9, with its associated certificate.
- **For Cribl Stream starting with version 4.8.2**: The FIPS provider version requirement must be >= 3.0.5, and the system-wide OpenSSL version is no longer a factor. Cribl Stream now verifies the OpenSSL version bundled within the binary.

Enterprise Linux 9 is the only version of Enterprise Linux that includes OpenSSL version 3.0.9 out of the box. While it's possible to manually install compatible versions of OpenSSL on older Enterprise Linux versions, this

process is complex and generally not recommended. Therefore, the minimum recommended Enterprise Linux version for running Cribl Stream in FIPS mode is 9.

Even when running Cribl Stream on a FIPS-enabled RHEL 9 system, you must still explicitly enable Cribl Stream's FIPS mode as described in the Running Cribl Stream in FIPS Mode section below. The OS being in FIPS mode is not sufficient for Cribl Stream to operate in a FIPS-140 compliant manner.

FIPS Mode Restrictions on Cryptographic Algorithms

When running in FIPS mode, Cribl Stream uses only those cryptographic algorithms that satisfy the FIPS standards. This means that Cribl Stream operates exclusively with FIPS-compliant cryptographic algorithms, disallowing the use of non-compliant algorithms like MD5 (message-digest) or CRC-32 (Cyclic Redundancy Check 32) algorithms.

To ensure proper functionality in FIPS mode, the FIPS provider version must be at least `3.0.5`.

Therefore, in FIPS mode:

Cribl expressions **that rely on MD5 or CRC-32 will fail silently.** This includes expressions that use the `C.Mask.md5()` method and can in turn affect the behavior of parent Functions and their existing parent Pipelines.

The UI will hide certain options normally made available by the typeahead feature.

No Source, Destination, or Email Notification target that uses the Amazon AWS SDK v2 will perform checksums. This is because that SDK uses MD5 to verify checksums, and it applies to both incoming or outgoing payloads. Specifically:

- The Amazon Kinesis Data Streams Source will not perform checksums and the **Verify KPL checksums** setting will not be available.
- The Amazon SNS Notification target will not perform checksums.
- The following Destinations will not perform checksums:
    - Amazon CloudWatch Logs
    - Amazon Kinesis Data Streams
    - Amazon SQS
    - Google Cloud Storage
    - MinIO
    - Prometheus

Role-Based Access Control (RBAC) Controls

Cribl Stream's FIPS mode requires RBAC to be enabled for enhanced security. This ensures proper access management within FIPS-compliant environments. To enable RBAC, the Cribl Stream instance has to be running in Distributed mode. FIPS mode is available on certain license tiers. For details, see Pricing.

If FIPS mode is already enabled and RBAC is subsequently disabled, Cribl Stream will automatically disable FIPS mode and log a message stating:

```
"Disabling FIPS due to Role-based Access Control (RBAC) being disabled."
```

Check `<cribl_install_dir>/local/_system/instance.yml` to ensure Cribl is running in Distributed mode. The default installation directory is typically `/opt/cribl`. The `mode` setting in this file should be set to `distributed`. If it is set to `standalone`, you will need to switch to a Distributed deployment to use FIPS mode with RBAC enabled.

FIPS Mode Password Rules

When Cribl Stream is in FIPS mode, all passwords must:

- Contain eight or more characters.
- Use characters from three or more of the following categories:
  - Lowercase letters.
  - Uppercase letters located after the first character in the password.
  - Digits located before the last character in the password.
  - Non-alphanumeric ASCII characters such as `#`, `!`, or `?`.
  - Non-ASCII characters such as `ñ`, `€`, or emoji.

As of Cribl Stream version 4.5.0, these rules apply for all passwords, whether or not Cribl Stream is running in FIPS mode, with one exception:

- For Cribl Stream not running in FIPS mode, local users whose passwords existed before Cribl Stream 4.4.4 was released can continue to use their passwords, even if those passwords do not satisfy the rules.
- When these users change to new passwords, the new passwords must satisfy the rules.
- New users must create passwords that satisfy the rules.

If you get locked out of your account, you need to reset your password manually.

Running Cribl Stream in FIPS Mode

To begin using Cribl Stream with FIPS enabled, complete either the non-systemd or the systemd procedure below, **before you start Cribl Stream for the first time after installing**.

Enabling FIPS Mode on a Non-systemd Distribution

If the Linux distribution on which you run Cribl Stream **does not** use systemd:

1. Complete **either** of the two following steps:
   - Set the `CRIBL_FIPS` environment variable to `1` (true), **or**,
   - Edit `cribl.yml`, adding this top-level element:

     ```
     fips: true
     ```

     Copy

     Note that `fips: true` is the entire top-level element. For this element, the following "stanza" syntax would be incorrect - **do not** use it:

     ```
     # stanza syntax is incorrect for the fips element
     # even if other elements use it
     fips
       fips: true
     ```

     Copy

2. If you have installed a non-default OpenSSL binary (otherwise, skip this step):
   - Run the following command, and in the output, note the values of `MODULESDIR` and `OPENSSLDIR`.

     ```
     <binary_of_non-default_OpenSSL> version -a
     ```

     Copy

   - Set `OPENSSL_MODULES` to the value of `MODULESDIR`.
   - Set `CRIBL_OPENSSL_DIR` to the value of `OPENSSLDIR`.
3. Ensure RBAC is configured in Cribl Stream.
4. Start Cribl Stream.

Finally, verify that you are in FIPS mode as explained below.

Enabling FIPS Mode on a systemd Distribution

If the Linux distribution on which you run Cribl Stream **does** use systemd:

1. Run the following command, and in the output, note the values of `MODULESDIR` and `OPENSSLDIR`. (As shown, the command will run the default OpenSSL binary. If you have installed a different OpenSSL binary, you can modify the command to run that one instead.)

```
openssl version -a
```

Copy

2. Run the following command, replacing the placeholders with the relevant paths from your environment. (This assumes that you do not already have a `nodejs.cnf` file. If you **do** have one, complete the workaround below instead of this step.)

```
<path_to_Cribl_binary> generateFipsConf -d <path_to_OpenSSL_root_directory>
```

Copy

   If you encounter issues generating the FIPS configuration file, a discrepancy in the OpenSSL path might be the cause. For more detailed troubleshooting specific to RHEL 9, refer to the knowledge base article: Installing Cribl Stream in FIPS Mode on RHEL 9.

   An example of a successful command and its output:

```
useralice@123456791234:/# /opt/cribl/bin/cribl generateFipsConf -d /lib/usr/ssl
/opt/cribl/state/nodejs.cnf
```

Copy

   This command creates `nodejs.cnf` - the configuration file that NodeJS uses to run in FIPS mode as Cribl Stream starts.

3. Run the following command to open an empty temporary file in an editor:

```
systemctl edit cribl
```

Copy

4. Add the following content to the file, replacing the first placeholder with the value of `MODULESDIR` that you noted in Step 1, and the second with the path to your `nodejs.cnf` file:

```
[Service]
Environment="OPENSSL_MODULES=<value_of_MODULESDIR>"
Environment="OPENSSL_CONF=<path_to_nodejs.cnf_file>"
```

Copy

5. Save and exit the editor.
6. Run the following commands to reload the systemctl overrides and start Cribl:

```
systemctl daemon-reload
systemctl start cribl
```

Copy

7. Finally, verify that you are in FIPS mode as explained below.

When a `nodejs.cnf` File Already Exists

If you already have a `nodejs.cnf` file, replace Step 2 above with the following workaround:

1. Create a file with the following content, replacing `<placeholder>` with the value of `OPENSSLDIR`. Name the file something other than `nodejs.cnf`.

```
nodejs_conf = nodejs_init
openssl_conf = nodejs_init

.include /<placeholder>/fipsmodule.cnf

[nodejs_init]
providers = provider_sect

[provider_sect]
default = default_sect
# The fips section name should match the section name inside the
# included fipsmodule.cnf.
fips = fips_sect
[default_sect]
activate = 1
```

Copy

2. Resume the main procedure above, starting with Step 3.

Verify FIPS Mode

- On the Leader or Worker Node, search `cribl.log` for this message:

```
"level":"info","message":"running with FIPS enabled", "OpenSSLVersion":"3.0.13+q
```

Copy

- The presence of the above message confirms that Cribl Stream is in FIPS mode.

4.5.2. Running Cribl Stream On A Hardened OS

When deploying Cribl Stream in environments with stringent security requirements, such as those adhering to the Department of Defense Security Technical Implementation Guides (STIGs), additional configuration steps are necessary to ensure the application can operate without interference.

Understanding STIG Compliance

Operating any system on a network, especially in restricted environments like the DoD, requires obtaining an Authority to Operate (ATO). This involves compliance with STIGs, which cover various aspects of system security, including the operating system, database, and application.

For Cribl Stream, the most relevant STIGs are those governing the operating system deployment. By adhering to these STIGs, you ensure a hardened environment with configured tools to safeguard your system. However, certain actions within these STIGs may be blocked by default, requiring explicit configuration to allow Cribl Stream to execute.

Security Tools and Configurations

To ensure Cribl Stream operates securely in a hardened environment, consider implementing the following essential configurations for your relevant security tools.

SELinux (Security Enhanced Linux)

When running Cribl Stream on a system with SELinux, it's crucial to download the `tgz` file directly to the `/opt/` directory. Additionally, ensure that the `.tgz` file is uncompressed within the `/opt/` directory. Moving files on a system with SELinux might lead to complications.

For details, see the SELinux Configuration topic.

firewalld

firewalld is a dynamic firewall management tool that offers support for network/firewall zones, allowing you to define the trust level of network connections or interfaces. It encompasses features for managing IPv4 and IPv6 firewall settings, Ethernet bridges, and IP sets. According to the DOD STIG, firewalld should enforce a deny-all rule and should selectively open ports.

In a Cribl Stream environment, the Leader Node requires ports `4200` and `9000` over TCP to be open. Additionally, Worker Nodes should have port `9000` over TCP open if Worker UI access is enabled, along with any necessary Source ports.

Refer to your specific firewall configuration guidelines for details.

fapolicy

fapolicy is a software framework that regulates application execution through a user-defined policy, adhering to an allow-list approach. In this case, you should explicitly permit or trust applications to execute within a deny-all catch-all policy. This framework serves as an efficient method to prevent the execution of untrusted and potentially malicious applications on the system.

Cribl Stream operates seamlessly in secure environments, but additional configuration steps are needed for systems using fapolicy (File Access Policy). To ensure Cribl Stream can execute without interference, we'll use trust files to explicitly authorize Cribl Stream binaries.

There are two main ways to achieve this:

Trust Files (Recommended): Trust files offer a more efficient and streamlined approach compared to writing custom fapolicyd rules. Trust files allow you to define authorized applications and their locations, enabling fapolicy to recognize Cribl Stream as a trusted application without the need for extensive rule configurations.

You define trusted applications and their locations in a trust file. fapolicy then recognizes Cribl Stream as trusted and allows it to run.

Custom Rules (Advanced): This method offers more granular control but requires writing custom fapolicy rules. While less performant, you can write custom fapolicyd rules to allow Cribl Stream execution. However, this approach is recommended for advanced users familiar with fapolicy rule syntax.

For most users, using trust files is the recommended approach due to its simplicity and performance benefits. However, if you require more granular control or have specific fapolicy configurations, custom rules might be a suitable alternative.

Upgrading Cribl Stream with fapolicy

When using fapolicy, UI-based upgrades might fail due to its tracking of individual binaries. To successfully upgrade, you must manually update the fapolicy trust file to recognize the new Cribl Stream binaries. This ensures fapolicy allows the upgraded components to execute properly.

Common Challenges and Solutions

- **Access Denied Errors:** If you encounter errors related to file access or execution permissions, ensure SELinux and fapolicy are configured correctly.
- **Network Connectivity Issues:** Verify that firewall rules allow necessary network traffic for Cribl Stream to communicate with other components.
- **Configuration Errors:** Double-check your configuration settings for SELinux, fapolicyd, and firewall rules to ensure they align with your security requirements.

For details and configuration templates, refer to the Guides section in the Cribl Curious Knowledge Base.

4.5.3. SELinux (Enforcing Mode) Configuration

SELinux (Security-Enhanced Linux) is a mandatory access control system for Linux that provides a granular level of security by defining access controls for applications, processes, and files. It uses security policies, which are a set of rules that tell SELinux what can or can't be accessed, to enforce the access allowed by a policy.

SELinux has two modes, **enforcing** and **permissive**. When running in enforcing mode, it strictly enforces the SELinux policy and denies access based on policy rules. This can sometimes cause conflicts with applications that aren't explicitly allowed to perform certain actions.

Cribl Stream and SELinux

To effectively integrate Cribl Stream with SELinux, consider the different policy requirements for in-stream processing and network ports.

- Cribl Stream's in-stream processing often involves predefined processes and files, allowing for a relatively restrictive SELinux policy.
- Network ports might require a more permissive policy because users configure them within the UI.

Key Steps for SELinux Configuration

To ensure optimal integration of Cribl Stream with SELinux, begin by verifying the current SELinux status and the context of the Cribl Stream binary.

Check SELinux Status

Use the `getenforce` command to determine the current status of SELinux on your system.

The output will be one of the following:

- `Enforcing`
- `Permissive`
- `Disabled`

If the output is `Enforcing`, SELinux is enabled and enforcing policies.

Verify Cribl Stream's SELinux Context

Use the command `ls -Z /opt/cribl/bin/cribl` to list the contents of the directory `/opt/cribl/bin` while displaying their SELinux security contexts.

The output should show the SELinux context for the Cribl Stream binary.

Example output:

```
-rwxr-xr-x. cribl cribl cribl unconfined /opt/cribl/bin/cribl
```

Deploy Cribl Stream directly to `/opt` to avoid context issues. Do not deploy Cribl Stream to a user's home directory, or to root (`/`), and then move it to `/opt`.

Revert Labels on Files and Directories

If needed, you can also use `restorecon`, an SELinux policy utility from the `policycoreutils-python-utils` package, to revert labels on files and directories to their default SELinux security context:

```
restorecon -R -v /opt/cribl
```

Copy

Check for SELinux Denials

Use `auditd` to log SELinux denials. Look for messages related to Cribl Stream in the `/var/log/audit/audit.log` file.

Modify SELinux Policy

If you encounter SELinux denials, you might need to modify the SELinux policy. There are three options ranging from least secure to most secure.

Use `semanage` (Less Secure)

Using `semanage` is the simplest but least secure method. It grants permissive access to the `cribl_t type`, allowing all operations.

```
semanage permissive -a cribl_t
```

Copy

Use `audit2allow` (Slightly More Secure)

Using `audit2allow` creates SELinux policy rules to permit actions that were previously denied.

To identify and address SELinux denials related to Cribl Stream, follow these steps:

1. Find denied operations:

```
sudo grep cribl /var/log/audit/audit.log | audit2allow -M cribl
```

Copy

2. Install the generated policy:

```
sudo semodule -i cribl.pp
```

Copy

Create a Custom Policy (Most Secure)

Use tools like `secil` or `policycoreutils` to create a custom policy tailored to Cribl Stream's needs. This provides the highest level of security but requires more technical expertise.

Reload SELinux Policy

Once you've altered the SELinux policy using tools like `semanage`, `audit2allow`, or through custom policy creation, use this command to ensure that the new rules take effect immediately.

```
sudo semanage reload -a
```

Copy

Specific Cribl Stream Considerations

In addition to the general SELinux guidelines, pay close attention to these Cribl Stream-specific factors:

- **Cribl Stream configuration**: Review Cribl Stream configuration files for any settings that might conflict with SELinux restrictions.
- **Integration with other applications**: Ensure that Cribl Stream's interactions with other applications are compatible with their SELinux contexts and permissions.

Additional Considerations

In addition to the basic configuration, these factors are also important for optimal performance:

- **SELinux booleans**: Check the SELinux documentation for relevant booleans that might affect Cribl Stream's behavior.

- **SELinux context labeling**: Ensure files and directories used by Cribl Stream have the correct SELinux context. Use tools like `restorecon` to relabel files.
- **SELinux troubleshooting tools**: Use tools like `audiscan` and `audit2why` to analyze SELinux audit logs.

SELinux Troubleshooting

SELinux problems usually surface when you try to start the Cribl Stream service. Users immediately receive an error like the following:

```
Job for cribl.service failed because the control process exited with error code.
See "systemctl status cribl.service" and "journalctl -xe" for details.
```

Copy

One indication that the problem is in SELinux configuration, not in the Cribl Stream service itself, is when (1) nothing has been logged in `$CRIBL_HOME/log/cribl.log`, and (2), when you run `journalctl -xe` you see errors like the following:

Errors from journalctl -xe

```
cribl.service: Failed to execute command: Permission denied
cribl.service: Failed at step EXEC spawning /opt/cribl/bin/cribl: Permission denied

cribl.service: Control process exited, code=exited status=203
cribl.service: Failed to execute command: No such file or directory
cribl.service: Failed at step EXEC spawning /bin/rm -f /opt/cribl/pid/cribl.pid: Perm
```

Copy

You can confirm that SELinux configuration is the issue by verifying that the Cribl Stream service is able to start up on its own. To do this, run the `cribl start` command.

Here are three commands that you can think of as a basic SELinux troubleshooting toolkit:

- `getenforce` verifies that SELinux is enabled and enforcing policy.
- `sestatus -v` shows details of the SELinux policy that's in effect.
- `ls` with the `-Z` or `--context` option shows the SELinux tags on files and/or folders. In the output, the fifth column reports the SELinux context in the form `user:role:type:level`.

Example Problem

Suppose you run `ls -lZ /opt/cribl` to see the SElinux context for `/opt/cribl`.

If the output looks like the following, there is a problem: the presence of "home" among the `type` part of the Linux context. In this case, `user_home_t` is the offending type. When the type is `user_home_t` or `admin_home_t`, SELinux will not permit the service to read, write, or execute files.

Problematic SELinux context output

```
$ ls -laZ /opt/cribl
total 0
drwxr-xr-x. 6 cribl cribl unconfined_u:object_r:user_home_t:s0  62 Feb 16 08:31 .
drwxr-xr-x. 3 root  root  system_u:object_r:usr_t:s0            19 Feb 16 19:54 ..
drwxr-xr-x. 2 cribl cribl unconfined_u:object_r:user_home_t:s0 202 Feb 16 08:31 bin
drwxr-xr-x. 5 cribl cribl unconfined_u:object_r:user_home_t:s0  49 Feb 16 08:20 data
drwxr-xr-x. 6 cribl cribl unconfined_u:object_r:user_home_t:s0  59 Feb 16 08:31 defau
drwxr-xr-x. 3 cribl cribl unconfined_u:object_r:user_home_t:s0  22 Feb 16 08:31 thirc
```

Copy

Example Solution

To fix the labeling, download Cribl Stream again directly to `/opt`. **Do not** download Cribl Stream to a user's home directory, or to root (`/`), and then move it to `/opt` - that was what caused our example problem.

Downloading Cribl to /opt

```
curl -Lso - $(curl -s https://cdn.cribl.io/dl/latest) | sudo tar zxvf - -C /opt
```

Copy

A simpler option is to revert the labels using `restorecon`:

Reverting labels with restorecon

```
restorecon -R -v /opt/cribl
systemctl restart cribl
```

Copy

Once you correct the problem, the output will look like the following. Instead of `*home_t` listings in the `type` field, this SELinux context has `usr_t`, a type that will satisfy SELinux's criteria for allowing the service to read, write, and execute files.

Corrected SELinux context output

```
# ls -laZ /opt/cribl
total 0
drwxr-xr-x. 6 cribl cribl unconfined_u:object_r:usr_t:s0  62 Feb 16 08:31 .
drwxr-xr-x. 3 root  root  system_u:object_r:usr_t:s0      19 Feb 16 20:05 ..
drwxr-xr-x. 2 cribl cribl unconfined_u:object_r:usr_t:s0 202 Feb 16 08:31 bin
drwxr-xr-x. 5 cribl cribl unconfined_u:object_r:usr_t:s0  49 Feb 16 08:20 data
drwxr-xr-x. 6 cribl cribl unconfined_u:object_r:usr_t:s0  59 Feb 16 08:31 default
drwxr-xr-x. 3 cribl cribl unconfined_u:object_r:usr_t:s0  22 Feb 16 08:31 thirdparty
```

Copy

5. Monitor Health And Metrics

The Monitoring section of Cribl Stream provides insights into the performance, health, and operational status of your Cribl Stream deployment and data pipelines. Use Monitoring for:

- **Real-time Pipeline monitoring** - Monitor the real-time status of Pipelines, including metrics such as data throughput. See Data Monitoring for more information.
- **Performance tuning and resource utilization tracking** - Get visibility into the utilization of system resources, such as CPU, memory, and network bandwidth for your nodes. Use this information to predict future resource needs and plan for capacity expansion or infrastructure upgrades as needed. Identify areas for optimization, such as tweaking Pipeline configurations or adjusting resource allocations. See Overview and System Monitoring for more information.
- **Alerting and notification** - View Notifications that are triggered when certain thresholds or conditions are met, such as a node going offline, data processing delays, or high error rates. Detect errors or anomalies in data processing, such as failed processing tasks, dropped events, or unexpected data patterns. See Notifications for more information.

Monitoring Page

From Cribl Stream's sidebar, select **Monitoring** to access the Monitoring section of Cribl Stream. The Monitoring section has these submenus:

Submenu	Description
Overview	Displays information about traffic in and out of the system, as well as collection jobs and tasks. It tracks events, bytes, splits by data fields over time, and broader system metrics.
Data	View isolated throughput data for: • Data Fields • Destinations • Packs • Pipelines • Projects • Routes • Sources • Subscriptions
System	View isolated throughput data for: • Job Inspector

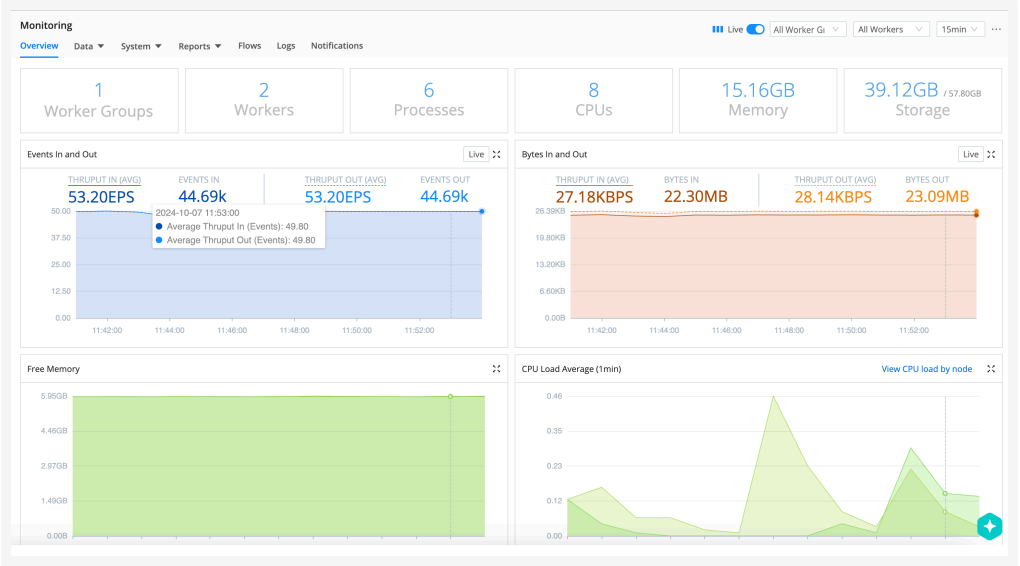
Submenu	Description
	• Jobs (and tasks in-flights; see Collector Sources) • Leaders (available only with High Availability enabled) • Licensing • Queues (Destinations) • Queues (Sources)
Reports	Examine the Top Talkers report to see your five highest-volume Sources, Destinations, Pipelines, Routes, and Packs.
Flows	A graphical, left-to-right visualization of data flow through your deployment.
Logs	Comprehensive information about an instance's status/health, inputs, outputs, Pipelines, Routes, Functions, and traffic.
Notifications	Notifications alert Cribl Stream admins about issues that require their immediate attention.

All Monitoring charts feature a local, configurable time range picker. These include charts for **Sources**, **Destinations**, **Data Fields**, **Packs**, **Pipelines**, **Projects**, **Routes**, and **Subscriptions**. Also included are system metrics like **CPU Load**,**Free Memory**, and **Bytes In/Out**. By default, each chart inherits the current global Monitoring time range (set from the control in the upper right of the page), but any changes you make to a chart's local time range affect only that chart and do not change the global time range.

Overview

The Overview page is the default page of the Monitoring section. It displays multiple dashboards with information about traffic in and out of the system, as well as collection jobs and tasks. It tracks events, bytes, splits by data fields over time, and broader system metrics.

The initial view shows aggregate data for all Worker Groups and all Workers. You can use the drop-downs at the upper right to isolate individual Worker Groups, or individual Workers.



Monitoring page

Using the drop-down menu at the upper right, you can change the time range shown in the Overview display from the default `15 min`, selecting time ranges from `5 min` up to `1 day`. The `1 day` setting covers the preceding 24 hours, and this maximum window is not configurable. You can also select `@midnight`, which displays data beginning at midnight on the current day. In addition, each Overview chart includes its own local time range picker, which starts from the current global time range but can be adjusted per chart without affecting the global selection.

Vertical lines across each chart display configuration changes. Click anywhere on the line to view summary information including time, data, and configuration versions.

Except for these configuration change markers, Monitoring data does not persist across Cribl Stream restarts. Keep this in mind before you restart the server.

Storage and Memory Calculations

The displayed **Storage** tile (upper right) represents the amount of free storage remaining on the partition that hosts Cribl Stream. (This quantity might not represent the maximum storage available for the selected Worker or Group. Also, it does not calculate the system free space.)

Similarly, the **Free Memory** graph reflects only the operating system's `free` statistic, matching Linux's strict `free` definition by excluding `buff/cache` memory. So this graph indicates a lower value than the OS' `available` memory statistic - and it does not necessarily indicate that the OS is running out of memory to allocate.

The Cribl Stream user interface displays metrics in units of KB, MB, GB, and TB. At each level, these are multiples of `1024`.

Byte Calculations

Byte-related charts show the uncompressed amounts and rates of data processed over the selected time range:

- Events (total) in and out
- Events per second in and out
- Bytes (total) in and out
- Bytes per second in and out

You can control the selected time range globally from the Monitoring page, or per chart using the local time range picker.

When Cribl Stream receives data from a Source:

- Bytes in gets counted after deserializing and event breaking.
- Most Sources generate a `_raw` string. For these sources, bytes are counted as the length of the `_raw` string.
- For Sources that do not generate a `_raw` string, bytes are counted as the size of the `__bytes` field, such as for metrics and network flows. In this case, `__bytes` will represent the size of the raw bytes read in, before being parsed into event fields.
- Byte count does not include the size of any fields other than `_raw`.

When Cribl Stream sends data to a Destination:

- For most typical Destinations, Cribl Stream calculates the bytes out based on the outgoing payload (typically a JSON string) prior to compression. A JSON-based destination will count the length of the JSON string post-serialization. This string would not include any internal fields or any fields that the Destination does not serialize, even if they are present on the event.
- For these typical Destinations, what is actually serialized may still differ. For example, protobuf-based Destinations will count the protobuf-encoded bytes, but a JSON-based Destination will count the length of the JSON string post-serialization. The result is that the same event may have two different reported bytes counts, depending on which Destination the event is sent through.
- However, some Destinations are exceptions to the typical bytes out calculation. These are the possible alternative calculation scenarios. For some Destinations, Cribl Stream calculates the byte count in one of the following ways:
    - The length of `_raw`, when sent as plain text instead of JSON.
    - A static number of bytes per event. For example, when sending metrics to Splunk using S2S.
    - The length of the stringified JSON, including all internal (__*) fields.

See the documentation for a specific Destination for more information on the bytes out calculation.

CPU Load Calculations

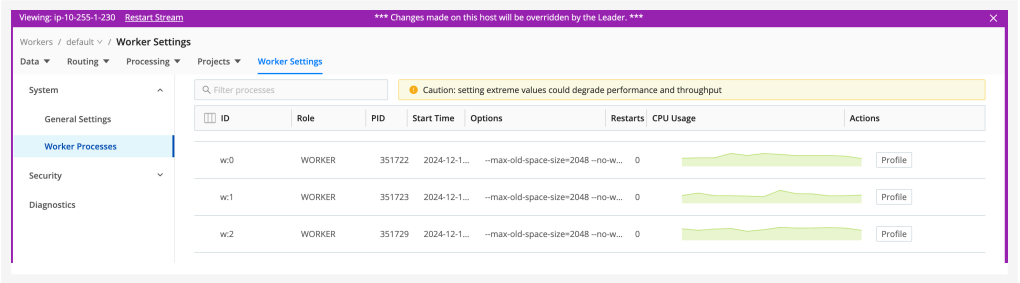
The displayed **CPU Load Average** is an average per Worker Process, updated at 1-minute granularity. (It is not an average for the Worker Node as a whole.)

Use the local time range picker on the **CPU Load** chart to narrow or expand the time window for analyzing load trends, independently of the global Monitoring time range.

Select **View CPU load by node** to open a modal showing CPU usage per Worker.

Worker Node CPU Usage modal

Select a **View Details** button to access a Worker Node's **Worker Settings**, with details per Worker Process. (To enable this button, you must first enable the parent Group's Remote UI Access.)

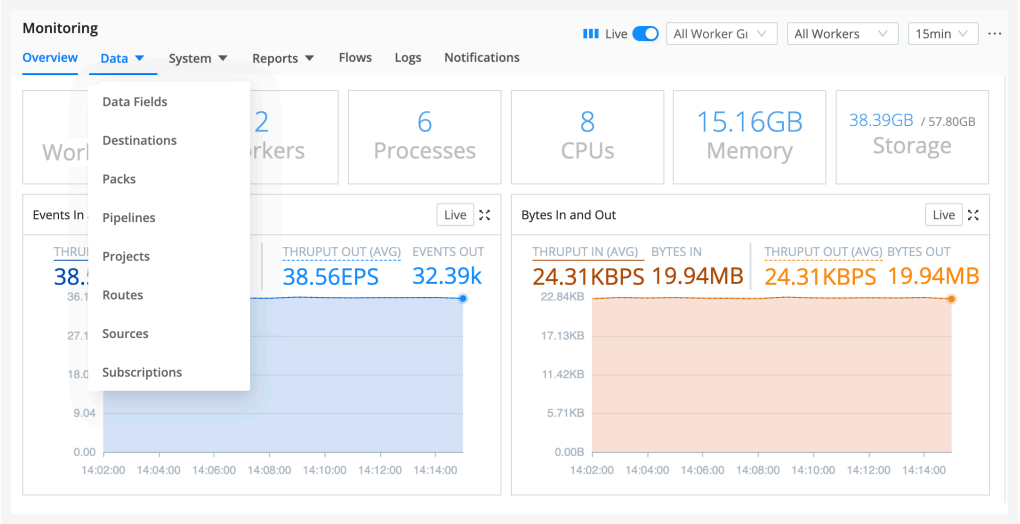


Worker Settings page

Data

From the **Monitoring** page, open the **Data** submenu to isolate throughput for any of the following:

- Data Fields
- Destinations
- Packs
- Pipelines
- Projects
- Routes
- Sources
- Subscriptions



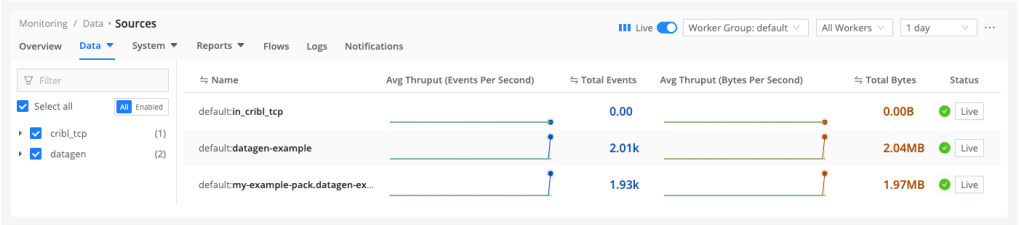
Monitoring > Data submenu

All **Monitoring** > **Data** views inherit the global Monitoring time range by default, and each chart on these pages also exposes a local time range picker you can use to zoom in or out on a specific chart without changing the global time range or other charts.

The Monitoring page provides a consolidated view of metrics for all resources (Sources, Destinations, Pipelines, Routes), including those contained within Packs. To distinguish between resources with identical IDs across different contexts, Cribl Stream uses a namespacing schema. Resource names are prefixed according to their location in your deployment:

Context	Resource name syntax	Example
Worker Group	`Worker-Group/Fleet-name:resource-name`	`default:my-syslog-example`
Pack	`Worker-Group/Fleet-name:Pack-name.resource-name`	`default:my-pack.my-syslog-example`

This allows you to filter metrics by a specific Worker Group or Pack using the prefixes.



Resource names with Worker Group and Pack prefixes

Sparklines

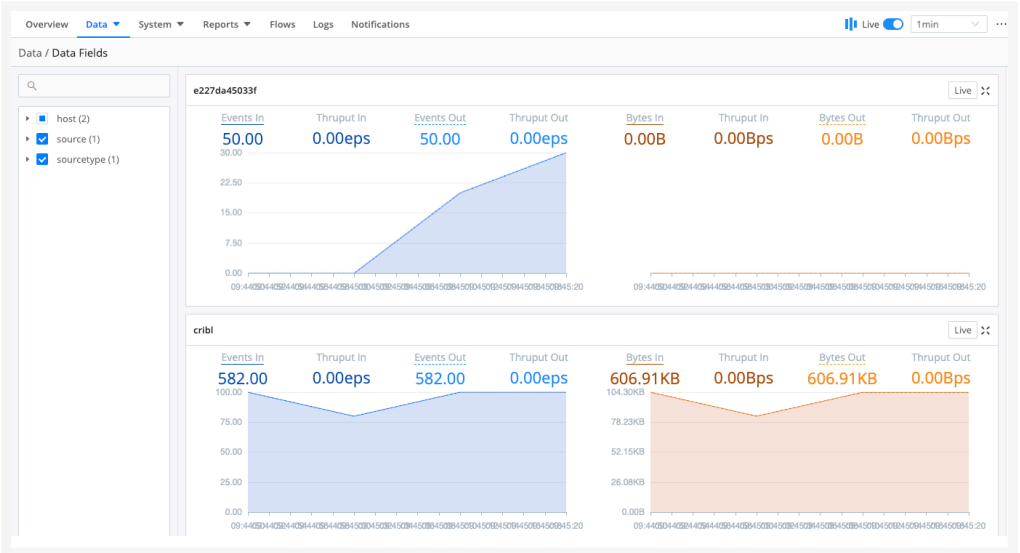
Dense displays are condensed to sparklines for legibility. Hover over the right edge to display Maximize buttons that you can select to zoom these up to detailed graphs.

Sparklines and fly-out

Data Fields

The **Data Fields** page lets you preview the flow of events that contain a specific data field. The left (blue) side summarizes events in/out. The right (orange) side summarizes bytes in/out.

To control which data fields get included in these graphs, use the Disable field metrics setting.



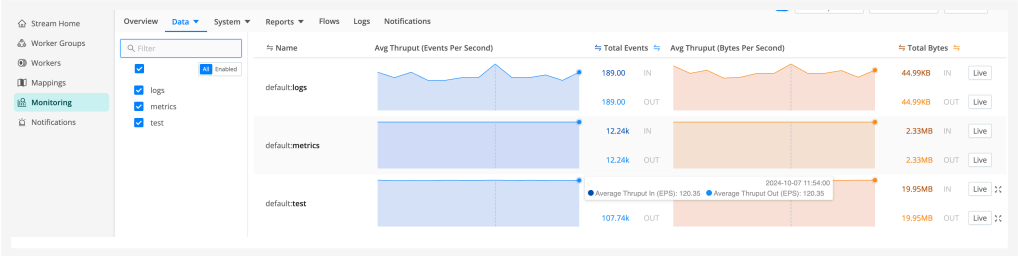
Data Fields graph in Monitoring

Disable field metrics settings are not available on Cribl.Cloud-managed Worker Nodes.

Routes

The **Routes** page condenses multiple details and options:

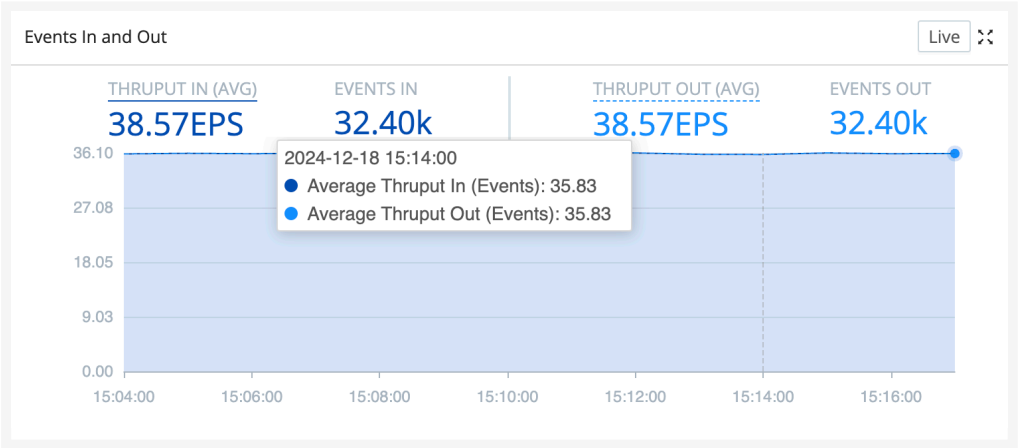
- Each row independently summarizes throughput for a separate Route.
- The left (blue) side summarizes events in/out. The right (orange) side summarizes bytes in/out.
- On each row, the top number summarizes average events/bytes into the Route, and the bottom number summarizes events/bytes out. Both are averaged over your selected time range.
- On each row, the upper Maximize button zooms up the left (events) graph, and the lower Maximize button zooms up the right (bytes) graph.



Routes page's multiple rows and controls

Fly-Out Details

You can hover over graphs to display a fly-out with details. The **Free Memory** and **CPU Load Average** graphs only display fly-outs for up to 10 Worker Nodes, even when more Nodes are active. These fly-outs will occasionally show transient `0` metrics for some Workers, because Cribl Stream prioritizes reporting current throughput over memory/load metrics.



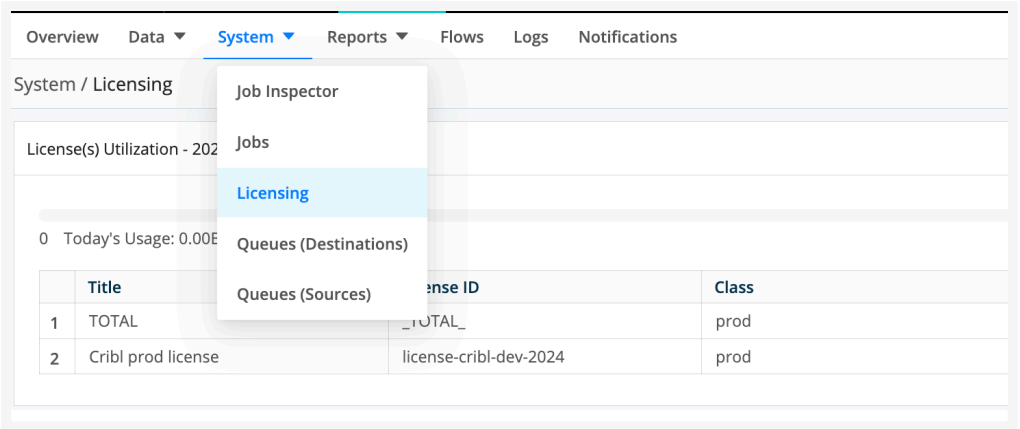
Throughput details

System

From the **Monitoring** page, open the **System** submenu to isolate throughput for any of the following:

- Job Inspector
- Jobs (and tasks in-flight, see Collector Sources)

- Leaders (available only with High Availability enabled)
- Licensing (on-prem deployments only)
- Queues (Destinations)
- Queues (Sources)



Monitoring > System submenu

Charts under **Monitoring** > **System** (including **CPU Load**, **Free Memory**, queue utilization, and related system metrics) use the global Monitoring time range by default and also provide local time range pickers on each chart so you can refine the time window for a specific metric without altering the global setting.

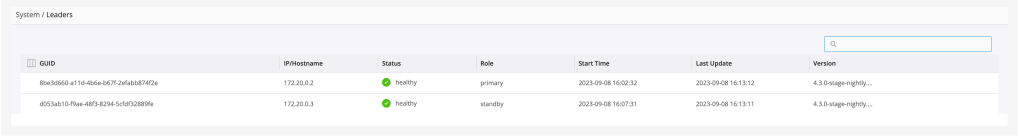
Job Inspector

Select **System** then **Job Inspector** from the Monitoring page to view and manage pending, in-flight, and completed collection jobs and their tasks. For details about the resulting page, see Monitoring and Inspecting Collection Jobs.

Leaders

This menu item is available only in Distributed Deployments with High Availability (HA) enabled.

Select **Monitoring**, then **System** and **Leaders** to view the status of your Leader Nodes. For more information on how to configure a second Leader Node for failover/durability, see Leader High Availability/Failover.



Monitoring Leader Nodes

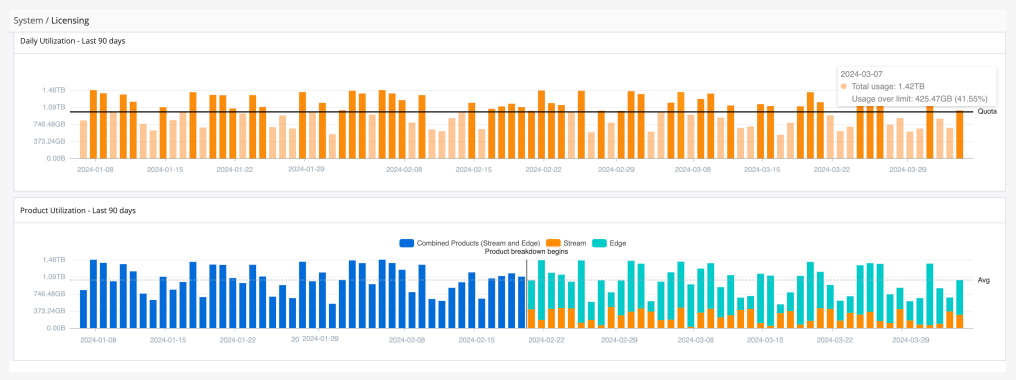
Licensing

Select **System**, then **Licensing** from the Monitoring page to check the expiration dates of your licenses, daily data throughput, and events quotas. You can also compare your daily Cribl Stream and Cribl Edge data

throughput against your license quota - and against granular and average throughput over the last 30 to 365 days.

In this graph, you'll see the following:

- A horizontal bar that indicates your quota limit.
- Legacy data is represented by blue bars that show licensing usage in versions older than 4.6.0. Starting in v.4.6.0, the system separated the data, displaying Cribl Edge usage in teal and Cribl Stream usage in orange. If only Cribl Stream is used, Cribl continues to display usage with blue bars.
- Tooltips display details about data usage, data amount over/under license quota, Cribl Edge dropped bytes, and data percentage over/under license quota.
- Dots on the daily usage bar graph represent configuration changes in the system.
- The **Daily Events In** chart only shows events that count towards your license, broken out by product (Cribl Stream and Cribl Edge). It filters out `datagen`, `cribl_http`, and `cribl_tcp` events.



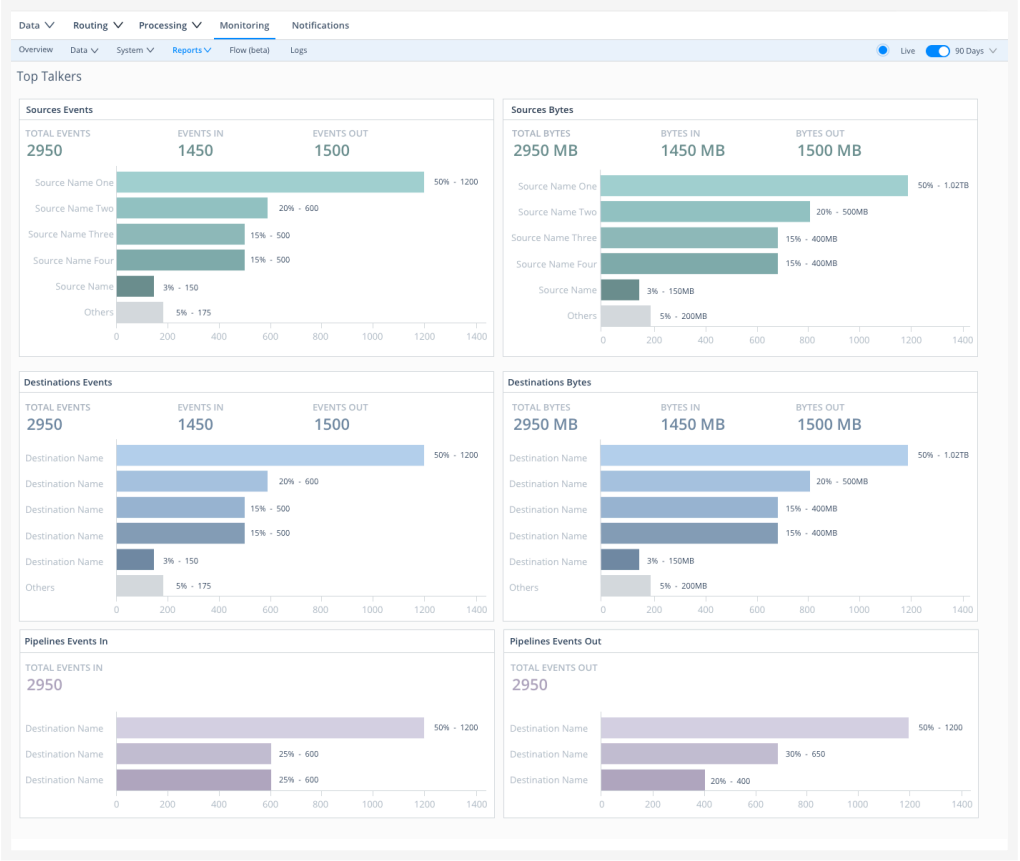
Licensing breakdown from the Monitoring page

Even on single-instance deployments, you must have `git` installed in order for the Monitoring > **Licensing** page to display configuration change markers.

For the most current and accurate throughput data, enable the Cribl Internal Source's CriblMetrics option. Forward metrics to your metrics Destination of choice, and run a report on `cribl.total.in_bytes`. CriblMetrics aggregates metrics at the Worker Process level, every 2 seconds.

Reports

Select **Monitoring** > **Reports / Top Talkers** > **Top Talkers**, where you can examine your five highest-volume Sources, Destinations, Pipelines, Routes, and Packs. Cribl Stream ranks all components by events throughput. Sources and Destinations get separate rankings by bytes in and out, respectively.



Monitoring > Reports

Flows

Select **Flows** from the Monitoring page's submenu or ••• overflow menu to see a graphical, left-to-right visualization of data flow through your Cribl Stream deployment.

Logs

Select **Logs** from the Monitoring page submenu to access the log viewer. This view lists the internal logs generated by Cribl Stream, which are records of various system events. Use these logs to monitor the health of your Cribl Stream deployment and troubleshoot operational issues.

Each log is structured data containing specific fields about the event that generated it. See Internal Logs for detailed information about the available logs and fields.

You can filter and analyze the list of log events to find specific data:

- **Filter by log type:** Use the main menu to select a type of log to view (such as the API Process, Access, Audit).
- **Search with expressions:** Enter a JavaScript expression in the main search bar to filter logs.
- **Interact with log events:** On the list of logs, you can select an event from an individual field to:
    - Add or exclude that field from the search bar.

- Copy the event value to your clipboard.

Notifications

Notifications alert Cribl Stream admins about issues that require their immediate attention, such as alerts for Sources and Destinations and upcoming license expiration dates. You can view all Notification events on this tab. See Notifications for more information about available Notifications and how to configure them.

Types of Logs

Cribl Stream provides the following log types, by originating process:

- API Server Logs - The `API/main` process is the primary source for these logs. They correspond to the top-level `cribl.log` that shows up on the Diag page. These include telemetry/license-validation logs. Filesystem location: `$CRIBL_HOME/log/cribl.log`
- Worker Process(es) Logs - These logs are emitted by all the Worker Processes, and are common on single-instance deployments and Workers. Filesystem location: `$CRIBL_HOME/log/worker/N/cribl.log`
- Worker Group/Fleet Logs - These logs are emitted by all processes that help a Leader Node configure Worker Groups/Fleets. Filesystem location: `$CRIBL_HOME/log/group/GROUPNAME/cribl.log`

In a distributed deployment (Stream), all Workers forward their metrics to the Leader Node, which then consolidates them to provide a deployment-wide view.

For details about generated log files, see Internal Logs. To work with logs in Cribl Stream's UI, see Search Internal Logs.

Log Rotation and Retention

Cribl Stream rotates logs every 5 MB, keeping the most recent 5 logs. Logs will reach the 5 MB threshold more quickly with verbose logging levels (such as `debug`) and with high volumes of system activity.

For long-term retention, Cribl recommends sending logs from the Leader, and from Workers' Cribl Internal > CriblLogs Sources, to a downstream service of your choice. See the next section for details.

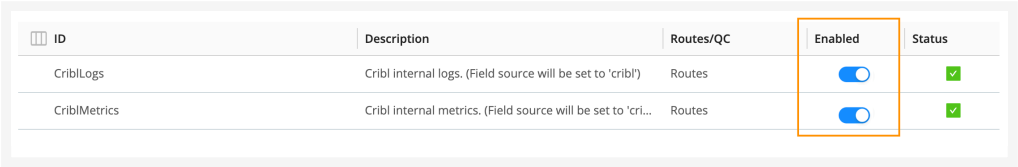
Forward Logs and Metrics Externally

Cribl Stream supports forwarding internal logs and metrics to your preferred external monitoring solution. The Cribl Internal Source captures internal logs and metrics to send to Destinations.

In the following steps, we'll use the graphical QuickConnect UI to set up the **Cribl Internal** Source and how data is sent to your Destination. See Cribl Internal for details on how to instead configure the **Cribl Internal** Source

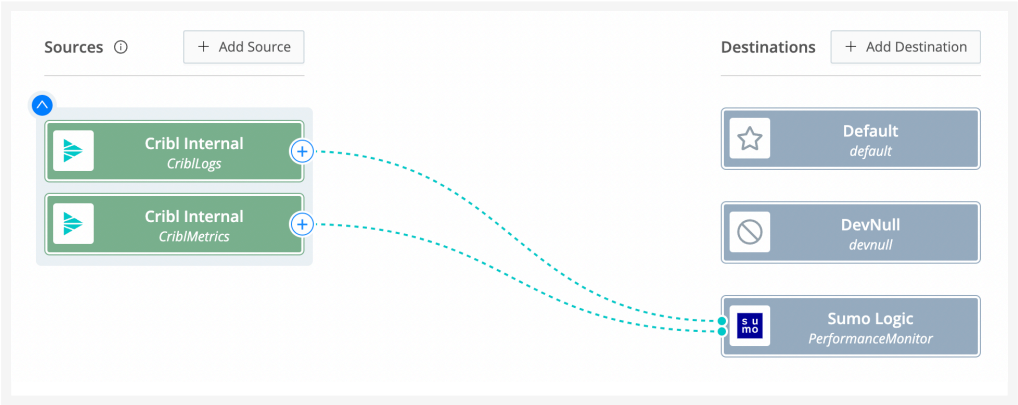
via the **Data** > **Sources** (Stream) or **More** > **Sources** (Edge) submenus.

1. On the top bar, select Products, and then select **Cribl Stream**. Then select a **Worker Group** to configure.
2. To open **QuickConnect**: Stream - select the **QuickConnect** tile on the **Overview** page. Edge - select the **Collect** submenu.
3. Select **Add Source**.
4. From the resulting drawer's tiles, select **System and Internal**, then hover over the **Cribl Internal** tile.
5. Select **Select Existing**.
6. On the **CriblLogs** and/or the **CriblMetrics** row, toggle **Enabled** on. Confirm your choice in the resulting message box.



Cribl Internal Sources - Select to configure

7. Notice how the **Routes/QC** column still says **Routes**. We want to use QuickConnect, so we'll change this by clicking on the **CriblLogs** and/or the **CriblMetrics** row again. Once clicked, you'll confirm **Yes** to the message box asking to switch the Source to send to QuickConnect instead of Routes.
8. Select and drag the **+** button on the right side of the Source to your desired Destination. A **Connection Configuration** modal will prompt you to select a **Passthru**, **Pipeline**, or **Pack connection**. See QuickConnect for details.



Cribl Internal forwarding to Sumo Logic

9. This will send internal logs and metrics to your Destination, just like another data Source. Both logs and metrics will have a field called `source`, set to the value `cribl`, which you can use in Routes' filters.

Note that the only logs supported here are Worker Process logs (see Types of Logs above). You can, however, use a Script Collector to listen for API Server or Worker Group events.

For recommendations about useful Cribl metrics to monitor, see Internal Metrics.

Controlling Metrics Volume

To reduce the volume of metrics sent through Cribl Stream, see options on the Cribl Internal Source.

Disable Field Metrics

To send fewer metrics to the Leader Node, you can disable specific metrics types.

By default, Cribl Stream does not send metrics for the `host`, `source`, `sourcetype`, `index`, and `project` fields. You can remove any of these defaults, or add other fields you do not want to send as metrics.

To do this, specify the fields in the **Disable field metrics** blocklist in **two** places:

- In **Settings** > **Global** > **System** > **General Settings** > **Limits** > **Metrics**.
- Per Worker Group, in **Worker Group Settings** > **System** > **General Settings** > **Limits** > **Metrics**.

However, when you enable the **Cribl Internal** Source, Cribl Stream ignores this **Disable field metrics** setting, and full-fidelity data will flow down the Routes.

Dropping Metrics

When the number of in-flight requests for sending metrics from Worker to Leader exceeds a limit (1,000 requests), Workers will stop sending metrics. Check `channel="clustercomm"` for a log of dropped metrics if needed.

You can exclude certain metrics from being dropped due to exceeding limits. Specify these metrics at **Settings** > **System** > **General Settings** > **Limits** > **Metrics** > **Metrics never-drop list**.

This setting is available only on Worker Nodes.

Metrics Garbage Collection

Metrics garbage collection (GC) runs when the total number of stored metrics exceeds the max metrics limit (default 1,000,000). Metrics are then removed, starting with the oldest ones. This happens both on Workers and Leaders.

You can define how often to run garbage collection on each Worker Group. Select **Group Settings** > **System** > **General Settings** > **Limits** > **Metrics** > **Metrics GC period**. The default is 60 seconds.

Metrics Tracking by Worker ID

Typically, Cribl Stream assigns all metrics their own Worker Node ID dimensions so they can be split by Worker if needed.

You can define which metrics are **not** assigned a Worker Node ID dimension by adding them to the list at **Group Settings** > **System** > **General Settings** > **Limits** > **Metrics** > **Metrics worker tracking**.

Control Metrics Lag and Disk Usage

If one or more Worker Groups has a large number of enabled Sources, and clicking into these Sources does not promptly display their status, a workaround is to prevent Cribl Stream from writing metrics to disk. On the Leader (versions 4.1.2 and newer), navigate to **Settings** > **Global Settings** > **System** > **General Settings** > **Limits** > **Storage**, and disable the **Persist metrics** toggle. Then restart the Leader.

If you keep **Persist metrics** enabled, you can use the adjacent **Metrics max disk space** field to control the written metrics' footprint. This threshold defaults to `64 GB`.

Cardinality Limit

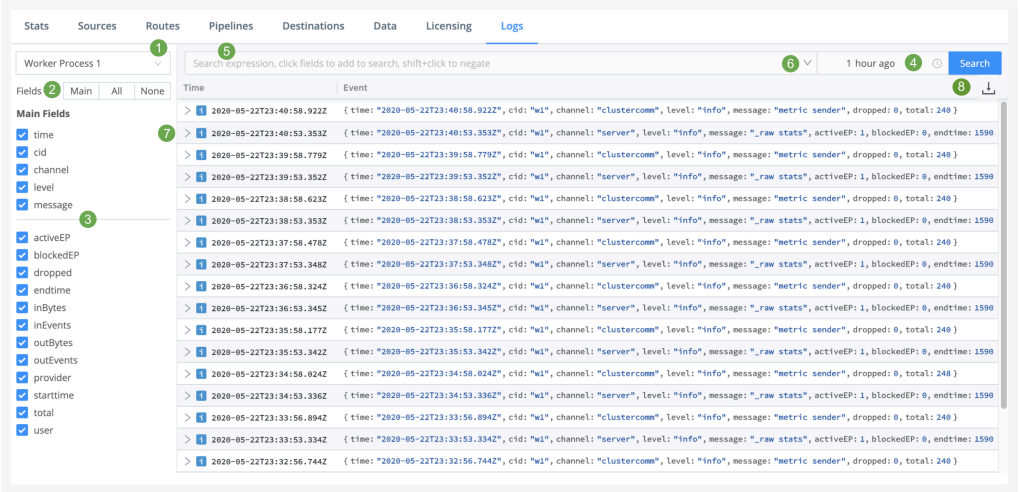
You can define the cardinality limit or metrics on each Worker Group. Select **Group Settings** > **System** > **General Settings** > **Limits** > **Metrics** > **Metrics cardinality limit** The default is 1,000.

Refer to **Monitoring** > **Data** > **Data Fields** to identify the fields that have the highest cardinality.

Search Internal Logs

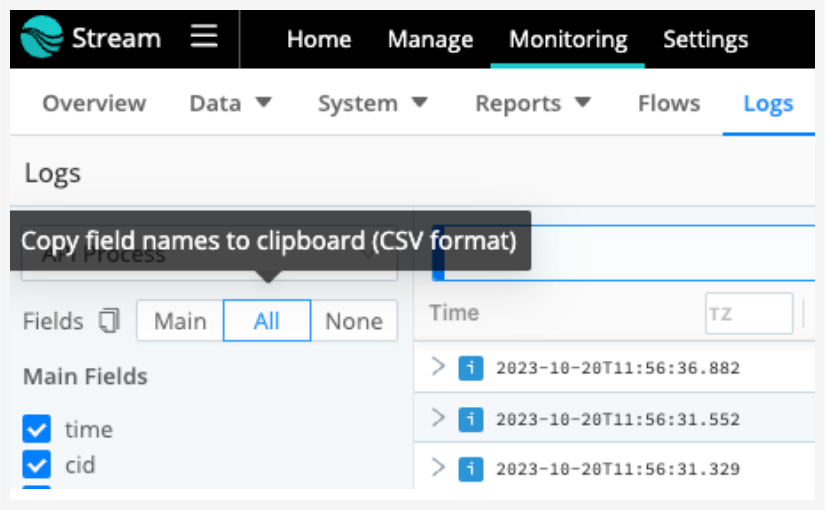
Cribl Stream exists because logs are great and wonderful things! Using Cribl Stream's **Monitoring > Logs** page, you can search all Cribl Stream's internal logs at once - from a single location, for both Leader and Worker. This enables you to query across all internal logs for strings of interest.

The labels on this screenshot highlight the key controls you can use (see the descriptions below):



Logs page (controls highlighted)

1. **Log file selector**: Choose the Node to view. In a distributed deployment (Stream), this list will be hierarchical, with Workers displayed inside their Leader.
2. **Fields selector**: Click the **Main** | **All** | **None** toggles to quickly select or deselect multiple check boxes below. Beside these toggles, a Copy button enables you to copy field names to the clipboard in CSV format.



Monitoring - Copy Fields Icon

3. **Fields**: Select or deselect these check boxes to change the columns displayed in the Results pane at right. (The upper **Main Fields** group will contain data for *every* event; other fields might not display data for all events.)
4. **Time range selector**: Select a standard or custom range of log data to display. (The **Custom Time Range** pickers use your local time, even though the logs' timestamps are in UTC.)
5. **Search box**: To limit the displayed results, enter a JavaScript expression here. An expression must evaluate to `truthy` to return results. You can press **Shift+Enter** to insert a newline.

Typeahead assist is available for expression completion:

Expression autocompletion

Click a field in any event to add it to a query:

Click to add a field

Click other fields to append them to a query:

Click to append a field

Shift+click to *negate* a field:

Shift-click to negate a field

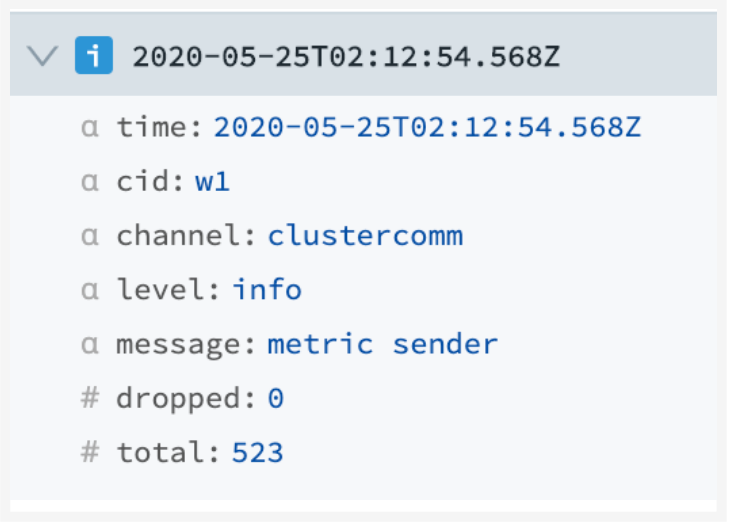
To modify the depth of information that is originally input to the Logs page, see Logging Settings.

6. Click the Search box's history arrow (right side) to retrieve recent queries:

Retrieve recent searches

7. The Results pane displays most-recent events first. Each event's icon is color-coded to match the event's severity level.

Click individual log events to unwrap an expanded view of their fields:



Log event details

8. **Export Logs as JSON** button: Exports logs as a file initially named `CriblMonitoringLogs.json`. (You can edit this name upon export.) The logs' scope will be filtered both by your Time range selector setting, and by how fully you've scrolled down to lazy-load that time range into the displayed UI.

Logging Settings

On Cribl Stream's Settings pages, you can adjust the level (verbosity) of internal logging data processed, per logging channel. You can also redact fields in customized ways. In a Distributed Deployment, you manage each of these settings per Worker Group.

The logging levels help to categorize the detail and severity of logged information. Choosing the right level ensures you get the information you need for troubleshooting without getting overwhelmed by a flood of data.

Here's a breakdown of each logging level:

Code	Description
`critical`	Logs unexpected failures that need to be resolved immediately.
`error`	Logs only errors which need immediate attention to ensure the system's integrity.
`warn`	Logs warnings about potential issues that might not be critical but warrant attention to prevent future problems.
`info`	Logs general information about the system's operation, providing insights into standard processes and procedures.
`debug`	Logs detailed information for diagnosing and troubleshooting potential problems, providing deep insights into the system's behavior.
`silly`	Logs extremely detailed information and metrics, primarily used for in-depth troubleshooting of inputs and other operations.Use it sparingly and only for short debugging bursts, treating it as "use at your own risk." Enable temporarily, then disable promptly.

By default, all the integration logging levels are set to `info`. This default setting displays logs whose level is `info` and more-severe (`warn` and `error`). To expose more-verbose `debug` and `silly` logs, adjust this setting per integration.

> In Cribl.Cloud, you can't directly change the logging level on Cribl-managed Worker Groups or their integrations. These levels take the default `info` level. To adjust these levels, please contact Cribl Support, who will engage the appropriate team to make the necessary changes for you.

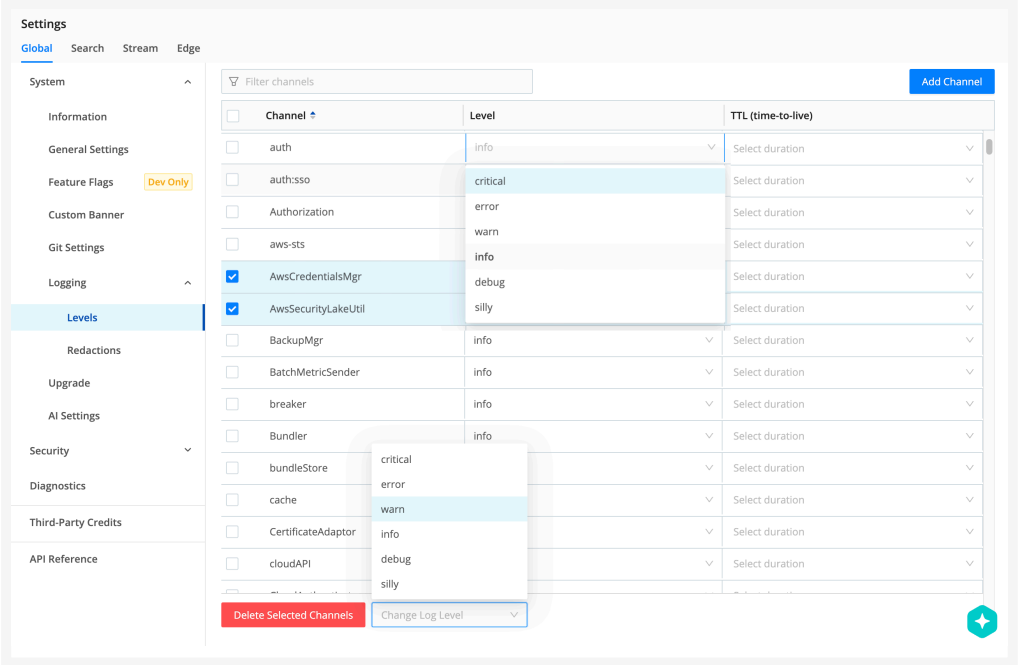
Change Logging Levels

To adjust logging levels:

- In a single-instance deployment, select **Settings** > **System** > **Logging** > **Levels**.
- For the Leader Node's own logs, select **Settings** > **Global Settings** > **System** > **Logging** > **Levels**.
- In a distributed deployment, select the group you want to configure. Next, select **Worker Group/Fleet Settings** > **System** > **Logging** > **Levels**.

On the resulting **Levels** page, you can:

- Modify one channel by clicking its **Level** column. In the resulting drop-down, you can set a verbosity level ranging from **critical** up to **silly**. (Top of composite screenshot below.)
- Modify multiple channels by selecting their check boxes, then clicking the **Change log level** drop-down at the bottom of the page. (Bottom of composite screenshot below.) You can select all channels at once by clicking the top check box. You can search for channels at top right.
- Set temporary logging levels for one or more channels.



Levels page for managing logging levels

Set Temporary Logging Levels

The **TTL (time-to-live)** setting allows you to temporarily change log levels so that you can debug without overwhelming your logs or incurring unnecessary storage costs. You can set temporary log levels for up to 24 hours.

To set temporary logging levels for logging channels:

1. Navigate to the **Levels** page.
2. Modify the logging level for the desired channels using either the **Level** column or the **Change log level** drop-down at the bottom of the page.

3. For each modified channel, select a duration for the temporary logging level in the **TTL (time-to-live)** column.
4. Select **Save**. The **TTL (time-to-live)** setting applies instantly.

Make sure to set temporary log levels for individual channels *after* you save any settings for the **DEFAULT** channel. The **DEFAULT** channel has special override behavior that affects log level settings for all other channels as follows:

- On Cribl.Cloud and in on-prem deployments, setting a temporary log level for **DEFAULT** overrides all channel levels, including existing TTLs on other channels.
- In on-prem deployments, setting a permanent log level (no **TTL (time-to-live)** duration) for **DEFAULT** overrides all log levels and deletes any existing temporary log level settings on other channels.
- Cribl.Cloud does not support changing the permanent log level for the **DEFAULT** channel when any individual channels have a temporary log level setting.

Log levels automatically revert to previous settings when the **TTL (time-to-live)** duration expires, falling back as needed according to the following order of precedence:

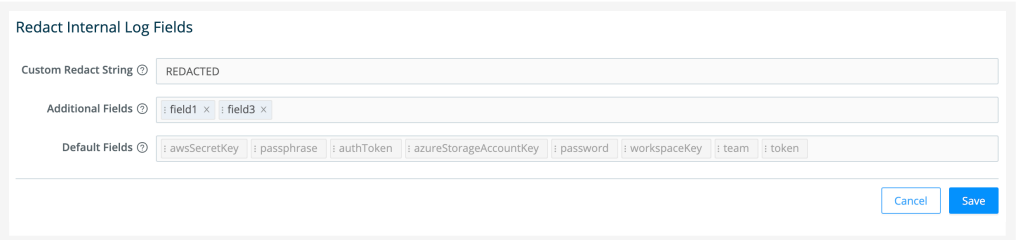
1. The last log level that was set for the channel without an associated **TTL (time-to-live)** duration.
2. The log level set for the **DEFAULT** channel.
3. The `info` log level.

The **TTL (time-to-live)** setting uses timestamps in Unix time for persistence across restarts. Temporary log level settings do not permanently update persisted configurations for logging channels.

Change Logging Redactions

On the **Redact Internal Log Fields** page, you can customize the redaction of sensitive, verbose, or just ugly data within Cribl Stream's internal logs. To access these settings:

- In a single-instance deployment, select **Settings** > **System** > **Logging** > **Redactions**.
- In a single-instance deployment, or for the Leader Node's own logs, select **Settings** > **Global Settings** > **System** > **Logging** > **Redactions**.
- In a distributed deployment, select the group you want to configure. Next, select that group's **Worker Group/Fleet Settings** > **System** > **Logging** > **Redactions**.



Redact Internal Log Fields page

It's easiest to understand the resulting **Redact Internal Log Fields** page's fields from bottom to top:

- **Default fields**: Cribl Stream always redacts these fields, and you can't modify this list to allow any of them through. However, you can use the two adjacent fields to define stricter redaction:
- **Additional fields**: Type or paste in the names of extra fields you want to redact. Use a tab or hard return to confirm each entry.
- **Custom redact string**: Unless this field is empty, it defines a literal string that will override Cribl Stream's default redaction pattern (explained below) on the affected fields.

Default Redact String

By default, Cribl Stream transforms this page's selected fields by applying the following redaction pattern:

- Echo the field value's first two characters.

5.1. Internal Metrics

When sending Cribl Stream metrics to a metric system of analysis - such as InfluxDB, Splunk, or Elasticsearch - some metrics are particularly valuable. You can use these metrics to set up alerts when Worker Nodes are having problems, a Node is down, a Destination is down, a Source stops providing incoming data, and so on.

Cribl Stream reports its internal metrics within the Cribl Stream UI, in the same way that it reports internal logs. Navigate to **Monitoring** > **Logs** (Cribl Stream), or to a Fleet's **Health** > **Sources** tab (Cribl Edge).

To expose metrics for capture or routing, enable **CriblMetrics** on the Cribl Internal Source.

You can also export metrics (and logs) to JSON files.

By default, Cribl Stream generates internal metrics every 2 seconds. To consume metrics at longer intervals, you can use or adapt the `cribl-metrics_rollup` Pipeline that ships with Cribl Stream. Attach it to your **Cribl Internal** Source as a pre-processing Pipeline. The Pipeline's **Rollup Metrics** Function has a default **Time Window** of 30 seconds, which you can adjust to a different granularity as needed.

You can also use our public endpoints to automate monitoring using your own external tools.

Counter-type metrics in Cribl Stream do not monotonically increase or decrease. They are reset at the end of each reporting period. Cribl Stream does not report counters when their value is 0. For example, if there aren't any Destinations reporting dropped events then the `total.dropped_events` metric won't be reported because its value would be 0.

Breaking Changes

- The `ci` and `co` dimensions are deprecated as of Cribl Stream 4.1.2. See Dimensions for their replacements.
- Disabled Sources no longer generate Cribl internal metrics as of Cribl Stream 4.2.

Total Throughput Metrics

Five important metrics below are prefixed with `total.` These power the top of Cribl Stream's **Monitoring** Dashboard. The first two report on Sources, the remainder on Destinations.

- `total.in_bytes`
- `total.in_events`
- `total.out_events`
- `total.out_bytes`

- `total.dropped_events` - helpful for discovering situations such as: you've disabled a Destination without noticing.

Interpreting Total Metrics

These `total.` metrics' values could reflect Cribl Stream's health, but could also report low activity simply due to the Source system. For example, logs from a store site will be low at low buying periods.

Also, despite the `total.` prefix, these metrics are each specific to the Worker/Edge Node Process that's generating them.

You can distinguish **unique** metrics by their `#input=<id>` dimension. For example, `total.in_events|#input=foo` would be one unique metric, while `total.in_events|#input=bar` would be another.

System Health Metrics

Health metrics are valuable for monitoring system health.

Cribl Stream composite metrics:

- `health.inputs`
- `health.outputs` - see the JSON Examples below for both `health.` metrics.

Hardware or VM infrastructure metrics:

- `system.load_avg`
- `system.free_mem`
- `system.disk_used` - valuable if you know your disk size, especially for monitoring Persistent Queues. Here, a `0` value typically indicates that the disk-usage data provider has not yet provided the metric with data. (Getting the first value should take about one minute.)

The Cribl Internal Source does not export these metrics to Routes or Pipelines, you can obtain them only by using the REST endpoints listed on this page.

All of the above metrics have these possible values:

- `0` = green = healthy.
- `1` = yellow = warning.
- `2` = red = trouble.

Health Inputs/Outputs JSON Examples

The `health.inputs` metrics are reported per Source, and the `health.outputs` metrics per Destination. The `health.inputs` example below has two configured Sources, and two Cribl Stream-internal inputs. The `health.outputs` example includes the built-in `devnull` Destination, and six user-configured Destinations.

Given all the `0` values here, everything is in good shape!

```
"health.inputs": [
  { "model": { "input": "http:http" }}, "val": 0},
  { "model": { "input": "cribl:CriblLogs" }}, "val": 0},
  { "model": { "input": "cribl:CriblMetrics" }}, "val": 0},
  { "model": { "input": "datagen:DatagenWeblog" }}, "val": 0 }
  ],
"health.outputs": [
  { "model": { "output": "devnull:devnull" }}, "val": 0},
  { "model": { "output": "router:MyOut1" }}, "val": 0},
  { "model": { "output": "tcpjson:MyTcpOut1" }}, "val": 0},
  { "model": { "output": "router:MyOut2" }}, "val": 0},
  { "model": { "output": "tcpjson:MyTcpOut2" }}, "val": 0},
  { "model": { "output": "router:MyOut3" }}, "val": 0},
  { "model": { "output": "router:MyOut4" }}, "val": 0 }
  ],
```

Copy

Worker Node Resource Metrics

The Cribl Internal Source reports two useful metrics on the resource usage of individual Worker Node Processes:

- `system.cpu_perc` - CPU percentage usage.
- `system.mem_rss` - RAM usage in bytes.

Worker Node Shutdown Metric

This metric tracks events lost during Worker Process shutdown:

- `shutdown.lost_events` - Number of events lost because they were not flushed before the **Drain timeout** elapsed during Worker Process shutdown, as explained on the Shutdown and Restart Sequence docs.

Persistent Queue Metrics

These metrics are valuable for monitoring persistent queues' behavior:

- `pq.queue_size` - Total bytes in the queue.
- `pq.in_bytes` - Total bytes in the queue for the given **Time Window**.
- `pq.in_events` - Number of events in the queue for the given **Time Window**.
- `pq.out_bytes` - Total bytes flushed from the queue for the given **Time Window**.
- `pq.out_events` - Number of events flushed from the queue for the given **Time Window**.

These are aggregate metrics, but you can distinguish unique metrics per queue. On the Destination side, use the `#output=<id>` dimension. For example, `pq.out_events|#output=kafka` would be one unique metric; `pq.out_events|#output=camus` would be another.

Integration Metrics

Integration metrics provide enhanced observability into the performance and health of Sources and Destinations. See how to control the granularity of integration metrics.

By exposing internal metrics for data flow, the platform helps you understand ingestion rates, queue behavior, connection health, and back-pressure. Depending on your metric granularity level, these I/O metrics provide sufficient signal to troubleshoot slow ingestion, rejected connections, and overloaded Nodes without requiring deep, log-level investigation.

For I/O observability, Cribl Stream uses Detailed metrics by default for all integrations (Sources and Destinations) that support I/O metrics. This standard applies to both Cribl.Cloud and on-prem Organizations.

View Integration Metrics

Metrics are aggregated every minute for timely monitoring. You can monitor integration metrics in three ways:

- **Logs Tab**: In the Sources and Destinations configuration modal, use the dropdown menu to select a specific metrics channel and view the details.
- **Charts Tab**: Performance charts are displayed directly on each supported Source and Destination configuration modal, offering immediate visualization and trend analysis.
- **Through the Cribl Internal Source (CriblMetrics)**: You can query the metrics and build automations using the Internal Cribl Source.

Available Metrics

The following metrics provide a detailed view of Source and Destination performance, covering connection status, request activity, latency, and data throughput:

- `logged.criticals` - Number of criticals logged to the channel.
- `logged.errors` - Number of errors logged to the channel.
- `iometrics.activeCxn` - Number of active inbound connections per Source.
- `iometrics.closeCxn` - Number of closed connections per Source.
- `iometrics.openCxn` - Number of opened connections per Source.
- `iometrics.rejectCxn` - Number of rejected connections per Source. Includes the `reject_reason` dimension (such as `concurrency_limit`, `high_load`, or `ip_validation_failed`).
- `iometrics.consumer_lag` - Offset between the last message in a partition and the consumer's position (Kafka-based Sources only).
- `iometrics.endpoints_healthy_percentage` - Percentage of healthy load-balanced endpoints.
- `iometrics.p95_duration_millis` - 95th percentile of request processing time.
- `iometrics.p99_duration_millis` - 99th percentile of request processing time.
- `iometrics.total_requests` - Number of sent/received requests.
- `iometrics.failed_requests` - Number of failed requests.
- `iometrics.ingest_bps_active` - Histogram of ingestion throughput rate (Bps) for active TCP sockets, S3 file transfers, or HTTP-based sources.
- `iometrics.ingest_bps_closed` – Histogram of lifetime ingestion throughput rate (Bps) for closed TCP sockets and HTTP-based sources.
- `source.in_bytes` - Inbound bytes per Source.
- `source.in_events` - Inbound events per Source.
- `source.out_bytes` - Outbound bytes per Source.
- `source.out_events` - Outbound events per Source.
- `sourcetype.in_bytes` - Inbound bytes per Source type.
- `sourcetype.in_events` - Inbound events per Source type.
- `sourcetype.out_bytes` - Outbound bytes per Source type.
- `sourcetype.out_events` - Outbound events per Source type.
- `total.activeCxn` - Total number of active inbound connections from all HTTP and TCP-based Sources.
- `total.closeCxn` - Total number of closed connections from all HTTP and TCP-based Sources.
- `total.openCxn` - Total number of opened connections from all HTTP and TCP-based Sources.
- `total.rejectCxn` - Total number of rejected connections from all HTTP and TCP-based Sources.

Other Internal Metrics

The Cribl Internal Source emits other metrics that, when displayed in downstream dashboards, can be useful for understanding Cribl Stream's behavior and health. These include:

- `pipe.in_events` - Inbound events per Pipeline.
- `pipe.out_events` - Outbound events per Pipeline.
- `pipe.dropped_events` - Dropped events per Pipeline.
- `metrics_pool.num_metrics` - The total number of unique metrics that have been allocated into memory.
- `collector_cache.size` - Each Collector function (`default/cribl/collectors/<collector>/index.js`) is loaded/initialized only once per job, and then cached. This metric represents the current size of this cache.
- `cluster.metrics.sender.inflight` - Number of metric packets currently being sent from a Worker/Edge Node Process to the API Process, via IPC (interprocess communication).
- `backpressure.outputs` - Destinations experiencing backpressure, causing events to be either blocked or dropped.
- `blocked.outputs` - Blocked Destinations. (This metric is more restrictive than the one listed just above.)
- `pq.queue_size` - Current queue size, in bytes, per Worker Process.
- `host.in_bytes` - Inbound bytes from a given host (host is a characteristic of the data).
- `host.in_events` - Inbound events from a given host (host is a characteristic of the data).
- `host.out_bytes` - Outbound bytes from a given host (host is a characteristic of the data).
- `host.out_events` - Outbound events from a given host (host is a characteristic of the data).
- `index.in_bytes` - Inbound bytes per index.
- `index.in_events` - Inbound events per index.
- `index.out_bytes` - Outbound bytes per index.
- `index.out_events` - Outbound events per index.
- `route.in_bytes` - Inbound bytes per Route.
- `route.in_events` - Inbound events per Route.
- `route.out_bytes` - Outbound bytes per Route.
- `route.out_events` - Outbound events per Route.
- `nodecluster.primary.messages` - Number of IPC (interprocess communication) calls made by a Worker/Edge Node Process to the API Process.
- `cribl.logstream.project.in_events` - Inbound events per Project.
- `cribl.logstream.project.out_events` - Outbound events per Project.

- `cribl.logstream.project.dropped_events` - Dropped events per Project.
- `cribl.logstream.project.in_bytes` - Inbound bytes per Project.
- `cribl.logstream.project.out_byte` - Outbound bytes per Project.
- `cribl.logstream.subscription.in_bytes` - Inbound bytes per Subscription.
- `cribl.logstream.subscription.out_bytes` - Outbound bytes per Subscription.
- `cribl.logstream.subscription.in_events` - Inbound events per Subscription.
- `cribl.logstream.subscription.out_events` - Outbound events per Subscription.

Dimensions

Cribl Stream internal metrics have extra dimensions. These include:

- `cribl_wp` - Identifies the Worker/Edge Node Process that processed the event.
- `event_host` - Machine's hostname from which the event was made.
- `event_source` - Set as `cribl`.
- `failure_reason` - Whether the failed request is `retryable` or `non_retryable`.
- `group` - Name of Cribl Worker Group from which the event was made.
- `input` - Entire **Input ID**, including the prefix, from which the event was made.
- `output` - Entire **Output ID**, including the prefix, from which the event was made.
- `status_code` - For failed requests, the status code for the error (`4xx` or `5xx`).
- `reject_reason` - Reason for a rejected connection.
- `ci` - Deprecated; use `input` instead. Cribl plans to drop support for this dimension after version 4.x.
- `co` - Deprecated; use `output` instead. Cribl plans to drop support for this dimension after version 4.x.

Control Metrics

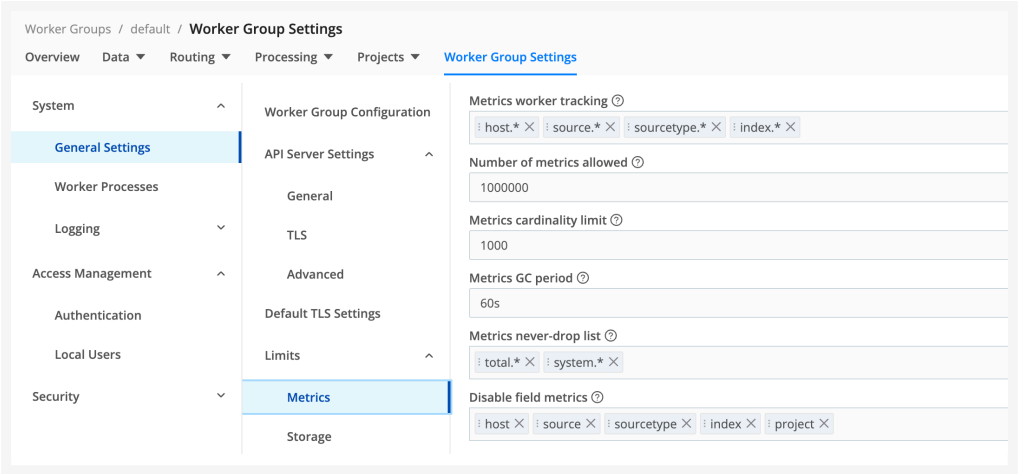
You can control the volume of internal metrics that Cribl Stream emits, by adjusting metrics limits. You access these as follows, per deployment type:

- Single-instance: **Settings** > **System** > **General Settings** > **Limits** > **Metrics**.
- Distributed Leader: **Settings** > **Global Settings** > **System** > **General Settings** > **Limits** > **Metrics**.
- Distributed Worker Groups: [Select a Worker Group] > **Worker Group/Fleet Settings** > **System** > **General Settings** > **Limits** > **Metrics**.

Here, you can adjust the maximum number of metric series allowed, metrics' cardinality, metrics to always include (**Metrics never-drop list**), metrics to always exclude in order to optimize performance (**Disable field**

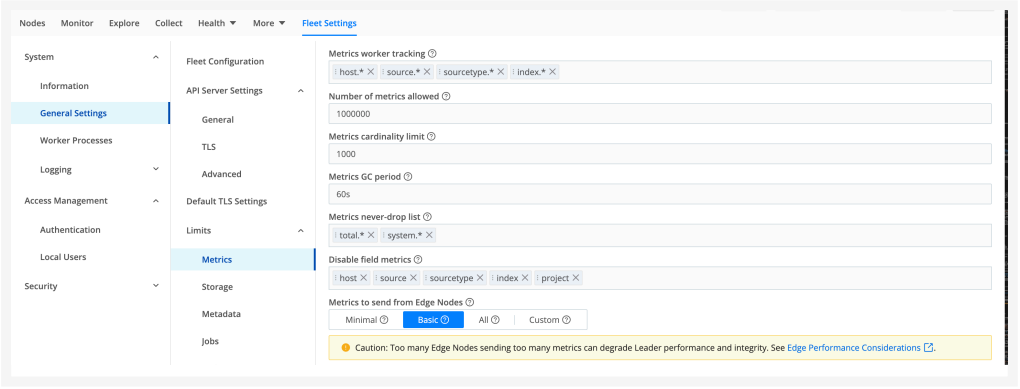
metrics), and other options.

Here's how these controls look in Cribl Stream:



Metrics settings for Cribl Stream Worker Groups

They're somewhat different in Cribl Edge - see the next section for an explanation:



Metrics settings for Cribl Edge Fleets

Control Integration Metrics in Cribl Stream

On Cribl.Cloud instances, you can control the granularity of integration metrics emitted by the Internal Metrics Source to optimize observability while managing system performance and resource usage.

Access the **Metrics Levels** setting by navigating to the **Worker Group** > **Worker Group/Fleet Settings** > **System** > **Metrics Levels**.

Only `Admin` and `Editor` roles can view or modify these settings.

On the **Metrics Limits** page, you'll find a table listing all configured integrations. For each Source or Destination, you can select the desired metrics level from a dropdown.

Level	Purpose	List of Metrics Sent

Minimal	Reports total throughput metrics, offering a high-level view of overall data flow without integration-specific details. Ideal for low-cardinality environments.	`logged.errors`, `logged.criticals`
		
		`total.openCxn`, `total.closeCxn`, `total.activeCxn`, `total.in_events`, `total.out_events`, `total.in_bytes`, `total.out_bytes`, `total.events`, and `total.bytes`
		
		`iometrics.endpoints_healthy_percentage`, `iometrics.openCxn`, `iometrics.closeCxn`, `iometrics.activeCxn`, and `iometrics.rejectCxn`.

Basic (default)	Includes health metrics, emphasizing error tracking, endpoint health, and success rates. Designed for monitoring the operational state of integrations.	`health.inputs` `iometrics.total_requests`, and `iometrics.failed_requests`.
Detailed	Includes latency metrics, enabling precise performance analysis by capturing percentile-based processing times. **Warning**: This level increases system load.	`iometrics.p95_duration_millis` and `iometrics.p99_duration_millis`.

Control Metrics in Cribl Edge

Cribl Edge lets you choose what metrics each Fleet sends to its Leader Node. Your goal should be to collect the metrics most important to you, within the limits of the Leader's capacity to process them. This is a matter of balancing Fleet cardinality and Leader performance, as explained in the Deployment Planning topic.

Use the **Metrics to send from Edge Nodes** buttons to select one of the metrics sets listed in the table below.

Set	Purpose	List of Metrics Sent
Minimal	Sends metrics for events and bytes in and out. Recommended for high-cardinality deployments.	Sent as aggregate totals per Fleet: `total.in_events`, `total.out_events`, `total.in_bytes`, `total.out_bytes` excluding metrics that are coming directly from Sources or Destinations.
Basic (default)	Contains all the metrics in the minimal set, and more. Sends metrics required to display all monitoring data available in the UI.	Sent as aggregate totals per Source/Destination: `total.in_events`, `total.out_events`, `total.in_bytes`, `total.out_bytes`. Also, Sent as aggregate totals per Source:

Set	Purpose	List of Metrics Sent

		`health.inputs`
		
		Sent as aggregate totals per Destination:
		
		`health.outputs`
		
		Sent as aggregate totals per Route:
		
		`route.in_events`, `route.in_bytes`, `route.out_events`, `route.out_bytes`, `route.dropped_events`
		
		Sent as aggregate totals per Pipeline:
		
		`pipe.in_events`, `pipe.out_events`, `pipe.dropped_events`
		
		Sent as aggregate totals per Pack:
		
		`pack.in_events`, `pack.out_events`, `pack.err_events`, `pack.dropped_events`

All	Send all metrics with no filtering.	All metrics without filtering, including **Basic**, plus Worker/Edge Node Resource Metrics, Persistent Queue Metrics, and Other Internal Metrics.
Custom	Send a customized set of metrics.	All metrics that match a JavaScript expression that you define, plus the metrics in the **Minimal** set.

Basic is the default selection, and includes the **Minimal** set of metrics. Every Edge Fleet always sends at least **Minimal** metrics, and selecting one of the other options adds more metrics.

Choosing the **Minimal** metrics set has the following effects in the Edge UI:

- Only one UI page, **Edge Home** > **Monitor**, will be fully populated. For the metrics that the **Minimal** set excludes, other pages will show zero values. This applies to the **[Select a Fleet]** > **Health** pages for **Sources**, **Destinations**, **Routes**, **Pipelines**, and **Packs**.
- In the **[Select a Fleet]** > **More** pages for **Sources** and **Destinations**, the health indicators in the tile and list views do not function when you select the **Minimal** set. Rather than red, green, or gray, they'll always show white, because the relevant metrics, `health.inputs` and `health.outputs`, will not be sent to the Leader.

The simplest form of a **Custom** set just specifies additional desired metrics by name, using an expression of the form: `name === "metric.name"`. Among the most useful metrics to consider adding are `system.load_avg`, `system.free_mem`, `system.disk_used`, `system.cpu_perc`, and `system.mem_rss`.

Custom Metrics and Their Impact on Monitoring

Enabling custom metrics switches monitoring from **Basic** to **Minimal**. This hides monitoring data for Fleets and Edge Nodes, giving the impression that data flow has stopped. To ensure system monitoring features work as expected with custom metrics, include the **Basic** metrics set within the custom configuration.

Other Metrics Endpoints

Below is basic information on using the `/system/metrics` endpoint, the `/system/info` endpoint, and the `cribl_wp` dimension.

`/system/metrics` Endpoint

`/system/metrics` is Cribl Stream's primary public metrics endpoint, which returns most internal metrics. Many of these retrieved metrics report configuration only, not runtime behavior. For details, see our API Docs.

`/system/info` Endpoint

`/system/info` generates the JSON displayed in the Cribl Stream UI at **Settings** > **Global Settings** > **Diagnostics** > **System Info**. Its two most useful properties are `loadavg` and `memory`.

`loadavg` Example

```
"loadavg": [1.39599609375, 1.22265625, 1.31494140625],
```

This property is an array containing the 1-, 5-, and 15-minute load averages at the UNIX OS level. (On Windows, the return value is always `[0, 0, 0]`.) For details, see the Node.js os.loadavg() documentation.

`memory` Example

```
"memory": { "free": 936206336, "total": 16672968704 },
```

Divide `total` / `free` to monitor memory pressure. If the result exceeds 90%, this indicates a risky situation: you're running out of memory.

`cpus` Alternative

The `cpus` metric returns an array of CPU/memory key-value pairs. This provides an alternative way to understand CPU utilization, but it requires you to query all your CPUs individually.

5.2. Internal Logs

Cribl Stream generates internal application logs that monitor its own operations and health. They provide valuable insights into the system's behavior, performance, and potential issues.

Distributed deployments emit a larger set of logs than single-instance deployments. We'll describe the distributed set first.

You can display and export all internal logs by selecting **Monitoring** in the sidebar, then selecting **Logs** (Stream) or selecting **Logs** in the sidebar (Edge). Logs' persistence depends on event volume, not time - for details, see Log Rotation and Retention.

> Several logs listed on this page are exposed only in customer-managed (on-prem) deployments. In Cribl.Cloud, Leaders support Cribl Stream Worker Node logs on hybrid Workers.
>
> However, Organization Members who have the Admin Permission on Cribl Search can use that product to search the `cribl_internal_logs` Dataset for additional details about the Leader and its Cribl-managed Workers.

Leader Node Logs (Distributed)

The `API/main` process emits the following logs into the Leader Node's `$CRIBL_HOME/log/` directory.

Logfile Name	Description	Equivalent on Logs page
`cribl.log`	Principal log in Cribl Stream. Includes telemetry/license-validation logs. Corresponds to top-level `cribl.log` on Diag page.	**Leader** > **API Process**
`access.log`	API calls, for example, `GET /api/v1/version/info`.	**Leader** > **Access**
`audit.log`	Actions pertaining to files, for example, `create`, `update`, `commit`, `deploy`, `delete`.	**Leader** > **Audit**
`notifications.log`	Messages that appear in the Notification list in the UI. For each event, the `time` and `_time` fields record the start of the monitored interval (matching `starttime`), not when the Notification was written.	**Leader** > **Notifications**
`ui-access.log`	Interactions with different UI components described as URLs, for example, `/settings/apidocs`, `/dashboard/logs`.	**Leader** > **UI Access**

The `API/main` process emits the following service logs into the Leader Node's `$CRIBL_HOME/log/service/` directory. Each service includes a `cribl.log` file that logs the service's internal telemetry and an `access.log` file that logs which API calls the service has handled.

Service Name	Description	Equivalent on Logs page
Connections Service	Handles all Worker connections and communication, including heartbeats, bundle deploys, teleporting, restarting, and so on. Workers are assigned to connection processes using a round-robin algorithm.	**Leader** > **Connections Service**
Lease Renewal Service	Handles lease renewal for the primary Leader Node.	**Leader** > **Lease Renewal Service**
Metrics Service	Handles in-memory metrics, merging of incoming packets, metrics persistence and rehydration, and UI queries for metrics.	**Leader** > **Metrics Service**
Notifications Service	Triggers Notifications based on its configuration.	**Leader** > **Notifications Service**

The Config Helper process for each Worker Group/Fleet emits the following log in `$CRIBL_HOME/log/group/GROUPNAME`.

Logfile Name	Description	Equivalent on Logs page
`cribl.log`	Messages about config maintenance, previews, and so on.	**GROUPNAME** > **Config helper**

Outpost Logs

The API Process emits the following logs into the `$CRIBL_HOME/log/service/outpost/N/` directory, where `N` is the Outpost Node Process ID:

Logfile Name	Description	Equivalent on the Logs tab in Outpost info drawer
`cribl.log`	Internal telemetry logs.	**Outpost Service N**
`metrics.log`	Outpost Node metrics.	**Outpost Service N - Metrics**

Worker Node Logs (Distributed)

The API Process emits the following log in `$CRIBL_HOME/log/`.

Logfile Name	Description	Equivalent on Logs page
`cribl.log`	Messages about the Worker/Edge Node communicating with the Leader Node (i.e., with its API Process) and other API requests, for example, sending metrics, reaping job artifacts.	**GROUPNAME** > **Worker:HOSTNAME** > **API Process**

Each Worker Process emits the following logs in `$CRIBL_HOME/log/worker/N/`, where `N` is the Worker/Edge Node Process ID. (The `metrics.log` file is written only when HTTP-based Destinations exist.)

Logfile Name	Description	Equivalent on Logs page
`cribl.log`	Messages about the Worker/Edge Node processing data.	**GROUPNAME** > **Worker:HOSTNAME** > **Worker Process N**
`metrics.log`	Messages about the Worker/Edge Node's outbound HTTP request statistics.	**GROUPNAME** > **Worker:HOSTNAME** > **Worker Process N**

For convenience, the UI aggregates the Worker/Edge Node Process logs as follows.

Logfile Name	Description	Equivalent on Logs page
N/A	Aggregation of all the `Worker Process N` logs and the API Process log.	**GROUPNAME** > **WORKER_NAME**

In Cribl Stream, the logs listed above are currently available only on customer-managed hybrid Workers. The single-instance logs listed below are not relevant to Cribl.Cloud.

Single-Instance Logs

The `API/main` process emits the same logs as it does for a Distributed deployment, in `$CRIBL_HOME/log/`:

- `cribl.log`
- `access.log`
- `audit.log`
- `notifications.log`
- `ui-access.log`

Each Worker/Edge Node Process emits the following logs in `$CRIBL_HOME/log/worker/N/`, where `N` is the Worker/Edge Node Process ID. (The `metrics.log` file is written only when HTTP-based Destinations exist.)

Logfile Name	Description	Equivalent on Logs page
`cribl.log`	Messages about the Worker/Edge Node processing data.	**GROUPNAME** > **Worker:HOSTNAME** > **Worker Process N**
`metrics.log`	Messages about the Worker/Edge Node's outbound HTTP request statistics.	**GROUPNAME** > **Worker:HOSTNAME** > **Worker Process N**

`_raw stats` Event Fields

Each Worker/Edge Node Process logs this information at a 1-minute frequency. So each event's scope covers only that Worker/Edge Node Process, over a 1-minute time span defined by the `startTime` and `endTime` fields.

Sample Event


```
{"time":"2022-11-17T16:54:05.349Z","cid":"w0","channel":"server","level":"info","mess
```


Copy

Field Descriptions

Field	Description
`abortCxn`	Number of TCP connections that were aborted.
`activeCxn`	Number of TCP connections newly opened at the time the `_raw` stats are logged. (This is a gauge when exported in internal metrics, and can otherwise be ignored as an instantaneous measurement. Only some application protocols count toward this; for example, any HTTP-based Source does not count.)
`activeEP`	Number of currently active event processors (EPs). EPs are used to process events through Breakers and Pipelines as the events are received from Sources and sent to destinations. EPs are typically created per TCP connection (such as for HTTP).
`blockedEP`	Number of currently blocked event processors (caused by blocking Destinations).
`closeCxn`	Number of TCP connections that were closed.
`cpuPerc`	CPU utilization from the combined user and system activity over the last 60 seconds.
`droppedEvents`	This is equivalent to the `total.dropped_events` metric. Drops can occur from Pipeline Functions, from Destination Backpressure, or from other errors. Any event not sent to a Destination is considered dropped.

Field	Description
`eluPerc`	Event loop utilization. Represents the percentage of time over the last 60 seconds that the NodeJS runtime spent processing events within its event loop.
`endTime`	The end of the timespan represented by these metrics. (Will always be 60 seconds after `startTime`.)
`inBytes`	Number of bytes received from all Sources (based only off `_raw`).
`inEvents`	Number of events received from all inputs after Event Breakers are applied. This can be larger than `outEvents` if events are dropped via Drop, Aggregation, Suppression, Sampling, or similar Functions.
`mem.buffers`	Memory allocated for ArrayBuffers and SharedArrayBuffers.
`mem.ext`	External section of process memory, in MB.
`mem.heap`	Used heap section of process memory, in MB.
`mem.heapTotal`	Total heap section of process memory, in MB.
`mem.rss`	Resident set size section of process memory, in MB.
`openCxn`	Same as `activeCxn`, but tracked as a counter rather than a gauge. So `openCxn` will show all connections newly opened each minute, and is more accurate than using `activeCxn`.
`outBytes`	Number of bytes sent to all Destinations (based only off `_raw`).
`outEvents`	Number of events sent out to all Destinations. This can be larger than `inEvents` due to creating event clones or entirely new unique events (such as when using the Aggregation Function).
`pqInBytes`	Number of bytes that were written to persistent queues, across all Destinations.
`pqInEvents`	Number of events that were written to persistent queues, across all Destinations.
`pqOutBytes`	Number of bytes that were flushed from persistent queues, across all Destinations.
`pqOutEvents`	Number of events that were flushed from persistent queues, across all Destinations.
`pqTotalBytes`	Amount of data currently stored in persistent queues, across all Destinations.
`rejectCxn`	Number of TCP connections that were rejected.
`startTime`	The beginning of the timespan represented by these metrics.

Field	Description
`tasksCompleted`	The number of tasks the process has started and completed for all collection jobs for which it was executing tasks.
`tasksStarted`	The number of tasks the process started for all collection jobs for which it was executing tasks.

Troubleshooting With the `cribl_stderr` Log

In rare instances, Cribl Stream's Node.js backend may encounter an OOM or fatal error, preventing logging to its usual location. To aid in troubleshooting, a special `cribl_stderr.log` file is created, containing timestamped error details in UTC. This file is intended for Cribl Support and is not accessible through the UI.

To prevent excessive disk usage, Cribl Stream implements log rotation for `cribl_stderr.log`, limiting files to 5MB each with a maximum of 5 rotated copies. This addresses an issue where the log file could rapidly expand due to logged errors, impacting system performance.

6. Work With Data

Cribl Stream is a tool to collect, process, and route data. After receiving events from a Source you can work with data to transform, enrich, reduce, and redact it, before sending it to one or more target Destinations.

Routes let you control the flow of events by filtering and directing them to selected Pipelines.

Pipelines are where the data transformation happens. Each Pipeline, a series of Functions, can modify the incoming events in a highly configurable way.

Cribl Stream offers tools to help you test and verify your data flow. Use Datagens to generate sample data, and check Data Preview to see and validate the transformation you are performing.

Packs offer portable sets of pre-packaged data routing and transformation tools can include ready-made Routes, Pipelines, Function, and others.

For a series of guides and better practices in enriching, sampling, and transforming data, see:

- Using Lookups
- Transforming Data

6.1. Ingest And Inspect Data

The foundation of any good data Pipeline is the ability to reliably onboard, simulate, and validate your data. Before applying complex transformations, you must ensure that your data is correctly shaped and that your processing logic behaves as intended under real-world conditions.

Cribl Stream has various tools that allow you to move from initial setup to a verified, production-ready state:

- Onboard Data
- Use Datagens to Simulate Live Data
- Create and Share Data Samples
- Data Sample Use Cases
    - Ingest-Time Sampling
    - Sample Logs
    - Access Logs: Apache, ELB, CDN, S3
    - Firewall Logs: VPC Flow Logs, Cisco ASA

6.1.1. Onboard Data

Onboarding data into Cribl Stream can vary in complexity, depending on your organization's needs, requirements, and constraints. Proper onboarding from all Sources is key to system performance, troubleshooting, and ultimately the quality of data and decisions both in Cribl Stream and in downstream Destinations.

General Onboarding Steps

Typically, a data onboarding process revolves around these steps, both before and after turning on the Source:

- Create configuration settings.
- Verify that settings do the right thing.
- Iterate.

Below, we break down individual steps.

Before Turning On the Source

Cribl recommends that you take the following steps to verify and tune incoming data, before it starts flowing.

Preview Sample Data

Use a sample of your real data in Data Preview. Sample data can come from a sample Source file that you upload or paste into Cribl Stream.

You can also obtain sample data in a live data capture from a Source. One way to do this **before** going to production is to configure your Source with a **devnull** Pipeline (which just drops all events) as a pre-processing Pipeline. Then, let data flow in for just long enough to capture a sufficient sample.

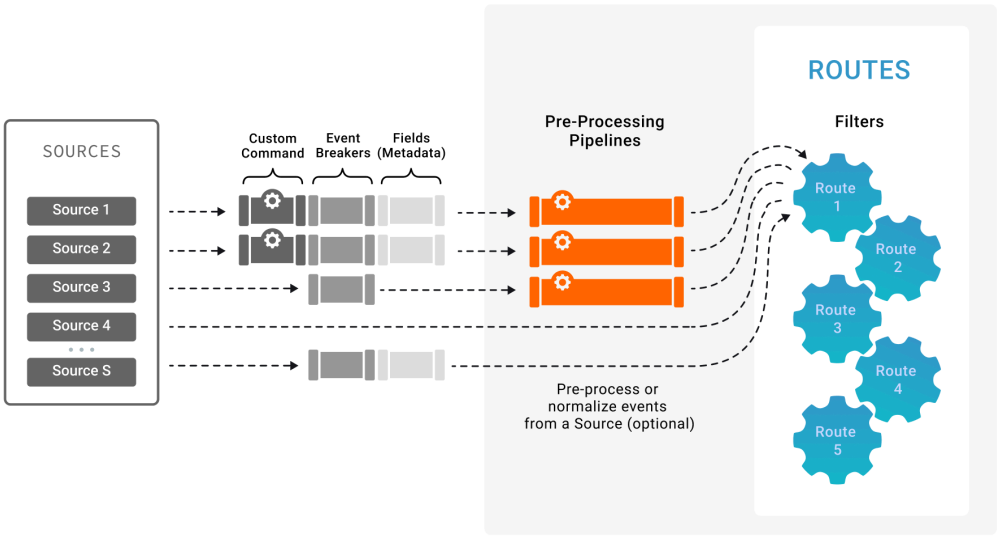
Very Large Integer Values

Cribl Stream's JavaScript implementation can safely represent integers only up to the Number.MAX_SAFE_INTEGER constant of about 9 quadrillion (precisely, {2^53}-1). Data Preview will round down any integer larger than this, and trailing 0's might indicate such rounding down.

Check the Processing Order

While events can be processed almost arbitrarily by Functions in Cribl Stream Pipelines, make sure you understand the event processing order. This is very important, as it tells you exactly where certain processing

steps occur. For instance, as we'll see just below, quite a few steps can be accomplished at the Source level, before data even hits Cribl Stream Routes.



Source-level processing options

Custom Command

Where supported, data streams will be handled by **custom commands**. These are external system commands that can (optionally) be used to pre-process the data. You can specify any command, script, etc., that consumes via `stdin` and outputs via `stdout`.

Verify that such commands are doing what's expected, as they are the very **first** in a series of processing steps.

Event Breakers

Next, data streams are handled by Event Breakers, which:

- Convert data streams into discrete events.
- Extract and assign timestamps to each event.

If the resulting events do not look correct, feel free to use **non-default** breaking rules and timestamp recognition patterns. Downstream, you can use the Auto Timestamp Function to modify `_time` as needed, if timestamps were not recognized properly. Examples of such errors are:

- Timestamps too far out in the future or past
- Wrong timezone.
- Incorrect timestamp is selected from multiple timestamps present in the event.

Fields

Next, events can be enriched with Fields . This is where you'd add static or dynamic fields to all events delivered by a particular Source.

Pre-Processing Pipeline

Next, you can optionally configure a pre-processing Pipeline on a particular Source. This is extremely useful in these cases:

- Drop non-useful events as early as possible (so as to save on CPU processing).
- Normalize events from this Source to conform a certain shape or structure.
- Fix/touch up events accordingly. E.g., if event breakers assigned the wrong timestamp, this is the best place to use the Auto Timestamp Function to adjust `_time`.

We Can't Say This Enough

Verify, verify, verify, data integrity before turning on the Source.

After Turning On the Source

Use data Destinations to verify that certain metrics of interest are accurate. This will depend significantly on the capabilities of each Destination, but here's a basic checklist of things to ensure:

- Timestamps are correct.
- All necessary fields are assigned to events.
- All expected events show up correctly. (E.g., if a Drop or Suppress Function was configured, ensure that it's not dropping unintended events.)
- Throughput - both in bytes and in events per second (EPS) - is what's expected, or is within a certain tolerance.

Iterate

Iterate on the steps above as necessary. E.g., adjust fields values and timestamps as needed.

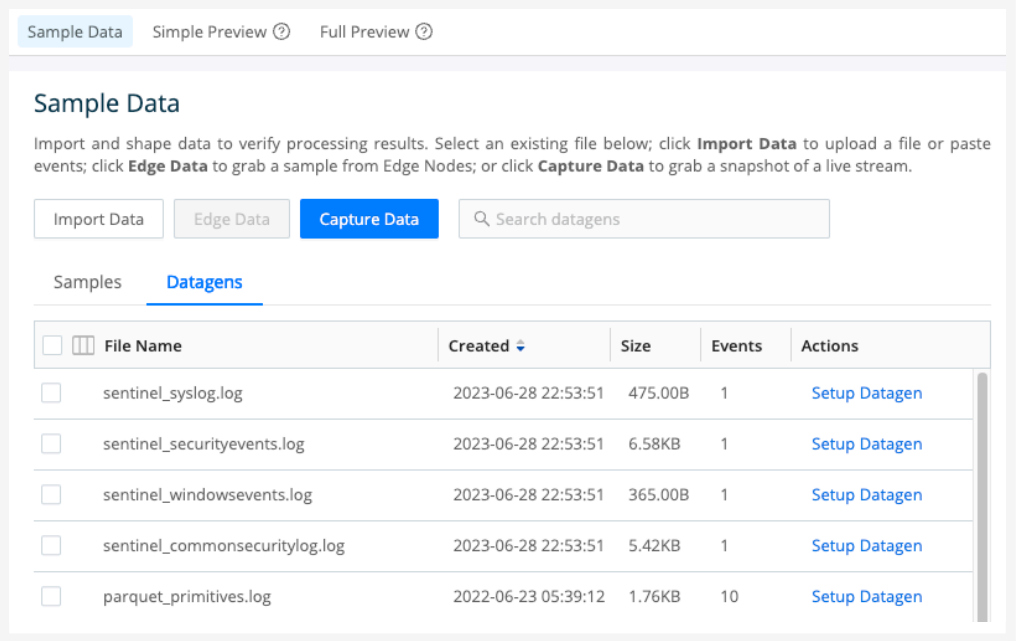
Remember that there is almost always a workaround. Any arbitrary event transformation that you need is likely just a Function or two away.

6.1.2. Use Datagens To Simulate Live Data

Data generators for testing and troubleshooting

Cribl Stream's datagens feature enables you to generate sample data for the purposes of troubleshooting Routes, Pipelines, Functions, and general connectivity.

Several datagen template files ship with the product, out of the box. You can create others from sample files or live captures.



Preview pane - add samples via paste, attach/upload file, or live capture

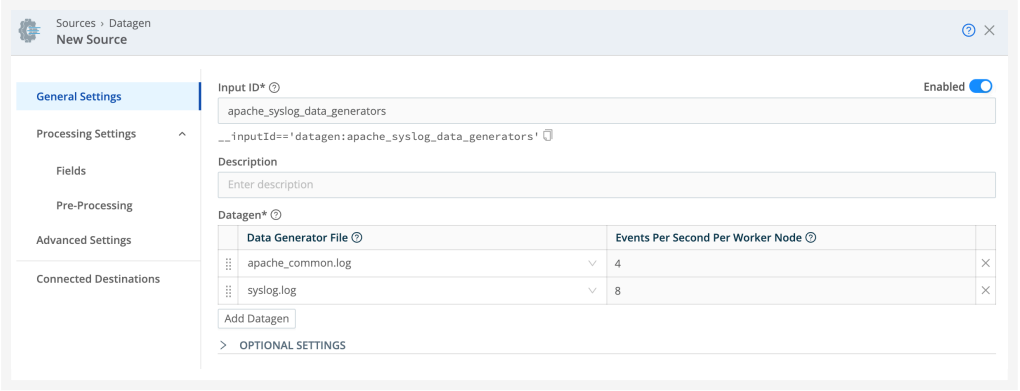
As outlined in the following tutorial: Once you've created a template, you can configure a Datagen Source to use the template to generate real-time data at a given EPS (events per second) rate.

Enable a Datagen

To see how datagens work, start by enabling a pair of Cribl Stream's out-of-the-box generators:

Navigate to **Sources** > **Datagens** and select **Add Source**.

After entering a unique ID and an optional description, select a data generator file (such as `apache_common.log`) and set it at `4` **Events Per Second Per Worker Node**. Select **Add Datagen** to select another data generator file (such as `syslog.log`) and set it at `8` **Events Per Second Per Worker Node**. Select **Save**.



Selecting datagen files and event rates

In the sidebar, select **Monitoring**, select **Data** > **Sources**, and search for `datagen` to confirm that the Source is generating data.

Create a Datagen Template from a Sample File

To convert a sample into a template:

From the **Data Routes** or **Pipelines** page, select **Import Data** in the right preview pane, and paste a sample like the AWS VPC Flow logs below:

Sample VPC Flow Logs

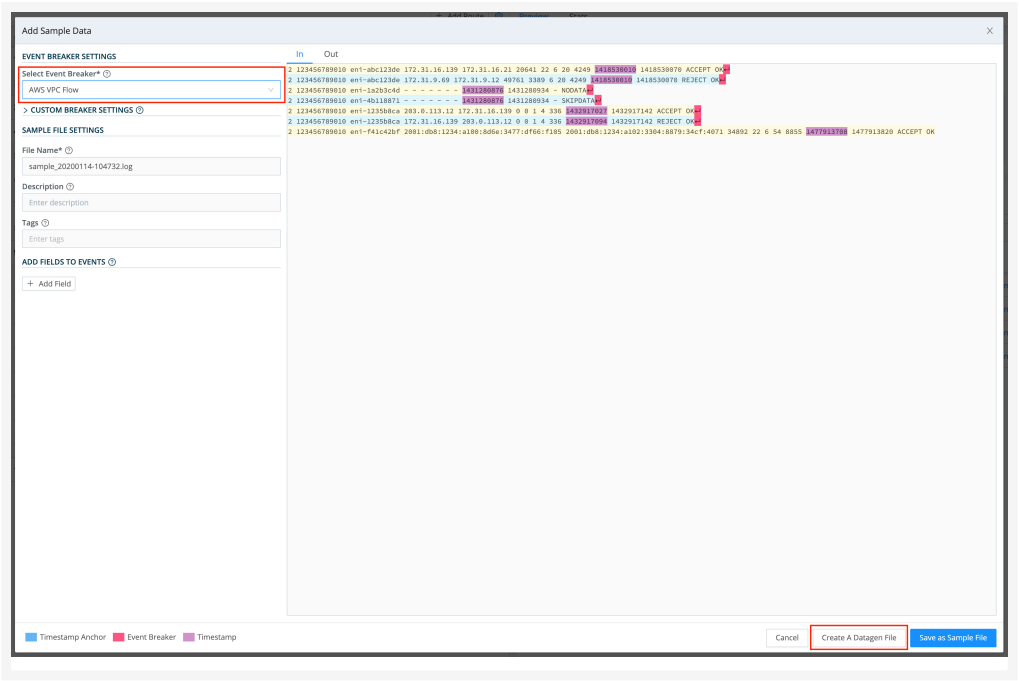
```
2 123456789010 eni-abc123de 172.31.16.139 172.31.16.21 20641 22 6 20 4249 1418530010
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1418530010 1
2 123456789010 eni-1a2b3c4d - - - - - - - 1431280876 1431280934 - NODATA
2 123456789010 eni-4b118871 - - - - - - - 1431280876 1431280934 - SKIPDATA
2 123456789010 eni-1235b8ca 203.0.113.12 172.31.16.139 0 0 1 4 336 1432917027 1432917
2 123456789010 eni-1235b8ca 172.31.16.139 203.0.113.12 0 0 1 4 336 1432917094 1432917
2 123456789010 eni-f41c42bf 2001:db8:1234:a100:8d6e:3477:df66:f105 2001:db8:1234:a102
```

Copy

From the **Select Event Breaker** drop-down, select **AWS VPC Flow** to ensure that:

- The pasted text gets broken properly into individual events (notice the Event Breaker on newlines).
- Timestamps are extracted correctly (text highlighted purple below).

Once you've verified these results, select **Create a Datagen File**.

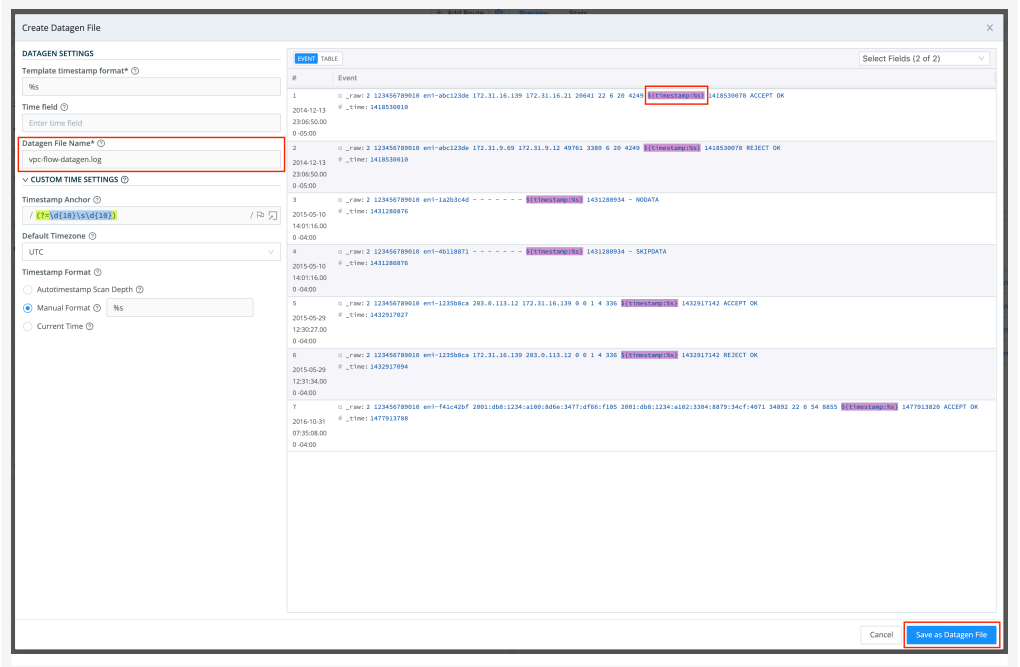


Creating a datagen template

On the resulting **Create Datagen File** screen:

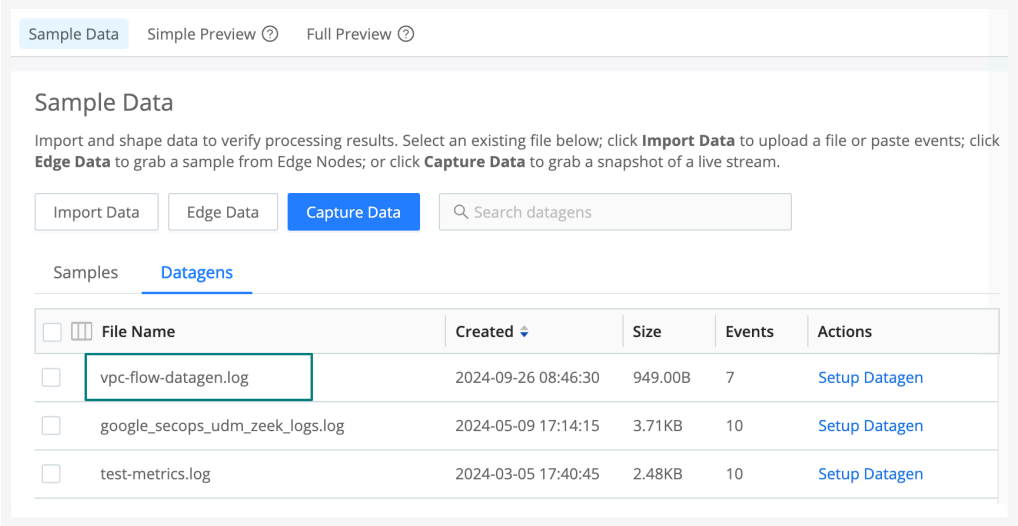
- Enter a file name, such as `vpc-flow-datagen.log`
- Ensure that the timestamp template format is correct: `${timestamp: %s}`
  `${timestamp: <format>}` is a template that the datagen engine uses to insert the current time - in each newly generated event - using the given format. In this case, `%s` is the desired `strftime` format for the timestamp (the epoch).

Once you've verified these results, click **Save as Datagen File**.



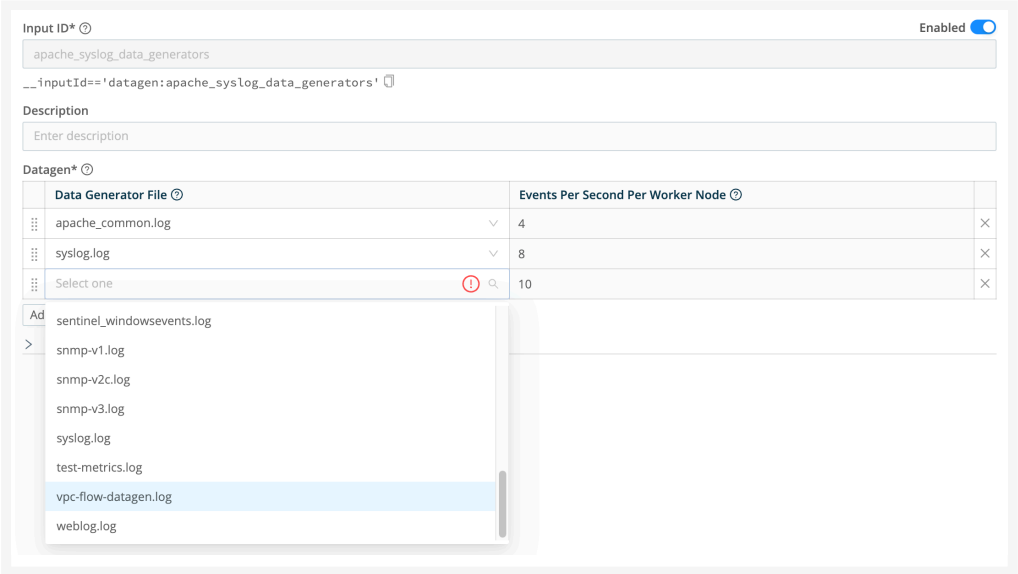
Saving a named datagen template

To confirm that the datagen file has been created, select the **Datagens** tab in the preview pane.



Verifying datagen file creation

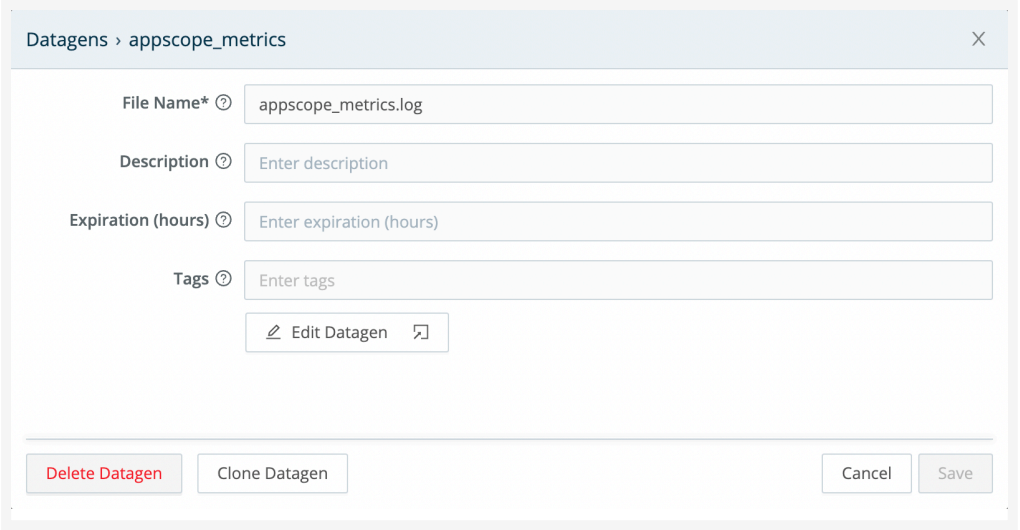
Now, to start using your newly created datagen file, go back to **Sources** > **Datagens**. Add it using the drop-down shown below.



Adding new template file to datagens Source

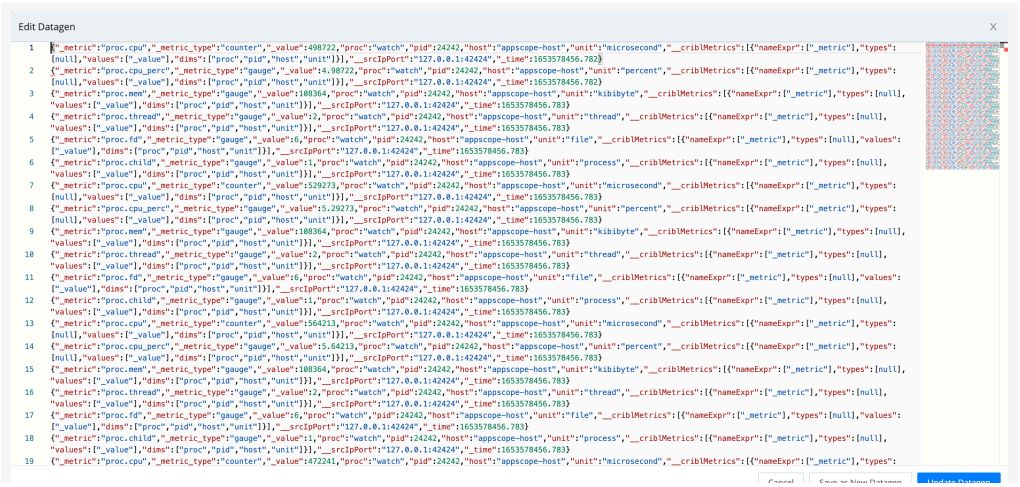
Modify a Datagen

1. In the right Preview pane, select the **Datagens** tab.
2. Hover over the file name you want to modify. This displays an edit (pencil) button to its left.
3. Select that button to open the modal shown below. It provides options to edit the datagen, clone it, delete it, or modify its metadata (**File name**, **Description**, **Expiration time**, and **Tags**).



Options for modifying a datagen

4. To make changes to the datagen, select the modal's **Edit Datagen** button. This opens the **Edit Datagen** modal shown below, exposing the raw data that this datagen uses to generate events.
5. Edit the raw data as desired.
6. Select **Update Datagen** to resave the modified datagen, or select **Save as New Datagen** to give the modified version a new name.



6.1.3. Create And Share Data Samples

A data sample (also called a data capture) is a small, representative subset of live or stored data from a given data stream. Sample files are essential tools for:

- Ensuring that proper data is flowing between the right Sources and Destinations.
- Confirming that data is routed correctly.
- Testing and validating Routes, Pipelines, and Functions to ensure they process data as expected.

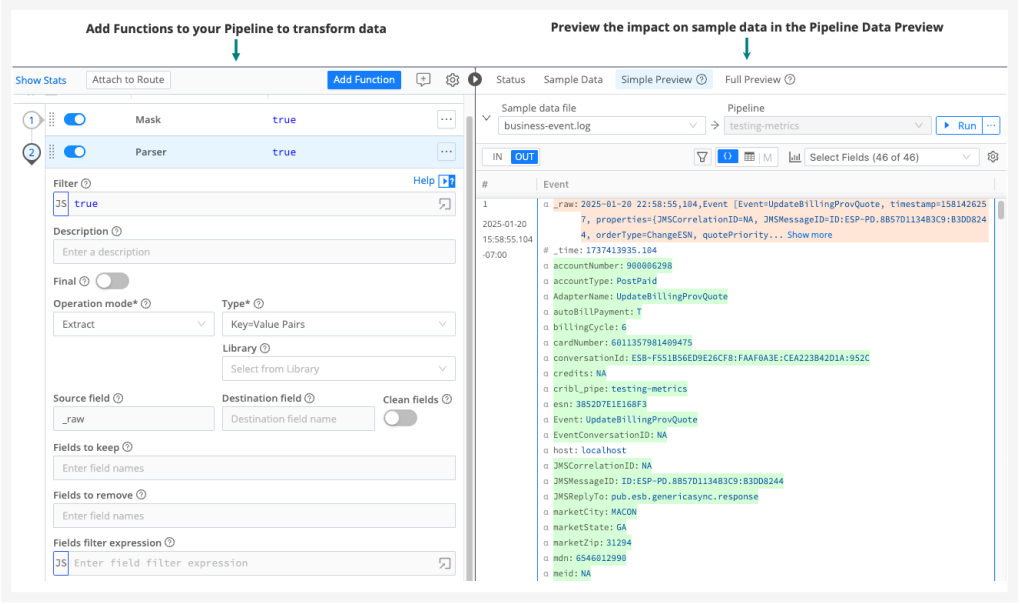
Access and Manage Data Samples

Whether you are capturing live data samples for immediate testing or managing a shared library for collaboration, you can work with data samples through a few main workflows:

Workflow	Description
Data Preview	Capture sample data, then test and validate Pipeline logic in real-time using the Data Preview tool as you build Routes and Pipelines to see your data processing logic in action.
Knowledge Objects	Save samples as Knowledge objects in a central location to manage and reuse them across Routes, Pipelines, Event Breakers, and datagens. Samples you create here can also be used in the Data Preview tool.
Datagens	Generate a stream of realistic sample log data to help you simulate live data. Build custom datagens (data generators) from your own sample files.

Data Preview

The primary way that you will use sample files is with the Data Preview tool for Pipelines and Routes. The Data Preview tool works by processing a set of sample events and passing them through the Pipeline or Route. Then, it displays the inbound and outbound results in a dedicated pane. Whenever you modify, add, or remove a Function, the output updates instantly to reflect your changes. This is helpful for validating your data processing logic in real-time to assess whether the logic works as you expected.



Add Functions to your Pipeline to transform data, then preview the impact on sample data in Data Preview

See Validate Pipeline Logic Using Data Preview for information about accessing and using the Data Preview tool.

Knowledge Objects

You can create and save sample data files as Knowledge objects at either the Worker Group level or the Pack level. The advantage of creating samples as Knowledge objects is that these objects can be accessed across your Worker Group by Routes, Pipelines, Event Breakers, and datagens as needed.

When working with sample files as Knowledge objects, be aware that sample file visibility is context-dependent:

- Worker Group samples are only visible within that specific Worker Group.
- Pack samples are only visible within that specific Pack.

To use a sample in a different context, you must manually copy it from the Worker Group to the Pack (or the Pack to the Worker Group). See Copy and Share Sample Files for specific instructions.

To access sample files in the Knowledge library at the Worker Group level:

1. Select **Worker Groups** from the sidebar, and choose a Worker Group.
2. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Sample Files**.

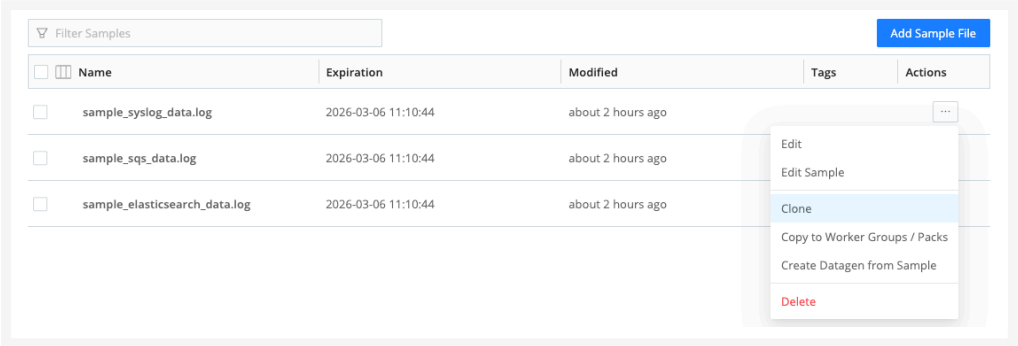
To access sample files in the Knowledge library from inside a Pack:

1. Ensure you are in the correct Pack.
2. Select **Knowledge** from the top bar, then **Sample Files**.

The Knowledge library lists any sample files you have captured. From the list view, you add new files, edit samples, clone, or copy sample files to other Worker Groups or Packs. See Copy and Share Sample Files for specific instructions.

Cross-copying samples between Worker Groups and Fleets is not supported.

You can also build a custom datagen (data generator) from these sample files to simulate live data. See the next section about datagens for more information.



View and manage sample files in the Knowledge objects library

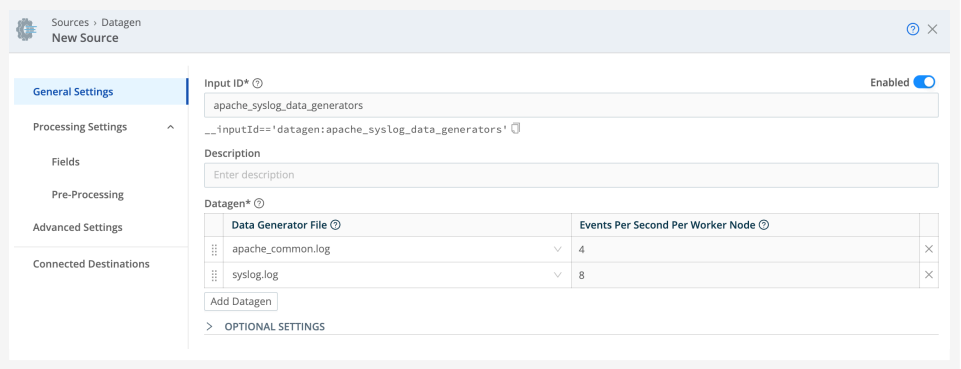
Datagens

Datagens (data generators) allow you to simulate live data by replaying sample log data. You can then use this live data to test and validate your Routes, Pipelines, and Functions in a sandbox environment before applying changes to production data.

Cribl provides several datagens that you can use out-of-the-box to simulate live data. You can also transform your own custom sample files into active generators. See Use Datagens to Simulate Live Data for more information.

If you create a sample file in the Knowledge object library, you can use this file to build a custom datagen. To create the datagen from a sample file:

1. Ensure you are in the correct context (either in a Worker Group or a Pack). Then access the Knowledge objects library in that context.
2. Next to the sample file you want to copy or share, select **Create Datagen from Samples** from the **Actions** (⋯) menu.
3. In the modal, update the datagen settings as needed. See Use Datagens to Simulate Live Data for more information.



Simulate live data with a data generator

4. Select **Save as Datagen File**.

The new datagen is automatically added to your list of Sources for that Worker Group or Pack. It will share the same name as the original sample file.

Add Sample Data

To validate your data, you first need sample data to work with. Some methods of getting sample data can only work with the Data Preview tool. If you are creating a data sample using the Knowledge object library, you can only capture sample data live from a Source or from a datagen.

You can get sample data using a few methods:

Method	Description	Requirements and Limitations
Capture new sample data	Capture live data from multiple incoming data streams to use for validating data routing and processing logic.	• Data captures can only work with events. • Data captures do not require event breaking. • To capture live data, you must have Worker Nodes registered to the Worker Group for which you're viewing events.
Capture live data from a single Source or Destination	Use this method if you only need to capture data from one Source or Destination to build your data sample.	• To capture live data from a single enabled Source or Destination, it's fastest to use the Sources or Destinations pages instead of the Preview pane. • To capture live data, you must have Worker Nodes registered to the Worker Group for which you're viewing events.
Upload a sample data file	Use this method to upload sample data file from another	• Available for Data Preview only.

Method	Description	Requirements and Limitations
	Cribl deployment or an external data source.	• The **Import Data** options require content that is broken into events, which means it needs Event Breakers.
Copy data from a clipboard	Use this method to copy and paste sample data from your computer's internal clipboard.	• Available for Data Preview only. • The **Import Data** options require content that is broken into events, which means it needs Event Breakers.
Import data from Cribl Edge Nodes	Use this method to import data from Cribl Edge.	• Available for Data Preview only. • You must have at least one working Edge Node.

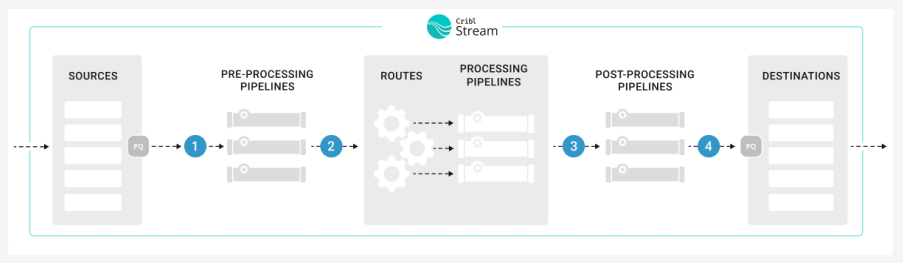
Capture New Sample Data

The **Capture Data** mode does not require event breaking. To capture live data, you must have Worker Nodes registered to the Worker Group for which you're viewing events. You can view registered Worker Nodes from the **Status** tab in the Source.

To capture new sample data:

1. Ensure you are in the correct context (such as the Worker Group or inside a Pack). To navigate to the Knowledge library:
    - **From a Worker Group:** Select **Worker Groups** from the sidebar, and choose a Worker Group. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Sample Files**.
    - **From a Pack:** Select **Knowledge** from the top bar, then **Sample Files**.
2. Select **Add Sample File**.
3. Select **Capture** and modify the settings as needed:
    - **Capture time (sec):** The length of the time you want to sample in seconds. Defaults to 10 seconds.
    - **Capture up to N events:** The maximum number of events to capture. Defaults to 10.
    - **Where to capture:** Select the stage in the data flow where you want to capture events. Options include:
        - Before pre-processing Pipeline (labeled as `1` in the following diagram)
        - Before the Routes (`2`)
        - Before post-processing Pipeline (`3`)

- Before the Destination (4)



Options for where to capture the data

  - **Initial Worker sample size:** Set a limit to the number of Worker Nodes that are included in the capture. Defaults to 50 Nodes. Set it to `0` to make it unlimited.
  - **Interval (sec):** Specify how long the system should wait for data before increasing the capture sample size. Must be at least 1 second. Defaults to 2 seconds.

4. Select **Start** and the system will start capturing data.

If the capture fails, check that you have committed and deployed all changes. Also check the **Filter** setting for the data capture to ensure it is not filtering out all events.

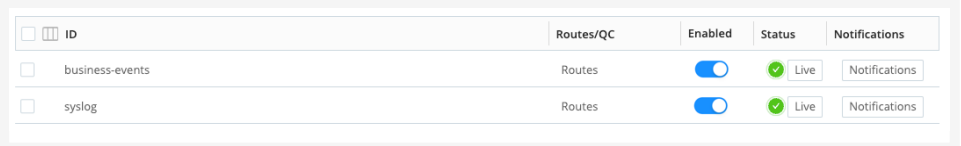
5. Select **Save as Sample File**.

Verify that the new sample file appears in the Data Preview pane or in the Knowledge objects library.

Capture Live Data from a Single Source or Destination

To capture data from a single enabled Source or Destination, it's fastest to use the Sources or Destinations pages instead of the Preview pane:

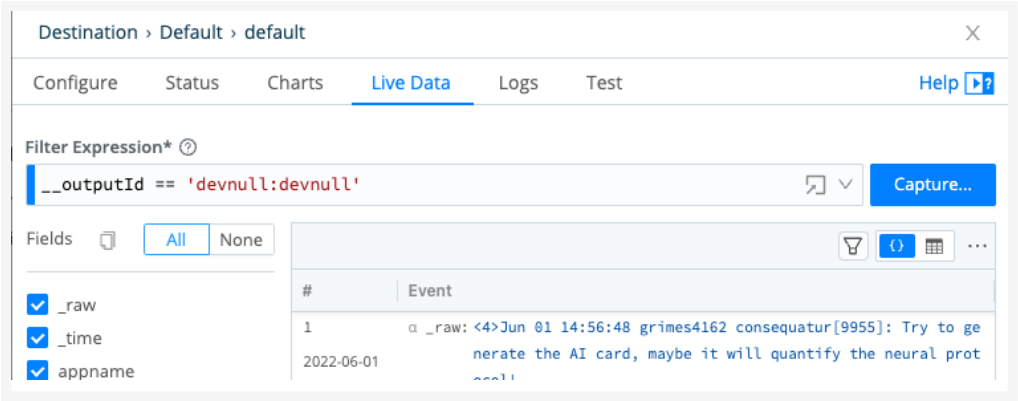
1. In the sidebar, select **Worker Groups**, then select a Worker Group. On the **Worker Groups** submenu, select **Data**, then **Sources** or **Destinations**.
2. Select **Live** on the Source or Destination configuration row to initiate a live capture.



Live button on a Source

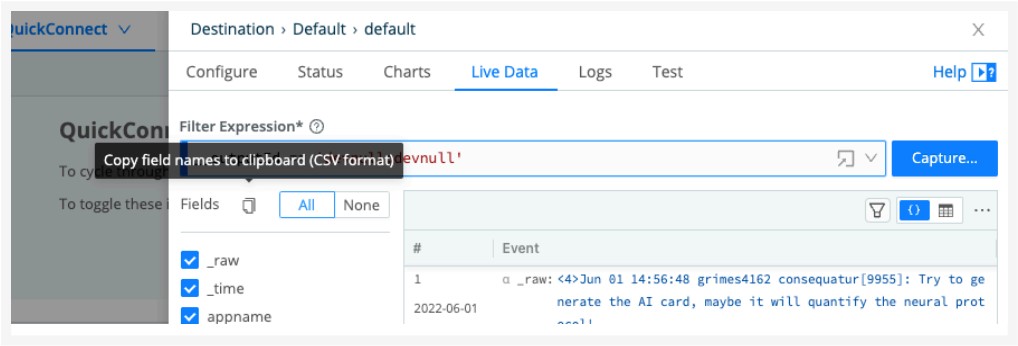
3. Select **Capture**. Accept or modify the default settings as needed and select **Start**.
4. When the modal finishes populating with events, select **Save as Sample File**. In the sample file settings, configure the sample's metadata. See Sample Data Settings for detailed information about these settings.

Alternatively, you can start an immediate capture from within an enabled Source's or Destination's configuration modal, by selecting the modal's **Live Data** tab.



The Live Data tab in the Destination modal

You can also use the **Fields** selector in the **Live Data** is a Copy button, which enables you to copy the field names to the clipboard in CSV format. The **Logs** tab also provides this copy button.

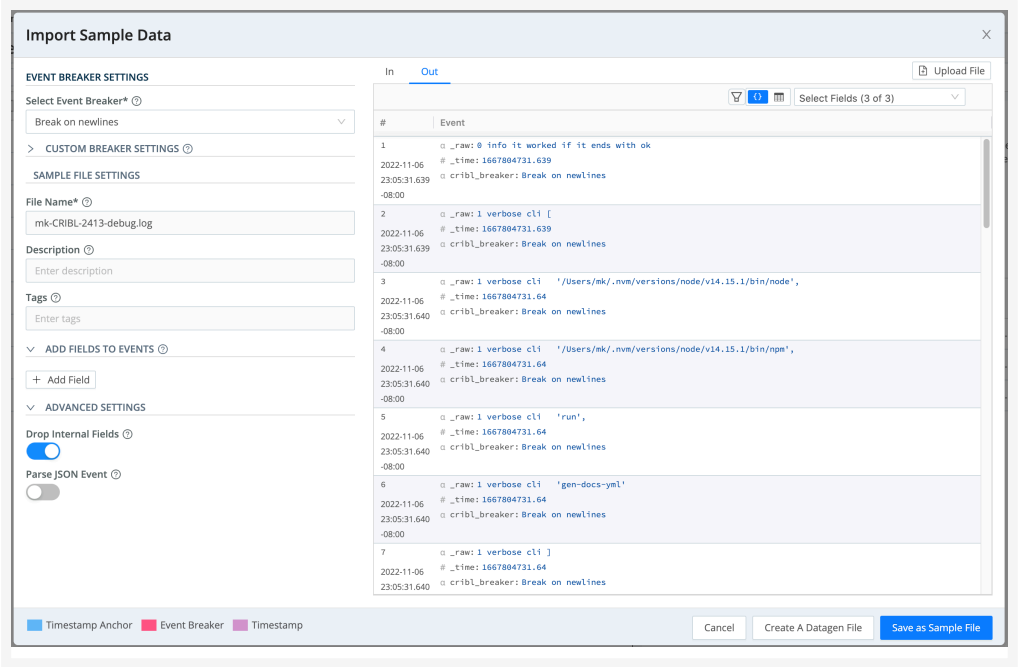


The Copy Fields Icon in the Live Data tab

Upload a Sample Data File

To upload a sample data file:

1. In the sidebar, select **Worker Groups**, then select a Worker Group. On the **Worker Groups** submenu, select **Processing**, then **Pipelines**.
2. Select the **Sample Data** tab in the **Data Preview** pane. Select **Import Data**.
3. In the **Import Sample Data** modal, drag and drop your sample file or select **Upload** to navigate to the file on your computer.
4. Select an appropriate option from the **Event Breaker Settings** menu.
5. Configure the sample's metadata. See Sample Data Settings for detailed information about these settings.
6. Add or drop fields as needed.



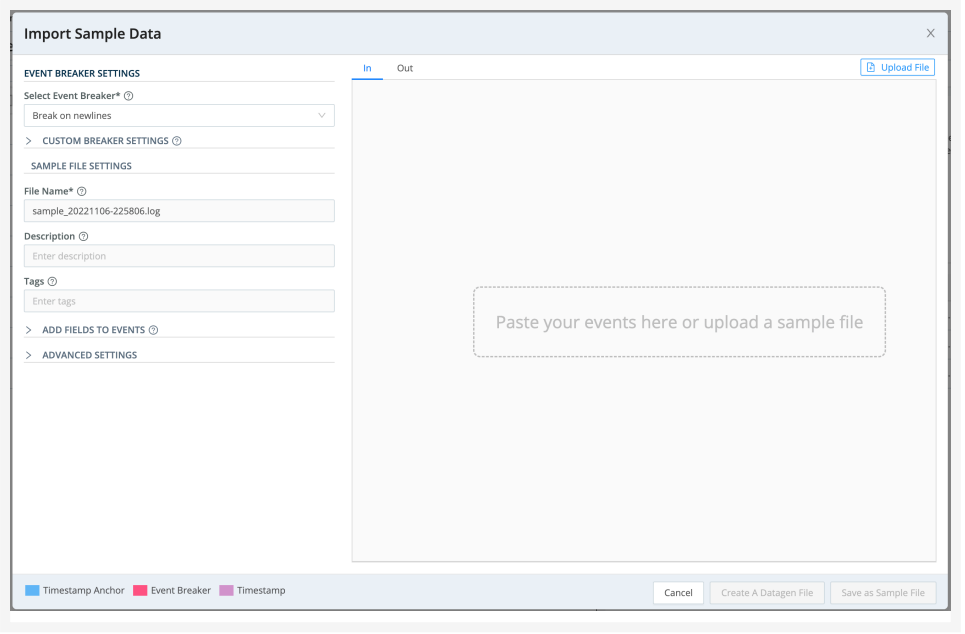
Example of imported data

If you notice imported events are fragmented, it could be because Cribl Stream has added a `__timeoutFlush` internal field to them. See the troubleshooting section about Imported Data Events Are Fragmented for more information.

Copy Data From a Clipboard

To copy sample data from a clipboard:

1. In the sidebar, select **Worker Groups**, then select a Worker Group. On the **Worker Groups** submenu, select **Processing**, then **Pipelines**.
2. Select the **Sample Data** tab in the **Data Preview** pane. Select **Import Data**.
3. In the **Import Sample Data** modal, paste data from your computer clipboard into the provided field.



Import Sample Data modal

4. Select an appropriate option from the **Event Breaker Settings** menu.
5. Configure the sample's metadata. See Sample Data Settings for detailed information about these settings.
6. Add or drop fields as needed.

If you notice imported events are fragmented, it could be because Cribl Stream has added a `__timeoutFlush` internal field to them. See the troubleshooting section about Imported Data Events Are Fragmented for more information.

Import Data from Cribl Edge Nodes

To import data from Cribl Edge Nodes, you must have at least one working Edge Node. To upload data from a file on an Edge Node:

1. In the sidebar, select **Worker Groups**, then select a Worker Group. On the **Worker Groups** submenu, select **Processing**, then **Pipelines**.
2. Select the **Sample Data** tab in the **Data Preview** pane. Select **Import Data**.
3. In the **Import Sample Data** modal, select **Edge Data** and navigate to the Edge Node that has the stored file.
4. In **Manual** mode, use the available filters to narrow the results:
    - **Path**: The location from which to discover files.
    - **Allowlist**: Supports wildcard syntax and supports the exclamation mark (!) for negation.

- **Max depth**: Sets which layers of files to return highlighted in bold typeface. By default, this field is empty, which implicitly specifies `0`. This default boldfaces only the top-level files within the **Path**.

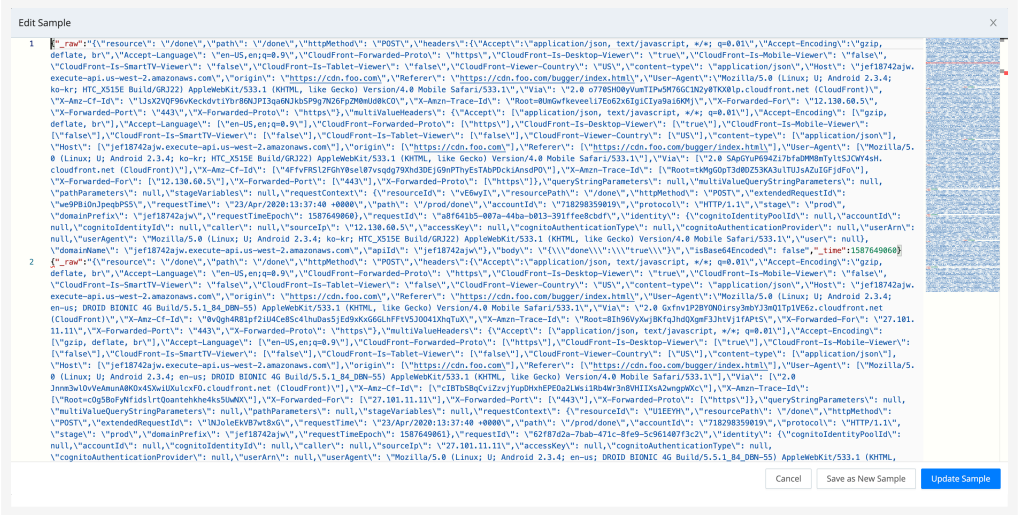
5. Once you find the file you want, select its name to add its contents to the **Add Sample Data** modal.
6. Select an appropriate option from the **Event Breaker Settings** menu.
7. Configure the sample's metadata. See Sample Data Settings for detailed information about these settings.
8. Add or drop fields as needed.

Modify Sample Files

To modify sample files that were captured and stored as Knowledge objects, you can edit samples by accessing the list of sample files in the Knowledge objects library. From there, select the **Edit Sample** option from the **Actions** (...) menu.

To modify existing sample files in the Data Preview tool in Routes or Pipelines:

1. In the right Preview pane, select the **Samples** tab.
2. Hover over the file name you want to modify to show an edit (pencil) button next to the file name.
3. Select that button to open the modal shown below. It provides options to edit, clone, or delete the sample, save it as a datagen Source, or modify its metadata. See Sample Data Settings for detailed information about these settings.
4. To make changes to the sample, select the modal's **Edit Sample** button. This opens the **Edit Sample** modal shown below, exposing the sample's raw data.
5. Edit the raw data as desired.
6. Select **Update Sample** to resave the modified sample, or select **Save as New Sample** to give the modified version a new name.



Editing a sample file

Copy and Share Sample Files

To clone a sample file in the Data Preview tool, follow the same steps to Modify Sample Files.

Sample files cannot be copied from a Worker Group to a Fleet.

To copy sample files that were captured and stored as Knowledge objects:

1. Select **Worker Groups** from the sidebar, and choose a Worker Group. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Sample Files**.
2. Next to the sample file you want to copy or share, select **Copy to Worker Groups / Packs** option from the **Actions** (...) menu.
3. In the **Copy Sample** modal, define your target:
    - **Worker Group:** Type or select the target Worker Group. Leave blank if you only want to copy the file to a Pack.
    - **Packs:** Type or select the target Pack. Leave blank if you only want to copy the file to a Worker Group.
4. Select **Copy Sample**.

A confirmation message will appear upon a successful copy.

If you want to share sample files across your Cribl deployment, add them to a Pack and then copy the Pack to other Worker Groups. From there, you can copy it to the target Worker Group. See Copy a Pack for specific instructions.

Sample Data Settings

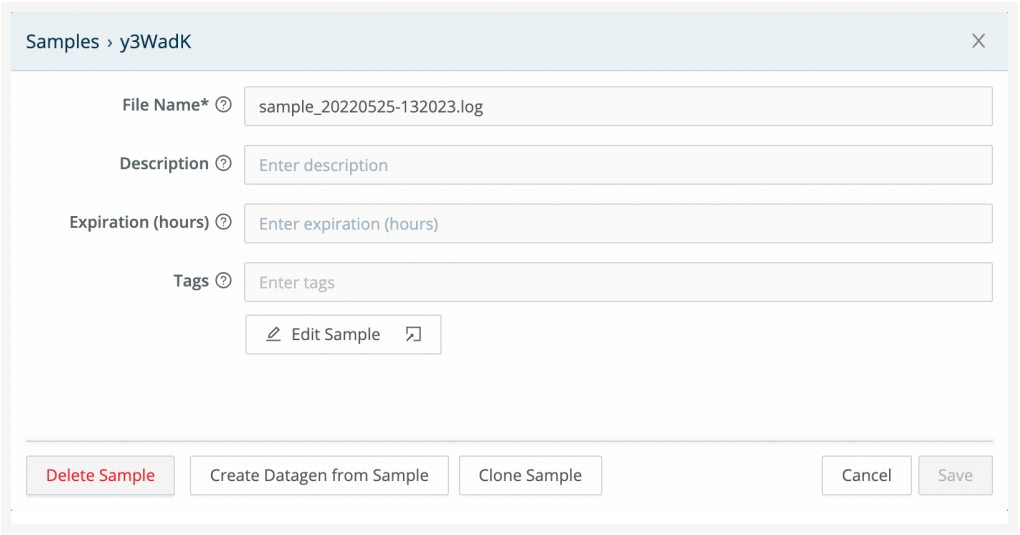
These metadata settings are available for all data samples:

- **File name**: The display name for the sample file. Choose a name that makes the sample easy to recognize in the **Sample Data** and **Samples** tabs.
- **Description**: An optional, free-form description that explains what the sample represents (source system, time range, scenario, and so on).
- **Expiration time**: A time-to-live (TTL), in hours, for the sample file. Each time you use the sample, for example in Data Preview, Pipeline testing, or when creating a datagen, the TTL resets. When a sample reaches its expiration without being used, Cribl Stream silently removes the sample and its backing file. Leave this field blank if you do not want the sample to expire automatically.

In deployments that use Git-backed configuration bundles, expiration only affects the live configuration on disk. If you commit while the sample exists, that commit continues to contain the sample. When the

sample later expires, its removal is just another change that you can commit in a later revision. If the sample expires before you commit, it is cleaned up without ever being recorded in Git.

- **Tags**: Optional labels that you can use to organize and search for samples (for example, by source, environment, or use case).



Options for modifying a sample

Troubleshoot Sample Data

This section describes some strategies for managing potential issues with sample data.

Integer Values Are Too Large

The JavaScript implementation in Cribl Stream can safely represent integers only up to the Number.MAX_SAFE_INTEGER constant of about 9 quadrillion (precisely, {2^53}-1). Data Preview will round down any integer larger than this, and trailing zeroes might indicate such rounding.

Imported Data Events Are Fragmented

The **Import Data** options work with content that needs to be broken into events, meaning it needs Event Breakers.

Cribl Event Breakers are regular expressions that tell Cribl Stream how to break the file or pasted content into events. Breaking occurs at the **start** of the match. Cribl Stream ships with several common breaker patterns out of the box, but you can also configure custom breakers. The UI here is interactive, and you can iterate until you find the exact pattern.

If you notice fragmented events, check whether Cribl Stream has added a `__timeoutFlush` internal field to them. The presence of this diagnostic field indicates that Cribl Stream flushed the events because the Event

6.1.4. Data Sample Use Cases

Data sampling lets you analyze and troubleshoot data and plan transformations you need to perform on a selected, representative sample of real-life data.

The following use cases guide you through preparing and using data samples in specific situations:

- Ingest-Time Sampling
- Sample Logs
- Access Logs: Apache, ELB, CDN, S3
- Firewall Logs: VPC Flow Logs, Cisco ASA

6.1.4.1. Ingest-Time Sampling

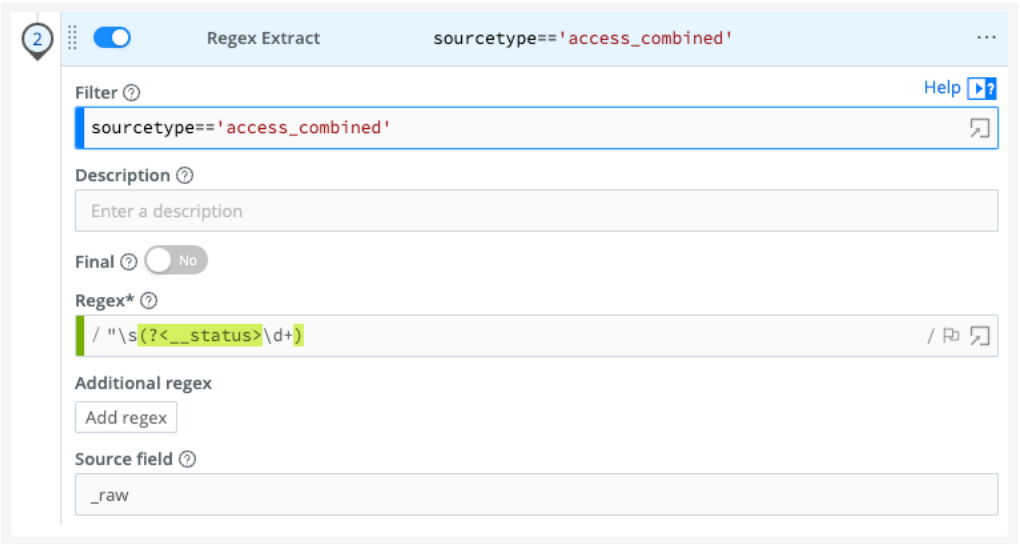
Let's say that you wanted to analyze and troubleshoot with **highly verbose/voluminous** data - for example, CDN logs, ELB Access Logs, or VPC Flows - but you were concerned about storage requirements and search performance. With Sampling, you can bring in enough samples that your analysis remains statistically significant, and also do all the necessary troubleshooting.

See the example below, or see more details in Access Logs and Firewall Logs.

Sampling Example

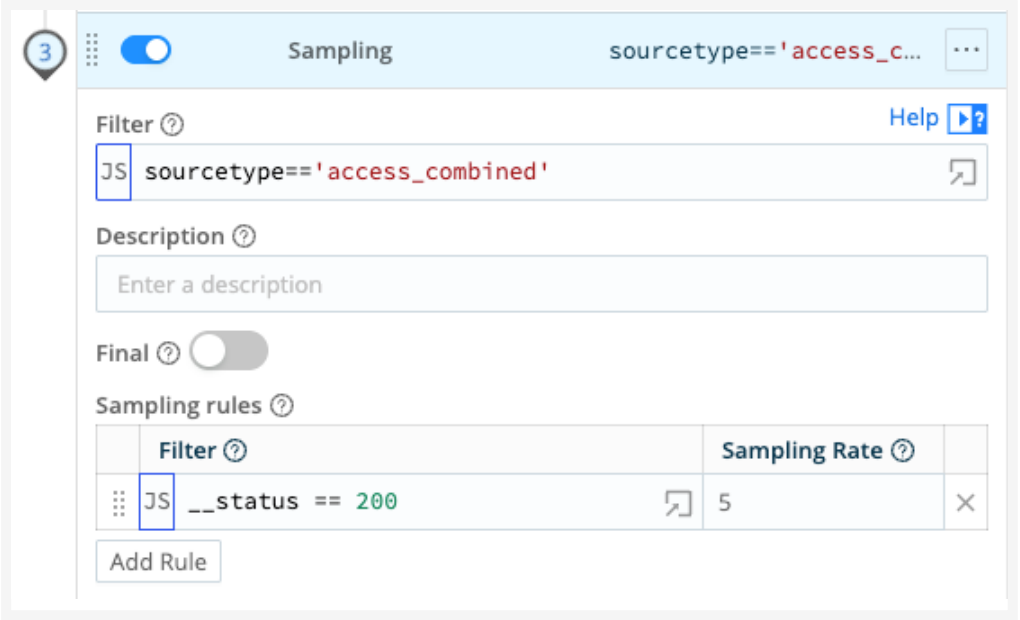
Let's use the out-of-the-box **Sampling** function to sample all events from `sourcetype=='access_combined'` where `status` is `'200'`. We'll sample these at 5:1 (and all other events at 1:1). This should lower the volume of all verbose successes (`200`s), but still bring in **all** potentially erroneous events (`400`s, `500`s, etc.) that can be used for troubleshooting.

- First, make sure you have a Route and Pipeline configured to match desired events.
- Next, let's add a **Regex Extract** Function to extract the status field from `_raw`, and let's call the resulting field `__status`. Remember, fields that start with `__` are special fields in Cribl Stream, and can be used anywhere in a Pipeline.



Extracting the `__status` field

Next, let's add a **Sampling** function, and scope it to all events where `sourcetype=='access_combined'`. Let's apply a filter condition of `__status == 200`, and a Sample Rate of `5`.



Sampling success responses

To confirm that sampling works, compare the event count of all `200`s before and after.

Each time an event goes through the **Sampling** function, an index-time `sampled::<rate>` field is added to it. You can use this field in your statistical functions, as necessary.

6.1.4.2. Sample Logs

Collecting samples of the event data you plan to work with in Cribl Stream can make your Cribl Stream onboarding experience even quicker and more efficient than if you don't.

When you set out to collect sample data, first decide whether to collect before or after the sending agent processes it.

- If you can, collect the data **after** the agent processes it. Then you can set up Cribl Stream Pipelines more accurately the first time. The best option is live capture with Cribl Stream. Next best is gathering files.
- If the only practical option is to collect data **before** it reaches the agent, that works, too. (This will usually be a matter of gathering files.) Since setting up Pipelines is an iterative process, you can still arrive at an optimal setup even if it takes a bit longer.

Regardless of what sending agent you're using, either the Capturing Data with Cribl Stream or the Gathering Original Files section will apply.

Beyond that, there are two supplemental sections: one about how to export data from Elasticsearch, and the other about exporting from Splunk. Of course, these are only two possibilities in a long list of sending agents. If you are using a sending agent that's not covered here, please join us on Cribl's Community Slack at https://cribl-community.slack.com/ and share your questions about collecting sample logs.

Describing Sample Data

For every group of sample files, create a `README` file which includes:

- Sourcetype (e.g., the type of log you're collecting samples from).
- Originating app or appliance.
- Delivery mechanism (e.g., syslog, Filebeat, Splunk UF, or something else).
- Whether the was data collected before or after processing by the agent.
- A brief description of the data.

Capturing Data with Cribl Stream

Though not always the easiest, this is the best option, because samples are captured exactly as they will look when handed off to Cribl Stream in production. You'll set Cribl Stream up as a passthrough which gets samples "right off the wire" and dumps them to DevNull.

Configuring Sources

In Cribl.Cloud:

Check the landing page to see whether the pre-defined Sources (Elastic, syslog, and Splunk, among others) are sufficient for your needs. If you need others, configure them using ports in the range 20000-20010.

Or, if your deployment is on-prem or in a private cloud:

Download and install Cribl Stream.

```
$ mkdir /opt; cd /opt
$ curl -Lso - $(curl https://cdn.cribl.io/dl/latest-x64) | tar zxvf -
$ cribl/bin/cribl start
```

Copy

Open `localhost:9000` in a browser.

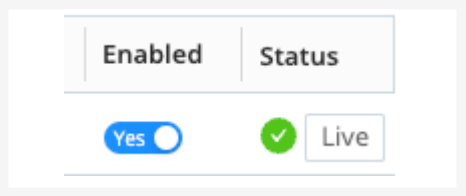
Configure the Source you need. For simplicity, use port 1024 or higher. (If you must use lower ports you'll need to follow a different installation procedure.)

Directing Data to Cribl Stream

Point your sending agent at the Cribl Stream Source you created above. How you do this depends on what sending agent you use. See this Cribl blog for a how-to that focuses on Cribl.Cloud but also applies to on-prem deployments.

Live-capturing Data

Once data has begun flowing, click the **Live** button for your chosen Source.



The **Live** button initiates capture

Grab 100 to 500 events, at a minimum. Try to capture as much variety as possible, even if that requires multiple captures.

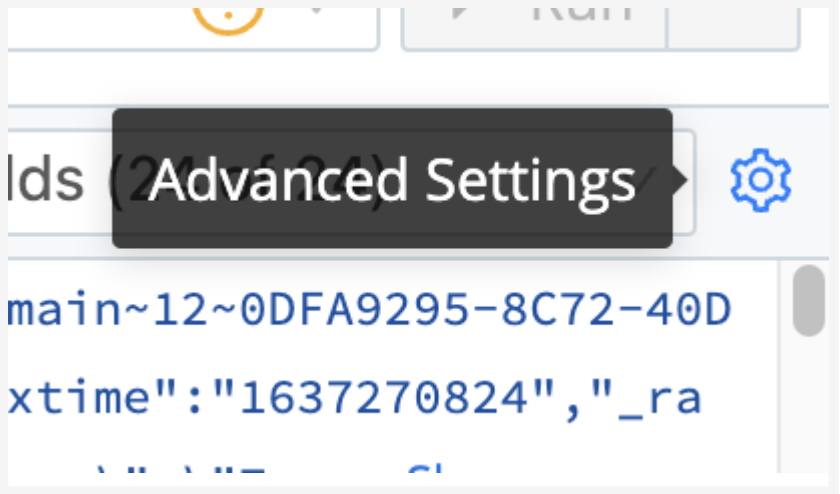
When finished, click **Save as Sample File**. Note the file name (and rename if you like). Close the sample window.

In the **Routing** > **Data Routes** tab, select the Route you are using.

Your sample file(s) should appear in the **Sample Data** pane on the right. If the **Sample Data** pane does not appear, type] (right square bracket) key to unhide it.

For each file you want to download:

- Click the file name.
- Click the (gear icon) for **Advanced Settings**:



The **Advanced Settings** icon

- In the resulting pop-up, select **Save** > **NDJSON**.
- Once the file finishes downloading:
  - Move the file into the directory you've created for that set of sample files - which should also include a `README` as described above.
- When all files have been moved into the directory, compress the directory as a `.tgz` archive.

Gathering Original Sample Files

If live-capturing data with Cribl Stream is impractical for you, gather sample files that the sending agent has not yet processed, using the following general workflow:

- Create a directory in which to store samples.
- Copy a sample file into the new directory.
- Repeat until all desired sample files have been copied.
- Add a `README` file to the directory, as described above.
- Redact sensitive content in the data, if required.
- Archive the directory (e.g., with a command like `tar -czf samples.tgz samples`) for portability; send it to your Cribl representative, if you are working with one.

Exporting Data from a Platform

If your data is in Elastic or Splunk, and you can neither capture live nor obtain original samples, use one of the following export procedures. Then complete the workflow above, using the exported files instead of original sample files.

If Splunk uses props and/or transforms to modify the `_raw` data of your logs, you **must** obtain your sample data from the source, because the exported data will **not** include events in their original form.

Exporting Data from Elastic

1. Elastic often transforms the data it stores. To the extent possible, note any particulars about the shape and content of the original data before it reached Elastic. Take special care to record the effects of any Logstash pipeline configs on the sample data. Record all this information in the `README` you create. This will make it easier to create the most effective filters in Cribl Stream.
2. Install the Logstash plugins logstash-input-elasticsearch and logstash-output-csv:

```
$ cd /opt/logstash
$ bin/logstash-plugin install logstash-input-csv
$ bin/logstash-plugin install logstash-output-csv
```

Copy

3. Compose a query to retrieve the data you want. Be sure to include the `message` field, whose value should be the original event.
4. Substitute your query for the placeholder in the following config snippet, and save the snippet as `output-csv.conf`:

```
input {
 elasticsearch {
    hosts => "elastic-host:9200"
    index => "target-index"
    query => '
  {"query": {
    # your ES query here
        }
      }
    }
}}'
  }
}
output {
  csv {
    fields => ["message", "field1", "field2"]
path => "/home/user/ES-sourcetype-export.csv"
  }
}
```

Copy

5. Run your export operation:

```
/opt/logstash/bin/logstash -f output-csv.conf
```

Copy

Exporting Data from Splunk

You can export logs from Splunk using the GUI or command line.

In the Splunk GUI:

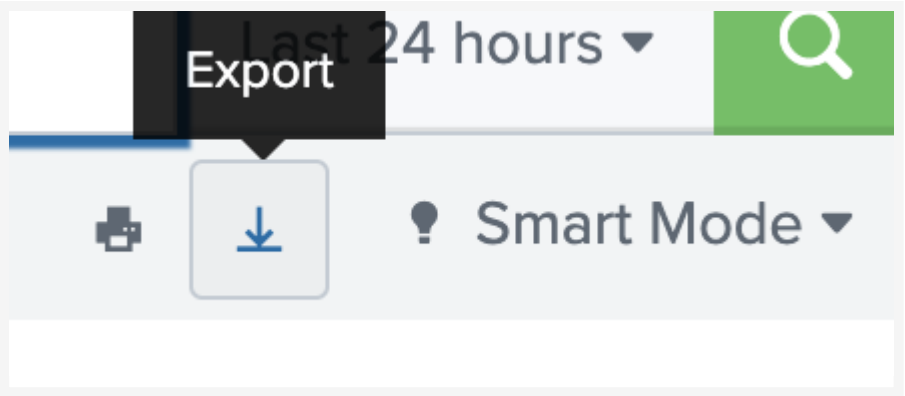
1. Run a search that returns the appropriate sample of target logs. Adjust the number of samples by appending `| head <number_of_samples_desired>` to your search. For example, to grab 100 samples of Cisco ASA firewall data:

```
index=firewall sourcetype=cisco:asa | head 100
```

Copy

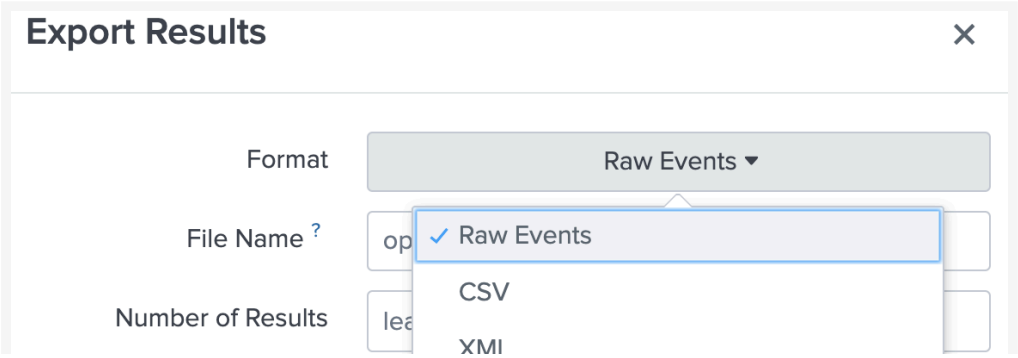
2. Adjust the search as needed to exclude or include the events you want to send.

3. When your search is collecting the desired samples, export the search results:



Exporting search results

4. Then select **Raw Events**, name the file after the source or sourcetype, and save.



6.1.4.3. Access Logs: Apache, ELB, CDN, S3

Recipe for Sampling Access Logs

Access logs are extremely common. They're often emitted by web servers or similar/related technologies (proxies, loadbalancers, etc.), and tend to be highly voluminous. Typical examples include Apache access logs, and CDN logs such as those from Amazon Cloudfront, Amazon S3 Server Access Logs, AWS ELB Access Logs, etc.

For large installations, bringing everything into an analytics tool is often so cost-prohibitive (in terms of storage, resources, and/or license) that most users don't even bother. However, some of the logs contain relevant information when looked at individually (such as errors). The much larger majority contains relevant information when looked at in the aggregate (e.g., successes to determine traffic patterns).

It would be great if we could find a middle ground. With the Sampling Function, you can! Specifically, you can:

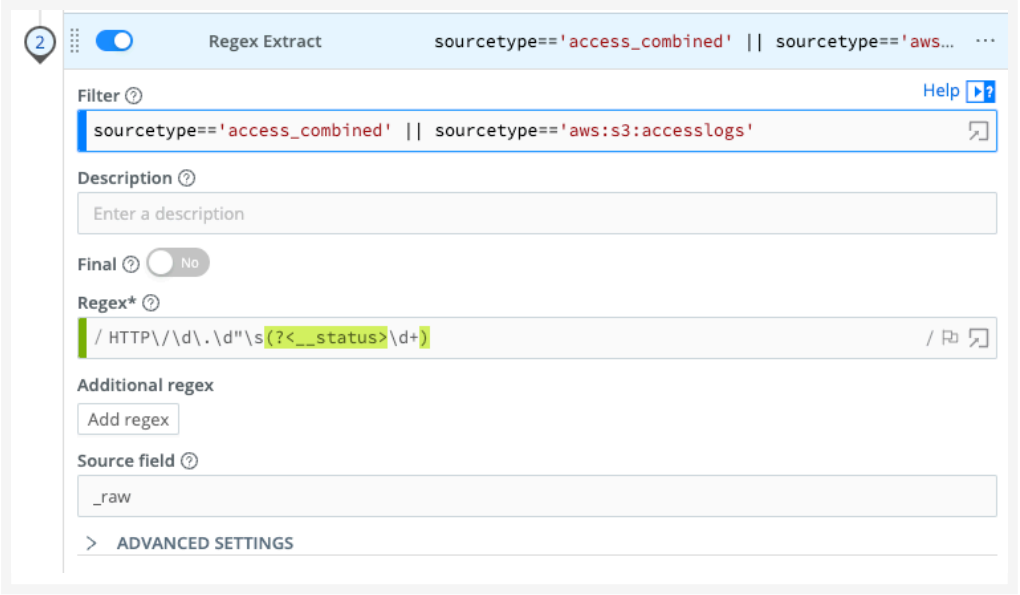
- Ingest enough sample events from the majority category that your aggregate analysis remains statistically significant.
- Ingest **all** events from the minority categories, and perform troubleshooting and introspection with full-fidelity data.

Using `status` as the Sampling Condition

Most of the access logs (including the ones mentioned above) have very similar formats. One quick way to sample is to look at the value of the `status` field. `2XX`s indicate success and tend to be, by far, the most common ones - with `200` being the top. **Therefore, `200` is the perfect candidate for sampling**. All other statuses occur much less frequently, indicate conditions that often need to be looked at, and can be brought in with full fidelity.

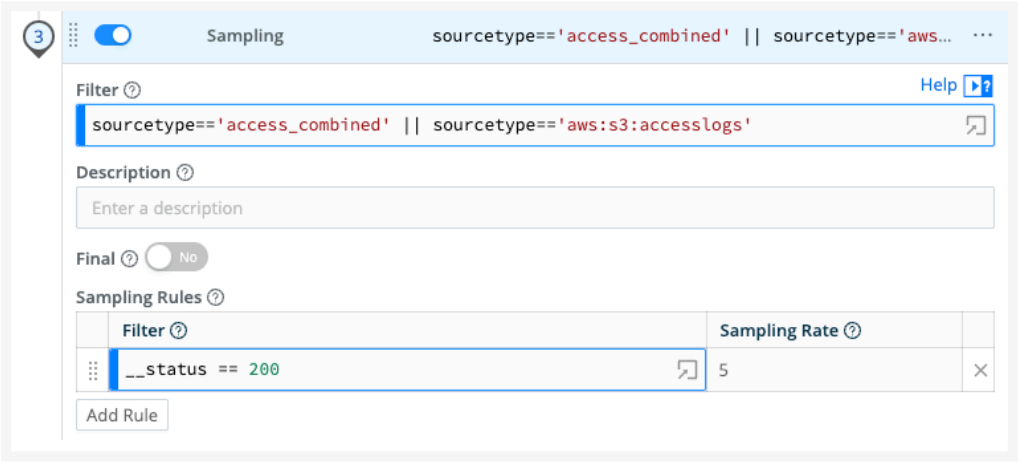
Sample Status 200 at 5:1

1. Add a **Regex Extract** Function that looks at these sourcetypes: `sourcetype=='access_combined' || sourcetype=='aws:s3:accesslogs'`
2. Configure that Function to extract a field called `__status` with this regex: `/HTTP\/\d\.\d"\s(?<__status>\d+)/`



Defining the Regex Extract Function

3. Add a Sampling Function to sample `5:1` when `__status==200`.
4. Save.



Sampling success reponses

Note About Sampling

Each time an event goes through the **Sampling** Function, an index-time `sampled::<rate>` field is added to it. Use this field in your statistical Functions, as necessary.

Other Sourcetypes

Examples of other sourcetypes that will benefit from sampling, but might need a different `__status` extraction regex:

Sourcetype	Filter Expression
Amazon Cloudfront Access Logs	`sourcetype=='aws:cloudfront:accesslogs'`

6.1.4.4. Firewall Logs: VPC Flow Logs, Cisco ASA

Recipe for Sampling Firewall Logs

Firewall logs are another source of important operational (and security) data. Typical examples include Amazon VPC Flow Logs, Cisco ASA Logs, and other technologies such as Juniper, Checkpoint, or pfSense.

As with Access Logs, bringing in **everything** for operational analysis might be cost-prohibitive. But sampling with Cribl Stream can help you:

- Ingest enough sample events from the majority category that your aggregate analysis remains statistically significant. E.g., sample all `ACCEPT`s at `5:1`.
- Ingest **all** events from the minority categories, and perform troubleshooting and introspection with full-fidelity data. E.g., bring in all `REJECT`s.

Sampling VPC Flow Logs

AWS' VPC Flow Logs feature enables you to capture information about the IP traffic going to and from network interfaces in your VPC. Flow Log data can be published to Amazon CloudWatch Logs and Amazon S3.

Typical VPC Flow Logs look like this:

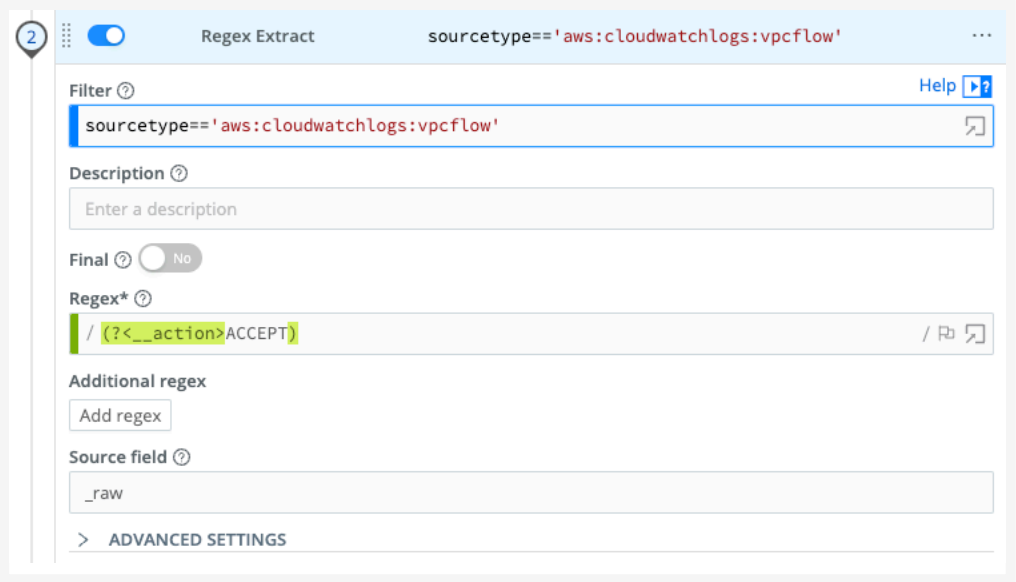
Flow Log Records for Accepted and Rejected Traffic

```
2 123456789010 eni-abc123de 172.31.16.139 172.31.16.21 20641 22 6 20 4249 1418530010
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1418530010 1
```

Copy

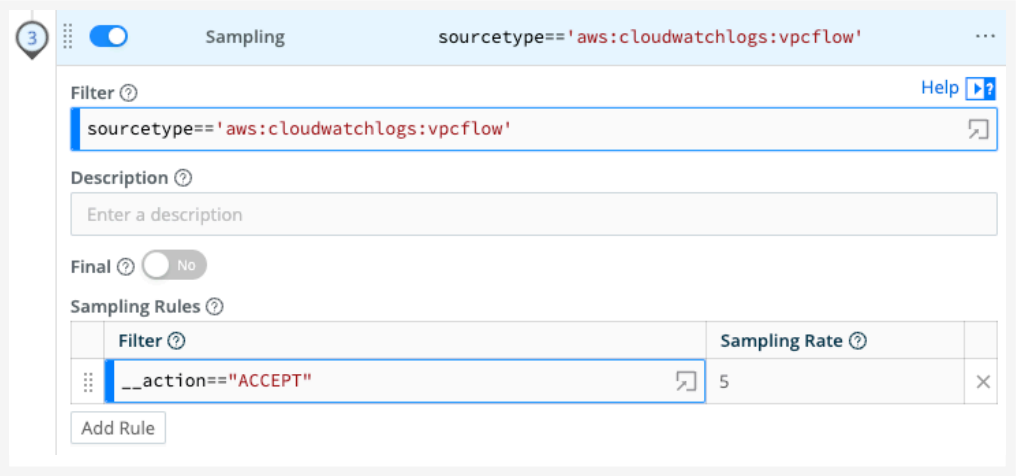
Let's use a **very simple** Filter condition and only look for `ACCEPT` events:

1. Add a **Regex Extract** Function that looks at: `sourcetype=='aws:cloudwatchlogs:vpcflow'`
2. Configure that Function to extract a field called `__action` with this regex: `/(?<__action>ACCEPT)/`



Extracting the `__action` field

3. Add a **Sampling** Function to sample `5:1` when `__action=="ACCEPT"`.
4. Save.



Sampling `ACCEPT` events

Note About Sampling

Each time an event goes through the Sampling Function, an index-time field is added to it: `sampled: <rate>`. It's advisable that you use that in your statistical functions, as necessary.

Other Sourcetypes

Other sourcetypes that will benefit from sampling, but might need a different `__action` extraction regex:

6.2. Event Data Structure And Flow

This section provides a complete reference for how Cribl Stream receives, formats, and segments raw data into structured events.

Core Concepts

- Event Model: Learn about the internal structure of a Cribl Stream event.
- Event Processing Order: A step-by-step breakdown of the evaluation order of operations for incoming data.

Event Breaking and Parsing

- Event Breakers: Explains how event breaking works in Cribl Stream and how to apply them to your raw data to turn it into structured data.
- Event Breaker Types: Learn about the configuration and use cases for each Event Breaker type, allowing you to choose the most efficient parser for your specific log format.

6.2.1. Event Model

Cribl Stream bases all data processing on discrete data entities commonly known as **events**. An event is a collection of key-value pairs (fields). Some Sources deliver events directly, while others might deliver bytestreams that need to be broken up by Event Breakers. Events travel from a Source through Pipelines' Functions, and on to Destinations.

The internal representation of a Cribl Stream event is as follows:

Event Model

```
{
  "_raw": "<body of non-JSON parse-able event>",
  "_time": "<timestamp in Unix time>",
  "__inputId": "<Id/Name of Source that delivered the event>",
  "__other1": "<Internal field1>",
  "__other2": "<Internal field2>",
  "__otherN": "<Internal fieldN>",
  "key1": "<value1>",
  "key2": "<value2>",
  "keyN": "<valueN>",
  "...": "..."
}
```

Copy

Internal Fields

Cribl Stream adds internal fields before Pipeline data processing. They serialize only to Cribl internal Destinations.

A few notes about internal fields:

- Internal fields start with a double-underscore.
- Each Source can add one or many internal fields to each event. For example, Syslog adds both a `__inputId` and a `__srcIpPort` field.
- Treat internal fields as read-only. Modifying them can result in unintended consequences to your data flows.
- Upon arrival from a Source, if an event cannot be JSON-parsed, Cribl assigns all of its content to `_raw`.

- If you do not configure a timestamp for extraction, the Pipeline process assigns the current time (in Unix time) to the `_time` field. (Note that when Cribl Stream emits events based on incoming Kafka messages, it handles the `_time` field a bit differently. See Kafka Source for more information.)
- Cribl reserves the right to change all internal fields that are not documented under Sources or Destinations.

System Fields

Cribl Stream automatically can add system fields to events during post-processing. By default, this includes `cribl_pipe`, a field that identifies the Cribl Stream Pipeline that processed the event. Other additional system fields include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

A few notes about system fields:

- System fields start with the string `cribl_`.
- Treat system fields as read-only. Like internal fields, modifying system fields can result in unintended consequences to your data flows. Adding a Pipeline function to trim system fields will not successfully remove system fields since Cribl Stream adds these fields during post-processing.
- Cribl reserves the right to change all system fields that are not documented under Sources or Destinations.

Use Capture

Use the Capture tool to:

- Troubleshoot events.
- Observe what an event looks like as it travels through the system.

While in Preview (right pane):

1. Select **Capture Data**.
2. In the resulting modal, enter a **Filter expression** to narrow down the events of interest.
3. Select **Capture...** and (optionally) change the default Time and/or Event limits.
4. Select the desired **Where to capture** option. There are four options:

	Option	Description
1	**Before the pre-processing Pipeline**	Capture events right after they're delivered by the respective Input (and after PQ, if enabled).
2	**Before the Routes**	Capture events right after the pre-processing Pipeline. before they go down any Routes (QuickConnect bypasses Routes, and can bypass processing Pipelines.)
3	**Before the post-processing Pipeline**	Capture events right after any Processing Pipeline that handled them, before any post-processing Pipeline.
4	**Before the Destination**	Capture events right after any post-processing Pipeline, before they go out to the configured Destination.

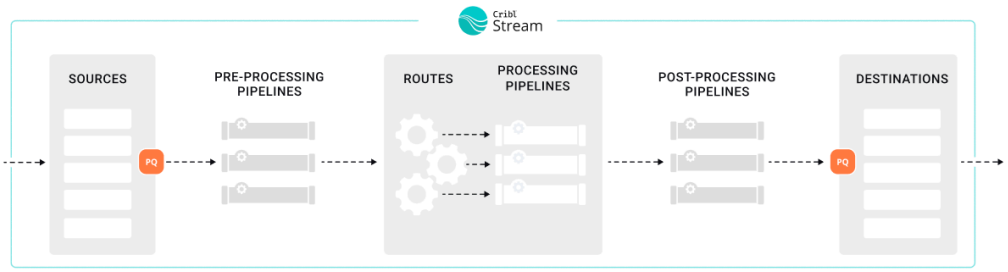
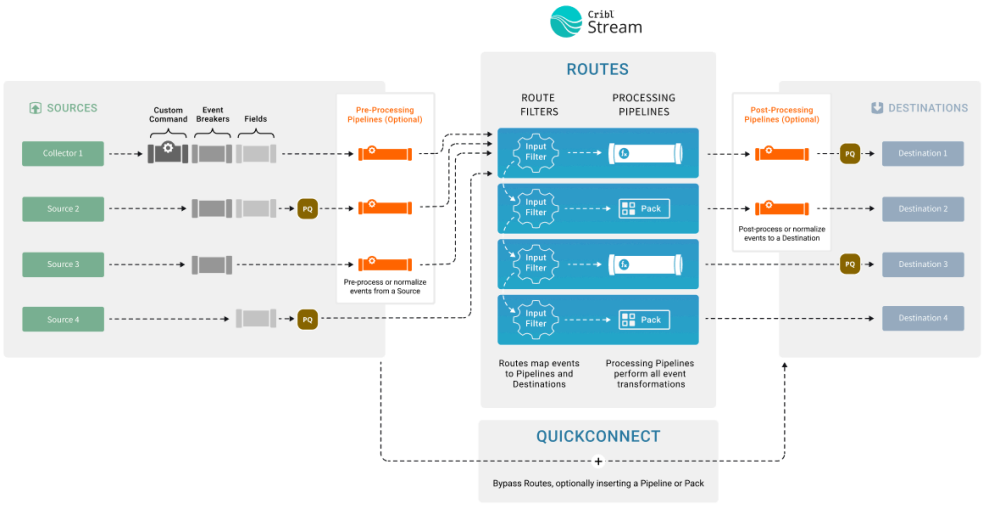


Illustration of the Capture options in the Pipeline

6.2.2. Event Processing Order

The expanded schematic below shows how all events in the Cribl Stream ecosystem are processed linearly, from left to right.



Cribl Stream in great detail

Here are the stages of event processing:

1. Sources: Data arrives from your choice of external providers. (Cribl Stream supports Splunk, HTTP/S, Elastic Beats, Amazon Kinesis/S3/SQS, Kafka, TCP raw or JSON, and many others.) Many internal fields are added during the event parsing within the Source.
2. Custom command: Optionally (where available), you can pass this input's data to an external command before the data continues downstream. This external command will consume the data via `stdin`, will process it and send its output via `stdout`.
3. Event Breakers can, optionally, break up incoming bytestreams into discrete events. (Note that because Event Breakers precede tokens, Breakers cannot see or act on tokens.)
4. Time filters in Collector jobs discard events outside the specified time range, ensuring only time-relevant events proceed further in the workflow.
5. Fields/Metadata: Optionally, you can add these enrichments to each incoming event. You add fields by specifying key-value pairs, per Source, in a format similar to Cribl Stream's Eval Function. Each key defines a field name, and each value is a JavaScript expression (or constant) used to compute the field's value.
6. Persistent queue (Source-side): Optionally, you can enable PQ per Source instance, to disk-buffer and maximize data delivery despite downstream backpressure. Events will be written to a disk queue at this point.
7. The internal field `__inputId` is added at this stage, after the event exits the Source.

8. Pre-processing Pipeline: Optionally, you can use a single Pipeline to condition (normalize) data from this input before the data proceeds further.
9. Routes map incoming events to Processing Pipelines and Destinations. A Route can accept data from multiple Sources, but each Route can be associated with only one Pipeline and one Destination. (QuickConnect bypasses Routes.)
10. Processing Pipelines perform most event transformations. Within a Pipeline, you define these transformations as a linear series of Functions. A Function is an atomic piece of JavaScript code invoked on each event. (QuickConnect optionally bypasses processing Pipelines.)
11. Post-processing Pipeline: Optionally, you can append a Pipeline to condition (normalize) data on its way into its Destination.
12. Persistent queue (Destination-side): Optionally, you can enable PQ per Destination instance, to disk-buffer and maximize data delivery during imbalances between inbound and outbound data rates. Events will be written to a disk queue at this point.
13. Destinations: Each Route/Pipeline combination forwards processed data to your choice of streaming or storage Destination. (Cribl Stream supports Splunk, Syslog, Elastic, Kafka/Confluent, Amazon S3, Filesystem/NFS, and many other options.)

Troubleshoot event processing in Cribl Stream from right to left. Start at the Destination, and check for block status from the Destination back to the Source. If one Worker Group is sending to another, the right-most destination will be in the receiving Worker Group.

Pipelines Everywhere

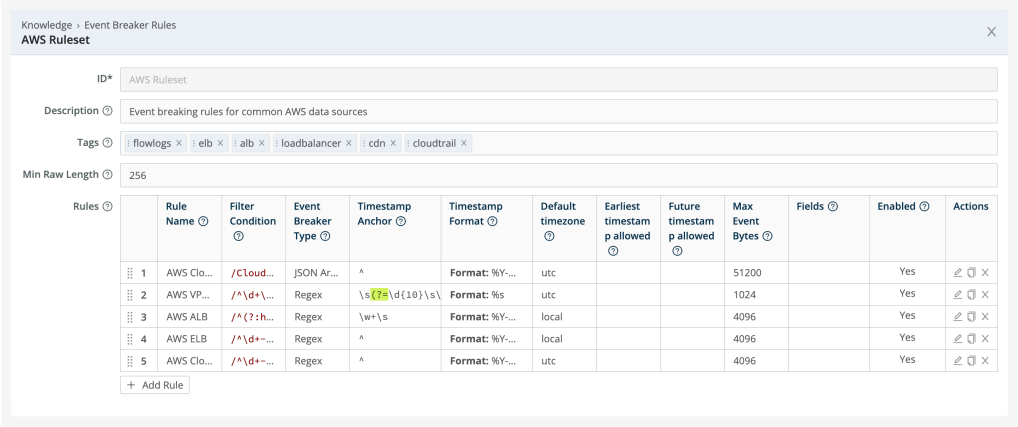
All Pipelines have the same basic internal structure - they're a series of Functions. The three Pipeline types identified above differ only in their position in the system.

6.2.3. Event Breakers

Event Breaker Rulesets are ordered collections of event-breaking rules that help you define the boundaries and structure of raw log data. Once you have converted unstructured raw data into structured events, you can then send them through Pipelines for further routing and data processing.

An Event Breaker Ruleset is a reusable knowledge object stored in the Knowledge Library. It acts as a parsing policy that bundles one or more individual Event Breaker rules into a single configuration that can be applied to Sources or Pipelines using the Event Breaker Function.

Each Ruleset contains a series of event breaking rules. Cribl compares incoming raw data against the filter condition for each rule in turn, based on the order that the rules appear in the Ruleset. For the first rule that matches the raw data, Cribl applies the event-breaking logic defined for that rule.



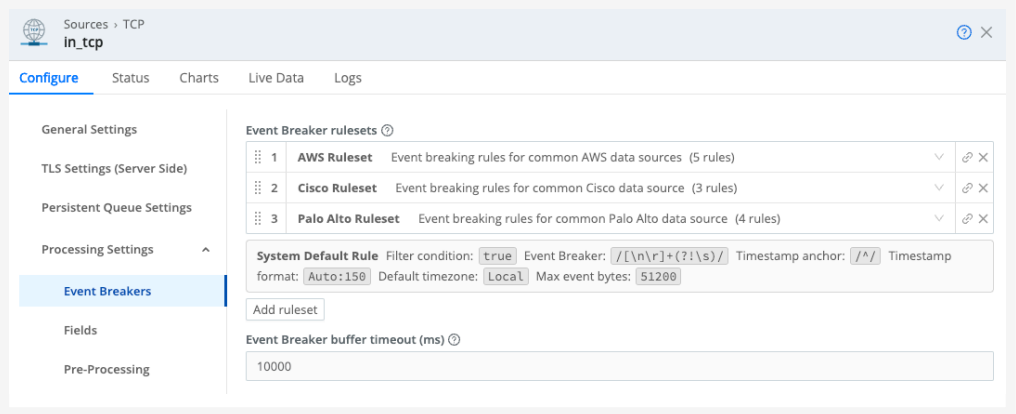
Rules within an example (AWS) ruleset that ships with Cribl Stream

Cribl provides several standard Event Breaker Rulesets for commonly used Sources out-of-the-box:

- Azure Blob Storage
- AWS
- Apache
- Bro
- Cisco
- Cribl
- Cribl Search
- CSV
- Office 365
- Palo Alto
- Splunk Search

- Wiz

You can adjust these standard Rulesets or you can create your own custom Rulesets as needed. Event Breaker rulesets shipped by Cribl will be listed under the **Cribl** tag, while user-built rulesets will be listed under **Custom**. In the case of an ID or Name conflict, the Custom pattern takes priority in listings and search.



Three Event Breaker rulesets added to a Source

How Event Breakers Work

Event Breakers consist of rulesets that can be applied to raw data one of two ways:

- **Directly on the Source:** For Event Breaker support for specific Sources, consult the documentation for that Source.
- **Using the Event Breaker Function:** You can apply event breaking directly in a Pipeline using the Event Breaker Function.

If a Source supports event breaking, you should apply Event Breaking Rulesets there. For Sources that do not support event breaking, use the the Event Breaker Function instead.

When a stream of raw data arrives at a Source or Event Breaker Function, it is processed through the Event Breaker Ruleset in a specific order of operations:

1. **Filter Condition:** When data enters the system, the engine checks it against each filter expression in sequence. If the expression evaluates to `true`, the rule configurations engage for the entire duration of that stream. Otherwise, the engine evaluates the next rule down the line. Different Source types define data streams differently:
    - **TCP Sources:** A stream equates to a single incoming TCP connection. The stream begins when the socket connection is established and ends when that connection closes.
    - **HTTP Sources:** A stream equates to a single HTTP request, with connections potentially being reused across different requests instead of corresponding to the entire connection lifecycle.

- **File-based Sources:** For Sources like File Monitor or Journal Files, each ingested file is treated as a distinct and complete stream, evaluated independently against filters.
- **Sources that support multiplexed data:** For Sources like Splunk S2S protocol, Cribl Stream demultiplexes the incoming stream based on channels. Each channel (which typically maps to a unique `source:sourcetype:host` combination in Splunk Sources) retains the same Event Breaker configuration applied consistently across its entire processing lifecycle.

Remember that Event Breaker Rulesets are evaluated in order, and the first matching rule is applied. If you define a rule with a broad or overly permissive filter, it may capture events that were intended for a more specific rule lower in the list. This means a broadly filtered rule placed high in the list can capture and incorrectly process events that were intended for a more specific rule below it.

To avoid this risk, order your rules from specific to general.

2. **The System Default Rule:** The System Default Rule is a built-in, uneditable set of Event Breaker rules that resides at the end of the ruleset order of operations. It is not displayed in the Event Breakers interface, but it automatically goes into effect for any data event that does not match the filter condition of any rule in your custom Event Breaker Ruleset. Its primary purpose is to ensure that streams of unstructured logs are broken into structured data:

Setting	Value	Explanation
Filter Condition	`true`	A condition of `true` guarantees that every event that makes it past your custom rules will be processed by the default rule.
Event Breaker	`[\n\r]+(?!\s)`	Breaks the event at a newline or carriage return, unless the following character is a whitespace (`\s`). This pattern can handle simple, single-line log formats while attempting to avoid breaking multiline events (like stack traces) where the subsequent lines are indented.
Timestamp Anchor	`^`	Starts scanning for a timestamp at the beginning of the event (`^`).
Timestamp Format	`Auto`	Uses the Auto Timestamp algorithm to automatically detect the timestamp format. This allows the default rule to handle the widest variety of logs without a pre-defined format.
Scan Depth	150 bytes	This limits how far into the event (150 bytes) the Auto Timestamp will look. It optimizes performance by reducing time scanning massive logs for a timestamp that should be near the front.
Max Event Bytes	`51200`	This sets the maximum size (50 KB) that an un-broken event can reach before Cribl forces a break, which handles events that have no newline or timestamp. It prevents a single event from consuming excessive memory.

Setting	Value	Explanation
Default Timezone	`Local`	If the timestamp does not contain timezone information, Cribl defaults to using the Worker Process's local timezone when assigning the `_time` field.

Limitations

Event Breaker Rulesets can be applied to raw data either directly on the Source or using the Event Breaker Function in Pipelines. These two methods have some limitations, as explained here.

Event Breaking in Sources

Event Breakers are directly accessible only on Sources that require incoming events to be broken into a better-defined format. For Event Breaker support for specific Sources, consult the documentation for that Source.

Always validate event breaking before turning on a Source.

Event Breaker Function

The Event Breaker Function has these limitations:

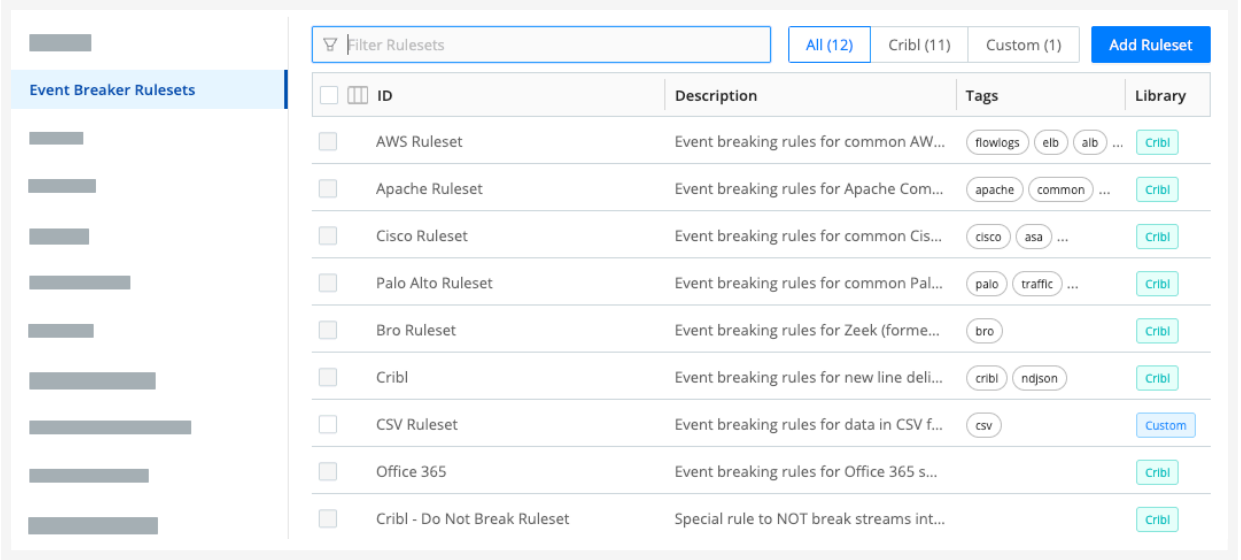
- The Event Breaker Function operates only on data in `_raw`. For other events, move the array to `_raw` and stringify it before applying the Function.
- The largest event that this Function can break is about about 128 MB (`134217728` bytes). Events exceeding this maximum size will be split into separate events, but left unbroken. Cribl Stream will set the `__isBroken` internal field to `false` on these events.

Configure Event Breaker Rulesets

You can add, edit, or customize Event Breaker Rulesets directly in Sources or Packs. You can also configure them in the Knowledge Library.

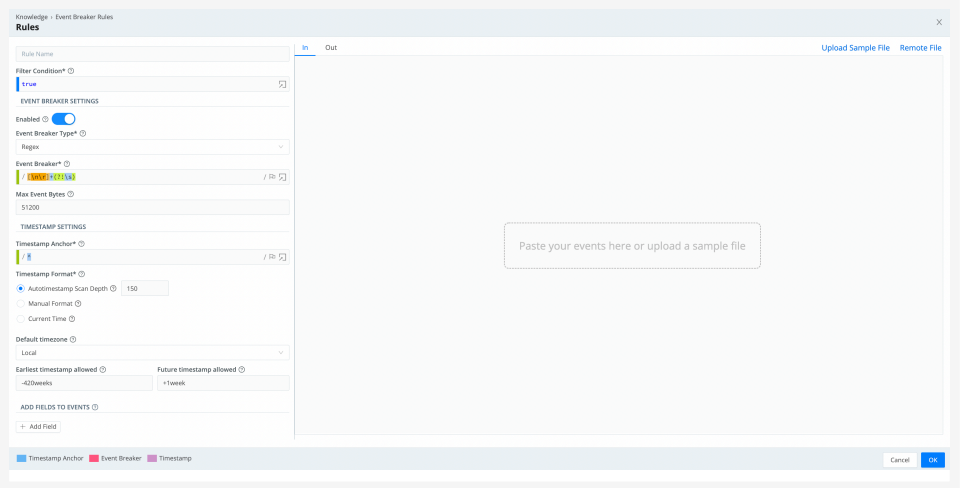
To access the Event Breakers Rulesets from the Knowledge Library:

1. Navigate to **Products** > **Cribl Stream** > **Worker Groups**.
2. Select a Worker Group, then go to **Processing** > **Knowledge** > **Event Breaker Rulesets**.
3. On the resulting **Event Breaker Rulesets** page, you can edit, add, delete, search, and tag Event Breaker rules and rulesets, as necessary.



Event Breaker Rulesets page

4. Select **Add Ruleset** to create a new Event Breaker Ruleset. Enter the Rulset name and an optional description. Then select **Add Rule** within a ruleset to add a new Event Breaker.



Adding a new Event Breaker rule

5. In the **Rules** modal, configure the following settings:
    - **Name:** Enter a unique name for the rule.
    - **Filter condition:** When data enters the system, the engine checks it against each filter expression in sequence. If the expression evaluates to `true`, the rule configurations engage for the entire duration of that stream. See How Event Breakers Work for more detailed information about how this setting works.
6. In the **Event Breaker Settings** section, configure the following settings:
    - **Enabled:** By default this is toggled on. Toggle off to turn this rule off and stop it from being applied for event breaking.
    - **Event Breaker type:** Select one of the available Event Breaker types from this menu. After you select a breaker pattern, it will apply on the data stream continuously. See Event Breaker Types for

guidance on selecting an appropriate Event Breaker type. Specific Event Breakers might have additional settings, so consult the documentation for your selected type for more information.

- **Max Event Bytes:** A break will automatically occur if the accumulated event size reaches this configured byte limit, regardless of whether the regex pattern has been matched.

The highest **Max Event Bytes** value that you can set is about 128 MB (`134217728` bytes). Events exceeding that size will be split into separate events, but left unbroken. Cribl Stream will set these events' `__isBroken` internal field to `false`.

7. In the **Timestamp Settings** section, configure event timestamp settings. See Timestamp Settings for details about each setting.
8. In the **Add Fields to Events** section, you can add one or more fields to events as key-value pairs. In each field's **Value Expression**, you can fully evaluate the field value using JavaScript expressions.

Event Breakers **always** add the `cribl_breaker` field to output events. Its value is the name of the chosen ruleset. Some examples on this page omit the `cribl_breaker` field for brevity, but in real life the field is always added.

Once the rule is configured, validate it using the Data Preview window in Pipelines to see how incoming events are structured after breaking.

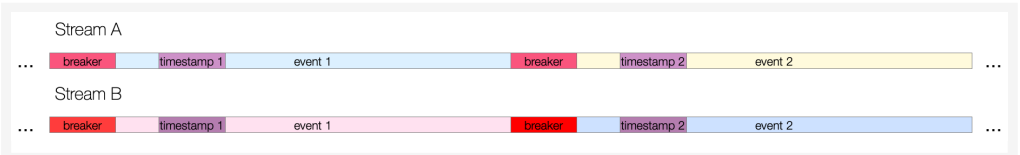
Timestamp Settings

Cribl Stream timestamps events after they are synthesized from streams, ensuring accurate time representation for event processing and analysis.

- If an Event Breaker has set an event's `_time` to the current time - rather than extracting the value from the event itself - it will mark this by adding the internal field `__timestampExtracted: false` to the event.
- Cribl Stream truncates timestamps to three-digit (milliseconds) resolution, omitting trailing zeros.

Anchor: Cribl Stream first identifies a timestamp anchor within the event. This anchor serves as the starting point for timestamp extraction. The end of the first regex match is the start of the timestamp.

The closer an anchor is to the timestamp pattern, the better the performance and accuracy - especially if multiple timestamps exist within an event.



Anchors preceding timestamps

Format: Starting from the anchor, the engine attempts the following methods, stopping after a successful timestamp extraction:

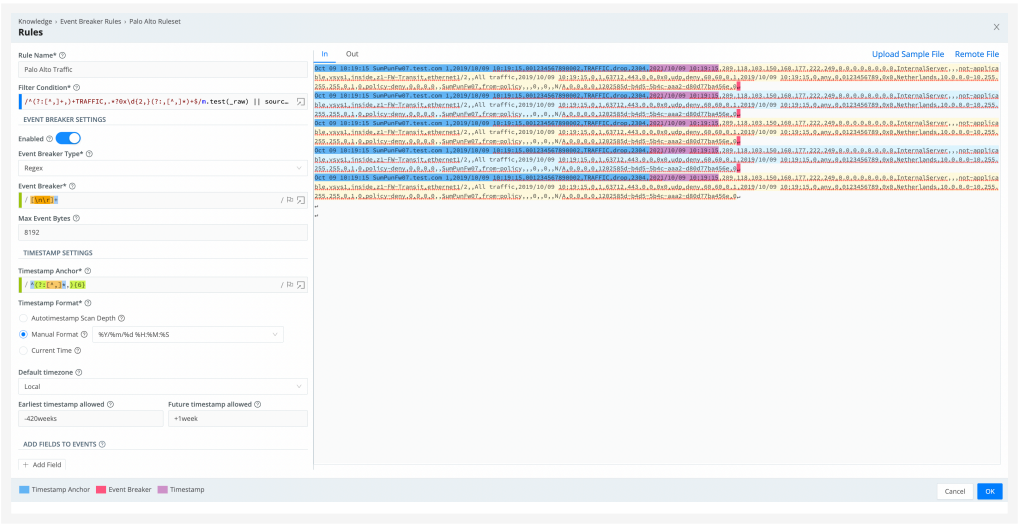
- **Autotimestamp**: Scans up to a configurable depth within the event, attempting to automatically identify and parse a timestamp. This uses the same algorithm as the Auto Timestamp Function and the `C.Time.timestampFinder()` native method.
- **Manual**: Select a `strptime` format to parse the timestamp immediately following the anchor. The anchor must lead the engine **right before** the timestamp pattern begins.
- **Current time**: The event is timestamped with the current time.

Default timezone: Assigns a timezone to timestamps that lack timezone information.

Earliest timestamp allowed and **Future timestamp allowed**: Constrain the parsed timestamp to prevent unrealistic values, setting timestamps outside these bounds to the default time.

Example Rule

This example rule breaks on newlines and uses Manual timestamping after the sixth comma, as indicated by this pattern: `^(?:[^,]*,){6}`.



An Event Breaker rule

Export and Import Rulesets

You can export and import Event Breaker Rulesets as JSON files. This facilitates sharing Rulesets among Worker Groups or Cribl Stream deployments.

To export a Ruleset:

1. Select to open an existing ruleset, or create a new ruleset.
2. In the resulting modal, select **Advanced Mode** to open the JSON editor.
3. Select **Export**, select a destination path, and name the file.

To import any Ruleset that has been exported as a valid JSON file:

1. Create a new ruleset.
2. In the resulting modal, select **Advanced Mode** to open the JSON editor.
3. Select **Import**, and choose the file you want.
4. Select **OK** to get back to the **New Ruleset** modal.
5. Select **Save**.

Every Ruleset must have a unique value in its top `id` key. Importing a JSON file with a duplicate `id` value will fail at the final **Save** step, with a message that the ruleset already exists. You can remedy this by giving the file a unique `id` value.

Build Event Breakers From Files (Edge Only)

You can use Cribl Edge to apply and author Event Breakers while exploring files, including files that you have not saved as sample files. This option includes remote files that are viewable only from Edge. See Exploring Cribl Edge on Linux or Exploring Cribl Edge on Windows.

Troubleshoot Event Breakers

If you notice fragmented events, check whether Cribl Stream has added a `__timeoutFlush` internal field to them. This diagnostic field's presence indicates that the events were flushed because the Event Breaker buffer timed out while processing them. These timeouts can be due to large incoming events, backpressure, or other causes.

Specify Minimum `_raw` Length

If you notice that the same Collector or Source is applying inconsistent Event Breakers to different datasets - such as intermittently falling back to the **System Default Rule** - you can address this by adjusting Breakers' **Min Raw Length** setting.

When determining which Event Breaker to use, Cribl Stream creates a substring from the incoming `_raw` field, at the length set by rules' **Max Event Bytes** parameters. If Cribl Stream does not get a match, it checks whether the substring of `_raw` has at least the threshold number of characters set by **Min Raw Length**. If so, Cribl Stream does not wait for more data.

6.2.4. Event Breaker Types

You can apply several types of Event Breakers to incoming data streams (listed alphabetically):

- Azure Virtual Network (VNet) Flow: For Azure VNet flow logs from Azure Blob Storage and Azure Event Hubs.
- CSV: For data that adheres to the CSV standard, such as output from databases, financial logs, spreadsheets, or network device reports. This Event Breaker handles row delineation, field extraction, and quoting rules for CSV data.
- File Header: For logs that use a standard file header structure, such as Bro, IIS, or Apache access logs. Use the File Header Breaker instead of Regex when the field names are complex but consistent across the log file.
- JSON Array: For single, large JSON objects containing a nested array of records. This Event Breaker is particularly useful for logs from services where a single log entry contains an array of event records, such as AWS CloudTrail, EKS CloudWatch, or Google Cloud Audit Logs.
- JSON Newline Delimited: For log data where each event is a complete, self-contained JSON object that is followed by a newline. This format is widely used by cloud providers, modern logging agents, and data platforms.
- Regex: For any log data that doesn't fit the structured formats of the CSV, JSON Array, or File Header breakers. This includes multi-line logs where a single event spans multiple lines and non-standard single-line logs that use custom formats.
- Timestamp: For log streams that use non-standard or highly varied timestamp formats, especially where writing a single, complex regex would be difficult or inefficient. This Event Breaker is also good for multi-line logs where every new event begins with a timestamp, but the exact format isn't guaranteed or known ahead of time.

If you're unsure which Event Breaker to use, the Regex Event Breaker is a good default Event Breaker because of its flexibility.

See Event Breakers for general information about event breakers.

6.2.4.1. Azure Virtual Network (VNet) Flow Event Breaker

The Azure Virtual Network (VNet) Flow Event Breaker segments and structures the highly nested JSON data generated by Azure VNet logs. It takes in raw data that uses the Azure VNet flow logs schema to identify individual flow tuples, then outputs the data into a structured data format.

Use this Event Breaker to ingest Azure VNet flow logs from Azure Blob Storage.

See Event Breakers for general information about event breakers.

Azure has deprecated Network Security Group (NSG) flow logs in favor of Azure VNet Flow logs. To optimize system performance, use this Azure VNet Flow Event Breaker to de-aggregate records at the Source. This is more efficient than ingesting raw blobs and using the JSON Unroll Function later in your Pipeline.

Settings

Selecting the Azure VNet Flow Event Breaker type does not expose any additional settings beyond the standard Event Breaker settings. See Event Breakers for a description of the available settings.

Configuration Example

The following is an example of data input before the Azure VNet Flow Event Breaker processes it:

Example raw input

```
{
    "records": [
        {
            "time": "2022-09-14T09:00:52.5625085Z",
            "flowLogVersion": 4,
            "flowLogGUID": "66aa66aa-bb77-cc88-dd99-00ee00ee00ee",
            "macAddress": "112233445566",
            "category": "FlowLogFlowEvent",
            "flowLogResourceID": "/SUBSCRIPTIONS/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e
            "targetResourceID": "/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e/
            "operationName": "FlowLogFlowEvent",
            "flowRecords": {
                "flows": [
                    {
                        "aclID": "00aa00aa-bb11-cc22-dd33-44ee44ee44ee",
                        "flowGroups": [
                            {
                                "rule": "DefaultRule_AllowInternetOutBound",
                                "flowTuples": [
                                    "1663146003599,10.0.0.6,192.0.2.180,23956,443,6,O
                                    "1663146003606,10.0.0.6,192.0.2.180,23956,443,6,O
                                    "1663146003637,10.0.0.6,203.0.113.17,22730,443,6,
                                    "1663146003640,10.0.0.6,203.0.113.17,22730,443,6,
                                    "1663146004251,10.0.0.6,203.0.113.17,22732,443,6,
                                    "1663146004251,10.0.0.6,203.0.113.17,22732,443,6,
                                    "1663146004622,10.0.0.6,203.0.113.17,22734,443,6,
                                    "1663146004622,10.0.0.6,203.0.113.17,22734,443,6,
                                    "1663146017343,10.0.0.6,198.51.100.84,36776,443,O
                                    "1663146022793,10.0.0.6,198.51.100.84,36776,443,O
                                ]
                            }
                        ]
                    },
                    {
                        "aclID": "00aa00aa-bb11-cc22-dd33-44ee44ee44ee",
                        "flowGroups": [
                            {
                                "rule": "BlockHighRiskTCPPortsFromInternet",
                                "flowTuples": [
                                    "1663145998065,101.33.218.153,10.0.0.6,55188,22,O
                                    "1663146005503,192.241.200.164,10.0.0.6,35276,119
                                ]
                            `
```

```
                        },
                        {
                            "rule": "Internet",
                            "flowTuples": [
                                "1663145989563,192.0.2.10,10.0.0.6,50557,44357,6,
                                "1663145989679,203.0.113.81,10.0.0.6,62797,35945,
                                "1663145989709,203.0.113.5,10.0.0.6,51961,65515,6
                                "1663145990049,198.51.100.51,10.0.0.6,40497,40129
                                "1663145990145,203.0.113.81,10.0.0.6,62797,30472,
                                "1663145990175,203.0.113.5,10.0.0.6,51961,28184,6
                                "1663146015545,192.0.2.10,10.0.0.6,50557,31244,6,
                            ]
                        }
                    ]
                }
            ]
        }
    ]
}
```

Copy

Output

The Azure VNet Flow Event Breaker unrolls the nested `flowTuples` array and maps the comma-separated values into structured fields. It flattens the metadata (like the `rule` and `resourceId`) into each output event, ensuring that context is preserved even after the original large object is split.

From the example raw data, the Azure VNet Event Breaker would generate two output events:

Example Output

```
{
  "_raw": "1663146003599,10.0.0.6,192.0.2.180,23956,443,6,O,B,NX,0,0,0,0",
  "flowLogVersion": 4,
  "flowLogGUID": "66aa66aa-bb77-cc88-dd99-00ee00ee00ee",
  "macAddress": "112233445566",
  "category": "FlowLogFlowEvent",
```

6.2.4.2. CSV Event Breaker

The CSV Event Breaker can take in Comma Separated Value (CSV) data streams and automatically extract fields. It can process structured, tabular data where a header row defines the field names for all subsequent rows.

Use this Event Breaker:

- Whenever you are ingesting data that adheres to the CSV standard.
- For output from databases, financial logs, spreadsheets, or network device reports. The Event Breaker handles row delineation, field extraction, and quoting rules for this type of data.

See Event Breakers for general information about event breakers.

Settings

Selecting the CSV Event Breaker type exposes these additional settings:

- **Delimiter**: The character used to separate or split individual field values within a row. For standard CSV, the default is a comma (`,`). Change the delimiter for TSV (Tab Separated Values) or when dealing with logs that use a pipe (`|`) or semicolon (`;`) as a separator. Defaults to `,`.
- **Quote Char**: The character used to enclose and protect field values that contain the delimiter character itself. For example, if a field is `"10,000"`, the quotes tell the breaker that the comma is part of the value, not a separator. Defaults to `"`.
- **Escape Char**: The character used to indicate that the next character should be treated literally. It is often used to escape the **Quote Char** itself within a quoted field. For example, to include a literal double-quote inside a quoted field, you might use an escape character: `"He said, ""Hello.""` Defaults to `"`.

Configuration Example

The following is an example of data input before the CSV Event Breaker processes it:

Example raw input - CSV format

```
time,host,source,model,serial,bytes_in,bytes_out,cpu
1611768713,"myHost1","anet","cisco","ASN4204269",11430,43322,0.78
1611768714,"myHost2","anet","cisco","ASN420423",345062,143433,0.28
```

Copy

Output

The CSV Event Breaker treats the first line of the input as the header row and automatically extracts those names as event fields for all subsequent rows. It will also properly add quotes around all values in the `_raw` field of the output events, regardless of their initial state.

From the example raw data, the CSV Event Breaker would generate two output events:

Example CSV Output as JSON


```
{
  "_raw": "\"1611768713\",\"myHost1\",\"anet\",\"cisco\",\"ASN4204269\",\"11430\",\"4
  "time": "1611768713",
  "host": "myHost1",
  "source": "anet",
  "model": "cisco",
  "serial": "ASN4204269",
  "bytes_in": "11430",
  "bytes_out": "43322",
  "cpu": "0.78",
  "_time": 1611768713
}
{
  "_raw": "\"1611768714\",\"myHost2\",\"anet\",\"cisco\",\"ASN420423\",\"345062\",\"1
  "time": "1611768714",
  "host": "myHost2",
  "source": "anet",
  "model": "cisco",
  "serial": "ASN420423",
  "bytes_in": "345062",
  "bytes_out": "143433",
  "cpu": "0.28",
  "_time": 1611768714
}
```


Copy

6.2.4.3. File Header Event Breaker

The File Header Event Breaker ingests highly structured log formats where the field names are explicitly defined in a header section at the top of the file. This Event Breaker relies on a combination of regular expressions to first identify the header, and then use the captured field names to parse all subsequent data lines.

Use this Event Breaker:

- For logs that use a standard file header structure, such as Bro, IIS, or Apache access logs.
- Instead of a simple Regex Breaker when the field names are complex but consistent across the log file.

See Event Breakers for general information about event breakers.

Settings

The File Header Event Breaker requires you to define the structure of the log using a set of coordinated regular expressions:

- **Header line:** The regex used to match and identify a line as part of the header section. Any line that matches this pattern will be treated as metadata and not as an event. For example, this expression matches any line starting with a hash symbol: `^#`.
- **Field delimiter:** The regex used to split the field names captured by the Field Regex into individual field names. This delimiter is also implicitly used to break the subsequent data lines into values. For example, this expression matches one or more whitespace characters, common in space-separated log formats (such as Bro): `\s+`.
- **Fields regex:** This is the most critical regex. It must have exactly one capturing group that isolates the list of field names from the rest of the header text. This captured group is what the Field Delimiter will then process. For example, this expression matches a line starting with `#fields` and captures everything that follows the label into a single group: `^#[Ff]ields[:]?\s+(.*)`.
- **Null value:** The string representation of a null or empty value in the data lines. Fields with this value will not be added to the event object, keeping the event clean. The field is blank by default. Enter a string such as `-` if necessary, which is common in firewall or security logs.
- **Clean fields:** Toggling this setting on will sanitize the extracted field names by replacing any non-alphanumeric characters with an underscore (`_`). This is highly recommended to ensure compatibility with downstream systems like Splunk or Elasticsearch. For example, a field named `id.orig_h` becomes `id_orig_h`.

Configuration Example

The following is an example of data input before the File Header Event Breaker processes it. The header section is typically at the top of the file. It can be a single line or more:

Example raw input - File Header format

```
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p
#types  time    string  addr    port    addr    port    enum
1331904608.080000       -     192.168.204.59  137     192.168.204.255 137     udp
1331904609.190000       -     192.168.202.83  48516   192.168.207.4   53      udp
```

Copy

Example Settings

Setting	Value	Purpose
Header Line	`^#`	Matches all header lines.
Field Delimiter	`\s+`	Separates fields by whitespace.
Field Regex	`^#[Ff]ields[:]?\s+(.*)`	Captures the field names.
Clean Fields	On	Converts `id.orig_h` to `id_orig_h`.

Output

In the output, the Header Line (starting with `#fields`) and the Type Line (starting with `#types`) are ignored. The field names extracted from the Field Regex line are used to key the values in the output events.

From the example raw data, the File Header Event Breaker would generate two output events:

Example Output as JSON

```
{
  "_raw": "1331904608.080000       -     192.168.204.59  137     192.168.204.255 137
  "ts": "1331904608.080000",
  "id_orig_h": "192.168.204.59",
  "id_orig_p": "137",
  "id_resp_h": "192.168.204.255",
  "id_resp_p": "137",
  "proto": "udp",
  "_time": 1331904608.08
}
{
  "_raw": "1331904609.190000       -     192.168.202.83  48516   192.168.207.4   53
  "ts": "1331904609.190000",
  "id_orig_h": "192.168.202.83",
  "id_orig_p": "48516",
```

6.2.4.4. JSON Array Event Breaker

The JSON Array Event Breaker handles log formats where multiple distinct records are bundled together within a single top-level JSON object. It takes one large event and performs a targeted unrolling operation: it identifies an array nested within that parent JSON object and splits each element of that array into a separate, individual event.

Use this Event Breaker for:

- Logs where a single log entry contains an array of event records.
- Sources like AWS CloudTrail, EKS CloudWatch, or Google Cloud Audit Logs.

If a Source supports Event Breakers (such as AWS Sources), it is much more efficient to unroll JSON in an Event Breaker rather than a Pipeline Function.

See Event Breakers for general information about event breakers.

Settings

The JSON Array Event Breaker has these settings:

- **Array field:** Optional path to array in a JSON event with records to extract. For example, `Records`.
- **Parent fields to copy:** This setting allows you to retain key metadata from the original parent event and include it with each of the extracted child events. By default, only the extracted events from the array are passed down the Pipeline. For example, you can use this to copy `host`, `source`, `region`, or other contextual information that is not contained within the array itself.

  To specify which fields to copy, enter a comma-separated list of top-level field names. You can also use `*` as a wildcard to copy all top-level fields. Be aware that:

  - This setting only supports copying fields from the top level of the JSON document. Nested fields are not supported.
  - The `Array field` you specify to be unrolled is always excluded from the copied fields. This prevents redundant data.
  - Enclose field names containing special characters (such as `my-field`) in single or double quotes.
- **JSON extract fields:** This toggle controls whether Cribl Stream parses the JSON array and automatically extracts key-value pairs from each event:
  - When enabled, Cribl Stream will fully parse each JSON object and extract all fields (such as `user_id`, `status`, `event_type`). This is the ideal setting for sending data to Destinations that

need structured data for analysis like Splunk or Elasticsearch for searching, filtering, and dashboard-building. When this toggle is on, it exposes the **Timestamp field** setting.

    - When disabled, Cribl Stream will not extract any fields from the JSON objects. Only the `_raw` (the full event text) and `time` will be available. Choose this option if your primary goal is to send raw data for archival purposes with the highest possible throughput.

The **JSON extract fields** toggle impacts the **Timestamp format** setting in the Timestamp Settings. If the **JSON extract fields** toggle is off, do not select the **Manual format** option. If the toggle is on, you can select any option.

- **Timestamp field:** This optional setting appears when **JSON extract fields** is toggled on. This allows you to specify a path to the timestamp field within each extracted event. For example, `eventTime` or `level1.level2.eventTime`. Cribl Stream will use this value to set the event's `_time` field.
- **Max event bytes**: The maximum size (in number of bytes) that an event can be before being flushed to the Pipelines.
- **Event byte limit**: The maximum size (in number of bytes) that an event can be before being flushed to the Pipelines.

Configuration Examples

The following is an example of data input before the JSON Array Event Breaker processes it:

Example raw input - JSON (Array)

```
{
  "someField": 42,
  "someArray": [1, 2, 3],
  "myObjectArray": [
    {"one": 1, "two": 2},
    {"three": 3, "four": 4}
  ],
  "Records": [
    { "eventVersion": "1.0" },
    { "eventVersion": "2.0" }
  ],
  "myObject": {
    "someField": {
      "test1": 1,
      "test2": "two"
    },
    anotherArray: [
      {"id": 1},
      {"id": 2}
    ]
  }
}
```

Copy

Example 1 - Copy All Top-Level Fields

This first example shows how to use the wildcard `*` to copy all top-level fields from the original JSON event to each of the new, extracted events.

JSON Configuration | User Interface Configuration

```
{
  "parentFieldsToCopy": ["*"],
  "jsonArrayField": "Records"
}
```

Copy

With these configurations, the original raw data will have this output:

Example 1 - Copy All Top-Level Fields

```
[
  {
    _raw: "{ "eventVersion": "1.0" }",
    "someField": 42,
    "someArray": [1, 2, 3],
    "myObjectArray": [
      {"one": 1, "two": 2},
      {"three": 3, "four": 4}
    ],
    "myObject": {
      "someField": {
        "test1": 1,
        "test2": "two"
      },
      anotherArray: [
        {"id": 1},
        {"id": 2}
      ]
    }
  },
  {
    _raw: "{ "eventVersion": "2.0" }",
    "someField": 42,
    "someArray": [1, 2, 3],
    "myObjectArray": [
      {"one": 1, "two": 2},
      {"three": 3, "four": 4}
    ],
    "myObject": {
      "someField": {
        "test1": 1,
        "test2": "two"
      },
      anotherArray: [
        {"id": 1},
        {"id": 2}
      ]
    }
  }
]
```

Copy

Example 2 - Copy Specific Fields

This example shows how to select only specific top-level fields to copy. By providing a comma-separated list of field names, you can control which metadata carries over to the extracted events, resulting in smaller events and reduced data volume.

JSON Configuration | User Interface Configuration

```
{
  "parentFieldsToCopy": ["someField", "myObject"],
  "jsonArrayField": "Records"
}
```

Copy

With these configurations, the original raw data will have this output:

Example 2 - Copy Specific Fields

```
[
  {
    _raw: "{ "eventVersion": "1.0" }",
    "someField": 42,
    "myObject": {
      "someField": {
        "test1": 1,
        "test2": "two"
      },
      anotherArray: [
        {"id": 1},
        {"id": 2}
      ]
    }
  },
  {
    _raw: "{ "eventVersion": "2.0" }",
    "someField": 42,
    "myObject": {
      "someField": {
        "test1": 1,
        "test2": "two"
      },
      anotherArray: [
        {"id": 1},
        {"id": 2}
      ]
    }
  }
]
```

Copy

Example 3 - Use a Wildcard with a Prefix

You can also use a wildcard with a prefix to copy fields that share a common name. This example uses `some*` to copy any top-level fields that start with "some," such as `someField` and `someArray`. This is useful when you have a consistent naming convention for related fields.

JSON Configuration | User Interface Configuration

```
{
  "parentFieldsToCopy": ["some*"],
  "jsonArrayField": "Records"
}
```

6.2.4.5. JSON New Line Delimited Event Breaker

The JSON New Line Delimited Event Breaker handles Newline Delimited JSON (NDJSON) data streams. It treats every newline character (`\n`) as an event boundary, and then automatically parses the JSON object on that line, extracting all key-value pairs as fields. This format is widely used by cloud providers, modern logging agents, and data platforms.

Use this Event Breaker:

- For log data where each event is a complete, self-contained JSON object that is followed by a newline.

Because this Event Breaker relies solely on the newline character, you cannot use it to parse multi-line JSON objects, or single events that contain embedded newlines. Instead, use either the JSON Array Event Breaker or Regex Event Breaker.

See Event Breakers for general information about event breakers.

Settings

This Event Breaker does not require any additional settings beyond the basic Event Breaker settings. That is because it uses a strict format:

- Events are automatically broken by a newline character (`\n`).
- All top-level fields within each JSON object are automatically extracted and added to the event.

Configuration Example

The following is an example of data input before the JSON New Line Delimited Event Breaker processes it:

Example raw input - NDJSON

```
{"time":"2020-05-25T18:00:54.201Z","cid":"w1","channel":"clustercomm","level":"info",
{"time":"2020-05-25T18:00:54.246Z","cid":"w0","channel":"clustercomm","level":"info",
```

Copy

Output

The JSON New Line Delimited Event Breaker processes an NDJSON stream by treating each line of the input as a separate event. All fields are automatically extracted. The event breaks cleanly at the end of each JSON object.

From the example raw data, the JSON New Line Delimited Event Breaker would generate two output events:

Example CSV Output as JSON


```
{
  "_raw": "{\"time\":\"2020-05-25T18:00:54.201Z\",\"cid\":\"w1\",\"channel\":\"cluste
  "time": "2020-05-25T18:00:54.201Z",
  "cid": "w1",
  "channel": "clustercomm",
  "level": "info",
  "message": "metric sender",
  "total": 720,
  "dropped": 0,
  "_time": 1590429654.201,
}
{
  "_raw": "{\"time\":\"2020-05-25T18:00:54.246Z\",\"cid\":\"w0\",\"channel\":\"cluste
  "time": "2020-05-25T18:00:54.246Z",
  "cid": "w0",
  "channel": "clustercomm",
  "level": "info",
  "message": "metric sender",
  "total": 720,
  "dropped": 0,
  "_time": 1590429654.246,
}
```


Copy

6.2.4.6. Regex Event Breaker

The Regex Event Breaker is the default and most flexible Event Breaker type, allowing you to define event boundaries using regular expressions (regex). It gives you more granular control over how unstructured or non-standard logs are segmented into individual events.

Use this Event Breaker for:

- Any log data that doesn't fit the structured formats of the CSV, JSON Array, or File Header breakers.
- Multi-line logs where a single event spans multiple lines and non-standard single-line logs that use custom formats.

See Event Breakers for general information about event breakers.

Considerations

When using the Regex Event Breaker, keep this in mind:

- **Breaks are continuous:** The pattern applies continuously to the data stream. Any content before the match is considered part of the current event, and the break occurs at the start of the match.
- **Consuming and retaining content:** By default, the matched content of the Event Breaker regex is consumed (thrown away). To define the break point without discarding the content that signifies the new event (such as a timestamp), use a positive lookahead regex, such as `(?=pattern)`. This is the standard practice for multiline logs.
- **Avoid capturing groups:** Do not use capturing groups (parentheses like `(pattern)`) inside the Event Breaker pattern. Using them will cause further, often unintended, splitting of the stream.

Settings

The Regex Event Breaker has two relevant settings:

- **Event Breaker:** The regular expression that identifies the start of a new event. Breaking occurs at the beginning of the matched pattern.
- **Max Event Bytes:** A break will automatically occur if the accumulated event size reaches this configured byte limit, regardless of whether the regex pattern has been matched.

Configuration Example

The following is an example of multi-line log data input before the Regex Event Breaker processes it:

Example raw input - Non-standard log format

```
2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP feed, then you can progr
   Try to connect the FTP sensor, maybe it will connect the digital bus!
   Try to navigate the AGP panel, maybe it will quantify the mobile alarm!
2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP feed, then you can progr
   Try to connect the FTP sensor, maybe it will connect the digital bus!
   Try to navigate the AGP panel, maybe it will quantify the mobile alarm!
```

Copy

Example Settings

Setting	Value	Purpose
Event Breaker	`[\n\r]+(?=\d+-\d+-\d+\s\d+:\d+:\d+)`	This setting breaks after a newline or carriage return, but only if followed by a timestamp pattern.
Max Event Bytes	`51200`	The default setting.

Output

From the example raw data, the Regex Event Breaker would generate two output events:

Example Output as JSON

```
{
  "_raw": "2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP feed, then yo
  "_time": 1589920332
}
{
  "_raw": "2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP feed, then yo
  "_time": 1589920332
}
```

Copy

6.2.4.7. Timestamp Event Breaker

The Timestamp Event Breaker is a low-configuration method to break events at the start of any line where Cribl Stream detects a valid timestamp.

Use this Event Breaker for:

- Log streams that use non-standard or highly varied timestamp formats where writing a single, complex regex would be difficult or inefficient.
- Multi-line logs (like application stack traces) where every new event begins with a timestamp, but the exact format isn't guaranteed or known ahead of time.

While highly flexible, the Timestamp Breaker may have a slight performance trade-off compared to a precise Regex Breaker because it requires more processing to dynamically detect the timestamp pattern. For maximum precision and performance on known log formats, use a Regex Event Breaker with a positive lookahead instead.

See Event Breakers for general information about event breakers.

Settings

The Timestamp Event Breaker is useful as an out of the box Event Breaker for event segmentation:

- There is no field to input a regex or pattern.
- A new event is defined when Cribl Stream identifies a valid time value at the beginning of a line.

Configuration Example

The following is an example of data input before the Timestamp Event Breaker processes it:

Example raw input - Timestamp format

```
{"level":"debug","ts":"2021-02-02T10:38:46.365Z","caller":"sdk/sync.go:42","msg":"Har
{"level":"debug","ts":"2021-02-02T10:38:56.365Z","caller":"sdk/sync.go:42","msg":"Har
```

Copy

Output

From the example raw data, the Timestamp Event Breaker would generate two output events:

Example CSV Output as JSON


```
{
  "_raw": "{\"level\":\"debug\",\"ts\":\"2021-02-02T10:38:46.365Z\",\"caller\":\"sdk/
  "_time": 1612262326.365
}
{
  "_raw": "{\"level\":\"debug\",\"ts\":\"2021-02-02T10:38:56.365Z\",\"caller\":\"sdk/
  "_time": 1612262336.365
}
```


Copy

6.3. Routes

Before incoming events are transformed by a processing Pipeline, Cribl Stream uses a set of filters to first select a **subset** of events to deliver to the correct Pipeline. This selection is normally made via Routes.

Don't Need Routes?

Routes are designed to filter, clone, and cascade incoming data across a related set of Pipelines and Destinations. If all you need are independent connections that link parallel Source/Destination pairs, you can use Cribl Stream's QuickConnect rapid visual configuration tool as an alternative to Routes.

Accessing Routes

To access Routes:

1. In the sidebar, select **Worker Groups**, then select a Worker Group.
2. On the **Worker Groups** submenu, select **Routing**, then **Data Routes**.

   On a Single-instance deployment, select **Routing** straight from the **Home** submenu.
3. To configure a new Route, select **Add Route**.

How Do Routes Work

Routes apply filter expressions on incoming events to send matching results to the appropriate Pipeline or Pack. Filters are JavaScript-syntax-compatible expressions that are configured with each Route. Examples are:

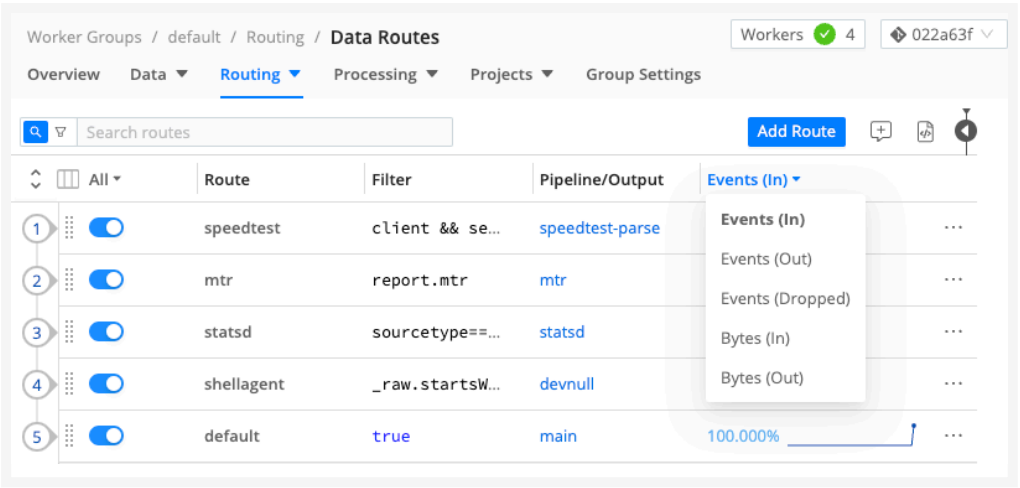
- `true`
- `source=='foo.log' && fieldA=='bar'`

There can be multiple Routes in a Cribl Stream deployment, but each Route can be associated with only **one processing** Pipeline or Pack. (Sources and Destinations can also have their own attached pre-/post-processing Pipeline or Pack.)

Each Route must have a unique name. The **Route Name** field is case-sensitive, so entering the same characters with different capitalization will create a Route with a separate name.

Routes are evaluated in their display order, from the first Route in the list to the last. The configuration in the following screenshot means that events are evaluated first against the Route named **speedtest**, then against

mtr, then against **statsd**, and so on. At the end, the **default** Route serves as a catch-all to explicitly route events that do not match any of the Route filters. We recommend always using a catch-all Route.



Routes and bytes

Use the drop-down menu for the **Events (In)** column to select whether the column displays Events In, Out, or Dropped or Bytes In or Out. The stats shown in this column are for the most recent 15 minutes.

Displayed percentages of Events or Bytes In and Out are by comparison to throughput or filtering across all Routes. For example, if you have four Routes that share an identical filter, each Route's Events In might show approximately 25% of the total inbound traffic across all Routes.

Design data paths to move through as few Routes as possible. This usually means we want to reduce volume of events as early as possible. If most of the events being processed are sent to a Destination by the first Route, this will spare all the other Routes and Pipelines wasting processing cycles and testing against whether filter criteria are met.

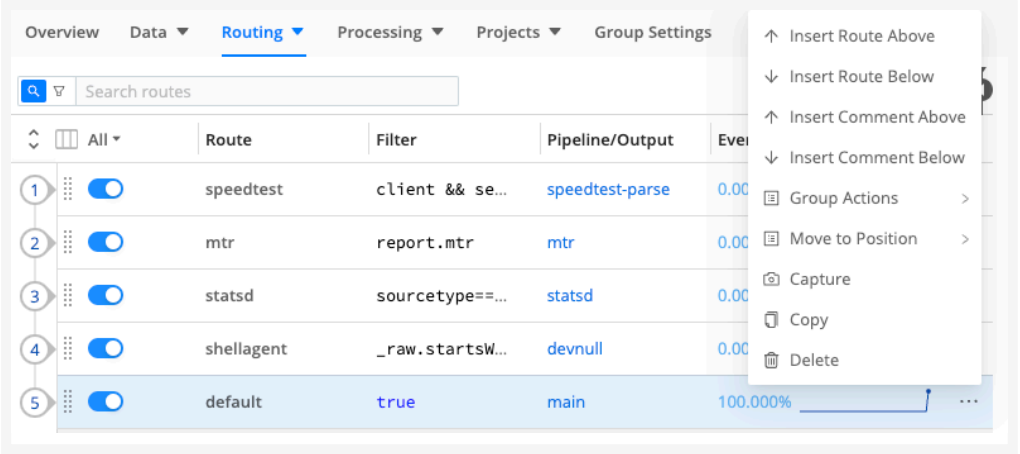
Managing the Routes Page

To apply a Route before another, simply drag it vertically. Toggle Routes or or off inline, as necessary, to facilitate development and debugging.

You can press the] (right-bracket) shortcut key to toggle between the Data Preview pane and the expanded Routes display shown above. (This works when no field has focus.)

Click a Route's Options (•••) menu to display multiple options:

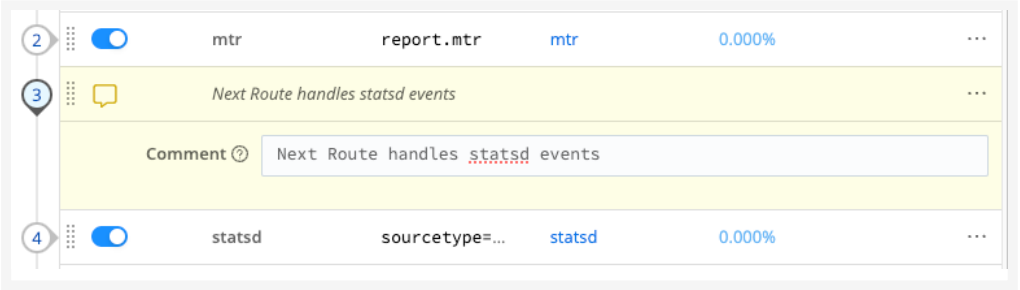
- Insert, group, move, copy, or delete Routes.
- Insert comments above or below Routes.
- Capture sample data through a selected Route.



Route > Options menu

Copying a Route displays a confirmation message and adds a Paste button next to **Add Route**. Pasting creates an exact duplicate of the Route, with a warning indicator to change its duplicate name.

You can use comments to describe Routes' purposes and/or interactions.



Comments make the Routing table self-documenting

Destination

You can configure each Route to include a **Destination** to send events to, after they're processed by the Pipeline or Pack.

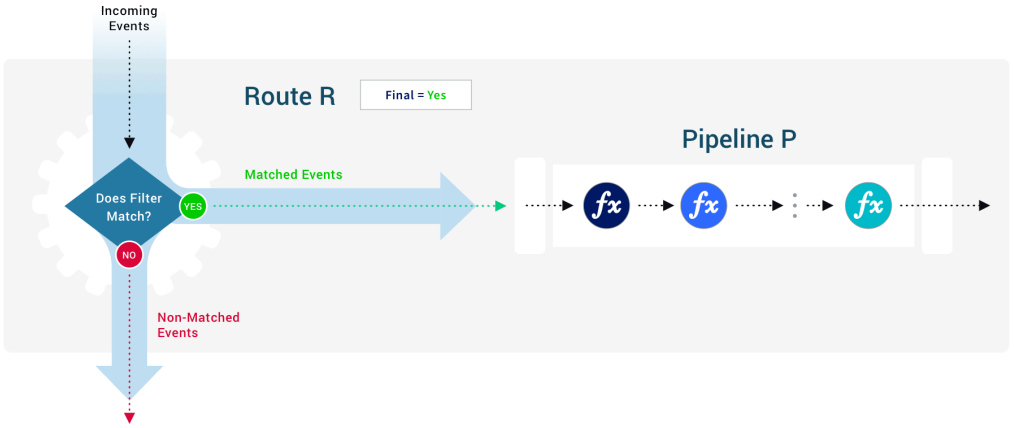
If you toggle **Enable Expression** on, the **Destination** field changes to a **Destination expression** field. Here, you can enter a JavaScript expression that Cribl Stream will evaluate as the name of the Destination. Sample expression format: `` `myDest_${C.logStreamEnv}` ``. (This evaluation happens at Route construction time, not per event.)

Destination expression enabled

The Final Toggle

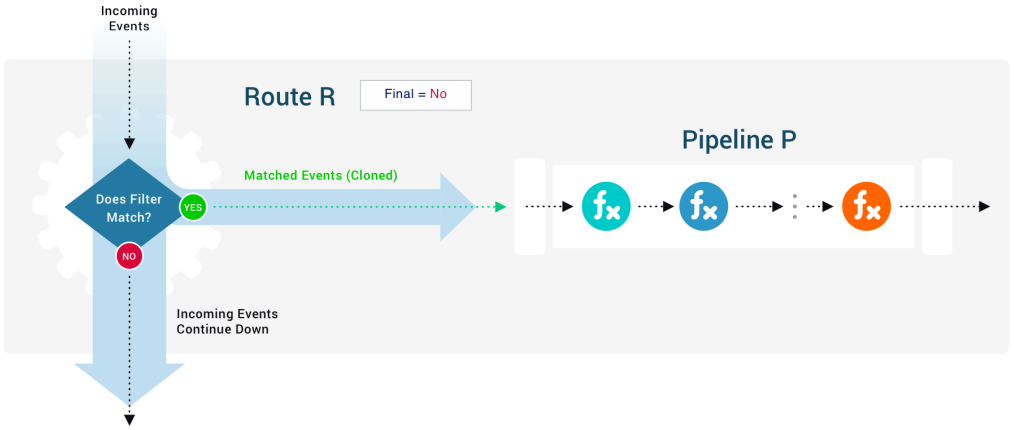
The **Final** toggle in Route settings controls what happens to an event after a Route-Pipeline pair matches it.

When **Final** is toggled on (default), the Route processes matched events and sends them to the appropriate Pipeline or Pack. Matched events **do not** continue to the next Route. Non-matched events **do** continue to the next Route.



Data flow for a Route with Final toggled on

When **Final** is toggled off, the Route still processes matched events and sends them to a Pipeline or Pack. However, all events (matched and non-matched) continue to the next route.



Data flow for a Route with Final toggled off

When you toggle **Final** off, the event sent to the current Route technically becomes a *clone* of the original event. The **Add Clone** button lets you add a fields - names and values - to the cloned events sent to this Route's Pipeline.

Non-final Route: add clone fields

Event Cloning Strategy

Depending on your cloning needs, you might want to follow a **most-specific first** or a **most-general first** processing strategy. The general goal is to minimize the number of filters/Routes an event gets evaluated against. For example:

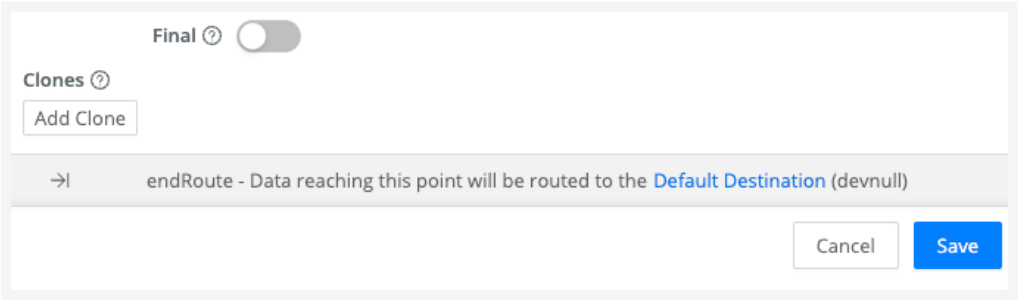
If you don't need to clone any events (that is, **Final** is toggled on everywhere), then it makes sense to start with the broadest expression at the top. This will consume as many events as early as possible.

If you need to clone a narrow set of events, then it might make sense to do that upfront. Then, follow it with a Route that consumes those clones immediately after.

Clone events as late as possible in the Routes. This will minimize the number of Functions acting on data, to the extent possible.

The `endRoute` Bumper

The `endRoute` bumper appears at the bottom of the Routing table. This is a reminder that, if **no** Route in the table has **Final** toggled on, events will continue to Cribl Stream's configured Default Destination.



endRoute warning and link

This is a backstop to ensure data flow. However, if that configured default is also configured as the **Destination** of a Route higher in the table, duplicate events will reach that Destination.

You can correct this either by toggling **Final** on for a Route or by changing the Default Destination. The bumper provides a link to the Default Destination's config, and identifies the currently configured default in parentheses.

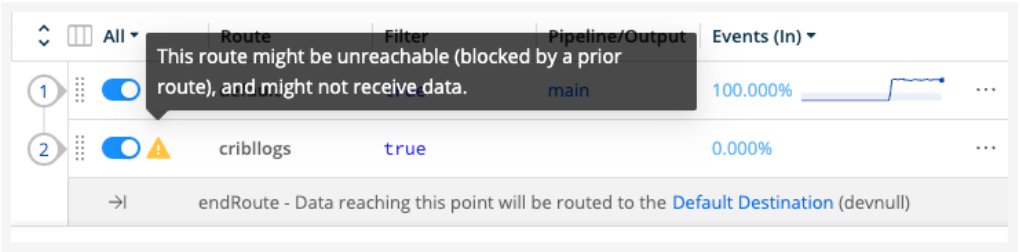
Route Groups

A Route Group is a collection of consecutive Routes that can be moved up and down the Route stack together. Groups help with managing long lists of Routes. They are a UI visualization only: While Routes are in a group, those Routes maintain their global position order.

Route groups work much like Function groups, offering similar UI controls and drag-and-drop options.

Unreachable Routes

Routes display an “unreachable” warning indicator (orange triangle) when data can’t reach them.



Unreachable Route warning, on hover

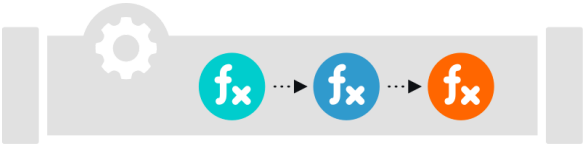
This condition will occur when, with your current configuration, any Route higher in the stack matches **all** three of these conditions:

- Previous Route is enabled.
- Previous Route’s **Final** toggle is on.
- Previous Route’s **Filter** expression evaluates to true, (for example, `true`, `1 === 1`).

Note that the third condition above can be triggered intermittently by a randomizing method like `Math.random()`. This might be included in a previous Route’s own Filter expression, or in a Pipeline Function (such as one configured for random data sampling).

6.4. Pipelines

Data matched by a given Route is delivered to a Pipeline. Pipelines are the heart of Cribl Stream processing. A Pipeline contains a logical sequence of Functions, which process data events and perform various operations. Functions are pieces of JavaScript code that Cribl Stream invokes on each event that passes through a Pipeline. Each Function performs some sort of processing action on data events.



Pipelines contain a logical sequence of Functions

Data flows from a Source into a Pipeline, where each Function processes the events in the order they appear. Once processing is complete, the data flows out of the Pipeline to a Destination. See How Pipelines Work for more information.

As with Routes, the order in which you list the Functions in your Pipeline matters. Cribl Stream evaluates the Functions in a Pipeline in top-down order.

While you are building or editing a Pipeline, you can use the Data Preview pane to preview and test the impact of Functions on your data. The Data Preview gives you confidence that your Pipeline configurations result in the intended output. If your data has a lot of metrics events, you can use the Metrics view to visualize metrics events while building your Pipeline. See Data Preview for more information.

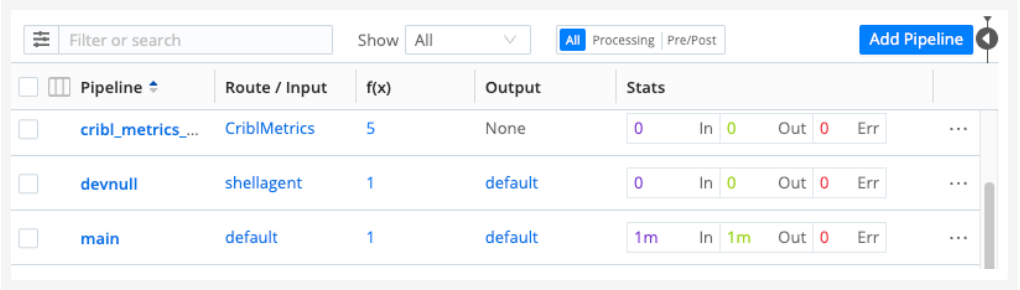
Consider using Cribl Copilot Editor for help building data Pipelines. Instead of building a Pipeline manually, you can describe what you want to do with your data and Copilot Editor will generate a working Pipeline for you. See Create Pipelines Using Cribl Copilot Editor for more information.

Access and View Pipelines

To access Pipelines:

1. In the sidebar, select **Worker Groups**, then select a Worker Group.
2. On the **Worker Groups** submenu, select **Processing**, then **Pipelines**.

On a Single-instance deployment, select **Processing** straight from the **Home** submenu.



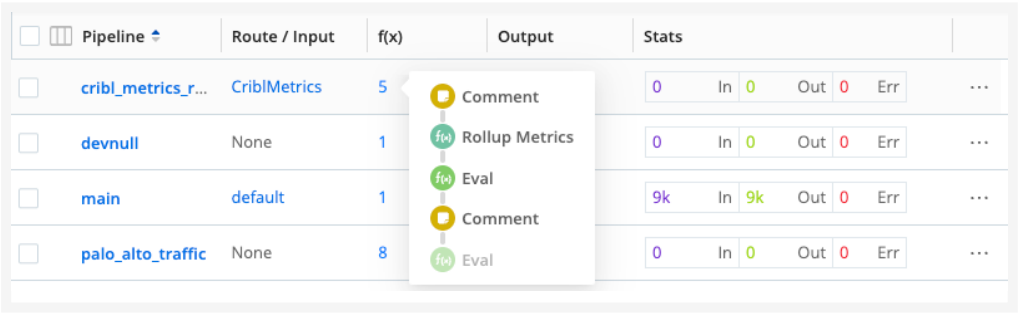
The Pipeline screen shows your current Pipelines and other information

The Pipeline screen shows:

- A table listing your current Pipelines and provides information such as which Routes they are connected to. The **Stats** column of the table shows data for the last 15 minutes.
- The Data Preview pane. When you open a Pipeline, this pane adds a Pipeline Status tab, which you can select to see the Pipeline's events throughput. See Data Preview for more information about using this pane to test and validate your Pipeline as you build it.

You can press the] (right-bracket) shortcut key to toggle between the Preview pane and an expanded Pipelines display. (This shortcut works when no field has focus.)

In the condensed Pipelines display, you can also hover over the **Functions** column for any Pipeline to see a vertical preview of the stack of Functions contained in the Pipeline:



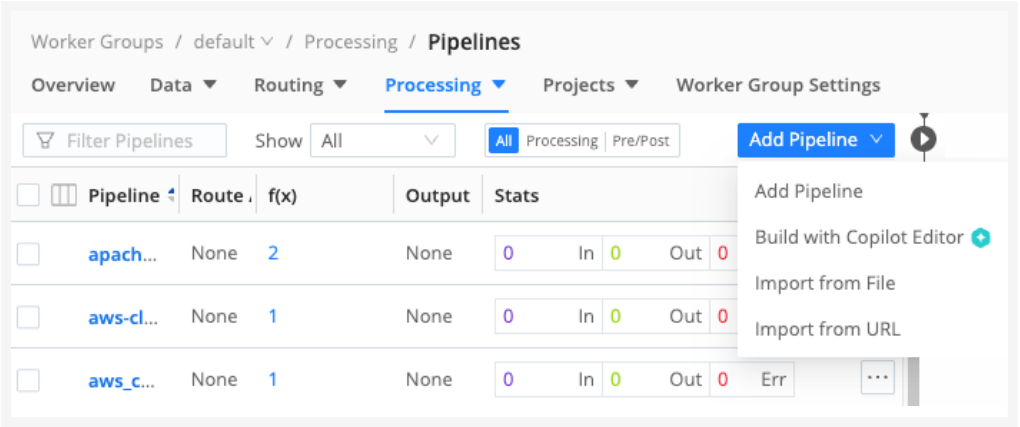
Preview on hovering over the top Pipeline (highlighted in gray)

Add Pipelines

To create a new Pipeline, or to import an existing Pipeline to a different Cribl Stream instance, select **Add Pipeline** at the upper right. The resulting menu offers three options:

- **Add Pipeline**: Configure a new Pipeline from scratch, by adding Functions in Cribl Stream's graphical UI.
- **Add Pipeline with Cribl Copilot Editor**: Use the Cribl Copilot Editor assistant to create a Pipeline using AI. See Create Pipelines Using Cribl Copilot Editor to learn more.
- **Import from File**: Import an existing Pipeline from a `.json` file on your local filesystem.

- **Import from URL**: Import an existing Pipeline from `.json` file at a remote URL. This must be a public URL ending in `.json` because the import option doesn't pass credentials to private URLs. Additionally, you must format the target file as a valid Pipeline configuration.



Create or import a Pipeline

To export a Pipeline, see Advanced Mode (JSON Editor).

To import or export a Pipeline along with broader infrastructure (like Knowledge Objects and/or sample data files), see Packs.

Default Pipelines

Cribl Stream ships with a set of built-in default Pipelines that are created automatically in each Worker Group. These include:

- `devnull`
- `main`
- `passthru`
- `prometheus_metrics`
- `cisco_asa`
- `cisco_estreamer`
- `palo_alto_traffic`
- `wineventlogs`
- `cribl_metrics_rollup`

These default Pipelines are managed by Cribl. Cribl Stream re-creates them from its default configuration when you create new Worker Groups, upgrade, or apply certain configuration bundles. As a result, they cannot be

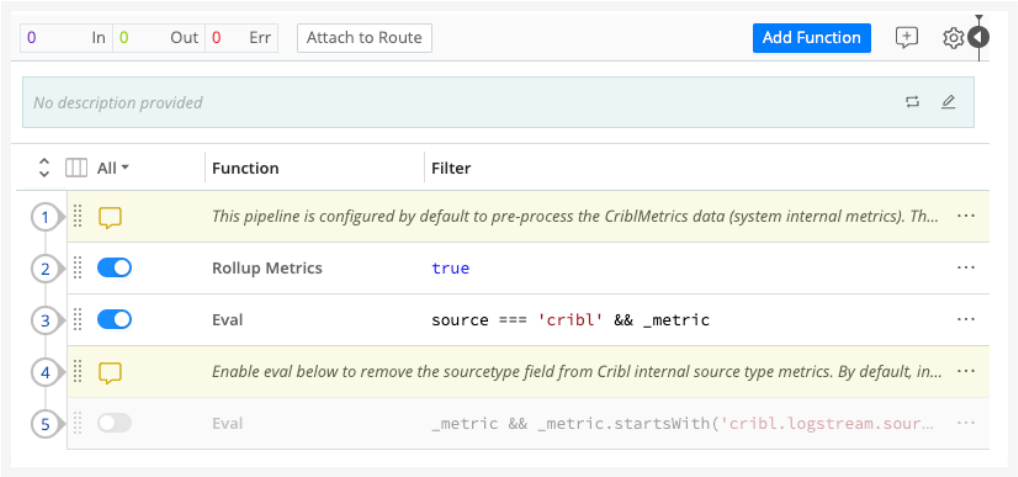
permanently deleted. Removing them in the UI only affects the current configuration copy, and they will reappear the next time defaults are re-applied.

For production, we recommend building your own custom Pipelines or using Pipelines provided by Packs, and simply not attaching the default Pipelines to any Routes or Destinations you don't intend to use.

How Pipelines Work

Events are always delivered to the beginning of a Pipeline through a Route. The **Stats** column shows data for the last 15 minutes.

Within the Pipeline, events are processed by each Function, in order. A Pipeline will always move events in the direction that points outside of the system. This is on purpose, to keep the design simple and avoid potential loops.



Pipeline Functions

Use the **Attach to Route** link at the top to associate a new Pipeline with a Route.

You can streamline a complex Pipeline's display by organizing related Functions into Function groups.

Pipeline Settings

Select the gear button at the top right to open the Pipeline's Settings. Here, you can:

- Use the **Async function timeout (ms)** to set the maximum amount of processing time, in milliseconds, that a Function is allowed to take before it is terminated. This prevents a Function from causing undesirable delays in your Pipeline (for example, a Lookup Function taking too long to process a large lookup file).
- Use the **Tags** field to attach arbitrary labels to the Pipeline. Once attached, you can use these tags to filter/search and group Pipelines.

Pipeline Settings

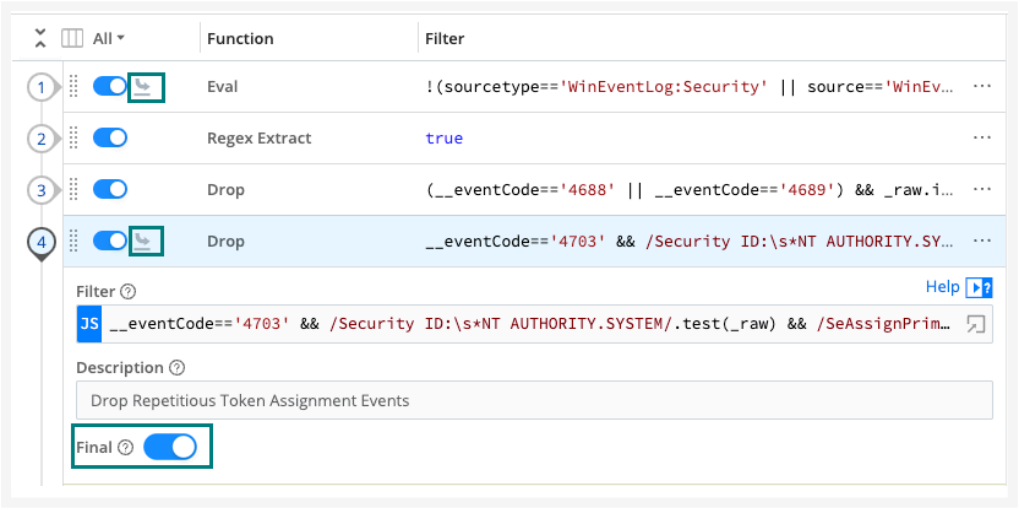
The Final Toggle

The **Final** toggle in Function settings controls what happens to the results of a Function.

When **Final** is toggled off (default), events will be processed by this Function, **and** then passed on to the next Function below.

When **Final** is toggled on, the Function "consumes" the results, meaning they will not pass down to any other Function below.

An icon in Pipeline view indicates that a Function is **Final**.



The Eval and first Drop Functions bearing the Final icons

Advanced Mode (JSON Editor)

Once you've selected the gear button to enter Pipeline Settings, you can select **Manage as JSON** at the upper right to edit the Pipeline's definition in a JSON text editor. In this mode's editor, you can directly edit multiple values. You can also use the **Import** and **Export** buttons here to copy and modify existing Pipeline configurations, as `.json` files.

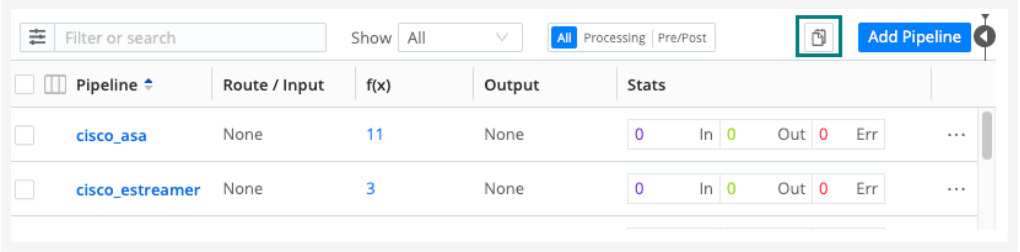
Advanced Pipeline Editing

Select **Edit in GUI** at upper right to return to the graphical Pipeline Settings page; then select **Back to <pipeline-name>** to restore the graphical Pipeline editor.

Pipeline Actions

Select a Pipeline's Actions (...) menu to display options for copying or deleting the Pipeline.

Copying a Pipeline displays a confirmation message and the (highlighted) Paste button shown below.



Paste button for copied Pipeline

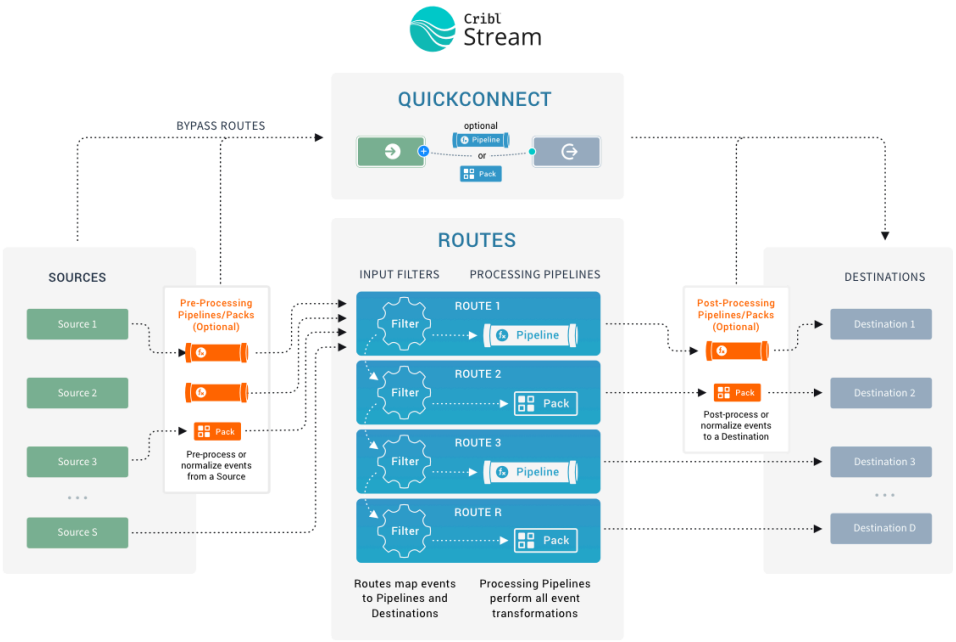
Pasting prompts you to confirm, or change, a modified name for the new Pipeline. The result will be an exact duplicate of the original Pipeline in all but name.

Chain Pipelines

You can use the Chain Function to send the output of a Pipeline to another Pipeline or Pack. There are scope restrictions within Packs, and general guardrails against circular references.

Types of Pipelines

You can apply various Pipeline types at different stages of data flow. All Pipelines have the same basic internal structure (a series of Functions) - the types below differ only in their position in the system.



Pre-processing, processing, and post-processing Pipelines

Pre-Processing Pipelines

These are Pipelines that are attached to a Source to condition (normalize) the events **before** they're delivered to a processing Pipeline. They're optional.

Typical use cases are event formatting, or applying Functions to **all** events of an input. (For example, to extract a `message` field before pushing events to various processing Pipelines.)

You configure these Pipelines just like any other Pipeline, by selecting **Processing** > **Pipelines** from the **Worker Group** submenu. You then attach your configured Pipeline to individual Sources, using the Source's **Pre-Processing** > **Pipeline** drop-down.

Fields extracted using pre-processing Pipelines are made available to Routes.

Processing Pipelines

These are "normal" event processing Pipelines, attached directly to Routes.

Post-Processing Pipelines

These Pipelines are attached to a Destination to normalize the events before they're sent out. A post-processing Pipeline's Functions apply to **all** events exiting to the attached Destination.

Typical use cases are applying Functions that transform or shape events per receiver requirements. (For example, to ensure that a `_time` field exists for all events bound to a Splunk receiver.)

You configure these Pipelines as normal, by selecting **Processing** > **Pipelines** from the **Worker Group** submenu. You then attach your configured Pipeline to individual Destinations, using the Destination's **Post-Processing > Pipeline** drop-down.

You can also use a Destination's **Post-Processing** options to add **System Fields** like `cribl_input`, identifying the Cribl Stream Source that processed the events.

Best Practices for Pipelines

Functions in a Pipeline are equipped with their own filters. Even though filters are not required, we recommend using them as often as possible. See Build Custom Logic to Route and Process Your Data for more information.

As with Routes, the general goal is to minimize extra work that a Function will do. The fewer events a Function has to operate on, the better the overall performance. For example, if a Pipeline has two Functions, **f1** and **f2**, and if **f1** operates on `source 'foo'` and **f2** operates on `source 'bar'`, it might make sense to apply `source=='foo'` versus `source=='bar'` filters on these two Functions, respectively.

Don't overload Pipelines. Do not use the same Pipeline for both pre-processing and post-processing. This makes isolation and troubleshooting extremely difficult.

Extract or parse by desired yield. If you need to extract one or just a few fields, use the Regex Extract Function. If you need to extract all or most of an event's fields, use the Parser Function.

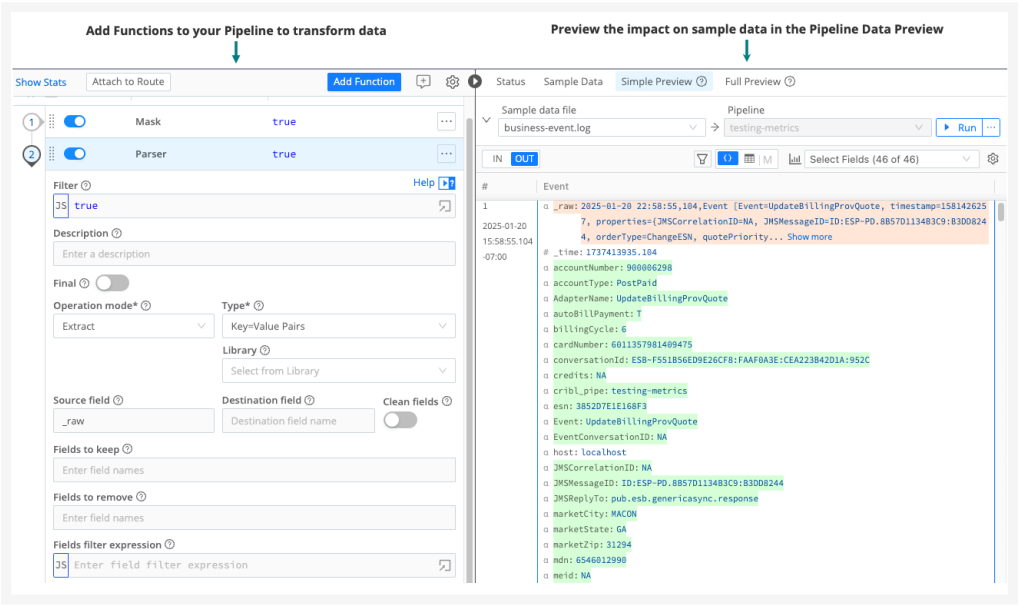
Use comments to preserve legibility. A lot of contextual information can become lost over time as users continue to advance and add Routes and Pipelines to Cribl Stream. A good principle is to keep the design decisions as simple and easy to understand as possible, and to document the assumptions around each Route

6.4.1. Validate Pipeline Logic Using Data Preview

You can use the Data Preview tool to preview, test, and validate the Functions you add to a Pipeline or Route. It helps ensure that your Pipeline and Function configurations produce the expected output, giving you confidence that Cribl Stream processes your data as intended.

The Data Preview tool works by processing a set of sample events and passing them through the Pipeline. Then, it displays the inbound and outbound results in a dedicated pane. Whenever you modify, add, or remove a Function, the output updates instantly to reflect your changes.

For example, suppose you add a Parser Function to your Pipeline to convert raw, unstructured data into structured key-value pairs. After parsing, you may choose to promote these key-value pairs to the field level to enhance data processing and analysis. To verify these changes meet your expectations, you can use the **OUT** tab in the Data Preview pane to review the transformed data.



Add Functions to your Pipeline to transform data, then preview the impact on sample data in Data Preview

In addition to previewing data, the Data Preview offers:

- **Metrics View**: This metrics-first view in the Data Preview pane automatically detects and visualizes metrics in your sample data, organizing them by metric name for a more intuitive experience. Designed for metrics-heavy datasets, this view provides insights into key metrics, dimensions, and time series. It helps you identify high-cardinality data (datasets with a large number of uncommon or unique values), enabling you to make informed decisions about data processing before forwarding it to potentially costly downstream receivers. See Metrics View and Manage Metrics and High Cardinality for more information.
- **Pipeline diagnostics**: Includes status updates and summary throughput statistics to assess the health of the Pipeline and its Functions. See Pipeline Diagnostics and Statistics for more details.

Data preview on the **Simple Preview** tab is limited to 10 MB of data. This cap prevents excessive data processing and safeguards system stability. If the dataset is more than 10 MB when a simple preview runs, Cribl Stream will limit the preview to 10 MB and stop processing data beyond that limit.

The Get Started With Cribl Stream tutorial provides hands-on experience with the Pipeline Data Preview if you would like to learn more about this tool through personal experimentation and exploration.

How to Use the Data Preview Tool

To use the Data Preview tool:

1. Create or import some sample data. See Create and Share Data Samples for information about different methods to generate sample data.
2. Then, use the different Data Preview views to test and validate the changes you make in your Pipeline. See Explore Data and Validate Pipelines with Different Views to learn about the different views you can apply to your data.

Explore Data and Validate Pipelines with Different Views

Once you have created or imported sample data for your Pipeline, you can explore and analyze it using the various views available in the Data Preview pane. These views allow you to preview and test changes in real time, helping ensure that your data processing Functions produce the expected results. By leveraging the Data Preview, you can confidently validate that your Pipeline transformations align with your intended outcomes.

Simple or Full Previews

Select **Simple Preview** or **Full Preview** beside a file name to display its events in the Preview pane:

- **Simple Preview**: View events on either the **IN** or the **OUT** (processed) side of a single Pipeline.
- **Full Preview**: View events on the **OUT** side of either the processing or post-processing Pipeline. Selecting this option expands the Preview pane's upper controls to include an **Exit Point** drop-down, where you make this choice.

Before and After Views

The Data Preview pane offers two time-based display options for previewing sample data:

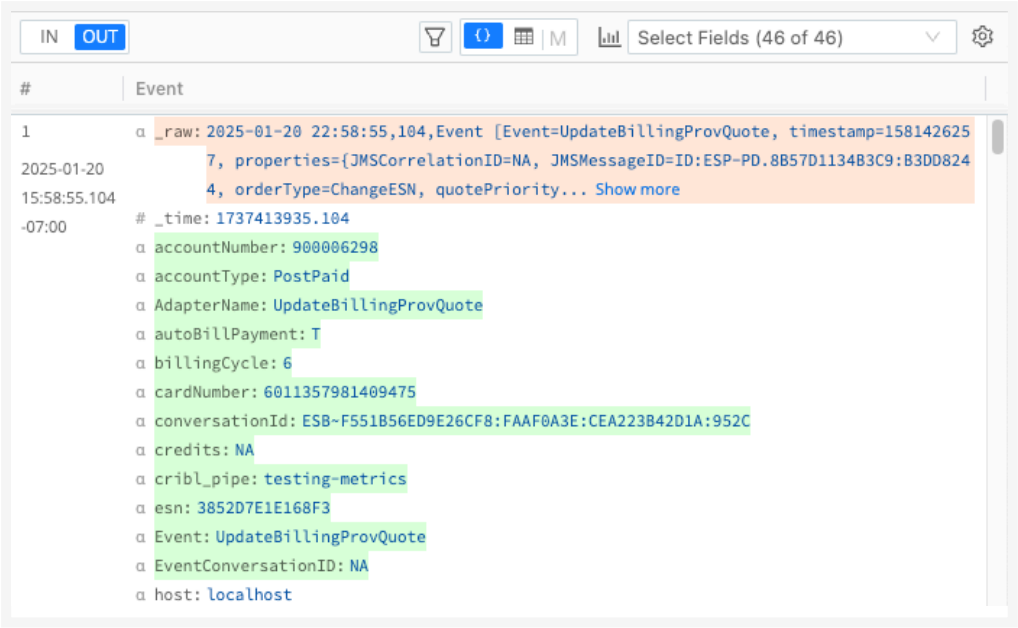
- The **IN** tab displays samples as they appear on the way *in* to the Pipeline, before data processing.
- The **OUT** tab displays the same information, but shows the sample data as it will appear on way *out* of the Pipeline after the data processing. Use the **Select Fields** drop-down menu to control which fields the **OUT** tab displays.

In most cases, you will typically use the **OUT** tab to preview, test, and validate your Pipeline.

Event and Table Views

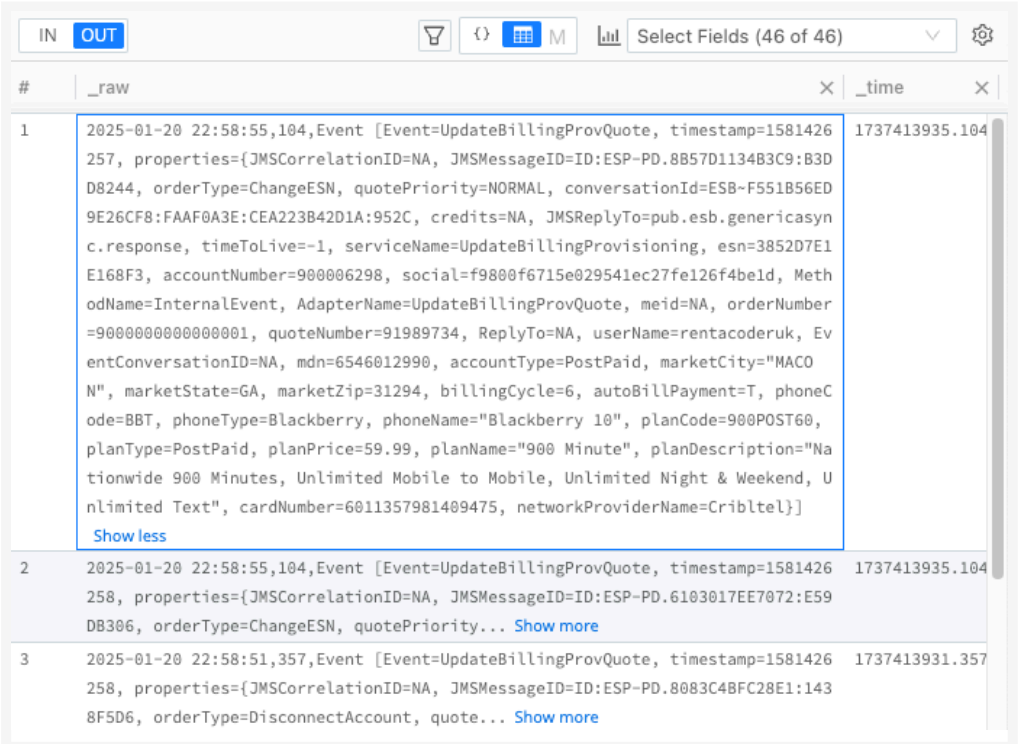
The Data Preview toolbar has several available views. Each format can be useful, depending on the type of data you are previewing.

The Event View displays data in JSON format for both the **IN** and **OUT** tabs.



Example Event View

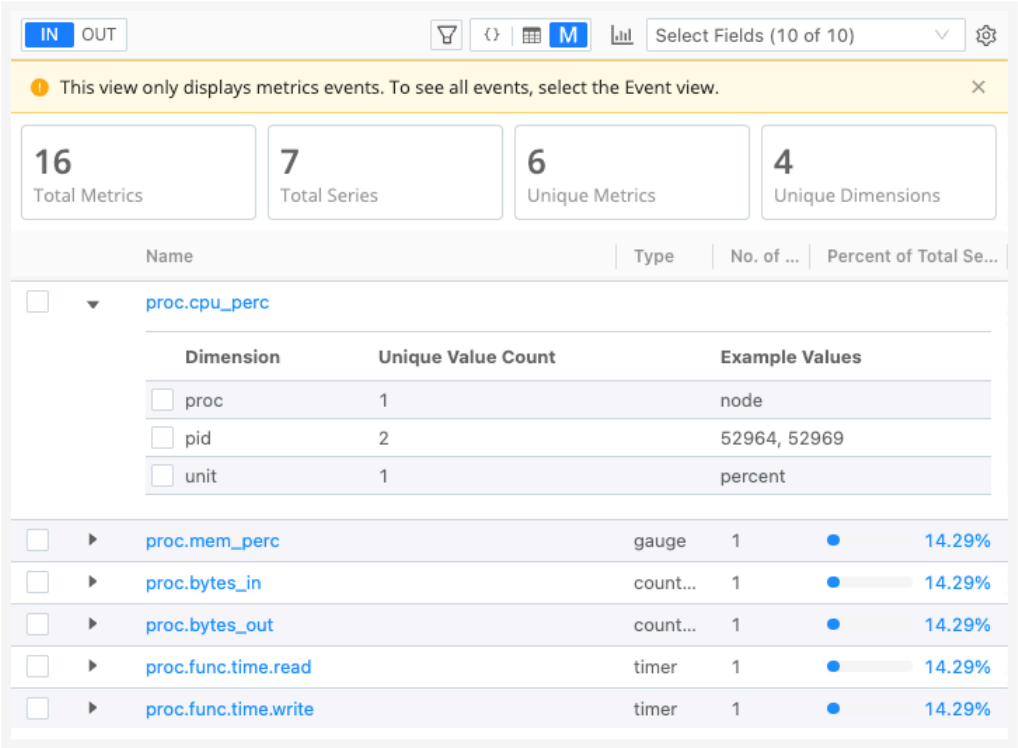
The Table View in displays data as a table for both the **IN** and **OUT** tabs.



Example Table View

Metrics View

If your data sample contains metric events, the Data Preview automatically switches to the Metrics view. It intelligently detects the underlying data structure and switches to this view when needed to display relevant insights. This metrics-first view allows you to visualize metric events as you build your Pipeline. It allows you to analyze metrics, dimensions, time series, the number of unique values, and the percentage each metric contributes to the total time series. See Manage Metrics and High Cardinality for more information about using Metrics View.



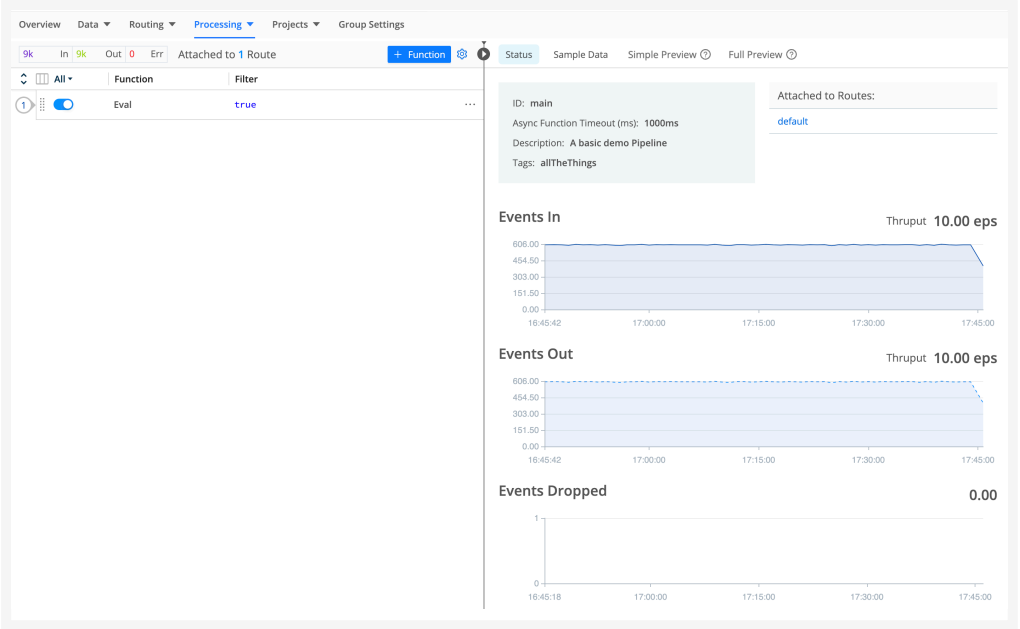
Example Metrics View

Pipeline Diagnostics and Statistics

This section explains the available tools in the Data Preview that you can use to get information about your Pipeline and Functions.

Check Pipeline Status

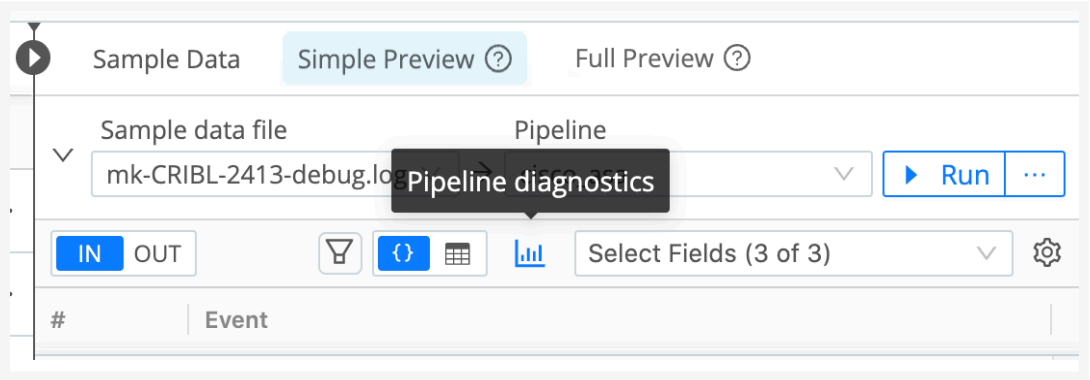
From the Pipelines page, select any Pipeline to display a **Status** tab in the Data Preview pane. Select this tab to show graphs of the Pipeline **Events In**, **Events Out**, and **Events Dropped** over time, along with summary throughput statistics (in events per second).



Pipeline status tab

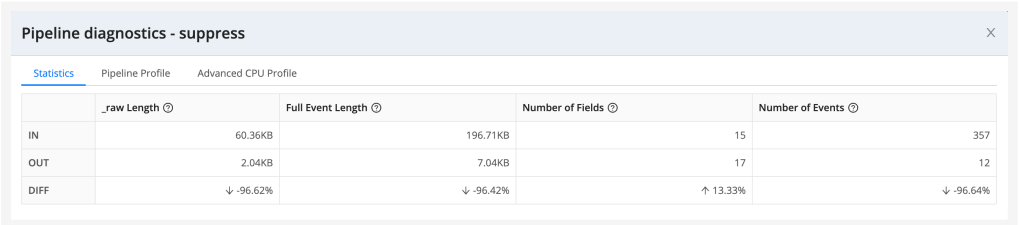
Pipeline Profiling

At top of the Preview pane, you can select a sample data file and Pipeline, then select the Pipeline diagnostics (bar-graph) button to access statistics on the selected Pipeline's performance.



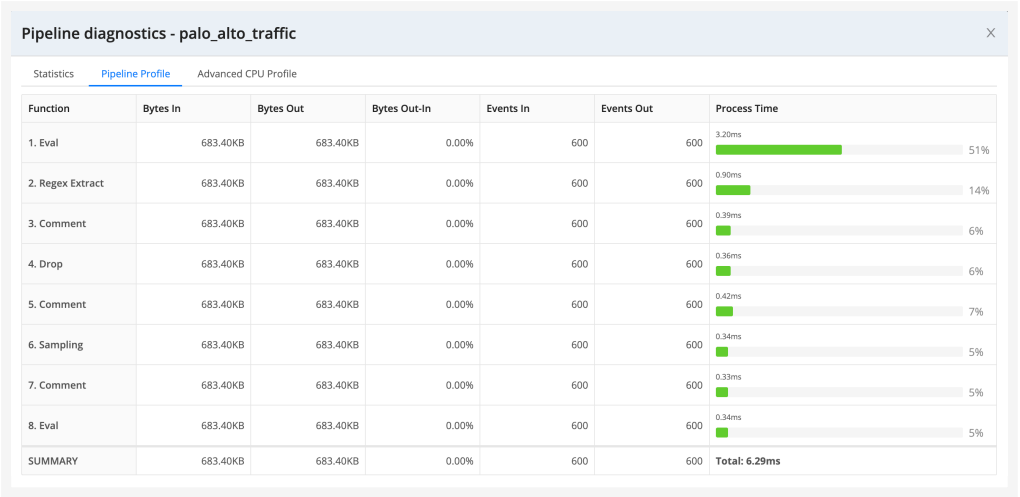
Pipeline diagnostics

The **Statistics** tab in the resulting modal displays the processing impact of the Pipeline on field length and counts.



Pipeline diagnostics > Statistics tab

The **Pipeline Profile** tab, available from the **Simple Preview** tab, helps you debug your Pipeline by showing the contribution of individual Functions to data volume, events count, and the Pipeline processing time. Every Function has a minimum processing time, including Functions that make no changes to event data, such as the Comment Function.

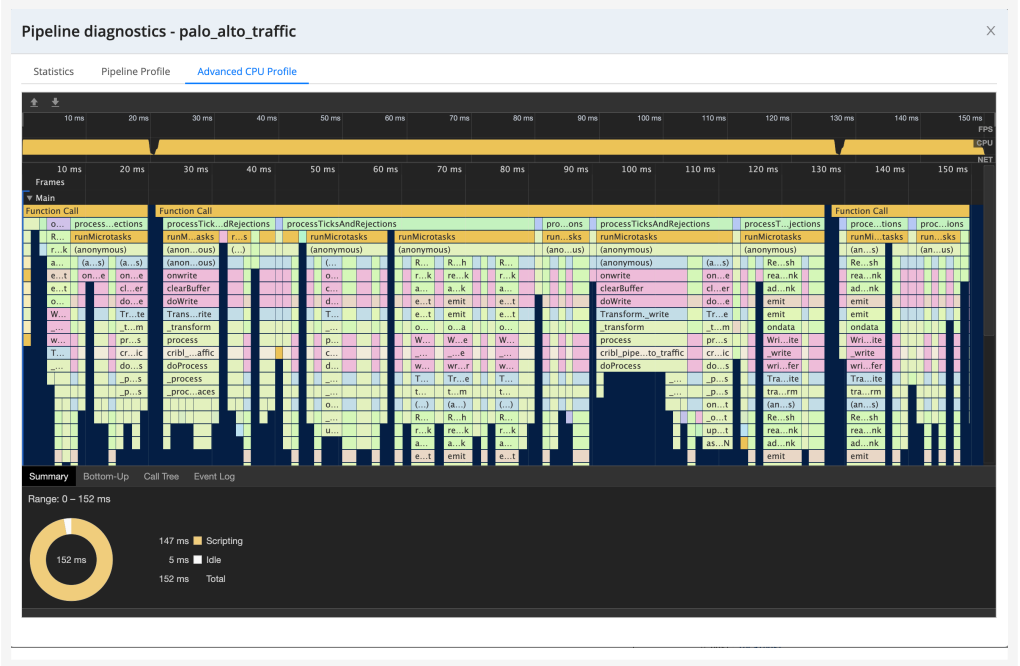


Function	Bytes In	Bytes Out	Bytes Out-In	Events In	Events Out	Process Time
1. Eval	683.40KB	683.40KB	0.00%	600	600	3.20ms 51%
2. Regex Extract	683.40KB	683.40KB	0.00%	600	600	0.90ms 14%
3. Comment	683.40KB	683.40KB	0.00%	600	600	0.39ms 6%
4. Drop	683.40KB	683.40KB	0.00%	600	600	0.36ms 6%
5. Comment	683.40KB	683.40KB	0.00%	600	600	0.42ms 7%
6. Sampling	683.40KB	683.40KB	0.00%	600	600	0.34ms 5%
7. Comment	683.40KB	683.40KB	0.00%	600	600	0.33ms 5%
8. Eval	683.40KB	683.40KB	0.00%	600	600	0.34ms 5%
SUMMARY	683.40KB	683.40KB	0.00%	600	600	**Total: 6.29ms**

Pipeline diagnostics > Pipeline Profile tab

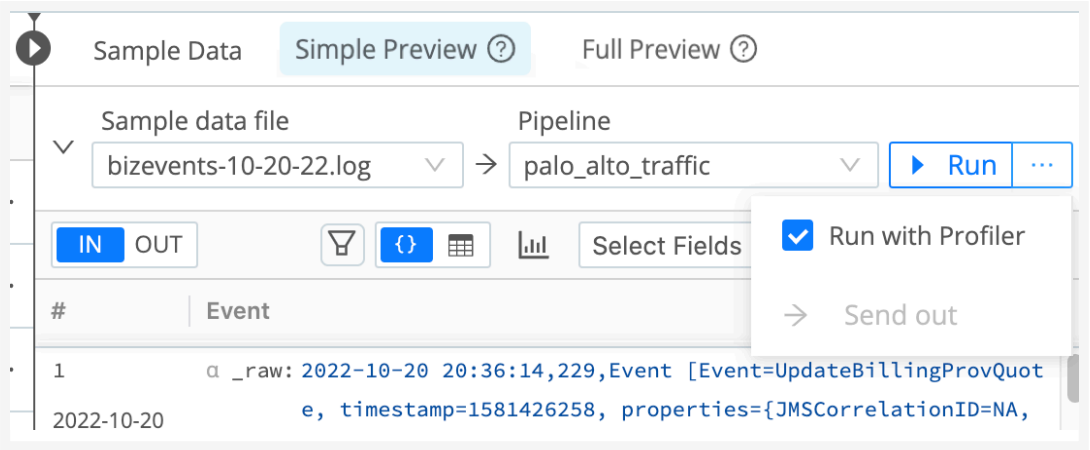
When you profile a Chain Function, this tab condenses all of the chained Pipeline's processing into a single **Chain** row. To see individual statistics on the aggregated Functions, profile the chained Pipeline separately.

The **Advanced CPU Profile** tab, available from the **Simple Preview** tab, displays richer details on individual Function stacks.



Pipeline diagnostics > Advanced CPU Profile tab

You can disable the profiling features to regain their own (minimal) resources. To disable profiling features, select the drop-down list beside the **Run** button and clear the **Run with Profiler** check box.



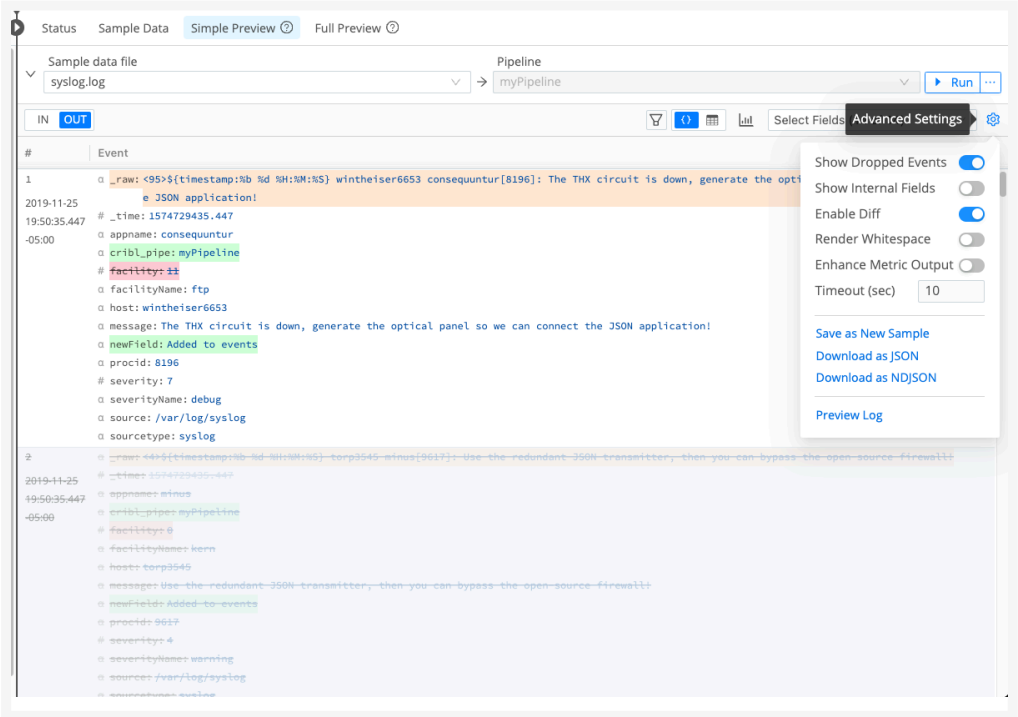
Disabling Pipeline profiling

Advanced Settings

The **Advanced Settings** menu ⚙ includes **Show Dropped Events**, **Show Internal Fields**, and **Enable Diff** toggles, which control how the **OUT** tab displays changes to event data.

If you enable the **Show Dropped Events** and **Enable Diff** toggles, the **OUT** tab displays event data as follows:

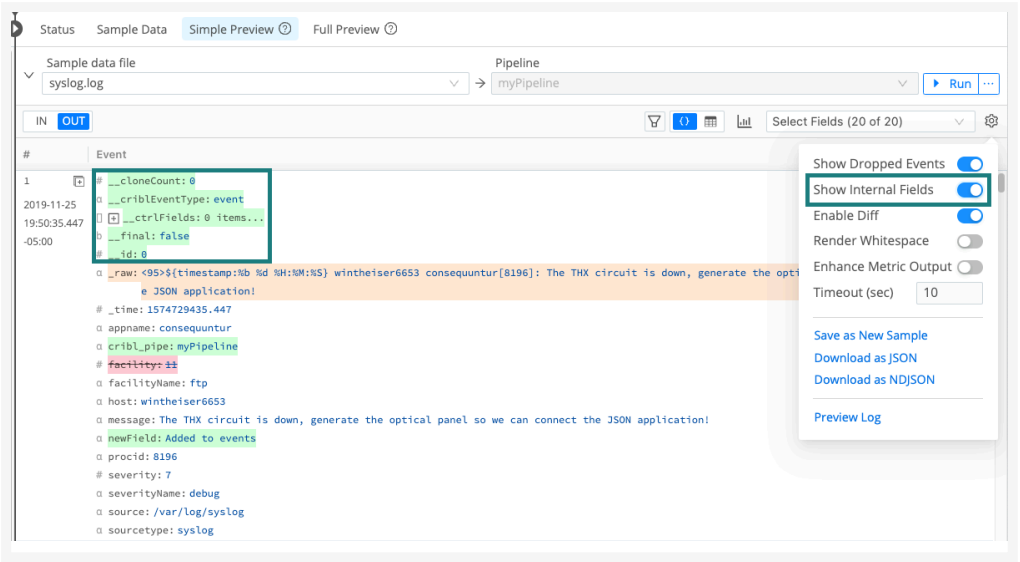
- Modified fields are highlighted in amber.
- Added fields are highlighted in green.
- Deleted fields are highlighted in red and displayed in strikethrough text.
- Dropped events are displayed in strikethrough text.



Highlighted fields in Pipeline output

The rest of this section describes the additional fields available in **Advanced Settings**.

Show Internal Fields: Display fields that Cribl Stream adds to events, as well as Source-specific fields that Cribl Stream forwards from upstream senders.



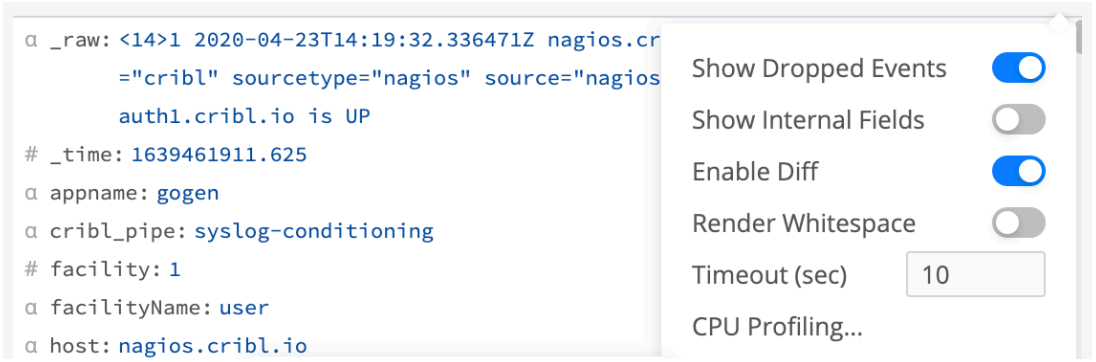
Internal fields highlighted in Pipeline output

Render Whitespace: This toggles between displaying carriage returns, newlines, tabs, and spaces as white space, versus as (respectively) the symbols , , →, and ·.

Enhance Metric Output: Controls how the **OUT** tab presents the **Metric name expression** resulting from the Publish Metrics Function. When you disable this setting (default), Cribl Stream represents the **Metric name expression** as literal expression string. Enable the setting to see the expression evaluated in the **OUT** tab.

Timeout (Sec): If large sample files time out before they fully load, you can adjust this setting to increase the default value of `10` seconds. Cribl Stream interprets a blank field as the minimum allowed timeout value of `1` second.

Save as New Sample: Enables you to save your captured data to a file, using either the **Download as JSON** or the **Download as NDJSON** (Newline-Delimited JSON) option.


α _raw: <14>1 2020-04-23T14:19:32.336471Z nagios.cr
="cribl" sourcetype="nagios" source="nagios
auth1.cribl.io is UP
_time: 1639461911.625
α appname: gogen
α cribl_pipe: syslog-conditioning
facility: 1
α facilityName: user
α host: nagios.cribl.io
Show Dropped Events
Show Internal Fields
Enable Diff
Render Whitespace
Timeout (sec)
10
CPU Profiling...

6.5. Manage Metrics And High Cardinality

Cribl Stream provides a streamlined workflow for managing metrics and metric events, especially to reduce high-cardinality data (datasets with a large number of uncommon or unique values).

As you build your Pipeline, you can:

- **Preview and analyze your metrics in Data Preview:** When you add sample data in your Pipeline, the Data Preview pane automatically switches to Metrics View, a metrics-first view designed specifically for viewing and analyzing metrics data. It provides summary statistics and visualizations to help you identify high-cardinality data and similar inefficiencies. This enables you to make informed decisions about data processing before forwarding it to downstream receivers, which can potentially reduce storage costs and improve query performance.
- **Automatically build a Pipeline to process metrics:** As you inspect metrics in Metrics View, you can select the specific metrics or dimensions you want to aggregate or drop. The interface automatically adds Functions to your Pipeline with an initial configuration based on your selection.

This workflow covers a variety of use cases for managing metrics:

- **Reduce high-cardinality data:** Consolidate, aggregate, or drop metrics or dimensions with a large number of unique or uncommon values to reduce time series, simplify data, and improve query performance.
- **Optimize metrics:** Group and segment similar metrics into fewer categories to reduce complexity. Create time-based rollups by aggregating high-frequency metrics into larger intervals (such as hourly or daily). Combine related or duplicate metrics together to create a unified metric.
- **Normalize metrics:** Aggregate metrics collected from different systems into a single, unified view for easier analysis and consistent reporting.

Overview of Metrics Management Workflow

The workflow for using Cribl Stream to optimize metrics has three phases:

1. Add metrics to your Pipeline: Before you start analyzing and managing metrics data, you first need to get metrics data flowing into Cribl Stream. Then, you need to capture or import some sample data to validate your Pipeline as you build it.
2. Analyze metrics in Data Preview: With sample metrics data open in Metrics View in Data Preview, you can analyze the data to find metrics and dimensions that are possibly contributing to issues with high cardinality or large data volumes.
3. Manage metrics: Use the interface to drop and aggregate metrics or drop dimensions. Cribl Stream automates most of the configuration process. Select the metrics you want to aggregate or drop, and Cribl

Stream will automatically add Functions to your Pipeline with an initial configuration based on your selection.

The following sections explain these three phases in more detail.

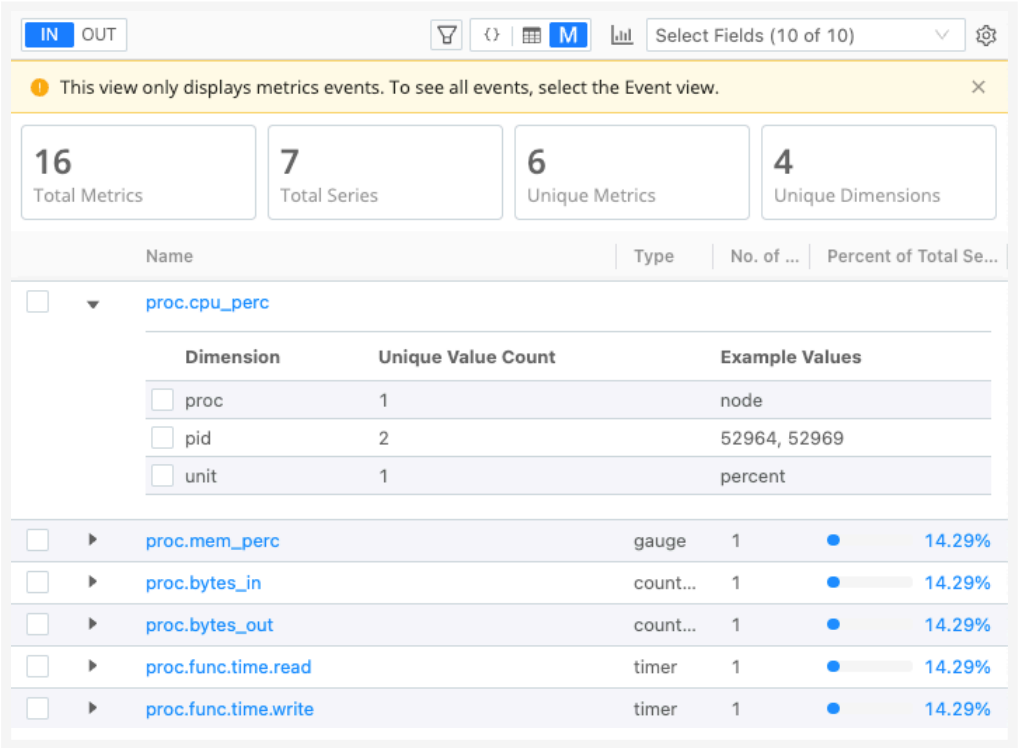
Add Metrics to Your Pipeline

Before you start analyzing and managing metrics data in Cribl Stream, you first need to get metrics data flowing:

1. Add a Source and Destination that can send and receive metrics data. See Integrations for instructions.
2. Add a new Pipeline or open an existing Pipeline. See Pipelines for instructions.
3. Add sample data or capture sample data that contains metrics. See Data Preview: Add Sample Data for instructions.

> **How Data Preview Works**
>
> Pipelines show a list of Functions and their configurations on the left and the Data Preview on the right. The Pipeline processes a set of sample metrics and passes them through Functions in the Pipeline. Then, the Pipeline displays the inbound and outbound results in the Data Preview pane. Whenever you modify, add, or remove a Function, the output in the Data Preview pane updates instantly to reflect your changes. See Data Preview for more information.

4. Because your data sample contains metric events, the Data Preview automatically switches to the Metrics view. It intelligently detects the underlying data structure and switches to this view to display relevant insights.



If your metrics data doesn't show up in Metrics View, it may not be formatted as a supported metric. See Supported Metric Formats and Types for more information.

Analyze Metrics in Data Preview

With sample metrics data open in Metrics View in the Data Preview, you can analyze the data for insights about your metrics. This section explains a variety of strategies for using the Data Preview to find metrics and dimensions that are possibly contributing to downstream storage or processing costs. It also suggests possible solutions and how to implement those solutions in your Pipeline.

Increased downstream storage and processing costs typically have a few common causes:

- **High-cardinality dimensions:** Each combination of dimension values creates a unique time series. If your metrics data has a high number of dimensions with a high number of possible unique values, that means your data has a problem with high cardinality. It has too many possible unique combinations and values (time series). If a downstream vendor charges you by the number of time series in your data, your costs increase with each unique combination of dimensions and dimension values.
- **High-frequency metrics:** If an upstream sender is capturing data points too frequently, you can end up producing a high volume of data. If a downstream vendor charges you by the amount of data you send it, high volumes of metrics data can increase costs or slow down data querying.
- **Low-value metrics:** Not all metrics you produce might be meaningful or valuable to you. Dropping unnecessary metrics or dimensions can help reduce the volume of metrics data so that you only consume what you truly need downstream.

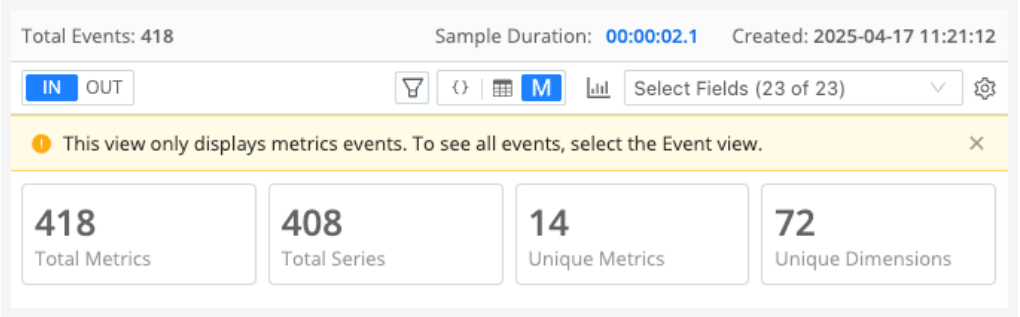
You can use the Metrics View to help detect these common problems in your metrics data.

Get an Overview of Your Metrics

The Data Preview has an **IN** and **OUT** tab so that you compare your incoming data with the changes to the outgoing data. It helps to test and validate the Functions you add to your Pipeline to verify that the data changes applied by the Pipeline match your expectations.

When you first look at the Metrics View, the charts at the top provide high-level summary statistics to help you understand the structure of your metrics, including the total metrics, total time series, the number of unique metrics, and the number of unique dimensions. See Metrics View Reference for detailed descriptions of the Metric View charts and elements.

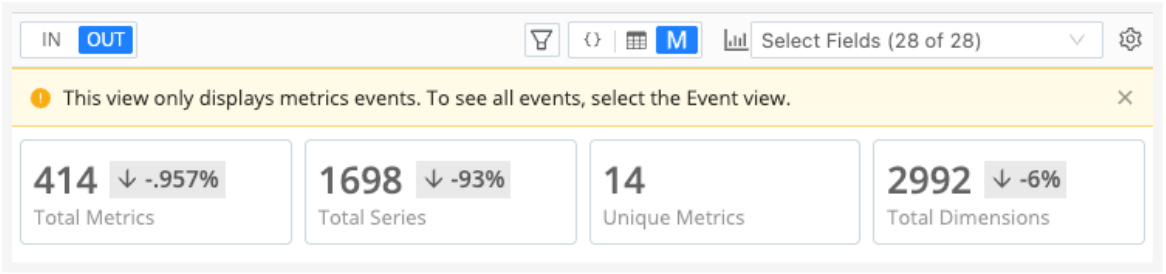
This screenshot shows an example of the summary statistics:



Example of inbound summary statistics in Metrics View

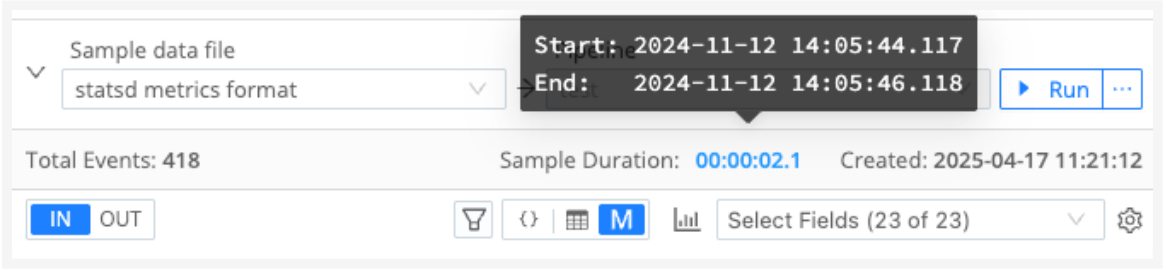
These high-level summaries are useful for before-and-after comparisons of your metrics. After you add Functions that drop or aggregate metrics or dimensions, these summaries display the differences.

This screenshot shows the same summary statistics on the **OUT** tab after applying metrics optimizations in the Pipeline:



Example of outbound summary statistics in Metrics View

To help you evaluate the scope and accuracy of your sample metrics data, the Metrics View in Data Preview also provides a header bar that summarizes key details about the sample. This header bar includes details about the total events, creation date, and the duration of the sampled data. Hovering over the duration reveals the sample's start and end times.



Hover over the Sample Duration to view start and end times

Inspect Time Series

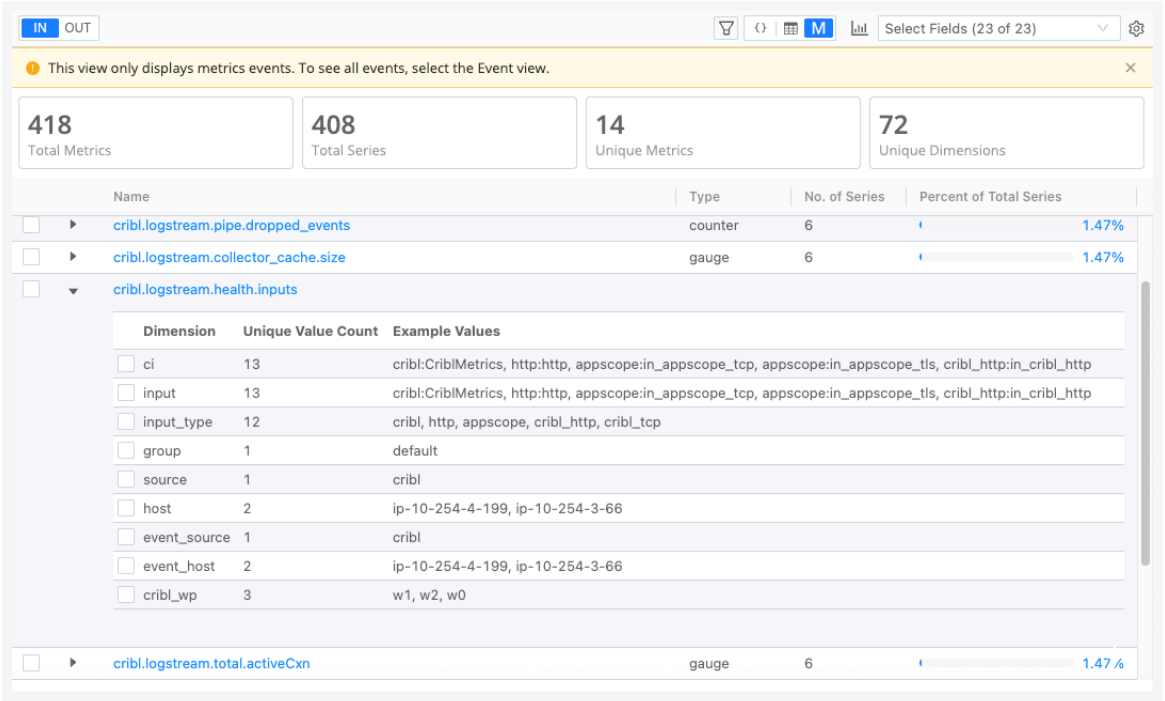
When you first look at the Metrics View, it displays a table listing each metric by name. In that initial view, pay attention to some of these key statistics and indicators for time series:

- **No. of Series:** This column lists the number of time series that this metric and its dimensions creates, calculated as the unique combination of a metric and a dimension pair. If a metric seems to have a high

number listed in this column, that could mean it is contributing to high cardinality.

- **Percent of Total Series**: This column visually represents the proportion of time series represented by this metric, shown as a percentage of the total series. If a metric contributes a higher percentage to the overall total of time series, that could mean it is contributing to high cardinality.

For example, the following screenshot shows a list of metric names. In the list, the metric named `cribl.logstream.health.inputs` stands out because it has a high number of time series (24336) compared to the other metrics. It also represents a significant percentage of total series (97.27%). This metric is likely causing a high-cardinality problem.



Example of a high-cardinality metric in Metrics View

Inspect Dimension Details

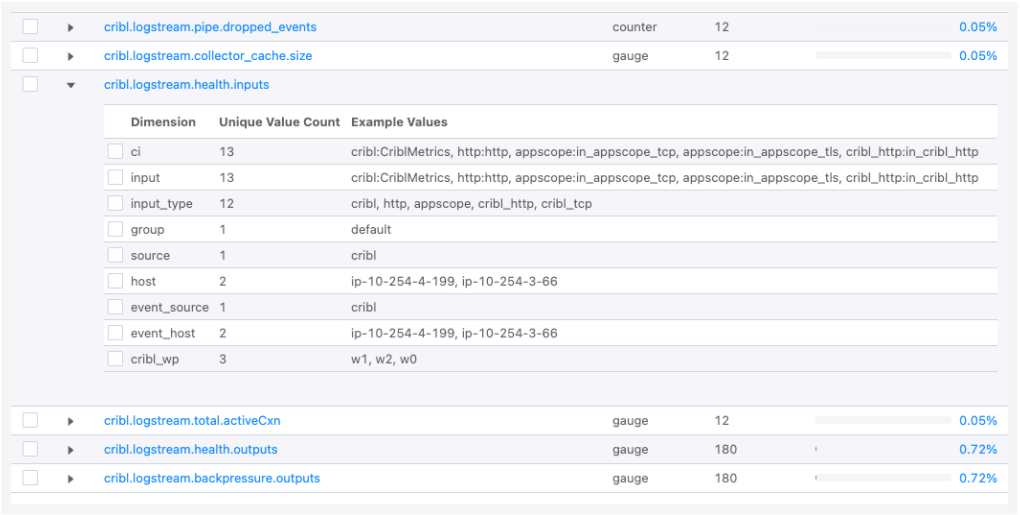
When you want to explore a metric in more detail, you can select the arrow next to the metric name to view its dimensions and the unique values for those dimensions. As you look at the dimension details, examine a few things:

- Are there dimensions with a high number of unique values?
- Do any of dimensions seem to be redundant or share duplicate values?
- As you compare the dimensions, do any seem similar enough that they could be combined together into a broader category or segment?
- Do any of the dimensions seem unnecessary or provide data that isn't valuable for your purposes?

If you answer "yes" to any of these questions, those dimension might be a good candidate for taking action on them in your Pipeline.

To continue the previous example, in the detailed dimensions view for the metric named `cribl.logstream.health.inputs`, several dimensions appear to contain duplicate values:

- `ci` and `input`
- `host` and `event_host`
- `source` and `event_source`



Example of high-cardinality dimensions in Metrics View

Identify Possible Solutions

When you identify metrics or dimensions that contribute to high cardinality and/or high volume, or which don't produce a lot of value for your purposes, you can:

- Drop Metrics: Drop unnecessary metrics to reduce data volume.
- Aggregate Metrics: Aggregate metrics to retain long-term trends while reducing the volume of detailed data, such as keeping averages or summaries only.
- Drop Dimensions: Drop unnecessary or redundant dimensions to minimize the number of unique dimension combinations and the number of time series.
- Aggregate dimensions, such as grouping them into broader categories or rolling them up into higher time frequencies.

Fortunately, Cribl Stream can build these operations into your Pipeline directly from the Metrics View interface, as explained in the next section.

Manage Metrics

Cribl Stream automates most of the configuration process for dropping and aggregating metrics. From the Data Preview, you can select the metric you want to aggregate, and Cribl Stream will automatically add Functions to your Pipeline with an initial configuration based on your selection.

Follow these best practices to troubleshoot common issues and ensure successful configuration when using the Metrics View interface:

- **Troubleshoot metrics data not appearing in Metrics View:** If your metrics data doesn't show up in Metrics View, your date may not be in a supported metric format. See Supported Metric Formats and Types for more information.
- **Find missing check boxes in Data Preview:** If check boxes don't appear next to your metrics data, you may not have added or opened a Pipeline yet. You must add or open a Pipeline to see check boxes.
- **Confirm configuration settings:** Always verify that the configuration settings are correct for your use case before saving.
- **Add and manage multiple Functions:** The interface adds a new Function every time you select metrics to drop or aggregate. Over time, this may lead to a large number of Functions in your Pipeline. To consolidate, you can delete the Functions, then re-add the metrics using the interface. You can also manually combine Functions together by copying and pasting configurations.

Drop Metrics

The Metrics View interface automatically adds a Drop Function to your Pipeline. It pre-configures the **Filter** field with the names of the metrics you select in Data Preview.

To drop metrics from inside the Data Preview:

1. Select the check box next to the names of one or more metrics you want to drop.
2. In the tray that appears under the metrics, select **Drop Metrics**.
3. Check the configuration settings for the Drop Function that now appear in your Pipeline. Optionally, add a **Description** or adjust the configuration settings as needed and select **Save**.
4. Test that you configured the Function correctly by comparing sample incoming data with outgoing data in the Data Preview pane and Pipeline Diagnostics tool. See Data Preview and Metrics View for more information.

Aggregate Metrics

The Metrics View interface automatically adds an Aggregate Metrics Function to your Pipeline. It pre-configures the configuration settings for the metrics you select in Data Preview.

To aggregate metrics from inside the Data Preview:

1. Select the check box next to the names of one or more metrics you want to drop.
2. In the tray that appears under the metrics, select **Aggregate Metrics**.
3. Check the configuration settings for the Aggregate Metrics Function that now appear in your Pipeline and select **Save**. Settings to check or adjust:

- **Description:** Review the generated description and adjust it to better reflect your use case.
- **Final:** Toggle on if this Function is the last Function in your Pipeline.
- **Time window:** Ensure the default duration of the tumbling window for aggregating events meets your needs.
- **Aggregates:** Confirm that the generated aggregation functions match your use case.
- **Group by dimensions:** Check the list of dimensions to group by for accuracy.
- **Output Settings:** Expand this section to reveal additional settings:
  - **Output prefix**: If needed, use a prefix to modify the metric name, making it easier to identify and search for aggregated metrics. A prefix helps distinguish aggregated metrics from non-aggregated ones, allowing you to route or search for them more easily without increasing cardinality.

Check the Aggregate Metrics Function for more information about these settings and additional information about how this Function works.

4. Test that you configured the Function correctly by comparing sample incoming data with outgoing data in the Data Preview pane and Pipeline Diagnostics tool. See Data Preview and Metrics View for more information.

Drop Dimensions

The Metrics View interface automatically adds a Drop Dimensions Function to your Pipeline. It pre-configures the **Filter** field with the names of the dimensions you select in Data Preview. It also adds an initial description to the **Description** field based on your select.

To drop dimensions from inside the Data Preview:

1. Expand a metric to view its dimensions.
2. Select the check box next to the names of one or more dimensions you want to drop.
3. In the tray that appears under the metrics, select **Drop Dimensions**.
4. Check the configuration settings for the Drop Dimensions Function that now appear in your Pipeline. Optionally, edit the **Description** or adjust the configuration settings as needed and select **Save**.
5. Test that you configured the Function correctly by comparing sample incoming data with outgoing data in the Data Preview pane and Pipeline Diagnostics tool. See Data Preview and Metrics View for more information.

Supported Metric Formats and Types

Cribl Stream supports the following metric formats:

- Open Telemetry (OTLP)
- Prometheus
- StatsD

For Open Telemetry metrics, Cribl Stream does not ingest batches. To ingest data from batches, you must enable the **Extract Metrics** setting on your Open Telemetry Source. See OpenTelemetry (OTel) Source for more information.

When Cribl Stream ingests Open Telemetry metrics, it explodes the batches into individual metric events for processing. Cribl Stream can then create new batches from the processed metric data before sending to a downstream receiver.

Metrics View Reference

The Metrics view provides the following high-level summary statistics to help you understand the structure of your metrics event data:

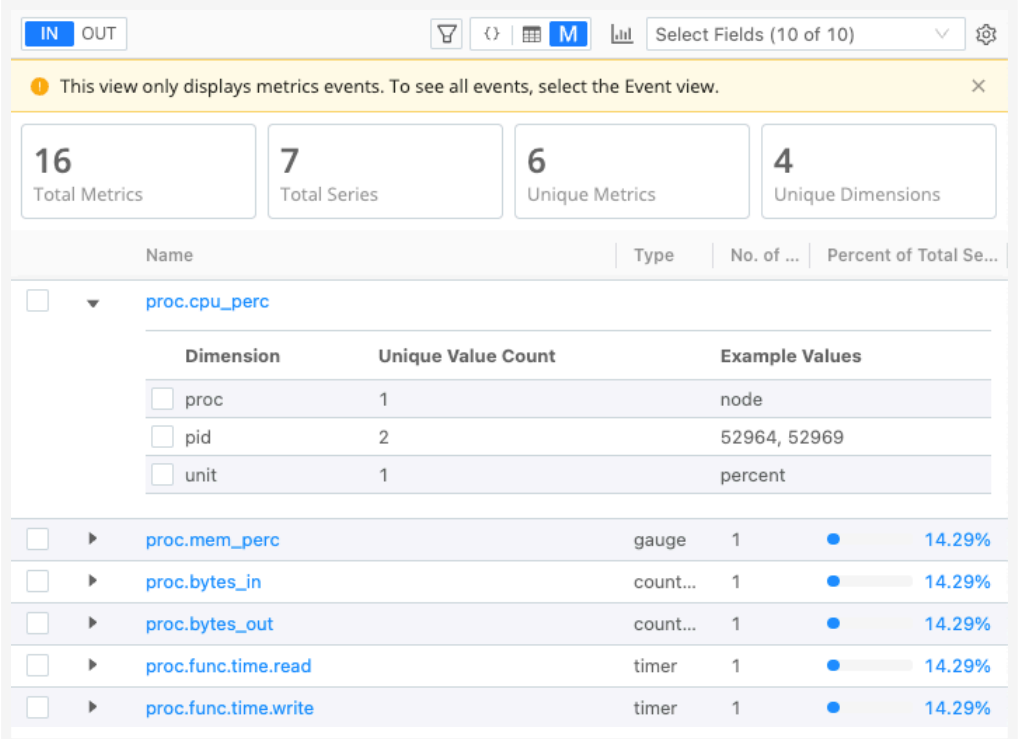
- **Total Metrics**: The total number of metrics in the sample data. If a metric event contains multiple metrics, each one is counted separately.
- **Total Series**: The total number of time series in the metric event, where a time series represents a unique combination of a metric and a dimension pair.
- **Unique Metrics**: The total number of distinct metrics within the sample data.
- **Unique Dimensions**: The total number of dimensions across all metric events.

Beneath the summary, the Metrics view presents a table with the following columns:

- **Name**: The name of a unique metric.
- **Type**: The type of metric, such as a gauge, histogram, and more.
- **No. of Series**: The number of time series that this metric and its dimensions creates, calculated as the unique combination of a metric and a dimension pair.
- **Percent of Total Series**: The proportion of time series represented by this metric, shown as a percentage of the total series.

Each row represents a single metric, which you can expand to show its associated dimensions. The expanded view displays additional details:

- **Dimension**: The name of the dimension.
- **Unique Value Count**: The number of unique values for this dimension, also known as dimensionality.
- **Example Values**: The first five recorded values in the sample data, providing insight into typical values.


IN
OUT
M
Select Fields (10 of 10)
This view only displays metrics events. To see all events, select the Event view.
16
Total Metrics
7
Total Series
6
Unique Metrics
4
Unique Dimensions
Name
Type
No. of ...
Percent of Total Se...
proc.cpu_perc
Dimension
Unique Value Count
Example Values
proc
1
node
pid
2
52964, 52969
unit
1
percent
proc.mem_perc
gauge
1
14.29%
proc.bytes_in
count...
1
14.29%
proc.bytes_out
count...
1
14.29%
proc.func.time.read
timer
1
14.29%
proc.func.time.write
timer
1
14.29%

6.6. Enrich Data Using Lookups

Use lookup files to enrich events in your Pipelines.

The following guides will help you use lookups to enrich data:

- About Lookups
- Configure Lookups
- Lookup Geographic Data from IP Addresses
- Lookups as Filters for Masks
- Lookups and Regex Magic

Consider also reading these related reference topics:

- Lookup Function
- C.Lookup() Expression

6.6.1. About Lookups

Lookup files are reference tables that you can apply to raw data events in your Pipeline. Typically, you use lookup files to enrich data by adding new fields or replacing the value of an existing field using the values contained in the lookup file.

There are many common use cases for lookup files, including:

- **Add fields to control how data is routed or filtered:** Use values from your raw events to look up related metadata (such as priority level, geographic region, or risk classification) and add it to the event as a new field. You can use these enriched fields later in your Pipeline to filter events, route them for additional processing, or send them to specific downstream Destinations. For example, you could enrich logs from `service_id="auth-prod"` with `priority="high"` and route them to a dedicated alerting Pipeline, while routing `priority="low"` events to long-term storage.
- **Make data more useful by human-readable business or operational fields:** Use lookup files to translate machine-readable values (such as user IDs, product SKUs, or error codes) into human-readable or business-relevant fields. This helps make logs and metrics easier to interpret and act on. For example, instead of just storing `product_id=8924`, you could enrich the event with additional fields: `product_name="Honeycrisp Apple"`, `product_category="Produce"`, and `region="Northwest"`.
- **Normalize inconsistent values across different sources:** Use lookup files to map inconsistent or vendor-specific field values (such as region codes, platform names, or status labels) to a standardized, canonical format. This helps ensure reliable aggregation, filtering, and reporting across diverse data sources. For example, you could map `region="US_WEST"`, `region="uswest"`, and `region="us-west-1"` to the same, consistent data format: `region="us-west"`.
- **Enrich events with time-based values:** Use lookup files that include time windows to enrich events with the correct value based on when the event occurred. This is useful for applying dynamic metadata such as pricing tiers, active campaigns, or software versions that change over time. For example, you could enrich a transaction event with the correct `discount_percentage` value by looking up the active promotion based on the `transaction_time` field.
- **Add geographic context to events using IP addresses:** Use lookup files or binary IP address databases to convert raw IP addresses in your event data into geographic fields such as country, region, city, or autonomous system. This added context makes it easier for downstream receivers to detect unusual activity, group traffic by region, or apply compliance policies. For example, you could enrich an event containing `client_ip="192.0.2.0"` with additional fields like `geo_country="US"`, `geo_city="Chicago"`, and `geo_org="ExampleNet"`. You could then route traffic from unexpected regions to a separate review path or generate alerts for geo anomalies. See Lookup Geographic Data from IP Addresses and the GeoIP Function documentation for information about this specific use case.

Overview of the Workflow for Using Lookups

To use lookups in your Cribl Stream deployment:

1. Plan your lookup strategy: Begin by making strategic decisions about how and where to use lookups in your deployment. This includes evaluating whether ingest-time or search-time lookup files are appropriate for your use case. You also need to estimate hardware needs and consider strategies for optimizing your system for better performance. The rest of this page will guide you through the planning process.
2. Upload a lookup file: Add your lookup files to either the global Knowledge Library or to a specific Pack, depending on your use case.
3. Add indexes to the lookup file: If you're using disk-based lookups, index the fields you'll use for lookups. Cribl Stream supports a maximum of four indexes per file.
4. Reference the lookup file in a Pipeline or Route: Use the Lookup Function or the `C.Lookup()` expression to enrich, route, or filter data based on lookup results.
5. Update lookup files as needed: You can update lookup files manually through the UI or programmatically using the API, which is the ideal method for automated or frequent lookup file updates.

Plan Your Lookup Strategy

Before adding lookup files to your deployment, it's important to define a strategy that aligns with your performance goals, data flow design, and operational needs. This includes:

- Deciding when to enrich your data, such as at ingest-time or search-time.
- Choosing your lookup storage mode based on the size of your files and system performance concerns.
- Preparing your lookup files for optimal performance.
- Understanding system resource requirements.
- Ensuring your implementation scales reliably over time.

Decide When to Enrich Data

One important consideration when using lookups in Cribl Stream is *when* to apply them to your data. The timing of enrichment can affect system performance, data freshness, and data consistency.

Cribl Stream performs lookup enrichment at ingest-time or transform-time, depending on where you use lookups in your Pipeline. This means Cribl Stream applies lookup values either at the moment data first enters the system or during a specific data processing stage. Once enriched, the new fields become part of the event and you can then use those fields for routing, filtering, or forwarding to downstream Destinations. Because this enrichment happens early, the enriched values remain consistent and available throughout the rest of your Pipeline.

Whether to enrich data at ingest-time or search-time depends on your broader data processing goals. As a general rule, use ingest-time lookups for stable reference data, and use search-time lookups for dynamic of frequently changing values.

- Choose ingest-time enrichment when the context is fixed and should reflect the moment in time when the event was processed.
- Choose search-time when the context is time-sensitive and needs to reflect the most current data at the time you execute the query.

The following table summarizes the tradeoffs:

Time	Description	Advantages	Disadvantages
Ingest-time or transform-time	Enrichment happens as the data is first received or as part of a processing stage.	• **Fast query performance:** Since the data is pre-enriched, downstream systems don't redo lookups. • **Immutable enrichment:** Enriched values are part of the stored record. • **Consistency:** All downstream consumers use the same enrichment snapshot.	• **Stale enrichment:** If the lookup source changes, the already-enriched events won't update. • **Higher ingest latency:** Lookups add work during processing, potentially impacting system performance. • **Larger storage footprint:** Enriched fields are stored with each event. • **Harder to fix mistakes:** Errors in enrichment logic persist with the data.
Search-time or query-time	Enrichment happens only when the system queries or analyzes the data.	• **Always fresh:** The data reflects lookup changes immediately. • **Lower ingest cost:** Faster ingestion and smaller payloads. • **Easier to improve:** Fix enrichment logic without reprocessing past data.	• **Slower query performance:** Lookups happen at query time, potentially impacting search performance. • **Inconsistent results:** Results may vary over time as lookup data changes. • **More complex logic:** You must apply enrichment logic during each query.

Because Cribl Stream applies lookup files during ingest or transformation, the enriched values become part of the event. These values won't reflect changes to the lookup file after data processing. This behavior ensures fast and consistent downstream behavior, but it also means you should carefully manage your lookup files and validate their accuracy before deploying it to production.

If you need enrichment to reflect current values at query time, consider using a downstream tool such as Cribl Search. Cribl Search supports search-time lookups against global lookup files in the Cribl Search Knowledge Library.

Cribl Search can also dynamically update lookup tables based on search results, which is useful for scenarios where enrichment conditions change after the event occurred or after initial processing. See Cribl Search Lookups for more information.

Choose a Lookup File Storage Mode

When uploading lookup files, Cribl Stream gives you two storage mode options:

- **Store in Memory (in-memory lookup):** The lookup file is stored in memory and packaged with your deployment. This option offers fast lookup access and is ideal for small lookup files that can be stored entirely in memory.
- **Store on Disk (disk-based lookup):** The lookup file is streamed to the Worker separately and accessed from disk at runtime. This option is suitable for larger files and more flexible deployments.

Every lookup function that references a disk-based file allocates a 16MB cache per Worker Process.

You can select storage modes on a file-by-file basis, and it's safe to mix both approaches in the same deployment.

If you are unsure which method to use, use in-memory lookups (Store in Memory) for better performance. However, there are times when it's better to use disk-based lookup storage (Store on Disk).

Use disk-based lookups when:

- Your lookup file is large (generally 25 MB or more) and when storing completely in memory is not feasible.
- You want to scale CPU cores independently from memory and reduce overall memory allocation per Worker.
- You manage lookup files using an external system outside of Cribl, and you want to migrate to an internal Cribl-based solution to reduce maintenance overhead.

Disk-based lookups have some limitations:

- **No in-place editing:** To update the lookup file, you must download, delete, and re-upload it.
- **Exact Match only:** When referencing disk-based lookups in Pipelines or Routes, the lookups are limited to Exact Match Mode only. They do not have CIDR or Regex support.

- **No Reload Period support:** The **Reload Period** setting in the Lookup Function is not supported for disk-based files.
- **Lookup file type restrictions:** Cribl Stream requires `.csv` format for disk-based lookup files. Use in-memory lookups instead if you need to use a different lookup file type, such as an external Redis database or binary IP address databases from either MaxMind (GeoLite2 or GeoIP2 `.mmdb` files) or IPinfo.

You can convert in-memory lookups to disk-based lookups, but not vice versa. Choose carefully before migrating, as the conversion is permanent.

Estimate Memory Usage

Disk-based lookups reduce memory usage by avoiding loading an entire file into memory. However, they introduce a fixed memory cost per function. Every lookup function that references a disk-based file allocates a 16MB cache per Worker Process.

You can estimate the expected memory usage for disk-based lookups with the following formula:

Required memory = `16MB` x `Number of Worker Processes` x `Unique Function Specs used.`

Optimize Your Lookup Files

To improve system performance and minimize processing overhead, follow these general best practices when creating lookup files:

- **Keep file sizes minimal:** Include only the fields necessary for data processing. Avoid storing extra metadata or unused columns.
- **Place your primary key in the first column:** Cribl Stream automatically indexes the left-most column by default. Organizing your primary lookup key in this position improves default performance and avoids the need for additional configuration. If your most important lookup field isn't in the left-most column, you'll miss out on automatic indexing and may experience slower performance until indexes are manually configured.
- **Avoid indexing low-cardinality fields:** Avoid indexing fields with very few unique values. These indexes offer little performance benefit and may increase overhead.

For disk-based lookups:

- **Index all fields you need to reference:** Cribl Stream requires you to index the field you use in the Lookup Function. If you reference a field that you have not indexed, the lookup will fail. For disk-based lookup files, you can define a maximum of four indexes to accelerate lookups on frequently queried fields. Choose fields that are used in routing, filtering, or enrichment logic to maximize performance benefits.

- **Avoid over-indexing:** While Cribl Stream supports a maximum of four indexes, indexing too many fields can increase memory usage and impact performance. Only index fields that are frequently queried or matched against.

Plan Your File Update Strategy

If you expect lookup values to change regularly over time, develop a clear strategy for keeping your lookup files accurate and up-to-date. An effective update strategy helps ensure consistent enrichment and prevents stale or incorrect data from propagating through your Pipelines. See Update a Lookup File for more information.

Consider using these strategies:

- Automate updates using the API if lookup files change frequently or if another system maintains the files.
- Version your files to track changes and allow rollbacks if needed.
- Validate file content before uploading to production to avoid enrichment errors.
- Coordinate updates across environments (such as staging and production) to ensure consistent behavior.

Supported System and File Requirements

This section outlines the supported file types, size limits, and hardware recommendations for using lookup files in Cribl Stream.

Memory Allocation

Consider whether an in-memory or disk-based lookup is more efficient in terms of memory usage.

Supported Lookup File Types

Cribl Stream supports and recommends using lookup files in `.csv` format for compatibility, performance, and ease of management. Disk-based lookups require `.csv` format.

Additional supported formats for in-memory lookups only:

- External Redis databases, which are supported using the Redis Function.
- Binary IP address databases from either MaxMind (GeoLite2 or GeoIP2 `.mmdb` files) or IPinfo, which are supported using the GeoIP Function. See Lookup Geographic Data from IP Addresses for information about this specific use case.

Supported Lookup File Sizes

Cribl Stream supports:

- **Maximum file size (per file):** 500 MB
- **Total storage limit (across all files):** Up to 2 GB (including both in-memory and disk-based lookup files)

Cribl.Cloud enforces these limits and does not allow changes. In hybrid and self-managed deployments, you can modify the default limits. However, increasing the limits is not officially supported, so make sure your deployment can safely handle this change before doing so.

For best performance and reliability, keep individual lookup files as small and focused as possible. Use multiple smaller files instead of one large one when feasible.

Hardware Recommendations

To optimize performance, ensure that you store disk-based lookups on high-performance storage. To avoid performance degradation, store disk-based lookup files on high-performance storage volumes.

Avoid:

- Network file systems (NFS)
- Non-solid state drives (HDDs)
- Any low IOPS environments

Use:

- Local SSD storage or similarly fast media for optimal throughput and lookup performance

In addition to storage needs, in-memory large lookup files can also require additional RAM on each Worker

6.6.2. Configure Lookups

This page walks you through the key tasks for configuring and using lookup files in Cribl Stream. You'll learn how to upload and index lookup files, reference them in Pipelines or Routes, update them, and remove them when they're no longer needed.

Overview of the Workflow for Using Lookups

To use lookups in your Cribl Stream deployment:

1. Plan your lookup strategy: Begin by making strategic decisions about how and where to use lookups in your deployment. This includes evaluating whether ingest-time or search-time lookup files are appropriate for your use case. You also need to estimate hardware needs and consider strategies for optimizing your system for better performance.
2. Upload a lookup file: Add your lookup files to either the global Knowledge Library or to a specific Pack, depending on your use case.
3. Add indexes to the lookup file: Improve performance by indexing the fields you'll use for disk-based lookups. Cribl Stream supports a maximum of four indexes per file.
4. Reference the lookup file in a Pipeline or Route: Use the Lookup Function or the `C.Lookup()` expression to enrich, route, or filter data based on lookup results.
5. Update lookup files as needed: You can update lookup files manually through the UI or programmatically using the API, which is the ideal method for automated or frequent updates.

Upload a Lookup File

You can upload lookup files as a global object in the Knowledge Library or as an object inside a Pack. The process for uploading a lookup file to a Pack is nearly the same as uploading it to the global Knowledge library. The processes are relatively similar. The key difference is that you must upload the file from inside the Pack in order to scope it correctly to that Pack's resources.

Before uploading a lookup file, make sure it's properly formatted and optimized for performance. See Optimize Your Lookup Files for guidance on preparing your file.

Upload a Lookup File to the Knowledge Library

To upload a new lookup file to the Knowledge Library:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.

2. Navigate to **Processing**, then **Knowledge.** The **Lookups** page opens by default. If it doesn't, select **Lookups** from the side menu.
3. Select **Add Lookup File**, then select how you want to create the file:
    - **Upload a New File:** Browse to and upload an existing lookup file from your computer. See Supported Lookup File Types for more information.
    - **Create with Text Editor:** Enter comma-separated values (CSV) directly using the built-in editor.
4. In the **New Lookup File** modal, configure the following settings:
    - **Filename:** Enter a unique name for the lookup file.
    - **Description:** Optionally, add a short description of the file's contents or purpose.
    - **Tags:** Optionally, add tags to help you group and filter lookup files on the **Lookups** page. These tags are not added to processed events. Separate tags with a tab or hard return.
    - **Mode:** Select how you want to store and access the lookup file. Options include **Store in Memory** or **Store on Disk**. See Choose a Lookup File Storage Mode for guidance about which mode to choose.
5. Select **Save**. The new lookup file appears in the table on the **Lookups** page. Check the **Status** column to confirm that the file is ready for deployment.
6. Select **Deploy** to deploy the lookup file to the Workers in the current Worker Group. In the **Deploy Group** modal, select or clear the check boxes next to the lookup files you want to deploy.

Disk-based lookups do not require you to commit lookup file changes in order to deploy them to Workers. Lookup filenames are automatically added to `.gitignore` when uploaded.

Upload a Lookup File to a Pack

Uploading a lookup file to a Pack follows the same general steps as uploading one to the global Knowledge Library. However, to ensure you have properly scoped the lookup file to the Pack, you must upload it from within the Pack itself. When referencing the file in Pipelines or Routes, be sure to use relative paths (often defined with environment variables) to maintain compatibility within the Pack context.

To upload a lookup file to a Pack:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Processing**, then **Packs.**
3. Select the Pack where you want to add a lookup file.
4. Inside the Pack, select the **Knowledge** tab. Then follow the standard lookup upload process. See Upload a Lookup File to the Knowledge Library for details.

5. After deploying the lookup file, validate that you can view the Lookup on your Workers.
6. When referencing the lookup file from within the Pack, ensure that you use a Pack-scoped path.

Do not reference lookup files using global paths or Leader paths, such as `$CRIBL_HOME/groups/<your group>/data/lookups/host_destination_test.csv`. Also avoid using unqualified filenames like `host_destination_test.csv`.

When working with disk-based lookup files inside a Pack, the **Undo All Changes** button in the Git commit modal does not revert changes to those lookup files. Because Cribl Stream excludes disk-based lookups from Git version control in the UI, you must track modifications and roll them back manually.

Add Indexes to a Lookup File (Disk-Based Only)

In a lookup file, an index is a fast-access data structure built from one or more header fields. For disk-based lookup files, Cribl Stream requires you to index the field you use in the Lookup Function. If you reference a field that you have not indexed, the lookup will fail.

Cribl Stream pre-processes the indexed fields to avoid row-by-row scans. Instead of searching the entire file for a match, the system can jump directly to the relevant row. This indexing mechanism significantly improves lookup performance, especially for large files that might otherwise affect system latency.

Some notes about index behavior for disk-based lookups:

- Cribl Stream automatically indexes the left-most column of each lookup file by default. To take advantage of this, place your primary key fields in the first column.
- You can define a maximum of four indexes per lookup file.
- You can set each index to have match values in a case-sensitive or case-insensitive manner.

Before uploading a lookup file, make sure it's properly formatted and optimized for performance. See Optimize Your Lookup Files for guidance on preparing your file.

To add indexes to a disk-based lookup file:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Processing**, then **Knowledge**. By default, this opens the **Lookups** page. If not, select it from the side menu.
3. On the **Lookups** page, select the name of the lookup file you want to edit.
4. In the **Indexes** setting, select **Add Index** or select an existing row to edit it. In the menu that appears, select the column you want to index.

5. Toggle **Case-sensitive** on to enable case-sensitive matching for the indexes listed in that row.
6. Select **Save** to apply your changes and return to the **Lookups** page.

Reference a Lookup File or GeoIP Database File in a Route or Pipeline

After uploading a lookup file or GeoIP database to Cribl Stream, you can then reference that file to enrich your data. You can reference a lookup file in one of two ways:

- **Use the Lookup Function in your Pipeline.** Add the Lookup Function to your Pipeline. Any lookup files you have uploaded to the Knowledge Library appear in the **Lookup file path** drop-down menu. You can also manually enter the lookup file path. If needed, you can reference environment variables using `$`, such as `$CRIBL_HOME/file.csv`. See Lookup Function for more information.
- **Use the `C.Lookup()` expression in a Route or Pipeline.** You can call the `C.Lookup()` expression in any field that accepts JavaScript expressions, such as in the **Filter** field. See C.Lookup - Inline Lookup Methods and Build Custom Logic to Route and Process Your Data for more information.

Default lookup file paths:

Deployment Type	Path
Cribl-managed (Cloud)	`$CRIBL_HOME/groups/<group-name>/data/lookups/<file-name.csv>`
Single instance (on-prem)	`$CRIBL_HOME/data/lookups/<file-name.csv>`
Distributed (on-prem)	`$CRIBL_HOME/groups/<group-name>/data/lookups/<file-name.csv>`
Lookups stored in a Pack	`$CRIBL_HOME/data/lookups/packs/<pack-name>/<file-name.csv>`

Delete a Lookup File

Deleting a lookup file that is actively referenced in Pipelines or Routes can result in runtime errors or failed deployments. Before deleting a file, verify that it is no longer in use or update all references as needed.

To delete a lookup file:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Processing**, then **Knowledge.** By default, this opens the **Lookups** page. If not, select it from the side menu.
3. On the **Lookups** page, check the box next to the names of any lookup files you want to delete.
4. Select **Delete Selected Lookup Files**.

5. Type `DELETE` to confirm.
6. Verify the lookup file no longer appears on the **Lookups** page.
7. Commit and deploy your changes.

Update a Lookup File

Your options for updating lookup files might be limited based on the storage mode you selected at configuration time. Disk-based lookup files cannot be edited in place using the UI. See Plan Your Lookup File Storage Mode for more information.

You don't need to upload a new disk-based lookup file just to update its indexes. You can modify indexes directly. See Add Indexes to a Lookup File for more information.

Update a Lookup File Using the API

You can update lookup files using the API if needed. Use the API if your lookup files need frequent updates or as part of an automated process. See Create and Update Lookups in the API documentation for more information.

Update a Lookup File in the UI (for In-Memory Lookups)

For in-memory lookups, you can edit lookup files directly through the UI:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Processing**, then **Knowledge.** By default, this opens the **Lookups** page. If not, select it from the side menu.
3. On the **Lookups** page, select the name of the lookup file you want to edit.
4. In the settings for the lookups file, you can edit the file directly using either table or text mode:
    - **Table:** Double-click cells to change their value.
    - **Text:** Directly edit the lookup file in comma-separated values (CSV) text format.
5. Select **Save** to apply your changes and return to the **Lookups** page.

Update a Lookup File in the UI (for Disk-Based Lookups)

For disk-based lookups, you can't edit lookup files directly through the UI. To update a file, you must delete the existing version and upload a new one.

You don't need to upload a new disk-based lookup file just to update its indexes. You can modify indexes directly. See Add Indexes to a Lookup File for more information.

To update a disk-based lookup file:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Processing**, then **Knowledge.** The **Lookups** page opens by default. If it doesn't, select **Lookups** from the side menu.
3. On the **Lookups** page, find the name of the lookup file you want to edit and select the **Download** icon in the **Action** column.
4. Open the downloaded file in a text editor or spreadsheet tool, and make the necessary changes.
5. Delete the original lookup file from the Knowledge Library.
6. Re-upload the updated version. See Upload a Lookup File for more information.

If the re-uploaded file uses the same filename, any references to the lookup file by name will continue to work without changes. However, if you upload the file under a different name, you'll need to update all references to point to the new filename.

6.6.3. Lookup Geographic Data From IP Addresses

In many observability and security use cases, raw event data includes an IP address, but not much context about where that IP originated from. By performing lookups with geographic metadata, you can enrich data events with human-readable fields such as country, region, city, or network ownership. These fields could help you route or filter data or normalize data for a downstream alerting or reporting tool.

Cribl Stream supports geographic enrichment using the GeoIP Function, which adds location-related fields to events based on an IP address. The GeoIP Function works with binary IP address databases from either MaxMind (GeoLite2 or GeoIP2 `.mmdb` files) or IPinfo.

Enriching events with geographic data lookups unlocks a wide range of operational and security workflows. For example:

- **Add geographic context to events using IP addresses:** Use IP-to-location data to convert raw IP addresses into meaningful fields like `geo_country`, `geo_city`, `geo_latitude`, and `geo_org`. This added context allows you to group, filter, or analyze events based on location. For example, enriching `client_ip="192.0.2.0"` could yield `geo_country="US"` and `geo_city="Chicago"`, helping you detect geographic anomalies or route the event to a specific Destination.
- **Detect suspicious or unexpected access patterns:** Identify events coming from countries or regions that fall outside expected patterns for a given service or customer base, such as sudden spikes in traffic from high-risk countries or VPN exit nodes.
- **Support compliance and data sovereignty policies:** Tag or route events differently based on geographic origin to comply with regulations like GDPR or regional data residency requirements.
- **Enable geo-based reporting and dashboards:** Aggregate and visualize metrics by country, region, or city to gain insight into user distribution, application access, or network flow patterns.
- **Correlate with external threat intelligence or allowlists:** Use geographic fields to compare against known malicious IP ranges or to filter out traffic from unapproved regions.

See About Lookups for information about how to generally use lookups in Cribl Stream.

How to Use MaxMind Geographic Databases with Cribl Stream

This section explains how to configure geographic lookups using MaxMind.

Step One: Download and Extract the MaxMind Database

We'll illustrate this with an example that often combines all three conditions: setting up the free, popular MaxMind GeoLite2 City database to support Cribl Stream's GeoIP Function. This example anticipates a Cribl Stream production distributed deployment, where the GeoLite database is updated nightly across multiple Workers.

This example includes complete instructions for this particular setup. However, you can generalize the example to other MaxMind databases, and to other large lookup files - including large `.csv`'s that similarly receive frequent updates.

To enable the GeoIP Function using the MaxMind GeoLite 2 City database, your first steps are:

1. Create a free MaxMind account, at the page linked above.
2. Log in to your MaxMind account portal and select **Download Databases**.
3. On the Download page, look for the database you want. (In this example, you'd locate the **GeoLite2 City** section.) Note the **Format: GeoIP2 Binary**, and select **Download GZIP**.

GeoLite2 City database: Download binary GZIP

4. Extract the archive to your local system.
5. Change to the directory created when you extracted the archive. This directory's name will correspond to the date you downloaded the file, so in the above `2020-10-06` example, you would use: `$ cd GeoLite2-City_20201006`

Step Two: Download and Extract the IPinfo Database

IPinfo's IP Geolocation, IP to privacy detection, ASN, IP to Company, Abuse contact, and WHOIS databases are supported. IPinfo also provides the following free IP databases: IP to Country, IP to ASN, and IP to Country ASN databases.

To enable the GeoIP Function using the IPinfo database:

1. Download the MMDB format database from your IPinfo account dashboard.
2. Extract the archive to your local system.
3. Change to the directory created when you extracted the archive.

Step Three: Copy the Database File to the Leader and Worker Nodes

Use the UI or the API to add the database file to each Worker in the Worker Group where you intend to use the GeoIP Function. You can only upload Maxmind database files as an in-memory lookup file type. See Configure Lookups for information about uploading and updating lookup files.

Step Four: Update the Database File

MaxMind updates its database files regularly, so consult the MaxMind documentation for update schedules.

Use the GeoIP Update program MaxMind provides to automatically update your databases. See MaxMind's Automatic Updates for GeoIP2 and GeoIP Legacy Databases documentation for details.

You'll need two modifications specific to Cribl Stream:

- GeoIP Update must be set up on the Leader, and on each Worker in any Group that uses GeoIP lookups.
- You need to configure GeoIP Update to point to the directory where the MaxMind database file resides. The program uses a configuration file named `GeoIP.conf` and you utilize the `DatabaseDirectory` parameter to define the location for storing the database file. See MaxMind's GeoIP.conf documentation for details.
  For example, the default setting in `GeoIP.conf` writes output to `/usr/local/share/GeoIP`. You must change this setting to the path where your databases actually reside. If you're using the recommended architecture above, you'd set: `DatabaseDirectory <$CRIBL_HOME>/state/`.

A Cribl Worker restart is necessary for the new data to be loaded. Even though the update happens automatically, the cached version loaded at the Worker's startup persists until the Worker is restarted. To ensure that updated MaxMind databases are used for processing, schedule regular Worker restarts to coincide with the automatic updates. This applies to all Cribl Stream deployments, both on-prem and Cribl.Cloud.

Upload and Update IPinfo Databases

You can download the IPinfo database through its storage URI. The storage system simply accepts the IPinfo authentication token and the database filename to allow you to download your IP database.

For example, to download the IPinfo's Free IP to Country ASN database, the URI/API command is as follows:

```
curl -L <https://ipinfo.io/data/free/country_asn.mmdb?token=><YOUR_TOKEN> -o
country_asn.mmdb
```

You can also use Wget or any tool you prefer. You can also stream the output to a file.

6.6.4. Lookups As Filters For Masks

You can make your data architecture more maintainable by using Lookups to route and transform events within Cribl Stream. This use case demonstrates an unusual solution, but one that served one Cribl customer's particular goals (which might overlap with yours):

- Ingest many (hundreds of) different `sourcetype/index` field combinations.
- Send all this data through a common Pipeline.
- Stack four Mask Functions in the Pipeline.
- Evaluate and process each `sourcetype/index` field combination **only** within its applicable Mask Functions - either two or three Masks per combination.

This last restriction reduces latency, by preventing Mask Functions from evaluating non-applicable events, simply to ignore them.

Just to reiterate, this use case outlined here responded to this customer's requirements - one Pipeline combining multiple Mask Functions, for many `sourcetype/index` combinations. More typically, you'd use multiple Pipelines to process different `sourcetype/index` combinations.

To enable this approach, the example below centralizes masking logic for multiple conditions in a Lookup table and corresponding Lookup Functions. The Lookup's output filters events to the applicable Mask Functions. Specifically, we'll show how to instruct Cribl Stream to:

- Check for a particular `index/sourcetype` combination in each event, and
- Based on that combination, determine which Masks to apply to that event.

Design the Lookup

To use a lookup as a filter, you'd start by creating a comma-separated lookup table in this format, and adding it to Cribl Stream:

index_tracker.csv

```
index,sourcetype,masks
apache_common, sourcetypec, ssn|credit_card|auth_token
syslog,sourcetypeb,ssn|auth_token
weblog,sourcetypea,auth_token|bearer_token
```

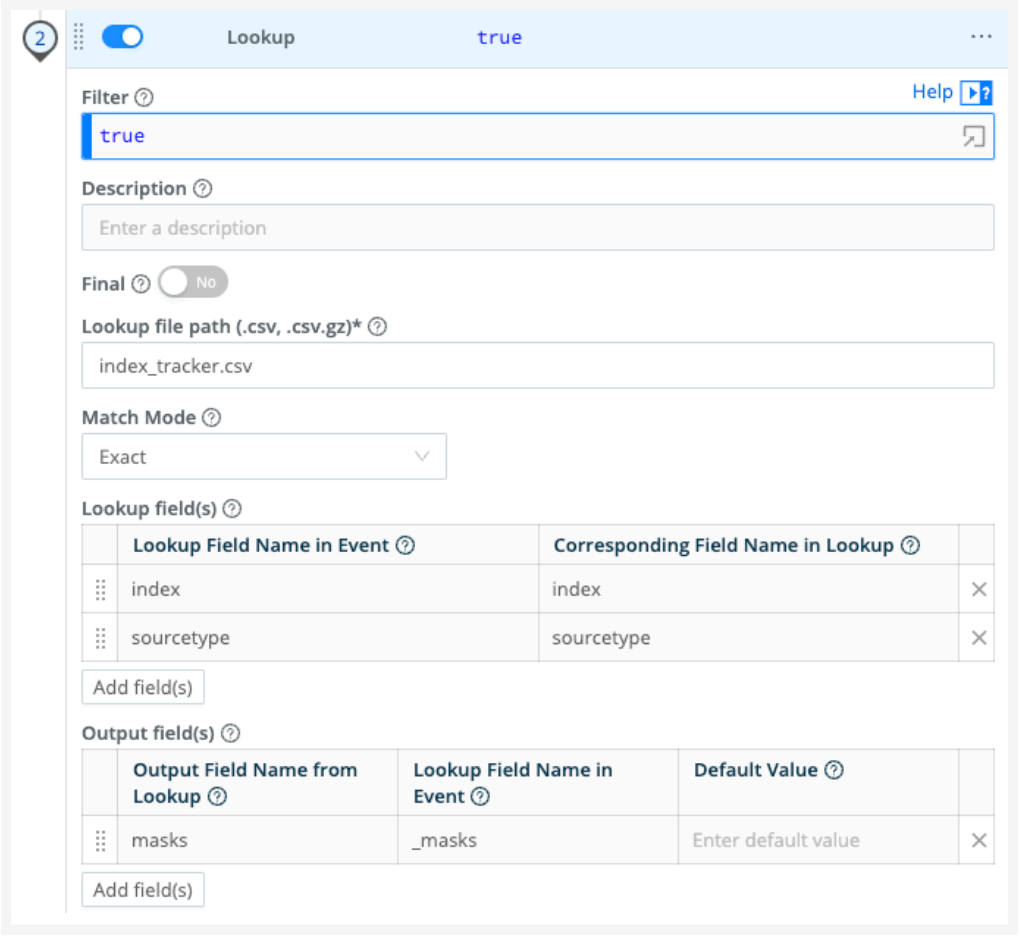
Copy

Below the header, each row specifies an index, a sourcetype, and (in the third column) a pipe-delimited list of applicable masks.

To make this example work, the table must have only **one** row for each index/sourcetype combination. (This unusual restriction is particular to this scenario.) So, as you build out the lookup table, you cannot add new masks for **existing** index/sourcetype combinations by appending new rows. Instead, you must modify the third column of the existing rows.

Configure the Pipeline

Create a Cribl Stream Pipeline with a Lookup Function configured like this, pointing to your lookup table:



Lookup Function's configuration

This Function keys against both the `index` and `sourcetype` fields. When it finds a matching combination, it adds a new key-value pair to your event for future filtering.

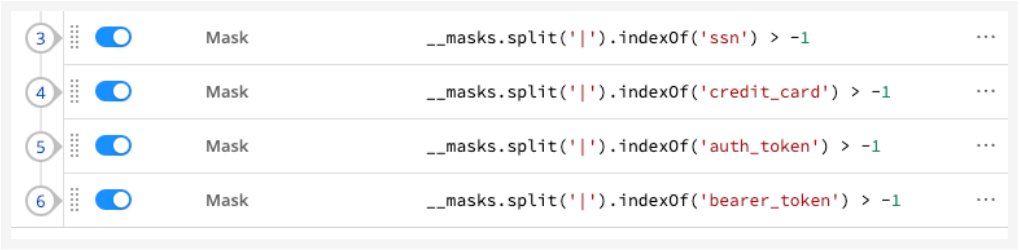
The key of that key-value pair (namely, `__masks`) starts with a double underscore, to make it a Cribl Stream internal field. This convention ensures that the key-value pair will **not** get passed along to the Destination.

However, you might prefer to export the key-value pair. For example, you might want a Splunk Destination to index the list of masks applied to a given event, alongside that event. (This approach applies to many forensic

use cases.) If so, remove the double underscore from the above Function's **Lookup Field Name in Event** value, and from the subsequent Filter expressions for each Mask Function.

Each Mask Function has a JavasScript Filter that breaks the pipe-delimited string into an array, and determines whether the tag for that type of mask (e.g., `bearer_auth`) is in the `__masks` key-value pair. If so, it applies the mask processing. If not, the event moves on to the Pipeline's next Mask Function.

Here are the four Mask Functions below the Lookup Function:



Mask Functions

In this particular example, the pipe-delimited mask tags in the lookup table's third column match the Mask Functions' names, as well as matching their **Filter** conditions. This is just for simplicity - the Functions could have any names, as long as the **Filter** expressions match the tags.

6.6.5. Lookups And Regex Magic

Regular expressions are not just for field extractions - they can also be used inside lookup tables, and in Functions, to replace and manipulate values within fields. Let's walk through a Pipeline that demonstrates four different ways to leverage regular expressions in Cribl Stream.

Why Lookup Tables Matter

When organizations use host naming standards, it is easy to understand things like regions, availability zones (AZs), IP addresses, and more. For example, consider an Amazon host called:

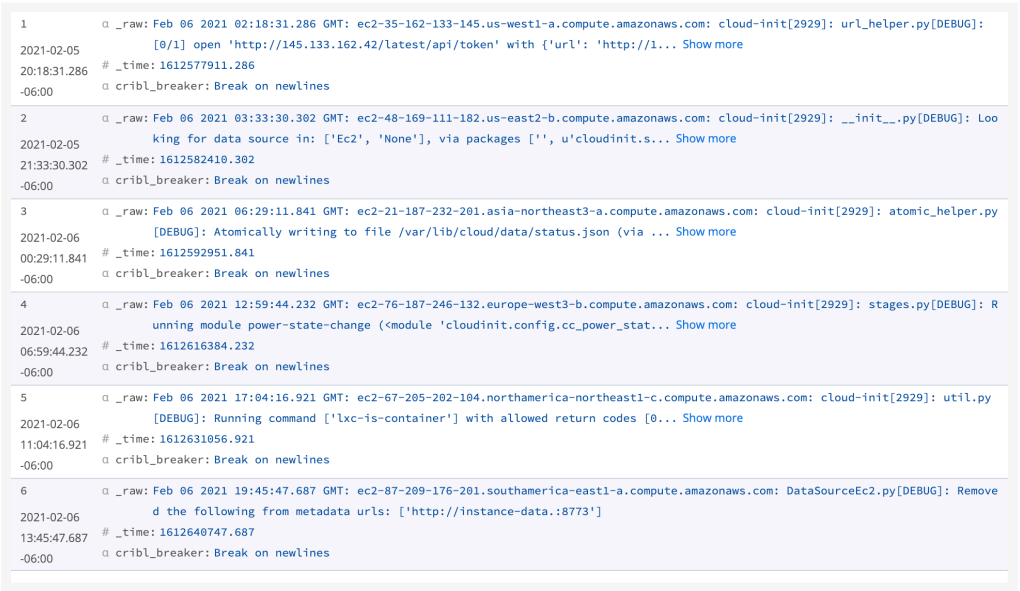
`ec2-35-162-133-145.us-west1-a.compute.amazonaws.com`

This is an EC2 host with a (dashed) IP address `35-162-133-145`, in the `us-west1` Region, in Availability Zone `a`. You can also see the domain: `compute.amazonaws.com`.

While we can understand the enriched host names, we don't know which indexes to route the data to, nor which sourcetypes to assign to the events, without looking up this information from another source. Doing so is often a huge challenge for organizations. To solve this challenge, let's combine Regex Extract, Lookup, and Eval Functions with some sample events to demonstrate the power of Cribl Stream.

Sample Events

The events below have timestamps broken out, but no indexes, sourcetypes, or other details have been assigned yet:



The Regex Extract Function

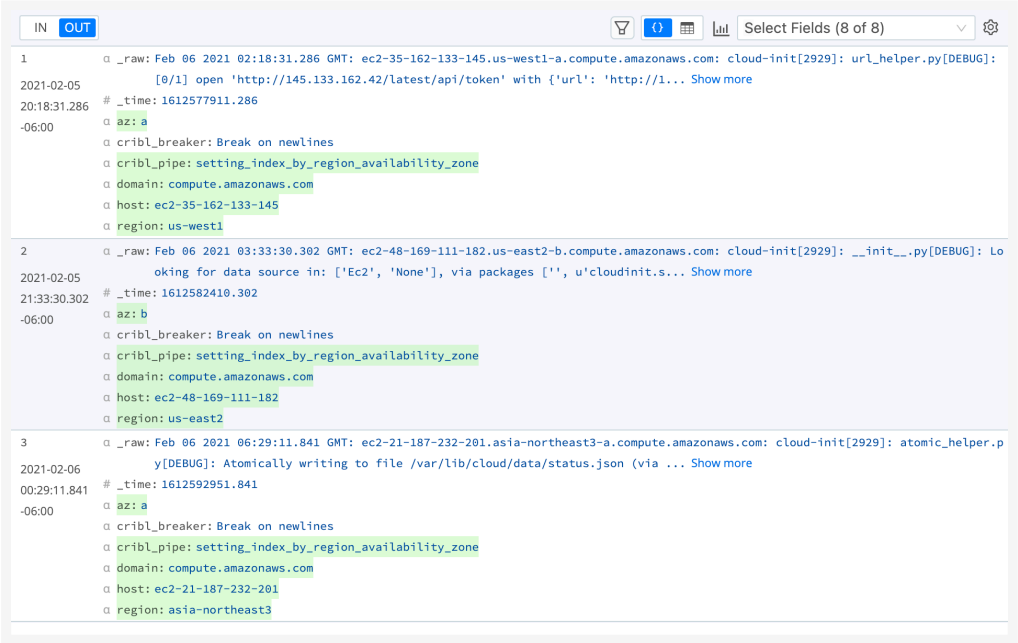
Before we can assign an index or sourcetype, we need to extract the `host`, `region`, `az`, and `domain` fields from the events. We can use a Regex Extract Function with this regular expression to extract all four fields:

```
GMT:\s+(?<host>[^.]+)\.(?<region>\w+-\w+\d+)-(?<az>[^.]+)\.(?<domain>[^:]+):
```

Here's that Regex Extract in a Cribl Stream Pipeline:

Results of the Regex Extract Function

On the **OUT** tab of Cribl Stream's Preview pane, the extracted fields of `az`, `domain`, `host`, and `region` now appear below the `_raw` event. You can use these extracted fields for searching in your preferred search solution.



Lookups

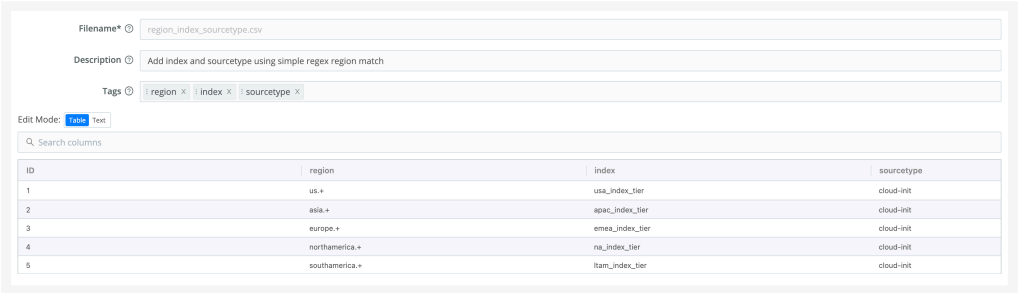
We still need to determine the index and sourcetype. Cribl Stream's Lookup Function enriches events with external fields. We'll use it with the newly extracted `region` field to assign an `index` and `sourcetype` to these events.

Lookup File

First, we need to create a lookup table for the Function to reference. For this, we'll use regex again.

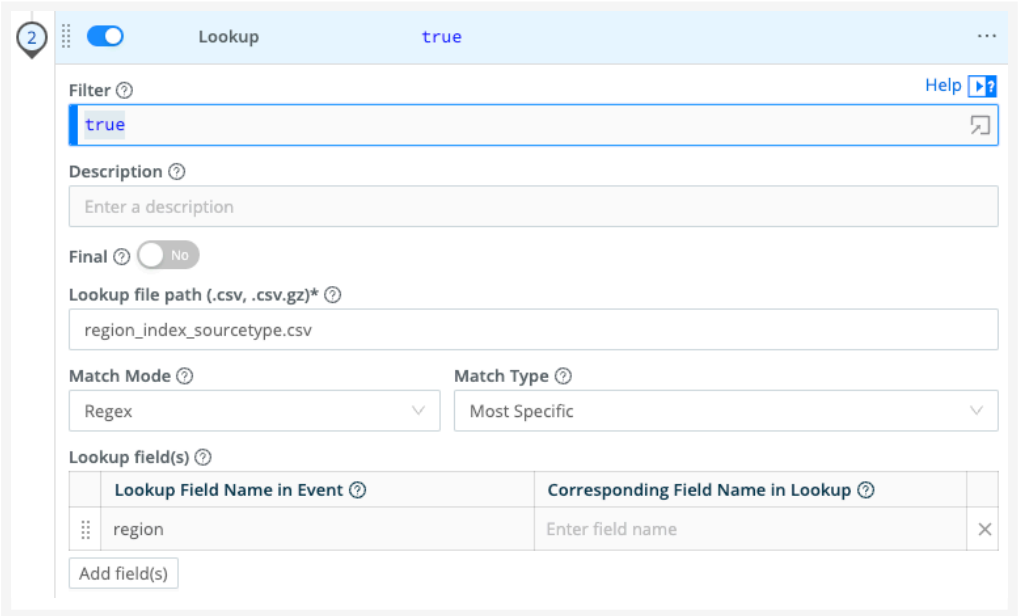
In the table below, five simple regular expressions map the extracted `region` field to the appropriate `index` and `sourcetype`. For example, the region `us-west1-a` starts with `us`, so it matches the first regular expression: `us.+`

We use this lookup table's first row to assign an index of `usa_index_tier`, and a sourcetype of `cloud-init`, to each matching event. The region patterns in the table's four remaining rows work the same way.



Lookup Function

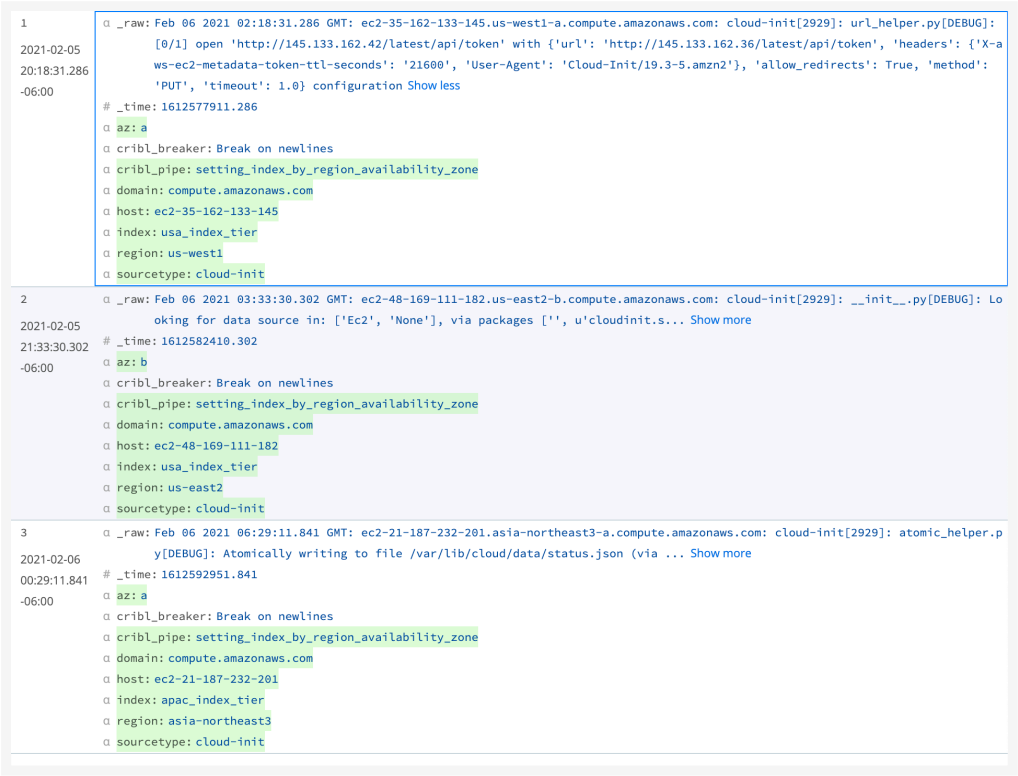
With the lookup table saved as `region_index_sourcetype.csv`, the Lookup Function below matches the events' extracted `region` field against the regular expressions, and returns the matching `index` and `sourcetype`.



There's actually more here than meets the eye. Note that we've specified no **Output Fields**. From the Lookup Function's documentation, we know this means that the Function will default to outputting **all** fields in the lookup table. So we get the contents of both remaining columns in the table we saw above: `index` and `sourcetype`.

Results of the Lookup Function

With the Lookup Function added to our Pipeline, the Preview pane's **OUT** tab shows that the `index` and `sourcetype` are now added to each event.

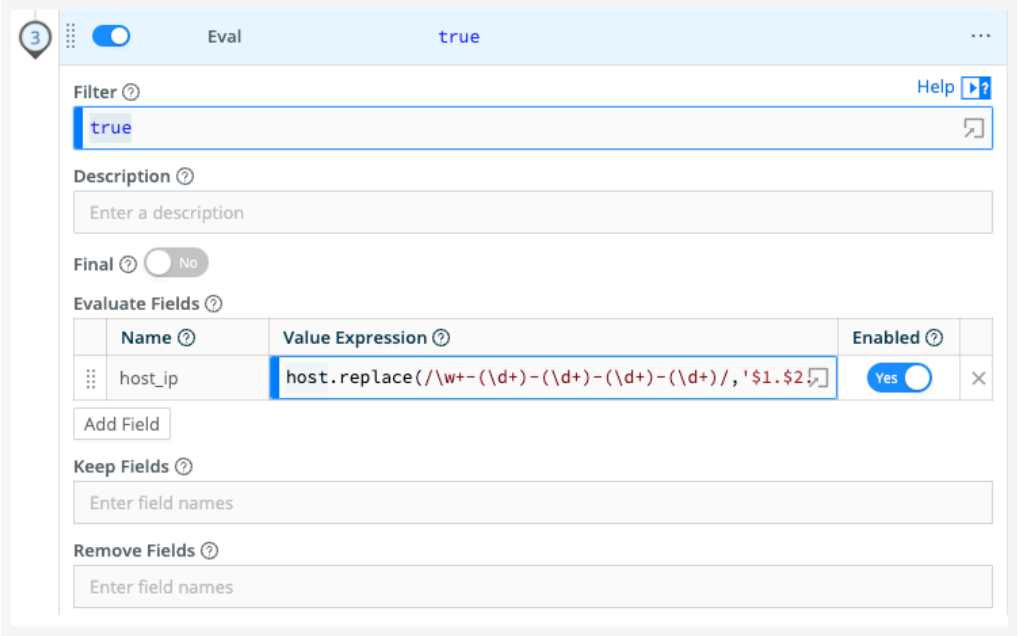


Getting Host IP Address from Host Name

Since the IP address is present in the `host` field, we can create the `host_ip` field using an Eval Function with this replace method:

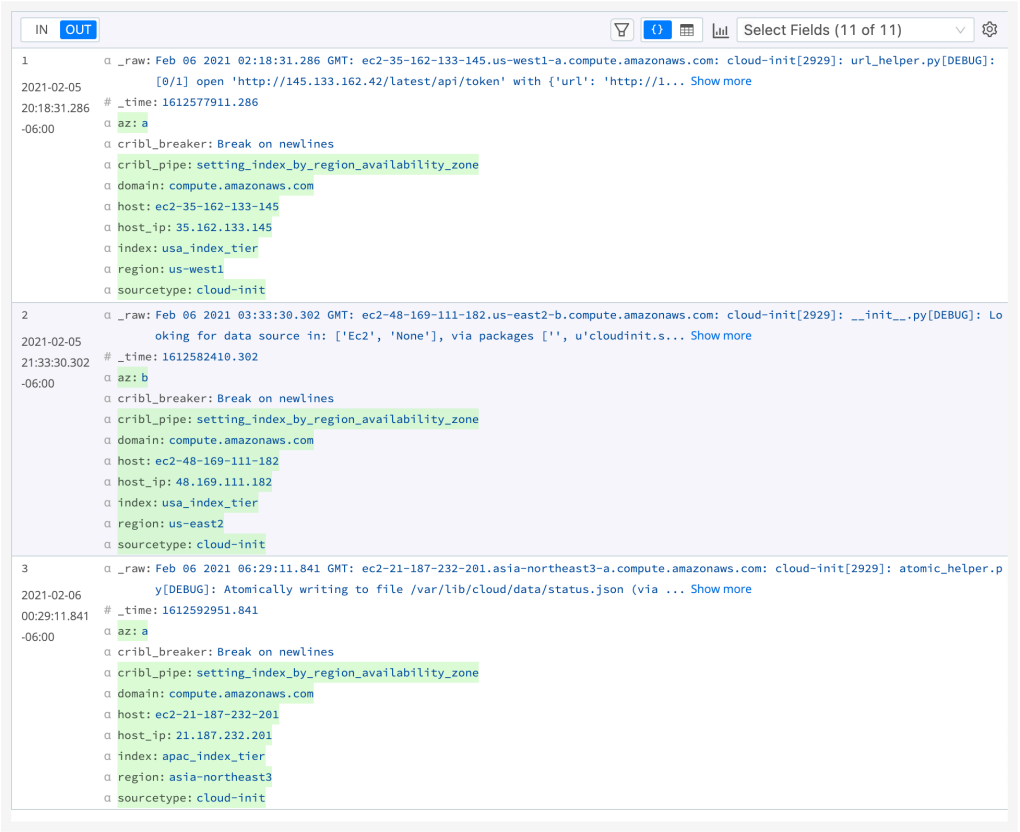
```
host.replace(/\w+-(\d+)-(\d+)-(\d+)-(\d+)/,'$1.$2.$3.$4')
```

This regular expression uses capture groups and pulls the four IP octets present in the hostname to build the `host_ip`. These four capture groups are noted as `$1.$2.$3.$4`, respectively. This method is very fast, and removes the need to perform a DNS lookup from the `host` field to get the host's IP address. Magic!



Results of the Eval Function and Replace Method

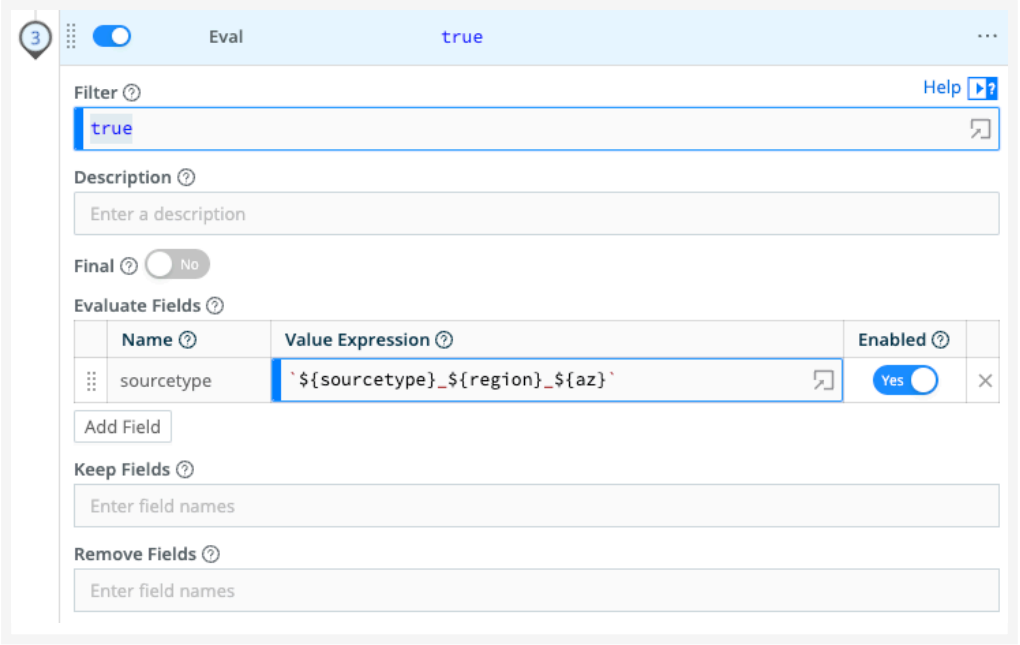
The `host_ip` field is now added to the events, displayed below `host`:



Customizing the Sourcetype

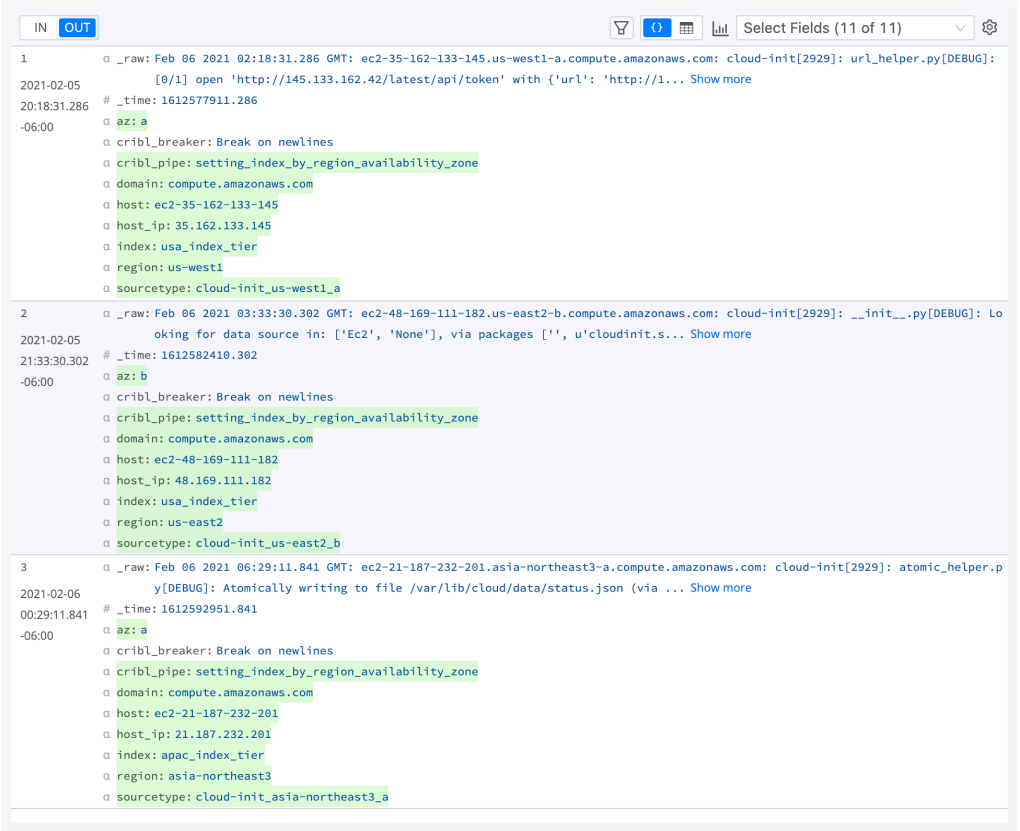
Finally, let's put some sense into the `sourcetype` field, using another Eval Function. By combining the values of the `${sourcetype}_${region}_${az}`, the sourcetype becomes `cloud-init_us-west1_a` - so now you can understand much more about the sourcetype at a glance.

Examine this Eval Function's value expression, taking careful note of the backticks (` `) and braces ({ }) that surround the field names, and the underscore (_) that separates them.



Results of the Eval Function to Combine Values

Take a look at the updated sourcetypes, and enjoy exploring Cribl Stream with your new knowledge!



Try This at Home

Below you'll find the lookup table, Pipeline, and sample events demonstrated in this use case. Create the lookup file first, and then import the Pipeline. (The order matters, because the Pipeline import depends on the lookup table's presence.)

Lookup Table

To create the lookup table in Cribl Stream's UI, select **Knowledge** > **Lookups**, then click **New Lookup File** and select **Create with Text Editor**.

Copy and paste in the header and rows listed below, then save the result as: `region_index_sourcetype.csv`.

region_index_sourcetype.csv

```
region,index,sourcetype
us.+,usa_index_tier,cloud-init
asia.+,apac_index_tier,cloud-init
europe.+,emea_index_tier,cloud-init
northamerica.+,na_index_tier,cloud-init
southamerica.+,ltam_index_tier,cloud-init
```

Copy

Pipeline

Below is an export of the whole Cribl Stream Pipeline presented here. Import this JSON to get a Pipeline named: `setting_index_by_region_availability_zone.json`.

Magic Pipeline

```
{
  "id": "setting_index_by_region_availability_zone",
  "conf": {
    "output": "default",
    "groups": {},
    "asyncFuncTimeout": 1000,
    "functions": [
      {
        "filter": "true",
        "conf": {
          "comment": "This pipeline demonstrates four different ways to leverage regu
        },
        "id": "comment"
      },
      {
        "filter": "true",
        "conf": {
          "source": "_raw",
          "iterations": 100,
          "overwrite": false,
          "regex": "/GMT:\\s+(?<host>[^.]+)\\.(?<region>\\w+-\\w+\\d+)-(?<az>[^.]+)\\
        },
        "id": "regex_extract",
        "disabled": false,
        "description": "Extract host, region, availability_zone, and domain"
      },
      {
        "filter": "true",
        "conf": {
          "matchMode": "regex",
          "matchType": "specific",
          "reloadPeriodSec": 60,
          "addToEvent": false,
          "inFields": [
            {
              "eventField": "region"
            }
          ],
          "ignoreCase": false,
          "file": "region_index_sourcetype.csv"
        },
        "id": "lookup",
```

```
        "disabled": false,
        "description": "Lookup index and sourcetype using regex matching"
      },
      {
        "filter": "true",
        "conf": {
          "add": [
            {
              "name": "host_ip",
              "value": "host.replace(/\\w+-(\\d+)-(\\d+)-(\\d+)-(\\d+)/,'$1.$2.$3.$4'
            }
          ]
        },
        "id": "eval",
        "description": "Create IP from hostname, removing DNS requirement",
        "disabled": false
      },
      {
        "filter": "true",
        "conf": {
          "add": [
            {
              "name": "sourcetype",
              "value": "`${sourcetype}_${region}_${az}`"
            }
          ]
        },
        "id": "eval",
        "description": "Append region and az to sourcetype",
        "disabled": false
      }
    ]
  }
}
```

6.7. Transforming Data

Cribl Stream offers multiple ways of transforming, enriching, reducing, and redacting data.

The following use cases guide you through transforming data in specific situations:

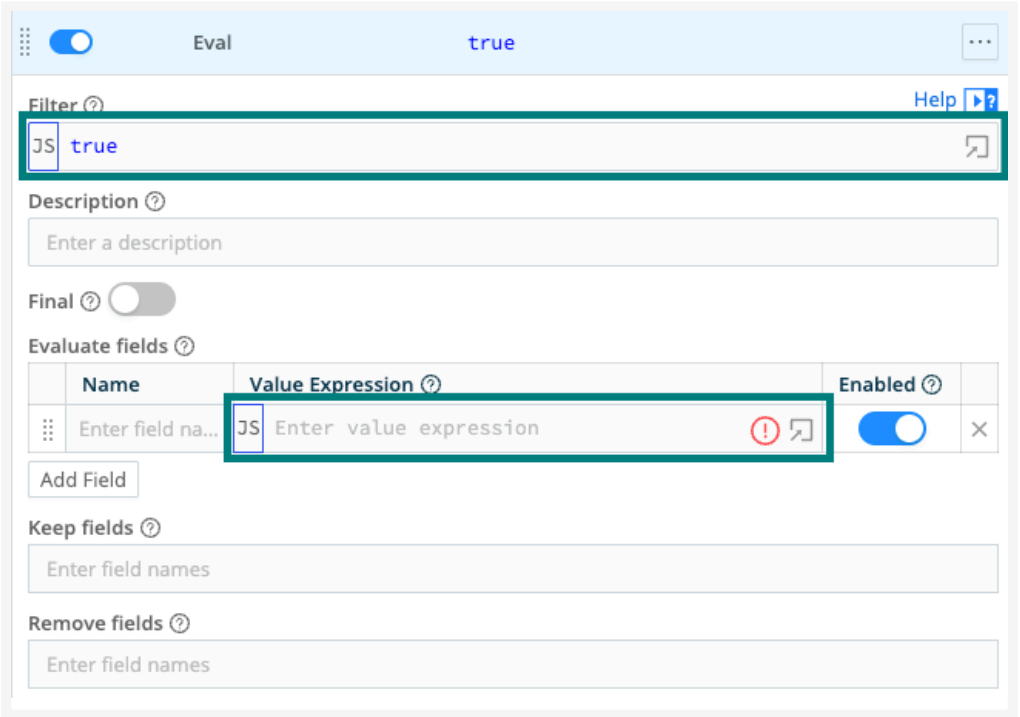
- Build Custom Logic to Route and Process Your Data
- Code Function Examples
- Ingest-Time Fields
- Masking and Obfuscation
- Encryption of Data in Motion
- Reducing Windows XML Events
- Regex Filtering

6.7.1. Build Custom Logic To Route And Process Your Data

Throughout Cribl Stream, many settings let you use custom JavaScript code and Cribl Expressions to control how data gets routed or processed. You will encounter JavaScript-enabled fields like these in several parts of Cribl Stream:

- **Routes:** Build custom logic to dynamically control where data gets routed.
- **Functions:**
  - Every Function has a **Filter** field that accepts JavaScript to determine which data the Function will process.
  - Some Functions also include additional fields for applying custom logic to event data.

This screenshot shows an example of two common types of JavaScript expression fields in the Eval Function: a filter expression and a value expression. These fields have a `JS` badge to visually indicate that they accept JavaScript input:



Examples of JavaScript Expression fields

You can use JavaScript syntax in any field that allows JavaScript expressions. You can also use Cribl Expressions, which are pre-built helpers designed for common operations in Cribl Stream. Cribl Expressions begin with `C.`, such as `C.Lookup` or `C.Time`.

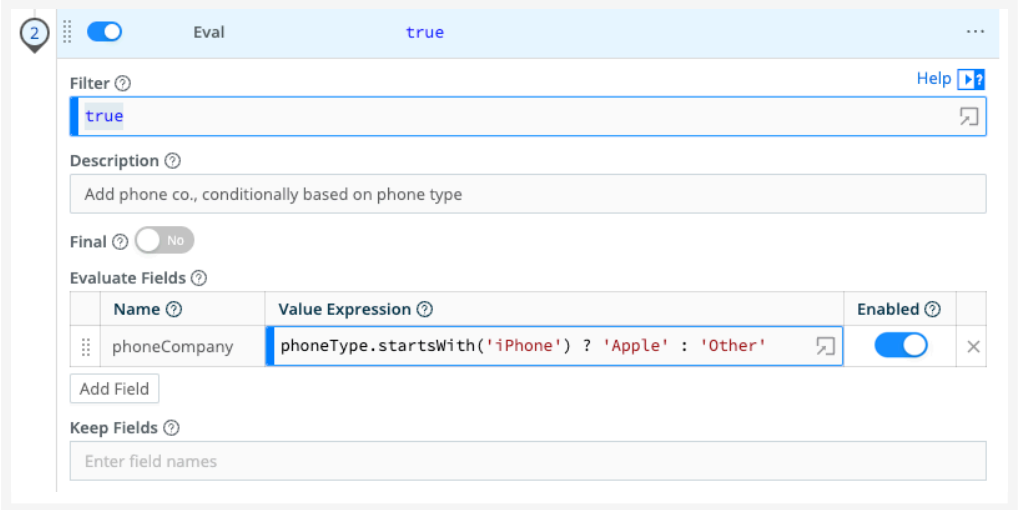
The JavaScript Expressions Sandbox tutorial provides hands-on experience if you would like to learn

more about them through personal experimentation and exploration.

Example Use Case

You can use JavaScript Expression fields to create custom logic for handling data based on your use case. To give a specific use case as an example, perhaps you want to enrich your incoming data by adding a new field called `phoneCompany`, which will assign a value of `Apple` or `Other` based on the contents of another field named `phoneType`.

You could use the Eval Function, which has a **Value Expression** field that accepts custom JavaScript expressions. In that field, you could use the expression `phoneType.startsWith('iPhone') ? 'Apple' : 'Other'`. This JavaScript expression checks if the string `phoneType` starts with `iPhone`. If it evaluates to true, it sets the return value to `Apple`. If not, it sets it to `Other`.



Adding a field to enrich data

Types of JavaScript Fields

Filter Expressions

You will use filter expressions in:

- Routes to dynamically control which data goes to which Pipelines.
- Functions to set the scope or narrow down which data gets processed by that Function.

The **Filter** field on Functions or Routes can accept custom filter expressions to control whether a Function or Route processes each event. Filter expressions must evaluate to either `true` or `false`:

- If the result is `true`, the data gets processed by that Route or Function.
- If the result is `false`, the data will pass through unchanged.

By default, the **Filter** field is set to `true`, meaning the Route or Function applies to all events. You can replace this with a custom JavaScript expression to apply the Route or Function only to data that matches specific conditions.

Value Expressions

Use value expressions in Functions to assign or calculate values dynamically. Use these expressions to create new fields, modify existing ones, or perform transformations on data.

For example, you can:

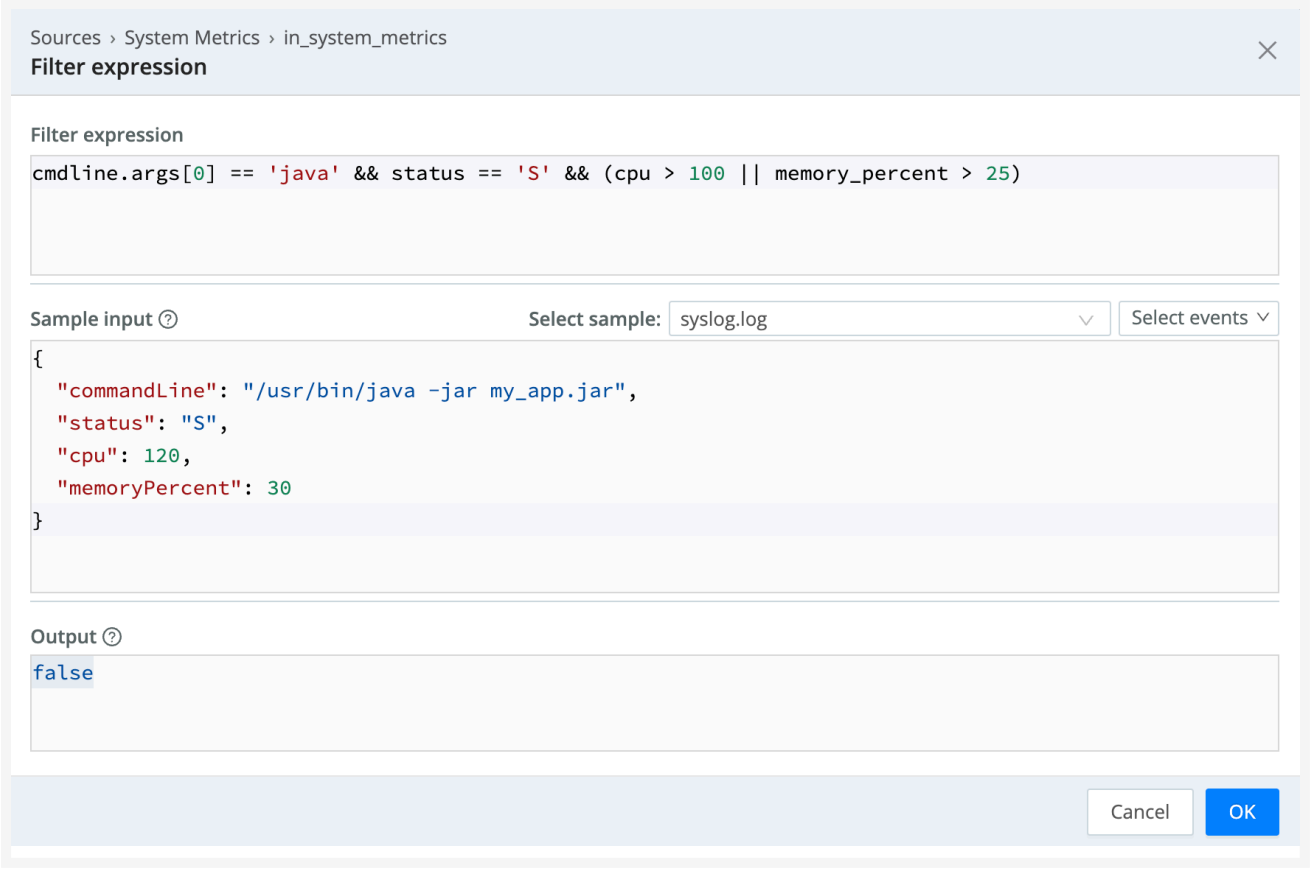
- Convert a timestamp to an hour-based value: `Math.floor(_time / 3600)`
- Mask part of a Source string: `source.replace(/.{3}/, 'XXX')`

Anywhere a Function lets you define a value with JavaScript, you can use a value expression to apply custom logic based on the content of the event.

Test and Validate Expressions in Advanced Mode

To test your filter expression against sample input data, click the **Advanced mode** button within the **Filter expression** field. The **Filter expression** modal will open.

Here, you can paste in your JavaScript expression to filter the data. You can also select a sample JSON input from the drop-down or enter your own sample data directly in the **Sample input** field.



The **Advanced mode** window

The **Output** shows you whether your expression is valid or not:

- For filter expressions, the output will resolve to `true` if it is a valid expression, which means that it can correctly apply the custom filter to the sample data. A value of `false` means you need to fix the syntax of the expression or ensure it applies to the sample data.
- For value expressions, the output will show you if the filter expression returns any matching processes, based on the filter expressions and JSON input.

In most cases, you can use Advanced Mode to test and validate JavaScript that uses Cribl Expressions as well.

If you are testing an expression that uses the Lookup Cribl Expression (`C.Lookup`), be aware of lookup limitations. To avoid overloading your web browser, lookup files larger than 10 MB are not loaded and will not evaluate in Advanced Mode. However, if the expression is valid, the updated values are still correctly applied to route and process data.

Best Practices to Create Predictable Expressions

- In a value expression, ensure that the source variable is not `null`, `undefined`, or `empty`. For example, assume you want to have a field called `len`, to be assigned the length of a second field called `employeeID`. But you're not sure if `employeeID` exists. Instead of `employeeID.length`, you can use a safer shorthand, such as: `(employeeID || '').length`.
- If a field does not exist (undefined), and you're doing a comparison with its properties, then the boolean expression will **always** evaluate to false. For example, if `employeeID` is undefined, then both of these expressions will evaluate to false: `employeeID.length > 10` , and `employeeID.length < 10` .
- `==` means "equal to," while `===` means "equal value **and** equal type." For example, `5 == 5` and `5 == "5"` each evaluate to **true**, while `5 === "5"` evaluates to **false**.
- A ternary operator is a very powerful way to create conditional values. For example, if you wanted to assign either `minor` or `adult` to a field `groupAge`, based on the value of `age`, you could do: `(age >= 18) ? 'adult' : 'minor'`.

Fields with Non-Alphanumeric Characters

If there are fields whose names include non-alphanumeric characters, such as `@timestamp` or `user-agent` or `kubernetes.namespace_name`, you can access them using `__e['<field-name-here>']`. (Note the single quotes.) More details are available on the Mozilla Developer Docs topic on Property accessors..

In any other places that reference the field, such as the Eval function's field names, you should use a single-quoted literal, of the form: `'<field-name-here>'`.

Wildcard Lists

Wildcard lists appear in various Functions, such as Eval, Mask, Publish Metrics, Parser, and Serialize.

Wildcard lists accept strings with asterisks (`*`) to represent one or more terms. They also accept strings that start with an exclamation mark (`!`) to **negate** one or more terms. This allows for implementing any combination of allowlists and blocklists.

Wildcard fields (`*`) support two use cases:

- Wildcards (`*`) to include multiple fields.
- Nested addressing, where you can reference a parent field and all its children using a wildcard. For example, if you convert `_raw` into an object, use `_raw*` to refer to `_raw` and all nested fields inside it.

Wildcard lists are order-sensitive, evaluated from left to right. This is especially relevant when you use negated terms. For negations to take precedence over wildcards when evaluated, you must list negations before wildcards.

Some examples:

Wildcard List	Value	Meaning
List 1	`!foobar, foo*`	All terms that start with **foo**, except **foobar**.
List 2	`!foo*, *`	All terms, except for those that start with **foo**.
List 3	`*, !foo`	All terms (wildcard matches first, negation isn't evaluated).

You cannot use wildcards to target Cribl Stream internal fields that start with `__` (double underscore). You must specify these fields individually. For example, `__foobartab` cannot be removed by specifying `__foo*`.

6.7.2. Code Function Examples

The Code Function is a powerful way to transform events without needing to write a custom function. Using JavaScript methods such as `map`, `reduce`, `forEach`, `some`, and `every` is possible.

Cribl still recommends that you use Cribl Stream's basic, built-in Functions, like Eval, as much as possible. But these use cases demonstrate some basic ways to use the new Code Function to solve questions asked in the Cribl's Slack Community.

Basic Examples

Example Event Data

The first several examples below use the following JSON object as a sample event. You can copy and paste this event into Cribl Stream's **Sample Data** pane. Add the `Do Not Break` Ruleset to your Source, select the `noBreak1MB` Event Breaker, and under your Source's **Advanced Settings**, enable `Parse JSON Event`.

```
{
  "cpus": [
    {"number": 1, "name": "CPU1", "value": 2.3},
    {"number": 2, "name": "CPU2", "value": 3.1},
    {"number": 3, "name": "CPU3", "value": 5.1},
    {"number": 4, "name": "CPU4", "value": 1.3}
  ],
  "arch": "Intel x64"
}
```

Copy

Accessing Fields in an Event

To access a field inside an event, you can use the `__e` special variable. The `__e` prefix allows for access to the `(context)` event inside a JavaScript expression. For example, if you want to access the extracted field `field-name`, use the following syntax:

```
__e['field-name']
```

Copy

In other words, think of your code executing in a context like this:

```
function(__e: Event) {
 // your code here
}
```

Copy

Eval a Field

To create a new field, it is as simple as assigning the field to a value. For example, if you want to create a field `test` with the value `Hello, Goats!`, use the following syntax:

```
__e['test'] = 'Hello, Goats!'
```

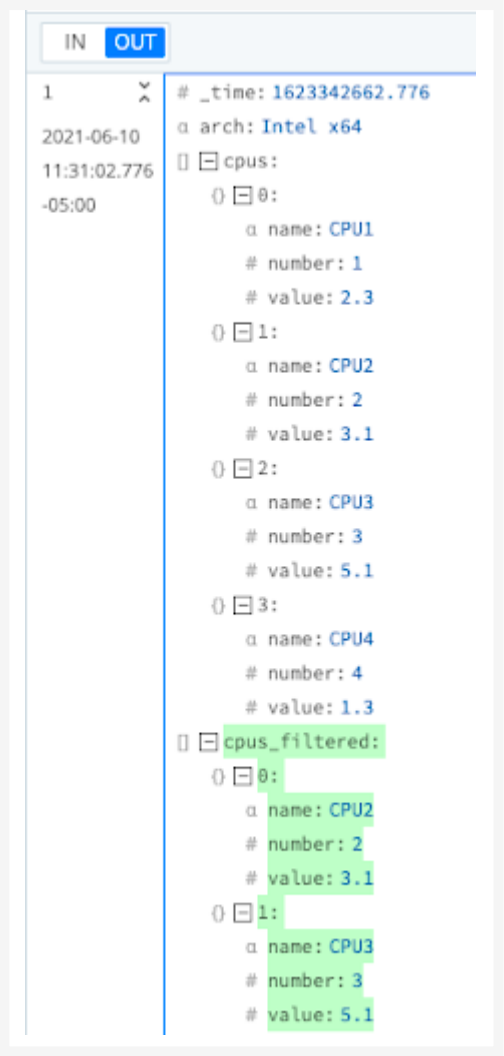
Copy

JSON Filter

To filter the `cpus` array inside the event, you can use the `filter` function to keep only certain values, based on a logic condition. In the example below, only entries where the value is greater than or equal to `3` are kept, and placed into a new field called `cpus_filtered`.

```
__e['cpus_filtered'] = __e['cpus'].filter(entry => entry.value >= 3)
```

Copy



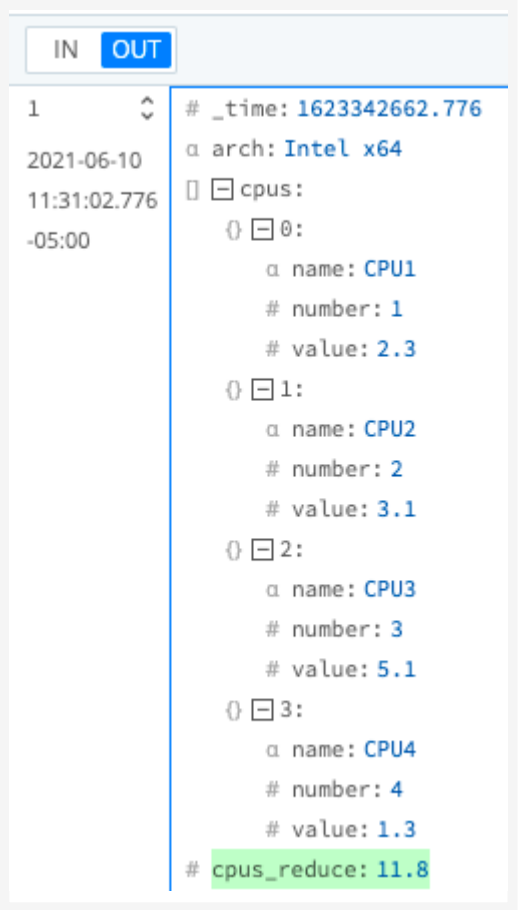
Code Function example: JSON Filter

Reduce

The `reduce` method allows you to summarize data across an array, with a returned accumulator value. In the example below, a new field, `cpus_reduce`, will be created with a value of `11.8`.

```
__e['cpus_reduce'] = __e['cpus'].reduce((accumulator, entry) => accumulator + entry.v
```

Copy



Code Function example: Reduce

Some/Every

The `some` and `every` methods return a boolean result (`true`/`false`).

The `some` method checks that there is at least one logic validation that returns `true`. In the example below, `cpus_some` would be set to `true` because there is at least one object with a value greater than or equal to `3`.

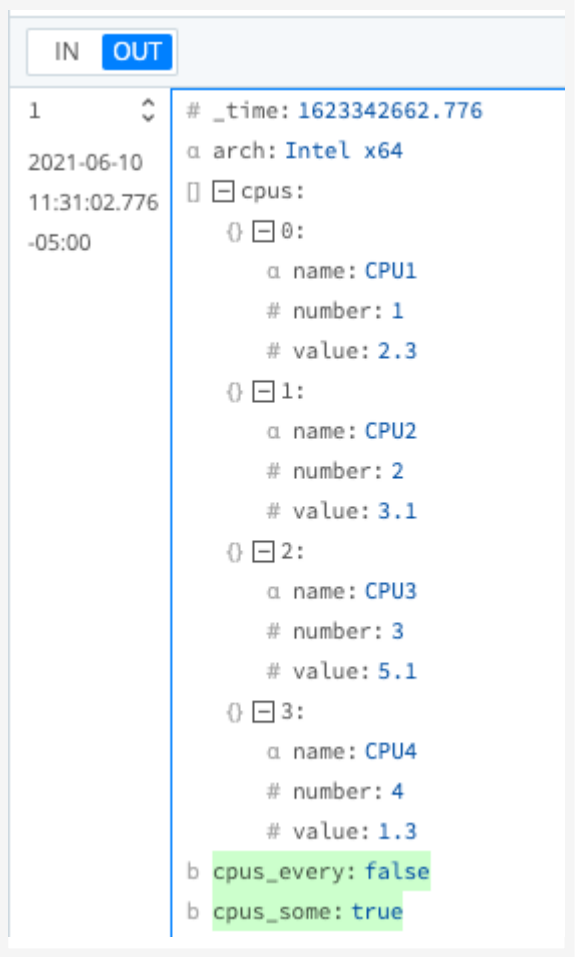
```
__e['cpus_some'] = __e['cpus'].some(entry => entry.value >= 3)
```

Copy

The `every` method checks that every entry in the array returns `true`. If so, the result is `true`; otherwise, this returns `false`. In the example below, the value for `cpus_every` would be `false`, because not all values in the event are greater than or equal to `10`.

```
__e['cpus_every'] = __e['cpus'].every(entry => entry.value >= 10)
```

Copy



Code Function example: Some/Every

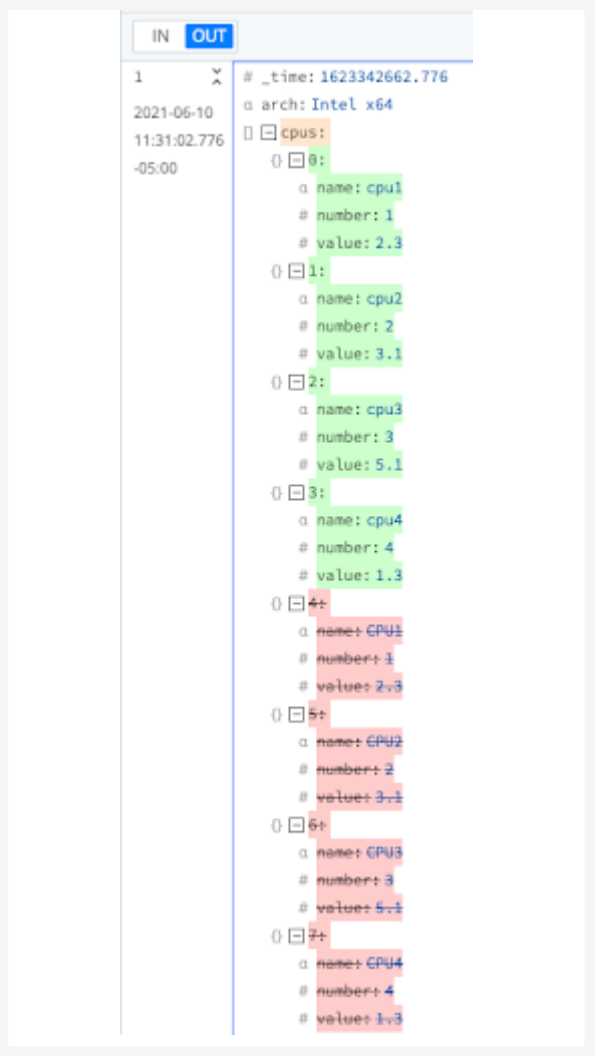
Advanced Examples

Transform a Specific Field

In this example, each `cpus` member will have its `name` field transformed to lowercase. Object.assign is used to keep the original object, while assigning the `name` field to the desired value.

```
__e['cpus'] = __e['cpus'].map(entry => Object.assign(entry, {'name': entry.name.toLow
```

Copy



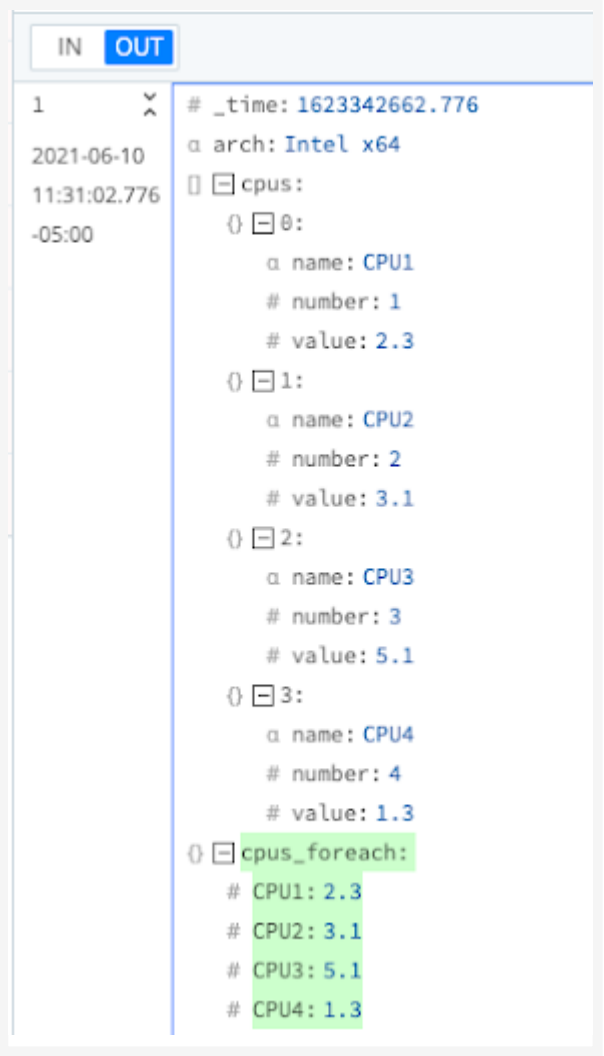
Code Function example: Transform a Field

forEach

The `forEach` method loops over each element of an array. However, unlike the `map` method, it does not return the value, and it requires a separate temporary variable for result collection.

```
let test = {};
__e['cpus'].forEach((entry, index) => test[entry.name] = entry.value);
__e['cpus_foreach'] = test;
```

Copy



Code Function example: forEach

You could also accomplish the same result by replacing the middle line above with the `map` method below:

```
__e['cpus'].map(item => test[item.name] = item.value);
```

Copy

Building a New Array

In this example, we create a new array to include some of the original values from each object in the `cpus` array, but we also dynamically inject a new field containing the `arch` field (CPU architecture) from the original event's top level.

```
__e['cpus_new'] = __e['cpus'].map(entry => {
    const container = {};

   for (const [key, value] of Object.entries(entry)) {
        container[key] = value;
    }

   container['arch'] = __e['arch'];

   delete container.name;
    return container;
})
```

Copy

Code Function example: Build a new array

Managing and Troubleshooting Code Execution

Cribl Stream provides the following options for tuning the logic you build with the Code Function.

Logging

Users can access the log messages generated by their Function from the **Preview Log**.

Log messages from a Code Function

Debugging

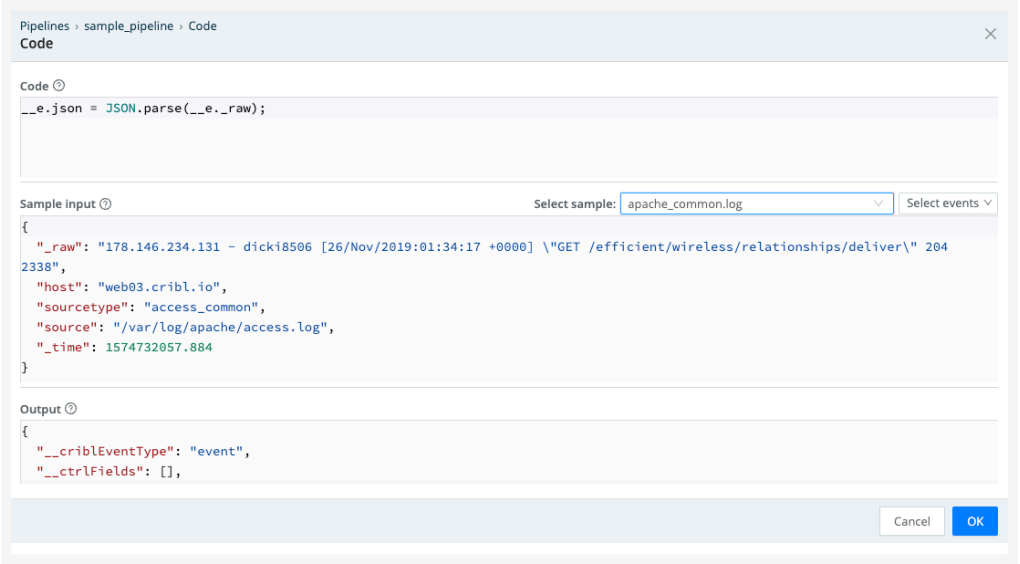
The Code Function's execution context defines a helper function called `debug` that can be used for debugging purposes.

Messages logged by this `debug` helper function are shown in the Preview Log by default.

You can also route these messages to regular logs, although this requires setting the Function's log level (`func:code`) to `debug`.

Previewing

By using the expanded editor, users can run the expression against a sample event, and can preview the transformation:



Previewing a Code Function’s transformation of an event

6.7.3. Ingest-Time Fields

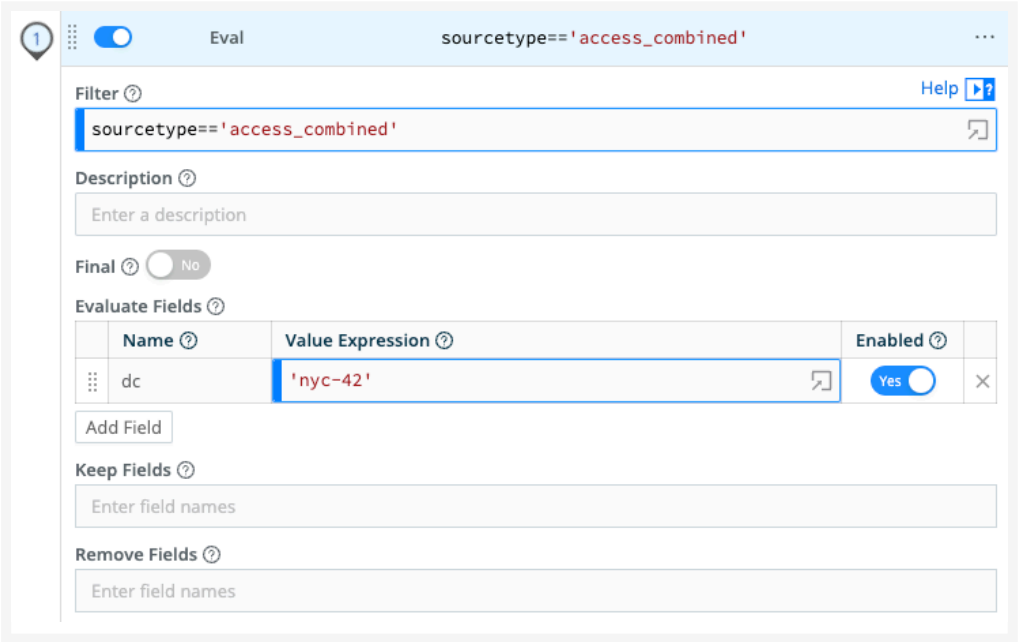
Adding Fields to Data in Motion

To add new fields to any event, we use the out-of-the-box **Eval** Function. We can either apply a Filter to select the events, or we can use the default `true` Filter expression to apply the Function to all incoming events.

Adding Fields Example

Let's see how we add `dc::nyc-42` to all events with `sourcetype=='access_combined'`:

- First make sure you have a Route and Pipeline configured to match desired events.
- Next, let's add an **Eval** function to it with `sourcetype=='access_combined'` as Filter.
- Next, let's click on **Add Field**, add our `dc` field, and click **Save**.



Adding the `dc` field

To confirm, verify that this search returns results: `sourcetype="access_combined" dc::nyc-42`

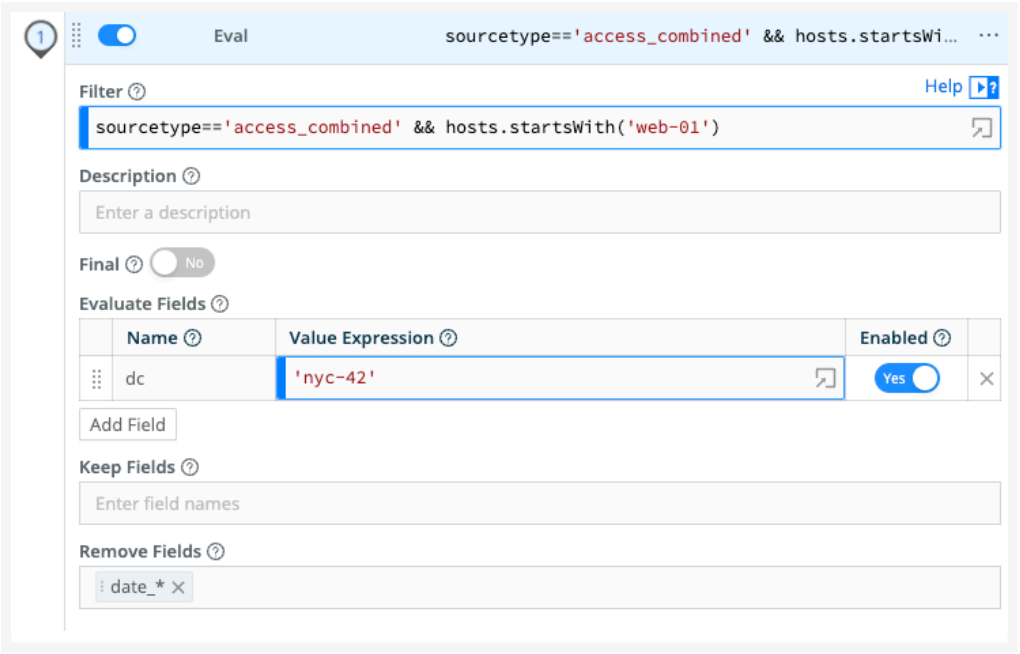
- You can add more conditions to the filter, if you'd like. For example, to limit the field to only events from hosts that start with `web-01`, we can change the filter input to `sourcetype=='access_combined' && hosts.startsWith('web-01')`.

This is a **very** powerful method to change incoming events in real time. In addition to providing the right context at the right time, users can further benefit substantially by using `tstats` for **faster** analytics.

Removing Fields

You can remove fields by listing and/or wildcarding field names. Let's see how we can remove all fields that start with `date_`.:

- First, make sure you have a Route and Pipeline configured to match desired events.
- Next, let's add an **Eval** function to it (as above).
- Next, in **Remove Fields**, add `date_*` and hit Save.



Goodbye `date_` field

To confirm, verify that this search: `sourcetype="access_combined" date_minute=*` will soon stop returning results. Enjoy a more efficient Splunk!

6.7.4. Masking And Obfuscation

Masking and Anonymization of Data in Motion

To mask patterns in real time, we use the out-of-the-box Mask Function . This is similar to `sed`, but with much more powerful functionality.

Masking Capabilities

The Mask Function accepts multiple replacement rules, and accepts multiple fields to apply them to.

Match Regex is a JS regex pattern that describes the content to be replaced. It can optionally contain matching groups. By default, it will stop after the first match, but using `/g` will make the Function replace all matches.

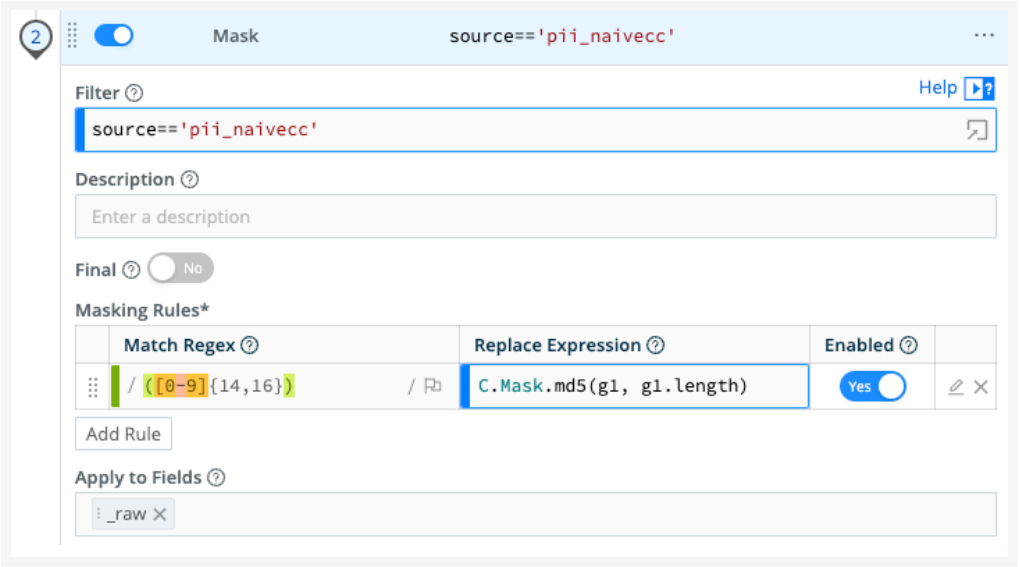
Replace Expression is a JS expression or literal to replace matched content.

Matching groups can be referenced in the **Replace Expression** as `g1`, `g2`... `gN`, and the entire match as `g0`.

There are several masking methods that are available under `C.Mask.`:

`C.Mask.random`: Generates a random alphanumeric string `C.Mask.repeat`: Generates a repeating char/string pattern, e.g., XXXX `C.Mask.REDACTED`: The literal 'REDACTED' `C.Mask.md5`: Generates a MD5 hash of given value `C.Mask.sha1`: Generates a SHA1 hash of given value `C.Mask.sha256`: Generates a SHA256 hash of given value

Almost all methods have an optional `len` parameter which can be used to control the length of the replacement. `len` can be either a number or string. If it's a string, its length will be used. For example:



Masking Examples

Let's look at the various ways that we can mask a string like this one: `cardNumber=214992458870391`. The **Regex Match** we'll use is: `/(cardNumber=)(\d+)/g`. In this example:

- `g0 = cardNumber=214992458870391`
- `g1 = cardNumber=`
- `g2 = 214992458870391`

Random Masking with default character length (4):

- **Replace Expression**: `` `${g1}${C.Mask.random()}` ``
- **Result**: `cardNumber=HRhc`

Random Masking with defined character length:

- **Replace Expression**: `` `${g1}${C.Mask.random(7)}` ``
- **Result**: `cardNumber=neNSm8r`

Random Masking with length preserving replacement:

- **Replace Expression**: `` `${g1}${C.Mask.random(g2)}` ``
- **Result**: `cardNumber=DroJ73qmyaro51u3`

Repeat Masking with default character length (4):

- **Replace Expression**: `` `${g1}${C.Mask.repeat()}` ``
- **Result**: **Result**: `cardNumber=XXXX`

Repeat Masking with defined character choice and length:

- **Replace Expression**: `` `${g1}${C.Mask.repeat(6, 'Y')}` ``
- **Result**: `cardNumber=YYYYYY`

Repeat Masking with length preserving replacement:

- **Replace Expression**: `` `${g1}${C.Mask.repeat(g2)}` ``
- **Result**: `cardNumber=XXXXXXXXXXXXXXX`

Literal REDACTED masking:

- **Replace Expression**: `` `${g1}${C.Mask.REDACTED}` ``
- **Result**: `cardNumber=REDACTED`

Hash Masking (applies to: md5, sha1 and sha256):

- **Replace Expression**: `` `${g1}${C.Mask.md5(g2)}` ``
- **Result**: `cardNumber=f5952ec7e6da54579e6d76feb7b0d01f`

Hash Masking with left N-length* substring (applies to: md5, sha1 and sha256):

- **Replace Expression**: `` `${g1}${C.Mask.md5(g2, 12)}` ``
- **Result**: `cardNumber=d65a3ddb2749` *Replacement length will **not** exceed that of the hash algorithm output; MD5: 32 chars, SHA1: 40 chars, SHA256: 64 chars.

Hash Masking with right N-length* substring (applies to: md5, sha1 and sha256):

- **Replace Expression**: `` `${g1}${C.Mask.md5(g2, -12)}` ``
- **Result**: `cardNumber= 933bfcebf992` *Replacement length will **not** exceed that of the hash algorithm output; MD5: 32 chars, SHA1: 40 chars, SHA256: 64 chars.

Hash Masking with length* preserving replacement (applies to: md5, sha1 and sha256):

- **Replace Expression**: `` `${g1}${C.Mask.md5(g2, g2)}` ``
- **Result**: `cardNumber= d65a3ddb27493f5` *Replacement length will **not** exceed that of the hash algorithm output; MD5: 32 chars, SHA1: 40 chars, SHA256: 64 chars.

6.7.5. Encryption Of Data In Motion

This page covers creating, modifying, and synchronizing symmetric keys.

Cribl Stream enables you to encrypt fields or patterns within events in real time, by using a `C.Crypto.encrypt()` expression in a Mask Function. The Mask Function accepts multiple replacement rules, and multiple fields to apply them to.

In that Function's configuration, a **Match Regex** defines the pattern of content to be replaced. The **Replace Expression** is a JavaScript expression or literal to replace matched content. You can use the `C.Crypto.encrypt()` method here to generate an encrypted string from a passed value.

Encryption Keys

You can configure symmetric keys through the CLI or UI. Users are free to define as many keys as required. Each key is characterized by the following:

- `keyId`: ID of the key.
- `group`: Worker Group associated with the encryption key.
- `algorithm`: Algorithm used with the key
- `keyclass`: Cribl Key Class (below) that the key belongs to.
- `kms`: Key management system for the key. Defaults to `local`.
- `created`: Time (epoch) when key was generated.
- `expires`: Time (epoch) after which the key is invalid. Useful for key rotation.
- `useIV`: Flag that indicates whether or not an initialization vector was used.

Key Classes

Key Classes in Cribl Stream are collections of keys that can be used to implement multiple levels of access control. Users (or groups of users) with access to data with encrypted patterns can be associated with key classes, for even more granular, pattern-level compartmentalized access.

Example

Users `U0, U1` have been given access to keyclass `0` which contains key IDs `0` and `1`. These keys are used to encrypt certain patterns in `datasetA`. Even though users `U0, U1, U2` have access to read this dataset, only `U0` and `U1` can decrypt its encrypted patterns.

Key Class	Dataset
`keyclass: 0` Keys: `keyId: 0, keyId: 1` Users: `U0, U1`	`datasetA` Users: `U0, U1, U2`

User `U1` has been given access to an **additional** keyclass, `1`, which contains key IDs `11` and `22`. These keys are used to encrypt certain **other** patterns in `datasetA`. Even though users `U0, U1, U2` have access to read this dataset - same as above - only `U1` can decrypt the additional encrypted patterns.

Key Class	Dataset
`keyclass: 1` Keys: `keyId: 11, keyId: 22` Users: `U1`	`datasetA` Users: `U0, U1, U2`

Configure Keys

- **UI**: See how to create and manage encryption keys in the UI.
- **Command Line**: Use the `keys` command.

Key Storage

When using the `local` key management system, encryption keys in Cribl Stream are encrypted with `$CRIBL_HOME/local/cribl/auth/cribl.secret` and stored in `$CRIBL_HOME/local/cribl/auth/keys.json`. Cribl monitors the `keys.json` file for changes every 60 seconds.

When installed as a Splunk app, `$CRIBL_HOME` is `$SPLUNK_HOME/etc/apps/cribl`.

6.7.6. Reducing Windows XML Events

Here, we demonstrate how to use just a few Cribl Stream Functions to parse WindowsXML events and reduce their volume by 34-70%, dramatically reducing your downstream infrastructure requirements.

Using Eval and C.Text.parseWinEvent

Cribl Stream's internal C.Text.parseWinEvent method parses Windows XML strings and returns a prettified JSON object. You can use this function within an Eval Function to parse an event or an ad hoc XML string. It works like C.Text.parseXml, but with Windows events, it produces more-compact output.

As you can see from its signature, `C.Text.parseWinEvent` accepts an optional `nonValues` parameter that can further reduce an event's size by discarding characters that you specify as redundant.

(method) `C.Text.parseWinEvent(xml: string, nonValues?: string[]): any`

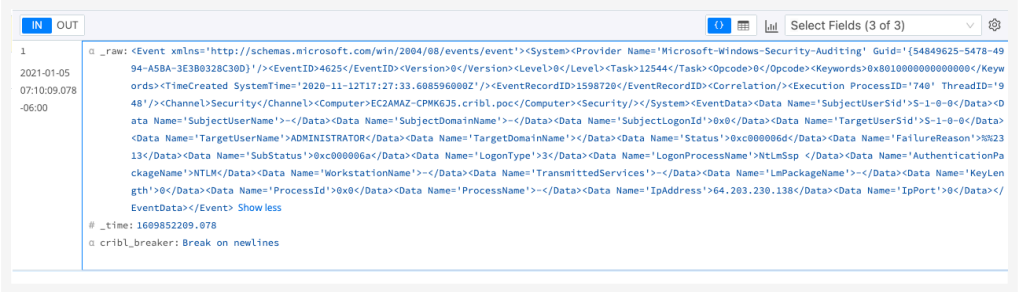
@param - `xml` - an XML string; or an event field containing the XML.

@param - `nonValues` - array of string values. Elements whose value equals any of the values in this array will be omitted from the returned object. Defaults to `['-']`, meaning that elements whose value equals `-` will be discarded.

@returns - an object representing the parsed Windows Event; or `undefined` if the input could not be parsed.

XML: Threat or Menace?

When working with XML, an anonymous Reddit user's quote sums up the challenge: "Some languages can be read by humans, but not by machines, while others can be read by machines but not by humans. XML solves this problem by being readable to neither." An example of a Windows XML event only reinforces this quote:



This Windows XML `_raw` event is 1.36KB in size:

	_raw Length	Full Event Length	Number of Fields	Number of Events
IN	1.36KB	1.43KB	3	1
OUT	1.36KB	1.47KB	4	1
DIFF	0.00%	↑ 2.73%	↑ 33.33%	0.00%

Eval to the Rescue

In our initial Eval Function below, the **Value Expression** uses `C.Text.parseWinEvent` to simply parse the `_raw` Windows XML event and turn it into a prettified JSON object:

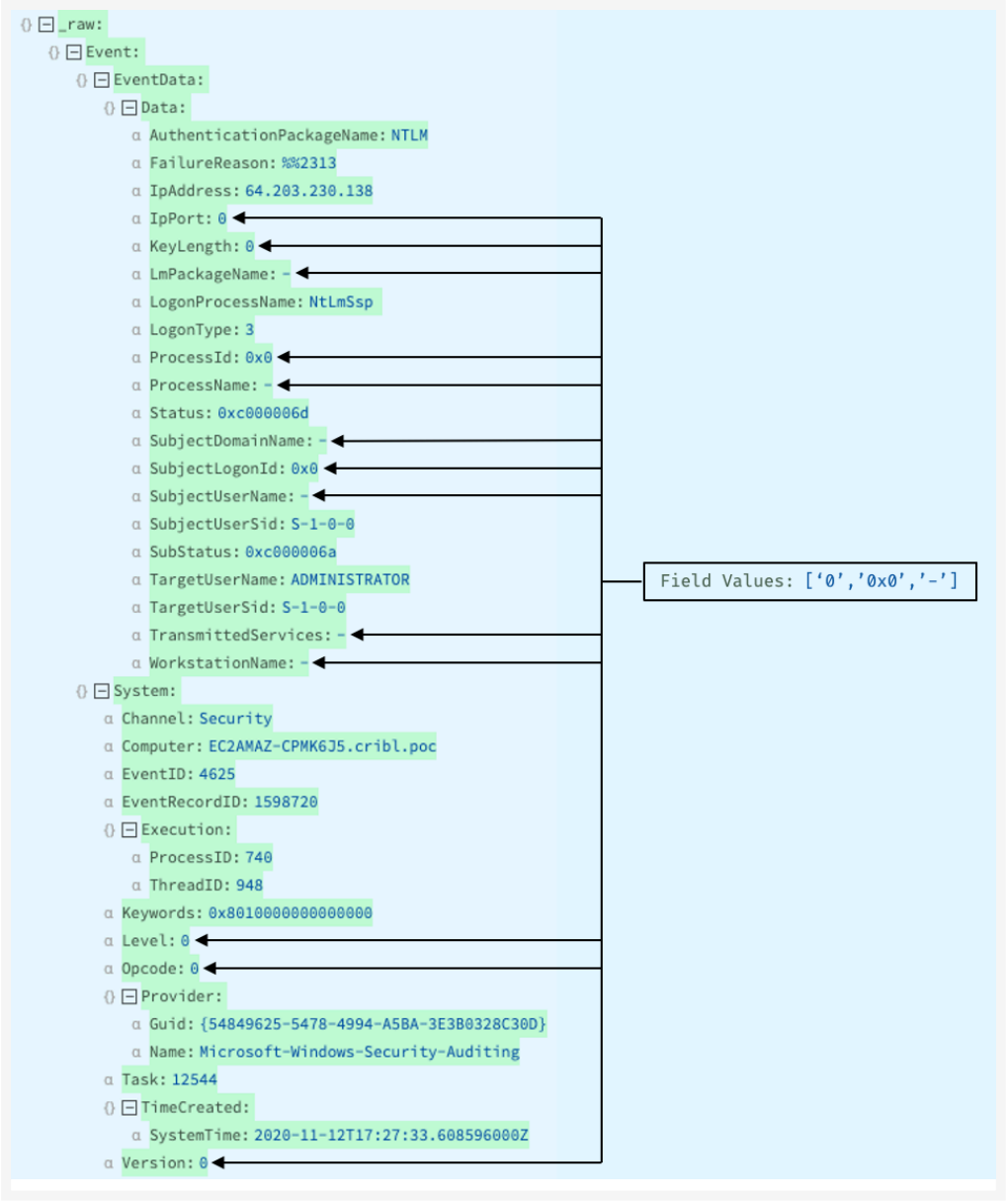
The resulting JSON event is now down to 921.00B in size, a 34.07% reduction of the event:

	_raw Length	Full Event Length	Number of Fields	Number of Events
IN	1.36KB	1.43KB	3	1
OUT	921.00B	1.00KB	4	1
DIFF	↓ -34.07%	↓ -29.76%	↑ 33.33%	0.00%

The `C.Text.parseWinEvent(_raw,[])` call yields verbose output that includes XML attributes. If you want flatter output, you can substitute `C.Text.parseXml(_raw, false)`.

Removing Unnecessary Fields with a Better Eval

But we can do better. The fields containing essentially null values (`'0'`,`'0x0'`, or `'-'`) bloat events, demanding extra infrastructure and storage:



Let's amplify the reduction by removing all of the fields whose values are in the set: `['0','0x0','-']`. This improved version of our Eval Function parses the Windows XML event, and discards `['0','0x0','-']` values. (Its preceding row also tidies up events by removing tabs and curly braces, and replacing newlines and returns with commas.) The result is an even **smaller** prettified JSON object.

If you compare this Preview-pane screenshot to the Preview screenshot above, you can confirm that the fields with values matching `['0','0x0','-']` are removed:

The event is now down to 678.00B in size, translating to a 51.47% reduction from the original event:

	_raw Length ⓘ	Full Event Length ⓘ	Number of Fields ⓘ	Number of Events ⓘ
IN	1.36KB	1.43KB	3	1
OUT	678.00B	785.00B	4	1
DIFF	↓ -51.47%	↓ -46.42%	↑ 33.33%	0.00%

Flatten Function

Cribl Stream's Flatten Function is designed to flatten fields out of a nested structure. Let's flatten the JSON object within `_raw`, to see if we can further reduce the event's size before we send it to our preferred

destinations.

Using Flatten, we've successfully created top-level fields from the nested JSON structure:

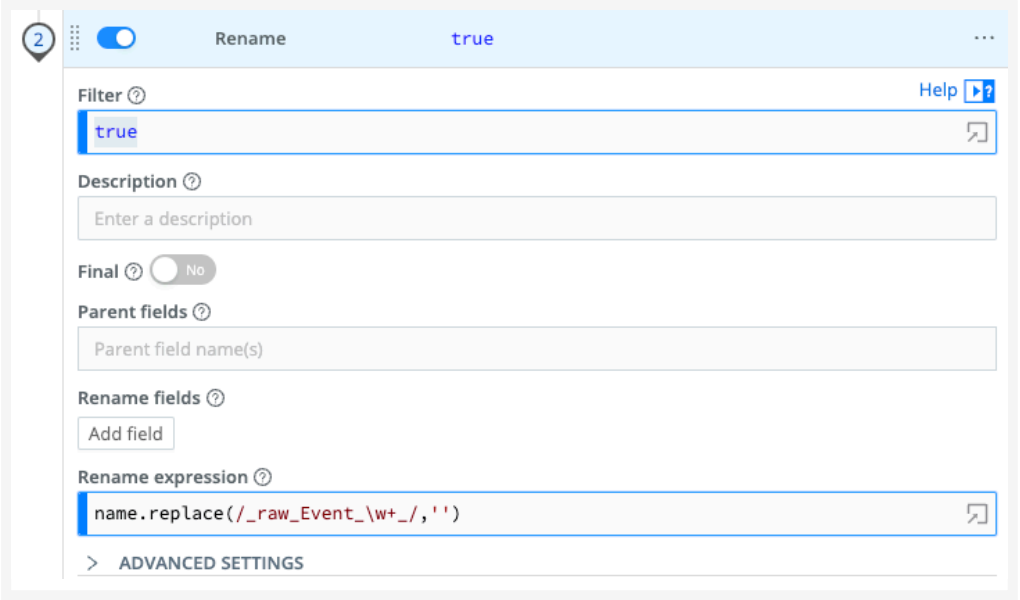
Don't worry about the `_raw` field's deletion (red strikeout). This is the Flatten Function's default behavior. We'll restore `_raw` after we clean and reduce the event even more.

The flattened field names are extracted from `_raw` and delimited with `_`. These field names are quite long. We can optimize them using the Rename Function.

Rename Function

Rename is designed to change fields' names, or to reformat their names (e.g., to normalize names to camelcase). You can use Rename to change specified fields (much like Eval), or to accomplish bulk renaming based on a JavaScript expression (much like the Parser Function). But Rename offers a streamlined way to alter only field names, without other effects.

Let's use Rename to remove any unnecessary prefixes from the field names, to further shrink our events. In the **Rename Expression**, we build a JavaScript expression to match the field names' prefixes (up to the underscore):



The resulting field names are now much more compact, and easier to work with and manage:

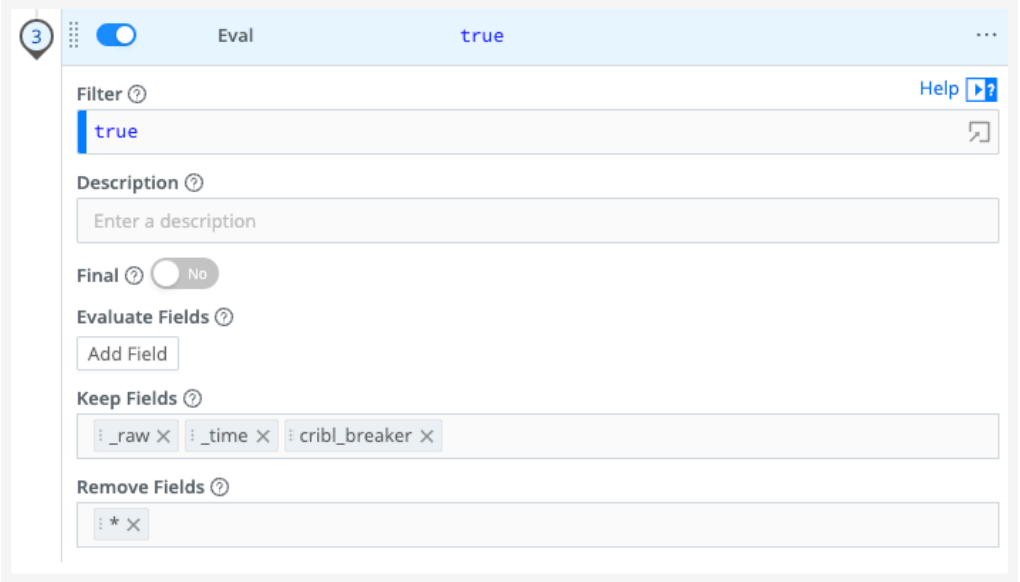
Serialize Function

We started with bloated Windows XML data, and we've parsed and prettified it into JSON. Next, we'll extract key-value pairs. We'll use the Serialize Function, which serializes an event's content into a predefined format.

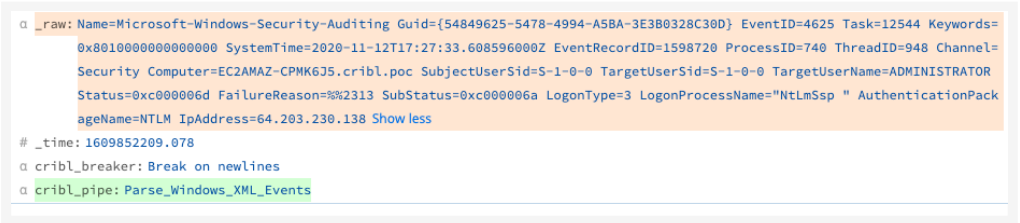
We set Serialize to change the **Type** to key-value pairs. (The Function's other supported target Types include JSON Object and CSV.) Serialize takes the extracted fields and puts them back into `_raw`.

In the Preview pane, the `_raw` field is now back, serialized into compact, tidy key-value pairs:

The last step is to remove any of the extracted fields you don't need before sending events to your destinations. We'll again call on the Eval Function, which adds or removes fields in events. (For a Splunk destination, these are index-time fields.) This final Eval Function looks like this:



To sum up, we've successfully transformed the original Windows XML event into key-value pairs:



And we've dramatically reduced the event's size, while retaining all of the necessary fields. The event is now down to 513.00B in size, translating to a 63.28% reduction from the original Windows XML:

	_raw Length	Full Event Length	Number of Fields	Number of Events
IN	1.36KB	1.43KB	3	1
OUT	513.00B	621.00B	4	1
DIFF	↓ -63.28%	↓ -57.61%	↑ 33.33%	0.00%

Try This at Home

Below is an export of the whole Cribl Stream Pipeline presented here. Import this JSON to experiment with it and modify it to match your own needs:

Win XML Pipeline

```
{
  "id": "Windows_Security_Events",
  "conf": {
    "output": "default",
    "groups": {},
    "asyncFuncTimeout": 1000,
    "functions": [
      {
        "filter": "true",
        "conf": {
          "comment": "This Cribl Stream Pipeline reduces Microsoft Windows XML events
        },
        "id": "comment"
      },
      {
        "filter": "true",
        "conf": {
          "add": [
            {
              "name": "_raw",
              "value": "_raw.replace(/[{}\\t]/gm,'').replace(/[\\n\\r]+/gm,',')"
            },
            {
              "name": "_raw",
              "value": "C.Text.parseWinEvent(_raw,['0x0','0','-'])"
            }
          ]
        },
        "id": "eval",
        "disabled": false,
        "description": "Remove tabs & curly braces; replace newlines & returns with c
      },
      {
        "filter": "true",
        "conf": {
          "fields": [
            "_raw"
          ],
          "prefix": "",
          "depth": 5,
          "delimiter": "_"
        },
```

```
  "id": "flatten",
  "disabled": false,
  "description": "Flatten the object into key value fields"
},
{
  "filter": "true",
  "conf": {
    "baseFields": [],
    "renameExpr": "name.replace(/_raw_Event_\\w+_/,'')",
    "rename": []
  },
  "id": "rename",
  "disabled": false,
  "description": "Rename the top level fields and remove the log suffix"
},
{
  "filter": "true",
  "conf": {
    "type": "kvp",
    "fields": [
      "!_*",
      "!cribl_*",
      "!index",
      "!host",
      "!source",
      "!sourcetype",
      "*"
    ],
    "dstField": "_raw",
```

6.7.7. Regex Filtering

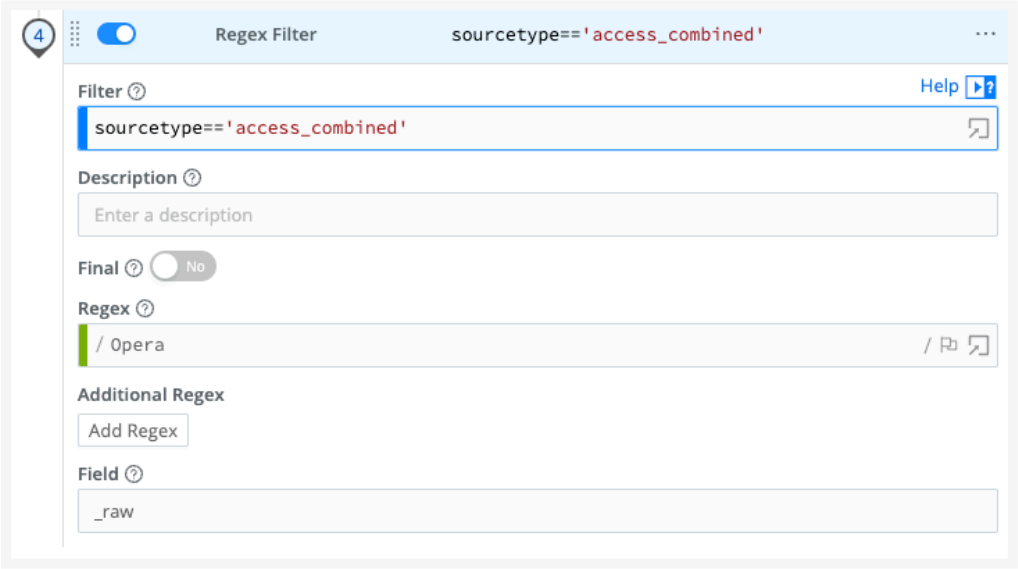
To filter events in real time (data in motion), we use the out-of-the-box **Regex Filter** Function. This is similar to nullqueueing with TRANSFORMS in Splunk, but the matching condition is way more flexible.

Regex Filtering Example

Let's see how we can filter out any `sourcetype=='access_combined'` events whose `_raw` field contains the pattern `Opera`:

First, make sure you have a Route and Pipeline configured to match desired events.

Next, let's add a **Regex Filter** Function to it:



Defining the Regex Filter Function

Next, verify that this search does **not** return any results: `sourcetype="access_combined" Opera`

You can add more conditions to the Filter input field. For example, to further limit the filtering to only events from hosts with domain `dnto.ca`, change the filter input to `sourcetype=='access_combined' && host.endsWith('dnto.ca')`.

This is a very flexible method for filtering incoming events in real time, on virtually any arbitrary conditions.

7. Functions

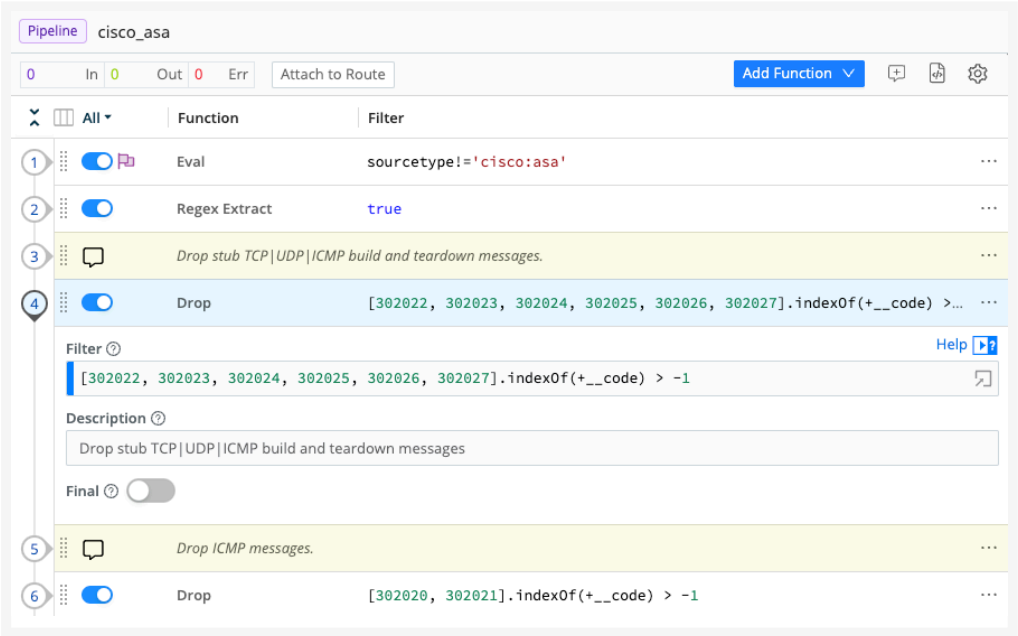
When events enter a Pipeline, they're processed by a series of Functions. At its core, a Function is code that executes on an event, and it encapsulates the smallest amount of processing that can happen to that event.

The term "processing" means a variety of possible options: string replacement, obfuscation, encryption, event-to-metrics conversions, and so on. For example, a Pipeline can be composed of several Functions - one that replaces the term `foo` with `bar`, another one that hashes `bar`, and a final one that adds a field (say, `dc=jfk-42`) to any event that matches `source=='us-nyc-application.log'`.

How Do They Work

Functions are atomic pieces of JavaScript code that are invoked on each event that passes through them. To help improve performance, configure Functions with filters to further scope their invocation to matching events only. See Build Custom Logic to Route and Process Your Data: Filter Expressions for more information.

You can add as many Functions in a Pipeline as necessary, though the more you have, the longer it will take each event to pass through. Also, you can enable and disable Functions within a Pipeline as necessary. This lets you preserve structure as you optimize or debug.



Functions stack in a Pipeline

You can reposition Functions up or down the Pipeline stack to adjust their execution order. Use a Function's left grab handle to drag and drop it into place.

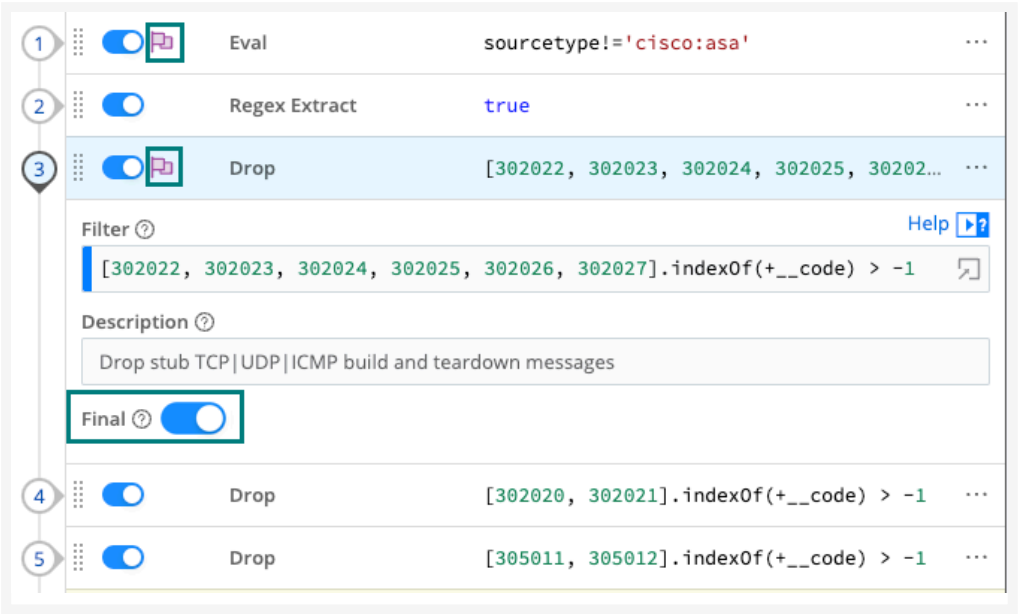
The Final Toggle

The **Final** toggle in Function settings controls what happens to the results of a Function.

When **Final** is toggled off (default), events will be processed by this Function, **and** then passed on to the next Function below.

When **Final** is toggled on, the Function "consumes" the results, meaning they will not pass down to any other Function below.

A flag in Pipeline view indicates that a Function is **Final**.



The Eval and first Drop Functions bearing the Final flag

Functions and Shared-Nothing Architecture

Cribl Stream is built on a shared-nothing architecture, where each Node and its **Worker Processes operate separately, and process events independently of each other**. This means that all Functions operate strictly in a Worker Process context - state is not shared across processes.

For example, consider two events that meet a Pipeline's criteria to be aggregated. If these two events arrive on **separate** Workers or Worker Processes, Cribl Stream will **not** aggregate them together.

This is particularly important to understand for certain Functions that might imply state-sharing, such as Aggregations, Rollup Metrics, Dynamic Sampling, Sampling, Suppress, and so on.

If you have a large number of Worker Processes, consider implementing a distributed caching tier, such as Redis, to aggregate events across Workers. (See our Redis Function topic.)

Out-of-the-Box Functions

Cribl Stream ships with several Functions out-of-the-box, and you can chain them together to meet your requirements. For more details, see individual **Functions**, and the **Use Cases** section, within this

documentation.

Accessing Event Fields with `__e`

The special variable `__e` represents the `(context)` event inside a JavaScript expression. Using `__e` with square bracket notation, you can access any field within the event object, for example, `__e['hostname']`. Make sure to use single quotes around the field name. Functions use `__e` extensively.

You also **must** use this notation for fields that contain a special (non-alphanumeric) character like `user-agent`, `kubernetes.namespace_name`, or `@timestamp`. For more details, see MDN's Property Accessors documentation. In any other place where such fields are referenced - for example, in an Eval Function's Field names - you should use a single-quoted literal, of the form: `'<field-name-here>'`.

Supported JavaScript Standard

Cribl Stream supports the ECMAScript® 2015 Language Specification.

Very Large Integer Values

Cribl Stream's JavaScript implementation can safely represent integers only up to the Number.MAX_SAFE_INTEGER constant of about 9 quadrillion (precisely, {2^53}-1). Cribl Stream Functions will round down any integer larger than this, in Data Preview and other contexts. Trailing 0's might indicate such rounding down of large integers.

Specify Fields' Precedence Order in Expressions

For Sources that support adding **Fields (Metadata)**, you can use a **Value** expression to allow fields in events to override the predefined field values. For example, the following expression's left-to-right OR logic ensures that if an inbound event includes an `index` field, its value is used: `${__e['index'] || 'myIndex'}`.

If no `index` field is present, the expression falls back to the constant `myIndex` defined in the expression.

What Functions to Use When

- Add, remove, update fields: Eval, Lookup, Regex Extract
- Find & Replace, including basic `sed`-like, obfuscate, redact, hash, and so on: Mask, Eval
- Add GeoIP information to events: GeoIP
- Extract fields: Regex Extract, Parser
- Extract timestamps: Auto Timestamp
- Drop events: Drop, Regex Filter, Sampling, Suppress, Dynamic Sampling

- Sample events (such as high-volume, low-value data): Sampling, Dynamic Sampling
- Suppress events (such as duplicates): Suppress
- Serialize events to CEF format (send to various SIEMs): CEF Serializer
- Serialize / change format (for example, convert JSON to CSV): Serialize
- Serialize compliant events into SNMP traps: SNMP Trap Serialize
- Convert JSON arrays into their own events: JSON Unroll, XML Unroll
- Flatten nested structures (for example, nested JSON): Flatten
- Aggregate events in real-time (statistical aggregations): Aggregations
- Convert events to metrics format: Publish Metrics, Prometheus Publisher (beta)
- Transforms dimensional metrics events into the OTLP (OpenTelemetry Protocol) format: OTLP Metrics
- Resolve hostname from IP address: Reverse DNS (beta)
- Resolve account names from Windows SID (**Cribl Edge only**): Windows SID Lookup
- Extract numeric values from event fields, converting them to type `number`: Numerify
- Send events out to a command or a local file, via `stdin`, from any point in a Pipeline: Tee
- Convert an XML event's elements into individual events: XML Unroll
- Duplicate events in the same Pipeline, with optional added fields: Clone
- Break events **within**, instead of before they reach, a Pipeline: Event Breaker
- Add a text comment within a Pipeline's UI, to label steps without changing event data: Comment

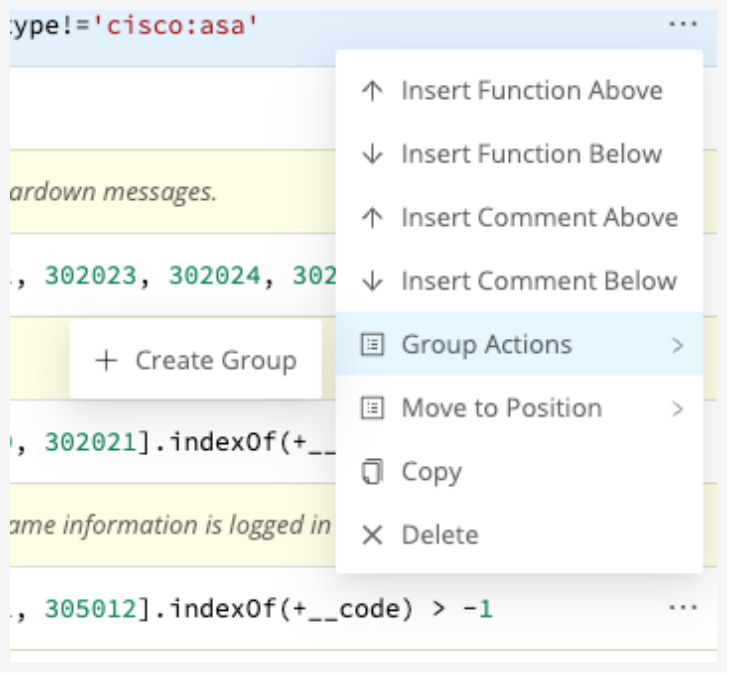
For more usage examples, download the official Tip Sheet: Cribl Stream.

Function Groups

A Function group is a collection of consecutive Functions that can be moved up and down a Pipeline's Functions stack together. Groups help you manage long stacks of Functions by streamlining their display. They are a UI visualization only, purely for visual context, and do not affect the movement of data through Functions. While Functions are in a group, those Functions maintain their global position order in the Pipeline: data moves down the listed Functions, ignoring any grouping assignments.

Function groups work much like Route groups.

To build a group from any Function, select the Function's ••• (Options) menu, then select **Group Actions** > **Create Group**.

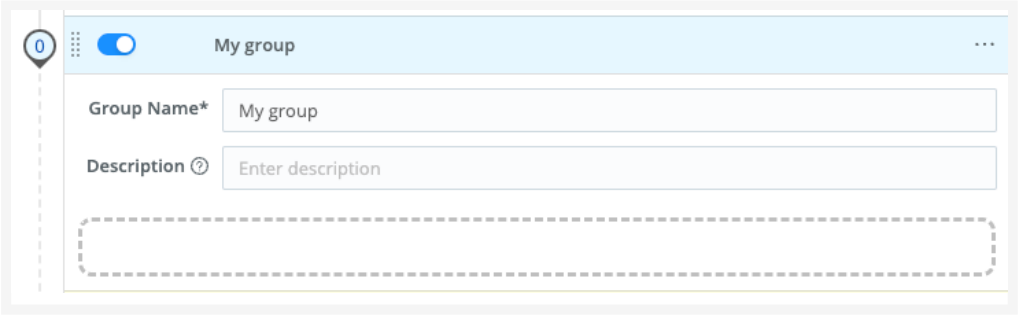


Creating a group

You'll need to enter a **Group Name** before you can save or resave the Pipeline. Optionally, enter a **Description**.

Once you've saved at least one group to a Pipeline, other Functions' ••• (Options) > **Group Actions** submenus will add options to **Move to Group** or **Ungroup/Ungroup All**.

You can also use a Function's left grab handle to drag it into, or out of, a group. A saved group that's empty displays a dashed target into which you can drag Functions.



Drag-and-drop target

7.1. Aggregations

The Aggregations Function computes summary statistics (like averages, sums, and counts) from event data.

Each Worker Process executes this Function independently on its share of events. For details, see Functions and Shared-Nothing Architecture.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Time window: The time span of the tumbling window for aggregating events. Must be a valid time string (e.g., `10s`). Must match pattern `\d+[sm]$`.

Aggregates: Aggregation function(s) to perform on events. E.g., `sum(bytes).where(action=='REJECT').as(TotalBytes)`. Expression format: `aggFunction(<FieldExpression>).where(<FilterExpression>).as(<outputField>)`. See more examples below.

When used without `as()`, the aggregate's output will be placed in a field labeled `<fieldName>_<aggFunction>`. If a conflict occurs, the last aggregate takes precedence. For example, given two aggregates - `sum(bytes).where(action=='REJECT')` and `sum(bytes)` - the latter one (`bytes_sum`) takes precedence.

Group by fields: Fields to group aggregates by. Supports wildcard expressions.

Using the wildcard character `*` causes causes all fields in the event to be included in the aggregation, which can result in high cardinality and increased memory usage. To avoid this impact, first exclude fields that can result in high cardinality, **then** introduce wildcards, if any. For example: `!_time, !_numericValue, *`

Evaluate fields: Set of key-value pairs to evaluate and add/set. Fields are added in the context of an aggregated event, before they're sent out. Does not apply to passthrough events.

Time Window Settings

Cumulative aggregations: Toggle on to retain aggregations for cumulative aggregations when flushing out an aggregation table event. Toggle off (default) to reset aggregations to `0` on flush.

Lag tolerance: The lag tolerance represents the tumbling window tolerance to late events. Must be a valid time string (e.g., `10s`). Must match pattern `\d+[sm]$`.

Idle bucket time limit: The amount of time to wait before flushing a bucket that has not received events. Must be a valid time string (e.g., `10s`). Must match pattern `\d+[sm]$`.

Output Settings

Passthrough mode: Toggle on to pass through the original events along with the aggregation events. Default is toggled off.

Metrics mode: Determines whether to output aggregates as metrics. The default is off, which outputs aggregates as events. Toggle on if you are working with metrics-heavy data.

If you toggle on **Metrics mode**, consider also toggling on **Treat dots as literals**, which is under **Advanced Settings**. This setting is strongly recommended if your metric names or dimensions use dot notation, as used in StatsD and Open Telemetry metrics formats.

Sufficient stats mode: Determines whether to output *only* statistics sufficient for the supplied aggregations. Default is toggled off, meaning output richer statistics.

Preserve group by fields: Determines whether to preserve the structure of the original aggregation event's group by fields.

Output prefix: A prefix that is prepended to all of the fields output by this Aggregations Function.

Advanced Settings

Aggregation event limit: The maximum number of events to include in any given aggregation event. Defaults to unlimited. Must be at least `1`.

Aggregation memory limit: The maximum memory usage for aggregations within each Worker Process. This limit is applied per process, not globally. By default, it's unlimited, allowing the use of all available system memory. Accepts numerals with multiple-byte units like KB, MB, GB (for example, `4GB`).

Treat dots as literals: Toggle on if your fields or dimensions use dot notation, which are used in StatsD and Open Telemetry metrics formats. Examples: `os.name` or `os.version`. If toggled off (default), metrics or fields using dot notation are enclosed in quotation marks and treated as strings. Example outputs when toggled off:

`"os.name"` or `"os.version"`. Use the Data Preview to confirm that the data output matches your expectations.

Flush on stream close: Toggle on (default) if you want aggregations to flush when an input stream is closed. If toggled off, the Time Window Settings will control flush behavior; this can be preferable in cases like the following:

- Your input data consists of many small files.
- You are sending data to Prometheus. Enabling **Flush on stream close** can send Prometheus multiple aggregations from the same Worker Process for the same time period. Prometheus cannot tell the multiple aggregations apart, and will ingest only the first one.

Aggregation Functions

- `avg(expr:FieldExpression)`: Returns the average `expr` values.
- `count(expr:FieldExpression)`: If `expr` is omitted, returns the number of events received over the time window. If `expr` is given, returns the number of events seen where `expr` evaluates to a value other than null or undefined. For example, with these three events in the same time window:
  - `{ _time: 20, _val: 5 }`
  - `{ _time: 22 }`
  - `{ _time: 24, _val: null }`

  `count()` would return `3`, while `count(_val)` would return `1`.
- `dc(expr: FieldExpression, errorRate: number = 0.01)`: Returns the estimated number of distinct values of `expr`, within a relative error rate. Lower error rates increase the accuracy of the result, at the cost of using more memory. For example, with these three events in the same time window:
  - `{ _time: 20, _name: 'Alice' }`
  - `{ _time: 22, _name: 'Bob' }`
  - `{ _time: 24, _name: 'Alice' }`

  `dc(_name)` would return `2`.
- `distinct_count(expr: FieldExpression, errorRate: number = 0.01)`: Identical to `dc(expr)`.
- `earliest(expr:FieldExpression)`: Returns the earliest (based on `_time`) observed `expr` value.
- `first(expr:FieldExpression)`: Returns the first observed `expr` value.
- `histogram(expr:FieldExpression, buckets: number[])`. Returns the average of the values of `expr` and generates a field with the same name as the aggregated output field, suffixed with `_data`.
  - `buckets` must contain at least one numeric value.

  The `_data` field in the output contains three pieces of information:

- `_sum` - the sum of all values of `expr`.
- `_count` - the number of events seen where `expr` evaluates to a value other than null or undefined.
- `_buckets` - an object whose keys are the buckets defined in the function, and whose values are the number of values that fall in that histogram bucket. In addition to the buckets defined by the user, the object will include a bucket labeled `Infinity`, into which all values will be placed. A value of `x` falls into a histogram bucket if it is less than or equal to the value of the bucket. A value of `45` would fall into a bucket with value `50`, but not one with value `40`. Counts are cumulative; all values counted in a bucket will also be counted in every bucket larger than it. For example, with these three events in the same time window:
  - `{ _time: 20, age: 20 }`
  - `{ _time: 22, age: 25 }`
  - `{ _time: 24, age: 30 }`

  `histogram(age, [10, 25, 40])` would result in the following on the output event:

```
{
  age_histogram: 25
  age_histogram_data: {
    _count: 3,
    _sum: 75
    _buckets: {
      10: 0,
      25: 2,
      40: 3,
      Infinity: 3
    }
  }
}
```

Copy

- `last(expr:FieldExpression)`: Returns the last observed `expr` value.
- `latest(expr:FieldExpression)`: Returns the latest (based on `_time`) observed `expr` value.
- `list(expr:FieldExpression[, max:number, excludeNulls: boolean = true])`: Returns a list of the observed values of `expr`.
  - Optional `max` parameter limits the number of values returned. If omitted, the default is `100`. If set to `0`, will return all values.
  - Optional `excludeNulls` boolean excludes null and undefined values from results. If included, defaults to `true`.

- `max(expr:FieldExpression)`: Returns the maximum `expr` value.
- `median(expr:FieldExpression)`: Returns the middle value of the sorted parameter.
- `min(expr:FieldExpression)`: Returns the minimum `expr` value.
- `mode(expr: FieldExpression[, excludeNulls: boolean = true])`: Returns the single most frequently encountered `expr` value.
    - Optional `excludeNulls` boolean excludes null and undefined values from results. If included, defaults to `true`.
- `per_second(expr:FieldExpression)`: Returns the rate of change of the values of `expr` over the aggregate time window. This is equivalent to `rate(expr, '1s')`. For example, with these three events in the same time window:
    - `{ _time: 20, _val: 5 }`
    - `{ _time: 22, _val: 15 }`
    - `{ _time: 24, _val: 35 }`

  `per_second(_val)` would give back `(35 - 5) / (24 - 20) = 7.5`, meaning that `_val` increased by 7.5 every second over the time window.
- `perc(level: number, expr: FieldExpression)`: Returns `<level>` percentile value of the numeric `expr` values.
- `rate(expr:FieldExpression, timeString: string = '1s')`: Returns the rate of change of the values of `expr` over the aggregate time window. Calculated as (latest value - earliest value) / (latest timestamp - earliest timestamp) * number of seconds in `timeString`. For example, with these three events in the same time window:
    - `{ _time: 20, _val: 5 }`
    - `{ _time: 22, _val: 15 }`
    - `{ _time: 24, _val: 35 }`

  `rate(_val, '2s')` would give back `(35 - 5) / (24 - 20) * 2 = 15`, meaning that `_val` increased by 15 every 2 seconds over the time window.
- `stdev(expr:FieldExpression)`: Returns the sample standard deviation of the `expr` values.
- `stdevp(expr:FieldExpression)`: Returns the population standard deviation of the `expr` values.
- `sum(expr:FieldExpression)`: Returns the sum of the `expr` values.
- `summary(expr:FieldExpression)[, quantiles: number[]])`: Returns the average of the `expr` values and generates a field with the same name as the aggregate output field, suffixed with `_data`.
    - Optional: `quantiles` values must be between `0` and `1` inclusive.

  The `_data` field in the output contains three pieces of information:
    - `_sum` - the sum of all `expr` values.

- `_count` - the number of events seen where `expr` evaluates to a value other than null or undefined.
- `_quantiles` - an object whose keys are the quantiles defined in the function, and whose values are the quantile value across the `expr` values. For example, a quantile key `0.5` and value `500` would indicate that the 50th percentile (or median) of the values seen was 500.
- For example, with these three events in the same time window:
  - `{ foo: 100 }`
  - `{ foo: 200 }`
  - `{ foo: 300 }`

  `summary(foo, [0.25, 0.5, 0.75])` would result in the following on the output event:

```
{
  foo_summary: 25
  foo_summary_data: {
    _count: 3,
    _sum: 75
    _quantiles: {
      0.25: 100,
      0.5: 150,
      0.75: 225
    }
  }
}
```

Copy

- `sumsq(expr:FieldExpression)`: Returns the sum of squares of the `expr` values.
- `top(expr: FieldExpression, count: number[, excludeNulls: boolean = true])`: Returns the most frequently encountered `expr` values, up to `count` number of results.
  - `count` parameter must be a positive integer.
  - Optional `excludeNulls` boolean excludes null and undefined values from results. If included, defaults to `true`.
- `values(expr:FieldExpression[, max: number, excludeNulls: boolean = true, errorRate: number])`: Returns a list of distinct `expr` values.
  - Optional `max` parameter limits the number of values returned; if omitted, the default is `0`, meaning return all distinct values.
  - Optional `excludeNulls` boolean excludes null and undefined values from results. If included, defaults to `true`.

    - Optional `errorRate` parameter controls how accurately the function counts "distinct" values. Range is `0-1`; if omitted, the default value is `0.01`. Higher values allow higher error rates (fewer unique values recognized), with the offsetting benefit of less memory usage.
- `variance(expr:FieldExpression)`: Returns the sample variance of the `expr` values.
- `variancep(expr:FieldExpression)`: Returns the population variance of the `expr` values.

Safeguarding Data

Upon shutdown, Cribl Stream will attempt to flush the buffers that hold aggregated data, to avoid data loss. If you set a **Time window** greater than 1 hour, Cribl recommends adjusting the **Aggregation memory limit** and/or **Aggregation event limit** to prevent the system from running out of memory.

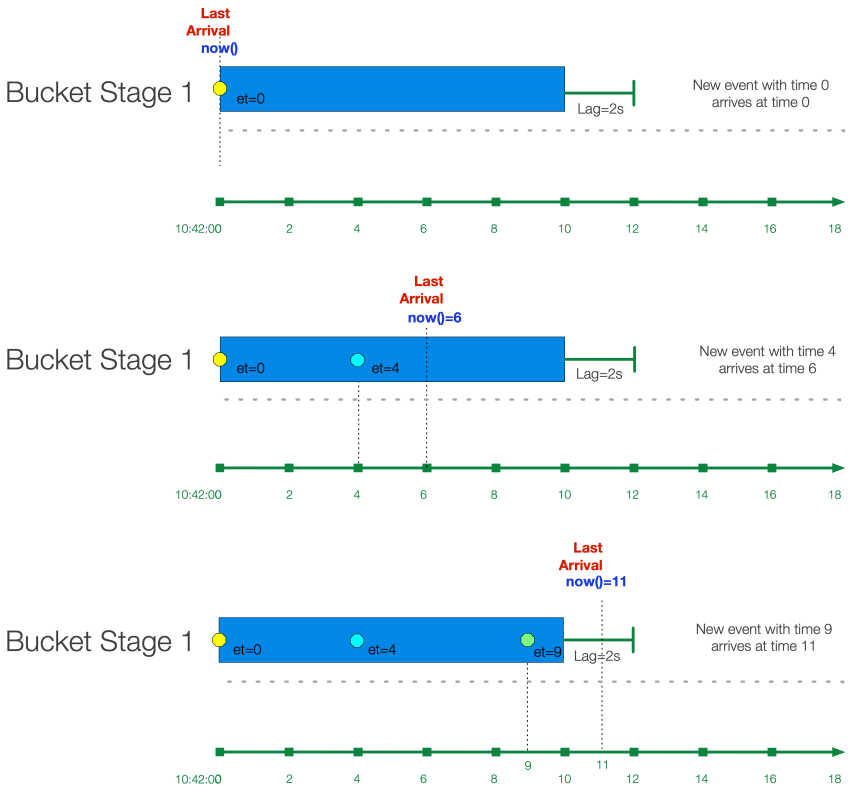
This is especially necessary for high-cardinality data. (Both settings default to unlimited, but we recommend setting defined limits based on testing.)

How Do Time Window Settings Work?

Lag Tolerance

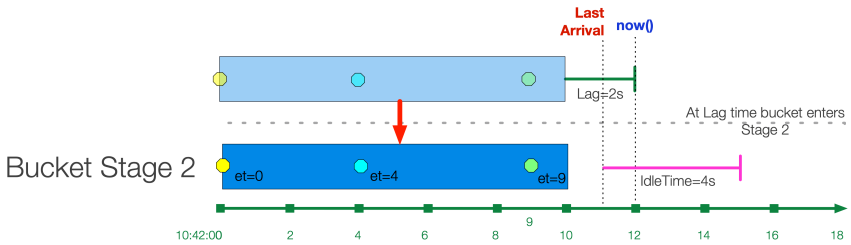
As events are aggregated into windows, there is a good chance that most will arrive later than their event time. For instance, given a `10s` window (`10:42:00 – 10:42:10`), an event with timestamp `10:42:03` might come in 2 seconds later at `10:42:05`.

In several cases, there will also be late, or lagging, events that will arrive **after** the latest time window boundary. For example, an event with timestamp `10:42:04` might arrive at `10:42:12`. Lag Tolerance is the setting that governs how long to wait - after the latest window boundary - and still accept late events.



The “bucket” of events is said to be in Stage 1, where it’s still accepting new events, but it’s not yet finalized. Notice how in the third case, an event with event time `10:42:09` arrives 1 second past the window boundary at `10:42:11`, but it’s still accepted because it happens before the lag time expires.

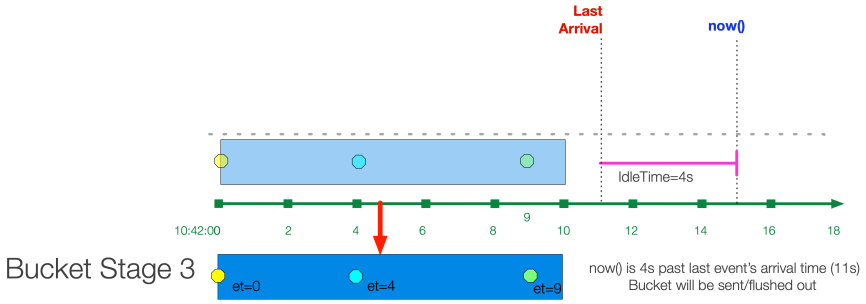
After the lag time expires, the bucket moves to Stage 2.



If the bucket is created from a historic stream, then the bucket is initiated in Stage 2. Lag time is not considered. A “historic” stream is one where the latest time of a bucket is before `now()`. E.g., if the window size is 10s, and `now()=10:42:42`, an event with `event_time=10` will be placed in a Stage 2 bucket with range `10:42:10 - 10:42:20`.

Idle Bucket Time Limit

While Lag Tolerance works with event time, Idle Bucket Time Limit works on arrival time (i.e., real time). It is defined as the amount of time to wait before flushing a bucket that has not received events.



After the Idle Time limit is reached, the bucket is “flushed” and sent out of the system.

Examples

Assume we’re working with VPC Flowlog events that have the following structure:

```
version account_id interface_id srcaddr dstaddr srcport dstport protocol packets bytes
start end action log_status
```

For example:

```
2 99999XXXXX eni-02f03c2880e4aaa3 10.0.1.70 10.0.1.11 9999 63030 6 6556 262256
1554562460 1554562475 ACCEPT OK 2 496698360409 eni-08e66c4525538d10b 37.23.15.38
10.0.2.232 4373 8108 6 1 52 1554562456 1554562466 REJECT OK
```

Scenario A:

Every 10s, compute sum of `bytes` and output it in a field called `TotalBytes`.

Time Window: `10s` Aggregations: `sum(bytes).as(TotalBytes)`

Scenario B:

Every 10s, compute sum of `bytes`, output it in a field called `TotalBytes`, group by `srcaddr`.

Time Window: `10s` Aggregations: `sum(bytes).as(TotalBytes)` Group by fields: `srcaddr`

Scenario C:

Every 10s, compute sum of `bytes` but only where action is `REJECT`, output it in a field called `TotalBytes`, group by `srcaddr`.

Time Window: `10s` Aggregations: `sum(bytes).where(action=='REJECT').as(TotalBytes)` Group by Fields: `srcaddr`

7.2. Aggregate Metrics

The Aggregate Metrics Function computes aggregate statistics for metrics and metric events. It can perform common aggregations such as count, sum, average, min, max, median, rate, and more. This Function helps you pre-aggregate your metrics data before sending it to downstream storage or analysis tools to reduce storage needs and costs. It also simplifies large datasets by reducing complexity and improving query efficiency.

Use this Function to:

- **Reduce high-cardinality data:** Consolidate datasets with a large number of unique or uncommon values to improve query performance and save money on storage or processing costs.
- **Create time-based rollups:** Aggregate high-frequency metrics into larger time intervals (such as hourly or daily) to simplify analysis and improve long-term data retention.
- **Normalize metrics from multiple sources:** Aggregate metrics collected from different systems into a single, unified view.
- **Combine related or duplicate metrics:** Merge related metrics together to create a unified metric.
- **Downsample metrics:** Retain long-term trends while reducing the volume of detailed data.

How the Aggregate Metrics Function Works

The Aggregate Metrics Function processes data in the following order:

1. **Filter:** Checks incoming data for metrics or metric events that match the specified filter values.
2. **Aggregate:** Combines the applicable metrics or metric events based on the configuration settings.
3. **Output:** Extracts, formats, and outputs the aggregated values.

When using this Function, keep these considerations in mind:

- The default metric type is `Automatic`, which allows the aggregation function to determine the appropriate output type.
- Not all methods for aggregating metrics (count, sum, average, and so forth) can convert aggregate data into a certain metrics types. For example, histograms can only convert to other histograms. See Metric Types for available types, recommendations, and supported aggregation functions.

The Aggregate Metrics Function handles metrics data only. In this context, this refers to events with the internal field `__criblMetrics`. If you need to aggregate other types of data, such as logs and traces, use the Aggregations Function instead. If the Aggregate Metrics Function receives non-metric data that do not contain the `__criblMetrics` internal field, it will pass through unchanged without any effect.

Configure the Aggregate Metrics Function

To simplify configuration, the Pipeline Data Preview pane can automate most of the configuration process for the Aggregate Metrics Function. When you build or edit a Pipeline with metrics data, the Data Preview pane automatically detects metrics in your sample data and switches to the Metrics View, which organizes metrics by name. From this view, you can select the metrics you want to aggregate, and Cribl Stream will automatically add an Aggregate Metrics Function to your Pipeline with an initial configuration based on your selection. See Manage Metrics and High Cardinality for more information.

To configure this Function manually:

1. Add a Source and Destination to begin generating metrics. See Integrations for more information.
2. Add a new Pipeline or open an existing Pipeline. Add or capture sample data test your Pipeline as you build it. See Pipelines for more information.
3. At the top of the Pipeline, select **Add Function** and search for `Aggregate Metrics`, then select it.
4. In the **Aggregate Metrics** modal, configure the following general settings:
    - **Filter:** A JavaScript filter expression that selects which metrics to process through the Function. Defaults to `true`, meaning it evaluates all events.
        - If you don't want to use the default setting `true`, you can use a JavaScript expression to apply this Function to specific metric events. See Usage Examples: Filter Expressions for examples.
        - If you use the Metrics View interface in the Pipeline Data Preview to add this Function to your Pipeline, Cribl Stream automatically builds a filter expression based on the metrics you selected. See Manage Metrics and High Cardinality for more information.
    - **Description:** A brief description of how this Function modifies your metrics to help other users understand its purpose later. Defaults to empty.
        - If you use the Metrics View interface in the Pipeline Data Preview to add this Function to your Pipeline, Cribl Stream automatically generates a description based on the selected metrics, including usage tips. Review the generated description and adjust it to better reflect your use case. See Manage Metrics and High Cardinality for more information.
    - **Final:** When enabled, stops data from continuing to downstream Functions for additional processing. Default: Off. Toggle **Final** on if Aggregate Metrics is the last Function in your Pipeline.
    - **Time window:** The duration of the tumbling window for aggregating events. Must be a valid time string (such as `10s`) and match the pattern `\d+[sm]$`. See Time Window Settings for additional settings.
    - **Aggregates**: A table that defines how to aggregate specific metric events. If you use the Metrics View interface in Pipeline Data Preview to add this Function to your Pipeline, Cribl Stream automatically populates this field. See Manage Metrics and High Cardinality for more information. The table includes two inputs:

- **Metric type**: Specifies the type of metric output to apply. Defaults to `Automatic`. Be aware not all aggregation functions can convert aggregate data into a certain metrics types. For example, histograms can only convert to other histograms. See Metric Types for available types, recommendations, and supported functions.
- **Aggregration:** Enter a JavaScript expression that performs the desired aggregation function (such as count, average, sum, and more). See Usage Examples: Aggregation Function Expressions for a full list of available aggregation functions and usage examples.

- **Group by dimensions**: Defines one or more dimensions to group aggregates by. List excluded dimensions first, followed by the dimensions to group by. This field is not a JavaScript expression. Instead it uses a lightweight syntax that accepts simple strings and operators, including wildcard characters. Accepted formats:

Syntax	Description
`!string`	Exclude the specified dimension or field from the grouping. Example: `!proc`.
`string`	Include the specified dimension or field in the grouping. Example: `proc`.
`!*`	Exclude all dimensions or fields from the grouping. Wildcard characters are not supported for dimensions or fields with dotted field names, such as StatsD and Open Telemetry formats.
`*`	Include all dimensions or fields in the grouping. **NOTE:** Use with caution. The wildcard character cannot search or filter for any dimensions or fields that use dot notation, such as `dimension.name` or `field.name`. StatsD and Open Telemetry use dot notation formats.
`string*` or `"string*"`	Include all dimensions or fields that begin with this string pattern. **NOTE:** To search or filter for dimensions or fields that use dot notation, enclose the string in quotation marks. Example: `"os.name"` or `"os.version"`. StatsD and Open Telemetry use dot notation formats.

Using the wildcard character `*` includes all dimensions in the aggregation, which can increase cardinality and lead to higher memory usage. To avoid this impact, first exclude dimensions that may cause high cardinality, then introduce wildcards, if needed. For example: `!_time, !_numericValue, *`

- **Evaluate fields**: A set of key-value pairs to evaluate and add or update. The system adds fields in the context of an aggregated event before sending them out. This setting does not apply to passthrough events.

5. Configure additional settings as needed. For more information, see:
   - Time Window Settings

- Output Settings
- Advanced Settings

6. Test that you configured the Function correctly by comparing sample incoming data with outgoing data in the Data Preview pane and Pipeline Diagnostics tool. See Data Preview and Metrics View for more information.

Time Window Settings

Cumulative aggregations: When enabled, retains cumulative aggregation values when flushing an aggregation table event. When disabled (default), resets aggregation values to `0` on flush.

Lag tolerance: Specifies the tolerance for late events in a tumbling window. Must be a valid time string (such as `10s`) and match the pattern `\d+[sm]$`.

Idle bucket time limit: The amount of time to wait before flushing a bucket that has not received events. Must be a valid time string (such as `10s`) and match the pattern `\d+[sm]$`.

Output Settings

Passthrough mode: When enabled, passes the original events along with the aggregation events. Default: Off.

Sufficient stats mode: Controls whether to output *only* the statistics required for the supplied aggregations. When disabled (default), outputs richer statistics.

Preserve group by fields: Controls whether to retain the original aggregation event's group by field structure.

Output prefix: Adds an optional prefix to all fields output by the Aggregate Metrics Function. Use a prefix to modify the metric name, making it easier to identify and search for aggregated metrics. A prefix helps distinguish aggregated metrics from non-aggregated ones, allowing you to route or search for them more easily without increasing cardinality.

Advanced Settings

Aggregation event limit: The maximum number of events to include in a single aggregation event. Defaults to unlimited. Minimum value: `1`.

Aggregation memory limit: The maximum memory that aggregations can use. Defaults to unlimited (the amount of memory available in the system). Accepts numerals with units such as KB, MB, and GB. Example: `4GB`.

Treat dots as literals: Toggle on (default) if your fields or dimensions use dot notation, which are used in StatsD and Open Telemetry metrics formats. Examples: `os.name` or `os.version`. If toggled off, metrics or fields using dot notation are enclosed in quotation marks and treated as strings. Example outputs when toggled off:

`"os.name"` or `"os.version"`. Use the Data Preview to confirm that the data output matches your expectations.

Flush on stream close: When enabled (default), flushes aggregations when an input stream closes. When disabled, the Time Window Settings control flush behavior. Disabling this setting may be preferable in cases such as:

- Your input data consists of many small files.
- You are sending data to Prometheus. Enabling Flush on stream close can cause Prometheus to receive multiple aggregations from the same Worker Process for the same time period. Prometheus cannot distinguish between them and will ingest only the first one.

Metric Types

This table explains the differences between the metric types and which aggregration functions they support:

Type	Description
Automatic	Default. Select this type to allow the aggregation function determine the appropriate metric output type.
Counter	A counter is a cumulative metric that increases over time. It represents a value that only increments or resets to zero. Counters never decrease unless the system restarts or the counter is manually reset. Use this metric type to monitor changes over time where you need to compute the rate of increase. If you need to capture the current value at the time of measurement rather than a cumulative value, use the gauge metric type instead. Supports aggregation for these functions: • count (recommended for the counter metric type) • avg • distinct_count • dc • earliest • latest • max • max • min • median • mode • perc • per_second • rate

Type	Description
	• stdev • stdev • stdevp • sum • sumsq • summary • variance • variancep
Distribution	**NOTE:** Available for Datadog only. A distribution is a collection of related measurements that capture the spread or variation of data points over time. Instead of tracking a single value, it records a set of values, allowing you to calculate statistics like mean, median, percentiles, and standard deviation. Use this metric type to identify patterns and outliers in performance. Supports aggregation for these functions: • avg • count • distinct_count • dc • earliest • latest • first • histogram • last • max • min • median • mode • perc • per_second • rate • stdev • stdevp • sum • sumsq • summary

Type	Description
	• variance • variancep
Gauge	A gauge represents a single value at a specific point in time. Unlike a counter, a gauge can increase or decrease to reflect the current state rather than its cumulative value. Use this metric type to provide a snapshot of a value at a particular moment. Supports aggregation for these functions: • sum (recommended for the gauge metric type) • avg • count • distinct_count • dc • earliest • latest • first • histogram • last • max • median • min • mode • per_second • perc • rate • stdev • stdevp • sumsq • summary • variance • variancep
Histogram	A histogram measures the distribution of a set of values over time. It tracks both the count of events and how those events are distributed across a range of values. Use this metric type to measure the frequency of values falling within predefined ranges. Histograms can be useful for identifying outliers and performance bottlenecks. Supports aggregation for histograms only.

Type	Description
Summary	A summary is similar to a histogram but computes specific quantiles over time (such as median or 90th percentile)instead of counting values in predefined ranges. Use this metric type to capture statistical trends in specific statistical measures over time. Supports aggregation for summary only.
Timer	A timer measures the duration of an event from start to finish. It records how long an operation takes and allows you to calculate statistics like min, max, average, and percentiles over time. Use this metric type when when your focus is specifically on measuring elapsed time. Supports aggregation for these functions: • avg • count • distinct_count • dc • earliest • latest • first • histogram • last • max • median • min • mode • perc • per_second • rate • stdev • stdevp • sum • sumsq • summary • variance • variancep

Usage Examples

This section provides typical JavaScript expressions that you might commonly use with the Aggregate Metrics Function.

Filter Expressions

The **Filter** setting is a JavaScript expression that selects which data to process through the Aggregate Metrics Function. The following is an example of a JavaScript expression that filters for three metrics named `proc.cpu_perc`, `proc.mem_perc`, and `proc.bytes_in` and other variants that may include these names:

```
(_metric == 'proc.cpu_perc' || __criblMetrics[0].nameExpr.includes("'proc.cpu_perc'")
(_metric == 'proc.mem_perc' || __criblMetrics[0].nameExpr.includes("'proc.mem_perc'")
(_metric == 'proc.bytes_in' || __criblMetrics[0].nameExpr.includes("'proc.bytes_in'")
```

Copy

Aggregation Function Expressions

The **Aggregates** table in the Aggregate Metric Function defines how to aggregate specific metric events. You can use the **Aggregation** column in this table to enter a JavaScript expression that performs the desired aggregation function. The Aggregate Metrics Function shares much of its functionality with the Aggregations Function. For more detailed examples, see the Aggregations Function documentation.

This table provides examples of JavaScript expressions for common aggregation functions for the Aggregate Metrics Function:

Aggregation	Example
Average	`avg(_value || proc.mem_perc).as(proc.mem_perc_avg)`
Count	`count(_value || proc.bytes_in).as(proc.bytes_in_count)`
Sum	`sum(_value || proc.mem_perc).as(proc.mem_perc_sum)`

7.3. Auto Timestamp

The Auto Timestamp Function extracts time to a destination field, given a source field in the event. By default, Auto Timestamp makes a first best effort and populates `_time`. When you add a sample (via paste or a local file), you should accomplish time and event breaking at the same time you add the data.

This Function allows fine-grained and powerful transformations to populate new time fields, or to edit existing time fields. You can use the Function's Additional timestamps section to create custom time fields using regex and custom JavaScript `strptime` functions.

The Auto Timestamp Function uses the same basic algorithm as Event Breakers and the C.Time.timestampFinder() native method.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. The default `true` setting passes all events through the Function.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Source field: Field to search for a timestamp. Defaults to `_raw`.

Destination field: Field to place extracted timestamp in. Defaults to `_time`. Supports nested addressing.

Default timezone: Select a timezone to assign to timestamps that lack timezone info. Defaults to `Local`. (This drop-down includes support for legacy names: `EST5EDT`, `CST6CDT`, `MST7MDT`, and `PST8PDT`.)

Additional timestamps: To extract additional timestamp formats, select **Add Timestamp** to define each format. Each row will provide these fields:

- **Regex**: Regex, with first capturing group matching the timestamp. You can type in a pattern, or use one of the predefined Regexes.
- **Strptime format**: Select or enter the `strptime` format for the captured timestamp.

Advanced Settings

Time expression: Expression with which to format extracted time. Current time, as a JavaScript Date object, is in global `time`. Defaults to `time.getTime() / 1000`. You can access other fields' values via `__e.<fieldName>`.

For details about Cribl Stream's Library (native) time methods, see: C.Time - Time Functions.

Start scan offset: How far into the string to look for a time string.

Max timestamp scan depth: Maximum string length at which to look for a timestamp.

Default time: How to set the time field if no timestamp is found. Defaults to **Current time**.

Two fields enable you to constrain (clamp) the parsed timestamp, to prevent the Function from mistakenly extracting non-time values as unrealistic timestamps:

- **Earliest timestamp allowed**: Enter a string that specifies the latest allowable timestamp, relative to now. (Sample value: `-42years`. Default value: `-420weeks`.) Parsed values earlier than this date will be set to the **Default time**.
- **Future timestamp allowed**: Enter a string that specifies the latest allowable timestamp, relative to now. (Sample value: `+42days`. Default value: `+1week`.) Parsed values after this date will be set to the **Default time**.

Format Reference

This references https://github.com/d3/d3-time-format#locale_format. Directives annotated with a `(†)` symbol might be affected by the locale definition.

```
%a - abbreviated weekday name. (†)
%A - full weekday name. (†)
%b - abbreviated month name. (†)
%B - full month name. (†)
%c - the locale's date and time, such as %x, %X. (†)
%d - zero-padded day of the month as a decimal number [01,31].
%e - space-padded day of the month as a decimal number [ 1,31]; equivalent to %_d.
%f - microseconds as a decimal number [000000, 999999].
%H - hour (24-hour clock) as a decimal number [00,23].
%I - hour (12-hour clock) as a decimal number [01,12].
%j - day of the year as a decimal number [001,366].
%m - month as a decimal number [01,12].
%M - minute as a decimal number [00,59].
%L - milliseconds as a decimal number [000, 999].
%p - either AM or PM. (†)
%Q - [Unix time](https://en.wikipedia.org/wiki/Unix_time) (milliseconds).
%s - Unix time (seconds).
%S - second as a decimal number [00,61].
%u - Monday-based (ISO 8601) weekday as a decimal number [1,7].
```

```
%U - Sunday-based week of the year as a decimal number [00,53].
%V - ISO 8601 week of the year as a decimal number [01, 53].
%w - Sunday-based weekday as a decimal number [0,6].
%W - Monday-based week of the year as a decimal number [00,53].
%x - the locale's date, such as %-m/%-d/%Y. (†)
%X - the locale's time, such as %-I:%M:%S %p. (†)
%y - year without century as a decimal number [00,99].
%Y - year with century as a decimal number.
%Z - time zone offset, such as -0700, -07:00, -07, or Z.
%% - a literal percent sign (%).
```

Complying with the Format

In order to use auto timestamping upon ingestion, the formatting used must match the `%Z` parameters above. For example, this Function will automatically parse all of these formats:

- `2020/06/10T17:17:35.004-0700`
- `2020/06/10T17:17:35.004-07:00`
- `2020/06/10T17:17:35.004-07`
- `2020/06/10T10:17:35.004Z`
- `2020/06/10T11:17:35.004 EST`

To parse other formats, you can use the Additional Timestamps section's internal **Regex** or **Strptime Format** operators.

Basic Example

Create a Pipeline with one Auto Timestamp Function. Leave the settings of the Function to their defaults.

Sample event:

```
Jan 20 12:03:55 PA-VM 1,2025/01/20 13:03:58,CRIBL,TRAFFIC,end,2049,2024/09/20
14:03:58,314.817.108.226,10.0.0.102,314.817.108.226,10.0.2.65,cribl,,,incomplete,vsys1,u
forwarding-default,2024/09/20
13:03:58,574326,1,53722,8088,53722,8088,0x400064,tcp,allow,296,296,0,4,2024/09/20
13:03:45,7,any,0,730277,0x0,United States,10.0.0.0-10.255.255.255,0,4,0,aged-
out,0,0,0,0,,PA-VM,from-policy,,,0,,0,,N/A,0,0,0,0
```

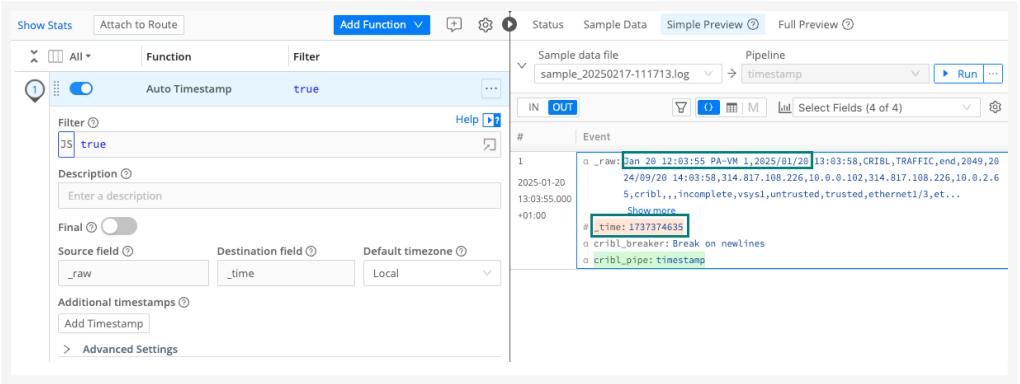
To add this sample (after creating an Auto Timestamp Function with the above **Filter** expression), in the Pipeline page go to **Sample Data** > **Import Data**, and paste the data snippet above. Do not make any changes to timestamping or line breaking, and select **Save as Sample File**.

By default, Cribl Stream will inspect the first 150 characters, and will extract the first valid timestamp it sees. You can modify this character limit under **Advanced Settings** > **Max Timestamp Scan Depth**.

Cribl Stream will grab the first part of the event, and will settle on the first matching value to display for `time`:

- `_time: 1737374635`
- **GMT**: Jan 20, 2025 12:03:55 PM GMT
- **Your Local Time**: Jan 20, 2025 1:03:55 PM GMT+01:00

Because no explicit timezone has been set (under **Default Timezone**), `_time` will inherit the **Local** timezone, which in this example is `GMT +01:00`.



Auto Timestamp Basic Example

Timezone Dependencies and Details

Cribl Stream uses ICU for timezone information. It does not query external files or the operating system. The bundled ICU is updated periodically.

For additional timezone details, see: https://www.iana.org/time-zones.

Advanced Settings Example

The `datetime.strptime()` method creates a datetime object from the string passed in by the **Regex** field.

Here, we'll use `datetime.strptime()` to match a timestamp in AM/PM format at the end of a line.

Sample:

```
This is a sample event that will push the datetime values further on inside the event.
This is still a sample event and finally here is the datetime information!:
Server_UTC_Timestamp="04/27/2020 2:30:15 PM"
```

Max timestamp scan depth: `210`

Click to add **Additional timestamps**:

Regex: `(\d{1,2}\/\d{2}\/\d{4}\s\d{1,2}:\d{2}:\d{2}\s\w{2})`

Strptime format: `'%m/%d/%Y %H:%M:%S %p'`

Gnarly Details

- This Function supports the `%f` (microseconds) directive.
- Cribl Stream will truncate timestamps to three-digit (milliseconds) resolution, omitting trailing zeros.
- For further examples, see Extracting Timestamps from Messy Logs.

7.4. CEF Serializer

The CEF Serializer takes a list of fields and/or values, and formats them in the Common Event Format (CEF) standard. CEF defines a syntax for log records. It is composed of a standard prefix, and a variable extension formatted as a series of key-value pairs.

Format

`CEF:Version|Device Vendor|Device Product|Device Version|Device Event Class ID|Name|Severity|[Extension]`

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Output field: The field to which the CEF formatted event will be output. Nested addressing supported. Defaults to `_raw`.

Header Fields

CEF Header field definitions. The field values below will be written pipe (|)-delimited in the Output Field. Names cannot be changed. Values can be computed with a JavaScript expression, or can be constants.

- **cef_version**: Defaults to `CEF:0`.
- **device_vendor**: Defaults to `Cribl`.
- **device_product**: Defaults to `Cribl`.
- **device_version**: Defaults to `C.version`.
- **device_event_class_id**: Defaults to `420`.
- **name**: Defaults to `Cribl Event`.
- **severity**: Defaults to `6`.

Extension Fields

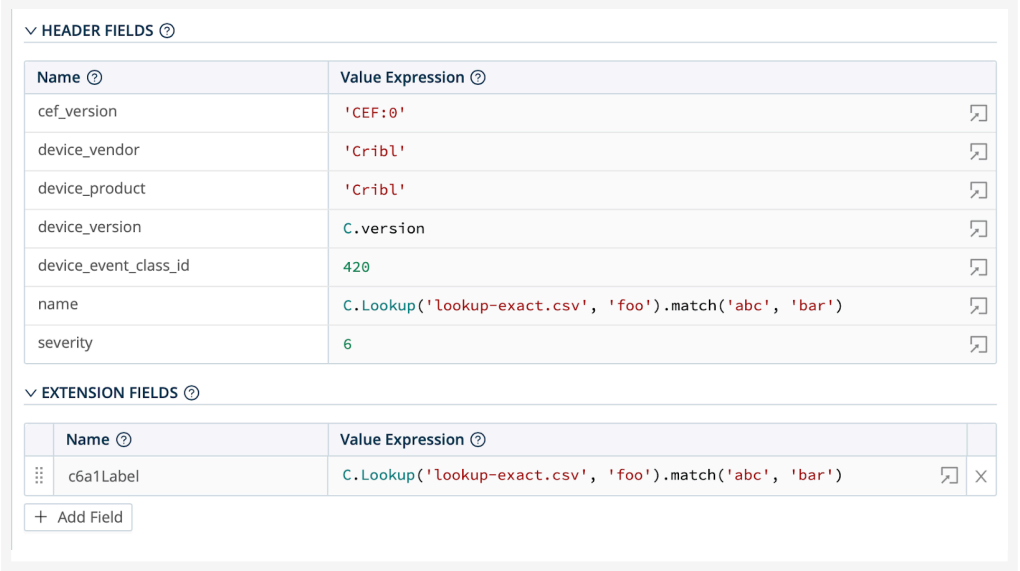
CEF Extension field definitions. Field names and values will be written in `key=value` format. Select each field's Name from the drop-down list. Values can be computed with a JavaScript expression, or can be constants.

Example

For each CEF field, allowed values include strings, plus any custom Cribl function. For example, if using a lookup:

Name: `Name` Value expression: `C.Lookup('lookup-exact.csv', 'foo').match('abc', 'bar')`

This can be used for any of the `CEF` **Header Fields**.



The resulting event has the following structure for an **Output Field** set to `_CEF_out`:

```
_CEF_out:CEF:0|Cribl|Cribl|42.0-61c12259|420|Business Group
6|6|c6a1Label=Colorado_Ext_Bldg7
```

7.5. Chain

The Chain Function does one thing: It connects (or *chains*) data processing from a Pipeline or Pack to another Pipeline or Pack. This can be useful for sequential processing, or just to separate groups of related Functions into discrete Pipeline or Pack units that make intuitive sense.

Control Flow

The chained Pipeline or Pack will, upon completion, normally return control to the parent Pipeline/Pack containing the Chain Function. However, if the chained Pipeline/Pack contains any Function with the **Final** flag enabled, processing will stop there. In this case, back in the parent Pipeline/Pack, no Function below Chain will execute.

Cycle Detection and Throughput

The Chain Function includes guardrails against circular references. In v.4.3 and newer, Cribl Stream detects cycles when you configure your Pipeline. This early detection (compared to older versions' runtime detection) means that:

- If you try to save a Pipeline with a cyclical reference, Cribl Stream will throw an error.
- If an existing Pipeline's configuration contains a cycle, Cribl Stream will disable the final Chain Function involved in the cycle.

Despite these safeguards, Cribl recommends that you keep chained configurations simple and understandable by all your users. Also, keep in mind that inserting a Chain Function can impose a slight performance hit, compared to including all processing in the original Pipeline or Pack.

Pipeline Versus Pack Scope

You will see different scope restrictions when using Chain in a Pipeline versus in a Pack:

- In a Pipeline, the **Processor** drop-down displays both Pipelines and Packs as targets to chain to.
- In a Pack, the **Processor** drop-down offers only Pipelines contained within that Pack.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

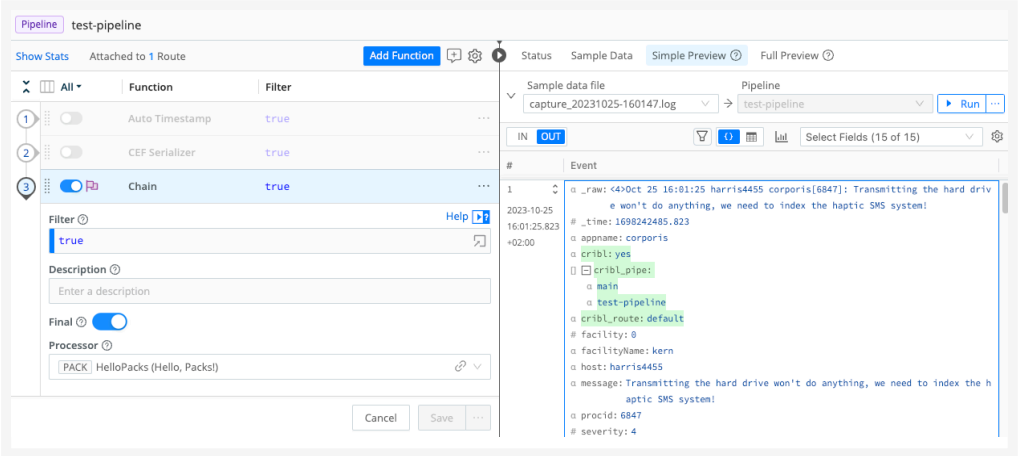
Description: Optionally, add a simple description of this Function's purpose. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off. (Note that this will not prevent data from flowing to the Function's defined **Processor**.)

Processor: Use this drop-down to select a configured Pipeline or Pack through which to forward events.

Example

This shows a simple preview of a `pipeline-1` Pipeline, which chains to a `pipeinpipe` Pipeline. Notice that each event's added `cribl_pipe` field lists all Pipelines/Packs through which the event was chained.



cribl_pipe field shows whole processing path

7.6. Clone

The Clone Function creates a copy of events in the same Pipeline, with optional added fields. Cloned events go to the same Destination as the original event, because they are in the same Pipeline. If your Destination is an Output Router, you can use filters to send cloned events to different Destinations.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Clones: Create clones with the specified fields added and set.

Fields: Set of key-value pairs to add. Nested addressing is supported.

Examples

Staging Example

In this example, the Destination will receive a clone with an `env` field set to `staging`.

Field: `env` **Value**: `staging`

Index Example

In this scenario, we insert a Clone Function at the beginning of a Pipeline to create cloned events. We can later use these events as a baseline to compare against the original events, after various Functions have processed them.

You can assign any meaningful fields to the cloned events - anything that will help you identify them when comparing. This example simply assigns a key-value pair of `index: clones`.

Field: `index` **Value**: `clones`

To keep the cloned events from being processed by Functions later in the same Pipeline, you'll need to specify `index!='clones'` in their **Filter** expressions.

7.7. Code

If you need to operate on data in a way that can't be accomplished with Cribl Stream's out-of-the-box Functions, the Code Function enables you to encapsulate your own JavaScript code. This Function imposes some restrictions for security reasons.

Code Function Walkthrough

This video provides a one-minute walkthrough of the Cribl Stream Code Function.

Restrictions

Generally speaking, anything forbidden in JavaScript strict mode is forbidden in the context of the Code Function. Specifically, the following are **not allowed**:

- `console`, `eval`, `uneval`, `Function (constructor)`, `Promises`, `setTimeout`, `setInterval`, `global`, `globalThis`, `window`, and `set`.

Code Functions **can** include `for` loops, `while` loops, and JavaScript methods such as `map`, `reduce`, `forEach`, `some`, and `every`. For further details, see Supported JavaScript Options.

Cribl Stream's predefined Functions, such as Eval, cover the vast majority of scenarios that users typically need to implement. You should use Code Functions only as a last resort, when you need to construct a complex

block of code.

Also, only skilled JavaScript developers should define Code Functions. This is to avoid unintended results - such as creating infinite loops, or otherwise failing to return - that could needlessly add to your throughput burden.

Usage

When added to a Pipeline, the Code Function offers the following configuration options:

Filter: JavaScript filter expression that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Optionally, add a simple description of this Function.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Code: The mini-editor where you type your JavaScript code.

Advanced Settings

Iteration limit: The maximum number of iterations per instance of this Code Function. Defaults to `5000`; highest allowed value is `10000`.

Error log sample rate: Specifies the rate at which this Code Function logs errors. For example, a value of `1` logs every error; a value of `10` logs every tenth error. The highest allowed value is `5000`. Defaults to `1`.

Use unique log channel: When enabled, Cribl Stream sends logs from this function to a unique channel in the form `func:code:${pipelineName}:${functionIndex}`. Toggle off to use the generic `func:code` log channel instead.

Notes and Examples

Functions (including the Code Function) always use the special variable `__e` to access the `(context)` event inside JavaScript expressions.

Possibly the simplest Code Function creates a new field and then assigns it a value:

A Basic Code Function

```
__e['foo'] = 'Hello, Goats!'
```

Copy

For more ambitious implementations, see Code Function Examples.

JavaScript Support

Cribl Stream supports the ECMAScript® 2015 Language Specification.

With some exceptions, the Code Function supports the options described in the following MDN JavaScript Guide topics:

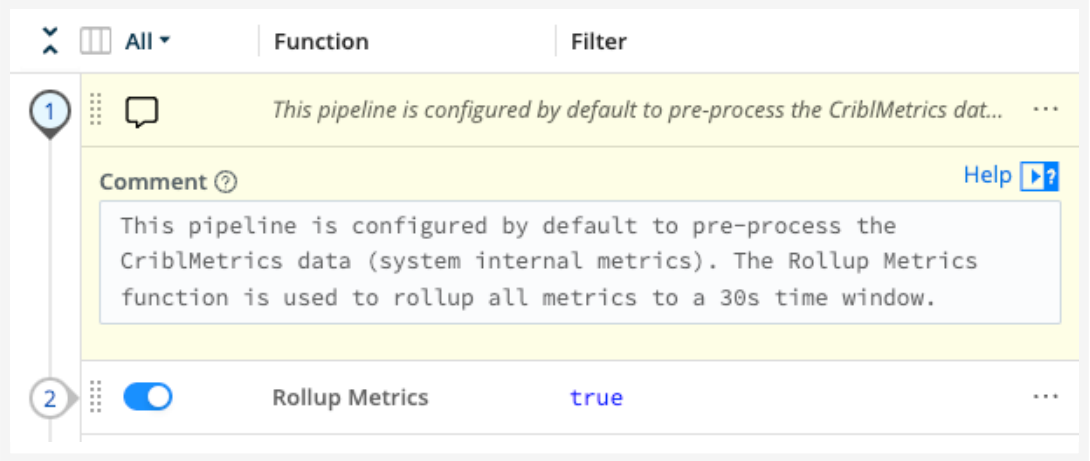
- Expressions and Operators
- Global variables
- Control flow and error handling
- Loops and iteration
- Functions
- Numbers and dates
- Text formatting
- Regular Expressions
- Indexed collections
- Keyed collections
- Working with Objects

7.8. Comment

The Comment Function adds a text comment in a Pipeline. It makes no changes to event data. The added comment is visible only within the Pipeline UI, where it is useful for labeling Pipeline steps. The Comment Function always has a minimum, non-zero processing time.

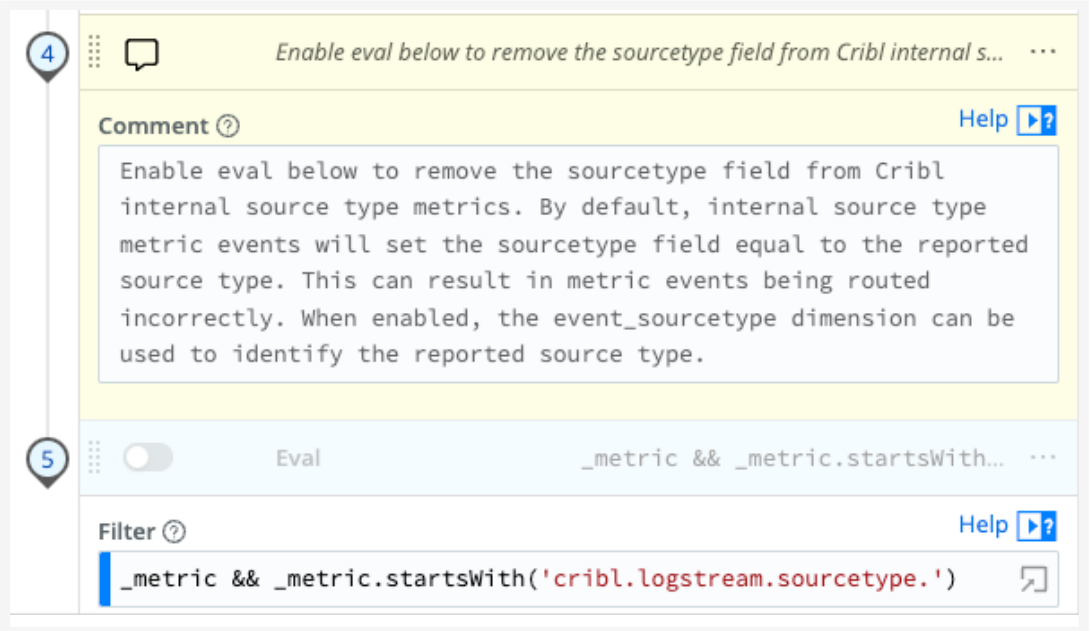
Usage

Comment: Add your comment as plain text in this field.



Examples

This comment labels the Pipeline's next function:



7.9. DNS Lookup

The DNS Lookup Function offers two operations useful in enriching security and other data:

- DNS lookups based on host name as text, resolving to A record (IP address) or to other record types.
- Reverse DNS Lookup. (This duplicates the behavior of Cribl Stream's prior Reverse DNS Function, which was deprecated as of v.2.4, and has been removed as of v.4.0.)

To reduce DNS lookups and minimize latency, the DNS Lookup Function incorporates a configurable DNS cache (including resolved and unresolved lookups). If you need additional caching, consider enabling OS-level DNS caching on each Cribl Stream Worker that will execute this Function. (OS-level caching options include DNSMasq, nscd, systemd-resolved, etc.)

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

DNS Lookup Fields Section

Lookup field name: Name of the field containing the domain to look up.

Resource record type: DNS record type (RR) to return. Defaults to `A`' record.

Output field name: Lookup result(s) will be added to this field. Leave blank to overwrite the original field specified in **Lookup field name**.

Reverse DNS Lookup Fields Section

Lookup field name: Name of the field containing the IP address to look up.

If the field value is not in IPv4 or IPv6 format, the lookup is skipped.

Output field name: Name of the field in which to add the resolved hostname. Leave blank to overwrite the original field specified in **Lookup field name**.

Fallback Resolving Methods

This section contains settings for resolving DNS short names.

Use /etc/resolv.conf: Toggle on if you want Cribl Stream to resolve DNS short names using the search or domain directive from `/etc/resolv.conf`. Default is toggled off. Not applicable for Windows hosts.

Use search or domain fallback(s): Add fallback value(s) for the DNS resolver to use when it cannot resolve a DNS short name.

Fall back to DNS.lookup(): Toggle on if you want Cribl Stream to make a `DNS.lookup()` call to resolve a DNS short name it can't resolve by other methods.

If you enable all of these methods at once. Cribl Stream will attempt to resolve short names by trying each method in the following order:

1. **Use /etc/resolv.conf**.
2. **Use search or domain fallback(s)**.
3. **Fall back to DNS.lookup()**.

If Cribl Stream can't resolve the short name after trying all of these methods, it will drop the output field from the event.

Potential Performance Impact

Making a `DNS.lookup()` call might degrade performance in unrelated areas of Cribl Stream. Use this method only if you have not been able to resolve DNS short names using other settings.

Advanced Settings

DNS server overrides: IP address(es), in RFC 5952 format, of the DNS server(s) to use for resolution. IPv4 examples: `1.1.1.1`, `4.2.2.2:53`. IPv6 examples: `[2001:4860:4860::8888]`, `[2001:4860:4860::8888]:1053`. If this field is not specified, Cribl Stream will use the system's DNS server.

Cache time to live (minutes): Determines the interval on which the DNS cache will expire, and its contents will be refetched. Defaults to `30` minutes. Use `0` to disable cache expiration/refresh behavior.

Cache size limit: Maximum number of DNS resolutions to cache locally. Before changing the default `5000`, consider the implications for your system. Higher values will increase memory usage. Highest allowed value is `100000`.

Log level for failed lookups: The log level at which to capture failed lookup events. Defaults to error.

Monitoring DNS Cache State

When the DNS Lookup Function succeeds in looking up a value it has stored in the DNS cache, that is called a **hit**. Each hit enables the Function to skip contacting a DNS server outside of Cribl Stream to perform the DNS lookup. The resulting performance improvement is the main benefit that the DNS cache provides.

When the DNS Lookup Function looks for a value in the DNS cache, but cannot find it, that is called a **miss**. In this case, the Function must contact a DNS server to perform the lookup.

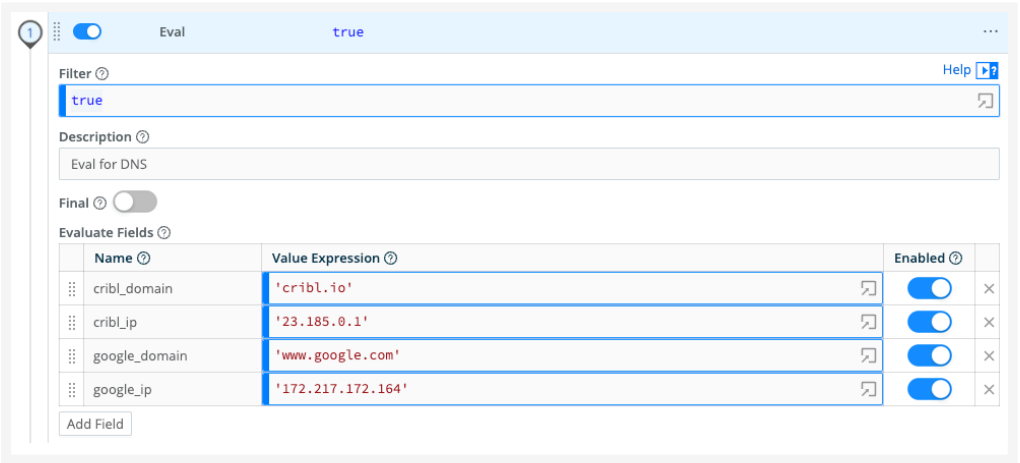
Cribl Stream logs the DNS cache's state in the `func:dns_lookup` channel. The table below explains some typical messages.

Log Message	Meaning
`hits`	Number of successful lookups from cache.
`misses`	Number of unsuccessful lookups from cache.
`keys`	Number of entries in cache.
`ksize`	Approximate amount of memory occupied by keys, in bytes.
`vsize`	Approximate amount of memory occupied by values, in bytes.

See Monitoring for a general explanation of logging in Cribl Stream.

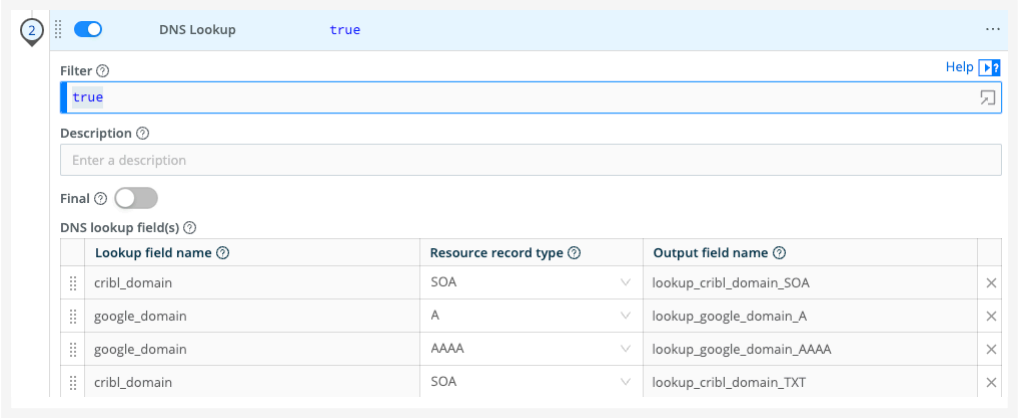
Example

This example Pipeline chains two Functions. First, we have an **Eval** Function that defines key-value pairs for two alphabetical domain names and two numeric IP addresses.



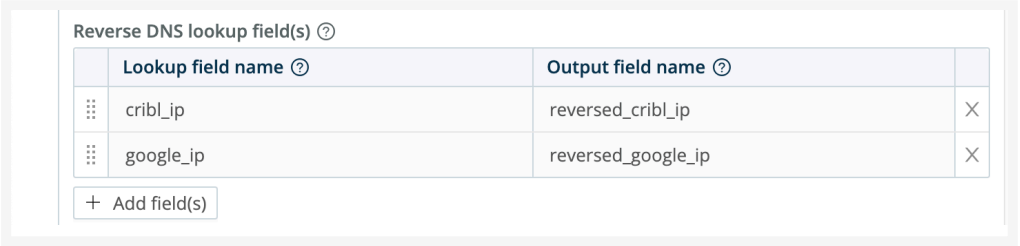
DNS Lookup: Eval Function

Next, the DNS Lookup Function looks up several record types for the two domain names, placing each retrieved record type in its own output field.



DNS Lookup: multiple record types

Finally, the same Function's **Reverse DNS lookup** section retrieves domain names for the two IP addresses.

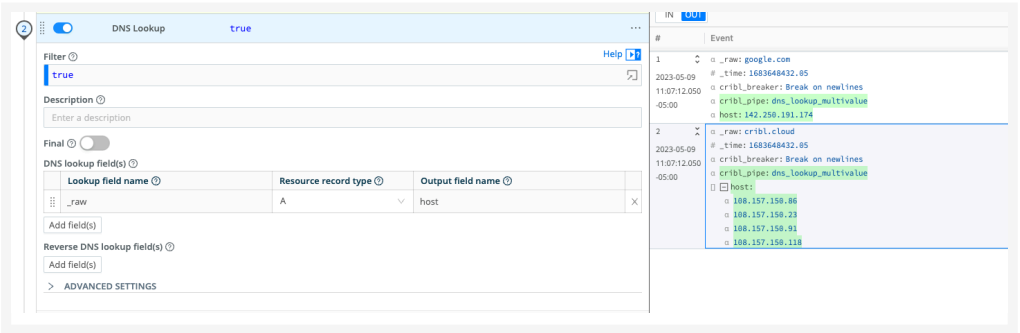


DNS Lookup: reverse lookups

Working with Multi-value Results

DNS records can contain single values mixed together with multi-value results. While this can be challenging to work with, the right code can adjust your results to the desired values.

In the screenshot below, the DNS lookup for `google.com` (event 1) returns a single record, while the `cribl.cloud` lookup (event 2) returns four addresses.



DNS lookup with multi-value result

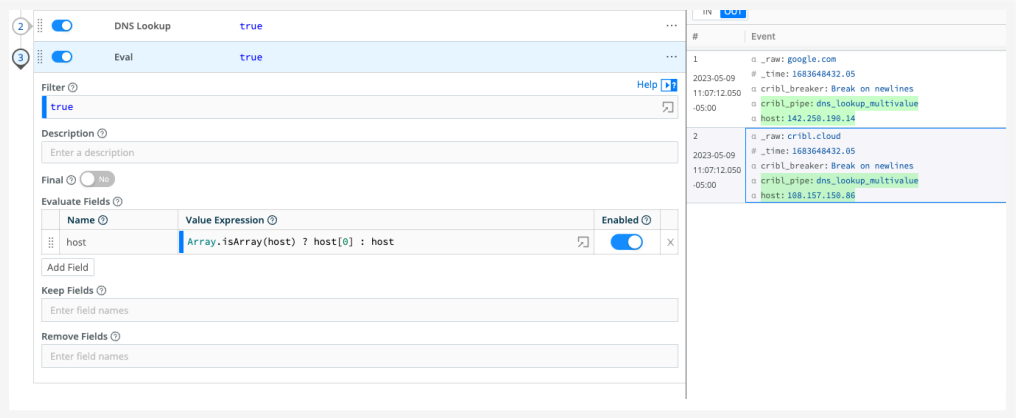
Sometimes a receiver is unable to consume the data in an array format. One solution to this problem is to pick the first result and convert it into a string. Here's a JavaScript eval to do this:

```
Array.isArray(host) ? host[0] : host;
```

Copy

This function checks whether the value of the `host` field is an array.

- If true, the result is the left side of the ternary expression - in this case, `host[0]`, which is the first element in the array, converted into a string. See event 2 in the screenshot below.
- If false, the result is the single-value string in the `host` field. See event 1.



DNS lookup multi-value result converted into a single value

Another option is to **randomly** pick an address from the result. To do this, use a modified version of the JavaScript eval:

```
Array.isArray(host) ? host[Math.floor(Math.random() * host.length)] : host
```

Copy

Here, instead of picking the first element, the code uses `Math.floor(Math.random() * host.length)` to randomly pick an element from the array.

7.10. Drop

The Drop Function deletes any events that meet its Filter expression. This is useful when you want to prevent certain events from continuing to a Pipeline's downstream Functions.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Examples

Scenario A:

Assume that we care only about errors, so we want to filter out any events that contain the word "success," regardless of case: "success," "SUCCESS," etc.

In our Drop Function, we'll use the JavaScript `search()` method to search the `_raw` field's contents for our target pattern. We know that `search()` returns a non-negative integer to indicate the starting position of the first match in the string, or `-1` if no match. So we can evaluate the Function as true when the return value is `>= 0`.

Filter: `_raw.search(/success/i)>=0`

Scenario B:

You can filter out specific JSON events based on their key-value pairs.

The following Filter expression uses a strict inequality operator to check, per event, whether `channel` has a different data type **or** value from `auth`. Matching events will drop, ensuring that only events with `"channel":"auth"` will pass to the next Function.

Filter: `channel !== 'auth'`

7.11. Drop Dimensions

The Drop Dimensions Function reduces the number of unique dimension combinations in outgoing metrics data to help address high cardinality. Use it to improve query performance and reduce storage costs by removing unnecessary dimensions.

This Function creates new aggregated events for the dropped dimensions. By dropping selected dimensions, you can reduce the number of unique time series, allowing you to group together incoming metrics and and roll them up into a single event. It merges frequently generated metrics into more manageable time windows, which creates smaller, coarser-grained datasets for downstream systems.

How the Drop Dimensions Function Works

The Drop Dimensions Function processes data in the following order:

1. **Filter:** Checks incoming data for events that match the specified filter values. You can filter by metric dimensions or any field on the incoming event.
2. **Drop:** Removes the specified dimensions from those events.
3. **Aggregate:** Combines all events with the same remaining fields down into a new aggregated event.
4. **Output:** Extracts, formats, and outputs the aggregated result as a new event.

Considerations

When using this Function, keep these considerations in mind:

- This Function does not perform any conversions from one metric type to another.
- Cribl Stream does not support aggregation of histogram, summary, or distribution metric types. The Function handles these metrics in special ways:
    - **Only unsupported metrics:** If an event only contains one of these unsupported metric types (histogram, distribution, or summary), these metrics pass through the Function unchanged.
    - **Mixed metrics:** If an event includes both supported and unsupported metric types (including histograms, distributions, or summaries), Cribl Stream splits the event into multiple events, each containing a single metric. These events are then processed by the Function normally.
- Cribl Stream creates new aggregated events for each supported metric. When you drop dimensions, it reduces the number of unique time series, enabling more events to be grouped and rolled up together. The result is a set of new, aggregated events that merge frequent metrics into more manageable time windows. Each new event includes only the fields specified in `__criblMetrics`:
    - A field defined by `nameExpr`, where the value of the referenced field becomes the new field name. For example, if `nameExpr: ['_metric']` and the event has `_metric: 'cpu.usage'`, the

output event will contain `cpu.usage: 1`.

- A value field as defined in the values array, where the referenced field provides the value of the new metric field. For example, if `values: ['_value']` and the event has `_value: 1`, then `1` will be the value for `cpu.usage`.
- Any matching dimension fields retained from the original event.

While `nameExpr` usually references a field (such as `_metric`), it can also be an expression. In such cases, Cribl Stream uses the evaluated result of the expression as the output field name.

Any fields that are on the event but not specified here will not be present on the outputted event.

Memory Usage and Cardinality

The Drop Dimensions Function processes metrics by grouping and aggregating data based on dimension combinations. This can lead to increased memory usage during processing if you are processing high-cardinality datasets where many unique combinations of metric names and dimensions exist.

To help identify and troubleshoot potential memory issues, the Function logs the following diagnostic values:

Diagnostic value	Description
`numUniqueMetricAndDims`	The number of unique combinations of metric names and dimensions currently being tracked.
`unFlushedEventCount`	The number of events still waiting to be aggregated based on their new (post-drop) dimension list.
`totalBytesTotal`	The total memory (in bytes) currently used by the Function to hold and process data.

If you observe unusually high values for `numUniqueMetricAndDims` or `totalBytesTotal`, this may be a sign of excessive cardinality. In extreme cases, high cardinality may cause the Drop Dimensions Function to exceed the memory limits of the Worker Processes. In these cases, consider dropping additional dimensions to reduce the number of unique time series generated.

Configure the Drop Dimensions Function

To configure this Function manually:

1. Add a Source and Destination to get data flowing. See Integrations for more information.
2. Add a new Pipeline or open an existing Pipeline. Add sample data or capture sample data to test your Pipeline as you build it. See Pipelines for more information.

3. At the top of the Pipeline, select **Add Function** and search for `Drop Dimensions`, then select it.
4. In the **Drop Dimensions** modal, configure the following general settings:
    - **Filter:** A JavaScript filter expression that selects which metrics to process through the Function. Defaults to `true`, meaning it evaluates all events. If you don't want to use the default, you can use a JavaScript expression to apply this Function to specific metric events. See Usage Examples: Filter Expressions for examples.
    - **Description:** A brief description of how this Function modifies your metrics to help other users understand its purpose later. Defaults to empty.
    - **Final:** When enabled, stops data from continuing to downstream Functions for additional processing. Default: Off. Toggle **Final** on if Drop Dimensions is the last Function in your Pipeline.
    - **Aggregation time window:** The duration of the tumbling window for aggregating events. Must be a valid time string (such as `10s`) and match the pattern `\d+[sm]$`.
    - **Dimensions to drop:** Add or edit the list of dimension names to drop. This field is not a JavaScript expression. Instead it uses a lightweight syntax that accepts simple strings and operators, including wildcard characters. For most dimension names, you can use the syntax `'dimensionName'` to access dimension names. See Usage Examples: Drop Dimensions for examples. Accepted formats:

Syntax	Description
`!string`	Exclude the specified dimension or field from the grouping. Example: `!proc`.
`string`	Include the specified dimension or field in the grouping. Example: `proc`.
`!*`	Exclude all dimensions or fields from the grouping.
`*`	Include all dimensions or fields in the grouping. **NOTE:** Use with caution. The wildcard character cannot search or filter for any dimensions or fields that use dot notation, such as `dimension.name` or `field.name`. StatsD and Open Telemetry use dot notation formats.
`string*` or `"string*"`	Include all dimensions or fields that begin with this string pattern.

    - **Flush on stream close:** When enabled (default), flushes aggregations when an input stream closes. When disabled, the **Aggregation time window** controls flush behavior. Disabling this setting may be preferable in cases such as:
        - Your input data consists of many small files.
        - You are sending data to Prometheus. Enabling Flush on stream close can cause Prometheus to receive multiple aggregations from the same Worker Process for the same time period. Prometheus cannot distinguish between them and will ingest only the first one.

5. Test that you configured the Function correctly by comparing sample incoming data with outgoing data in the Data Preview pane and Pipeline Diagnostics tool. See Data Preview and Metrics View for more information.

Usage Examples

This section provides typical JavaScript expressions that you might commonly use with the Drop Dimensions Function.

Filter Expressions

The **Filter** setting is a JavaScript expression that selects which data to process through the Drop Dimensions Function. The following is an example of a JavaScript expression that filters for a metric named `proc.cpu_perc` that contains the dimensions `proc`, `pie`, and `unit` and other variants that may include these dimension names:

```
(_metric == 'proc.cpu_perc' || __criblMetrics[0].nameExpr.includes("'proc.cpu_perc'")
(_metric == 'proc.cpu_perc' || __criblMetrics[0].nameExpr.includes("'proc.cpu_perc'")
(_metric == 'proc.cpu_perc' || __criblMetrics[0].nameExpr.includes("'proc.cpu_perc'")
```

Copy

Drop Dimensions

You can use the **Drop dimension** setting to add or edit the list of dimension names to drop. This field is not a JavaScript expression. Instead it uses a lightweight syntax that accepts simple strings and operators, including wildcard characters.

For most dimension names, you can use the syntax `'dimensionName'` to access dimension names. For example:

```
'proc'
'pie'
'unit'
'!proc'
'proc*'
'unit*'
'!*'
'*'
```

Copy

7.12. Dynamic Sampling

The Dynamic Sampling Function filters out events based on an expression, a sample mode, and the volume of events. Your sample mode's configuration determines what percentage of incoming events will be passed along to the next step.

Each Worker Process executes this Function independently on its share of events. For details, see Functions and Shared-Nothing Architecture.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Sample mode: Defines how sample rate will be derived. For formulas and usage details, see Sample Modes below. Supported methods:

- Logarithmic (the default): `log(previousPeriodCount)`.
- Square root: `sqrt(previousPeriodCount)`.

Sample group key: Expression used to derive sample group key. For example: `${domain}:${httpCode}`. Each sample group will have its own derived sampling rate, based on the volume of events. Defaults to `` `${host}` ``.

All events without a host field passing through the Function will be associated with the same group and sampled the same.

Advanced Settings

- **Sample period Sec**: How often (in seconds) sample rates will be adjusted. Defaults to `30`.
- **Minimum events**: Minimum number of events that must be received, in previous sample period, for sampling mode to be applied to current period. If the number of events received for a sample group is less than this minimum, a sample rate of 1:1 is used. Defaults to `30`.
- **Sampling rate limit**. Maximum sampling rate. If the computed sampling rate is above this value, the rate will be limited to this value.

How Does Dynamic Sampling Work

Compared to static sampling, where users must first select a sample rate, Dynamic Sampling allows for **automatically adjusting** sampling rates, based on the volume of incoming events per sample group. This Function allows users to set only the aggressiveness/coarseness of this adjustment. Square Root is more aggressive than Logarithmic mode.

As an event passes through the Function, it's evaluated against the Sample Group Key expression to determine the sample group it will be associated with. For example, given an event with these fields: `...ip=1.2.3.42, port=1234...`, and a Sample Group Key of `` `${ip}:${port}` ``, the event will be associated with the `1.2.3.42:1234` sample group.

If the Sample Group Key is left at its `` `${host}` `` default, all events without a host will be associated with the same group and sampled the same.

When a sample group is new, it will initially have a sample rate of `1:1` for `Sample Period` seconds (this value defaults to 30 seconds). Once `Sample Period` seconds have elapsed, a sample rate will be derived based on the configured `Sample Mode`, using the sample group's event volume during the **previous** sample period.

For example, assuming a Logarithmic Sample Mode:

Period 0 (first 30s) – Number of events in sample group: `1000`, Sample Rate: `1:1`, Events allowed: `ALL` Sample Rate calculation for **next** period: `Math.ceil(Math.log(1000)) = 7`

Period 1 (next 30s) – Number of events in sample group: `4000`, Sample Rate: `7:1`: Events allowed: `572` Sample Rate calculation for **next** period: `Math.ceil(Math.log(4000)) = 9`

Period 2 (next 30s) – Number of events in sample group: `12000`, Sample Rate: `9:1`: Events allowed: `1334` Sample Rate calculation for **next** period: `Math.ceil(Math.log(12000)) = 10`

Period 3 (next 30s) – Number of events in sample group: `2000`, Sample Rate: `10:1`: Events allowed: `200` Sample Rate calculation for **next** period: `Math.ceil(Math.log(2000)) = 8`
...

Sample Modes

1. Logarithmic - The sample rate is derived, for each sample group, using a natural log: `Math.ceil(Math.log(lastPeriodVolume))`. This mode is **less aggressive**, and drops fewer events.
2. Square Root - The sample rate is derived, for each sample group, using: `Math.ceil(Math.sqrt(lastPeriodVolume))`. This mode is **more aggressive**, and drops more events.

Example

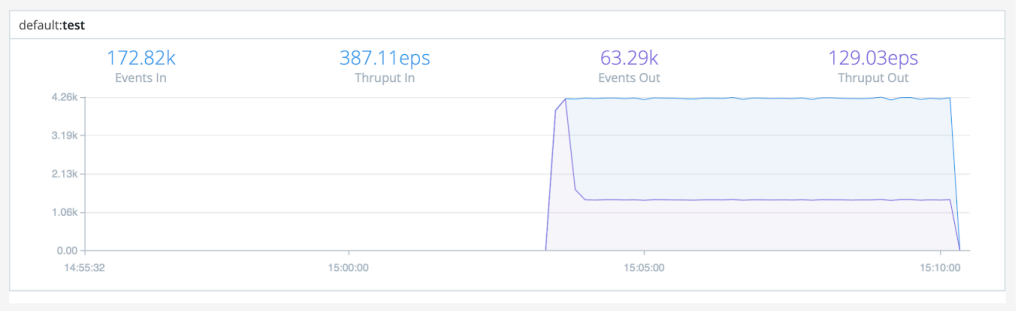
Here's an example that illustrates the effectiveness of using the Square Root sample mode.

Settings:

Sample Mode: `Square Root` Sample Period (sec): `20` Minimum Events: `3` Max. Sampling Rate: `3`

Results:

Events In: 4.23K Events Out: 1.41K



In this generic example, we reduced the incoming event volume from 4.23K to 1.41K. Your own results will vary depending on multiple parameters - the **Sample Group Key**, **Sample Period**, **Minimum Events**, **Max Sampling Rate**, and rate of incoming events.

For further examples, see Getting Smart and Practical With Dynamic Sampling.

See Also

- Sampling Function: Filter out events based on an expression and a sampling rate.

7.13. Eval

The Eval Function adds, modifies, or removes event fields. Eval is optimized for simple, single-value assignments and transformations using JavaScript expressions. For complex logic or multi-step processing, use the Code Function.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Evaluate fields: Table of event fields to evaluate and add/set. Select **Add Field** to add each field as a key-value pair, in a row with the following controls:

- **Name**: Enter the new (or modified) field's name.
- **Value Expression**: Enter a JavaScript expression to set the field's value - this can be a constant. Expressions can contain nested addressing. When you insert JavaScript template literals, strings intended to be used as values must be delimited with single quotes, double quotes, or backticks. For specific constraints, see Value Expression Limitations. For details on fields with non-alphanumeric characters, see Build Custom Logic to Route and Process Your Data: Fields with Non-Alphanumeric Characters.
- **Enabled**: Toggle this off to disable evaluating individual expressions, while retaining their configuration in the table. Useful for iterative development and debugging.

Keep fields: List of fields to keep in the event after processing by this Function. Takes precedence over the **Remove fields** setting. The field accepts wildcards (`*`), which supports both:

- Wildcards (`*`) to include multiple fields.
- Nested addressing, where you can reference a parent field and all its children using a wildcard. For example, if you convert `_raw` into an object, use `_raw*` to refer to `_raw` and all nested fields inside it. See Wildcard Lists for more information.

Remove fields: List of fields to remove from the event. Supports wildcards (*) and nested addressing. Cannot remove fields that match against the **Keep fields** list. Cribl Stream internal fields that start with `__` (double underscore) **cannot** be removed via wildcard. Instead, you must specify them individually. For example, `__myField` cannot be removed by specifying `__myF*`.

Value Expression Limitations

The **Value Expression** supports simple, single-value assignments and transformations. For advanced logic and complex operations, use the Code Function.

The following limitations apply to expressions:

1. **Reserved Words**: Expressions cannot utilize JavaScript **reserved words** like `constructor`, `prototype`, and `__proto__`.
   Cribl responds with: `Error: Expression cannot make use of the reserved word '<word>'.`
2. **Setters**: The definition or use of JavaScript **setters** is not allowed within expressions.
   Cribl responds with: `Error: Expression cannot use setter.`
3. **Assignment Operators**: Assignment operators like `=`, `+=`, `-=` are not permitted, except for direct assignment to the configured field. For example, `myField = <expression>`.
   Cribl responds with: `Error: Unallowed assignment operator found in expression.`
4. **Nested Scopes**: Expressions cannot introduce **nested scopes**. This includes, but is not limited to:
   - Loops (`for`, `while`, `do-while`)
   - Array iteration methods (`.map`, `.forEach`, `.reduce`, and so on)
   - Control structures (`if`, `switch`, `try-catch`)

   Cribl responds with: `Error: Expression cannot have nested scopes.`
5. **Internal Expressions**: Expressions are prevented from referencing internal Cribl expressions, such as `C.internal`.
   Cribl responds with: `Error: Expression cannot reference 'C.internal'.`
6. **Declarations**: Variable or function declarations are not allowed. This includes:
   - Variable declarations (`let`, `const`, `var`)
   - Function definitions (function declarations, arrow functions, function expressions)

   Cribl responds with: `Error: Expression cannot have any declarations.`

Using Keep and Remove

A field matching an entry in *both* **Keep** (wildcard or not) and **Remove** will *not* be removed. This is useful for implementing "remove all but" functionality. For example, to keep only `_time, _raw, source, sourcetype, host`, we can specify them all in **Keep**, while specifying `*` in **Remove**.

Negated terms are supported in both **Keep fields** and **Remove fields**. The list is order-sensitive when negated terms are used. Examples:

- `!foobar, foo*` means "All fields that start with 'foo' except `foobar`."
- `!foo*, *` means "All fields except for those that start with 'foo'."

Examples

Note that Functions use the special variable `__e` to access the `(context)` event inside JavaScript expressions.

Scenario A: Create field `myField` with static value of `value1`:

- **Name:** `myField`
- **Value Expression:** `'value1'`

Scenario B: Set field `action` to `blocked` if `login==error`:

- **Name:** `action`
- **Value Expression:** `login=='error' ? 'blocked' : action`

Scenario C: Create a multivalued field called `myTags` (array):

- **Name:** `myTags`
- **Value Expression:** `['failed', 'blocked']`

Scenario D: Add value `error` to the multivalued field `myTags`:

- **Name:** `myTags`
- **Value Expression:** `login=='error' ? [...myTags, 'error'] : myTags`

(The above expression is literal, and uses JavaScript spread syntax.)

Scenario E: Rename an `identification` field to the shorter `ID` - copying over the original field's value, and removing the old field:

- **Name**: `ID`
- **Value Expression**: `identification`
- **Remove Field**: `identification`

See Also

For more usage examples, see these Better Practices topics:

- Ingest-time Fields, which demonstrates both adding and removing fields.
- Reducing Windows XML Events, which demonstrates using Eval with the `C.Text.parseWinEvent` expression method to parse `_raw` Windows XML events into prettified JSON objects.

Usage Notes

Consider the following when working with Eval Functions.

Create Parent Objects First

Before you can use the Eval function on a new child object, you must create the parent object - then define the children using Eval Functions.

Example:

```
parent  =  (parent || { child1: child1Value })
parent.child2 = child2Value
parent.child3 = child3Value
<some other Evals, if you need them>
```

Copy

To Append:

```
parent =  Object.assign(parent, { child2: child2Value })
```

Copy

To Create a New Field:

```
parent2 =  Object.assign(parent, { child2: child2Value, child3: child3Value })
```

Copy

Execution Without Assignment

The Eval Function can execute expressions **without** assigning their value to the field of an event. You can do this by simply leaving the left-hand side input empty, and having the right-hand side do the assignment.

Example: Parse and Merge to Existing Field

`Object.assign(foo, JSON.parse(bar), JSON.parse(baz))` on the right-hand side (and left-hand side empty) will JSON-parse the strings in `bar` and `baz`, merge them, and assign their value to `foo`, an already existing field.

Example: Reference Event with `__e`

To parse JSON, enter `Object.assign(__e, JSON.parse(_raw))` on the right-hand side (and left-hand side empty). `__e` is a special variable that refers to the `(context)` event **within** a JavaScript expression. In this

case, content parsed from `_raw` is added at the top level of the event.

7.14. Event Breaker Function

This Function enables you to split large blobs or streams of events into discrete events within a Pipeline. This is useful for Sources like Azure Event Hubs, which do not natively support Event Breakers.

Even with Sources that do support Event Breakers (like Raw HTTP), it sometimes makes sense to use both a Source-configured Event Breaker on the incoming stream, and an Event Breaker Function within a Pipeline.

Limitations

The Event Breaker Function operates only on data in `_raw`. For other events, move the array to `_raw` and stringify it before applying this Function.

The largest event that this Function can break is about about 128 MB (`134217728` bytes). Events exceeding this maximum size will be split into separate events, but left unbroken. Cribl Stream will set these events' `__isBroken` internal field to `false`.

Unlike regular Event Breakers, Event Breaker Functions do not have names, only descriptions.

Usage

Use the following options to define this Event Breaker Function.

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Existing or new?: Whether to use an existing ruleset or create a new one. Defaults to `Use Existing`.

- When **Existing or new?** is set to `Use Existing`, the **Existing ruleset** drop-down appears and you choose a ruleset from there.
- When **Existing or new?** is set to `Create New`, the ruleset creation UI appears and you configure its settings.

Advanced Settings

Add to cribl_breaker: Whether to add the `cribl_breaker` field to output events. Defaults to toggled on.

How this field behaves depends on whether you are **also** using a regular Event Breaker with your Source. That matters because a regular Event Breaker **always** adds the `cribl_breaker` field to events, which it does **before** the data reaches your Event Breaker Function.

When you are **not** using a regular Event Breaker, there is no `cribl_breaker` field yet when the data reaches your Event Breaker Function. At this point, `cribl_breaker` is added, and:

- When **Existing or new?** is set to `Use Existing`, `cribl_breaker`'s value is set to the name of the existing ruleset you chose.
- When **Existing or new?** is set to `Create New`, the `cribl_breaker`'s value is set to `"event_breaker_func"`. Since Event Breaker Functions do not have names, the string `event_breaker_func` serves as a kind of generic name that represents whatever new Event Breaker Function you created.

When you **are** also using a regular Event Breaker, the `cribl_breaker` field already exists when the data reaches your Event Breaker Function. At this point, the value of `cribl_breaker` is changed from a string to an array, with the first item in the array being the value originally set by the regular Event Breaker, and the second item determined as follows:

- When **Existing or new?** is set to `Use Existing`, the second item's value is set to the name of the existing ruleset you chose.
- When **Existing or new?** is set to `Create New`, the second item's value is set to `"event_breaker_func"`. Since Event Breaker Functions do not have names, the string `event_breaker_func` serves as a kind of generic name that represents whatever new Event Breaker Function you create.

Cribl Stream truncates timestamps to three-digit (milliseconds) resolution, omitting trailing zeros.

In Cribl Stream 3.4.2 and above, where an Event Breaker Function has set an event's `_time` to the current time – rather than extracting the value from the event itself – it will mark this by adding the internal field `__timestampExtracted: false` to the event.

Examples

Handling syslog data and nested JSON data are two primary use cases for Event Breaker Functions.

Event Breaker Functions for syslog Data

Event Breaker Functions can help you deal with syslog data that does not break correctly. Examples include multi-line syslog data, and, the kinds of non-standard syslog data emitted by Blue Coat proxy appliances, Layer7 API Gateways, and other appliances. See the Cribl video Scaling syslog for an in-depth discussion.

(In this context, “non-standard” means syslog data that only partly conforms to the standard Syslog Protocol defined in RFC 5424, or its predecessor, the BSD syslog Protocol defined in RFC 3164.)

Event Breaker Functions for Nested JSON Data

Kafka-based data from Confluent, Azure Event Hubs, Google Pub Sub, or Kafka itself, is nicely structured as events according to the Kafka protocol. But when these events contain JSON objects which you want to break into their constituent objects, you can use an Event Breaker Function to do that.

Troubleshooting

If you notice fragmented events, check whether Cribl Stream has added a `__timeoutFlush` internal field to them. This diagnostic field’s presence indicates that the events were flushed because the Event Breaker buffer timed out while processing them. These timeouts can be due to large incoming events, backpressure, or other causes.

7.15. Flatten

The Flatten Function pulls fields out of a nested structure, moving nested key-value pairs (fields) up to a higher level in the object. You can specify:

- Individual fields to flatten.
- The depth at which to flatten fields.
- The delimiter to use when concatenating keys.
- An optional prefix to add to transformed field names.

The Flatten Function creates fully qualified names for promoted fields. If you simply need to promote fields without transforming their names, use the `eval` Function.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Fields: List of top-level fields to include for flattening. Defaults to an empty array, which means all fields. Limit to specific fields by typing in their names, separated by hard returns. Supports wildcards (`*`). Supports double-underscore (`__`) internal fields only if individually enumerated - not via wildcards.

Prefix: Prefix string for flattened field names. Defaults to empty.

Depth: Number representing the nested levels to consider for flattening. Minimum `1`. Defaults to `5`.

Delimiter: Delimiter to use for flattening. Defaults to `_` (underscore).

Example

Add the following test sample in **Preview** > **Paste a Sample**:

```
input
```

```
{
        "accounting": [{
                "firstName": "John",
                "lastName": "Doe",
                "age": 23
        }, {
                "firstName": "Mary",
                "lastName": "Smith",
                "age": 32
        }],
        "sales": [{
                "firstName": "Sally",
                "lastName": "Green",
                "age": 27
        }, {
                "firstName": "Jim",
                "lastName": "Galley",
                "age": 41
        }]
}
```

Copy

Under **Select Event Breaker**, choose **ndjson** (newline-delimited JSON), and select **Save as a Sample File**.

Here's sample output with all settings at default:

output

```
{
  "accounting_0_firstName": "John",
  "accounting_0_lastName": "Doe",
  "accounting_0_age": 23,
  "accounting_1_firstName": "Mary",
  "accounting_1_lastName": "Smith",
  "accounting_1_age": 32,
  "sales_0_firstName": "Sally",
  "sales_0_lastName": "Green",
  "sales_0_age": 27,
  "sales_1_firstName": "Jim",
```

7.16. Fold Keys

The Fold Keys Function transforms flat field names that include a common separator (such as `.` or `/`) into a more logical, nested structure. Use this Function to normalize hierarchical data that expresses logical relationships through field names. The Fold Keys Function restructures the field names into grouped, nested objects without changing their values.

Use this Function to:

- **Normalize data from systems that encode hierarchy in field names:** Standardize keys from sources that encode similar data differently. For example, you could transform keys such as `http.status`, `http.method`, and `http.url` into a nested structure under `http`.
- **Reduce field name redundancy for more efficient storage and easier parsing:** Group similar fields under a parent key to reduce clutter and improve readability. With nested structures also give you the option to apply logic to an entire group (`http.*`) rather than individual fields using another Function in your Pipeline.
- **Improve schema compatibility or prepare data for JSON-based Destinations or downstream receivers that expect nested objects:** Structure your data for downstream consumption by tools such as Elasticsearch or certain APM platforms that expect hierarchical data structures for querying or visualizing.

See Usage Examples for an in-depth example of how this Function processes data.

How the Fold Keys Function Works

The Fold Keys Function is a simpler version of the Eval Function, designed specifically to group related fields by a common separator in their names. If you're familiar with Cribl Search, the Function behaves like the foldkeys operator, but for use within Cribl Stream.

The Fold Keys Function processes data in the following order:

1. **Search:** It looks for field names that contain the specified separator string (such as a period or underscore). If a field name does not include the separator, the Function skips it and passes it through unchanged.
2. **Group:** For each matching field name, the Function splits the name into parts based on the separator. It uses these parts to organize related fields under a shared top-level key.
3. **Nest:** The Function creates a nested data structure schema based on the grouped parts. For example, a field like `http.status.code` would become:

```
"http": {
  "status": {
    "code": 404
    }
  }
```

Copy

See Usage Examples for a practical, real-world example of how this Function processes data.

Configure the Fold Keys Function

To configure this Function:

1. Add a Source and Destination to begin generating data. See Integrations for more information.
2. Add a new Pipeline or open an existing Pipeline. Add or capture sample data test your Pipeline as you build it. See Pipelines for more information.
3. At the top of the Pipeline, select **Add Function** and search for `Fold Keys`, then select it.
4. In the **Fold Keys** modal, configure the following general settings:
    - **Filter:** A JavaScript filter expression that selects which data to process through the Function. Defaults to `true`, meaning it evaluates all data events. If you don't want to use the default, you can use a JavaScript expression to apply this Function to specific data events. See Build Custom Logic to Route and Process Your Data for examples.
    - **Description:** A brief description of how this Function modifies your data to help other users understand its purpose later. Defaults to empty.
    - **Final:** When enabled, stops data from continuing to downstream Functions for additional processing. Default: Off. Toggle **Final** on if Fold Keys is the last Function in your Pipeline.
    - **Delete original:** When toggled on (default), the data output only includes the folded field names, meaning it removes the original data from the raw location. When toggled off, the data output includes both the folded and original field names.
    - **Separator string:** Set a separator character to use as the path separator. For example: `/` or `_`. This separator determines how to group the fields. Defaults to `.` (dot).
    - **Selection regular expression (optional):** Input a regular expression. If defined, the Function only groups field names that match this expression, along with the separator string.
5. Test that you configured the Function correctly by comparing sample incoming data with outgoing data in the Data Preview pane and Pipeline Diagnostics tool. See Data Preview for more information.

Usage Examples

The following is an example of raw data before applying the Fold Keys Function:

```
{
  "_raw": "Mar 27 23:45:05 PA-VM 2020/05/07 02:40:14,44A1B3FC68F5304,TRAFFIC,end,192.
  "_time": 1743144305,
  "source": "cribl",
  "data_metrics_bytes": 12345678,
  "data_metrics_count": 987,
  "data_metrics_latency_p95": 123,
  "data_metrics_latency_p90": 56,
  "host_ip": "203.0.113.0"
}
```

Copy

In this example, there are multiple fields that begin with `data_metrics`. You could use the Fold Keys Function to group these field names together into a more logical, nested data structure.

In this example, the Fold Keys Function uses these settings:

- **Delete original:** Toggled on
- **Separator string:** `_`
- **Selection regular expression:** `^data`

Together, these settings nest all fields that start with `data_` and creates two hierarchies using the underscore `_` character as the separator string. The Regular Expression `^data` matches any string that starts with the word `data`.

After the Fold Key Function processes the data, the output looks like this:

```
{
  "_raw": "Mar 27 23:45:05 PA-VM 2020/05/07 02:40:14,44A1B3FC68F5304,TRAFFIC,end,192.
  "_time": 1743144305,
  "source": "cribl",
  "data": {
    "metrics": {
      "bytes": 12345678,
      "count": 987,
```

7.17. GeoIP

The GeoIP Function enriches events with geographic fields, given an IP address. It works with MaxMind's GeoIP or IPinfo's IP address binary database.

Prerequisite

You need to host the `.mmdb` database file. The following steps cover this process at a high level. They link to our Lookup Geographic Data from IP Addresses topic, where you can find additional details.

1. Download and extract the `.mmdb` database file.
2. Determine where to place the database file. We recommend the `$CRIBL_HOME/state/` subdirectory, which is already listed in the default `.gitignore` file that ships with Cribl Stream.
   You always have the option to upload the file to Cribl Stream's Lookups Library. In a Cribl.Cloud deployment, this is currently the only option. For details, see About Lookups.
3. Place the database file on your Worker Node(s). In a distributed deployment, we recommend having the file on the Leader and all Worker Nodes. Smaller deployments can get away with hosting the file only on the Leader Node.
4. Optionally, set up automatic updates to the MaxMind and IPinfo database files.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

GeoIP file (.mmdb): Path to the database, in binary format, with `.mmdb` extension.

If the database file is located within the lookup directory (`$CRIBL_HOME/data/lookups/`), the **GeoIP file** does not need to be an absolute path.

In distributed deployments, ensure that the database file is in the same location on the Leader Node and all Worker Nodes. However, if you've uploaded `.mmdb` files via Cribl Stream's Lookups Library UI, just select this box to display and select them on a drop-down list. Your selection here will handle the path management automatically.

IP field: Field name in which to find an IP to look up. Can be nested. Defaults to `ip`.

Result field : Field name in which to store the GeoIP lookup results. Defaults to `geoip`.

Examples

Assume that you are receiving SMTP logs, and need to see geolocation information associated with IPs using the SMTP service.

Here's a sample of our data, from IPSwitch IMail Server logs:

```
03:19 03:22 SMTPD(00180250) [192.168.1.131] connect 74.136.132.88 port 2539 03:19 03:22 SMTPD(00180250) [74.136.132.88] EHLO msnbc.com 03:19 03:22 SMTPD(00180250) [74.136.132.88] MAIL FROM:<info-jjgcdshx@test.us> 03:19 03:22 SMTPD(00180250) [74.136.132.88] RCPT To:<user@domain.com>
```

In this example, we'll chain together three Functions. First, we'll use a Regex Extract Function to isolate the host's IP. Next, we'll use the GeoIP Function to look up the extracted IP against our geoIP database, placing the returned info into a new `__geoip` field. Finally we'll use an Eval Function to parse that field's city, state, country, ZIP, latitude, and longitude.

Function 1 - Regex Extract

Regex: `\[(?<ip>\S+)\]` Source field: `_raw` Result: `74.136.132.88`

Function 2 - GeoIP

Event's IP field: `ip` Result field: `__geoip`

Function 3 - Eval

Name	Value Expression
`City`	`__geoip.city.names.en`
`Country`	`__geoip.country.names.en`
`Zip`	`__geoip.postal.code`
`Lat`	`__geoip.location.latitude`
`Long`	`__geoip.location.longitude`

In the Eval Function's **Remove fields** setting, you could specify the `__geoip` field for removal, if desired. However, its `__` prefix makes it an internal field anyway.

For a hosted tutorial on applying the GeoIP Function, see Cribl's GeoIP and Threat Feed Enrichment Sandbox.

7.18. Grok

The Grok Function extracts structured fields from unstructured log data, using modular regex patterns.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Optional description of this Function's purpose in this Pipeline. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Pattern: Grok pattern to extract fields. You can use a Grok pattern from any of the configured files. Click the Expand button at right to open a preview/validation modal.
Syntax supported: `%{PATTERN_NAME:FIELD_NAME}`.

Select **Add pattern** to chain more patterns.

Source field: Field on which to perform Grok extractions. Defaults to `_raw`.

Management

You can add and edit Grok patterns via Cribl Stream's UI by selecting **Knowledge** > Grok Patterns. Pattern files are located at: `$CRIBL_HOME/(default|local)/cribl/grok-patterns/`

Example

Example event:

```
{"_raw": "2020-09-16T04:20:42.45+01:00 DEBUG This is a sample debug log message"}
```

Copy

Pattern: `%{TIMESTAMP_ISO8601:event_time} %{LOGLEVEL:log_level} %{GREEDYDATA:log_message}`
Source Field: `_raw`

Event after extraction:

```
{"_raw": "2020-09-16T04:20:42.45+01:00 DEBUG This is a sample debug log message",
  "_time": 1600226442.045,
  "event_time": "2020-09-16T04:20:42.45+01:00",
  "log_level": "DEBUG",
  "log_message": "This is a sample debug log message",
}
```

Copy

Note the new fields added to the event: `event_time`, `log_level`, and `log_message`.

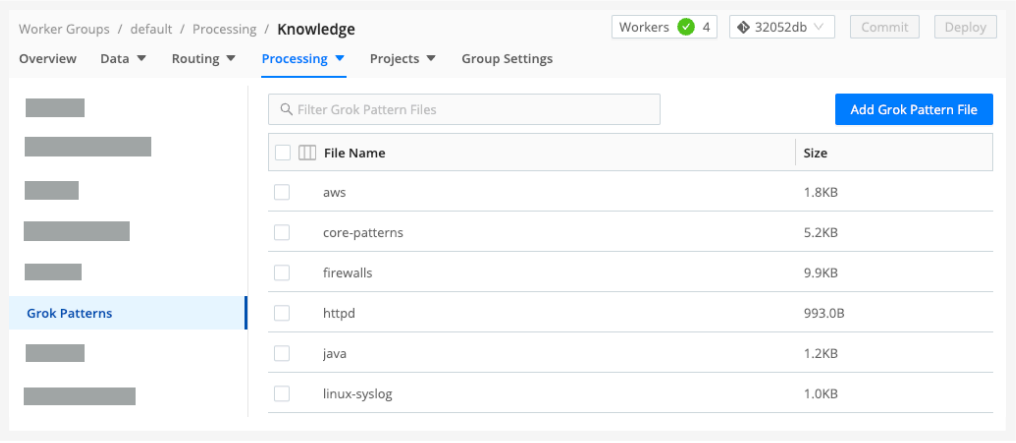
References

- Syntax for a Grok pattern is `%{PATTERN_NAME:FIELD_NAME}`. Example: `%{IP:client} %{WORD:method}`.
- Useful link for creating and testing Grok patterns: http://grokconstructor.appspot.com
- Additional patterns are available here: https://github.com/logstash-plugins/logstash-patterns-core/tree/main/patterns

Available Grok Patterns

Cribl Stream ships common Grok Patterns for basic scenarios that you can use with the Grok Function, organized as a collection of files.

You can refer to a single pattern in multiple Grok Functions in different Pipelines. When editing them, be aware that any change will affect all places it is used in.



Available Grok Patterns

To find the patterns, select **Worker Groups** from the sidebar, and choose a Worker Group. Then, on the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Grok Patterns**.

Add a Grok Pattern File

To add a new Grok pattern file, follow these steps:

1. In the sidebar, select **Worker Groups** from the sidebar, and choose a Worker Group or Fleet.
2. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Grok Patterns**.
3. Select **Add Grok Pattern File**.
4. Enter a unique **File name**.
5. Paste the content of the Grok file into **Content**.
6. Select **Save**.

7.19. GUARD

In Cribl Stream, the Cribl Guard Function allows you to dynamically scan incoming data for sensitive fields like personally identifiable information (PII) and then obfuscate those events in a Pipeline. This Function helps you identify sensitive data in your data streams and prevent it from being exposed in downstream analytics systems whose primary users don't need PII access.

Use this Function to:

- **Reduce PII in your data:** and thereby decrease your risk of legal risk in the event of a data breach.
- **Responsibly handle sensitive data:** in highly regulated industries like finance, healthcare, law firms, and technology.

How the Cribl Guard Function Works

The Cribl Guard Function processes data in the following order:

1. **Filter:** Searches events that match the JS filter and that are located within the **Apply to** fields and skips the **Fields to ignore**. Applies the rules in the selected scanning ruleset(s).
2. **Scan:** Looks for sensitive data in the matching filter and rules.
3. **Mitigate:** Applies mitigation to sensitive data based on the rules in your ruleset. For example, the Default ruleset redacts sensitive data values.

You can add the Cribl Guard Function to any Pipeline to manually scan incoming data streams. We recommend adding the Cribl Guard Function in your Post-Processing Pipeline.

Pipeline Recommendations for the Cribl Guard Function

There are no hard constraints around where in your Pipeline you add the Cribl Guard Function. However, there are considerations we suggest you take into account.

- We strongly recommend that you only use the Cribl Guard Function in **one** of your Pipelines (Post-Processing is our formal recommendation). This is because Cribl Guard is billed based on GB of data scanned. You could inadvertently be double- or triple-charged (or more) if you use the Cribl Guard Function in multiple Pipelines that scan the same set of data.

Post-Processing Pipeline

We highly recommend configuring the Cribl Guard Function in a **Post-Processing Pipeline**. Here's why:

- With the Cribl Guard Function in your Post-Processing Pipeline, sensitive data is mitigated before Cribl sends it to your Destinations. This means sensitive data is much less likely to reach your Destinations, reducing the risk of personal data leaks.
- You'll avoid accidentally incurring multiple charges on processed data. Cribl Guard charges based on how many GB of data it scans, so by placing the Guard Function in your Post-Processing Pipeline, you only scan data that you're certain you want to send to a Destination.
- Cribl Guard scans data in its final form: *after* data goes through the JavaScript filter in the Pipeline, is enriched, adjusted, enhanced, and so on. This ensures you're scanning the smallest subset of events.

Of course, by placing the Cribl Guard Function in a Post-Processing Pipeline, there is the possibility that Cribl might scan the same data multiple times as it routes to different Destinations.

Pre-Processing Pipeline

By configuring the Cribl Guard Function in your Pre-Processing Pipeline, you can quickly identify Sources that might be unreliable or untrustworthy. This enables you to filter and mask data as soon as Cribl Guard identifies it.

However, it's not possible to keep unredacted and redacted data separate when you use Cribl Guard in a Pre-Processing Pipeline. Also, you might end up scanning (and paying for) data that you discard later in the Pipeline.

Configure the Cribl Guard Function

1. Add a Source and Destination to begin generating data. See Integrations for more information.
2. Add a new Pipeline or open an existing Pipeline. Add or capture sample data and test your Pipeline as you build it. See Pipelines for more information.
3. At the top of the Pipeline, select Add Function and search for Cribl Guard, then select it.
4. In the resulting Cribl Guard Function modal, configure the following general settings:
    - **Filter:** A JavaScript filter expression that selects which metrics to process through the Function. Defaults to `true`, meaning it evaluates all events.
    - **Description:** A brief description of how this Function modifies your metrics to help other users understand its purpose later. Defaults to empty.
    - **Final:** When enabled, stops data from continuing to downstream Functions for additional processing. Default: Off. Toggle **Final** on if Cribl Guard is the last Function in your Pipeline.
    - **Scanning Rulesets:** The Rulesets in this Pipeline that scan incoming events for sensitive data. You can add as many Scanning Rulesets as you like to a Pipeline.
        - **Ruleset ID:** The unique ID of the ruleset used in this scan.
        - **Mitigation Expression:** Content that matches the scanning rule will be replaced by a new value defined in this mitigation expression, which can be either a JavaScript expression (that

manipulates the data) or a literal (like 'REDACTED'). You can use capturing groups to isolate specific parts of matching text, then reference them in your mitigation expression as `g1`, `g2`, and so on.

        - **Enabled:** You can enable or disable scanning rulesets within a Pipeline.
    - **Apply to fields:** Cribl Guard will only scan data that is included in the fields you designate here.
    - **Fields to ignore:** Cribl Guard will not scan data that is included in the fields you designate here.
    - **Advanced Settings**
        - **Add fields:** Enter fields that Cribl Guard will add to mitigated events. `_sensitive` is `true` by default, which means it will appear in event data when sensitive fields have been mitigated.
        - **Add matched rulesets:** When enabled, Cribl adds an internal field called `__detected` to the incoming event. This field will contain the ruleset ID or array of IDs that match the sensitive data. For example, `__detected: rule1` or `__detected: [rule1, rule2 ....ruleN]`.
5. Use the Simple Preview or Full Preview pane combined with a sample data file to test that your Function is accurately scanning, filtering, and mitigating any sensitive data.
6. Send the newly configured Pipeline Function to your Stream Workers using **Commit & Deploy**.

Cribl Guard uses rulesets, rather than relying on rigid pattern matching, which results in faster, more accurate data processing. You can use a combination of out-of-the-box rulesets and custom rulesets. To see all the Cribl Guard rulesets included out-of-the-box, go to the Cribl Guard Rulesets reference documentation.

Event Fields and Internal Fields

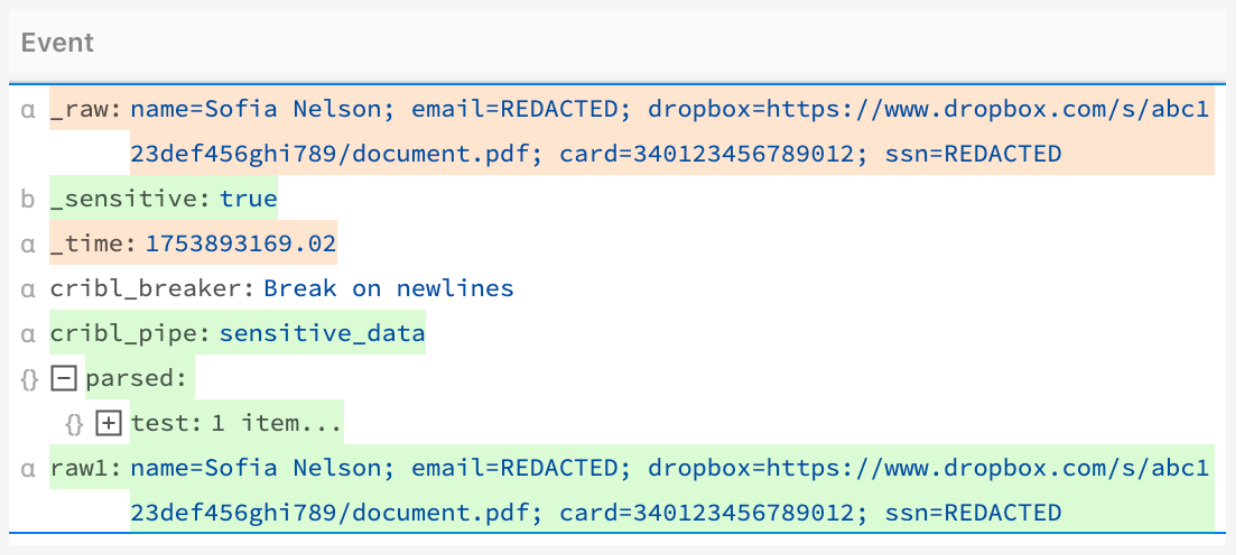
There are four fields that the Cribl Guard Function can add to your events:

- `_sensitive`: Included by default in **Advanced Settings**, this field is added to mitigated event data.
- `__detected`: When **Add to matched rulesets** is enabled, your events will contain this internal field.
- `__potential_sensitive`: This internal field is added to event data when a data string matches the **Pattern to match** expression but does not match an anchor term. You can then manually adjust the match pattern or anchor terms to mitigate the data if it is sensitive.
- `__potential_sensitive_data`: This internal field shows you which events are potentially sensitive.

You must enable **Show internal fields** in the preview settings to see the `__potential_sensitive` and `_potential_sensitive_data` fields in your data preview. See Data Preview Advanced Settings for setup steps.

Usage Examples

A common usage example for Cribl Guard is to redact plain text as replacement expressions for the sensitive data. For example, this Pipeline uses the Default ruleset, which applies `REDACTED` as the mitigation expression to matching sensitive data.



Example Guard mitigation

7.20. JSON Unroll

The JSON Unroll Function processes a JSON object string in the `_raw` field. It unrolls (explodes) an array of objects within this JSON into individual events, while inheriting the top-level fields from the original JSON object. This function is ideal for scenarios where processing individual items within an array is required while retaining the context provided by parent fields.

For efficiency, we recommend avoiding the JSON Unroll Function for certain types of data, such as CloudTrail, Office 365, or Kubernetes CloudWatch logs, which are often collected as JSON arrays. Instead, use an Event Breaker for inputs that support it. However, Event Breakers like JSON Array do not retain parent fields from the original JSON object. If retaining parent fields is critical, use the JSON Unroll Function or use a pre-processing Pipeline to extract parent fields before breaking events.

How the Function Works

When JSON Unroll runs, it parses the JSON object string in `_raw` into an internal field named `__json`. This internal field contains the full parsed structure of the event, including the array that you configure the Function to unroll. For example, if your array field is named `data`, the parsed JSON is available as `__json.data`, which you can reference in downstream Functions without reparsing `_raw`.

JSON Unroll then:

1. Locates the array at the configured **Path** within `__json`.
2. Iterates over each element in that array.
3. For each element, creates a new event that:
    - Inherits the top-level fields from the original JSON object. For example, `date`, `name`, or `age`.
    - Adds the current array element under the field specified in **New name**, or under its original field name if **New name** is left empty.
    - Serializes the resulting object back into `_raw` for the new event.

To inspect the parsed JSON and verify how JSON Unroll is working, enable **Show Internal Fields** in Data Preview. This exposes `__json` and other internal fields so you can see the parsed structure the Function is operating on.

Configuration

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Path: Path to array to unroll, such as `foo.0.bar`. This path is resolved against the parsed JSON stored in `__json`.

New name: The name that the exploded array element will receive in each new event. Leave empty to expand the array element with its original name.

Example

Assume you have an incoming event that has a `_raw` field as a JSON object string like this:

Sample _raw field

```
{"date":"9/25/18 9:10:13.000 PM",
   "name":"Amrit",
   "age":42,
   "allCars": [
       { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
       { "name":"GM", "models":[ "Trans AM", "Oldsmobile", "Cadillac" ] },
       { "name":"Fiat", "models":[ "500", "Panda" ] },
       { "name":"Blackberry", "models":[ "KEY2", "Bold Touch 9900" ] }
   ]
}
```

Copy

With the following settings configured:

- **Path**: `allCars`
- **New Name**: `cars`

Your output events would look like:

Resulting Events

```
Event 1:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"For

Event 2:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"GM"

Event 3:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"Fia

Event 4:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"Bla
```

Copy

Each element under the original **allCars** array is now placed in a **cars** field in its own event, inheriting original top level fields: **date**, **name**, and **age**. The parsed representation of each output event is also available under the internal `__json` field, which you can use in downstream Functions instead of reparsing `_raw`.

See Also

- The Cribl Knowledge Pack provides sample Pipelines that demonstrate converting a JSON string into an object literal, and validating JSON data against a schema.

Troubleshoot High Memory Usage or Slow Throughput

Issue: Pipelines are sluggish or Worker Nodes are hitting memory limits when processing Azure flow data.

Cause: Using the JSON Unroll Function on highly aggregated logs (like Azure VNet Flow logs) forces the Pipeline to hold massive objects in memory before they are split.

7.21. Lookup

The Lookup Function enriches events with external fields, using lookup table files in CSV, compressed `.csv.gz`, or binary `.mmdb` format.

CSV files are text-based and untyped. This means all data within the file is treated as strings, regardless of its intended type (like, numbers or dates).

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Lookup file path (.csv, .csv.gz): Path to the lookup file. Select an existing file that you've uploaded via Cribl Stream's UI at **Knowledge > Lookups**, or specify the path. You can reference environment variables via `$`, such as `$CRIBL_HOME/file.csv`.

When you configure this field via a distributed deployment's Leader Node, Cribl Stream will swap `$CRIBL_HOME/groups/<groupname>/` for `$CRIBL_HOME` when validating whether the file exists. In this case, the default upload path changes from `$CRIBL_HOME/data/lookups` (single-instance deployments) to `$CRIBL_HOME/groups/<groupname>/data/lookups/` (distributed deployments).

Use Disk-Based Lookup: Default: off. Toggle on if the lookup file referenced in the **Lookup file path** is a disk-based lookup. When toggled on, this setting removes the **Match Mode** and **Reload Period (sec)** options, as they do not apply to disk-based lookups.

Cribl Edge does not support disk-based lookups, only in-memory lookups.

Match mode: Defines the format of the lookup file, and indicates the matching logic that will be performed. Defaults to `Exact`. This option is only available for in-memory lookups and is disabled when **Use Disk-Based Lookup** is selected.

Reload period (sec): For single-instance deployments, changes to lookup files require an application restart unless a reload interval is enabled. Specify how often (in seconds) Cribl Stream should check for updates and

reload the file. Use a positive number to enable automatic reloading. Leave the `-1` (default) to disable. This option is only available for in-memory lookups and is disabled when **Use Disk-Based Lookup** is selected.

In distributed deployments, enabling a **Reload period** can generate conflicts with configuration updates, causing Pipelines to skip executing some Lookup Functions. Cribl recommends that you enable it only for lookup files not managed by Cribl Stream's UI, or lookup files that can be updated by an external process (such as a threat list that you update via a cron job).

For lookup files that **are** managed by Cribl Stream's UI, a distributed Cribl Stream deployment will override this setting as necessary - skipping checks to prevent conflicts that could trigger skipped lookups. These restrictions do **not** apply to single-instance deployments.

Match type: For CIDR and Regex **Match mode**s, this attribute refines how to resolve multiple matches. `First match` will return the first matching entry. `Most specific` will scan all entries, finding the most specific match. `All` will return all matches in the output, as arrays. (Defaults to `First match`. Not displayed for Exact **Match mode**.)

Lookup fields (.csv): Field(s) that should be used to key into the lookup table.

- **Lookup Field Name in Event**: Exact field name as it appears in events. Nested addressing supported.
- **Corresponding Field Name in Lookup**: The field name as it appears in the lookup file. Defaults to the **Lookup field name in event** value. This input is optional.

For disk-based lookups, the field you reference in the Lookup Function must be indexed, or the lookup will fail. Make sure all lookup key fields are properly indexed before deploying. See Add Indexes to a Lookup File for more information.

Case-Sensitive / Multiple Matches

Lookups are case-sensitive by default. (See the **Ignore case** option below.)

If the lookup file contains duplicate key names with different values, all **Match mode**s of this Function will use **only** the value in the key's **final** instance, ignoring all preceding instances.

Output field(s): Field(s) to add to events after matching the lookup table. Defaults to **all** if not specified.

- **Output Field Name from Lookup**: Field name, as it appears in the lookup file.
- **Lookup Field Name in Event**: Field name to add to event. Defaults to the lookup field name. This input is optional. Nested addressing is supported.
- **Default Value**: Optional string value to assign to the field when the lookup entry is not found.

Advanced Settings

Ignore case: Ignore case when performing **Match mode: Exact** lookups. Default is toggled off.

Add to raw event: Whether to append the looked-up values to the `_raw` field, as key=value pairs. Default is toggled off.

Examples

Example 1: Regex Lookups

Assign a `sourcetype` field to events if their `_raw` field matches a particular regex.

paloalto.csv

```
regex,sourcetype
"^[^,]+,[^,]+,[^,]+,THREAT",pan:threat
"^[^,]+,[^,]+,[^,]+,TRAFFIC",pan:traffic
"^[^,]+,[^,]+,[^,]+,SYSTEM",pan:system
```

Copy

Match mode: Regex

Match type: First match

When using the Lookup Function with Regex and `First match`, ensure that your lookup file contains no empty lines - not even at the bottom. Any empty row will cause the function to always return `true`.

Lookup field name in event: `_raw`

Corresponding field name in lookup: `regex`

Events before and after

```
### BEFORE:

{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,THREAT,end,2049,2018/09/


### AFTER:

{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09
  "sourcetype": "pan:traffic"
  }
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,THREAT,end,2049,2018/09/
  "sourcetype": "pan:threat"
  }
```

Copy

Example 2: CIDR Lookups

Assign a `location` field to events if their `destination_ip` field matches a particular CIDR range.

paloaltoips.csv

```
range,location
10.0.0.0/24,San Francisco
10.0.0.0/16,California
10.0.0.0/8,US
```

Copy

Match mode: CIDR

Match type: See options below

Lookup field name in event: `destination_ip`

Corresponding field name in lookup: `range`

In **Match mode: CIDR** with **Match type: Most specific**, the lookup will implicitly search for matches from most specific to least specific. There is no need to pre-sort data.

Note that **Match mode: CIDR** with **Match type: First Match** is likely the most performant with large lookups. This can be used as an alternative to **Most specific**, if the file is sorted with the most

specific/relevant entries first. This mode still performs a table scan, top to bottom.

Events before and after

```
### BEFORE:

{"_raw": "Sep 20 13:03:55 PA-VM 1, 2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/0
  "destination_ip": "10.0.0.102"
  }

### AFTER with Match Type: First Match

{"_raw": "Sep 20 13:03:55 PA-VM 1, 2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/0
  "destination_ip": "10.0.0.102",
  "location": "San Francisco"
  }

### AFTER with Match Type: Most Specific

{"_raw": "Sep 20 13:03:55 PA-VM 1, 2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/0
  "destination_ip": "10.0.0.102",
  "location": "San Francisco"
  }

### AFTER with Match Type: All

{"_raw": "Sep 20 13:03:55 PA-VM 1, 2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/0
  "destination_ip": "10.0.0.102",
  "location": [
    "San Francisco",
    "California",
    "US",
  ]}
```

Copy

See Also

- About Lookups
- Configure Lookups

7.22. Mask

The Mask Function replaces patterns in events, a process called *masking*. Masking is especially useful for redacting PII (personally identifiable information) and other sensitive data.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Optionally, enter a simple description of this Function.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Masking rules: This table defines pairs of patterns to match and replace. Defaults to empty. Select **Add Rule** to add each rule. Each row of the resulting table provides the following options:

- **Match Regex**: Pattern to replace. Supports capture groups. Use `/g` to replace all matches, like so: `/foo(bar)/g`. You can type in a pattern, or use one of the predefined Regexes.
- **Replace Expression**: A JavaScript expression or literal to replace all matching content. Capture groups can be referenced with `g` and the group number – for example, `g2` to reference the second capture group.
- **Enabled**: Toggle this off to disable matching individual rules, while retaining their configuration in the table. Useful for iterative development and debugging.

Apply to fields: Fields on which to apply the masking rules. Defaults to `_raw`. Add more fields by typing in their names, separated by hard returns. Supports wildcards (`*`) and nested addressing. Supports double-underscore (`__`) internal fields only if individually enumerated - not via wildcards.

Negated terms are supported. When you negate field names, the fields list is order-sensitive. For example, `!foobar` before `foo*` means "Apply to all fields that start with `foo`, except `foobar`." However, `!foo*` before `*` means "Apply to all fields, except for those that start with `foo`."

Advanced Settings

Evaluate fields: Optionally, specify fields to add to events in which one or more of the **Masking rules** were matched. These fields can be useful in downstream processing and reporting. You specify the fields as key-value expression pairs, like those in the Eval Function.

- **Name**: Field name.

- **Value Expression**: JavaScript expression to compute the value (can be a constant).

Depth: The depth to which the Function will traverse nested properties in an event. Defaults to 5.

Evaluating the Replace Expression

The **Replace expression** field accepts a full JS expression that evaluates to a value, so you're not necessarily limited to what's under `C.Mask`. For example, you can do conditional replacement: `g1%2==1 ? `fieldA="odd"` : `fieldA="even"``

The **Replace expression** can reference other event fields as `event.<fieldName>`. For example, `` `${g1}${event.source}` ``. Note that this is slightly different from other expression inputs, where event fields are referenced without `event.` Here, we require the `event.` prefix for the following reasons:

- We don't expect this to be a common case.
- Expanding the event in the replace context would have a high performance hit on the common path.
- There is a slight chance that there might be a `gN` field in the event.

The output of the Mask Function is a string even when the **Replace expression** produces a number.

For example, suppose a **Replace expression** that generates a random integer is applied to a field named `bytes` and produces the number `8403`. Cribl Stream will output an event where the value of the `bytes` field is the **string** `8403`.

If you need to convert the string value back to a number, you could add a Numerify Function to your Pipeline.

Examples

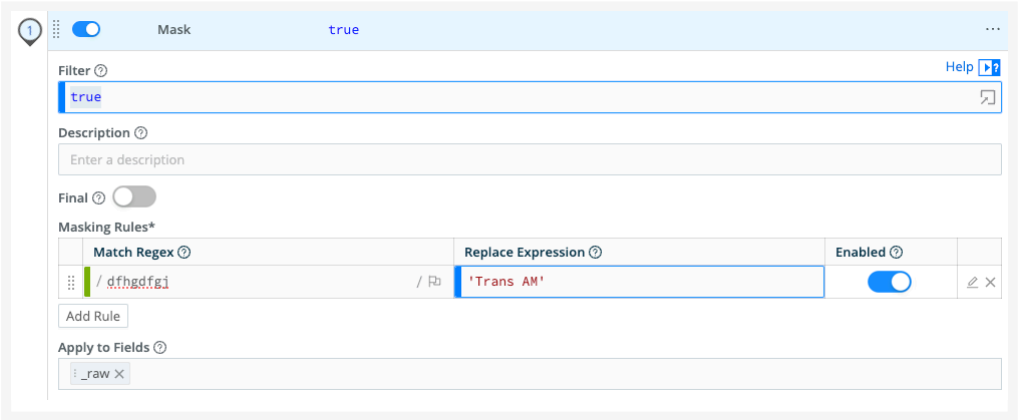
Example 1: Transform a String

Here, we'll simply search for the string `dfhgdfgj`, and replace that value (if found) with `Trans AM`. This will help close America's muscle-car gap:

Event before masking

Configure the Mask Function > Masking rules as follows:

Match Regex: `dfhgdfgj` Replace Expression: `Trans AM`



Mask Function configuration

Result: Vroom vroom!

Event after masking

Example 2: Mask Sensitive Data

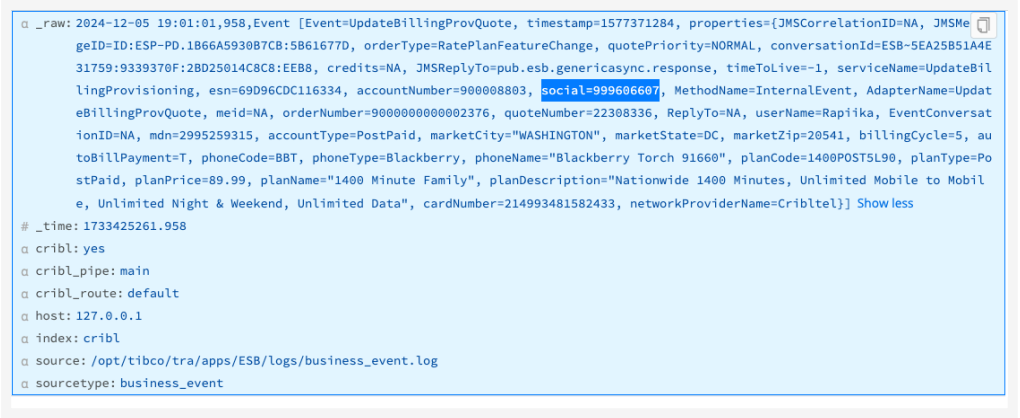
This section demonstrates how to use Mask Functions to anonymize sensitive data, with examples for Social Security numbers and credit card numbers.

In these examples, the Mask Function runs an md5 hash of the values for the specified keys and replaces the original values with the hashed values. Everything in the **Match Regex** field is replaced by the expression defined in the **Replace Expression** field. If you prefer, you can use capture groups in **Match Regex** to define individual string components for manipulation. You can also use string literals in **Replace Expression** to retain static text. Cribl does not retain any content that matches the **Match Regex** that is not inserted into the **Replace Expression**.

If you need to send the unmodified values to a Destination, such as an archival stores, you can use the **Destination** field for the associated Route to narrow the scope of the Mask Function.

Mask Social Security Numbers

This example assumes that you're ingesting data whose `_raw` fields contain unredacted Social Security numbers in the Key=Value pattern `social=#########`.



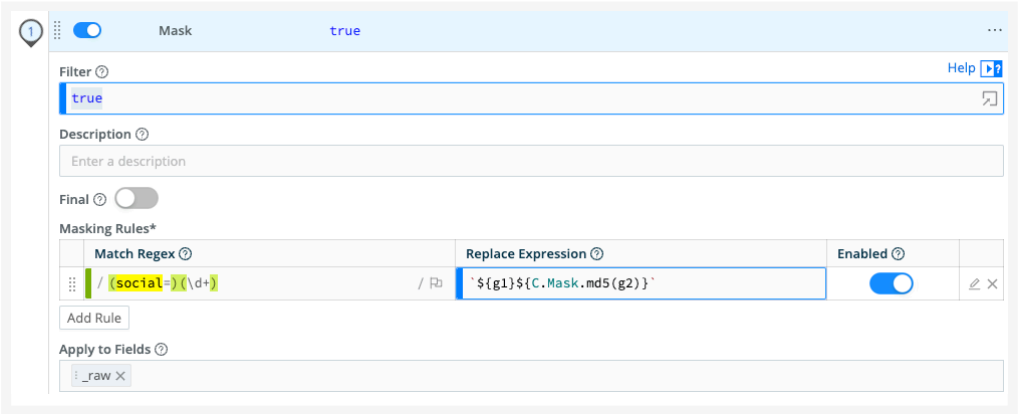
Event with unredacted SSNs

To use a Mask Function to replace the original numeric values for all `social` keys with hashed values, enter the following values into the **Masking rules** fields:

- **Match Regex**: `(social=)(\d+)`
- **Replace Expression**: `` `${g1}${C.Mask.md5(g2)}` ``

Everything in **Match Regex** is replaced by the expression defined in the **Replace Expression** field. `social=` is assigned to capture group `g1` for later reference. The value of `social=` will be hashed because it is referenced as `g2` in the md5 function.

If you don't make `social=` its own capture group (or specify `social=` as a literal in **Replace Expression**) then you cannot reference it using `g1` in **Replace Expression**. The value will be assigned to `g1` along with `social=`, and the entire `social=#########` string will be replaced with a hash of the Social Security number. Without the key name preceding the hashed value, it isn't clear what value is being hashed.



Mask Function configuration for masking Social Security numbers

Result: The sensitive values of `social=` are replaced by their md5 hashes.

Event with hashed Social Security numbers

Mask Credit Card Numbers

This example assumes that you're ingesting data whose `_raw` fields contain unredacted credit card numbers in the values of the `cardNumber` key. The `cardNumber` values may be digits alone or may include hyphens or spaces as separators.

Event with unredacted credit card numbers

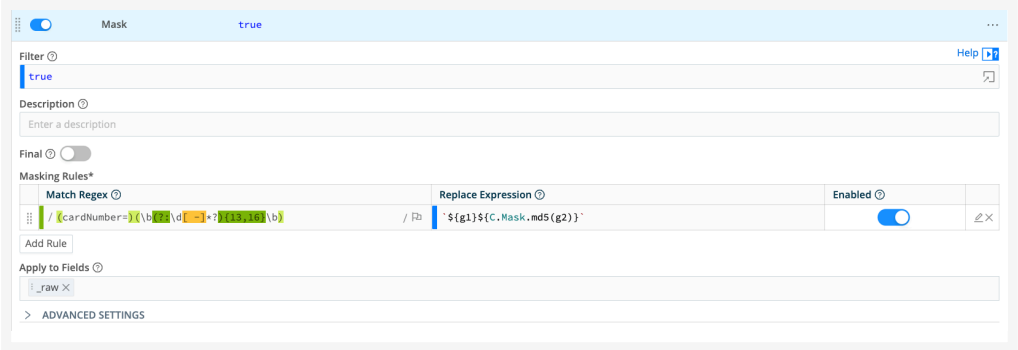
To use a Mask Function to replace the original numeric values for all `cardNumber` keys with hashed values, configure the **Masking rules** fields:

- **Match Regex**: `(cardNumber=)(\b(?:\d[ -]*?){13,16}\b)`
- **Replace Expression**: `` `${g1}${C.Mask.md5(g2)}` ``

Everything in **Match Regex** is replaced by the expression defined in the **Replace Expression** field. `cardNumber=` is assigned to capture group `g1` for later reference. The value of `cardNumber=` will be hashed because it is referenced as `g2` in the md5 function.

If you don't make `cardNumber=` its own capture group (or specify `cardNumber=` as a literal in **Replace Expression**) then you cannot reference it using `g1` in **Replace Expression**. The value will be assigned to `g1`

along with `cardNumber=`, and the entire string, including `cardNumber=`, will be replaced with a hash of the credit card number. Without the key name preceding the hashed value, it isn't clear what value is being hashed.



Mask Function configuration for masking credit card numbers

Result: The sensitive values of `cardNumber=` are replaced by their md5 hashes.

Event with hashed credit card numbers

Example 3: Replace with an Event Field

In this example, we'll replace the IP address `127.0.0.1` in the `_raw` field with the IP address `192.168.123.25` from an existing field named `source_ip`. The following is a snippet of `_raw`:

```
ProcessId=0x0 IpAddress=127.0.0.1 IpPort=0
```

Copy

To match the IP address, we'll define three capture groups. Set **Match Regex** to:

```
/(IpAddress=)((?:\d{1,3}\.){3}\d{1,3})(\s)/
```

Copy

In the **Replace Expression**, we'll reference the `source_ip` field by prepending `event` to it, like this:

```
${g1}${event.source_ip}${g3}
```

7.23. Numerify

The Numerify Function converts event fields that are numbers to type `number`.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Depth: Controls how deep the function searches nested events. Higher values may slow performance. Defaults to `5`. Maximum is `10`.

Ignore fields: Specify fields to **not** numerify. Type in field names, separated by hard returns. Supports wildcards (`*`) and nested addressing. When empty (the default), Numerify applies to **all** fields. When populated, takes precedence over the **Include expression**. Also supports negated terms. For example, `!foo*` before `*` means "Ignore all fields, except for those that start with `foo`." See Build Custom Logic to Route and Process Your Data: Wildcard Lists for more information.

Include expression: Optional JavaScript expression to specify fields to numerify. If empty (the default), the Function will attempt to numerify all fields - except those listed in **Ignore fields**, which takes precedence. Use the `name` and `value` global variables to access fields' names/values. (Example: `value != null`.) You can access other fields' values via `__e.<fieldName>`.

Format: Optionally, reformat or truncate the extracted numeric value. Select one of:

- **None**: Applies no reformatting (the default).
- **Floor**: Rounds the number down to the lower adjacent integer (truncates it).
- **Ceil**: Rounds the number up to the higher adjacent integer, removing decimal digits.
- **Round**: Rounds (truncates) the number to a specified number of digits. This option exposes an extra field:
    - **Digits**: Number of digits after the decimal point. Enter a value between `0-20`; defaults to `2`.

Examples

Scenario A:

Assume an event whose text contains a numeric value that must be extracted to perform some numeric analysis. The text looks like this:

```
version=11.5.0.0.1.1588476445
```

We can extract the numeric value by chaining together two Functions:

1. A Regex Extract Function. Set its **Regex** field to `/version=(?<ver>\d+)/`, to capture the first set of digits found in the event string.
2. Then use Numerify.

This captures the substring `11` and converts it to a numeric `11` value.

Scenario B:

Assume email transaction log events like the sample below. The final field is the message's size, in bytes. We want to extract this as a numeric value, for analysis in Cribl Stream or downstream services:

```
03:19 03:22 SMTPD (00180250) [209.221.59.70] C:\IMail\spool\D28de0018025017cd.SMD 3827
```

Again, we can accomplish this with two Functions:

1. A Regex Extract Function. To capture a substring of digits that follows six other substrings (all separated by white space), we set the **Regex** field to: `\S+\s+\S+\s+\S+\s+\S+\s+\S+\s+\S+\s+(?<bytes>\d+)`
2. Then use Numerify.

7.24. OTLP Logs

The OTLP Logs Function ingests logs sent by the Cribl Stream OTel Source, normalizes (that is, formats) and optionally batches them, and sends them on to the Cribl Stream OTel Destination.

How the OTel Source Transforms Log Events

The events that this Function processes can only come from the Cribl Stream OTel Source which itself always ingests events in the form of OTLP logs protocol buffers (protobufs). These protobufs are structures that nest objects within other objects. When you get to the bottom of the nested structure, what you've got is log entries!

Each OTLP logs protobuf contains top-level `ResourceLogs` objects that associate an OTLP Resource with one or more log entries. To get familiar with the internal nomenclature of the OTLP logs protobuf, let's look at its nested objects from the top down.

Object	Description
Log protobuf message	Contains one or more `ResourceLogs`
`ResourceLogs`	Contains one `Resource` and one or more `ScopeLogs`
`ScopeLogs`	Contains one `Scope` and one or more `LogRecords`
`LogRecord`	The log entry

The OTel Source takes these OTLP log protobufs and transforms them into Cribl events that the OTLP Logs Function can ingest.

The OTel Source can send Cribl events in two forms: with or without logs extracted. Which form it sends depends on whether **Advanced Settings** > **Extract logs** is turned on or off in the Source.

When **Extract logs** is off:

- The OTLP Logs Function cannot batch the incoming events, even when **Batch OTLP logs** is turned on. The Function just cleans up the events and sends them on.
- The Cribl events sent to the OTLP Logs Function can include log events, non-log events, or both. The OTLP Function will always send log events through to the OTel Destination. If non-log events arrive, the Function will send them through or drop them, depending on whether its **Drop non-log events** setting is turned on or off.

When **Extract logs** is on:

- The OTLP Logs Function will batch the incoming events if **Batch OTLP logs** is turned on. Otherwise the Function just cleans up the events and sends them on.
- Compared to the original OTLP log protobufs, the Cribl events sent by the OTel Source have a flatter structure: the `resource` and `scope` objects and the extracted records are all peers. The OTLP Logs Function detects this, and if batching is enabled, batches the events such that for each `resource` object there is one `scope_logs` object containing multiple pairs of `scope` and `log_records` objects.

How This Function Batches Logs

For OTLP logs, batching means grouping log events that arrive within a specified time window and whose OTLP Resource Attributes are identical. For example, if the Function receives a thousand events in a period of 30 seconds and those events originate from a set of 100 servers, each event will have a set of Resource Attributes, one or more of which will describe the server where it came from. For simplicity, let's assume that the Resource Attributes of all these events are identical except for the ones that describe the server. The Function will create separate batches for each of those 100 servers because the Resource Attributes differentiate events based on which server it comes from.

Each incoming log event contains both its resource attributes and the log entry itself. In a batch, the resource attributes need only be written once, greatly reducing the amount of data sent over the wire.

For detailed insights into events that passed through the Function but were dropped, set the logging level for channel `func:otlp_logs` to `debug`, and there will be a stats log message output once per minute with the relevant information.

Usage

Ideally, this Function should process events and/or logs **after** other Functions have formatted them. Therefore Cribl recommends that you place this Function last in the Pipeline.

If no metric events appear in Simple or Full Preview, try enabling **Show Dropped Events**: If Cribl Stream processed and dropped events, there's an issue with your data. Either the incoming events are malformed, or, the resource attributes in those events do not match any of the resource attribute prefixes you configured. See **Resource attribute prefixes** below.

Filter: Filter expression (JavaScript) that selects data to feed through the Function. The default `true` setting passes all events through the Function.

Description: Simple description of this Function's purpose in this Pipeline. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Drop non-log events: Whether or not to drop any non-log events. Default is toggled off.

Batch OTLP logs: Whether to batch OTLP logs by shared top-level `resource` attributes. The Function only tries to batch events where the logs have been extracted - that is, where the sending OTel Source had **Advanced Settings** > **Extract logs** enabled when processing the events.

When enabled, exposes these additional controls:

- **Batch size**: The **number of log records** after which a batch will be sent, regardless of whether the timeout duration has been reached.
- **Batch timeout (ms)**: Time duration after which a batch will be sent regardless of size. Defaults to `200` milliseconds.
- **Batch size limit (kb)**: Maximum batch size **in kilobytes**. `0` means no limit.
- **Batch log metadata keys**: Optionally, add metadata keys that you want the Function to create batcher instances for, in addition to any batching based on resource attributes. When set, this processor will create one batcher instance per distinct combination of values in the metadata.
- **Metadata cardinality limit**: Limits the number of unique combinations of metadata key values that the batcher will process. Once the limit has been reached, the Function will drop events whose metadata key values constitute a new combination. Defaults to `1000`.

Basic Example

Filter: `__inputId=='open_telemetry:open_telemetry'`

This will process all events from an OTel Source; in this example we assume that the Source has **Advanced Settings** > **Extract logs** turned off. Metrics and traces will either be dropped, or emitted without modification, depending on configuration. Unextracted logs will be formatted and emitted; extracted logs will be formatted and either batched or emitted (again depending on configuration).

The Function converts the Cribl event such that the Cribl Stream OTel Destination can ingest it and convert it to an OTLP log event.

See the sample incoming and outgoing events.

Advanced Example

Here, we have two incoming events sent by a OTel Source with **Advanced Settings** > **Extract logs** turned on.

The Function, with **Batch OTLP logs** turned on, batches the Cribl events such that the Cribl Stream OTel Destination can ingest the batch and convert to a batched OTLP log event.

See the sample incoming and outgoing events.

Sample Events

Basic Example Incoming Event

From OTel Source with extract logs turned off.


```
{
    "resource": {
        "attributes": {
            "telemetry.sdk.language": "python"
        }
    },
    "scope_logs": [{
        "scope": {
            "name": "opentelemetry.sdk._logs._internal",
            "version": "unknown"
        },
        "log_records":[{
            "time_unix_nano": 1717010319837985500,
            "observed_time_unix_nano": 0,
            "body": "Recommendation service started, listening on port 9001",
            "dropped_attributes_count": 0,
            "severity_number": 9,
            "severity_text": "INFO"
        }]
    }],
    "_time": 1717010325.183
}
```


Copy

Basic Example Outgoing Event

When the OTLP Function processes the incoming event above, the event below is the result. It does not matter whether **Batch OTLP logs** is turned on or off - the Function will not try to batch because in the incoming event, logs were not extracted.

```
{
    "resource": {
        "attributes": [{
            "key": "telemetry.sdk.language",
            "value": {
                "string_value": "python"
            }
        }]
    },
    "scope_logs": [{
        "scope": {
            "name": "opentelemetry.sdk._logs._internal",
            "version": "unknown",
            "attributes": []
        },
        "log_records":[{
            "time_unix_nano": 1717010319837985500,
            "observed_time_unix_nano": 0,
            "body": { "string_value": "Recommendation service started, listening on p
            "attributes": [],
            "dropped_attributes_count": 0,
            "severity_number": 9,
            "severity_text": "INFO"
        }]
    }],
    "_time": 1717010325.183
}
```


Copy

Advanced Example Incoming Events

In order to show batching in the outgoing events, let's use two incoming events from an OTel Source where **Advanced Settings** > **Extract logs** is turned on.

Incoming Sample Event 1

```
{
    "schema_url": "https://opentelemetry.io/schemas/1.23.1",
    "resource": {
        "attributes": {
            "telemetry.sdk.language": "python"
        }
    },
    "scope": {
        "name": "kafka.server.KafkaRaftServer",
        "version": "unknown"
    },
    "time_unix_nano": 1717016223073301200,
    "observed_time_unix_nano": 1717016223073302800,
    "body": "[KafkaRaftServer nodeId=1] Kafka Server started",
    "dropped_attributes_count": 0,
    "severity_number": 9,
    "severity_text": "INFO",
    "_time": 1717016223.0733013
}
```


Copy

Incoming Sample Event 2

```
{
    "schema_url": "https://opentelemetry.io/schemas/1.23.1",
    "resource": {
        "attributes": {
            "telemetry.sdk.language": "python"
        }
    },
    "scope": {
        "name": "kafka.server.KafkaBoatServer",
        "version": "unknown"
    },
    "time_unix_nano": 1717016223073401200,
    "observed_time_unix_nano": 1717016223073402800,
    "body": "[KafkaBoatServer nodeId=1] Boat Server started",
    "dropped_attributes_count": 0,
    "severity_number": 9,
    "severity_text": "INFO",
    "_time": 1717016224.0733013
}
```


Copy

Advanced Example Outgoing Batched Events

When the OTLP Function processes the incoming events above, the event below is the result. Here, **Batch OTLP logs** is turned on - the Function is able to batch because in the incoming events, logs were extracted.

```
{
    "resource": {
        "attributes": [{
            "key": "telemetry.sdk.language",
            "value": {
                "string_value": "python"
            }
```

7.25. OTLP Metrics

The OTLP Metrics Function transforms dimensional metrics events into the OTLP (OpenTelemetry Protocol) format. This Function can be used to send metric events (events that contain the internal field `__criblMetrics`) to any OpenTelemetry Metrics-capable destinations.

For detailed insights into events that passed through the Function but were dropped, set the logging level for channel `func:otlp_metrics` to `debug`, and there will be a stats log message output once per minute with the relevant information.

This Function will delete the internal attribute `__criblMetrics`, making events only suitable for destinations that use the OTLP format.

Usage

Ideally, this Function should process events and/or metrics **after** other Functions have formatted them. Therefore Cribl recommends that you place this Function last in the Pipeline.

If no metric events appear in Simple or Full Preview, try enabling **Show Dropped Events**: If Cribl Stream processed and dropped events, there's an issue with your data. Either the incoming events are malformed, or, the resource attributes in those events do not match any of the resource attribute prefixes you configured. See **Resource attribute prefixes** below.

Filter: Filter expression (JavaScript) that selects data to feed through the Function. The default `true` setting passes all events through the Function.

Description: Simple description of this Function's purpose in this Pipeline. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Resource attribute prefixes: The prefixes of top-level attributes to add as resource attributes. See the OpenTelemetry documentation for Resource Semantic Conventions.

- Use periods and underscores for precise matching. For example, `process.path`.
- Remove all the prefixes if you want to bring none of the resources over.
- Use the Eval Function to copy nested attributes to the top level for matching. For example, configure **Evaluate Fields** as follows to create a new top-level field named `process_path` with the value copied

from the nested attribute `process.path`. This new field can then be matched using the OTLP Metrics Function's **Resource attribute prefixes**:

- **Name**: `process_path`
- **Value Expression**: `process.path`

Drop non-metric events: Whether or not to drop any non-metric events.

OTLP version: The drop-down offers `0.10.0` (default) and `1.3.1`.

Batch OTLP metrics: Whether to batch OTLP metrics by shared top-level `resource` attributes. When enabled, exposes these additional controls:

- **Batch size**: The number of metric data points after which a batch will be sent, regardless of whether the timeout duration has been reached.
- **Batch timeout (ms)**: Time duration after which a batch will be sent regardless of size.
- **Batch size limit (kb)**: Maximum batch size. `0` means no limit.
- **Batch metrics metadata keys**: Optionally, add metadata keys that you want the Function to create batcher instances for, in addition to any batching based on resource attributes. When set, this processor will create one batcher instance per distinct combination of values in the metadata.
- **Metadata cardinality limit**: Limits the number of unique combinations of metadata key values that the batcher will process. Once the limit has been reached, the Function will drop events whose metadata key values constitute a new combination.

Choosing or Combining OTLP Versions

Because this Function resides in between Cribl Stream OTel Sources and OTel Destinations, each of which can be independently configured to use the same or different OTLP versions, it's important to know the effects of different choices and combinations.

- Cribl Stream supports starting with OTLP 0.10.0 events and converting them to OTLP 1.3.1 events, but does not support the reverse (1.3.1 to 0.10.0).
- When **OTLP version** is set to `1.3.1`, Cribl Stream will attempt to convert metrics or logs formatted for any previous version to 1.3.1. Since beginning with version 1.0.0, OTLP has introduced no new breaking changes, you can use the Function configured for OTLP 1.3.1 as an "OTLP version 0.x to version 1" converter. Most downstream OTLP receivers comply with OTLP 1.0.0 at minimum; but you should verify this for whatever downstream receiver you use.

Basic Example

Filter: `__inputId=='prometheus_rw:prom_rw_in'`

This will grab all events from a Prometheus Remote Write Source and convert the metrics to OTLP format using only default settings in the Function.

Ensure that the internal attribute `__criblMetrics` does not contain `null` or blank values in any of the child object arrays.

See the sample incoming and outgoing events.

Advanced Example

Here, we have two incoming events sent by a OTel Source with **Advanced Settings** > **Extract Metrics** turned on.

The Function, with **Batch OTLP Metrics** turned on, batches the Cribl events such that the Cribl Stream OTel Destination can ingest the batch and convert to a batched OTLP metric event.

See the sample incoming and outgoing events.

Sample Events

Basic Example Incoming Event

Here is an example incoming event.

```
{
  "__criblEventType": "event",
  "__ctrlFields": [],
  "__final": false,
  "__cloneCount": 0,
  "_time": 1700236380.37,
  "_value": 123.456,
  "_metric": "cribl_logstream_health_outputs",
  "_metric_type": "gauge",
  "cribl_host": "P4MD6R-Pv8x",
  "cribl_wp": "w0",
  "event_host": "P4MD6R-Pv8x",
  "event_source": "cribl",
  "host": "P4MD6R-Pv8x",
  "output": "open_telemetry:otel_webhook_dest",
  "source": "cribl",
  "__criblMetrics": [
    {
      "nameExpr": ["_metric"],
      "values": ["_value"],
      "types": ["gauge"],
      "dims": [
        "cribl_host",
        "cribl_wp",
        "event_host",
        "event_source",
        "host",
        "output",
        "source"
      ]
    }
  ],
  "__offset": 8099,
  "__bytes": 228,
  "__srcIpPort": "127.0.0.1:50868",
  "__inputId": "prometheus_rw:prom_rw_in",
  "service.name": "serviceName",
  "service.instanceId": "serviceInstanceId",
  "system": "system",
  "system.name": "systemName",
  "telemetry.sdk.name": "opentelemetry",
  "telemetry.sdk.language": "java",
```

```
  "k8s.cluster.name": "opentelemetry-cluster",
  "cloud.account.id": 1111111,
  "cloud.platform": "aws"
}
```

Copy

Basic Example Outgoing Event

Here is the above sample event after conversion by the OTLP Metrics Function.

```
{
  "_time": 1700236380.37,
  "resource": {
    "attributes": [
      {
        "key": "host",
        "value": {
          "string_value": "P4MD6R-Pv8x"
        }
      },
      {
        "key": "service.name",
        "value": {
          "string_value": "serviceName"
        }
      },
      {
        "key": "service.instanceId",
        "value": {
          "string_value": "serviceInstanceId"
        }
      },
      {
        "key": "system",
        "value": {
          "string_value": "system"
        }
      },
      {
        "key": "system.name",
        "value": {
          "string_value": "systemName"
        }
      },
      {
        "key": "telemetry.sdk.name",
        "value": {
          "string_value": "opentelemetry"
        }
      },
      {
        "key": "telemetry.sdk.language",
```

```
      "value": {
        "string_value": "java"
      }
    },
    {
      "key": "k8s.cluster.name",
      "value": {
        "string_value": "opentelemetry-cluster"
      }
    },
    {
      "key": "cloud.account.id",
      "value": {
        "int_value": 1111111
      }
    },
    {
      "key": "cloud.platform",
      "value": {
        "string_value": "aws"
      }
    }
  ]
},
"instrumentation_library_metrics": [
  {
    "schema_url": "",
    "metrics": [
      {
        "name": "cribl_logstream_health_outputs",
        "gauge": {
          "data_points": [
            {
              "as_double": 123.456,
              "time_unix_nano": 1700236380370000000,
              "start_time_unix_nano": 1700236380370000000,
              "attributes": [
                {
                  "key": "cribl_host",
                  "value": {
                    "string_value": "P4MD6R-Pv8x"
                  }
                },
```

```
            {
              "key": "cribl_wp",
              "value": {
                "string_value": "w0"
              }
            },
            {
              "key": "event_host",
              "value": {
                "string_value": "P4MD6R-Pv8x"
              }
            },
            {
              "key": "event_source",
              "value": {
                "string_value": "cribl"
              }
            },
            {
              "key": "host",
              "value": {
                "string_value": "P4MD6R-Pv8x"
              }
            },
            {
              "key": "output",
              "value": {
                "string_value": "open_telemetry:otel_webhook_dest"
              }
            },
            {
              "key": "source",
              "value": {
                "string_value": "cribl"
              }
            }
          ]
        }
      ]
    },
    "__type": "gauge"
  }
]
```

```
      }
    ]
}
```

Copy

Advanced Example Incoming Events

In order to show batching in the outgoing events, let's use two incoming events from an OTel Source where **Advanced Settings** > **Extract metrics** is turned on.

Incoming Sample Event 1 - Gauge Metric

```
{
    "_time": 1700236380.37,
    "resource":
    {
        "attributes":
        [
            {
                "key": "service.name",
                "value":
                {
                    "string_value": "serviceName"
                }
            },
            {
                "key": "service.instanceId",
                "value":
                {
                    "string_value": "serviceInstanceId"
                }
            },
            {
                "key": "system.name",
                "value":
                {
                    "string_value": "systemName"
                }
            },
            {
                "key": "telemetry.sdk.name",
                "value":
                {
                    "string_value": "opentelemetry"
                }
            },
            {
                "key": "telemetry.sdk.language",
                "value":
                {
                    "string_value": "java"
                }
            },
            {
```

```
            "key": "k8s.cluster.name",
            "value":
            {
                "string_value": "opentelemetry-cluster"
            }
        }
    ]
},
"instrumentation_library_metrics":
[
    {
        "schema_url": "",
        "metrics":
        [
            {
                "name": "cribl_logstream_health_outputs",
                "gauge":
                {
                    "data_points":
                    [
                        {
                            "as_double": 123.456,
                            "time_unix_nano": 1700236380370000000,
                            "start_time_unix_nano": 1700236380370000000,
                            "attributes":
                            [
                                {
                                    "key": "cribl_host",
                                    "value":
                                    {
                                        "string_value": "P4MD6R-Pv8x"
                                    }
                                },
                                {
                                    "key": "cribl_wp",
                                    "value":
                                    {
                                        "string_value": "w0"
                                    }
                                },
                                {
                                    "key": "event_host",
                                    "value":
```

```
                                {
                                    "string_value": "P4MD6R-Pv8x"
                                }
                            },
                            {
                                "key": "event_source",
                                "value":
                                {
                                    "string_value": "cribl"
                                }
                            },
                            {
                                "key": "host",
                                "value":
                                {
                                    "string_value": "P4MD6R-Pv8x"
                                }
                            },
                            {
                                "key": "output",
                                "value":
                                {
                                    "string_value": "open_telemetry:otel_webh
                                }
                            },
                            {
                                "key": "source",
                                "value":
                                {
                                    "string_value": "cribl"
                                }
                            }
                        ]
                    }
                ]
            },
            "__type": "gauge",
            "description": "This is a metric description"
        }
    ],
    "instrumentation_library":
    {
        "name": "io.opentelemetry.sdk.trace",
```

```
                "version": "1.33.0-alpha"
            }
        }
    ],
    "__outLen": 1266,
    "__bufToken": "otlp-metrics-batcher1"
}
```


Copy

Incoming Sample Event 2 - Summary Metric

```
{
    "__criblEventType": "event",
    "__ctrlFields":
    [],
    "__final": false,
    "__cloneCount": 0,
    "__bytes": 404,
    "__criblMetrics":
    [
        {
            "dims":
            [
                "cribl_host",
                "cribl_wp"
            ],
            "nameExpr":
            [
                "_metric"
            ],
            "types":
            [
                "summary"
            ],
            "values":
            [
                "_value"
            ]
        }
    ],
    "__inputId": "prometheus_rw:prom_rw_in",
    "__offset": 5083,
    "__srcIpPort": "127.0.0.1:59491",
    "_metric": "request_duration_summary",
    "_metric_type": "summary",
    "_time": 1701793930,
    "description": "This is an event description",
    "_value": 0,
    "cribl_host": "P4MD6R-Pv8x",
    "cribl_wp": "w0",
    "request_duration_summary_data":
    {
        "_count": 0,
```

```
        "_quantiles":
        {
            "0.5": 0,
            "0.9": 0
        },
        "_sum": 0
    },
    "instrumentation_library":
    {
        "name": "io.opentelemetry.sdk.trace",
        "version": "1.33.0-alpha"
    },
    "service.name": "serviceName",
    "service.instanceId": "serviceInstanceId",
    "system": "system",
    "system.name": "systemName",
    "telemetry.sdk.name": "opentelemetry",
    "telemetry.sdk.language": "java",
    "k8s.cluster.name": "opentelemetry-cluster",
    "cloud.account.id": 1111111,
    "cloud.platform": "aws"
}
```

Copy

Advanced Example Outgoing Batched Events

Here is the resulting converted OTLP metric events in a single batch.

```
{
    "_time": 1700236380.37,
    "resource":
    {
        "attributes":
        [
            {
                "key": "service.name",
                "value":
                {
                    "string_value": "serviceName"
                }
            },
            {
                "key": "service.instanceId",
                "value":
                {
                    "string_value": "serviceInstanceId"
                }
            },
            {
                "key": "system.name",
                "value":
                {
                    "string_value": "systemName"
                }
            },
            {
                "key": "telemetry.sdk.name",
                "value":
                {
                    "string_value": "opentelemetry"
                }
            },
            {
                "key": "telemetry.sdk.language",
                "value":
                {
                    "string_value": "java"
                }
            },
            {
```

```
            "key": "k8s.cluster.name",
            "value":
            {
                "string_value": "opentelemetry-cluster"
            }
        }
    ]
},
"instrumentation_library_metrics":
[
```

7.26. OTLP Traces

The OTLP Traces Function ingests traces sent by the Cribl Stream OTel Source, normalizes (that is, formats) and optionally batches them, and sends them on to Cribl Stream Destinations.

How the OTel Source Transforms Trace Events

This Function can process only events coming from the Cribl Stream OTel Source, which always ingests events in the form of OTLP trace protocol buffers (protobufs). These protobufs are structures that nest objects within other objects. Each OTLP trace protobuf contains top-level `ResourceSpans` objects that associate an OTLP Resource with one or more spans.

Object	Description
Trace protobuf message	Contains one or more `ResourceSpans`
`ResourceSpans`	Contains one Resource and one or more `ScopeSpans`
`ScopeSpans`	Contains one Scope and one or more `Span`
`Span`	The trace entry

How This Function Batches Traces

For OTLP traces, batching means grouping extracted spans that arrive within a specified time window and whose OTLP Resource Attributes are identical. The Function will create separate batches for each unique combination of resource attributes and metadata keys.

Usage

Ideally, this Function should process events and/or traces **after** other Functions have formatted them. Therefore, Cribl recommends that you place this Function last in the Pipeline.

Filter: Filter expression (JavaScript) that selects data to feed through the Function. The default `true` setting passes all events through the Function.

Description: Simple description of this Function's purpose in this Pipeline. Defaults to empty.

Final: If toggled to **Yes**, stops feeding data to the downstream Functions. Defaults to **No**.

Drop non-trace events: Whether or not to drop any non-trace events. Defaults to **No**.

OTLP version: Specifies the version of the OTLP protocol used for formatting and sending output events downstream.

- Version `0.10.0` (Default): This version uses the older OTLP format. It is recommended for users who need compatibility with legacy systems.
- Version `1.3.1`: This version supports newer OTLP features, including `scope_spans` instead of `instrumentation_library_spans`, and additional fields like `flags` in `Link` and `Span` objects. It is recommended for users who want to leverage the latest OTLP features and improvements.

Batch OTLP traces: When set to **Yes**, the Function batches together extracted spans that have the same resource and (optionally) metadata keys until they reach a flush criteria. When set to **No**, the Function will emit extracted/unextracted span events that have been normalized, but they will not be combined with other spans - each event passed to the Function will result in one event being returned.

When set to **Yes**, exposes these additional controls:

- **Batch size**: The number of spans after which a batch will be sent, regardless of whether the timeout duration has been reached.
- **Batch timeout (ms)**: Time duration after which a batch will be sent regardless of size. Defaults to `200` milliseconds.
- **Batch size limit (kb)**: Maximum batch size in kilobytes. `0` means no limit.
- **Batch trace metadata keys**: Optionally, add metadata keys that you want the Function to create batcher instances for, in addition to any batching based on resource attributes. When set, this processor will create one batcher instance per distinct combination of values in the metadata.
- **Metadata cardinality limit**: Limits the number of unique combinations of metadata key values that the batcher will process. Once this limit is reached, events with new combinations of metadata key values will be dropped. Defaults to `1000`.

Example

This example demonstrates how the OTLP Traces Function processes and batches trace events when the **Batch OTLP traces** setting is enabled. The settings used in this example are:

- **Batch OTLP traces**: **Yes**
- **Batch size**: 2
- **Batch timeout (ms)**: `200`
- **Batch size limit (kb)**: `0` (no limit)
- **Batch trace metadata keys**: **None**
- **Metadata cardinality limit**: `1000`
- **Drop non-trace events**: **Yes**

Incoming Event 1


```
{
  "resource": {
    "attributes": {
      "service.name": "currencyservice",
      "service.instance.id": "instance-1"
    }
  },
  "scope": {
    "name": "currencyservice",
    "version": "1.0.0"
  },
  "spans": [{
    "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
    "span_id": "00f067aa0ba902b7",
    "name": "GetSupportedCurrencies",
    "start_time_unix_nano": 1616587310000000000,
    "end_time_unix_nano": 1616587311000000000,
    "attributes": {
      "http.method": "GET",
      "http.url": "/currencies"
    }
  }]
}
```


Copy

Incoming Event 2

```
{
  "resource": {
    "attributes": {
      "service.name": "currencyservice",
      "service.instance.id": "instance-1"
    }
  },
  "scope": {
    "name": "currencyservice",
    "version": "1.0.0"
  },
  "spans": [{
    "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
    "span_id": "00f067aa0ba902b8",
    "name": "ConvertCurrency",
    "start_time_unix_nano": 1616587312000000000,
    "end_time_unix_nano": 1616587313000000000,
    "attributes": {
      "http.method": "POST",
      "http.url": "/convert"
    }
  }]
}
```


Copy

Outgoing Batched Event

```
{
  "resource": {
    "attributes": [
      {
        "key": "service.name",
        "value": {
          "string_value": "currencyservice"
        }
      },
      {
        "key": "service.instance.id",
        "value": {
          "string_value": "instance-1"
        }
      }
    ]
  },
  "scope_spans": [
    {
      "scope": {
        "name": "currencyservice",
        "version": "1.0.0"
      },
      "spans": [
        {
          "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
          "span_id": "00f067aa0ba902b7",
          "name": "GetSupportedCurrencies",
          "start_time_unix_nano": 1616587310000000000,
          "end_time_unix_nano": 1616587311000000000,
          "attributes": [
            {
              "key": "http.method",
              "value": {
                "string_value": "GET"
              }
            },
            {
              "key": "http.url",
              "value": {
                "string_value": "/currencies"
              }
            }
```

```
    }
  ]
},
{
  "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
  "span_id": "00f067aa0ba902b8",
  "name": "ConvertCurrency",
  "start_time_unix_nano": 1616587312000000000,
  "end_time_unix_nano": 1616587313000000000,
  "attributes": [
    {
```

7.27. Parser

The Parser Function can be used to extract fields out of events or reserialize (rewrite) events with a subset of fields. Reserialization will preserve the format of the events.

For example, if an event contains comma-delimited fields, and `fieldA` and `fieldB` are filtered out, those fields' positions will be set to `null` but not deleted completely.

The Parser Function can't remove fields that it didn't create. A subsequent Eval Function can do this.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning evaluate all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Operation mode: **Extract** will create new fields. **Reserialize** will extract, filter fields, and then reserialize.

Type: Parser/Formatter type to use. Options:

- **CSV**: Comma-separated values.
- **Extended Log File Format**: Extended Log File Format.
- **Common Log Format**: Common Log Format.
- **Key=Value Pairs**: A set of data that represents two associated groups through a key and a value.
- **JSON Object**: JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.
- **Delimited values**: A character identifies the beginning or the end of a character string.
- **Regular Expression**: A sequence of characters that specifies a match pattern in text.
- **Grok**: A string of special characters and regular expressions (pattern) that match data.

The following two additional options appear in the drop-down, but can only be used when **Operation mode** is set to **Extract**; both expand the UI:

- **Grok**
- **Regular Expression**

Library: Select an option from the collection of available Parsers. Although specifying a Library will auto-generate an example list of fields, the list may still need to be modified to accommodate the desired fields from

the events as well as the actual field order.

Source field: Field that contains text to be parsed. Not usually needed in Reserialize mode.

Destination field: Name of field in which to place the extracted and serialized fields. If the field already exists, it will be overwritten. If multiple new fields are created and this field is populated, then all new fields are created as elements of an array. The array's name will be set to this field's value. If you want all new fields to be independent, rather than in an array, then specify them using **List of fields**, below. (Extract and Reserialize modes only.)

Clean fields: This option appears for **Type: K=V Pairs**. Toggle on to clean field names by replacing non-alphanumeric characters with `_`. This will also strip leading and trailing `"` symbols.

List of fields: Fields expected to be extracted, in order. If not specified, Parser will auto-generate fields.

List of Fields will not display, when the Parser/Formatter **Type** is set to `JSON Object` or `Key=Value Pairs`. This option is only relevant for structured formats like `CSV`, `CLF`, and `ELFF`. It helps name extracted fields when they aren't explicitly defined in the raw data. For `JSON Object` and `Key-Value Pair` formats, field names are inherent and this option is ignored.

Fields to keep: List of fields to keep. Takes precedence over the **Fields to remove** setting. The field accepts wildcards (`*`), which supports both:

- Wildcards (`*`) to include multiple fields.
- Nested addressing, where you can reference a parent field and all its children using a wildcard. For example, if you convert `_raw` into an object, use `_raw*` to refer to `_raw` and all nested fields inside it. See Wildcard Lists for more information.

Fields to remove: List of fields to remove. Supports wildcards (`*`). Cannot remove fields matching **Fields to keep**. Supports wildcards (`*`) to include multiple fields and nested addressing. See Wildcard Lists for more information.

Fields filter expression: Expression to evaluate against each field's `{index, name, value}` context. Return truthy to keep fields, or falsy to remove fields. Index is zero-based. See details below.

Types That Expand the UI

Setting **Type** to **Delimited Values** displays the following extra options:

- **Delimiter**: Delimiter character to split value. Defaults to comma (`,`). You can also specify pipe (`|`) or tab characters.
- **Quote char**: Character used to quote literal values. Defaults to `"`.
- **Escape char**: Character used to escape the delimiter character. Defaults to: `\`

- **Null value**: Field value representing the null value. These fields will be omitted. Defaults to: –

Setting **Type** to **Grok** displays the Grok Function UI; setting it to **Regex Extract** displays the Regex Extract UI. These Parser preserve all previewing and editing. Also, **Grok** enables you to bring Grok Patterns into Cribl Search.

Advanced Settings

Allowed key characters: Enter characters permitted in a key name, even though they are normally separator or control characters. Separate entries with a tab or hard return. This setting does not affect how the value is parsed.

Allowed value characters: Enter characters permitted in a value name, even though they are normally separator or control characters. Separate entries with a tab or hard return. This setting does not affect how the key is parsed.

Fields/Values Interaction

This Function's **Filter**, **Type**, **Fields to keep**, **Fields to remove**, and **Fields filter expression** settings interact as follows.

Order of Evaluation

The order is: **Fields to keep** > **Fields to remove** > **Fields filter expression**.

Precedence

If a field is in both **Fields to keep** and **Fields to remove**, **Fields to keep** takes precedence.

If a field is in both **Fields to remove** and **Fields filter expression**, **Fields to remove** takes precedence.

Negation

Both **Fields to remove** and **Fields to keep** support negated terms. When you use negated terms, your fields list is order-sensitive. For example, `!foobar, foo*` means "All fields that start with `foo`, except `foobar`." However, `!foo*, *` means "All fields, except for those that start with `foo`."

Characters to Escape

Particular characters need special treatment when parsed when you select the Key=Value Pairs or CSV Parser/Formatter type.

Key-Value Pairs

When a Parser Function's **Type** is set to `Key=Value Pairs`: Within your data, field values that contain an `=` delimiter **must** be surrounded by quotes (`"..."`), or the Function will return unexpected results. As an example, here is a set of four fields and values, in which two values are quoted to parse predictably:

```
one=1,two="value=2",three=3,four="value=4"
```

These would resolve to the following parsed values:

Field Name	Value
one	1
two	"value=2"
three	3
four	"value=4"

CSV

When a Parser Function's **Type** is set to `CSV`: Within your data, column values that contain quotes or commas **must** quote those characters, or the Function will return unexpected results. To illustrate this, here is a properly formatted example of four CSV values:

```
"value with ""escaped"" quotes is 1",2,3,"you know, this is 4"
```

The first value includes quotation marks, which are each escaped with surrounding quotes, and the whole value is delimited by outer quotes. The fourth value includes a comma, so that value is delimited by outer quotes. These four columns would cleanly evaluate to:

---

value with "escaped" quotes is 1 2 3 you know, this is 4

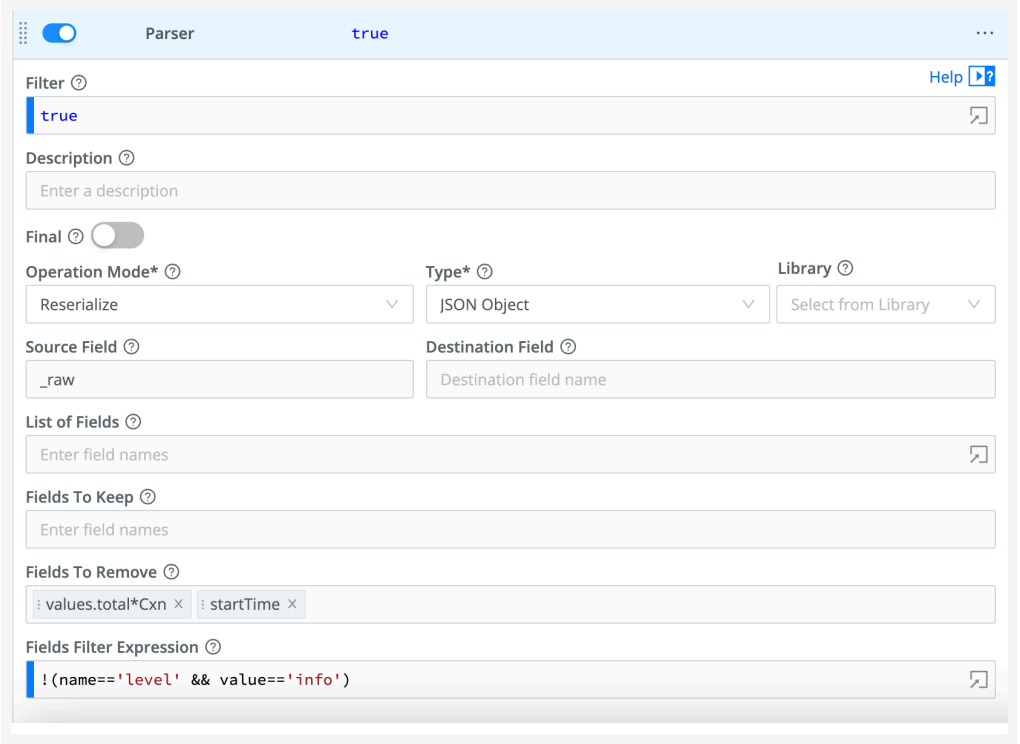
Examples

The following examples show creating and dropping fields, reserialization, and serializing to CSV versus JSON.

Example 1

Insert the following sample, using **Sample Data** > **Import Data**: `2019/06/24 05:10:55 PM Z a=000,b=001,c=002,d=003,e=004,f=005,g1=006,g2=007,g3=008`

Create the following test Parser Function.



Parser Function initial configuration

First, set the **Type** to `Key=Value Pairs`.

Scenario A

Keep fields `a`, `b`, `c`. Drop the rest.

Expected result: `a`, `b`, `c`

- Fields to Keep: `a`, `b`, `c`
- Fields to Remove: `*`
- Fields Filter Expression: <empty>

Result: The event will gain four new fields and values, as follows.

- a: `000`
- b: `001`
- c: `002`
- cribl_pipe: `parser2`

Scenario A result

You can check your stats by selecting the **Preview** pane's **Basic Statistics** (chart) button. In the resulting pop-up, the **Number of Fields** should have incremented ty four.

Now that you have the hang of it, try out the other simple scenarios below.

Scenario B

Keep fields `a`, `b`, those that start with `g`. Drop the rest.

Expected result: `a`, `b`, `g1`, `g2`, `g3`

- Fields to keep: `a`, `b`
- Fields to remove: [empty]
- Fields filter expression: `name.startsWith('g')`

Scenario C

Keep fields `a`, `b`, those that start with `g` but only if value is `007`. Drop the rest.

Expected result: `a`, `b`, `g2`

- Fields to keep: `a`, `b`
- Fields to remove: [empty]
- Fields filter expression: `name.startsWith('g') && value=='007'`

Scenario D

Keep fields `a`, `b`, `c`, those that start with `g`, unless it's `g1`. Drop the rest.

Expected result: `a`, `b`, `c`, `g2`, `g3`

- Fields to keep: `a`, `b`, `c`

- Fields to remove: `g1`
- Fields filter expression: `name.startsWith('g')`

Scenario E

Keep fields `a`, `b`, `c`, those that start with `g` but only if index is greater than `6`. Drop the rest.

Expected result: `a`, `b`, `c`, `g2`, `g3`

- Fields to keep: `a`, `b`, `c`
- Fields to remove: [empty]
- Fields filter expression: `name.startsWith('g') && index>6`

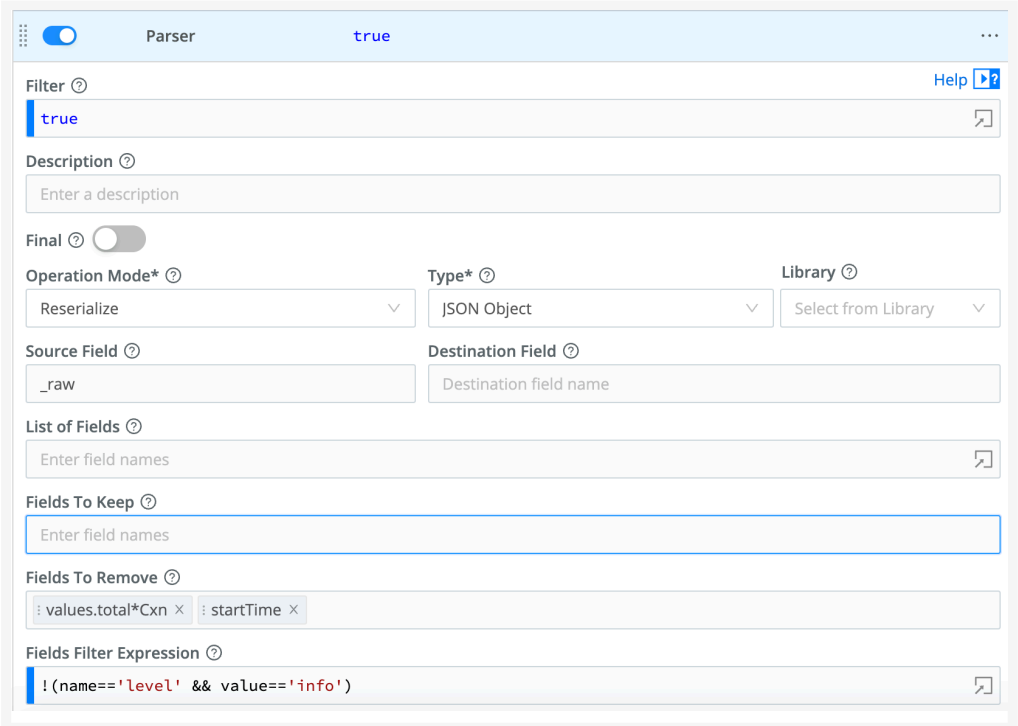
The `index` refers to the location of a field in the array of all fields extracted by **this** Parser. It is zero-based. In the case above, `g2` and `g3` have `index` values of `7` and `8`, respectively.

Example 2

Assume we have a JSON event that needs to be **reserialized**, given these requirements:

1. Remove the `level` field only if it's set to `info`.
2. Remove the `startTime` field, and all fields in the `values.total.` path that end in `Cxn`.

Parser Function configuration:



Parser Function configuration for Example 2

JSON event after being processed by the Function:

Example 2 event transformation

Example 3

Insert the following sample, using **Sample Data** > **Import Data**:

```
2019/06/24 15:25:36 PM Z a=000,b=001,c=002,d=003,e=004,f=005,g1=006,g2=007,g3=008,
```

For all scenarios below, first create a Parser Function to extract all fields, by setting the **Type** to `Key=Value Pairs`. Then add a second Parser Function with the configuration shown under **Parser 2**.

Scenario A

Serialize fields `a, b, c, d` in CSV format.

Expected result: `_raw` field will have this value `000,001,002,003`

Parser 2:

- Operation mode: `Serialize`
- Source field: [empty]

- Destination field: [empty]
- Type: `CSV`
- List of fields: `a, b, c, d` (needed for positional formats)

Scenario B

Serialize fields `a, b, c` in JSON format, under a field called `bar`.

Expected result: `bar` field will be set to: `{"a":"000","b":"001","c":"002","d":"003"}`

Parser 2:

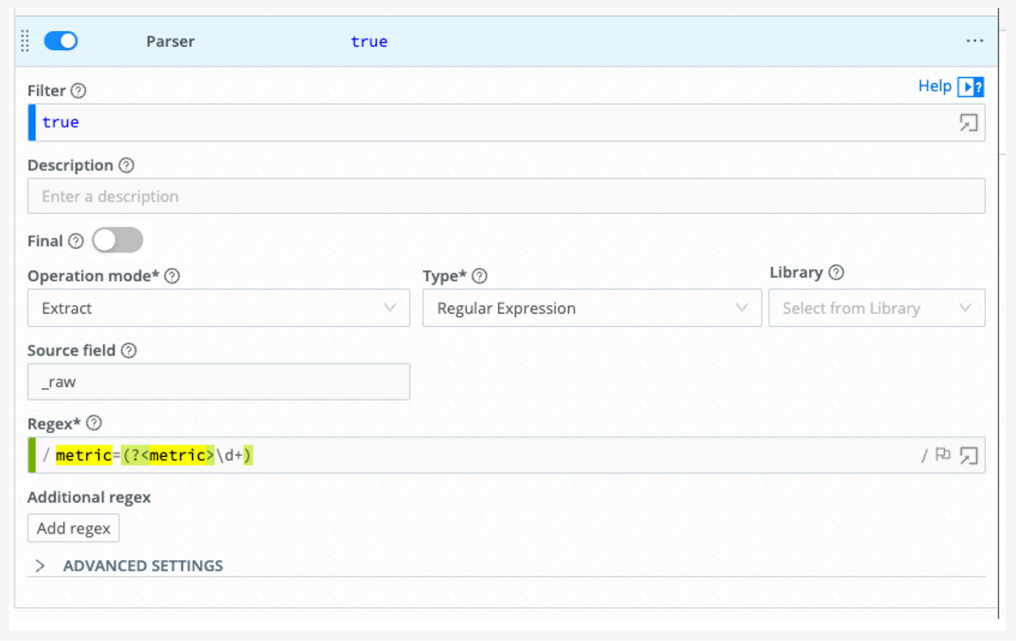
- Operation mode: `Serialize`
- Source field: [empty]
- Destination field: `bar`
- Type: `JSON`
- List of fields: [empty]
- Fields to keep: `a, b, c, d`

Example 4

Assume we have a simple event that looks like this:

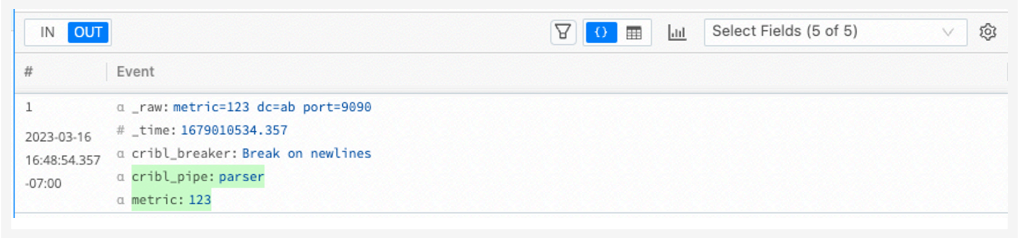
```
metric=123 dc=ab port=9090
```

If we want to extract only the `metric` field, we can configure a Parser of **Type** `Regular Expression`, where the **Regex** field is set to `metric=(?<metric>\\d+)`:



Parser Function configuration for Example 4

The output will look like this:



Parser Function regex output

Example 5

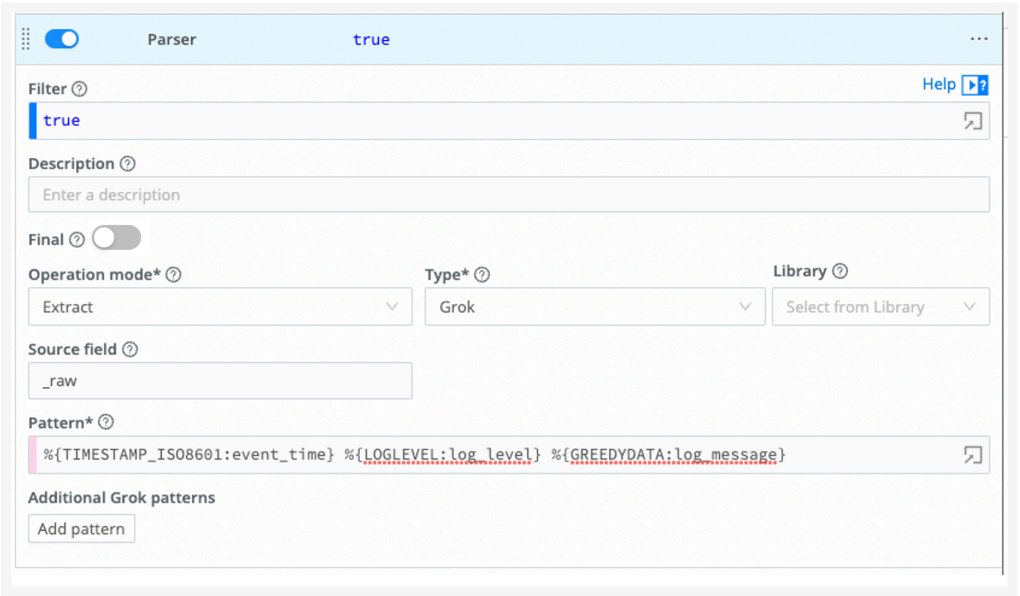
Assume we have an event that looks like this:

```
2020-09-16T04:20:42.45+01:00 DEBUG This is a sample debug log message
```

We can use a Grok pattern to parse the event so that the timestamp, log level, and the message are separated. A Parser of **Type** `Grok` with the following Pattern will work:

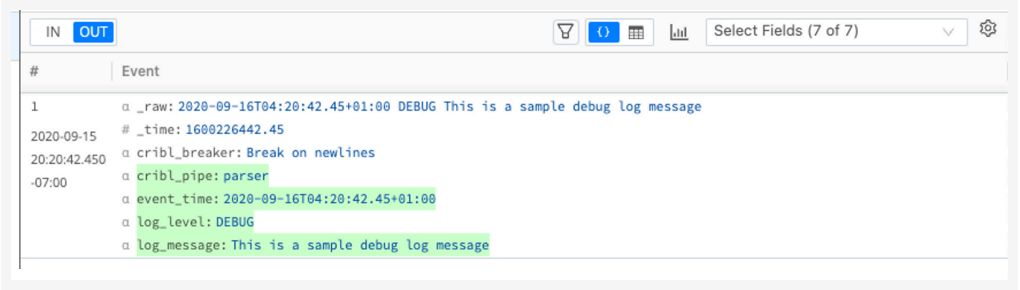
```
%{TIMESTAMP_ISO8601:event_time} %{LOGLEVEL:log_level} %{GREEDYDATA:log_message}
```

The Parser Function configuration looks like this:



Parser Function configuration for Example 5

The output will look like this:

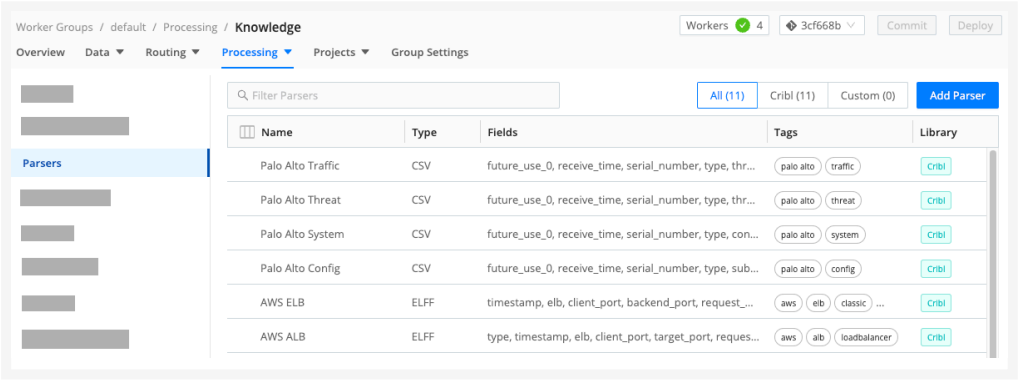


Parser Function grok output

Available Parsers

You can edit existing and create new Parsers for use in the Parser Function. The Parser collection is searchable, and each parser can be tagged as necessary.

To find the collection, select **Worker Groups** from the sidebar, and choose a Worker Group. Then, on the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Parsers**.



Available Parsers

You can refer to a single Parser in multiple Parser Functions in different Pipelines. When editing them, be aware that any change will affect all places it is used in.

Supported Parser Types

- **CSV**: Comma-separated values.
- **Extended Log File Format**: Extended Log File Format.
- **Common Log Format**: Common Log Format.
- **Key=Value Pairs**: A set of data that represents two associated groups through a key and a value.
- **JSON Object**: JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.
- **Delimited values**: A character identifies the beginning or the end of a character string.
- **Regular Expression**: A sequence of characters that specifies a match pattern in text.

- **Grok**: A string of special characters and regular expressions (pattern) that match data.

Create a Parser

To create a parser, follow these steps:

1. In the sidebar, select **Worker Groups** from the sidebar, and choose a Worker Group or Fleet.
2. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Parsers**.
3. Select **Add Parser**.

7.28. Publish Metrics

The Publish Metrics Function extracts, formats, and outputs metrics from events.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Overwrite: If toggled on, overwrite previous metric specs. Otherwise, append. Default is toggled off.

Metrics

Add metrics: List of metrics to extract from the event and format. Destinations can pass the formatted metrics to a metrics aggregation platform. Select **Add metrics** to add new rows containing the following options:

- **Event Field Name**: The name of the field (in the event) that contains the metric value. Should contain only letters, numbers, underscores (`_`), and `.` characters (to separate names in nested structures).
- **Metric Name Expression**: JavaScript expression to evaluate the metric field name. Defaults to the **Event field name** value.

The JavaScript expression will evaluate the metric field name only after the metrics are processed for transport to the Destination. While in the processing Pipeline, the metric name expression appears as a literal.

To have metric names evaluated in preview, enable the Enhance Metric Output setting.

- **Metric Type**: Select `Gauge` (the default), `Counter`, `Timer`, `Distribution`, `Summary`, or `Histogram`. General definitions that can vary across senders:
  - `Gauge`: A numeric value that can increase or decrease over time - like a temperature or pressure gauge.
  - `Counter`: A cumulative numeric value - it can only increase over time.
  - `Timer`: Generally measures how long a given event type takes (duration), and how often it occurs (frequency).
  - `Distribution`: The statistical distribution of a set of values over a time interval. (This type generally provides raw data, not an aggregation.)

- `Summary`: Tracks the count, sum, and average of the values. Optionally, also tracks quantiles across the values.
- `Histogram`: Tracks the count, sum, and average of the values. Also groups the observations in their corresponding intervals, or buckets.

Remove metrics: Optionally, enter a List of field names to look for when removing metrics. Where a metric's field name matches an element in this list, Cribl Stream will remove that metric from the event.

Dimensions

Add dimensions: Optional list of dimensions to include in events. Supports wildcards. If you don't specify metrics, values will be appended to every metric found in the event. When you add a new metric, dimensions will be present only in those new metrics. Defaults to `!_* *`.

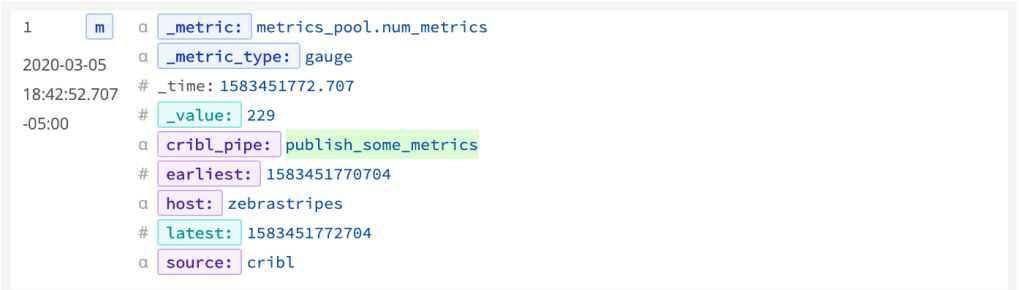
Remove dimensions: Optional list of dimensions to associate with every extracted metric value. Leave blank if this Function processes output from the Aggregations Function as dimensions will be automatically discovered.

The **Add dimensions** and **Remove dimensions** fields support wildcards and negated terms. When you use negated terms, the list is order-sensitive. For example, `!foobar` before `foo*` means "All fields that start with `foo`, except `foobar`." However, `!foo*` before `*` means "All fields, except for those that start with `foo`."

Cribl Stream can send out multi metrics, but this must be configured on compatible Destinations.

Fields Color Coding

On the right Preview pane's **OUT** tab, the Publish Metrics Function adds the following color codes to field labels:



Dimension: purple | **Value**: cyan (light blue) | **Info**: dark blue

These are in addition to the color codes applied to field values, which are listed here.

Examples

Scenario A:

Assume we're working with AWS VPC Flowlog events that have the following structure:

```
version account_id interface_id srcaddr dstaddr srcport dstport protocol packets bytes start end action log_status
```

For example:

```
2 99999XXXXX eni-02f03c2880e4aaa3 10.0.1.70 10.0.1.11 9999 63030 6 6556 262256 1554562460 1554562475 ACCEPT OK
```

... and we want to use values of `packets` and `bytes` as metrics across these dimensions: `action`, `interface_id`, and `dstaddr`.

To reference the `packets` and `bytes` fields by name, as `'packets'` and `'bytes'`, our Pipeline will need a Parser Function before the Publish Metrics Function.

Parser Function

Filter: Set as needed Operation mode: Extract Type: Extended Log File Format (automatically set when specifying a library) Library: AWS VPC Flow Logs Source: `_raw` (No need to specify any other fields.)

Publish Metrics Function

Below, the `metric_name` prefix was arbitrarily chosen. Because there is no JavaScript expression to evaluate - this is literal text - the strings specified for the **Metric name expression** will be identical to those in the final metrics data sent to the Destination. See Raw Output below.

Metrics

Event Field Name	Metric Name Expression	Metric Type
`bytes`	`` `metric_name.bytes` ``	**Gauge**
`packets`	`` `metric_name.packets` ``	**Gauge**

Dimensions

```
action interface_id dstaddr
```

All specified dimension names must align with those from the original event. When you preview the Function's output, the metrics and dimensions will all have special highlighting to separate them from other fields. Additional highlighting is used to differentiate the metrics from the dimensions. (If one or more metrics/dimensions are not highlighted as expected, check the Function's configuration.)

Raw Output

```
metric_name.bytes:262256|g#action:REJECT,interface_id:eni-02f03c2880e4aaa3,dstaddr:10.0.1.11

metric_name.packets:6556|g#action:REJECT,interface_id:eni-02f03c2880e4aaa3,dstaddr:10.0.1.11
```

Compatible Destinations

All text after the `#` symbol represents the dimensions as key-value pairs. In order for dimension data to be included in metrics, the Destination type cannot be standard **StatsD**. However, **StatsD Extended**, **Splunk**, and **Graphite** do support dimensions.

Formatted Output


```
{
  "action": "REJECT",
  "interface_id": "eni-02f03c2880e4aaa3",
  "dstaddr": "10.0.1.11",
  "metric_name.bytes": 262256,
  "metric_name.packets": 6556,
}
```


Copy

Scenario B:

Assume that we want to extract some metrics from specific fields in PANOS logs, whose events have the following structure:

```
future_use_0,receive_time, serial_number, type, threat_content_type, future_use_1, generated_time, source_ip, destination_ip, nat_source_ip, nat_destination_ip, rule_name, source_user, destination_user, application, virtual_system, source_zone, destination_zone, inbound_interface, outbound_interface, log_action, future_use_2, session_id, repeat_count, source_port, destination_port, nat_source_port, nat_destination_port, flags, protocol, action, bytes, bytes_sent, bytes_received, packets, start_time, elapsed_time, category, future_use_3, sequence_number, action_flags, source_location, destination_location, future_use_4, packets_sent, packets_received, session_end_reason, device_group_hierarchy_level_1, device_group_hierarchy_level_2, device_group_hierarchy_level_3, device_group_hierarchy_level_4, virtual_system_name, device_name, action_source,
```

`source_vm_uuid, destination_vm_uuid, tunnel_id_imsi, monitor_tag_imei, parent_session_id, parent_start_time, tunnel_type, sctp_association_id, sctp_chunks, sctp_chunks_sent, sctp_chunks_received`

For example:

```
Jan 10 10:19:15 DMZ-internal.nsa.gov 1,2019/01/10
10:19:15,001234567890002,TRAFFIC,drop,2304,2019/01/10
10:19:15,209.118.103.150,160.177.222.249,0.0.0.0,0.0.0.0,InternalServer,,,not-
applicable,vsys1,inside,z1-FW-Transit,ethernet1/2,,All traffic,2019/01/10
10:19:15,0,1,63712,443,0,0,0x0,udp,deny,60,60,0,1,2019/01/10
10:19:15,0,any,0,0123456789,0x0,Netherlands,10.0.0.0-10.255.255.255,0,1,0,policy-
deny,0,0,0,0,,DMZ-internal,from-policy,,,0,,0,,N/A,0,0,0,0,1202585d-b4d5-5b4c-aaa2-
d80d77ba456e,0
```

Our goal is to use the four values of `bytes_sent`, `bytes_received`, `packets_sent`, and `packets_received` as metrics across these dimensions: `destination_ip`, `inbound_interface`, `outbound_interface`, and `destination_port`.

Here again, our Pipeline will need a Parser Function before the Publish Metrics Function.

Parser Function

Filter: Set as needed Operation mode: Extract Type: Extended Log File Format (automatically set when specifying a Library) Library: Palo Alto Traffic Source: `_raw` (No need to specify any other fields.)

Publish Metrics Function

Set up the Publish Metrics Function as follows.

Metrics

Event Field Name	Metric Name Expression	Metric Type
`bytes_sent`	`` `metric.${host}.bytes_sent` ``	Counter
`bytes_received`	`` `metric.${host}.bytes_rcvd` ``	Counter
`packets_sent`	`` `metric.${host}.pkts_sent` ``	Counter
`packets_received`	`` `metric.${host}.pkts_rcvd` ``	Counter

Added Dimensions

`destination_ip`, `inbound_interface`, `outbound_interface`, `destination_port`

Raw Output

```
metric.10.10.12.192.bytes_sent:60|c|#destination_ip:160.177.222.249,inbound_interface:et
metric.10.10.12.192.bytes_rcvd:0|c|#destination_ip:160.177.222.249,inbound_interface:eth
metric.10.10.12.192.pkts_sent:1|c|#destination_ip:160.177.222.249,inbound_interface:ethe
metric.10.10.12.192.pkts_rcvd:0|c|#destination_ip:160.177.222.249,inbound_interface:ethe
```

Here again, all text after the `#` symbol represents the dimensions as key-value pairs. (See the Compatible Destinations note above.) Unlike the first example, this example uses JavaScript expressions, which you can see evaluated in the raw output where the `${host}` has been converted to `10.10.12.192`.

7.29. Redis

The Redis Function interacts with Redis stores, setting and getting key-hash and key-value combinations. It uses in-memory caching of these key pairs to enable large lookup tables that would be cumbersome with a CSV or binary lookup file.

You can use Cribl Stream Collectors (for example, a REST Collector) to retrieve reference data from desired endpoints, and then use this Function to store the data on Redis and retrieve it to enrich your data. By default, Cribl Stream does not cache the data returned from this Redis Function, but you can enable client-side caching.

Your Redis Function can interact with three kinds of Redis instances: Standalone (the default); Redis Cluster, which scales horizontally; or Redis Sentinel, a distributed system that provides high availability.

Managing Redis Connections

Using Redis Functions can create a large number of TCP connections to the Redis server ("Redis connections" in short). This is because Cribl Stream creates a separate TCP connection to Redis for every Redis Function and every *reference to* a Redis Function. This means that certain usage patterns will increase the number of Redis Connections, including:

- Sending the output of a Chain Function to a regular Pipeline that includes a Redis Function.
- Using a Redis Function in a Source's pre-processing Pipeline.
- Using a Redis Function in a Destination's post-processing Pipeline.

To limit the number of Redis connections, you can *reuse* connections that share certain properties. These properties depend on the type of the Redis instance:

- Standalone: username, password, and URL.
- Cluster: username, password, and all root node hosts and ports.
- Redis Sentinel: username, password, master name, and all root node hosts and ports.

To enable reusing Redis connections:

1. In the sidebar, select **Worker Groups** and select the Worker Group you want to configure.
2. On the **Worker Group** submenu, select **Group Settings**.
3. Select **General Settings** > **Limits** > **Redis** > **Connections**.
4. Toggle **Reuse Redis connections** on.
5. Configure **Connection limit** and confirm with **Save**.

In a Single-instance deployment, the Redis connections settings are located in the main **Settings** menu.

When you enable **Reuse Redis connections**, Cribl Stream attempts to reuse an existing Redis connection instead of creating a new one if:

- A new Redis connection is necessary, and
- The new connection's properties match those of an existing, open connection.

The example below illustrates how powerfully this can affect the demands Cribl Stream places on a Redis server.

Determining Number of Connections

Connection limit is the upper limit of connections to Redis that Cribl Stream can establish before starting to reuse them. To determine the number to configure for your deployment, consider among others:

- Type of Redis instance you're connecting to (Standalone, Cluster, or Redis Sentinel)
- Size of the Redis server
- Number of connections the Redis server is configured to allow
- Number of other clients using the same Redis server

If you already have Redis connections established, be careful not to reduce **Connection limit** *below* the current connection number.

Example: Managing Redis Connections

Referencing a Redis Function from a Pipeline and/or another Function can dramatically multiply the number of Redis connections created, unless we mitigate this effect by applying the **Reuse Redis connections** control. Let's see how this plays out in a detailed example. Suppose we do the following:

1. Create a Pipeline (we'll call it "Pipeline Zero") that contains 10 Redis Functions that connect to the same Redis server, with identical connection properties. Cribl Stream will create 10 TCP connections to the Redis server ("Redis connections").
2. Reference Pipeline Zero in a Chain Function that's in another Pipeline. This creates 10 more connections, for a total of 20 Redis connections.
3. Reference Pipeline Zero in an additional, separate Chain Function that's in yet another Pipeline, creating 10 more connections, for a total of 30 Redis connections.
4. Use Pipeline Zero as a pre-processing Pipeline for a Source, creating 10 more connections, for a total of 40 Redis connections.

5. Use Pipeline Zero as a post-processing Pipeline for a Destination, creating 10 more connections. Now we have a grand total of 50 Redis connections.

Now consider the effect of these Pipelines within Worker Nodes and their Worker Processes. For a typical deployment scenario with 10 Worker Nodes, each of which have 10 Worker Processes, the calculation looks like this:

- 50 connections times 10 Worker Processes = 500 connections, times 10 Worker Nodes = 5000 connections.

A single Redis server can be hard-pressed to handle so many connections. Apart from that, the number of connections is unnecessarily high for the expected throughput. Let's use **Reuse Redis connections** to reduce the number of Redis connections.

Enable **Reuse Redis connections** and set **Connection limit** to `4`. Now revise the calculation:

- 4 connections times 10 Worker Processes = 40 connections, times 10 Worker Nodes = 400 connections.

This reduces the number of connections by better than a factor of 10, making our deployment much less likely to overwhelm the Redis server.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Result field: Name of the field in which to store the returned value. (Leave empty to discard the returned value.)

Command: Redis command to perform. Required. (A complete list of Redis commands is at: https://redis.io/commands.)

Key: A JavaScript expression to compute the value of the key to operate on. Can also be a constant such as `username`. This is a required field. Select the icon at right to open a validation modal.

Args: A JavaScript expression to compute arguments to the operation. Select the icon at right to open a validation modal. When the expression you enter is a plain value or an array, it resembles the options and arguments of a Redis command in the Redis CLI.

For example:

- Suppose you want to `SET` a key named `_mascot`. And, of the options that the `SET` command takes, you want to use two: `GET` and `EX`. The first, `GET`, takes no arguments; the second, `EX`, takes an integer in

seconds.

- You want to set `_mascot` to the value `goat`, and `EX` to 5 seconds. Here is the array you'd enter in the Args column:
  `['goat', 'GET','EX',5 ]`

Deployment Type

From the drop-down, select `Standalone`, `Cluster`, or `Sentinel`. Some of these options display additional controls below. Hostnames must be JavaScript expressions (which can evaluate to a constant value), enclosed in quotes or backticks. They can be evaluated only at init time.

Standalone

This is the default deployment type. Enter a **Redis URL** to connect to. The format is:
`[redis[s]:]//[[user][:password@]][host][:port][/db-number][?db=db-number[&password=bar[&option=value]]]`

Example:
`redis://user:secret@localhost:6379/0?foo=bar&qux=baz`

Example with no `user` specified:
`redis://secret@localhost:6379/0?foo=bar&qux=baz`

Standalone with TLS

To enable TLS, specify a Redis URL that starts with `rediss://` rather than `redis://`. This URL reveals the **TLS** section of the config UI.

Example:
`rediss://user:secret@localhost:6379/0?foo=bar&qux=baz`

Example with no `user` specified:
`rediss://secret@localhost:6379/0?foo=bar&qux=baz`

Cluster

Specify **Root nodes** that the cluster will connect to. A single node is sufficient, but Cribl recommends listing two or more. Each of the **Root Nodes** consists of a **Hostname** and a **Port**.

In multi-key commands, ensure that all keys in the command are in the same hash slot. Otherwise, the command will fail. As a workaround, you can split the multi-key command into separate commands, where each command operates only on keys from a single hash slot.

Scale Reads determines which nodes in the Redis cluster should handle read commands. This setting helps optimize read performance and manage load distribution across the cluster. The available options are:

- **Masters**: Directs all read commands to master nodes only. This ensures strong consistency but may limit read scalability.
- **Replicas**: Directs all read commands to replica nodes only. This can improve read scalability but may result in stale data due to eventual consistency.
- **Masters and Replicas**: Distributes read commands across both master and replica nodes. This option balances load and improves read performance but may occasionally return stale data.

Sentinel

Specify a **Master group name**, plus every **Sentinel** in your Redis deployment. Each **Sentinel** consists of a **Hostname** and a **Port**.

Authentication Method

Use the **Authentication method** buttons to select one of the following options, some of which display additional controls below.

- **None**: Select this option where authentication either is not required, or is provided in the URL.
- **Basic**: This displays **Username** and **Password** fields for you to enter your Redis credentials.
- **User Secret**: This option exposes a drop-down in which you can select a stored text secret that references a Redis username and password, as described above. A **Create** link is available to store a new, reusable secret.
- **Admin Secret**: This option exposes a drop-down in which you can select a stored text secret that references a Redis admin password. A **Create** link is available to store a new, reusable secret.

TLS

Validate server certs: Reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (such as the system's CA). Defaults to toggled on.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting. The minimum TLS version for Redis is version 6.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Minimum TLS Requirement

The minimum Redis version required for TLS support is version 6.

Advanced Settings

Blocking time limit: Maximum amount of time (in seconds) before assuming that Redis is down and passing events through. Defaults to `60` seconds. Use `0` to disable timeouts.

Client-side cache: Toggle on to enable. Then navigate to **Settings** > **General Settings** > **Limits** > **Redis Cache** to configure the cache, as described in the next section.

Overall Redis Cache Settings

Together, these settings determine how the Cribl Stream client-side cache behaves. We'll start with definitions:

Key TTL in seconds: The maximum time-to-live of a key in the cache, in seconds. `0` means no limit. Defaults to 10 minutes.

Max # of keys: The maximum number of keys the cache can contain. `0` (the default) means no limit.

Max cache size (bytes): The maximum number of bytes the cache can contain. `0` (the default) means no limit.

Service period (seconds): The interval at which the cache checks whether any keys need to be evicted. Defaults to 30 seconds.

Server assisted: This feature requires Redis version 6 or newer. Toggle on to enable server-assisted caching and display the **Client tracking mechanism** drop-down. See the Tradeoffs section for an explanation of the available options.

Now here's the context:

For each Redis instance your Redis Functions interact with, Cribl Stream will operate one client-side cache. That cache will be shared between all the Redis Functions instances configured to interact with it. This is why the settings are in a central location, and why you cannot configure the cache at the individual function level.

Once per **Service period**, the cache checks whether any keys need to be evicted. The cache will evict:

- The least-recently-accessed key, if the cache has reached either (1) its **Max cache size (bytes)** or (2) its **Max # of keys**.
- Any key that exceeds its **Key TTL in seconds**.
- Any key that is stale according to a notification to Cribl Stream from your Redis instance. Redis sends these notifications (a.k.a. invalidations, or invalidation messages) only when you have enabled **Server assisted** caching.

The overall benefit that client-side caching provides is having fewer stale keys in your cache for less time. The greater the proportion of `GET`-type, as compared to `SET`-type (write) commands your Cribl Stream-Redis interactions have, the greater that benefit will be. When your Cribl Stream Pipelines either have no `SET`-type operations, or use filters to keep those operations to a minimum, you should consider client-side caching. In sum, you must decide whether more accurate data in your cache is worth the cost of memory and processing.

Tradeoffs Inherent in Server-Assisted Caching

Consider the common scenario where other applications besides Cribl Stream are writing to your Redis instance. Say one of those applications modifies a key's value. Now the value that Cribl Stream client-side cache has for that key is stale. Cribl Stream does not "know" about this, and retains the stale value until its TTL is up. This degrades the overall accuracy of data from the client-side cache.

Server-assisted caching is a remedy for this problem. With server-assisted caching enabled, your Redis instance will notify your client-side cache when Redis determines that a key's value has changed. Then the Cribl Stream client-side cache will immediately evict the stale key. Now the client-side cache is more accurate. Redis also has documentation about server-assisted client-side caching.

There are two mutually-exclusive ways you can have Redis notify Cribl Stream. These are the two modes (delivery mechanisms) available in the **Client tracking mechanism** drop-down:

1. In **Default** mode, the Redis server remembers which keys the Cribl Stream client has requested. Redis only notifies Cribl Stream when the value of one of **those** keys has changed. This is most efficient for Cribl Stream but needs more memory and some extra processing time from the Redis server.
2. In **Broadcast** mode, the Redis server need not remember which keys came from where, because it simply notifies Cribl Stream when the value of **any** key has changed. This is most efficient for Redis, but now the Cribl Stream client needs to inspect every key that Redis says has changed. This consumes some extra processing time on the client, and generates extra network activity to accommodate notifications about keys that Cribl Stream never sees.

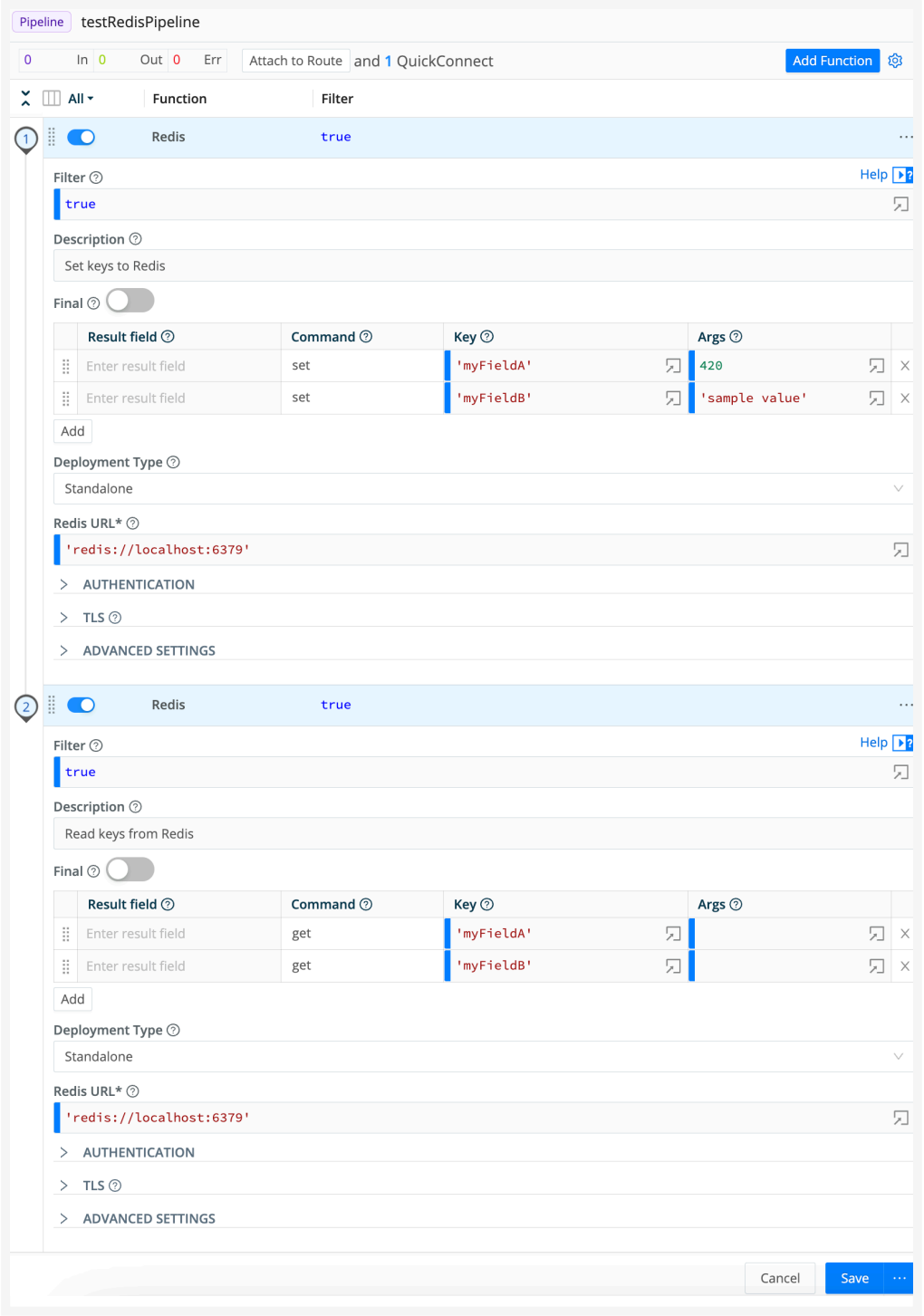
You should select whichever of these mechanisms best suits your environment. Does it make more sense to impose the heavier burden on the Redis server (as in **Default** mode) or on Cribl Stream (as in **Broadcast** mode)?

In both scenarios, for a given key, any write operation from any application (including Cribl Stream) that's interacting with Redis, causes Cribl to evict the key from the client-side cache when Redis notifies Cribl.

Examples

Scenario A: Set and Get

This Pipeline demonstrates the use of a pair of Redis Functions. The first Function sets two key-value pairs in Redis. The second Function gets their values, by key, into two corresponding new **Result field**s.



Redis set and get Functions

Redis Function #1

Description: `Set keys to Redis`

Command: `set` **Key:** `'myFieldA'` **Args:** `420`

Command: `set` **Key:** `'myFieldB'` **Args:** `'sample value'`

Redis Function #2

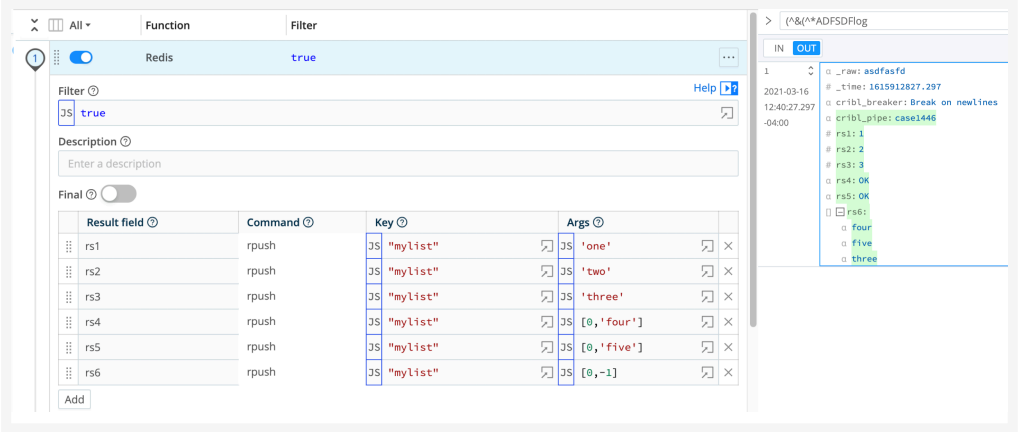
Description: `Read keys from Redis`

Result field: `myField_AA` **Command**: `get` **Key**: `'myFieldA'`

Result field: `myField_BB` **Command**: `get` **Key**: `'myFieldB'`

Scenario B: Multiple-Argument Arrays

This example demonstrates how to configure a Redis Function that supplies an array of multiple arguments to Redis commands (in this example, `lset` and `lrange`).



Redis Function, arrays of multiple arguments, and sample output

Redis Function

Description: `Push arrays of multiple arguments to Redis`

Result field: `rs1` **Command**: `rpush` **Key**: `"mylist"` **Args**: `'one'`

Result field: `rs2` **Command**: `rpush` **Key**: `"mylist"` **Args**: `'two'`

Result field: `rs3` **Command**: `rpush` **Key**: `"mylist"` **Args**: `'three`

Result field: `rs4` **Command**: `lset` **Key**: `"mylist"` **Args**: `[0,'four']`

Result field: `rs5` **Command**: `lset` **Key**: `"mylist"` **Args**: `[-2,'five']`

Result field: `rs6` **Command**: `lrange` **Key**: `"mylist"` **Args**: `[0,-1]`

Try This at Home

The Pipeline below contains only this example Function. You can import it into your own Cribl Stream environment, fill in the `url` with your own credentials, and further modify it to meet your needs.

redis-multiple-args.json

```
{
  "id": "redis-multiple-args",
  "conf": {
    "output": "default",
    "groups": {},
    "asyncFuncTimeout": 1000,
    "functions": [
      {
        "filter": "true",
        "conf": {
          "commands": [
            {
              "outField": "rs1",
              "command": "rpush",
              "keyExpr": "\"mylist\"",
              "argsExpr": "'one'"
            },
            {
              "outField": "rs2",
              "command": "rpush",
              "keyExpr": "\"mylist\"",
              "argsExpr": "'two'"
            },
            {
              "outField": "rs3",
              "command": "rpush",
              "keyExpr": "\"mylist\"",
              "argsExpr": "'three'"
            },
            {
              "command": "lset",
              "keyExpr": "\"mylist\"",
```

7.30. Regex Extract

The Regex Extract Function extracts fields using named capture groups. Fields that start with `__` (double underscore) are special in Cribl Stream. They are ephemeral: any Function downstream can use them, but they will be serialized only to Cribl internal Destinations.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of the Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Regex: Regex literal. Must contain named capturing groups: `(?<foo>bar)`. Can contain special `_NAME_N` and `_VALUE_N` capturing groups, which extract **both the name and value** of a field: `(?<_NAME_0>[^\s=]+)=(?<_VALUE_0>[^\s]+)`. Defaults to empty. You can type in a pattern, or use one of the predefined Regexes. See Examples below.

Additional regex: Select **Add Regex** to chain extra regex conditions.

Source field: Field on which to perform regex field extraction. This field supports nested addressing. Defaults to `_raw`.

Overwrite existing fields: Whether to overwrite existing event fields with extracted values. If toggled off (default), the function will convert existing fields to an array. If toggled on, Regex Extract will create array fields if applied multiple times, or if fields exist. For example, if you extract `src_ip` in an input Pipeline where the assigned value is `10.1.2.2`, and in a processing Pipeline with a value of `10.2.3.3`, then the resulting field is `["10.1.2.2", "10.2.3.3"]`.

Advanced Settings

Max exec: The maximum number of times to apply the **Regex** to the source field when the global flag is set, or when using `_NAME_N` and `_VALUE_N` capturing groups. Named capturing groups will always use a value of `1`. Defaults to `100`.

Field name format expression: JavaScript expression to format field names when using the `_NAME_n` and `_VALUE_n` capturing groups. For example, to append `XX` to all field names, use: `` `${name}_XX` `` (backticks are literal). If not specified, names will be sanitized using regex: `/^[_0-9]+|[^a-zA-Z0-9_]+/g`. The **original** field name is in the global `name`. You can access other fields' values via `__e.<fieldName>`.

Examples

To test Regex Extract Functions use Data Preview to see the events as they flow into and out of the Pipeline.

Example 1: Single Field from Simple Event

Assume a simple event that looks like this: `metric1=23 metric2=42 dc=23 abc=xyz`

Extract **only** the `metric1` field:

Regex: `metric1=(?<metric1>\d+)` **Result**: `metric1:"23"`

Example 2: Extracting Multiple Fields

Use this sample data:

```
462559d4a487[471]: 172.23.0.6 - - [26/Feb/2024:20:22:38 +0000] "GET /catalog/view/jav
```

Copy

Use a regex to extract the fields you know are in the event.

Regex: `^(?<container_id>[^\s]+)\[(?<process_id>\d+)\]:\s+(?<remote_host>[^\s]+)\s+(?<remote_user>-)\s+(?<auth_user>-)\s+\[(?<timestamp>[^\]]+)\]\s+"(?<request_method>\w+)\s+(?<requested_url>[^\s]+)\s+(?<http_version>[^"]+)"\s+(?<status>\d+)\s+(?<bytes>.+)$`

Results:

Example 2 results

Example 3: Key-Value Pairs from Multiple Fields

Use this sample data:

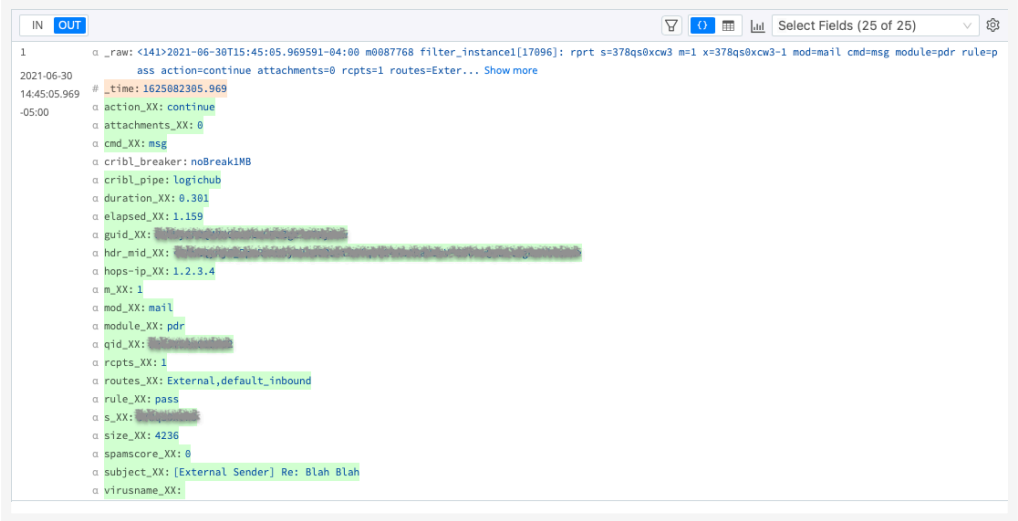
```
rec_type=71 rec_type_simple=RNA dest_port=443 snmp_out=0 netflow_src="00000000-0000-0
```

Copy

Use a regex to extract **all** k=v pairs, then use **Field Name Format Expression** to append an `_XX` suffix to each extracted field:

Regex: `(?<_NAME_0>[\w-]+)="?(?<_VALUE_0>(?<=")[^"]*|\S*)` **Field Name Format Expression**: `${name}_XX`

Results:



Example 3 results

Example 4: Multi-Stage Extraction, Complex Events

This example builds on the syntax in Example 3, to tackle a more complex event structure.

In the right **Sample Data** pane, select **Paste** and insert the following sample:

Sample Data

```
<134>1 2020-12-22T17:06:08Z CORP_INT_NLB CheckPoint 18160 - [action:"Accept"; conn_di
```

Copy

This event is from a CheckPoint Firewall CMA system. With this type of event structure, properly extracting each event field into a separate metadata field requires two-stage processing. So we'll use two Regex Extract Functions.

The first Regex Function splits the event to separate the actual data from the header information. We'll split after the `CheckPoint 18160` string, by capturing everything between the `[` and `]`:

Regex: `\[(?<__fields>.*)\]` **Source**: `_raw`

Next, add this second Regex Extract Function to extract all k=v pairs:

Regex: `(?<_NAME_0>[^ :]+):(?<_VALUE_0>[^;]+);` **Source**: `__fields`

Results:

Example 4 results

For further examples, see Using Cribl to Analyze DNS Logs in Real Time - Part 2.

7.31. Regex Filter

The Regex Filter Function filters out events based on regex matches.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Regex: Regex to test against. Defaults to empty. You can type in a pattern, or use one of the predefined Regexes.

Additional regex: Select **Add Regex** to chain extra regex conditions.

Field: Name of the field to test against the regex. Defaults to `_raw`. Supports nested addressing.

Examples

For a usage example, see our Regex Filtering topic.

7.32. Rename

The Rename Function changes field names or reformats their names (such as normalizing names to camelcase).

You can use Rename to change specified fields (much like the Eval Function), or for bulk renaming based on a JavaScript expression (much like the Parser Function). Compared to these alternatives, Rename offers a streamlined way to alter only field names, without other effects.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Optionally, enter a simple description of this step in the Pipeline. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Parent fields: Specify fields whose children will inherit the **Rename fields** and **Rename expression** operations. Supports wildcards. If empty, only top-level fields will be renamed.

This Function cannot operate on internal fields whose names begin with double underscores (`__`).

Note that you can still use `__e` to access fields within the `(context)` object, as long as those fields do not begin with double underscores. That's because `__e` is a special variable, not a field.

Rename fields: Each row here is a key-value pair that defines how to rename fields. The current name is the key, and the new name is the value. Select **Add Field** to add more rows.

- **Current Name**: Original name of the field to rename. You must quote literal identifiers (non-alphanumeric characters such as spaces or hyphens).
- **New Name**: New or reformatted name for the field. Here again, you must quote literals.

Rename expression: Optional JavaScript expression whose returned value will be used to rename fields. Use the `name` and `value` global variables to access fields' names/values. Example: `name.startsWith('data') ? name.toUpperCase() : name`. You can access the other fields' values via `__e.<fieldName>`.

A single Function can include both **Rename fields** (to rename specified field names) and **Rename expression** (to globally rename fields). However, the **Rename fields** strategy will execute first.

Advanced Settings

Parent field wildcard depth: For wildcards specified in **Parent fields**, sets the maximum depth within events to match and rename fields. Enter `0` to match only top-level fields. Defaults to `5` levels down.

Example

Change the `level` field, and all fields that start with `out`, to all-uppercase.

Example event:

```
{"inEvents": 622,
  "level": "info",
  "outEvents": 311,
  "outBytes": 144030,
  "activeCxn": 0,
  "openCxn": 0,
  "closeCxn": 0,
  "activeEP": 105,
  "blockedEP": 0
}
```

Copy

Rename Fields:

Current name: `level`
New name: `LEVEL` **Rename expression**: `name.startsWith('out') ? name.toUpperCase() : name`

Event after Rename:

```
{"inEvents": 622,
  "LEVEL": "info",
  "OUTEVENTS": 311,
  "OUTBYTES": 144030,
  "activeCxn": 0,
  "openCxn": 0,
  "closeCxn": 0,
  "activeEP": 105,
  "blockedEP": 0
}
```

Copy

Applications

1. Remove filename prefix `<myPrefix>`:

Rename expression: `name.replace(/<myPrefix>/, '')`

2. Add a wildcard to rename a set of fields named `json.record[0]`, `json.record[1]`, etc., preserving the variable numbers in the brackets:

Rename expression: `name.replace(/(json)\.(\w+)/,'MYNEWNAME-$2-$1')`

See Also {see-also}

- Reducing Windows XML Events, which demonstrates using Rename to remove unnecessary prefixes from field names, reducing events' size.

7.33. Rollup Metrics

The Rollup Metrics Function merges frequently generated incoming metrics into more manageable time windows.

Each Worker Process executes this Function independently on its share of events. For details, see Functions and Shared-Nothing Architecture.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Optional description of this Function's purpose in this Pipeline. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Dimensions: List of data dimensions across which to perform rollups. Supports wildcards. Defaults to `*` wildcard, meaning all original dimensions.

Time window: The time span over which to roll up (aggregate) metrics. Must be a valid time string (such as `10s`). Must match pattern: `\d+[sm]$`.

With high-cardinality data, beware of setting long time windows. Doing can cause high memory consumption and/or lost data, because memory is flushed upon restarts and redeployments.

Gauge update: The operation to use when rolling up gauge metrics. Defaults to **Last**; other options are **Maximum**, **Minimum**, or **Average**.

Examples

Scenario A:

Assume that you have metrics coming in at a rate that is too high. For example, Cribl Stream's internal metrics come in at a 2s interval.

To roll up these metrics to 1-minute granularity, you would set up the Rollup Metrics Function with a **Time Window** value of `60s`.

Scenario B:

Assume that you have metrics coming up with multiple dimensions - such as `host`, `source`, `data_center`, and `application`. You want to aggregate these metrics to eliminate some dimensions.

Here, you would configure Rollup Metrics Function with a **Time Window** value that matches the metrics' generation - such as `10s`. In the **Dimensions** field, you would remove the default `*` wildcard, and would specify only the dimensions you want to keep - such as `host`, `data_center`.

7.34. Sampling

The Sampling Function filters out events, based on an expression and a sampling rate.

Each Worker Process executes this Function independently on its share of events. For details, see Functions and Shared-Nothing Architecture.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

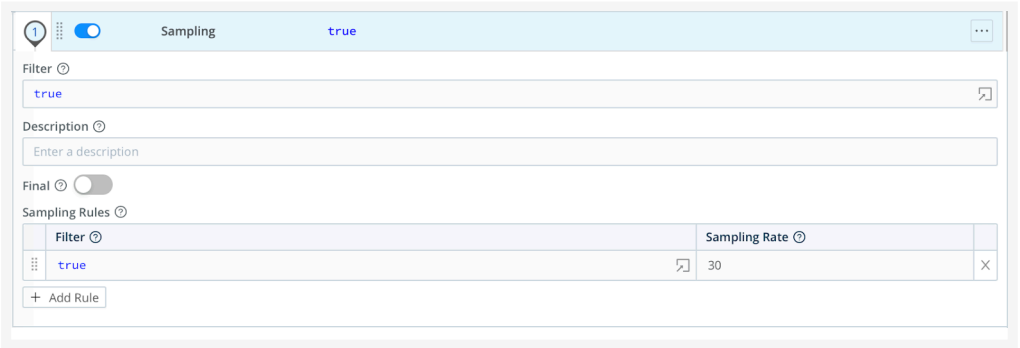
Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Sampling rules: Events matching these rules will be sampled at the rates you specify:

- **Filter**: Filter expression matching events to be sampled. Use `true` to match all.
- **Sampling Rate**: Enter an integer `N`. (Defaults to `1`.) Sampling will pick `1/N` events matching this rule.

How It Works

Setting this Function's **Sampling rate** to `30` would mean that only 1 of every 30 events would be kept.



Sampling Rate

Let's assume that we save this setting, and then capture data from a datagen Source by selecting **Preview** > **Start a Capture** > **Capture**. In the **Capture Sample Data** modal, select: `100` seconds, `100` events, and **As they come in**. Then start the capture, and **Save as Sample File**.

Next, in the **Preview** pane, select **Simple** beside the new file's name. If you then select the **Basic Statistics** (chart) button, you should see that we've kept about 4 of the original 100 events, or close to 1 in 30.

	Full Event Length ⓘ	Number of Fields ⓘ	Number of Events ⓘ
IN	28.82KB	41	100
OUT	1.42KB	38	4
DIFF	↓ -95.08%	↓ -7.32%	↓ -96.00%

Examples and Scenarios

For usage examples, see these Better Practices topics:

- Sampling: Sampling verbose or voluminous data at ingest time, to make analysis and troubleshooting more efficient.
- Sample Logs: Sampling event data for efficient onboarding into Cribl Stream.
- Access Logs: Apache, ELB, CDN, S3, Etc.: Sampling voluminous logs from sources like Amazon S3, Amazon Cloudfront, and AWS ELBs.
- Firewall Logs: VPC Flow Logs, Cisco ASA, Etc.: Sampling voluminous logs from firewalls.

See Also

- Dynamic Sampling Function: Filter out events based on an expression, a sample mode, and the volume of events

7.35. Serialize

Use the Serialize Function to convert an event's content into a predefined format.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Type: Data output format. Defaults to `CSV`. Options:

- **CSV**
- **Extended Log File Format** - Extended Log File Format.
- **Common Log Format** - Common Log Format.
- **Key=Value Pairs** - selecting this type displays additional options:
  - **Clean fields**: Toggle on to clean field names by replacing non-alphanumeric characters with `_`. This will also strip leading and trailing `"` symbols.
  - **Pair delimiter**: Delimiter used to separate `key=value` pairs. Defaults to a single space character. Multiple characters are allowed. Cannot be an equal sign (`=`).
- **JSON Object**
- **Delimited values** - selecting this type displays additional options:
  - **Delimiter**: Delimiter character to split value. If left blank, defaults to comma (`,`). You can also specify pipe (`|`) or tab characters.
  - **Quote char**: Character used to quote literal values. If left blank, defaults to `"`.
  - **Escape char**: Character used to escape delimiter or quote characters. If left blank, defaults to **Quote char**.
  - **Null value**: Field value representing the null value. These fields will be omitted. Defaults to: `-`

Library: Browse Parser/Formatter library.

Fields to serialize: List of fields to include in the output after this Function processes the event. For `CSV`, `ELFF`, `CLF`, and `Delimited values` formats, you must list each field explicitly. Wildcards (`*`) are not supported for these formats. For all other formats, you can use wildcard (`*`) field lists, which support both:

- Wildcards (`*`) to include multiple fields.
- Nested addressing, where you can reference a parent field and all its children using a wildcard. For example, if you convert `_raw` into an object, use `_raw*` to refer to `_raw` and all nested fields inside it. See Wildcard Lists for more information.

When using wildcard lists, all exclusions (such as `!fieldname`) must come before any inclusions (such as `fieldname` or `*`).

Source field: Field containing the object to serialize. Leave blank to serialize top-level event fields.

Destination field: Field to serialize the data into. Defaults to `_raw`.

Examples

Scenario A: JSON to CSV

Assume a simple event that looks like this: `{"time":"2019-08-25T14:19:10.240Z","channel":"input","level":"info","message":"initializing input","type":"kafka"}`

We want to serialize these fields: `_time`, `channel`, `level`, and `type` into a single string, in CSV format, stored in a new destination field called `test`.

To properly extract the key-value pairs from this event structure, we'll use a built-in Event Breaker:

1. Copy the above sample event to your clipboard.
2. In the **Preview** pane, select **Paste a Sample**, and paste in the sample event.
3. Under **Select Event Breaker**, choose **ndjson** (newline-delimited JSON), and select **Save as a Sample File**.

Now you're ready to configure the Serialize Function, using the settings below:

Type: CSV Fields to Serialize: `_time channel level type` Destination Field: `test` Source Field: [leave empty] **Result**: `test: 1566742750.24,input,info,kafka`

In the new `test` field, you now see the `time`, `channel`, `level`, and `type` keys extracted as top-level fields.

Scenario B: CSV to JSON

Let's assume that a merchant wants to extract a subset of each customer order, to aggregate anonymized order statistics across their customer base. The transaction data is originally in CSV format, but the statistical data must be in JSON.

Here's a CSV header (which we don't want to process), followed by a row that represents one order:

```
orderID,custName,street,city,state,zip 20200622102822,john smith,100 Main St.,Anytown,AK,99911
```

To convert to JSON, we'll need to first parse each field from the CSV to a manipulable field in the Pipeline, which the Serialize Function will be able to reference. In this example, the new manipulable field is `message`.

Use the `Parser` Function:

Filter: `true` Operation mode: `Extract` Type: CSV Source field: `_raw` Destination field: `message` List of fields: `orderID custName street city state zip`

Now use the Serialize Function:

Filter: `true` Type: `JSON` Fields to serialize: `city state` Source field: `message` Destination field: `orderStats`

7.36. SNMP Trap Serialize

Use the SNMP Trap Serialize Function to convert compliant events into SNMP traps for sending out through SNMP Trap Destinations that were modified within a Pipeline from an SNMP Trap Source. This ensures compatibility with existing SNMP infrastructure and supports integration with network management tools that use SNMP for monitoring and alerting.

Limitations

- The SNMP Trap Source the events came from must have **Include varbind types** enabled for this function to work. The SNMP Trap Serialize Function needs the varbind types to correctly serialize the events.
- For v2c events, `sysUpTime` and `snmpTrapOID` appear both at the top level and within varbinds. During serialization, the Function uses the values in varbinds as they reflect the event's generation time. If the values differ, varbinds take precedence. Modifying the top-level values will not affect serialization.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Enforce required fields: When toggled on (default), v2c and v3 traps must meet the following requirements:

- At least two varbinds must be present.
- The first varbind must have the `sysUpTime` OID (`1.3.6.1.2.1.1.3.0`) and type `67` (`TimeTicks`).
- The second varbind must have the `snmpTrapOid` OID (`1.3.6.1.6.3.1.1.4.1.0`) and type `6` (`OID`).

These requirements are mandated by RFC 3416 Section 4.2.6, RFC 1066 Section 5.1, and RFC 1905 Section 3.1.2. While traps can be serialized without these fields, downstream receivers may drop them. With this option enabled, non-compliant traps are dropped before being sent downstream.

Drop failed events: When toggled on (default), events that fail to serialize into valid traps are dropped entirely, preventing them from passing through the Pipeline. This is useful when sending to an SNMP Trap Destination, as invalid traps will not be sent.

When toggled off, the event will pass through without `__snmpRaw` and will include a `snmpSerializeErrors` field for troubleshooting.

Regardless of whether **Drop failed events** is toggled on or off, if an event fails to serialize, the `__snmpRaw` field is deleted, preventing the event from leaving the SNMP Trap Destination.

Recommended setup flow:

1. Get a sample of the traps to modify and configure the SNMP Trap Serialize Function.
2. Enable **Drop failed events** and run a Data Preview.
3. Modify the SNMP Trap Serialize Function and preceding Eval Function until no `snmpSerializeErrors` remain.
4. Disable **Drop failed events** when confident in the Pipeline setup and ready to flow actual data.

SNMPV3 User: SNMPv3 provides enhanced security by enabling user authentication and data encryption. These settings are crucial for ensuring that only authorized users can send SNMP traps and that the data remains secure during transmission.

- **Username**: The username for SNMP v3 authentication. Ensure the username matches the one configured on the SNMP manager.
- **Authentication protocol**: The protocol to use for SNMP v3 authentication. Options include `None`, `MD5`, `SHA1`, `SHA224`, `SHA256`, `SHA384`, and `SHA512`.
    - **V3 authentication key**: The key used for SNMPv3 authentication. This field appears when an **Authentication protocol** is selected.
    - **Privacy protocol**: The protocol to use for SNMPv3 encryption. This field appears when an **Authentication protocol** is selected, as it is not possible to have an encrypted trap without authentication. Options include:
        - `None`
        - `AES128`
        - `AES256b (Blumenthal)`
        - `AES256r (Reeder)`
    - **V3 privacy key**: The key used for SNMPv3 encryption. This field appears when a **Privacy protocol** is selected.

Maintain Original Source IP in SNMP Traps

This setup maintains the original source IP in SNMP traps by adding it to the varbinds array, ensuring accurate data transmission even when traps traverse network devices that alter IP headers.

Before the SNMP Trap Function in your Pipeline, use the Eval Function to append a new object to the varbinds array. The object includes an OID, type, and the value from the host field, effectively capturing the original source IP. For example:

- **Name**: `varbinds`
- **Value expression**: `[...varbinds, {oid: '.1.3.6.1.6.3.18.1.3', type: 64, value: __e.host}]`

Next, for the SNMP Trap Serialize Function we recommend enabling both **Enforce required fields** and **Drop failed events**. This will ensure that the SNMP trap is serialized correctly, enforcing necessary fields and dropping events that fail serialization.

Change Authorization and/or Privacy Level

To change the authorization or privacy level of an SNMP v3 trap, you must modify two settings:

- **SNMPV3 User**: Update the user configuration within the SNMP Trap Serialize Function.
- **msgFlags**: Set the appropriate value at the top level of the event in an Eval Function before the serialization.

The `msgFlags` value should be a string representation of the desired number:

- `00`: No authorization, no privacy
- `01`: Authorization, no privacy
- `03`: Authorization and privacy

Change the SNMPv3 User

To change the SNMPv3 user, you need to update the `msgUserName` field within the SNMP Trap object.

Before the SNMP Trap Function in your Pipeline, use the Eval Function to update `msgSecurityParams.msgUserName` to the new intended username. For example, **Eval expression**: `msgSecurityParams.msgUserName = 'newUser';`

7.37. Suppress

The Suppress Function either drops or tags specified events over a specified time period, based on evaluation of a key expression.

Each Worker Process executes this Function independently on its share of events. For details, see Functions and Shared-Nothing Architecture.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Key expression: Suppression key expression used to uniquely identify events to suppress. For example, `` `${ip}:${port}` `` will use the fields `ip` and `port` from each event to generate the key.

Number to allow: The number of events to allow per time period. Defaults to `1`.

Suppression period (sec): The number of seconds to suppress events after **Number to allow** events are received. Defaults to `30`.

Drop suppressed events: Specifies if suppressed events should be dropped, or just tagged with `suppress=1`. Defaults to toggled on, meaning drop.

Advanced Settings

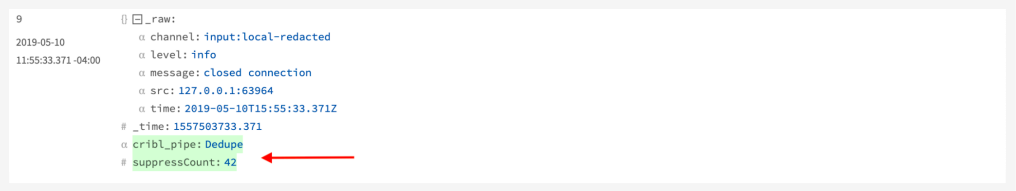
Cache size limit: The maximum number of keys that can be cached before idle entries are removed. Before changing the default `50000`, contact Cribl Support to understand the implications.

Suppression period timeout: The number of suppression periods of inactivity before a cache entry is considered idle. This defines a multiple of the **Suppression period (sec)** value. Before changing the default `2`, contact Cribl Support to understand the implications.

Num events to trigger cache clean-up: Check cache for idle sessions every N events when cache size exceeds the **Cache size limit**. Before changing the default `10000`, contact Cribl Support to understand the implications.

Seeing the Results

If you've enabled **Drop suppressed events**, such events will be omitted from logs as they exit this Function. However, the next event allowed through will include a `suppressCount: N` field, whose `N` value indicates the number of events dropped in the preceding **Suppression period**.



suppressCount shows number of events dropped

Examples

In the examples below, **Filter** is the Function-level Filter expression:

1. Suppress by the value of the `host` field: Filter: `true` Key expression: `host` Number to allow: `1` Suppression period (sec): `30`
   Using a datagen sample as a source, generate at least 100 events over 2 minutes.

Result: One event per unique `host` value will be allowed in every 30s. Events without a `host` field will **not** be suppressed.

2. Suppress by the value of the `host` and `port` tuple : Filter: `true` Key expression: `` `${host}:${port}` `` Number to allow: `1` Suppression period (sec): `300`

Result: One event per unique `host:port` tuple value will be allowed in every 300s.

Suppression will **also** apply to events without a `host` or a `port` field. The reason is that if `field` is not present, `` `${field}` `` results in the literal `undefined`.

3. To **guarantee** that suppression applies **only** to events with `host` and `port`, check for their presence using a Filter: Filter: `host && port!=null` Key expression: `` `${host}:${port}` `` Number to allow: `1` Suppression period (sec): `300`
4. Decorate events that qualify for suppression: Filter: `true` Key expression: `` `${host}:${port}` `` Number to allow: `1` Suppression period (sec): `300` Drop suppressed events: Toggle off.

Result: No events will be suppressed. But all qualifying events will gain an added field `suppress=1`, which can be used downstream to further transform these events.

For further use cases, see Cribl's Streaming Data Deduplication with Cribl blog post. For further details on filter usage, see Build Custom Logic to Route and Process Your Data: Filter Expressions.

7.38. Tee

The Tee Function tees events out to a command of choice, via `stdin`. The output is one JSON-formatted event per line. You can send the events to (for example) a local file on the Cribl Stream Worker. This can be useful in verifying the data being processed in a Pipeline.

The Filesystem/NFS Destination offers similar capability, but only after the data leaves the Pipeline. Tee, by comparison, can be inserted at any point in the Pipeline.

In Cribl.Cloud, the Tee Function is only available on customer-managed hybrid Worker Nodes.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Command: Command to execute and receive events (via `stdin`) - one JSON-formatted event per line.

Args: Select **Add Arg** to supply arguments to the command.

Restart on exit: Restart the process if it exits and/or we fail to write to it. Defaults to toggled on.

Environment variables: Environment variables to set or overwrite. Select **Add Variable** to add key-value pairs.

Communication Protocol

Data is passed to the command through its `stdin`, using the following protocol:

- First line: Metadata serialized in JSON, containing the following fields:
  - **format**: Serialization format for event. Defaults to `JSON`.
  - **conf**: Full Function configuration.
- Remaining: Payload.

Examples

Assume that we are parsing PANOS Traffic logs, and want to see how they look at a particular step in the processing Pipeline. We'll assume that the `Parser` Function is already in place, so we'll insert the Tee Function at any (arbitrary) later point in the Pipeline.

Scenario A:

The Tee Function itself requires only that we define the **Command** field. In this particular example, that **Command** will be `tee` itself.

We've also selected **Add Arg**, to specify a local output file in the resulting **Args** field. (A file path would normally be the first argument to a `tee` command executed from the command line. The Cribl Stream user must have write permission on the specified file path.)

Command: `tee`

Args: `/opt/cribl/foo.log`

In this first scenario, assume that we have the `Parser` configured to parse, but not keep any fields. After changes are deployed and PANOS logs are received, if we tail `foo.log`, we'd see the following:

```
Line 1: {"format":"json","conf":{"restartOnExit":true,"env":{},"command":"tee","args":["/opt/cribl/foo.log"]}
```

```
Line 2: {"_raw":"Oct 09 10:19:15 DMZ-internal.nsa.gov 1,2019/10/09 10:19:15,001234567890002,TRAFFIC,drop,2304,2019/10/09 10:19:15,209.118.103.150,160.177.222.249,0.0.0.0,0.0.0.0,InternalServer,,,not-applicable,vsys1,inside,z1-FW-Transit,ethernet1/2,,All traffic,2019/10/09 10:19:15,0,1,63712,443,0,0,0x0,udp,deny,60,60,0,1,2019/10/09 10:19:15,0,any,0,0123456789,0x0,Netherlands,10.0.0.0-10.255.255.255,0,1,0,policy-deny,0,0,0,0,,DMZ-internal,from-policy,,,0,,0,,N/A,0,0,0,0,1202585d-b4d5-5b4c-aaa2-d80d77ba456e,0","_time":1593185574.663,"host":"127.0.0.1"}
```

In Line 2 above, note that the `_raw` field makes up most of the contents, with only the `_time` and `host` fields added.

Scenario B:

Assume that we use the Tee Function, using the same **Command** and arguments, but we've modified the `Parser` Function to retain five fields: `receive_time`, `source_port`, `destination_port bytes_received`, and `packets_received`.

This time, if we tail `foo.log`, we'll see something like the following. If you compare this output to the previous output example, you'll notice the five fields appended to this event:

```
Line 3: {"_raw":"Oct 09 10:19:15 DMZ-internal.nsa.gov 1,2019/10/09
10:19:15,001234567890002,TRAFFIC,drop,2304,2019/10/09
10:19:15,209.118.103.150,160.177.222.249,0.0.0.0,0.0.0.0,InternalServer,,,not-
applicable,vsys1,inside,z1-FW-Transit,ethernet1/2,,All traffic,2019/10/09
10:19:15,0,1,63712,443,0,0,0x0,udp,deny,60,60,0,1,2019/10/09
10:19:15,0,any,0,0123456789,0x0,Netherlands,10.0.0.0-10.255.255.255,0,1,0,policy-
deny,0,0,0,0,,DMZ-internal,from-policy,,,0,,0,,N/A,0,0,0,0,1202585d-b4d5-5b4c-aaa2-
d80d77ba456e,0","_time":1593185606.965,"host":"127.0.0.1","receive_time":"2019/10/09
10:19:15","source_port":"63712","destination_port":"443","bytes_received":"0","packets_r
```

In this Function's **Command** field, you can specify commands other than `tee` itself. For example: By using `nc` as the command, and specifying `localhost` and a port number (as two separate arguments), you'll see event data being received via `nc` on the specified port.

7.39. Unroll

The Unroll Function accepts an array field - or an expression to evaluate an array field - and breaks (or *unrolls*) the array into individual events.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Source field expression: Field in which to find/calculate the array to unroll. Example: `_raw`, `_raw.split(/\n/)`. Defaults to `_raw`.

Destination field: Field (within the destination event) in which to place the unrolled value. Defaults to `_raw`.

Example

Assume we want to break/unroll each line of this event:

Sample Event

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.5  38000  5356 ?        Ss    2018   2:02 /lib/systemd/systemc
root         2  0.0  0.0      0     0 ?        S     2018   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     2018   1:51 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<    2018   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S     2018   3:55 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S     2018   0:00 [rcu_bh]
```

Copy

Settings

Source field expression: `_raw.split(/\n/)`

The `split()` JavaScript method breaks `_raw` into an ordered set of substrings/values, puts these

values into an array, and returns the array.

Destination field: `_raw`

Resulting Events

```
Event 1:
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND

Event 2:
root         1  0.0  0.5  38000  5356 ?        Ss    2018   2:02 /lib/systemd/systemc

Event 3:
root         2  0.0  0.0      0     0 ?        S     2018   0:00 [kthreadd]

Event 4:
root         3  0.0  0.0      0     0 ?        S     2018   1:51 [ksoftirqd/0]

Event 5:
root         5  0.0  0.0      0     0 ?        S<    2018   0:00 [kworker/0:0H]

Event 6:
root         7  0.0  0.0      0     0 ?        S     2018   3:55 [rcu_sched]

Event 7:
root         8  0.0  0.0      0     0 ?        S     2018   0:00 [rcu_bh]
```

Copy

7.40. XML Unroll

The XML Unroll Function accepts a proper XML event with a set of elements, and converts the elements into individual events.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Unroll elements regex: Path to the array to unroll. Example: `^root\.child\.ElementToUnroll$`

Copy elements regex: Regex matching elements to copy into each unrolled event. Example: `^root\.(childA|childB|childC)$`

Unroll index field: Cribl Stream will add a field with this name, containing the 0-based index at which the element was located within the event. In Splunk, this will be an index-time field. Supports nested addressing. Name defaults to `unroll_idx`.

Pretty print: Whether to pretty print the output XML.

Examples

Assume that the following sample is ingested as a single event:

sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Parent>
    <myID>123456</myID>
    <branchLocation>US</branchLocation>
    <Child>
        <state>NY</state>
        <city>New York</city>
    </Child>
    <Child>
        <state>NJ</state>
        <city>Edgewater</city>
    </Child>
    <Child>
        <state>CA</state>
        <city>Oakland</city>
    </Child>
    <Child>
        <state>CA</state>
        <city>San Francisco</city>
    </Child>
</Parent>
```

Copy

If you insert this sample using **Preview** > **Add a Sample** > **Paste a Sample**, adjust Event Breaker settings to add the sample as a single event. One way to do this is to add a regex Event Breaker that (by design) will not match anything present in the sample. For example: `/[\n\r]+donotbreak(?!\s)/`. In current Cribl Stream versions, you can also use the built-in **Do Not Break** Ruleset.

Set up the XML Unroll Function using these settings:

Unroll elements regex: `^Parent\.Child$` **Copy elements regex**: `^Parent\.(myID|branchLocation)$`

Output 4 Events:

Resulting Events

```
# Event 1
<?xml version="1.0"?>
<Child>
  <myID>123456</myID>
  <branchLocation>US</branchLocation>
  <state>NY</state>
  <city>New York</city>
</Child>

# Event 2
<?xml version="1.0"?>
<Child>
  <myID>123456</myID>
  <branchLocation>US</branchLocation>
  <state>NJ</state>
  <city>Edgewater</city>
</Child>

# Event 3
<?xml version="1.0"?>
<Child>
  <myID>123456</myID>
  <branchLocation>US</branchLocation>
  <state>CA</state>
  <city>Oakland</city>
</Child>

# Event 4
<?xml version="1.0"?>
```

7.41. Prometheus Publisher (Deprecated)

This Function was deprecated as of Cribl Stream 3.0, and has been removed from Cribl Stream as of v.4.0. Please instead use the Prometheus Destination to send metrics to Prometheus-compatible endpoints.

The Prometheus Publisher Function allows for metrics to be published to a Prometheus-compatible metrics endpoint. These can be upstream metrics received by Cribl Stream, or metrics derived from the output of Cribl Stream's `Publish Metrics` or `Aggregation` Functions. A Prometheus instance is responsible for collecting the metrics at that endpoint, and for performing its own processing of the metric data.

In the current Cribl Stream version, the endpoint is: `http://<worker_node_IP>:<api-port>/metrics`. Within Cribl Stream, that endpoint redirects from `http://<worker_node_IP>:9000/metrics` to `http://<worker_node_IP>:9000/api/v1/metrics`.

If used, this Function **must** follow any Publish Metrics or Aggregations Functions within the same Pipeline. This is to ensure that any data **not** originating from a metrics input is transformed into metrics format.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Fields to publish: Wildcard list of fields to publish to the Prometheus endpoint.

Advanced Settings

Batch write interval: How often, in milliseconds, the contents should be published. Defaults to `5000`.

Passthrough mode: Toggle off (default) to override the **Final** setting and suppress output to downstream Functions' Destinations. Toggle on to allow events to flow to consumers beyond the Prometheus endpoint. In effect, when previewing the pipeline output what you'll see is your event fields will have strikethrough font applied to them. This does not mean the Prometheus function is not matching your events but rather indicative of the Passthrough being disabled.

Update mode: Toggle off (default) to supporess output to downstream Functions' Destinations. (This overrides the **Final** setting.) Toggle on to allow events to flow to consumers beyond the Prometheus endpoint.

Example

This example uses the same PANOS sample data as the Publish Metrics Function, and is similarly preceded in a Pipeline by a Parser Function that extracts fields from the PANOS log.

Filter: Set as appropriate. **Fields to publish**: Set as appropriate. We'll use the default of `*` for this example. **Advanced settings**: Accept defaults.

After committing and deploying changes, you should be able to use a `curl` command (-L needed to follow the redirect mentioned above) to verify that metrics are being published, just a few seconds after data is ingested on an idle system.

curl output

```
$ curl -L http://<worker_node_IP>:9000/metrics
# TYPE perf_192_168_1_248_bytes_sent counter
metric_192_168_1_248_bytes_sent {destination_ip="160.177.222.249",inbound_interface='

# TYPE perf_192_168_1_248_bytes_rcvd counter
metric_192_168_1_248_bytes_rcvd {destination_ip="160.177.222.249",inbound_interface='

# TYPE perf_192_168_1_248_pkts_sent counter
metric_192_168_1_248_pkts_sent {destination_ip="160.177.222.249",inbound_interface="e

# TYPE perf_192_168_1_248_pkts_rcvd counter
metric_192_168_1_248_pkts_rcvd {destination_ip="160.177.222.249",inbound_interface="e
```

Copy

Now, we need to have Prometheus scrape the metrics. In this very basic example, you can add the target endpoint to the `prometheus.yml` file, under the `scrape_configs` -> `static_configs` section. Specify the endpoint in `IP:port` syntax, because Prometheus assumes (and requires) `/metrics` for all endpoints.

Restart Prometheus. Within just a few seconds, you should be able to use its query interface to retrieve metrics published by Cribl Stream.

7.42. Reverse DNS (Deprecated)

The Reverse DNS Function resolves hostnames from a numeric IP address, using a reverse DNS lookup.

This Function was deprecated as of v.2.4, and has been removed from Cribl Stream as of v.4.0. Please instead use the DNS Lookup Function's reverse lookup feature.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description of this Function. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Lookup Fields

Lookup field name: Name of the field containing the IP address to look up.

If the field value is not in IPv4 or IPv6 format, the lookup is skipped.

Output field name: Name of the field in which to add the resolved hostname. Leave blank to overwrite the lookup field.

Reload period (minutes): How often to refresh the DNS cache. Use `0` to disable refreshes. Defaults to `60` minutes.

Example

Lookup field name: `dest_ip` **Output field name**: `dest_host` **Result**: See the `dest_ip` field, and the newly created `dest_host` field, in the events.

1
2020-06-29 12:51:03.551 -07:00
α _raw: rec_type=71 rec_type_simple=RNA dest_port=443 snmp_out=0 netflow_src=00000000-0000-0000-0000-000000000000 ssl_server_cert_status="Not Checked" dest_ip=8.8.8.8 sec_intel_event=No mac_address=00:00:0... Show more
_time: 1593460263.551
α app_proto: HTTPS
α client_app: SSL client
α client_version:
α connection_id: 21378
α cribl_breaker: Break on newlines
α cribl_pipe: rev_dns
α dest_autonomous_system: 0

7.43. Trim Timestamp (Deprecated)

The Trim Timestamp Function removes timestamp patterns from events, and (optionally) stores them in a specified field.

This Function is deprecated as of Cribl Stream 4.7, and will be removed in the next major version. Please use the Eval Function instead.

This Function looks for a timestamp pattern that exists between the characters indicated by numeric `timestartpos` and `timeendpos` fields. **From events whose `timestartpos` value is set to `0`**, the Function removes `timestartpos` and `timeendpos` along with the timestamp pattern.

Usage

Filter: Filter expression (JavaScript) that selects data to feed through the Function. Defaults to `true`, meaning it evaluates all events.

Description: Simple description about this step in the Pipeline. Defaults to empty.

Final: Toggle on to stop feeding data to the downstream Functions. Default is toggled off.

Field name: Name of field in which to save the timestamp. (If empty, timestamp will not be saved to a field.)

Example

Remove the timestamp pattern (indicated by `timestartpos` and `timeendpos`) from `_raw`, and stash it in a field called `time_field`.

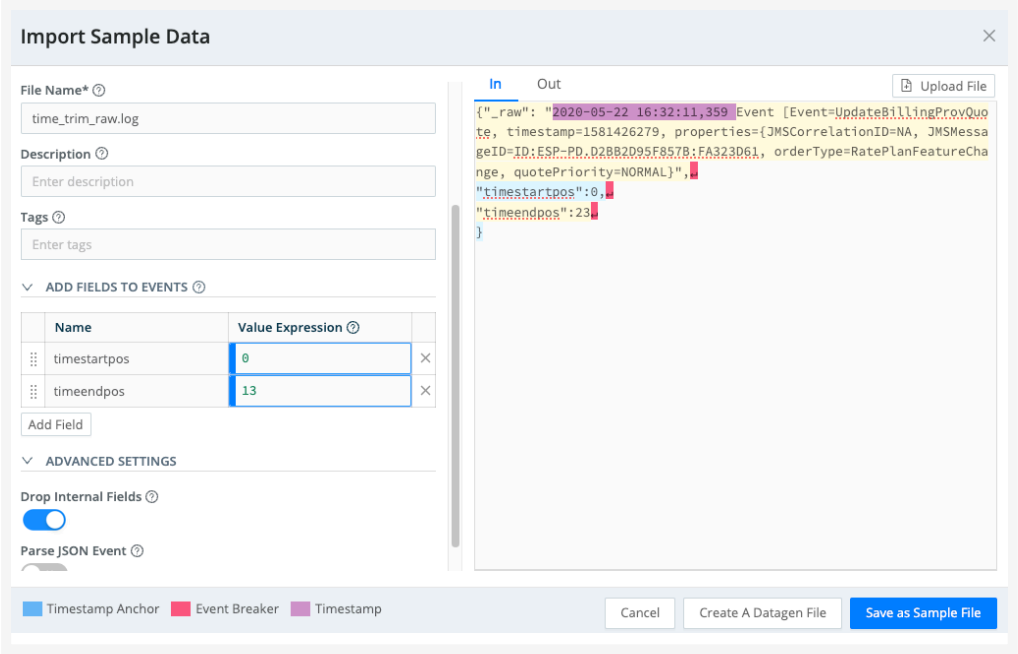
Field name: `time_field`

Example event before:

```
{"_raw": "2020-05-22 16:32:11,359 Event [Event=UpdateBillingProvQuote, timestamp=1581
"timestartpos":0,
"timeendpos":23
}
```

Copy

To create this example payload, we selected **Sample Data** > **Import**, pasted the `_raw` field's original contents into the resulting modal, and then added the two required position fields:



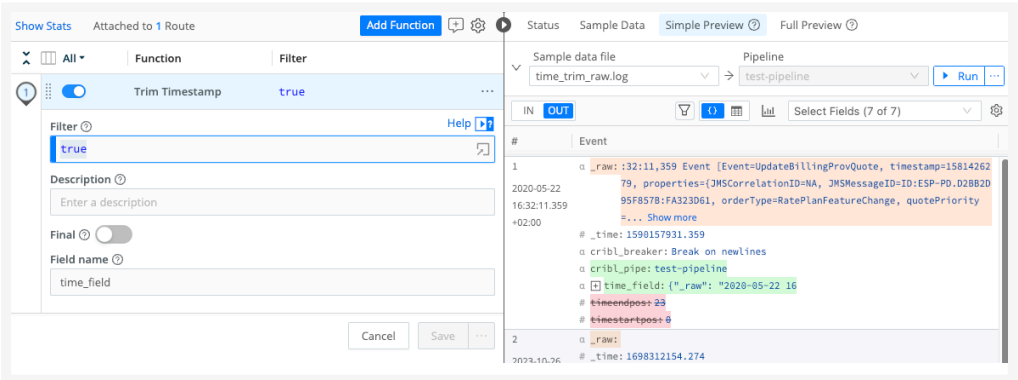
Example event setup

Example event after:

```
{"_raw": "Event [Event=UpdateBillingProvQuote, timestamp=1581426279, properties={JMSC
"time_field":"2020-05-22 16:32:11,359"
}
```

Copy

In the Preview pane's **OUT** view, the original timestamp has been removed from `_raw`, and lifted into the new `time_field` we specified in the Function. The `timestartpos` and `timeendpos` fields have been removed.



Example event, saved and transformed

8. Integrations

Integrations enable seamless data flow between various systems. They encompass both Sources and Destinations, allowing you to get data from multiple origins and deliver processed data to various endpoints. Integrations are the entry and exit points in the event processing order.

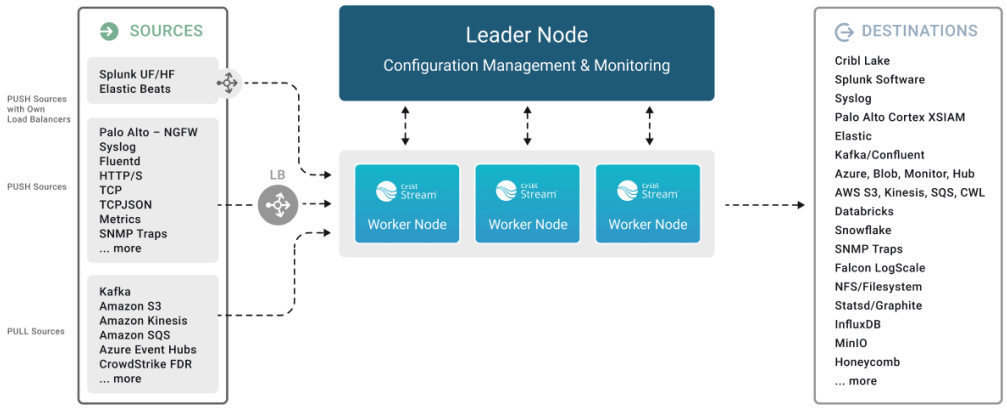
- Sources are configurations that enable Cribl to receive data from various origins. They serve as the entry points in the event processing order, enabling the collection of raw data for subsequent processing.
- Destinations are configurations that define where Cribl sends processed data. They act as the exit points in the event processing order, sending enriched and transformed events to the appropriate systems for storage, analysis, or further processing.

Key Concepts for Optimal Data Flow

- Destination Backpressure Triggers: Understanding the conditions that can cause backpressure in Destinations is crucial for preventing data loss and ensuring smooth data flow, especially during peak load periods.
- Persistent Queues: To safeguard against data loss when integrations are slow or unavailable (backpressure events), persistent queues provide a temporary storage mechanism on disk.
- Load Balancing: Distributing data evenly across multiple Sources and Destinations is essential for maximizing performance and reliability.

8.1. Sources Overview

Cribl Stream can receive continuous data input from various Sources, including Splunk, HTTP, Elastic Beats, Kinesis, Kafka, TCP JSON, and many others. Sources can receive data from either IPv4 or IPv6 addresses.



Push and Pull Sources

Sources Summary

Cribl Stream supports a variety of HTTP-, TCP-, and UDP-based Sources. All HTTP-based Sources are proxyable.

Data Ingestion Overview

A quick conceptual walkthrough of Cribl Stream data ingestion capabilities.

Collector Sources

Collectors ingest data **intermittently** in on-demand bursts ("ad hoc collection"), on preset schedules, or by "replaying" data from local or remote stores.

All Collector Sources are proxyable.

These Sources can ingest data only if a Leader is active:

- Amazon S3 (HTTPS only)
- Azure Blob Storage (HTTPS only)
- Cribl Lake
- Database
- File System/NFS
- Google Cloud Storage (HTTPS only)
- Health Check (HTTP/S)
- REST/API Endpoint (HTTP/S)
- Script (executes any shell script you provide and returns the output of that script/command)
- Splunk Search (HTTP/S)

For background and instructions on using Collectors, see:

- Collectors
- Scheduling and Running
- Job Limits

Check out the example REST Collector configurations in Cribl's Collector Templates repository. For many popular Collectors, the repo provides configurations (with companion Event Breakers, and event samples in some cases) that you can import into your Cribl Stream instance, saving the time you'd have spent building them yourself.

Push Sources

Supported data Sources that **send** to Cribl Stream.

In the absence of an active Leader, these Sources can generally continue data ingestion. However, some Sources may experience functional limitations during prolonged Leader unavailability. See the respective Source documentation for details.

- Amazon Data Firehose (HTTPS only)
- Cloudflare (HTTP/S)
- Datadog Agent (HTTP/S)
- Elasticsearch API (HTTP/S)
- Grafana (HTTP/S)
- HTTP/S (Bulk API)
- Raw HTTP/S
- Loki (HTTP/S)
- Metrics (TCP or UDP)
- Model Driven Telemetry (gPRC)
- NetFlow & IPFIX (NetFlow v5, v9, and IPFIX over UDP)
- OpenTelemetry (OTel) (gRPC or HTTP/S)
- Prometheus Remote Write (HTTP/S)
- SNMP Trap (UDP)
- Splunk HEC (HTTP/S)
- Splunk TCP
- Syslog (TCP or UDP)

- TCP JSON
- TCP (Raw)
- UDP (Raw)
- Windows Event Forwarder (HTTP/S)
- Zscaler Cloud NSS (HTTP/S)

Data from these Sources is normally sent to a set of Cribl Stream Workers through a load balancer. Some Sources, such as Splunk forwarders, have native load-balancing capabilities, so you should point these directly at Cribl Stream.

Comparison of Generic Push Sources

This table compares generic, protocol-level Push Sources. Refer to this table when you need to ingest raw data or custom formats over basic protocols like HTTP, TCP, or UDP.

Source	Key Differentiator	Best For...
HTTP/S (Bulk API)	**Protocol-aware:** Understands and responds using specific API semantics, including acknowledgments.	Receiving data from systems that require custom or generic HTTP/S event ingestion.
Raw HTTP/S	**Protocol-agnostic:** Accepts any HTTP/S payload without validating its structure.	Capturing any arbitrary or custom HTTP/S traffic where only the raw payload matters.
TCP JSON	**Content-aware:** Specifically designed to parse a stream of distinct JSON objects.	Ingesting a continuous stream of well-formed JSON objects over a reliable TCP connection.
TCP (Raw)	**Content-agnostic:** Treats the incoming data as an unstructured stream of bytes.	Ingesting binary data, custom application protocols, or any non-JSON data over TCP.
UDP (Raw)	**Stateless with best-effort delivery:** Offers lightweight, connectionless data ingestion.	High-volume, loss-tolerant data streams like syslog, NetFlow, or SNMP traps.

Pull Sources

Supported Sources that Cribl Stream **fetches** data from.

- Amazon Kinesis Data Streams (HTTPS only) - requires active Leader
- Amazon SQS (HTTPS only)
- Amazon S3 (HTTPS only)
- Amazon Security Lake (HTTPS only)

- Google Cloud Pub/Sub (HTTPS only)
- Azure Event Hubs (TCP)
- Azure Blob Storage (HTTPS only)
- Confluent Cloud (TCP)
- CrowdStrike FDR (HTTPS only)
- Office 365 Services (HTTPS only) - requires active Leader for job management
- Office 365 Activity (HTTPS only) - requires active Leader for job management
- Office 365 Message Trace (HTTPS only) - requires active Leader for job management
- Microsoft Graph (HTTPS only) - requires active Leader for job management
- Prometheus Scraper (HTTP/S) - requires active Leader for job management
- Kafka (TCP)
- Amazon MSK (TCP)
- Splunk Search (HTTP/S) - requires active Leader for job management
- Wiz (HTTPS only) - requires active Leader for job management
- OpenAI - requires active Leader for job management

Pull Sources fetch data from various external systems, with state management varying by Source. Some Sources, like Kafka, manage state independently, while others, such as Office 365 and Kinesis, require an active Leader for job management. This dependency is noted in the Pull Sources list above, indicating which Sources rely on the Leader for state and job management. For details on how each Source interacts with Cribl Stream, refer to the documentation for each Source.

System Sources

Sources supply information generated by Cribl Stream about itself or move data among Workers within your Cribl Stream deployment.

- AppScope (TCP or Unix socket)
- Datagen
- Exec (on-prem only)
- File Monitor (on-prem only)
- Journal files (on-prem only)
- System Metrics (on-prem only)
- System State (on-prem only)

Internal Sources

Similar to System Sources, Internal Sources also supply information generated by Cribl Stream about itself. However, unlike System Sources, they don't count towards your credit usage or ingest quota.

- Cribl HTTP (HTTP/S)
- Cribl TCP
- Cribl Internal

Configure and Manage Sources

For each Source *type*, you can create multiple definitions, depending on your requirements.

From Stream Home, under **Worker Groups**, select a Worker Group to configure. Next, you have two options:

- To configure via QuickConnect:
    1. Select **Routing**, then **QuickConnect** (Stream) or **Collect** (Edge).
    2. Select **Add Source** at left, and select the Source you want from the list, choosing either **Add Existing** or **Add New**.
- To configure via Routing:
    1. Select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge).
    2. Select the Source you want.
    3. Select **Add Source** to open a **New Source** modal.

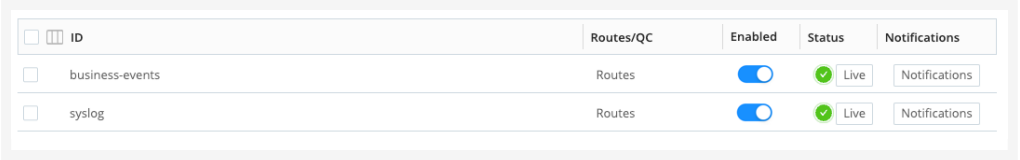
To edit any Source's definition in a JSON text editor, select **Manage as JSON** at the bottom of the **New Source** modal, or on the **Configure** tab when editing an existing Source. You can directly edit multiple values, and you can use the **Import** and **Export** buttons to copy and modify existing Source configurations as `.json` files.

When JSON configuration contains sensitive information, it is redacted during export.

Capture Source Data

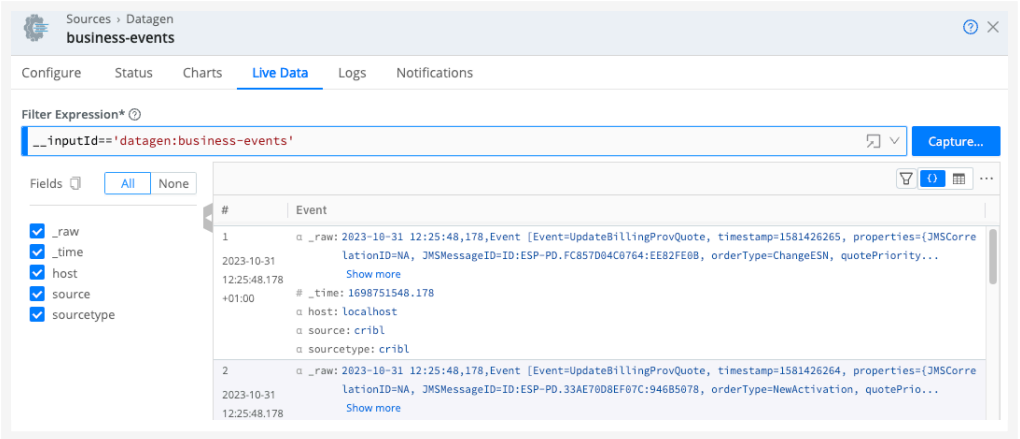
To capture data from a single enabled Source, you can bypass the Data Preview pane, and instead capture directly from the list of Sources. Just select the Source type, and then select the **Live** button beside the Source you want to capture.

To capture live data, you must have Worker Nodes registered to the Worker Group for which you're viewing events. You can view registered Worker Nodes from the **Status** tab in the Source.



Source > Live button

You can also start an immediate capture from within an enabled Source's config modal by selecting the **Live Data** tab.



Source modal > Live Data tab

Monitor Source Status

You can get a quick overview of Source health status by referring to their status icons.

Additionally, each Source's configuration modal offers two tabs for monitoring: Status and Charts.

Source Status Icons

Source status icons are available on the **Data** > **Destinations** page and for each individual Source in the list for a specific Source type.

The icons have the following meanings:

Icon	Meaning
	Healthy. Operating correctly.
	Warning. Experiencing issues. The Source is not functioning fully. Specific conditions will depend on the Source type.

Icon	Meaning
	Critical. Experiencing critical issues. Drill down to the Source's Status tab to find out the details.
	Disabled. The Source is configured, but not enabled.
	No health metrics available. This may mean that a Source is enabled, but has not been deployed yet.
	Inactive. When using GitOps, a Source appears `Inactive` if its **Environment** field (configured under **Advanced Settings**) does not match the currently active environment determined by the deployed Git branch. This ensures integrations only activate in their designated environments, preventing unintended data flow or misconfiguration.

You can also find status icons on the Cribl Stream front page on Worker Group tiles. ⚠ or ❗ on any tile indicate that one or more Sources or Destinations in this Worker Group are experiencing issues. Hover over the icon to see more details.

Status Tab

The **Status** tab provides details about the Workers in the group and their status. An icon shows whether the Worker is operating normally.

You can select each Worker's row to see specific information, for example, to identify issues when the Source displays an error. The specific set of information provided depends on the Source type. The data represents only process 0 for each Worker Node.

The content of the **Status** tab is loaded live when you open it and only displayed when all the data is ready. With a lot of busy Workers in a group, or Workers located far from the Leader, there may be a delay before you see any information.

The statistics presented are reset when the Worker restarts.

Charts Tab

The **Charts** tab presents a visual representation of Source activity using internal metrics. Unlike the status tab, this data is read almost instantly and does not reset when restarting a Worker.

Configuring Time Ranges

At the top of the modal, a time range picker controls which time window the charts display:

- The picker offers the same presets as the main Monitoring time selector (for example, `5 min`, `15 min`, `1 hour`, `6 hours`, `1 day`).
- When you open the Source I/O metrics modal from **Monitoring**, the charts inherit the current Monitoring time range.
- When you open the modal from other areas of the UI (for example from **Data** > **Sources** or directly from a Source configuration modal), the Charts tab defaults to `10 minutes`, matching the previous behavior.
- Changing the time range in the modal only affects that modal and is not persistent. It does not change the global time range on the originating page (for example, the **Monitoring** overview or the page-level time picker).

Use this picker to zoom in on short spikes or expand to longer windows when you're troubleshooting Source performance or throughput issues.

Standard Metrics

The following metrics are available for all deployments:

Data	Metric
Total events in	`total.in_events`
Average throughput (events per second)	Derived from `total.in_events`
Total bytes in	`total.in_bytes`
Average throughput (bytes per second)	Derived from `total.in_bytes`

Integration Metrics

On Cribl.Cloud deployments, the following additional charts are available when the `iometrics` integration metrics are enabled at the appropriate metrics level.

Data	Metric	Level Required
P95 latency	`p95_duration_millis` - 95th percentile request latency in milliseconds.	`Detailed`
P99 latency	`p99_duration_millis` - 99th percentile request latency in milliseconds.	`Detailed`

Data	Metric	Level Required
Failure rate	Percentage of failed requests, calculated as `(failed_requests / total_requests) * 100`.	`Basic`

Logs Tab

Select the **Logs** Tab to access the log viewer. This view lists the internal logs generated by Cribl Stream, which are records of various system events. Use these logs to monitor the health of your Cribl Stream deployment and troubleshoot operational issues.

Each log is structured data containing specific fields about the event that generated it. See Internal Logs for detailed information about the available logs and fields.

You can filter and analyze the list of log events to find specific data:

- **Filter by log type:** Use the main menu to select a type of log to view (such as the API Process, Access, Audit).
- **Search with expressions:** Enter a JavaScript expression in the main search bar to filter logs.
- **Interact with log events:** On the list of logs, you can select an event from an individual field to:
  - Add or exclude that field from the search bar.
  - Copy the event value to your clipboard.

Notifications Tab

Notifications alert Cribl Stream admins about issues that require their immediate attention, such as alerts for Sources such as high, low, or no data volume or Source Persistent Queue Usage. You can view all Notification events on this tab. See Notifications for more information about available Notifications and how to configure them.

Preconfigured Sources

To accelerate your setup, Cribl Stream ships with several common Sources configured for typical listening ports, but not switched on. Open, clone (if desired), modify, and enable any of these preconfigured Sources to get started quickly.

On a Cribl.Cloud deployment, **do not delete** any preconfigured Sources. If you don't plan to use them, keep them disabled.

- Syslog - UDP Port 9514, TCP Port 9514, TCP + TLS Port 6514

- Splunk TCP - Port 9997
- Splunk HEC - Port 8088
- TCP JSON - Port 10070
- TCP - Port 10060
- HTTP - Port 10080
- Elasticsearch API - Port 9200
- SNMP Trap - Port 9162 (preconfigured only on on-prem setups)
- OpenTelemetry - Port 4317 (preconfigured only on Cribl.Cloud)

System and Internal Sources:

- Cribl Internal > CriblLogs (preconfigured only on on-prem setups)
- Cribl Internal > CriblMetrics
- Appscope
- Cribl HTTP - Port 10200 (preconfigured only on distributed setups)
- Cribl TCP - Port 10300 (preconfigured only on distributed setups)
- System Metrics (preconfigured only on on-prem setups)
- System State (preconfigured only on on-prem setups)
- Journal files (preconfigured only on on-prem setups)

In a preconfigured Source's configuration, never change the **Address** field, even though the UI shows an editable field. If you change these fields' value, the Source will not work as expected.

After you create a Source and deploy the changes, it can take a few minutes for the Source to become available in Cribl.Cloud's load balancer. However, Cribl Stream will open the port, and will be able to receive data, immediately.

Single Source Support

The following Source types are limited to preconfigured Sources:

- Cribl Internal
- System Metrics
- System State

Cribl.Cloud Ports and TLS Configurations

Cribl.Cloud provides several data Sources and ports already enabled for you, plus 11 additional TCP ports (`20000-20010`) that you can use to add and configure more Sources.

The Cribl.Cloud portal's **Data Sources** page displays the pre-enabled Sources, their endpoints, the reserved and available ports, and protocol details. For each existing Source listed here, Cribl recommends using the preconfigured endpoint and port to send data into Cribl Stream.

To access the **Data Sources** page, on the top bar, select **Products**, and then select **Cribl** > **Workspace** > **Data Sources**.

Cribl HTTP and Cribl TCP Sources/Destinations

Use the Cribl HTTP Destination and Source, and/or the Cribl TCP Destination and Source, to relay data between Worker Nodes connected to either the same Leader or different Leaders. This traffic does not use Cribl Credits or count against your ingestion quota on the receiving Organization. (For related details, see Transfer Data Between Workspaces or Environments.)

Backpressure Behavior and Persistent Queues

By default, a Cribl Stream Source will respond to a **backpressure** situation by blocking incoming data. Backpresssure triggers exist when an in-memory buffer is full and/or when downstream Destinations/receivers are unavailable. The Source will refuse to accept new data until it can flush its buffer.

This will propagate block signals back to the sender, if it supports backpressure. Note that UDP senders (including SNMP Traps and some syslog senders) do not provide this support. In this situation, Cribl Stream will simply drop new events until the Source can process them.

8.1.1. Collector Sources

Unlike other Cribl Stream Sources, Collectors are designed to ingest data intermittently, rather than continuously. You can use Collectors to dispatch on-demand (ad hoc) collection tasks, which fetch or "replay" (re-ingest) data from local or remote locations.

Collectors also support **scheduled** periodic collection jobs - recurring tasks that can make batch collection of stored data more like continual processing of streaming data. You configure Collectors prior to, and independently from, your configuration of ad hoc versus scheduled collection runs.

Collectors are integral to Cribl Stream's larger story about optimizing your data throughput. Send full-fidelity log and metrics data ("everything") to low-cost storage, and then use Cribl Stream Collectors to selectively route ("replay") only needed data to your systems of analysis.

Collector Resources

- Video introduction to Data Collection, in < 2 minutes.
- Video introduction to Data Collection Scheduling, in < 2 minutes.
- Free, interactive try-out of Collectors in Cribl's Data Collection & Replay sandbox.
- Example Collector configurations - ready to import into Cribl Stream - in Cribl's Collector Templates repository.
- Using Collectors guides: S3 Storage and Replay | REST API Collectors | Microsoft Graph API Collection | ServiceNow API Collection | Creating a Custom Collector.

Collector Types

Cribl Stream currently provides the following Collector options:

- Azure Blob - enables data collection and replay from Azure Blob Storage objects.
- Cribl Lake - enabled data collection and reply from Cribl Lake Datasets.
- Database - enables data collection from database management systems like MySQL and SQL Server.
- File System/NFS - enables data collection and replay from local or remote filesystem locations.
- Google Cloud Storage - enables data collection and replay from Google Cloud Storage buckets.
- Health Check - monitors the availability of system endpoints.
- REST/API Endpoint - enables data collection and replay via REST API calls. Provides four Discover options, to support progressively more complex (and dynamic) item enumerations.
- S3 - enables data collection and replay from Amazon S3 buckets or S3-compatible stores.

- Script - enables data collection and replay via custom scripts.
- Splunk Search - enables data collection and replay from Splunk queries. Supports both simple and complex queries, as well as real-time searches.

If you are exploring Collectors for the first time, the File System/NFS Collector is the simplest to configure, while the REST/API Collector offers the most complex configuration options.

How Do Collectors Work

You can configure a Cribl Stream Node to retrieve data from a remote system by selecting **Manage** from the top nav, then a **Worker Group** to configure. Next, click **Data** > **Sources** > **Collectors**. Data collection is a multi-step process:

1. First, define a Collector instance. In this step, you configure **collector-specific settings** by selecting a Collector type and pointing it at a specific target. For example, the target will be a directory if the type is File System, or an S3 bucket/path if the type is Amazon S3.
2. Next, schedule or manually run the Collector. In this step, you configure either scheduled-job-specific or run-specific settings - such as the run Mode (Preview, Discovery, or Full Run), the Filter expression to match the data against, the time range, and so on.

In a Distributed environment, the Leader Node orchestrates collection jobs. The Leader maintains the job definitions and breaks them down into individual tasks. It then distributes these tasks to registered Worker Nodes for execution. Workers do not pull job definitions, they execute the tasks assigned to them by the Leader.

A collection job is typically made up of one or more tasks that: discover the data to be fetched; fetch data that match the run filter; and finally, pass the results either through the Routes or (optionally) into a specific Pipeline and Destination.

Collector Changes

A key difference in how Collectors work relates to when configuration changes are applied. Unlike many other configurations, certain Collector modifications do not require a **Commit & Deploy** to take effect.

Leader-Scheduled Collectors

For the following Collector types, the job definition is managed entirely on the Leader Node:

- **REST/API Endpoint**
- **Database**
- **Health Check**

Any changes made to the configuration of these Collectors - such as updating a URL, adding a field, or changing a query - are applied on the next scheduled run automatically. No **Commit & Deploy** is needed for the

Collector's own configuration. This is because the Leader always uses its current, saved configuration to generate tasks for Workers at execution time.

For all other Collector types not listed above, changes require a **Commit & Deploy** to be propagated to the Worker Nodes.

Dependencies Require Commit & Deploy

While the job definition for a Leader-scheduled Collector doesn't need to be deployed, any dependencies it relies on do. If your Collector job references Knowledge objects like Event Breakers, Pipelines, or Lookups, those objects must be deployed to the Workers.

If you update a dependent object but fail to deploy it, the collection job will fail with an error, such as "missing event breaker."

Advanced Collector Configuration

You can edit the configuration of an existing Collector. Or, you can create a new Collector from a template. A template is just a Collector configuration file (in JSON, as usual) intended to copied and edited.

Editing an Existing Collector

1. When configuring the Collector, click **Manage as JSON** on the **Configure** tab.
2. Cribl Stream will open a JSON editor.
3. Edit the Collector as desired.
4. (Optional) If you want to make the Collector configuration available locally, click **Export**.

When JSON configuration contains sensitive information, it is redacted during export.

5. Click **OK** to exit the **Manage as JSON** modal.
6. Finish configuring the Collector and click **Save**.

Create a New Collector from a Template

You can create a new Collector from a template like those in Cribl's Collector Templates repository.

For many popular Collectors, the Collector Templates repository provides configurations (with companion Event Breakers, and event samples in some cases) that you can import into your Cribl Stream instance, saving the time you'd have spent building them yourself. For many Collectors, you will need to import both the Collector itself (a `collector.json` file) and its Event Breaker (a `breaker.json` file). See the Event Breakers topic for instructions on how to import them.

Collector configurations can contain placeholders defined in the form `<Label|Description>`, where `Label` is the input field label and `Description` is the tooltip text. The Cribl Stream JSON editor makes it easy to work with placeholders, as you'll see in the following procedure and placeholders example.

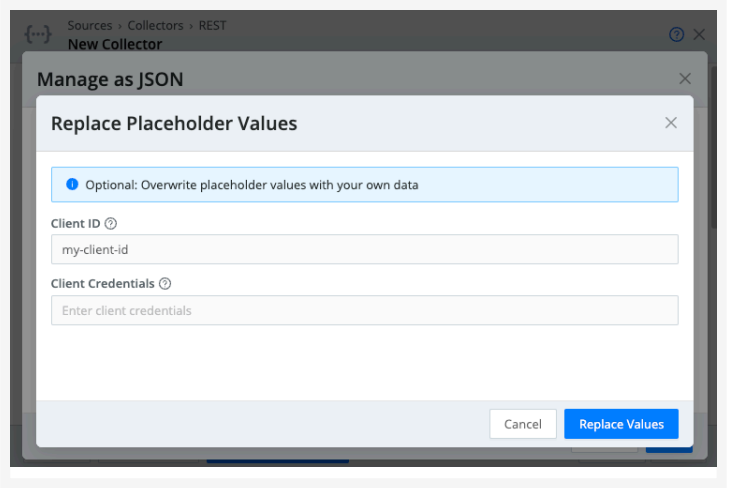
1. Click **Manage as JSON** at the bottom of the **New Collector** modal.
2. Cribl Stream will open a JSON editor.
3. Import the Collector configuration (that is, the template) using either of the two following methods:
    - **Import method**: If your desired Collector configuration file is available locally, click **Import**, navigate to the file, select it, and click **Open**.
    - **Copy and paste method**: If your desired Collector configuration file is open in a local text editor, or is in a Git repository, copy the file. (In the Cribl's Collector Templates repo, navigate to the `collector.json` file in the desired Collector's folder, then click the copy icon.) Back in the JSON text editor, paste the Collector configuration.
4. If the template contains placeholders:
    - If you used the import method, Cribl Stream will open the **Replace Placeholder Values** modal.
    - If you used the copy and paste method, click **OK** to open the modal.
5. Enter desired values for any fields defined as placeholders.
6. (Optional) Edit the configuration further as desired.
7. (Optional) If you want to make the Collector configuration available locally, click **Export**.

When JSON configuration contains sensitive information, it is redacted during export.

8. Click **OK** to exit the **Manage as JSON** modal.
9. Finish configuring the Collector and click **Save**.

Placeholders Example

Here's an example where the configuration defines placeholders for fields named `Client ID` and `Client Credentials`:



Filling in placeholder information

Filling in fields is optional; if you leave a field empty, its placeholder will remain in the JSON configuration, and you can enter a value later.

Scheduled Collection Jobs

You might process data from inherently non-streaming sources, such as REST endpoints, blob stores, and so on. Scheduled jobs enable you to emulate a data stream by scraping data from these sources in batches, on a set interval.

You can schedule a specific job to pick up new data from the source - data that hadn't been picked up in previous invocations of this scheduled job. This essentially transforms a non-streaming data source into a streaming data source.

Collectors in Distributed Deployments

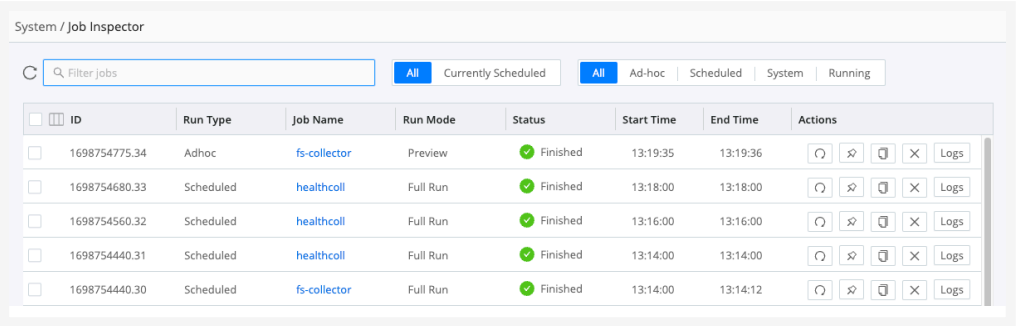
In a Distributed deployment, you configure Collectors at the Worker Group level, and Worker Nodes execute the tasks. However, the Leader Node oversees the task distribution, and tries to maintain a fair balance across jobs.

When Workers ask for tasks, the Leader will normally try to assign the next task from a job that has the least tasks in progress. This is known as "Least-In-Flight Scheduling," and it provides the fairest task distribution for most cases. If desired, you can change this default behavior by opening **Group Settings** > **General Settings** > **Limits** > **Jobs**, and then setting **Job dispatching** to **Round Robin**.

More generally: In a Distributed deployment, you configure Collectors and their jobs on individual Worker Groups. But because the Leader manages Collectors' state, if the Leader instance fails, Collection jobs will fail as well. (This is unlike other Sources, where Worker Groups can continue autonomously receiving incoming data if the Leader goes down.)

Monitor and Inspect Collection Jobs

Select **Monitoring** > **System** > **Job Inspector** to view and manage pending, in-flight, and completed collection jobs and their tasks.



Job Inspector

Here are the options available on the Job Inspector page:

- **All** vs. **Currently Scheduled** tabs: Click **Currently Scheduled** to see jobs forward-scheduled for future execution - including their cron schedule details, last execution, and next scheduled execution. Click **All** to see all jobs initiated in the past, regardless of completion status.
- Job categories (buttons): Select among **Ad-hoc**, **Scheduled**, **System**, and **Running**. (At this level, **Scheduled** means scheduled jobs already running or finished.)
- Group selectors: Select one or more check boxes to display action buttons at the bottom of the table, like **Pause** and **Resume**.
- Sortable headers: Click any column to reverse its sort direction.
- Search bar: Click to filter displayed jobs by arbitrary strings.
- Action buttons: For finished jobs, the icons (from left to right) indicate: **Rerun**; **Keep job artifacts**; **Copy job artifacts**; **Delete job artifacts**; and **Display job logs** in a modal. For running jobs, the options (again from left to right) are: **Pause**; **Stop**; **Copy job artifacts**; **Delete job artifacts**; and **Live** (show collection status in a modal).

Monitor Job Artifacts

Collection jobs create artifacts that include internal accounting data about the job execution process and log files. Because the artifacts have limited value after the job is completed, the system automatically deletes them

8.1.1.1. Azure Blob Storage Collector

Cribl Stream supports collecting data, and replaying specific events, both from Azure Blob Storage, and from Azure Data Lake Storage Gen2, which implements a hierarchical namespace over blob data. This page covers how to configure the Collector.

For configuration examples, see Resources below.

Configure an Azure Blob Storage Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. For example, `azure_42-a`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description**: Optionally, enter a description.
    - **Auto-populate from**: Optionally, select a predefined Destination that will be used to auto-populate Collector settings. Useful when replaying data.
    - **Container name**: Container to collect from. This value can be a constant, or a JavaScript expression that can be evaluated only at init time. For example, referencing a Global Variable: `myBucket-${C.vars.myVar}`.

Container names can include only lowercase letters, numbers, and/or hyphens (`-`). This restriction is imposed by Azure.

3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Use this default option to enter your Azure Storage connection string directly. Exposes a **Connection string** field for this purpose. (If left blank, Cribl Stream will fall back to `env.AZURE_STORAGE_CONNECTION_STRING`.)
    - **Secret**: This option exposes a **Connection string (text secret)** drop-down, in which you can select a stored secret that references an Azure Storage connection string. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need to generate a new secret.
    - **Client secret**: This option allows you to use the Azure service principal's client secret to authenticate, and exposes Azure service principal settings.

- **Certificate**: This option allows you to use a certificate registered with the Azure service principal to authenticate, and exposes Azure service principal settings.

4. Next, you can configure the following **Optional Settings**:
   - **Path**: The directory from which to collect data. Templating is supported (for example, `myDir/${datacenter}/${host}/${app}/`). Time-based tokens are also supported (for example, `myOtherDir/${_time:%Y}/${_time:%m}/${_time:%d}/`). More on templates and Filters.
   - **Path extractors**: Extractors allow using template tokens as context for expressions that enrich discovery results.
   - Select **Add Extractor** to add each extractor as a key-value pair, mapping a **Token** name on the left (of the form `/<path>/${<token>}`) to a custom JavaScript **Extractor expression** on the right (for example, `{host: value.toLowerCase()}`). See an example of the Extractor Expression below.
   - **Recursive**: Toggle on (default) if you want data collection to recurse through subdirectories.
   - **Include metadata**: Toggle on (default) if you want Cribl Stream to include Azure Blob metadata in collected events (at `__collectible.metadata`).
   - **Include tags**: Toggle on (default) if you want Cribl Stream to include Azure Blob tags in collected events (at `__collectible.tags`). To prevent errors, toggle off when using a Shared Access Signature connection string, especially on storage accounts that do not support Azure Blob index tags.
   - **Max batch size (objects)**: Maximum number of metadata objects to batch before recording as results. Defaults to `10`. To override this limit in the Collector's Schedule/Run modal, use Advanced Settings > Upper task bundle size.
   - **Encoding**: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.
7. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

The sections described below are spread across several tabs.

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

> Cribl Stream supports data collection and replay from Azure's **hot** and **cool** access tiers, but not from the **archive** tier - whose stated retrieval lag, up to several hours, cannot guarantee data availability.

Authentication Options

This section provides additional details to configure access to the Azure Blog Storage.

Connection String Format

Manual and **Secret** authentication methods use an Azure Storage connection string in this format: `DefaultEndpointsProtocol=[http|https];AccountName=<your-account-name>;AccountKey=<your-account-key>`

A fictitious example, using Microsoft's recommended HTTPS option, is: `DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=12345678...32`

Service Principal Settings

Client secret and **Certificate** authentication methods expose the following additional settings:

- **Storage account name**: Enter the name of your Azure Storage Account.
- **Tenant ID**: Enter the service principal's tenant ID.
- **Client ID**: Enter the service principal's client ID.
- **Client secret (text secret)** (with **Client secret** method selected): Select a text secret containing the client secret, or create a new one.
- **Certificate** (with **Certificate** method selected): Select the certificate you registered as credentials for your app in the Azure portal, or create a new one.
- **Azure Cloud**: Select the Azure cloud environment to use `Azure Public Cloud` (default) or `Azure China`.
    - `Azure Public Cloud` sets the authority host to `login.microsoftonline.com` and the endpoint suffix to `core.windows.net`.
    - `Azure China` sets the authority host to `login.chinacloudapi.cn` and the endpoint suffix to `core.chinacloudapi.cn`.
- **Endpoint suffix**: (Optional) Overrides the default endpoint suffix set by the **Azure Cloud** setting. Use this only if you need to connect to a custom or non-standard Azure Blob Storage endpoint.

Using Shared Access Signature

You can authenticate the Azure Blob Storage Collector using a shared access signature (SAS) token expressed as a storage account–level connection string. Container-level SAS tokens are not supported.

When you use a SAS-based connection string:

- The SAS must be created at the **storage account** scope and configured for the **Blob** service.
- It must allow at least **List** and **Read** operations on containers and blobs, so the Collector can discover and read objects.
- If you leave **Include tags** enabled in the Collector, the SAS must also permit **blob index** operations. For example, **Read/Write/Filter** of blob index tags on the account. If those permissions are missing, discovery or collection can fail with an authorization error.

During the Discover phase, the Collector validates the target container by calling Azure's `GetContainerProperties` API. If you construct a connection string from a container-scoped SAS, this check fails with a `403 This request is not authorized to perform this operation using this permission.` For additional details, see Common Error. Instead, always use a **storage account–level SAS** when configuring the Collector connection string.

If you are using Azure Premium storage, you can't set blob index permissions. In such a case, in your Collector's setting, enable the **Include metadata** option. This will allow you to use shared access signature, but as a side effect will strip defined tags for downloaded blobs.

Extractor Expression Example

Each expression accesses its corresponding `<token>` through the `value` variable, and evaluates the token to populate event fields. Here is a complete example:

Token	Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`{program: value.split('.')[0]}`	`/var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

- **Enabled**: Defaults to off. Toggle on to enable the custom command.
- **Command**: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).
- **Arguments**: Select **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

- **Event Breaker rulesets**: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.
- **Event Breaker buffer timeout**: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, and so on. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. This keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields**: List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs UI. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

Replay

See these resources that demonstrate how to replay data from object storage. Both are written around Amazon S3-compatible stores, but the general principles apply to Azure blobs as well:

- Data Collection & Replay sandbox: Step-by-step tutorial, in a hosted environment, with all inputs and outputs preconfigured for you. Takes about 30 minutes.
- Using S3 Storage and Replay: Guided walk-through on setting up your own replay.

How the Collector Pulls Data

In the Discover phase, the first available Worker returns the list of files to the Leader Node. In the Collect phase, Cribl Stream spreads the list of files to process spread across 1..N Workers, based on file size, with the goal of distributing tasks as evenly as possible across Workers. These Workers then stream in their assigned files from the remote Azure Blob Storage location.

Azure Permissions

When configuring an Azure Collector in Cribl Stream, you must grant the necessary permissions for it to interact with your Azure environment. Cribl recommends creating an Azure service principal with the minimum required permissions using Azure role-based access control (Azure RBAC). This approach ensures Cribl Stream has a dedicated identity with precisely the access it needs, following the principle of least privilege.

The minimum permissions required for Collectors are:

- **Storage Blob Data Reader**
- **Storage Blob Data Owner**: Required if **Include Tags** is enabled, which is the default.

See Authorization in the official Microsoft documentation for more information.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
  - As mentioned above in **Optional Settings**, this object will include Azure Blob's own metadata at `__collectible.metadata` and Azure Blob tags at `__collectible.tags`, unless you deselect the corresponding UI toggles.
  - `collectorType`: Indicates the type of Collector used for the job.
  - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
  - Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
    - `<timestamp>`: Timestamp when the job was initiated (in Unix time).
    - `<randomId>`: A random identifier to ensure uniqueness.
    - `adhoc`: Indicates the job was manually triggered.
    - `<Collector ID>`: The ID of the Collector.
  - Scheduled jobs are formatted as `collection:<Collector ID>`.
    - `<Collector ID>`: The ID of the Collector.

Resources

For examples of configuring Cribl Stream to interoperate with Azure services, see these guides:

- SSO with Microsoft Entra ID (Cribl.Cloud).
- SSO with Microsoft Entra ID (On-Prem).

Also see these Packs on the Cribl Packs Dispensary, which provide processing Pipelines to transform and optimize incoming Azure data. You can directly import these Pipelines and adapt them to your needs:

- Microsoft Azure NSG Source to Common Schema.
- Azure NSG Flow Logs to OCSF.
- Azure Audit Logs to OCSF.
- Microsoft Office Activity & Azure Logs.

Troubleshooting

When permissions are correct on the object store, and events are reaching the Collector, the Preview pane will show events and the Job Inspector will show an **Events collected** count.

However, if previewing returns no events and throws no error, first check your Filter expression by previewing without it (for example, simplify the Filter expression to `true`). Then check the Job Inspector: If the **Total size** is greater than `0`, and the **Received size** is `NA` or `0`, make sure you have list and read permissions on the object store.

Common Error

The error `This request is not authorized to perform this operation using this permission.` can occur for either of the following reasons:

- **Container-scoped SAS used as a connection string**:
  If you build the Collector connection string from a container-level SAS, the Discover phase's `GetContainerProperties` check fails with this error because it requires account-level authorization. In this case, configure the Collector to use a storage account–level SAS connection string instead. For details, see Using Shared Access Signature.
- **Insufficient permissions for Azure Blob index tags**:
  If your SAS or principal lacks permissions to access blob index tags, you can see the same error when **Include tags** is enabled:
  - If your storage account is Premium Storage, disable **Include tags** in the Azure Blob Storage Collector **Optional Settings** because Premium Storage does not support Azure Blob index tags.
  - For other storage account types, ensure **Allow blob index permissions** is enabled for your Azure Blob Storage account to support the **Include tags** functionality. If the **Include tags** option is

enabled, the Azure app will need `Owner Access`. If tags are not needed, you can disable the **Include tags** option to avoid requiring `Owner Access`.

8.1.1.2. Cribl Lake Collector

The Cribl Lake Collector gathers data from Cribl Lake.

Requirements

Cribl Lake Collector is available only in Cribl.Cloud.

Cribl Lake Collector can gather data from both Cribl-managed Cloud and customer-managed hybrid Stream Worker Groups. Hybrid Worker Groups must be running version 4.8 or higher.

Configure a Cribl Lake Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. For example: `myLakeCollector`.
    - **Lake dataset**: Lake Dataset to collect data from.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
3. Optionally, configure any Result and Advanced settings outlined in the below sections.
4. Select **Save**, then **Commit & Deploy**.

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

Event Breaker rulesets: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, or `GB`. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. This keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Time to live: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

Remove Discover fields: List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).

Resume job on boot: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

Verify Data Flow

To verify that the Collector actually collects data, you can start a single run in the Preview mode.

1. Select your Collector's **Run** action.
2. Make sure mode **Preview** is selected and accept other default settings.
3. Confirm with **Run**.
4. Look at the preview screen to check that data is being collected from Cribl Lake.

8.1.1.3. Database Collector

You can configure Database Collectors to pull events from:

- MySQL
- Oracle
- Postgres
- SQL Server database management systems (DBMSs)

The Database Collector allows you to combine structured data from these sources with unstructured machine data. You can then route combined data to downstream analysis systems to gain new insights.

Cribl Stream supports Oracle database version 12.1 and newer. Queries can retrieve up to 1 GB of large objects (CLOBS, LOBS, and NCLOBS) per field and use the `node-oracledb` library in Thin mode.

How the Collector Pulls Data

Like other Cribl Stream Collectors, a Database Collector can perform Preview, Discovery, and Full Run operations in ad hoc or scheduled runs.

A Database Collector runs a single `collect` task on a single Worker Process and executes a SQL query from it. However, if the Database Collector is scheduled to run while the first `collect` task is still in progress, it will start a simultaneous `collect` task on a different Worker Process. This also happens when you schedule a Database Collector run in Preview mode.

Prerequisites

Before you can configure a Database Collector, there are some required prerequisite steps to connect Cribl Stream with your target database. The overall workflow for setting up a new Database Collector has these phases:

1. Create a Connection String or Configuration Object: Before you start, you need to gather access and authentication information for your database. Using that information, build a connection string or configuration object that you will use to establish a valid connection to the database in the next phase.
2. Configure a Database Connection: Database Collectors rely on Cribl Stream Database Connections, which are reusable objects that you can re-use with multiple Database Collectors to authenticate to the same database.
3. Test the Connection: Ensure the Database Connection is configured correctly and that data is flowing.

4. Configure a Database Collector: Configure a Database Collector to input data from an upstream database. This data can then be routed to a Pipeline for further data processing. The rest of this page explains how to configure the Collector.

Configure a Database Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. For example, `db2GetStuff`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description**: Optionally, enter a description.
    - **Connection**: Select **Create** to configure a new Database Connection or use the drop-down to select an existing Database Connection.
    - **SQL query**: Enter a JavaScript expression, enclosed in backticks, that resolves to a query string for selecting data from the database. For details, see SQL Query Example.
3. Next, you can configure the following **Optional Settings**:
    - **Validate query**: Enforces a basic validation, allowing only a single, self-contained `SELECT` statement. Disable for more complex queries or when using semicolons. Defaults to toggled on.

      Disabling query validation allows potentially destructive queries to be executed, such as data definition language (DDL) and data manipulation language (DML) statements. Such queries can be harmful to your database. Back up your database before running such a query, and proceed with caution.

    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.
6. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

The sections described below are spread across several tabs. Click the tab links at left to navigate among tabs.

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

- **Event Breaker rulesets**: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.
- **Event Breaker buffer timeout**: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Optionally, use the drop-down to select an existing Pipeline to process results before sending them to Routes.

This final field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, and so forth. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when you're configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. This keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields** : List of fields to remove from the Discovery results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

SQL Query Example

Supports only a single statement, and only `SELECT`. Has access to the special `${earliest}` and `${latest}` variables, which will resolve to the Collector run's start and end time. - For Oracle, do not include a semicolon `;` at the end of the **SQL query**. - The Database Collector's `earliest` and `latest` variables are formatted using ISO 8601 conventions, unlike other Collectors' corresponding variables.

Here's an example:

```
`SELECT * FROM table_name WHERE timestamp_field > ${earliest} AND timestamp_field < $
```

Copy

You can also craft a query that says, in effect, "give me everything new since my last collection job." To do this, you specify a **tracking column** in the query. The tracking column (also known as a **rising column**) must be one whose values you know always increase from one record to the next. See the explanation and example below.

Working with a Tracking Column

You can enable state tracking under the `State Tracking` section when scheduling or performing a full run of the Collector. (Preview and Discovery runs do not support state tracking.) With state tracking enabled, the following field will appear:

Tracking Column: A numeric column whose values increase monotonically. The Collector will track the column's greatest value seen in a previous run.

You can configure the Collector to use a **tracking column** (also known as a **rising column**). This prevents overlap between jobs, which occurs when the Collector fetches the same record more than once. It likewise prevents the occurrence of gaps between data collected by consecutive jobs. (If your database already has mechanisms that protect against overlap and gaps, you probably don't need to use a tracking column.)

For your tracking column, you can choose any column whose values always increase from one record to the next. (A sequence of values that behaves like this is said to increase monotonically.) Many database tables have monotonically increasing `id`, `time`, and/or `date` columns.

State tracking is available only for scheduled and full ad-hoc collection (but not for preview or discovery ad-hoc runs). Once you configure your Collector, you can enable state tracking and specify a tracking column while scheduling a job or performing a full ad-hoc run.

To explain the syntax of queries that work with state tracking, we'll break down a few examples.

Tracking Column Queries: A Basic Example

```
`SELECT * FROM my_table WHERE ${state && state.id ? `id > ${state.id.value}` : '1 = 1
```

Copy

The `SELECT` statement must satisfy one requirement: that the tracking column be among the columns named. `SELECT * FROM my_table` satisfies this requirement because the wildcard (`*`) means "all columns."

The `WHERE` clause illustrates the key techniques required for state tracking:

- First, we verify that state has been saved (on a prior run of the Collector); and, if it has, that it includes the desired tracking column – for this example, a column named `id`.

```
WHERE ${state && state.id
```

Copy

- If state **has** been saved, we'll apply the test right after the `?` in the ternary operator. This determines whether the value of the `id` column for a given row is greater than the last-saved value for that column. If it is not, that means that the row was collected already, and should not be collected again.

```
? `id > ${state.id.value}` : '1 = 1'}`
```

Copy

- If state has **not** been saved, this means that the Collector is now about to run **for the first time**.

Calculated temporary columns cannot be used directly in the `WHERE` clause. The `WHERE` clause can only reference existing columns or aliases of existing columns.

To collect all rows on the first run, the query needs an expression that always evaluates to true. The example uses `'1 = 1'`; however, this specific syntax is not compatible with all databases. You should confirm this works or replace it with a direct numerical comparison similar to `1=1` that is universally understood.

Finally, the `ORDER BY` clause, `ORDER BY id`, guarantees that the query returns rows such that the value of the tracking column (here, `id`) increases monotonically from row to row. For state tracking to work correctly, rows **must** be returned such that the value of the tracking column increases monotonically. Some databases have mechanisms that achieve this without an `ORDER BY` clause, but one way or another, this requirement must be satisfied.

Tracking Column Queries: Starting in the Middle

The previous example assumes that you want the Collector's first run to start with the first row of your table. Sometimes, though, you might need the Collector to start somewhere in the middle. If so, you'll need to replace the expression `'1 = 1'` with an expression that finds the desired starting point.

For example:

- If the tracking column is `id` and you want the Collector to begin with the record whose `id` is `4200`, you'd write `id > 4199` instead of `'1 = 1'`.
- Now, the first Collector run will start where `id`'s value is `4200`, and its saved state will include the location where the Collector leaves off.
- In all subsequent runs, the initial test of `state && state.id` will evaluate to true – because there is saved state.
- This means that the Collector will apply `id > ${state.id.value}` and start each new run with next not-yet-collected record.

Tracking Column Queries: Advanced Example

Timestamp columns are a popular choice for state tracking. Here's an example:

```
SELECT id, text_data, timestamp, UNIX_TIMESTAMP(timestamp) as tracking_timestamp FROM
WHERE ${state && state.tracking_timestamp ? `UNIX_TIMESTAMP(timestamp) > ${state.trac
ORDER BY tracking_timestamp
```

Copy

This query differs from the basic example above in one important way: We use the SQL function `UNIX_TIMESTAMP()` to convert the value in the database's `timestamp` column, from a date to an integer. (This example assumes that we're dealing with a MySQL database, where the `timestamp` column's value will be a date.)

Cribl Stream requires the tracking column's value to be an integer; and here, we satisfy that requirement by using the `UNIX_TIMESTAMP()` function. In your own queries, depending on the nature of the columns you want to use for tracking, you might be able to adapt this syntax - or you might need to get more creative.

Working with Stateful Columns

You can configure a Database Collector to process tables only when new rows have been added. Two configurations are possible - one using stateful sequence columns and one using stateful timestamp columns.

In either case, using a stateful column is a four-step process:

1. Set up a database if you do not already have one.
2. Create a database connection in Cribl Stream.
3. Create a Database Collector.
4. Configure the Database Collector to track the sequence or timestamp.

The first time your Collector runs, it will process all rows in the relevant table. On subsequent runs, it will process only rows added since the last run.

Creating a Database Connection in Cribl Stream

To create a database connection:

1. Select **Processing** > **Knowledge** > **Database Connections**.
2. Click **Add Database Connection**.
3. Give the connection an ID of your choosing.
4. Select the **Database type** appropriate for your application.
5. Enter a connection string that is appropriate for your database. The following example is for a SQL Server database:

```
Server=<server-name>; Database=<database>;User Id=<user>; Password=
<password>;trustServerCertificate=true;
```

6. Click **Test Connection** to verify the connection.
7. Click **Save**.
8. Commit and deploy your changes.

Stateful Column Queries: Sequence Example

To set up a Database Collector to use stateful sequence columns:

1. Create a Collector.
2. Give the connection an ID of your choosing.
3. Select the database connection you created previously.
4. For **SQL Query**, input the following string, including the backticks:

```
`SELECT * FROM my_table WHERE ${state && state.id ? `id > ${state.id.value}` : '1
```

Copy

1. Click **Save**. The Collector modal will close.
2. Commit and deploy your changes.
3. Reopen your Collector and click **Schedule**.

In the scheduling modal:

1. Enable scheduling for the Collector.
2. Under **State Tracking**, toggle **Enabled** on and provide the name of the column storing your auto-incrementing ID in the **Tracking Column** field.
3. Click **Save**.

The Collector will process all rows in the table when you run it for the first time. On subsequent runs, it will process only rows added since the last run.

Stateful Column Queries: Timestamp Example

To set up a Database Collector to use stateful timestamp columns:

1. Create a Collector.
2. Give the connection an ID of your choosing.
3. Select the database connection you created previously.

4. For **SQL Query**, input a string like the following. Include the backticks, and use the table and column names from your own database. In this example, the `DATEDIFF` function is specific to a SQL Server database; the function for your database may differ.
   ```
   `select <id>, <text_data>, <timestamp>, DATEDIFF(s, '1970-01-01 00:00:00', timestamp) as tracking_timestamp from <table-name> where ${state && state.tracking_timestamp ? `DATEDIFF(s, '1970-01-01 00:00:00', timestamp) > ${state.tracking_timestamp.value}` : "1=1"} order by tracking_timestamp`
   ```
5. Click **Save**. The Collector modal will close.
6. Commit and deploy your changes.
7. Reopen your Collector and click **Schedule**.

In the scheduling modal:

1. Enable scheduling for the Collector.
2. Under **State Tracking**, toggle **Enabled** on and provide the name of the column that contains the timestamp in the **Tracking Column** field.
3. Click **Save**.

The Collector will process all rows in the table when you run it for the first time. On subsequent runs, it will process only rows added since the last run.

> In this example the SQL Server function `DATEDIFF(s, '1970-01-01 00:00:00', <timestampColumn>)` must be used to convert the database date value to a numeric Epic (seconds since Epoch) value. This conversion is needed because most databases return a formatted timestamp string `mm/dd/yyyy hh:mm:ss (2023-07-11 14:24:18)`, which must be converted to a number for state tracking. In this case, Cribl Stream returns the raw timestamp column as well as the timestamp converted to a number, which is required for state tracking.
>
> Use a Function like Eval to remove `tracking_timestamp` from events if you don't want to send this field to a Destination.

Troubleshoot State and Tracking Column Errors

When you encounter an error like `Tracking column <name> contained non-number/non-string value` the issue is usually related to data type, data integrity, or case-sensitivity in your query configuration.

Follow these steps to diagnose and resolve the issue:

1. **Increase Log Level to Debug:** Set the Collector log level to `debug`. This will display a log message starting with `Query` for every run, showing the user the full SQL statement that is executed against the database.

2. **Check Data Integrity (NULLs):** Verify that the underlying database column used to derive the tracking value does not contain any `NULL` values. Add the clause `WHERE <column-name> IS NOT NULL` to your Collector query to filter out records that would cause the Collector to fail.
3. **Verify State Tracking Column Type:** Ensure the data returned for the final tracking column is a standardized numeric or string format the Collector can easily process, such as a timestamp in Unix time or a simple integer. Confirm your query successfully casts the original column (like a date/time field) into a standardized format before assigning it to the tracking alias.
4. **Validate Query Alias Consistency:** Ensure the tracking column alias used in your `SELECT` statement is the exact same name referenced in the Collector's **State Tracking** configuration. For example, If your query uses `AS TRACKING_DATETIME`, the configuration field must also be `TRACKING_DATETIME` (matching case exactly).
5. **Review state Variable Case-Sensitivity:** If you are using JavaScript to construct the incremental query logic, verify the case of the state variable matches the tracking column name. Ensure the state logic uses the correct variable case.

Manage Collector State

See the Manage Collector State topic for instructions on editing or deleting Collector state.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
    - `collectorType`: Indicates the type of Collector used for the job.
    - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
    - Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
        - `<timestamp>`: The timestamp in Unix time when the job was initiated.
        - `<randomId>`: A random identifier to ensure uniqueness.
        - `adhoc`: Indicates the job was manually triggered.
        - `<Collector ID>`: The ID of the Collector.

- Scheduled jobs are formatted as `collection:<Collector ID>`.

8.1.1.4. File System/NFS Collector

Cribl Stream supports collecting data from a locally mounted filesystem location that is available on all Worker Nodes.

In Cribl.Cloud, the File System/NFS Collector is only available on customer-managed hybrid Worker Nodes.

Configure a File System Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. E.g., `DysonV11Roomba960`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description**: Optionally, enter a description.
    - **Auto-populate from**: Select a Destination with which to auto-populate Collector settings. Useful when replaying data.
    - **Directory**: The directory from which to collect data. Templating is supported (e.g., `/myDir/${host}/${year}/${month}/`). You can also use templating to specify (e.g.) a Splunk bucket from which to collect. Symlinks will not be followed. More on templates and Filters.
3. Next, you can configure the following **Optional Settings**:
    - **Path extractors**: Extractors allow using template tokens as context for expressions that enrich discovery results.
    - Select **Add Extractor** to add each extractor as a key-value pair, mapping a **Token** name on the left (of the form `/<path>/${<token>}`) to a custom JavaScript **Extractor expression** on the right (e.g., `{host: value.toLowerCase()}`). See an example of the Extractor Expression below.
    - **Recursive**: If toggled on (default), data collection will recurse through subdirectories.
    - **Batch size limit (files)**: Maximum number of lines written to the discovery results files each time. Defaults to `10`. To override this limit in the Collector's Schedule/Run modal, use Advanced Settings > Upper task bundle size.
    - **Destructive**: If toggled on, the Collector will delete files after collection. Default is toggled off.
    - **Encoding**: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.

- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.
6. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

Cribl Stream automatically detects gzip compression where a file name ends in `.gz`.

The sections described below are spread across several tabs. Click the tab links at left to navigate among tabs.

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

Extractor Example

Each expression accesses its corresponding `<token>` through the `value` variable, and evaluates the token to populate event fields. Here is a complete example:

Token	Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`{program: value.split('.')[0]}`	`/var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

Enabled: Defaults is toggled off. Toggle on to enable the custom command.

Command: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).

Arguments: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

Event Breaker rulesets: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, etc. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. This keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields** : List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

How the Collector Pulls Data

When you run a Filesystem/NFS Collector in Discovery mode, the first available Worker returns the list of available files to the Leader Node.

In Full Run mode, the Leader distributes the list of files to process across 1..*N* Workers as evenly as possible, based on file size. These Workers then stream in their assigned files from the filesystem location.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
  - `collectorType`: Indicates the type of Collector used for the job.
  - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:

- Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
  - `<timestamp>`: Timestamp when the job was initiated (in Unix time).
  - `<randomId>`: A random identifier to ensure uniqueness.
  - `adhoc`: Indicates the job was manually triggered.
  - `<Collector ID>`: The ID of the Collector.
- Scheduled jobs are formatted as `collection:<Collector ID>`.
  - `<Collector ID>`: The ID of the Collector.

8.1.1.5. Google Cloud Storage Collector

Cribl Stream supports collecting data objects from Google Cloud Storage buckets. This page covers how to configure the Collector.

Google Cloud Roles and Permissions

Your Google Cloud service account requires, at a minimum, the following specific permissions to collect data:

- `storage.buckets.get`
- `storage.objects.get`
- `storage.objects.list`

Assign the following role combinations to grant the required permissions:

- `roles/storage.legacyObjectReader` **plus** `roles/storage.legacyBucketReader`
- `roles/storage.objectViewer` **plus** `roles/storage.insightsCollectorService`

For additional details, see the Google Cloud Access Control topic.

Configure a Google Cloud Storage Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. E.g., `gcs_24-7`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description**: Optionally, enter a description.
    - **Auto-populate from**: Optionally, select a predefined Destination that will be used to auto-populate Collector settings. Useful when replaying data.
    - **Bucket name**: Google Cloud Storage bucket to collect from. This value can be a constant, or a JavaScript expression that can be evaluated only at init time. E.g., referencing a Global Variable: `myBucket-${C.vars.myVar}`.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Auto**: This option uses the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to provide the location of a credential JSON file. When this option is set, Cribl Stream will first check whether the `GOOGLE_APPLICATION_CREDENTIALS` environment variable exists and properly set with valid

credentials. If this environmental variable is missing or incorrect, Cribl Stream checks for credentials in two additional fallback locations, as described in the official Google Cloud documentation: How Application Default Credentials works.

In addition to using the `GOOGLE_APPLICATION_CREDENTIALS` environment variable, you can use credentials attached to your Google Cloud instance.

- **Secret**: This option exposes a **Connection string (text secret)** drop-down, in which you can select a stored secret that references an Azure Storage connection string. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need to generate a new secret.
- **Manual**: Use this default option to enter your Google Cloud Storage connection string directly. Exposes a **Service account credentials** field for this purpose. The **Service account credentials** are the contents of a Google Cloud service account credentials (JSON keys) file. To upload a file, click the upload button at this field's upper right.
- **Secret**: This option exposes a **Service account credentials (text secret)** drop-down, in which you can select a stored secret that references an Google Cloud Storage connection string. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need to generate a new secret.

You can access service account credentials in the Google Cloud Console under **Service Accounts** > <service account associated with bucket> > **Keys**. The key file must be in JSON format.

4. Next, you can configure the following **Optional Settings**:
   - **Path**: The directory from which to collect data. Templating is supported (e.g., `myDir/${datacenter}/${host}/${app}/`). Time-based tokens are also supported (e.g., `myOtherDir/${_time:%Y}/${_time:%m}/${_time:%d}/`). More on templates and Filters.
   - **Path extractors**: Extractors allow using template tokens as context for expressions that enrich discovery results.
   - Select **Add Extractor** to add each extractor as a key-value pair, mapping a **Token** name on the left (of the form `/<path>/${<token>}`) to a custom JavaScript **Extractor expression** on the right (e.g., `{host: value.toLowerCase()}`). See an example of the Extractor Expression below.
   - **Endpoint**: Google Cloud Storage service endpoint. If empty, the endpoint will be automatically constructed using the service account credentials.
   - **Disable time filter**: Toggle on to bypass the collection job event timestamp filtering. By default, Collectors apply time filtering based on the configured time range. Events with timestamps outside this range are discarded. Enabling this setting disables timestamp-based filtering, allowing all returned events to be processed regardless of their timestamps. This is useful when the data source's timestamps do not align with the configured time range, causing unexpected filtering, or when you need to process all data regardless of their timestamps. If you encounter zero results

despite the data source returning events, disabling the time filter can help isolate whether timestamp filtering is the issue. For details, see No Discover or Collector Event Results.

- **Recursive**: If toggled on (the default), data collection will recurse through subdirectories.
- **Max batch size (objects)**: Maximum number of metadata objects to batch before recording as results. Defaults to `10`. To override this limit in the Collector's Schedule/Run modal, use Advanced Settings > Upper task bundle size.
- **Encoding**: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.
7. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

The sections described below are spread across several tabs. Click the tab links at left to navigate among tabs.

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

Extractor Expression Example

Each expression accesses its corresponding `<token>` through the `value` variable, and evaluates the token to populate event fields. Here is a complete example:

Token	Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`{program: value.split('.')[0]}`	`/var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

- **Enabled**: Defaults to toggled off. Toggle on to enable the custom command.
- **Command**: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).
- **Arguments**: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

- **Event Breaker rulesets**: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.
- **Event Breaker buffer timeout**: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, etc. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. This keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional, and available only when **Send to Routes** is toggled on.

Throttling: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, etc. (Example: `42 MB`.) Default value of `0` indicates no throttling.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields** : List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

Replay

See these resources that demonstrate how to replay data from object storage. Both are written around Amazon S3-compatible stores, but the general principles apply to Google Cloud buckets as well:

- Data Collection & Replay sandbox: Step-by-step tutorial, in a hosted environment, with all inputs and outputs preconfigured for you. Takes about 30 minutes.
- Using S3 Storage and Replay: Guided walk-through on setting up your own replay.

How the Collector Pulls Data

In the Discover phase, the first available Worker returns the list of files to the Leader Node. In the Collect phase, Cribl Stream spreads the list of files to process spread across 1..N Workers, based on file size, with the goal of distributing tasks as evenly as possible across Workers. These Workers then stream in their assigned files from the remote Google Cloud Storage location.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
  - `collectorType`: Indicates the type of Collector used for the job.
  - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
  - Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
    - `<timestamp>`: Timestamp when the job was initiated (in Unix time).
    - `<randomId>`: A random identifier to ensure uniqueness.
    - `adhoc`: Indicates the job was manually triggered.
    - `<Collector ID>`: The ID of the Collector.
  - Scheduled jobs are formatted as `collection:<Collector ID>`.
    - `<Collector ID>`: The ID of the Collector.

Troubleshooting

When permissions are correct on the object store, and events are reaching the Collector, the Preview pane will show events and the Job Inspector will show an **Events collected** count.

However, if previewing returns no events and throws no error, first check your Filter expression by previewing without it (e.g., simplify the Filter expression to `true`). Then check the Job Inspector: If the **Total size** is greater than `0`, and the **Received size** is `NA` or `0`, make sure you have list and read permissions on the object store.

8.1.1.6. Health Check Collector

The Health Check Collector monitors endpoints by sending requests and generating events based on the responses. This Collector provides multiple Discover types and Health Check options.

Configure a Health Check Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. For example, `healthMonitoring`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description**: Optionally, enter a description.
3. Within the **Discover** accordion, the **Discover type** drop-down provides four options, corresponding to different use cases. Each **Discover type** selection will expose a different set of fields. Below, we cover the **Discover type**s from the simplest to the most complex.
    - Discover type: None matches cases where one simple API call will retrieve all the data you need. This default option suppresses the Discover stage. Use the modal's **Health check** section to specify the endpoint and other details. Cribl Stream will assign the Health Check task to a single Worker.
    - Discover type: Item List matches cases where you want to enumerate a known list of **Discover items** to retrieve, like specific AWS services to identify hosts to check. Discovery will return one Health Check task per item in the list, and Cribl Stream will spread each of those tasks across all available Workers.
    - Discover type: JSON Response provides a **Discover result** field where you can (optionally) define Discover tasks as a JSON array of objects. Each entry returned by Discover will generate a Health Check task, and Cribl Stream will spread each of those tasks across all available Workers.
    - Discover type: HTTP Request matches cases where you need to dynamically discover what you can Health Check from a REST endpoint. This Discover type most fully exploits Cribl Stream's Discover-and-then-health-check architecture. (Example: Make a REST call to get a list of available hosts, then run Health Checks against each of those hosts.) Each item returned will generate a Health Check task, and Cribl Stream will spread each of those tasks across all available Workers.
4. In the **Authentication** accordion, use the **Authentication** drop-down to select one of these options:
    - **None**: Don't use authentication. Compatible with REST servers like AWS, where you embed a secret directly in the request URL.
    - **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.

- **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret.
- **Login**: Enables you to specify several credentials, then perform a POST to an endpoint during the Discover operation. The POST response returns a token, which Cribl Stream uses for later Health Check operations. Exposes multiple extra fields - see Login Authentication below.
- **Login (credentials secret)**: Like **Login**, except that you specify a secret referencing the credentials, rather than the credentials themselves. Exposes multiple extra fields - see Login Secret Authentication below.
- **OAuth**: Enables you to directly enter credentials for authentication via the OAuth protocol. Exposes multiple extra fields - see OAuth Authentication below.
- **OAuth (text secret)**: Like the **OAuth** option, except that you specify a stored secret referencing the credentials. Exposes multiple extra fields - see OAuth Secret Authentication below.

5. In the **Health Check** accordion, configure the following:
   - **Health check URL**: URL (constant or JavaScript expression) to use for the Health Check operation. Refer to Health Checks below for details.
   - **Health check method**: Select the HTTP verb to use for the Health Check operation - `GET`, `POST`, or `POST with body`.
   - **Health check POST body**: Template for POST body to send with the Health Check request. (This field is displayed only when you set the **Health check method** to `POST with body`.) You can reference parameters from the Discover response using template params of the form: `` `${variable}` ``.
   - **Health check parameters**: Optional HTTP request parameters to append to the request URL. These refine or narrow the request. For details, see Health Check Parameters.
   - **Health check headers**: Optional HTTP request headers to append to the request URL. For details, see Health Check Headers.
   - **Authenticate health check**: Toggle on if you want to authenticate Health Checks. Toggle off (default) to apply **Authentication** only to Discover.

   By adding the appropriate **Health check headers**, you can specify authentication based on API keys, as an alternative to the **Authentication**: `Basic` or `Login` options below.

6. In the **Retries** accordion, configure the following:
   - **Retry type**: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.
   - **Initial retry interval (ms)**: Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.

- **Retry limit**: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.
- **Backoff multiplier**: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, etc.
- **Retry HTTP codes**: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.
- **Honor Retry-After header**: If toggled on (default) and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds. Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
  Toggle off if you want Cribl Stream to ignore all `retry-after` headers.

7. Next, you can configure the following **Optional Settings**:
   - **Request Timeout (secs)**: Here, you can set a maximum time period (in seconds) for an HTTP request to complete before Cribl Stream treats it as timed out. Defaults to `0`, which disables timeout metering.
   - **Reject unauthorized certificates**: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.
   - **Safe headers**: Optionally, list headers that you consider safe to log in plain text. Separate the header names with tabs or hard returns.
   - **Encoding**: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
8. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.
9. Select **Save**, then **Commit & Deploy**.
10. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

The sections described below are spread across several tabs, some of which contain collapsible accordions. Click the tab links at left to navigate among tabs. Click **Save** when you've configured your Collector.

See Formatting Expressions below for guidance on this Collector's specific requirement to enter JavaScript expressions as template literals.

Collector Sources cannot be selected or enabled in the QuickConnect UI.

Discover

This section guides you through configuring Discover Types.

Discover Type: Item List

Setting the **Discover type** to `Item List` exposes this additional field above the Health Check settings:

Discover items: List of items to return from the Discover task. Each returned item will generate a Health Check task, and can be referenced using `${id}` in the **Health Check URL**, the **Health Check parameters**, or the **Health Check headers**. Sample list with Cribl's health endpoint:

```
http://localhost:8080/services/collector/health,http://0.0.0.0:3000/services/collecto
```

Copy

Discover Type: JSON Response

Setting the **Discover type** to `JSON Response` exposes these additional fields above the Health Check settings:

Discover result: Allows hard-coding the Discover result. Must be a JSON object. Works with the **Discover data field**. Sample JSON entry:

```
`[ {"url": "http://localhost:8088/services/collector/health"},{ "url": "http://local
```

Copy

Discover data field: Within the response, this is the name of the field that contains discovery results. (Leave blank if the result is an array.) Sample JSON entry:

```
items, json: { items: [{id: 'first'},{id: 'second'}] }
```

Copy

In an XML response, this is the name of the element that contains discovery results. Sample XML entry:

```
result.items
<response>
 <items>
  <element>
   <id>first</id>
  </element>
  <element>
   <id>second</id>
  </element>
 </items>
</response>
```

Copy

Discover Type: HTTP Request

Setting the **Discover type** to `HTTP Request` exposes these additional fields above the Health Check settings:

Discover URL: Enter the URL to use for the Discover operation. This can be a constant URL, or a JavaScript expression to derive the URL.

> Where a URL (path or parameters) includes variables that might contain unsafe ASCII characters, encode these variables using `C.Encode.uri(paramName)`. (Examples of unsafe characters are: space, `$`, `/`, `=`.) Example URL with encoding: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)`.
>
> Request parameters that are not contained directly in the URL are automatically encoded. Cribl Stream URLs/expressions specified in the **Discover URL** field follow redirects.

Discover method: Select the HTTP verb to use for the Discover operation - `GET`, `POST`, or `POST with body`.

Discover POST body: Template for POST body to send with the Discover request. (This field is displayed only when you set the **Discover method** to `POST with body`.)

Discover parameters: Optional HTTP request parameters to append to the Discover request URL. These refine or narrow the request. Click **Add Parameter** to add parameters as key-value pairs:

- **Name**: Parameter name.
- **Value**: JavaScript expression to compute the parameter's value, normally enclosed in backticks (for example, `` `${id}` ``). Can also be a constant, enclosed in single quotes (`'id'`). Values without delimiters (for example, `id`) are evaluated as strings.

Discover headers: Optional Discover request headers.: Click **Add Header** to add headers as key-value pairs:

- **Name**: Header name.
- **Value**: JavaScript expression to compute the header's value, normally enclosed in backticks (for example, `` `${id}` ``). Can also be a constant, enclosed in single quotes (`'id'`). Values without delimiters (for example, `id`) are evaluated as strings.

Discover data field: Within the response JSON, name of the field that contains Discover results. Leave blank if the result is an array.

The following sections describe the remaining tabs, whose settings and content apply equally to all **Discover type** selections.

Health Check

Each request that this Collector sends constitutes one "health check." One source of insight into the health of the endpoint is the internal fields that the Collector emits. For example, to observe how slowly or quickly an endpoint is responding, you can monitor the value of the `elapsedMS` element nested in the `__collectStats` internal field.

Where a URL (path or parameters) includes variables that might contain unsafe ASCII characters, encode these variables using `C.Encode.uri(paramName)`. (Examples of unsafe characters are: space, `$`, `/`, `=`.) Example URL with encoding: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)`.

Request parameters that are not contained directly in the URL are automatically encoded. Cribl Stream URLs/expressions specified in the **Health check URL** field follow redirects.

Health Check Parameters

In **Health check parameters**, select **Add parameter** to add parameters as key-value pairs:

- **Field Name**: Unique name for the field you're adding.
- **Value**: JavaScript expression to compute the field's value, normally enclosed in backticks (for example, `` `${id}` ``). Can also be a constant, enclosed in single quotes (`'id'`). Values without delimiters (for example, `id`) are evaluated as strings.

Health check headers

In **Health check headers**, select **Add Header** to (optionally) add request headers as key-value pairs:

- **Name**: Header name.
- **Value**: JavaScript expression to compute the header's value, normally enclosed in backticks (for example, `` `${id}` ``). Can also be a constant, enclosed in single quotes (`'id'`). Values without delimiters (for example, `id`) are evaluated as strings.

Authentication

This section covers the process of configuring authentication options.

Login Authentication

Selecting `Login` exposes the following additional fields:

- **Login URL**: URL for the login API call, which is expected to be a POST call.
- **Username**: Login username.
- **Password**: Login password.
- **POST body**: Template for POST body to send with the login request. The `${username}` and `${password}` variables specify the corresponding credentials' locations in the message.
- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes.
- **Authorize expression**: JavaScript expression used to compute the Authorization header to pass in Discover and Health Check calls. Uses `${token}` to reference the token obtained from the login POST request.
- **Authentication headers**: Optionally, click **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.

Login Secret Authentication

Selecting **Login (credentials secret)** exposes the following additional fields:

- **Login URL**: URL for the login API call, which is expected to be a POST call.
- **Credentials secret**: Select a stored text secret in this drop-down, or click **Create** to configure a new secret.
- **POST body**: Template for POST body to send with the login request. The `${username}` and `${password}` variables specify the corresponding credentials' locations in the message.
- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes.
- **Authorize expression**: JavaScript expression used to compute the Authorization header to pass in Discover and Health Check calls. Uses `${token}` to reference the token obtained from the login POST

request.

- **Authentication headers**: Optionally, click **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.

OAuth Authentication

Selecting **OAuth** exposes the following additional fields:

- **Login URL**: Endpoint for the OAuth API call, which is expected to be a POST call.
- **Client secret parameter**: The name of the parameter to send with the **Client secret value**.
- **Client secret value**: The OAuth access token to authorize requests.
- **Extra authentication parameters**: Optionally, click **Add parameter** for each additional OAuth request parameter you want to send in the body of POST requests. Automatically sets the `Content-Type` header to `application/x-www-form-urlencoded`.
  In the resulting table, enter each parameter row's **Name**. The corresponding **Value** is a JavaScript expression to compute the parameter's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.
- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes.
- **Authorize expression**: JavaScript expression used to compute the Authorization header to pass in Discover and Health Check calls. Uses `${token}` to reference the token obtained from the login POST request.
- **Authentication headers**: Optionally, click **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.

OAuth Secret Authentication

Selecting **OAuth (text secret)** exposes the following additional fields:

- **Login URL**: Endpoint for the OAuth API call, which is expected to be a POST call.
- **Client secret parameter**: The name of the parameter to send with the **Client secret value**.
- **Client secret value (text secret)**: Select a stored text secret in this drop-down, or click **Create** to configure a new secret.
- **Extra authentication parameters**: Optionally, click **Add parameter** for each additional OAuth request parameter you want to send in the body of POST requests. Automatically sets the `Content-Type`

header to `application/x-www-form-urlencoded`.
In the resulting table, enter each parameter row's **Name**. The corresponding **Value** is a JavaScript expression to compute the parameter's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.

- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes.
- **Authorize expression**: JavaScript expression used to compute the Authorization header to pass in Discover and Health Check calls. Uses `${token}` to reference the token obtained from the login POST request.
- **Authentication headers**: Optionally, click **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the Health Check data.

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

- **Enabled**: Defaults to toggled off. Toggle on to enable the custom command.
- **Command**: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).
- **Arguments**: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

- **Event Breaker rulesets**: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `Cribl`.
- **Event Breaker buffer timeout**: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, etc. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. One use case might be a Health Check Collector that gathers a known, simple type of data from a single endpoint. This approach keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields**: List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc Health Check jobs if Cribl Stream restarts during the jobs' execution.

Formatting Expressions

JavaScript expression fields in this Health Check Collector behave differently from those elsewhere in Cribl Stream. Here, you must enter expressions as template literals, with placeholders referencing variables and JS functions. Here are a few examples:

- To reference the `__collectible.id` internal field in a URL, header, or parameter (etc.), you would write the variable as `` `${id}` ``.
- To reference a function call: `` `${Date.now()}` ``.
- To call a function that references a variable: `` `${Math.floor(earliest)}` ``.

Responses

The Health Check Collector generates error or success events based on the received response from the Health Check request.

Error Events

The Health Check Collector generates error events when no response is received, a request times out, or receives an error (non 200) response. The following is an example error event:

```
{
  "host": "Cribl.local",
  "_raw": "{\"url\":\"http://0.0.0.0:3000/services/collector/health\",\"method\":\"ge
  "method": "get",
  "url": "http://0.0.0.0:3000/services/collector/health",
  "statusCode": 404,
  "error": "Http Error, statusCode: 404, details: {\"host\":\"0.0.0.0:3000\",\"port\'
  "_time": 1660152577.93,
  "cribl_breaker": "Cribl:ndjson"
}
```


Copy

Success Events

The Health Check Collector generates success events when a 200 success status code is received. The following is an example success event:


```
{
  "host": "Cribl.local",
  "_raw": "{\"url\":\"http://0.0.0.0:3000/api/v1/health\",\"method\":\"get\",\"status
  "method": "get",
  "url": "http://0.0.0.0:3000/api/v1/health",
  "statusCode": 200,
  "body": "{\"status\":\"healthy\",\"startTime\":1660152114781}",
  "_time": 1660152114.781,
  "cribl_breaker": "Cribl:ndjson"
}
```


Copy

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These “meta” fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
  - `collectorType`: Indicates the type of Collector used for the job.
  - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
  - Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
    - `<timestamp>`: Timestamp when the job was initiated (in Unix time).
    - `<randomId>`: A random identifier to ensure uniqueness.
    - `adhoc`: Indicates the job was manually triggered.
    - `<Collector ID>`: The ID of the Collector.
  - Scheduled jobs are formatted as `collection:<Collector ID>`.
    - `<Collector ID>`: The ID of the Collector.
- `__collectStats` - This object's nested fields are:
  - `method`: The value you set for **Health Check** > **Health check method**.
  - `url`: The value you set for **Health Check** > **Health check URL**.
  - `elapsedMS`: The time elapsed from the moment the Health Check request was sent until the response was received.

Mitigating Stuck-Job Problems

8.1.1.7. REST / API Endpoint Collector

Cribl Stream supports collecting or pulling data from a REST API endpoint. The REST Collector is versatile and can collect data from a wide variety of upstream services. It provides multiple Discover types and Collect options.

Use REST Collectors when you need to ingest data from:

- Internal APIs built and used inside your Organization.
- External APIs from which your Organization uses data.
- Vendors or upstream API services that you use but which Cribl doesn't yet support natively.

For more usage examples, see Using REST / API Collectors and our adjacent guides.

Once you've configured a REST Collector, you can either run it immediately (ad hoc) to collect data, or schedule it to run on a recurring interval. See Schedule and Run Collector Jobs for more information.

How REST Collectors Handle Data

When the REST Collector runs, Cribl Stream executes a collection job. A collection job consists of tasks, which are the units of work to authenticate, discover, or collect the data. For example, a job that first authenticates and then collects data creates two tasks.

Cribl Stream processes collection jobs that run in these phases:

1. Discovery (optional)
2. Collection
3. Event Breaking
4. Filtering (optional)

Authentication takes place during the Collection phase if needed and can also happen during Discovery for HTTP calls.

You can also filter data based on specific criteria, such as time and/or field values, which you specify during the Collector's ad hoc or scheduled run.

Prerequisites

At a minimum, you need the URL of the API that you intend to collect data from. If the API requires authentication, you will also need authentication credentials.

Configure a REST Collector

Currently, you cannot configure Collector Sources in QuickConnect.

In Cribl Stream, set up a REST Collector:

1. From inside Cribl Stream, select a **Worker Group** to configure. Then, select **Data** > **Sources**. Select **REST** from the list of tiles or the **Collectors** tab. Next, select **Add Collector** to open the **Add Collector** modal.
2. In a simple setup, you only need to fill in the **General** and **Collect** settings in the Collector modal. You will also need the **Authentication** settings if the API requires authentication. Configure the following options under the **General** settings:
    - **Collector ID**: Unique ID for this Collector. This ID must be unique from other Collector IDs in your Cribl Stream instance. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description** Optionally, add an explanation of the function or purpose of the Collector.
3. Configure the following options under the **Collect** settings:
    - **Collect URL**: URL (constant or JavaScript expression) to use for the Collect operation. This URL can use an `authToken` variable to access an auth token. For example, if you define `1234567890` as an auth token and you use `http://myurl.com?authToken=${authToken}` as the Collect URL, the Collect operation will use `http://myurl.com?authToken=1234567890`. See Use JavaScript Expressions to Shape Incoming API Data for more information about using JavaScript with this field.

    If a URL (a path or parameters) includes variables that might contain unsafe ASCII characters, encode these variables using `C.Encode.uri(paramName)`. Considerations:

        - Examples of unsafe characters are: space, `$`, `/`, `=`.
        - Example URL with encoding: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)`. See Cribl Expressions - C.Encode.uri() for more information.
        - Request parameters that are not contained directly in the URL are automatically encoded.
        - Cribl Stream URLs/expressions specified in the **Collect URL** field can follow redirects.

    - **Collect method**: Select the HTTP verb to use for the Collect operation - `GET`, `POST`, `POST with body`, or `Other`.
        - **Collect POST body** (displays when you set the **Collect method** to `POST with body`): Template for POST body to send with the Collect request. See Use JavaScript Expressions to Shape Incoming API Data for more information about using JavaScript with this field.

- **Collect verb** (displays when you set the **Collect method** to `Other`): Custom HTTP method to use for the Collect operation.

- **Collect parameters**: Optional HTTP request parameters to append to the request URL. These refine or narrow the request. Select **Add Parameter** to add parameters as key-value pairs:
  - **Name** is the parameter name.
  - **Value** is a JavaScript expression to compute the parameter's value, normally enclosed in backticks (for example, `` `${earliest}` ``). The value can also be a constant, enclosed in single quotes (`'earliest'`). Cribl Stream evaluates values without delimiters (such as `earliest`) as strings.
  - This field can also accept the `${earliest}` and `${latest}` time range variable. See Time Range Variables for more information.

If both the **Collect URL** and **Collect parameters** use the same **Name** (key), the value from the **Collect URL** takes precedence.

- **Collect headers**: Optional collection request headers to add to the API call. Select **Add Header** to add collection request headers as key-value pairs:
  - **Name** is the header name.
  - **Value** is a JavaScript expression to compute the header's value, normally enclosed in single quotes (`'someConstantValue'`). The value can also be a template literal, enclosed in backticks. For example, `` `${new Date().getFullYear()}` ``. Cribl Stream evaluates values without delimiters (such as `someConstantValue`) as strings.
  - This field can also accept the `${earliest}` and `${latest}` time range variable. See Time Range Variables for more information.

You can also use **Collect headers** as an alternative to the Basic, Login, or OAuth authentication options. This method authenticates requests based on API keys using secrets. See How to Use Collect Headers for Authentication for more information.

- **Pagination**: Use this menu to select the pagination scheme for collection results. The available options are:

Option	Description
None	The default selection. Use this option if you do not need a pagination scheme.
Response Body Attribute / Response Header Attribute	Select `Response Body Attribute` to extract a value from the response body that identifies the next page of data to retrieve. Select `Response Header Attribute` to extract this next-page value from the response header. These selections expose additional

Option	Description
	fields. See Response Body Attribute / Response Header Attribute for information about these fields.
RFC 5988 - Web Linking	Select this option for APIs that follow RFC 5988 conventions to provide the next-page link in a header. This selection exposes three additional fields. See RFC 5988 - Web Linking for information about these fields.
Offset/Limit	Select this option to receive data from APIs that use offset/limit pagination. This selection exposes several additional fields. See Offset/Limit for information about these fields.
Page/Size	Select this option to receive data from APIs that use page/size pagination. This selection exposes several additional fields. See Page/Size for information about these fields.

4. In the **Authentication** settings, select one of the following options from the menu:

Option	Description
None	The default selection. Use this option if you do not need to use authentication. You can also select this option if you decide to embed the authentication secret directly in the request URL, such as with compatible REST servers like AWS. See How to Use Collect Headers for Authentication for more information.
Basic	Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
Basic (credentials secret)	Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret.
Login	Enables you to specify several credentials, then perform a POST to an endpoint during the Discover operation. The POST response returns a token, which Cribl Stream uses for later Collect operations. Exposes multiple extra fields. See Login Authentication for more information.
Login (credentials secret)	Like **Login**, except that you specify a secret referencing the credentials, rather than the credentials themselves. Exposes multiple extra fields. See Login Secret Authentication for more information.
OAuth	Enables you to directly enter credentials for authentication via the OAuth protocol. Exposes multiple extra fields. See OAuth Authentication for more information.
OAuth (text secret)	Like the **OAuth** option, except that you specify a stored secret referencing the credentials. Exposes multiple extra fields. See

Option	Description
	OAuth Secret Authentication for more information.
Google service account OAuth	Uses Google Cloud Platform (GCP) service account credentials to authenticate. Exposes multiple extra fields. See Google Service Account OAuth Authentication for more information.
Google service account OAuth (text secret)	Similar to **Google service account OAuth**, but uses the service account credentials directly as text within the configuration, instead of a separate file. Exposes multiple extra fields. See Google Service Account OAuth Secret Authentication for more information.
HMAC	HMAC (Hash-based Message Authentication Code) authentication allows you to securely authenticate your data collection requests by generating a unique signature for each request. This signature ensures the integrity and authenticity of the request, making it tamper-proof and verifiable by the server. Select an existing HMAC authentication signature or select **Create HMAC Function to continue**. See Generate HMAC Functions for REST Collector Requests for more information about generating an HMAC signature.

5. For the remaining options, refer to the settings listed for each section:
    - Discover Settings
    - Retries
    - Results
    - Advanced Settings
6. Select **Save**, then **Commit & Deploy**.
7. Verify that data is making it to Cribl Stream by viewing the **Live Data** feed from the Collector.

Configure as JSON

As an alternative to configuring a REST Collector with the user interface, you can configure REST Collectors in JSON mode instead. To use the JSON configuration mode, select the **Configure as JSON** tab at the top of the REST Collector configuration modal. Select the **Configure Collector** tab to return to the user interface configuration mode.

Some important considerations when working in the JSON configuration mode:

- Use this feature with caution. Only experienced and advanced Cribl administrators should configure as JSON.
- The advantage of using the JSON configuration mode is that you can cut and paste or import/export JSON configuration objects, such as from the Cribl Collector Templates repository or your own Git repository where you store version-controlled configurations that you have already validated and which you deploy regularly.

- The JSON configuration editor does not validate your REST Collector configuration. The editor only validates whether your JSON is well-formed, not whether the configuration is valid. Validate your REST Collector configuration with the user interface, not the JSON configuration editor.
- When you make a configuration change in the user interface mode and save it, that change appears in in the JSON configuration mode as well. Similarly, changes made and saved (and vice versa).

Discover Settings

Use the Discover step to better understand what data is available and how it is configured when interacting with a third party API. The results of a Discover call can range in variety and size, depending on the API.

The **Discover type** field provides four options, corresponding to different use cases. Each **Discover type** selection will expose a different set of **Collector Settings** fields:

Discover type	Description	Example use case
HTTP Request	Makes a REST call to dynamically get Discover results. This Discover type matches cases where you need to dynamically discover what you can collect from a REST endpoint. It most fully exploits Cribl Stream's Discover-and-then-Collect architecture. Each item returned will generate a Collect task, and Cribl Stream will spread each of those Collect tasks across all available Workers. As of Cribl Stream 3.0.2, this Discover type supports XML responses.	Make a REST call to get a list of available log files, then run Collect against each of those files.
JSON Response	Use when you need to manually define the Discovery results as a JSON object. This Discover type provides a **Discover result** field where you can (optionally) define Discover tasks as a JSON array of objects. Each entry returned by Discover will generate a Collect task, and Cribl Stream will spread each of those Collect tasks across all available Workers.	You can collect data for specific geo-locations in the National Weather Service API stream of worldwide weather data. This particular API requires multiple parameters in the request URL (latitude, longitude, and so forth), so **Item List** discovery would not work.

Discover type	Description	Example use case
Item List	Use when you need to enumerate Discover results as a list of strings. This Discover type matches cases where you want to enumerate a known list of **Discover items** to retrieve. Discovery will return one Collect task per item in the list, and Cribl Stream will spread each of those Collect tasks across all available Workers.	• Collect network traffic data that is tagged with specific subnets. • Collect weather data for a known list of ZIP codes.
None	Use when you want to skip the Discover step. This default option suppresses the Discover stage. Alternatively, you select this option and use the modal's **Collect** section to specify the endpoint and other details. Cribl Stream will assign the Collect task to a single Worker.	• When one simple API call will retrieve all the data you need. • Collect a list of configured Cribl Stream Pipelines.

Discover Type: HTTP Request

Setting the **Discover type** to `HTTP Request` exposes these additional fields:

- **Discover URL**: Enter the URL to use for the Discover operation. This can be a constant URL, or a JavaScript expression to derive the URL. This field can also accept the `${earliest}` and `${latest}` time range variable. See Time Range Variables for more information.

  If a URL (a path or parameters) includes variables that might contain unsafe ASCII characters, encode these variables using `C.Encode.uri(paramName)`. Considerations:

  - Examples of unsafe characters are: space, `$`, `/`, `=`.
  - Example URL with encoding: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)`.
  - Request parameters that are not contained directly in the URL are automatically encoded.
  - Cribl Stream URLs/expressions specified in the **Collect URL** field follow redirects.

- **Discover method**: Select the HTTP verb to use for the Discover operation - `GET`, `POST`, `POST with body`, or `Other`.
- **Discover POST body**: Template for POST body to send with the Discover request. (This field is displayed only when you set the **Discover method** to `POST with body`.)

- **Discover verb**: Custom HTTP method to use for the Discover operation. (This field is displayed only when you set the **Discover method** to `Other`.)
- **Discover parameters**: Optional HTTP request parameters to append to the Discover request URL. These refine or narrow the request. Select **Add Parameter** to add parameters as key-value pairs:
    - **Name**: Parameter name.
    - **Value**: JavaScript expression to compute the parameter's value, normally enclosed in backticks (for example, `` `${earliest}` ``). Can also be a constant, enclosed in single quotes (`'earliest'`). Values without delimiters (for example, `earliest`) are evaluated as strings.
    - This field can also accept the `${earliest}` and `${latest}` time range variable. See Time Range Variables for more information.
- **Discover headers**: Optional Discover request headers. Select **Add Header** to add headers as key-value pairs:
    - **Name**: Header name.
    - **Value**: JavaScript expression to compute the header's value, normally enclosed in backticks (for example, `` `${earliest}` ``). Can also be a constant, enclosed in single quotes (`'earliest'`). Values without delimiters (for example, `earliest`) are evaluated as strings.
    - This field can also accept the `${earliest}` and `${latest}` time range variable. See Time Range Variables for more information.
- **Pagination**: Use this menu to select the pagination scheme for Discover results. Defaults to `None`. The available options are:

Option	Description
None	The default selection. Use this option if you do not need a pagination scheme.
Response Body Attribute / Response Header Attribute	Select `Response Body Attribute` to extract a value from the response body that identifies the next page of data to retrieve. Select `Response Header Attribute` to extract this next-page value from the response header. These selections expose additional fields. See Response Body Attribute / Response Header Attribute for information about these fields.
RFC 5988 - Web Linking	Select this option for APIs that follow RFC 5988 conventions to provide the next-page link in a header. This selection exposes three additional fields. See RFC 5988 - Web Linking for information about these fields.
Offset/Limit	Select this option to receive data from APIs that use offset/limit pagination. This selection exposes several additional fields. See Offset/Limit for information about these fields.

Option	Description
Page/Size	Select this option to receive data from APIs that use page/size pagination. This selection exposes several additional fields. See Page/Size for information about these fields.

- **Discover data field**: Within the response, this is the name of the field that contains discovery results.
  - If the response is a JSON object that contains an array, and you want to generate a separate Collect task for each item in the array, set the **Discover data field** to the key of the array in the JSON object. For example, if the response is `{ "logFiles": ["cribl.log", "cribl.log.1", "cribl.log.2", "audit.log"] }`, setting **Discover data field** to `logFiles` would result in four Collect tasks, one for each log file.
  - If the response itself is a JSON array, like `["cribl.log", "cribl.log.1", "cribl.log.2", "audit.log"]`, leave the **Discover data field** blank for each item in the array to become a collection task.

See Parse Discover Returns With Arrays as JSON Object for an example of how to parse arrays inside JSON objects as separate collection tasks.

- **Strict discover response parsing**: Toggle on to explicitly set the **Discover response format** to handle `JSON`, `XML`, `NDJSON` or `Plain text` data. This functionality provides a more direct and configurable way to handle data from a wider variety of APIs. This functionality is particularly useful when an API's `Content-Type header` is missing or inaccurate, as it allows you to bypass the Collector's default best-effort parsing and instead enforce a specific format. When the configured strict format is enabled, the Collector will fail if the response does not match the expected format, which helps ensure data integrity and predictable behavior.
- **Format discover result with custom code**: Toggle on to provide custom JavaScript code, in the **Format discover result** field, that manipulates the data returned by a Discover request before it's used for collection. This functionality allows you to influence the collection process by transforming Discover results into a specific set of collection tasks.

The custom code operates on the `__e` variable, which is a JSON object or array containing the original Discover results. You can modify the contents of `__e` to define tasks for parallel collection. Here are some key points to remember when working with `__e`:

- Mutable object/array: `__e` is mutable, meaning you can directly modify its contents.
- Response structure: The structure of `__e` depends on the specific response from your Discover request.

For example, to split data by user IDs, assume the Discover result contains user IDs:

```
`[{ "name": "root", "id": 1}, { "name": "joe", "id": 2 }, { "name": "jane", "id": 3 },
```

To achieve a collection task that runs for all users except `root`, you'd use:

`__e['tasks'] = __e.filter(item => item.name != 'root');`

As another example, imagine an API with two endpoints:

- Endpoint that returns the count of objects that match search criteria.
- Endpoint for collection that returns items using search criteria that also supports pagination.

You can leverage the **Format discover results** option to do the following:

1. Run a Discover REST API call to return the number of items that match search criteria, like an SQL count query.
2. Use the count matching items to generate collect tasks, one for each page of data to retrieve.

For example, assume the discover REST API call returns the following:

`{ "numMatchingItems": 100000 }`

Assume the max page size is 10,000 for the collect API.

The **Format discover result with custom code** option allows you to implement logic that generates an array of collection tasks, one for each page. You can then leverage this array in conjunction with the collection request parameters to retrieve a unique page of data for each collection task.

For example: `http://0.0.0.0:8004/query?sysparm_offset=${offset}&sysparm_limit=${limit}`

Now you'll transform a Discover result of `{ "numMatchingItems": 100000 }` into 10 distinct collect tasks, each retrieving 10,000 results per page. Some of the tasks can be run in parallel depending on job settings and system load.

```
// Expecting: { \"numMatchingItems\" : xxxxx };
if (Array.isArray(__e)) {
  const count = __e['numMatchingItems'];
  let pageSize=10000;
  let numPages=count/pageSize;
  if (count % pageSize) {
     numPages++; // count not evenly divisible by page size, add one more collect tas
  }
  // Add a collect task for each page of data to be retrieved.
  for (let i = 0; i < numPages; i++) {
    __e.push({ offset: i*pageSize, limit: pageSize, count });
  }
}
else {
  __e['__resultOut'] = {"error": "Unexpected - discover result is NOT an array"}
}
```

Copy

See another use case example in Using REST/API Collectors.

Discover Type: Item List

Setting the **Discover type** to `Item List` exposes this additional field:

- **Discover items**: List of items to return from the Discover task. Each returned item will generate a Collect task, and can be referenced using `${id}` in the **Collect URL**, the **Collect parameters**, or the **Collect headers**.

Discover Type: JSON Response

Setting the **Discover type** to `JSON Response` exposes these additional fields above the Common Collector Settings:

Discover result: Allows hard-coding the Discover result. Must be a JSON object. Works with the **Discover data field**.

Discover data field: Within the response, this is the name of the field that contains discovery results.

- If the response is a JSON object that contains an array, and you want to generate a separate Collect task for each item in the array, set the **Discover data field** to the key of the array in the JSON object. For example, if the response is `{ "logFiles": ["cribl.log", "cribl.log.1", "cribl.log.2", "audit.log"] }`, setting **Discover data field** to `logFiles` would result in four Collect tasks, one for each log file.

- If the response itself is a JSON array, like `["cribl.log", "cribl.log.1", "cribl.log.2", "audit.log"]`, leave the **Discover data field** blank for each item in the array to become a collection task.

Sample JSON entry:

```
items, json: { items: [{id: 'first'},{id: 'second'}] }
```

Copy

In an XML response, this is the name of the element that contains discovery results. Sample XML entry:

```
result.items
<response>
 <items>
  <element>
   <id>first</id>
  </element>
  <element>
   <id>second</id>
  </element>
 </items>
</response>
```

Copy

Collect Settings

See Configure a REST Collector for information about most of the settings in the **Collect** section of the REST Collector configuration. The following sections provide additional information about these settings.

How to Use Collect Headers for Authentication

You can use collect headers as an alternative to the Basic, Login, or OAuth authentication options. This method authenticates requests based on API keys using secrets.

To use this authentication method:

1. In the REST Collector configuration **Collect** section, select **Add Header** to add a new header key-value pair.
2. In the **Name** field, assign the header a name.

3. In the **Value** field, use the `C.Secret` Javascript expression method in this field to provide the key-value for the Authentication header. For example, depending on the requirements of your application, you could use this key-value pair:
   `` `Bearer: ${C.Secret('<secretId>', 'text').value}` ``
   or:
   `` `Token: ${C.Secret('<secretId>', 'text').value}` ``
4. In the **Authentication** section, select `None` in the **Authentication** menu.

Pagination Settings

Use the Pagination menu in the **Collect** section to select the pagination scheme for collection results. Defaults to `None`.

Response Body Attribute / Response Header Attribute

Select `Response Body Attribute` to extract a value from the response body that identifies the next page of data to retrieve.

Select `Response Header Attribute` to extract this next-page value from the response header. Either of these selections exposes these additional fields:

- **Response attribute**: Name(s) of attribute(s) in the response payload or header that contain next-page information. If you have multiple attributes with the same name within the response body, you can provide the path to the one you need, for example: `foo.0.bar`. If Cribl Stream cannot find an exact match for a specified attribute name, it will fallback to the last-found partial match.
- **Page limit**: The maximum number of pages to retrieve per Collect task. Defaults to `50` pages. Set to `0` to retrieve all pages.

Selecting `Response Body Attribute` exposes one more field:

- **Last-page expression**: Optionally, enter a JavaScript expression that Cribl Stream will use to determine when the last page has been reached. The values tested by this expression must be present in the **Response attributes** section above. Example expression: `morePages === false`.

Use this expression with APIs like Smartsheet Event Reporting, which return an attribute that signals whether more pages are present, rather than simply returning an empty result set beyond the last page.

RFC 5988 - Web Linking

Select this option with APIs that follow RFC 5988 conventions to provide the next-page link in a header. This selection exposes three additional fields:

- **Next page relation name**: Header substring that refers to the next page in the result set. Defaults to `next`, corresponding to the following example link header: `<https://myHost/curPage>; rel="self" <https://myHost/nextPage>; rel="next"`
- **Current page relation name**: Optionally, specify the relation name within the link header that refers to the current result set. In this same example, `rel="self"` refers to the current page of results: `<https://myHost/curPage>; rel="self" <https://myHost/nextPage>; rel="next"`
- **Page limit**: The maximum number of pages to retrieve. Defaults to `50` pages. Set to `0` to retrieve all pages.

Offset/Limit

Select this option to receive data from APIs that use offset/limit pagination. This selection exposes several additional fields:

- **Offset field name**: Query string parameter that sets the index from which to begin returning records. For example: `/api/v1/query?term=cribl&limit=100&offset=0`. Defaults to `offset`.
- **Starting offset**: (Optional) offset index from which to start request. Defaults to undefined, which will start collection from the first record.
- **Limit field name**: Query string parameter to set the number of records retrieved per request. For example: `/api/v1/query?term=cribl&limit=100&offset=0`. Defaults to `limit`.
- **Record limit**: Maximum number of records to collect per request. Defaults to `50`.
- **Total record count field name**: Identifies the attribute, within the response, that contains the total number of records for the query.
- **Page limit**: The maximum number of pages to retrieve. Defaults to `50`. Set to `0` to retrieve all pages.
- **Zero-based index**: Toggle on to indicate that the requested data's first record is at index `0`. The default (No) indicates that the first record is at index `1`.

Page/Size

Select this option to receive data from APIs that use page/size pagination. This selection exposes several additional fields:

- **Page number field name**: Query string parameter that sets the page index to be returned. For example: `/api/v1/query?term=cribl&page_size=100&page_number=0`. Defaults to `page`.
- **Starting page number**: (Optional) page number from which to start request. Defaults to undefined, which will start collection from the first page.
- **Page size field name**: Query string parameter to set the number of records retrieved per request. For example: `/api/v1/query?term=cribl&page_size=100&page_number=0`. Defaults to `size`.
- **Record limit**: Maximum number of records to collect per page. Defaults to `50`.

- **Total page count field name**: Identifies the attribute, within the response, that contains the total number of pages for the query.
- **Total record count field name**: Identifies the attribute, within the response, that contains the total number of records for the query.
- **Page limit**: The maximum number of pages to retrieve. Defaults to `50`. Set to `0` to retrieve all pages.
- **Zero-based index**: Toggle on to indicate that the requested data's first record is at index `0`. The default (No) indicates that the first record is at index `1`.

Authentication Settings

See Configure a REST Collector for information about most of the options in the **Authentication** section of the REST Collector configuration. The following sections provide additional information about these authentication options.

You can also use **Collect headers** as an alternative to the Basic, Login, or OAuth authentication options. This method authenticates requests based on API keys using secrets. See How to Use Collect Headers for Authentication for more information.

Login Authentication

Login is the best authentication type to use for all services that return a JSON web token (JWT) or other authentication token whose content type is `application/json` or `content/plain`.

Selecting **Login** exposes the following additional fields:

- **Login URL**: URL for the login API call, which is expected to be a POST call.
- **Login username**: Login username.
- **Login password**: Login password.
- **POST body**: Template for POST body to send with the login request. The `${username}` and `${password}` variables specify the corresponding credentials' locations in the message. You can also use the `C.Secret()` method within the POST body for dynamic values.
- **Get auth token from header**: Toggle on to parse the authentication token from the HTTP Authorization response header of the login call, instead of the standard JSON response body. This is useful for APIs that return the token in the header rather than in the response payload, which is a non-standard pattern. Format the header value as `Bearer <authToken>`. When on, **Token attribute** is hidden and not used.
- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes. If the response from the auth request provides the JWT as `plain/text`, leave this field blank.
- **Authorization header**: Authorization header key to pass in Discover and Collect calls. Defaults to the literal name `Authorization`. (Cribl Stream 4.0 and newer.)

- **Authorize expression**: JavaScript expression used to compute an Authorization header to pass in Discover and Collect calls. Uses `${token}` to reference the token obtained from a Login or OAuth authorization request. If you need to pass the token in a Collector or Discover request parameter, URL, or request body, you can access it with `${authToken}`.
- **Authentication headers**: Optionally, select **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `earliest` instead of `` `${earliest}` `` - will be evaluated as strings.

Login Secret Authentication

Login is the best authentication type to use for all services that return a JSON web token (JWT) or other authentication token whose content type is `application/json` or `content/plain`.

Selecting **Login (credentials secret)** exposes the following additional fields:

- **Login URL**: URL for the login API call, which is expected to be a POST call.
- **Credentials secret**: Select a stored text secret in this drop-down, or select **Create** to configure a new secret.
- **POST body**: Template for POST body to send with the login request. The `${username}` and `${password}` variables specify the corresponding credentials' locations in the message. You can also use the `C.Secret()` method within the POST body for dynamic values.
- **Get auth token from header**: Toggle on to parse the authentication token from the HTTP Authorization response header of the login call, instead of the standard JSON response body. This is useful for APIs that return the token in the header rather than in the response payload, which is a non-standard pattern. Format the header value as `Bearer <authToken>`. When on, **Token attribute** is hidden and not used.
- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes. If the response from the auth request provides the JWT as `plain/text`, leave this field blank.
- **Authorization header**: Authorization header key to pass in Discover and Collect calls. Defaults to the literal name `Authorization`. (Cribl Stream 4.0 and newer.)
- **Authorize expression**: JavaScript expression used to compute an Authorization header to pass in Discover and Collect calls. Uses `${token}` to reference the token obtained from a Login or OAuth authorization request. If you need to pass the token in a Collector or Discover request parameter, URL, or request body, you can access it with `${authToken}`.
- **Authentication headers**: Optionally, select **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `earliest` instead of `` `${earliest}` `` - will be evaluated as strings.

OAuth Authentication

Selecting **OAuth** exposes the following additional fields:

- **Login URL**: Endpoint for the OAuth API call, which is expected to be a POST call.
- **Client secret parameter**: The name of the parameter to send with the **Client secret value**.
- **Client secret value**: The OAuth access token to authorize requests.
- **Extra authentication parameters**: Optionally, select **Add parameter** for each additional OAuth request parameter you want to send in the body of POST requests. Automatically sets the `Content-Type` header to `application/x-www-form-urlencoded`.
  In the resulting table, enter each parameter row's **Name**. The corresponding **Value** is a JavaScript expression to compute the parameter's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `earliest` instead of `` `${earliest}` `` - will be evaluated as strings.
- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes.
- **Authorization header**: Authorization header key to pass in Discover and Collect calls. Defaults to the literal name `Authorization`. (Cribl Stream 4.0 and newer.)
- **Authorize expression**: JavaScript expression used to compute an Authorization header to pass in Discover and Collect calls. Uses `${token}` to reference the token obtained from a Login or OAuth authorization request. If you need to pass the token in a Collector or Discover request parameter, URL, or request body, you can access it with `${authToken}`.
- **Authentication headers**: Optionally, select **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `earliest` instead of `` `${earliest}` `` - will be evaluated as strings.

OAuth Secret Authentication

Selecting **OAuth (text secret)** exposes the following additional fields:

- **Login URL**: Endpoint for the OAuth API call, which is expected to be a POST call.
- **Client secret parameter**: The name of the parameter to send with the **Client secret value**.
- **Client secret value (text secret)**: Select a stored text secret in this drop-down, or select **Create** to configure a new secret.
- **Extra authentication parameters**: Optionally, select **Add parameter** for each additional OAuth request parameter you want to send in the body of POST requests. Automatically sets the `Content-Type` header to `application/x-www-form-urlencoded`.
  In the resulting table, enter each parameter row's **Name**. The corresponding **Value** is a JavaScript expression to compute the parameter's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `earliest` instead of `` `${earliest}` `` - will be evaluated as strings.

- **Token attribute**: Path to the token attribute in the login response body. Supports nested attributes.
- **Authorization header**: Authorization header key to pass in Discover and Collect calls. Defaults to the literal name `Authorization`. (Cribl Stream 4.0 and newer.)
- **Authorize expression**: JavaScript expression used to compute an Authorization header to pass in Discover and Collect calls. Uses `${token}` to reference the token obtained from a Login or OAuth authorization request. If you need to pass the token in a Collector or Discover request parameter, URL, or request body, you can access it with `${authToken}`.
- **Authentication headers**: Optionally, select **Add Header** for each custom auth header you want to define. In the resulting table, enter each header row's **Name**.
  The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `earliest` instead of `` `${earliest}` `` - will be evaluated as strings.

Google Service Account OAuth Authentication

Selecting **Google service account OAuth** exposes the following additional fields:

- **Scopes**: OAuth 2.0 scopes used to request access to Google APIs during authentication. See the Google page OAuth 2.0 Scopes for more information. Select **Add Scope** to add a scope.
- **Service account credentials**: Contents of the Google Cloud service account credentials (JSON keys) file. To upload a file, select the upload icon in this field's upper right corner.
- **Impersonated account's email address**: Email address of a user account with super admin permissions to the resources the Collector will retrieve.

Google Service Account OAuth Secret Authentication

Selecting **Google service account OAuth (text secret)** exposes the following additional fields:

- **Scopes**: OAuth 2.0 scopes used to request access to Google APIs during authentication. See the Google page OAuth 2.0 Scopes for more information. Select **Add Scope** to add a scope.
- **Service account credentials (text secret)**: Text secret containing the value of the Google service account credentials. Select an existing text secret or select **Create** to create a new secret.
- **Impersonated account's email address**: Email address of a user account with super admin permissions to the resources the Collector will retrieve.

Retries

The Retries Settings determine how to handle API calls that fail or timeout. It has the following fields:

- **Retry type**: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.

- **Initial retry interval (ms)**: Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.
- **Retry limit**: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.
- **Backoff multiplier**: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so forth.
- **Longest interval between retries (ms)**: Maximum time to wait between retry attempts. This is useful for APIs with high backoff timeouts, enabling you to configure longer wait times to avoid excessive retry attempts. Defaults to `20000` ms (20 seconds). Supports values up to `900000` ms (15 minutes) to handle longer backoff requirements. Setting a long retry interval can lead to significantly extended job execution times, which could prevent other jobs from running on the Worker because of concurrent task limits.
- **Retry HTTP codes**: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.
- **Honor Retry-After header**: When toggled on (the default) and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds. Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms). When toggled off, Cribl Stream ignores all `retry-after` headers.
    - Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
    - When toggled off, Cribl Stream ignores all `retry-after` headers.
- **Retry Connection Timeout**: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.
- **Retry Connection Reset**: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.
- **Retry-After header name**: Optionally, if the service uses a different header name for conveying retry information, specify that custom name here. For example, a service might use `x-ratelimit-retry-after` instead.

Results

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Custom Commands

Custom Command: When enabled, you can pass the data from this input to an external command for processing, before the data continues downstream. Enabling this field exposes these additional fields:

- **Command**: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).
- **Arguments**: Select **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breaking

Event Breaker rulesets: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`. Select **Add Ruleset** to add a new row to the ruleset table. Select the ruleset to see a list of available recommended rulesets for your API. Change the order of the rules to determine which rule is applied in order.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Event Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Results Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, and so forth. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. One use case might be a REST Collector that gathers a known, simple type of data from a single endpoint. This approach keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional, and available only when **Send to Routes** is toggled on.

Throttling: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, and so forth. (Example: `42 MB`.) Default value of `0` indicates no throttling.

Advanced Settings

The **Advanced Settings** section of the REST Collector configuration has the following options:

- **Request timeout (secs)**: Here, you can set a maximum time period (in seconds) for an HTTP request to complete before Cribl Stream treats it as timed out. Defaults to `0`, which disables timeout metering.
- **Round-robin DNS**: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.
- **Disable time filter**: Toggle on to bypass the collection job event timestamp filtering. By default, Collectors apply time filtering based on the configured time range. Events with timestamps outside this range are discarded. Enabling this setting disables timestamp-based filtering, allowing all returned events to be processed regardless of their timestamps. This is useful when the data source's timestamps do not align with the configured time range, causing unexpected filtering, or when you need to process all data regardless of their timestamps. If you encounter zero results despite the data source returning events, disabling the time filter can help isolate whether timestamp filtering is the issue. For details, see No Discover or Collector Event Results.
- **Decode URL**: When toggled off, Cribl Stream will not decode URLs before sending the HTTP request. This setting applies to collect, authentication, and pagination requests. Defaults to toggled on.
- **Reject unauthorized certificates**: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

- **Capture response headers**: Toggle on to include the headers from the API response of a collect call under a `resHeaders` key in the `__collectible` object that is added to collected events.
- **Stop on empty results:** Toggle on to use your Event Breaker Ruleset to optimize paginated data collection. When toggled on, the REST Collector immediately stops requesting subsequent pages if a current page returns no new data events. The Collector determines a page has no data by checking the output of the Event Breaker rules. If the Event Breaker returns zero events from the received payload, the pagination loop will terminate. This setting prevents the Collector from making unnecessary API calls when the data source reaches its current tail, improving Pipeline efficiency and reducing the consumption of external API rate limits.

  To ensure the **Stop on empty results** setting works as expected, validate that you have properly configured the Event Breaker rules to successfully identify and parse all valid events from the payload. If you accidentally misconfigure the Event Breaker and it returns zero events incorrectly, the collection process will stop early.

- **Safe headers**: Optionally, list headers that you consider safe to log in plain text. Separate the header names with tabs or hard returns. These headers will not be redacted when exporting the Collector configuration as JSON.
- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields** : List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
  - `collectorType`: Indicates the type of Collector used for the job.

    - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
    - `resHeaders`: Added only when **Advanced Settings** > **Capture response headers** is toggled on.
- `__collectStats` - This object's nested fields are:
    - `method`: The value you set for **Collect** > **Collect method**.
    - `url`: The value you set for **Collect Settings (Common)** > **Collect URL**.
    - `elapsedMS`: The time elapsed from the moment the Collector's request was sent until the response was received.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
    - Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
        - `<timestamp>`: Timestamp when the job was initiated (in Unix time).
        - `<randomId>`: A random identifier to ensure uniqueness.
        - `adhoc`: Indicates the job was manually triggered.
        - `<Collector ID>`: The ID of the Collector.
    - Scheduled jobs are formatted as `collection:<Collector ID>`.
        - `<Collector ID>`: The ID of the Collector.

Use JavaScript Expressions to Shape Incoming API Data

While you configure a REST Collector, you may need to translate the incoming data structure from an upstream API into the data structure that Cribl Stream expects and can work with. To effectively translate data, you need to know:

- The data structure of returns sent by the upstream API.
- How that upstream data structure differs from the data structure that Cribl Stream expects to receive.
- How to shape (transform) the data structure from the upstream API into the data structure expected by Cribl Stream.

You can shape incoming API data using JavaScript expressions when you configure a REST Collector. Various fields in the Collector accept JavaScript expressions. Fields that accept JavaScript are set apart by a blue JS bar, such as this example of the **Collect URL** field:

Example of a field that accepts Javascript expressions

JavaScript expression fields in the REST Collector behave differently from other similar JavaScript fields in Cribl Stream. In the REST Collector,you must enter expressions as template literals. Template literals (also known as template strings) allow you to use placeholders to reference dynamic variables and JavaScript functions. Cribl Stream uses some built-in variables or you can build your own custom placeholder variables to transform incoming data as needed.

For example:

- To reference the earliest time variable in a URL, header, or parameter, you would write the variable as `` `${earliest}` ``.
- To reference a function call: `` `${Date.now()}` ``.
- To call a function that references a variable: `` `${Math.floor(earliest)}` ``.

Time and Date Strings

Refer to the API documentation that you want to pull data from to gather information about how the API formats timestamps. The API could return the date in a variety of different formats, such as:

- `yyyy/MM/dd`
- `yyyy-MM-dd`
- `dd-MM-yyyy`

To effectively parse different patterns of dates in Cribl Stream, you may need to use a combination of different settings and JavaScript expressions.

Time Range Variables: `` `${earliest}` `` and `` `${latest}` ``

The following fields accept `` `${earliest}` `` and `` `${latest}` `` variables, which reference any **Time Range** values that have been set in manual or scheduled collection jobs:

Collect settings:

- Collect URL
- Collect parameters
- Collect headers

Discover settings:

- Discover URL
- Discover parameters

- Discover headers

As an example, here is a **Collect URL** entry using these variables: `http://localhost/path?from=${earliest}&to=${latest}`

APIs vary in their requirements for time formatting required. Consult the API documentation for the API to format it correctly.

For additional help with time filtering, see How Time Filtering Works for Scheduled Jobs.

Parse Discover Returns With Arrays as JSON Objects

When events are returned as an array, you need a JSON array event breaker is needed to separate out the individual fields so that the filter can be applied.

Some APIs return data as an array, such as `{ "logFiles": ["file-name-1.log", "file-name-2.log", "file-name-3.log.2", "audit.log"] }`. In this case, Cribl Stream treats each item in the array as a collection task.

However, some APIs return a JSON object that contains an array and you need to tell Cribl Stream to generate a separate Collect task for each item in the array, as in this example:

```
[
  {
    "LogFile": "/collect?filename=1",
    "attributes": {
      "field": 1
    }
  },
  {
    "LogFile": "/collect?filename=2",
    "attributes": {
      "field": 1
    }
  }
]
```

Copy

To properly parse the array inside the JSON object:

1. In the **Discover Settings**, select **HTTP Request** as the **Discover type**.

2. Set the options for your API in the standard required fields (such as **Discover URL**) as needed.
3. In the **Discover data field**, type `file` to create a custom property to store the `LogFile` array values.
4. Toggle **Format discover result with custom code** on.
5. In the **Format discover result** field, type:

```
__e.file = __e.map(item => item.LogFile)
```

Copy

This expression processes an array `__e`, extracts the LogFile property from each element in that array, and assigns the resulting array of `LogFile` values to `__e.file`. Essentially, it creates a new array of log file paths (or similar information) from the original array and stores it in the file property of `__e`. Each property becomes a separate collection task.

This JavaScript expression is performing the following actions:

- `__e.map(...)`: This is using the `map()` method on the JSON object. The `map()` method creates a new array by applying the provided function to each element in the original array.
- `item => item.LogFile`: This is an arrow function that takes each item in the `__e` array and returns the `LogFile` property of that `item`.
- `__e.file =`: The result of the `map()` function, which is a new array containing only the `LogFile` values of each item in the original `__e` array, is assigned to the `file` property of `__e`.

Mutually Exclusive HTTP Request Parameters

Some APIs have mutually exclusive request, query, or URL parameters. In this situation, you may have parameters that cannot be used together in an API call. You must use either one or the other. For example, you may need to send one parameter in the first query and a different parameter for all subsequent queries.

To solve this problem, try using some of the formulas discussed in this Cribl User Group Community video at approximately the 30 minute mark: ServiceNow and Smartsheet.

For example, one user working with the Smartsheet API needed to send `since` in the first pagination query and `streamPosition` in all subsequent pagination queries. This user used these two **Collect Parameters**:

Name	Value
`since`	`${(!nextStreamPosition ? C.Time.strftime(Date.now()/1000-300, '%Y-%m-%d%T%H:%M:%S.000Z') : '')}`
`streamPosition`	`${((moreAvailable==false && nextStreamPosition!=null) ? '' : nextStreamPosition.toString())}`

Then, the user:

1. Set **Pagination** to `Response Body Attribute`.
2. Defined the **Response attribute** as `nextStreamPosition`.
3. Set the **Last-page expression** to `moreAvailable=='false'`.

Work with State Tracking

You can configure the Collector to track state, either by time or another arbitrary value. This can help prevent overlaps between jobs, where subsequent Collector runs may return some of the same results as previous runs. Similarly, it can help prevent gaps in data by allowing a Collector run to pick up from where the last run ended.

The Collector updates its saved state even if a collection job does not fully succeed. As long as new data is collected during the run, regardless of whether the job completes successfully or encounters errors, the state is updated to reflect the latest collected point.

State tracking can be particularly useful for tracking the greatest `_time` value for events returned from a collection run, but you can also use it to derive arbitrary state for custom use cases. You can access saved state values in any Collector field that supports JavaScript expressions.

For a hands-on example of setting up a Collector with state tracking enabled, see Using REST/API Collectors.

Track State for Collector Runs

You can enable state tracking under the `State Tracking` section when scheduling or performing a full run of the Collector (Preview and Discover runs do not support state tracking). Once enabled, two more fields will appear for state tracking configuration.

State update expression: JavaScript expression that defines how to update the state from an event. Use the event's data and the current state to compute the new state.

State merge expression: JavaScript expression that defines which state to keep when merging a task's newly reported state with the previously saved state. Evaluates `prevState` and `newState` variables, resolving to the state to keep.

The default values for these fields are configured to track state by the latest `_time` field found in events gathered in a collection run.

Understand State Expression Fields

The **State update** and **State merge** expressions control how state is derived from a collection run and how it is merged with existing state, respectively. They're preconfigured to work with the common use case of tracking

state by latest `_time`, but you may need to update them for other use cases. To understand what these fields do, let's break down the default values.

State update expression

This expression has a default value of:

```
__timestampExtracted !== false && {latestTime: (state.latestTime || 0) > _time ? stat
```

Copy

__timestampExtracted !== false - the `__timestampExtracted` field is set to false if the Event Breaker was unable to parse time for the event. If this is the case, we don't want to update state (the event's `_time` value defaults to `Date.now()` if the Event Breaker was unable to parse out the correct time). If `__timestampExtracted` is `false`, we take advantage of short-circuit evaluation to not update state.

State values must resolve to an object, such as:

```
{ "latestTime": 17122806161 }
```

Copy

If the expression does not resolve to an object, Cribl Stream will ignore the result.

{latestTime: (state.latestTime || 0) > _time ? state.latestTime : _time} - compare `state.latestTime` to the event's `_time` value, keeping whichever value is greater.

State merge expression

This expression has a default value of:

```
prevState.latestTime > newState.latestTime ? prevState : newState
```

Copy

It compares `prevState` (the state that was previously saved) to `newState` (the state reported from the most recent collection task), keeping the state with the greatest `latestTime` value.

Manage Collector State

Select **Manage State** to view, modify, or delete Collector state. For more information, see Manage Collector State.

Proxy Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshoot

Response Errors

The Collector treats all non-`200` responses from configured URL endpoints as errors. This includes `1xx`, `3xx`, `4xx`, and `5xx` responses.

On Discovery, Preview, and most Collect jobs, it interprets these as fatal errors. On Collect jobs, the Collector treats these exceptions as non-fatal:

- Where a Collect job launches multiple tasks, and only a subset of those tasks fail, Cribl Stream places the job in failed status, but treats the error as non-fatal. (Note that Cribl Stream does not retry the failed tasks.)
- Where a Collect job receives a `3xx` redirection error code, it follows the error's treatment by the underlying library, and does not necessarily treat the error as fatal.

Mitigate Stuck-Job Problems

Occasionally, a scheduled job fails, but continues running for hours or even days, until someone intervenes and cancels it. If left alone, such a "stuck", "orphaned," or "zombie" job will never complete. This can cause missing events in downstream receivers, along with `HTTP timeout` or similar errors in Cribl Stream's logs.

To keep stuck jobs from running excessively long:

- First, try setting **Optional Settings** > **Request timeout (secs)** - whose default of `0` means "wait forever" - to a desired maximum duration.
- If adjusting **Timeout (secs)** does not fix the problem, try the global setting **Job Timeout** - whose default of `0` allows a job to run indefinitely - to a desired maximum duration. You'll find **Job Timeout** among the task manifest and buffering limits.

Using these settings in tandem works like this:

- **Timeout (secs)** limits the time that Cribl Stream will wait for an HTTP request to complete.
- Then, if a job gets stuck and keeps running beyond that limit, **Job Timeout** can catch and terminate the job, because it monitors the overall time the job has been running.

Error Message: `TypeError [ERR_INVALID_URL]`

The full text of this error message:

```
"reason.stack == `TypeError [ERR_INVALID_URL]: Invalid URL: https%3A%2F%2Ftype.fit%2F
```

Copy

Cause: This error message is caused by unnecessarily encoding the Discover/Collect URL.

Recommendation: To fix this error message, remove the encoding function for the URL. You will rarely need to encode URLs. When they do, you should only encode the parts that need it. Otherwise, you will see this error because the entire URL is encoded unnecessarily.

Example Invalid URL message

Error Message: `statusCode: 429...Too many requests`

When running a REST Collector, you may encounter rate-limiting errors if the volume of outgoing requests exceeds the allowed threshold of the target API or Destination.

8.1.1.8. S3 Collector

Cribl Stream supports collecting data from Amazon S3 stores, including optimized support for Splunk Dynamic Data Self Storage (DDSS) datasets. This page covers how to configure the Collector.

For a step-by-step tutorial on using Cribl Stream to replay data from an S3-compatible store, see our Data Collection & Replay sandbox. The sandbox takes about 30 minutes. It provides a hosted environment, with all inputs and outputs preconfigured for you.

Also see our Amazon S3 Better Practices and Using S3 Storage and Replay guides.

How the Collector Pulls Data

When you run an S3 Collector in Discovery mode, the first available Worker returns the list of available files to the Leader Node.

In Full Run mode, the Leader distributes the list of files to process across 1..*N* Workers, as evenly as possible, based on file size. Each Worker then streams the files from the S3 bucket/path to itself.

Compression

Cribl Stream can ingest compressed S3 files if they meet all the following conditions:

- Compressed with the `x-gzip` MIME type.
- End with the `.gz` extension.
- Can be uncompressed using the `zlib.gunzip` algorithm.

The job's **Total size** displayed in the Job Inspector reflects the compressed file size reported by S3. Use this to optimize your path expression for more efficient data access.

Storage Class Compatibility

Cribl Stream does **not** support data preview, collection, or replay from S3 Glacier or S3 Deep Glacier storage classes, whose stated retrieval lags (variously minutes to 48 hours) cannot guarantee data availability when the Collector needs it.

Cribl Stream **does** support data preview, collection, and replay from S3 Glacier Instant Retrieval when you're using the S3 Intelligent-Tiering storage class.

Network Requirements

For S3 Collector operation, ensure outbound TCP port `443` is open from all Worker Nodes to the relevant S3 endpoints. This is required for connectivity in Cribl.Cloud, hybrid, and on-prem deployments.

Avoid SSL inspection or filtering on S3 Collector traffic, as this can disrupt secure connectivity and data collection.

Configure an S3 Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. For example, `Attic42TreasureChest`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description**: Optionally, enter a description.
    - **Auto-populate from**: Select a Destination with which to auto-populate Collector settings. Useful when replaying data.
    - **S3 bucket**: Simple Storage Service bucket from which to collect data.
3. Under **Authentication**, select an **Authentication method** from the dropdown: **Auto** or **Manual**. For details, see Authentication Methods
4. Under the **Assume Role** accordian, configure the following.
    - **Enable Assume Role**: Toggle on to enable Assume Role behavior.
    - **AssumeRole ARN**: Amazon Resource Name (ARN) of the role to assume.
    - **External ID**: External ID to use when assuming role. Enter this if defined in your IAM policy's trust relationship; otherwise, leave blank. (Usage example: AWS Cross-Account Data Collection.)
    - **Duration (seconds)**: Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).
      When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.
5. Next, you can configure the following **Optional Settings**:
    - **Region**: S3 Region from which to retrieve data.

You must grant your IAM role or user the `GetBucketLocation` permission to the S3 bucket when no **Region** is selected. See Amazon S3 Permissions for an example.

- **Path**: Path, within the bucket, from which to collect data. For DDSS, point to the root of the DDSS structure and use **Filters** in the collection job for selection. See About Paths for details.
- **Partitioning scheme**: How data is organized within an S3 bucket. This is important for optimizing data retrieval and storage efficiency, especially when integrating with Cribl Search. Your options are:
  - **Defined in Path**: Use the **Path** structure you specify.
  - **DDSS**: Optimize for datasets extracted using Splunk Dynamic Data Self Storage (DDSS) tool. Cribl Stream assumes you've configured a DDSS root path in your Splunk environment. This root path typically points to the root directory of your self-storage bucket in S3.
- **Path extractors**: Extractors enable you to use template tokens as context for expressions that enrich discovery results. For usage details and examples, see Using Path Extractors.
- **Endpoint**: S3 service endpoint. If empty, Cribl Stream will automatically construct the endpoint from the AWS Region. To access the AWS S3 endpoints, use the path-style URL format. You don't need to specify the bucket name in the URL, because Cribl Stream will automatically add it to the URL path. For details, see AWS' Path-Style Requests topic.
- **Signature version**: Signature version to use for signing S3 requests. Defaults to `v4`.
- **Max batch size (objects)**: Maximum number of metadata objects to batch before recording as results.
- **Recursive**: If toggled on (default), data collection will recurse through subdirectories. This option is available only when **Partitioning scheme** is set to `Defined in Path`.
- **Reuse connections**: Whether to reuse connections between requests. Toggling on (default) can improve performance.
- **Reject unauthorized certificates**: Whether to accept certificates that cannot be verified against a valid Certificate Authority, such as self-signed certificates. Defaults to toggled on.
- **Verify bucket permissions**: Toggle off if you can access files within the bucket, but not the bucket itself. This resolves errors of the form: `discover task initialization failed...error: Forbidden`.
- **Disable time filter**: Toggle on to bypass the collection job event timestamp filtering. By default, Collectors apply time filtering based on the configured time range. Events with timestamps outside this range are discarded. Enabling this setting disables timestamp-based filtering, allowing all returned events to be processed regardless of their timestamps. This is useful when the data source's timestamps do not align with the configured time range, causing unexpected filtering, or when you need to process all data regardless of their timestamps. If you encounter zero results

despite the data source returning events, disabling the time filter can help isolate whether timestamp filtering is the issue. For details, see No Discover or Collector Event Results.

- **Encoding**: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

6. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.
7. Select **Save**, then **Commit & Deploy**.
8. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

The sections described below are spread across several tabs. Click the tab links at left to navigate among tabs.

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

About Paths

Templates and tokens are supported to provide flexibility and automation in defining the directory structure, allowing you to organize data efficiently based on various parameters. This value can be a constant or a JavaScript expression. More on templates and tokens here. If this S3 Collector will be used to round-trip data through our Destination and replay it in Cribl Stream, we have a recommended **Path**. See the Replay Collector for details.

- Templating allows for dynamic insertion of variables like host, year, and month. For example, `/myDir/${host}/${year}/${month}/`).
- Time-based tokens enable time-specific data organization. For example, `myOtherDir/${_time:%Y}/${_time:%m}/${_time:%d}/`.

Path Configuration for DDSS Partitioning

For S3 Collectors using the DDSS partitioning scheme, set the **Path** to the root directory of your DDSS dataset, like `/my_ddss_root/`. Do not specify subdirectories or index names in the **Path**.

The Collector expects the following structure:

```
<root>/
  <index>/
    [db/]  # optional
      db_<latest>_<earliest>_<bucket_id>/
        rawdata/
          journal.(gz|zst|lz4)
```

Copy

When **DDSS** is selected, the Collector recursively scans all subfolders under the configured **Path**, treating each immediate subfolder as an index. For each index, it searches for data buckets using the following logic:

1. Looks for a subfolder named `db/`.
2. If `db/` is present, it traverses this folder to find the data buckets (directories named `db_*`).
3. If `db/` is not present, it looks directly under the index folder for the data buckets (`db_*`).

Therefore, filtering must be done using the collection job **Filter**, not by modifying the **Path**. If you set the **Path** to a specific index subdirectory like `/my_ddss_root/myindex/`, collection will fail or miss data because the Collector's traversal logic starts from the path you provide.

Example DDSS Configuration:

- **Path**: `/my_ddss_root/`
- **Partitioning scheme**: `DDSS`
- **Filter** (in the Run modal): `index == 'myindex'`

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

- **Enabled**: Defaults to toggled off. Toggle on to enable the custom command.
- **Command**: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).
- **Arguments**: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

- **Event Breaker rulesets**: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.
- **Event Breaker buffer timeout**: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, etc. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. This keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields** : List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

Authentication Methods

The **Authentication method** dropdown give you a choice of two options.

The **Auto** option (default) will use environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, or the attached IAM role. Will work only when running on AWS.

The **Manual** option exposes these fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials. This value can be a constant or a JavaScript expression.
- **Secret key**: Enter your AWS secret key. if not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials. Optional when running on AWS. This value can be a constant or a JavaScript expression.

The **Secret** option swaps in this drop-down:

- **Secret key pair**: Select a secret key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

The **Manual** option's **Access key** and **Secret key** values override the **Auto** option. For any Workers that you want to associate with an AWS IAM role, using the **Assume Role** controls below ensure that the

Manual option's key fields are empty.

Using Path Extractors

Click **Add Extractor** to add each extractor as a key-value pair, mapping a **Token** name on the left (a substring of the **Path**) to a custom JavaScript **Extractor expression** on the right. Example `{host: value.toLowerCase()}`

Each expression accesses its corresponding token through its `value` variable, and evaluates the token to populate event fields.

You can define each token in only one **Extractor expression** per S3 Collector config.

Example #1

Here is a simple, complete example, using a single expression:

Token	Extractor Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`{program: value.split('.')[0]}`	`/var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Example #2

Here is a multi-expression example. Assuming that your S3 bucket path supports the Collector's configured **Path**, this shows how to extract each segment of this example **Path**, and transform it into a reformatted field:

First, the path:

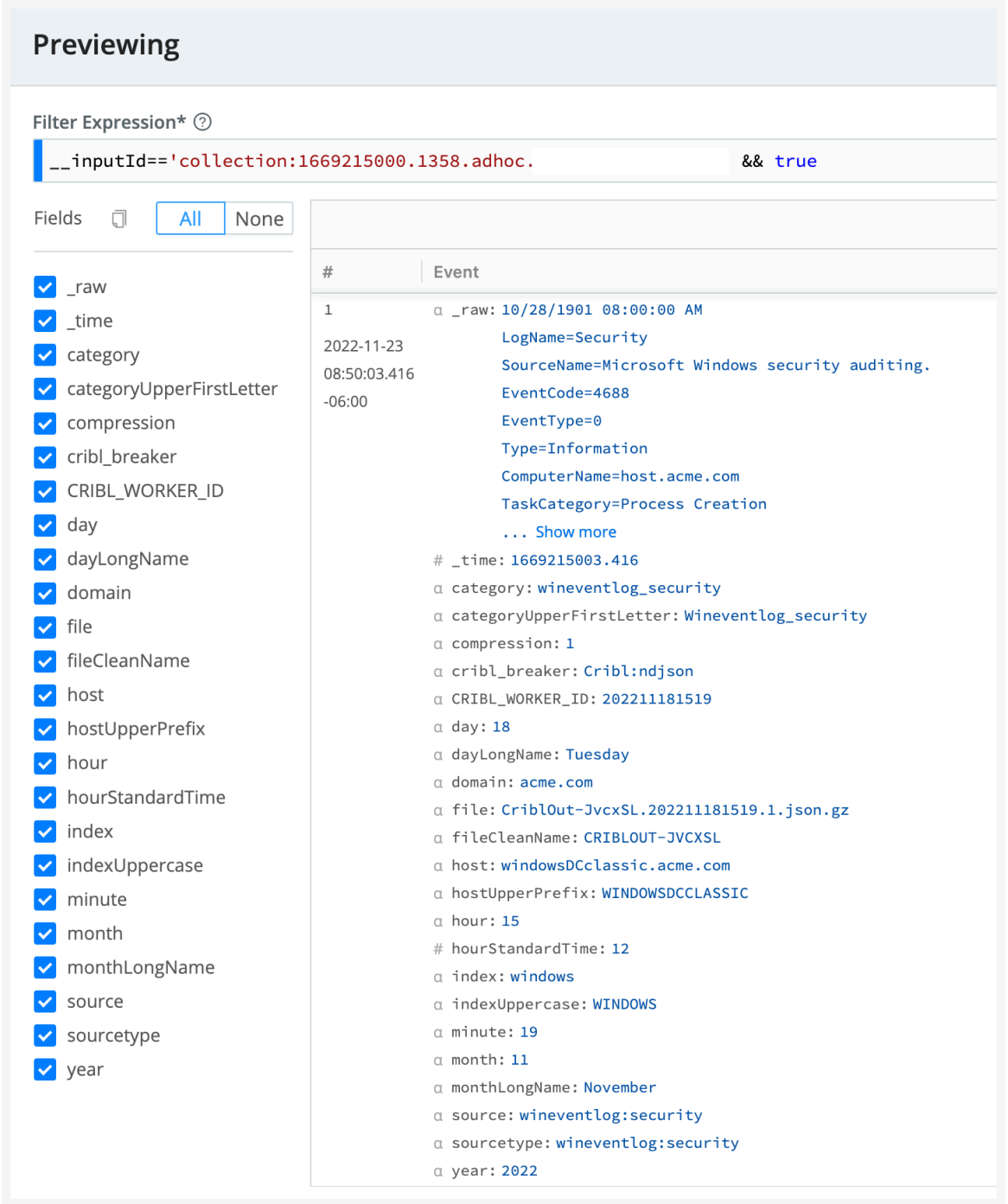
```
/${index}/${_time:%Y}/${_time:%m}/${_time:%d}/${host}/${category}/${_time:%H}/${file}
```

Next, the tokens and extractor expressions:

Token	Extractor Expression	Purpose
`index`	`{"indexUppercase" : value.toUpperCase()}`	Uppercase th `index`
`host`	`{"hostUpperPrefix" : value.toUpperCase().replace(/(.+?)\..+/,'$1'), "domain" : value.match(/.+?\.(.+)/)[1]}`	Use one toker two fields: `hostUpperP` and `domain`

Token	Extractor Expression	Purpose
`category`	`{"categoryUpperFirstLetter" : `${value.substring(0,1).toUpperCase()}${value.substring(1)}`}`	Uppercase th `category` str first letter
`file`	`{"fileCleanName" : value.replace(/ARCHIVE-/i,'').toUpperCase().split('.')[0], "CRIBL_WORKER_ID" : value.split('.')[1], "compression" : value.split('.')[2]}`	Split the objec name) into th fields

Here is a corresponding example event, as collected/replayed from S3. The metadata fields at the bottom show the results of the path extractor expressions:



Event with path extractors' transformations

S3 Permissions

The following permissions are required on the S3 bucket:

- `s3:GetObject`
- `s3:ListBucket`
- `s3:GetBucketLocation` - Needed if **Region** is not configured.

Temporary Access via SSO Provider

You can use Okta or SAML to provide access to S3 buckets using temporary security credentials.

Troubleshooting

When permissions are correct on the object store, and events are reaching the Collector, the Preview pane will show events and the Job Inspector will show an **Events collected** count.

However, if previewing returns no events and throws no error, first check your **Filter** expression by previewing without it by simplifying the **Filter** expression to `true`. Then check the Job Inspector: If the **Total size** is greater than `0`, and the **Received size** is `NA` or `0`, make sure you have list and read permissions on the object store.

If you're still not seeing events, try setting your S3 Collector's **Path expression** to a `/`.

Encrypted Data

To collect encrypted data, such as CloudTrail logs, you'll need to add access to the KMS resources in your KMS Key Policy. Add the IAM user or role to the **Principal** section of your Key Policy and provide the `kms:Decrypt` **Action**. See Amazon's documentation on Key Policies for more information, including how to view and change your policy.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
    - `collectorType`: Indicates the type of Collector used for the job.
    - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
    - Ad hoc jobs are formatted as `collection:<timestamp>_<randomId>_adhoc_<Collector`

8.1.1.9. Script Collector

Cribl Stream supports flexible data collection configured by your custom scripts.

In Cribl.Cloud, the Script Collector is only available on customer-managed hybrid Worker Nodes.

How the Collector Pulls Data

When you run a Script Collector in Discovery mode, the first available Worker returns one line of data per discovered item. Each item (line) turns into a Collection task on the Leader Node.

In Full Run mode, the Leader passes the item to collect into the script in the `$CRIBL_COLLECT_ARG` variable, and spreads collection across all available Workers.

If you run the Script Collector with a given Time Range, the `earliest` and `latest` values will be available to the script via the environment variables `EARLIEST` and `LATEST`, respectively.

Important: Secure and Use the Script Collector Properly

Be aware that scripts will allow you to execute almost anything on the system where Cribl Stream is running. Make sure you understand the impact of what you're executing before you do so. These scripts run with the same level of permissions as the user running Cribl Stream. So, if you are running it as `root`, these commands will run with root user permissions. If your system requires greater security, consider disabling the Script Collector. See How to Disable the Script Collector for more information.

Also, always validate and test your scripts before running them on your system. Scripts must finish on their own. The Collector will not stop a script once its task has started. You must ensure scripts do not use infinite loops.

Configure a Script Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. E.g., `sh2GetStuff`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Discover script**: Script to discover which objects/files to collect. This script should output one task per line in `stdout`. Discovery is especially important in a distributed deployment, where the

Leader must track all tasks, and must guarantee that each is run by a single Worker. See Examples below.

- **Collect script**: Script to perform data collections. Pass in tasks from the Discover script as `$CRIBL_COLLECT_ARG`. Should output results to `stdout`.

3. Next, you can configure the following **Optional Settings**:
   - **Shell**: Shell in which to execute scripts. Defaults to `/bin/bash`.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.
6. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

The sections described below are spread across several tabs. Click the tab links at left to navigate among tabs.

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

- **Enabled**: Defaults to toggled off. Toggle on to enable the custom command.
- **Command**: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).
- **Arguments**: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

Event Breaker rulesets: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, etc. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. This keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.
- **Remove Discover fields** : List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).
- **Resume job on boot**: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

How the Collector Pulls Data

In the Discover phase, the first available Worker returns one line of data per item discovered. Each item line turns into a Collect task on the Leader Node. In the Collect phase, the items to collect are passed into the script as the variable `$CRIBL_COLLECT_ARG`, and are spread across all available Workers.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
  - `collectorType`: Indicates the type of Collector used for the job.
  - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
  - Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
    - `<timestamp>`: Timestamp when the job was initiated (in Unix time).
    - `<randomId>`: A random identifier to ensure uniqueness.
    - `adhoc`: Indicates the job was manually triggered.
    - `<Collector ID>`: The ID of the Collector.

- Scheduled jobs are formatted as `collection:<Collector ID>`.
  - `<Collector ID>`: The ID of the Collector.

Examples

Telemetry Collector

You could define this Collector to check for Cribl Stream telemetry errors, which could cause license validation to fail, eventually (after a delay) blocking data input.

Collector type: `Script`

Discover script: `ls $CRIBL_HOME/log/cribl*`

Collect script: `grep 'Failed to send anonymized telemetry metadata' $CRIBL_COLLECT_ARG`

S3 Collector

In this example, the Discover script retrieves file names from a specified Amazon S3 bucket, and then writes them (one per line) to the standard output. The Collect script processes each line as its `$CRIBL_COLLECT_ARG`, and uses `zcat` to decompress the buckets' data.

Collector type: `Script`

Discover script:
```
aws s3api list-objects --bucket <bucket-name> --prefix <subfolder>/ --query
'Contents[].Key' --output text
```

Collect script: `aws s3 cp s3://<bucket-name>/$CRIBL_COLLECT_ARG - | zcat -f`

Simple Collector

This example essentially spoofs the Discover script with an `echo` command, which simply announces what the Collect script (itself simple) will do.

Collector type: `Script`

Discover script: `echo "speedtest"`

Collect script: `speedtest --json`

How to Disable the Script Collector

You can disable the Script Collector if your organization's security policy prevents you from executing deliberate code on your remote nodes. Disabling the Script Collector is done using the Cribl command-line interface (CLI) to prevent potential bad actors from enabling the Script Collector through the API. See Secure and Use the Script Collector Properly for more information about the potential security risks.

Disable the Script Collector with caution. Disabling the Script Collector is not easily reversible. Restoring the Script Collector after disabling it requires a full re-installation of Cribl Stream.

To fully disable the Script Collector in your entire instance of Cribl Stream:

1. Inside the Cribl CLI, navigate to the root directory of your Cribl instance.
2. Run the following command:

```
opt/cribl/bin/cribl limits -c 0
```

Copy

If your deployment is configured to use failover and relies on environment variables for that configuration, ensure you also include the following flag: `-d ABSOLUTE/FAILOVER/PATH`.

3. Restart Cribl Stream.
4. Inside Cribl Stream, verify that you have disabled the Script Collector. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Verify that the Script Collector does not show up in the list of Sources.
5. Commit and deploy your changes to disable the Script Collector for all Worker Nodes.

If you create future Groups after running this procedure, they will not include the Script Collector.

8.1.1.10. Splunk Search Collector

Cribl Stream supports collecting search results from Splunk queries. The queries can be both simple and complex, as well as real-time searches. This page covers how to configure the Collector.

Splunk Cloud Prerequisite: IP Allowlist

To enable Splunk Cloud to accept connections from your Cribl Workers, you must allowlist their egress IP addresses as described in the Splunk IP allow-list documentation. For Cribl.Cloud, egress IPs are group-specific and dynamic. Cribl might update them during infrastructure changes, so you must monitor and update your allowlist accordingly. For on-prem and hybrid deployments, you control your network and can assign static egress IPs, allowing you to configure the allowlist once unless your network changes.

Configure a Splunk Search Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Choose the Source and select **Add Collector**.
2. In the **New Collector** modal, configure the following under **Collector Settings**:
    - **Collector ID**: Unique ID for this Collector. For example, `splunk2search`. If you clone this Collector, Cribl Stream will add `-CLONE` to the original **Collector ID**.
    - **Description**: Optionally, enter a description.
    - **Search endpoint**: Rest API used to conduct a search. Defaults to `/services/search/v2/jobs/export`.
    - **Output mode**: Format of the returned output: `JSON` or `CSV`. Defaults to `JSON` format.
3. Under **Search**, configure the following.
    - **Search**: Enter the Splunk query. For example: `index=myAppLogs level=error channel=myApp` OR `| mstats avg(myStat) as myStat WHERE index=myStatsIndex`. The optional **Preserve escaped quotes** setting maintains escaped quotes.
    - **Search head**: Enter the search head base URL. The default is `https://localhost:8089`.
    - **Earliest**: You can enter the earliest time boundary for the search. This can be an exact or relative time. For example: `2022-01-14T12:00:00Z` or `-16m@m`.
    - **Latest**: You can enter the latest time boundary for the search. This can be an exact or relative time. For example: `2022-01-14T12:00:00Z` or `-16m@m`.
4. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **None**: Don't use authentication. Compatible with REST servers like AWS, where you embed a secret directly in the request URL.

- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret.
- **Bearer token**: Provide the `token` value configured and generated in Splunk.
- **Bearer token (text secret)**: Provide the Bearer token referenced by a secret. Select a stored text secret in the resulting **Token (text secret)** drop-down, or select **Create** to configure a new secret.

> You can also use **Extra Headers** as an alternative way of authenticating a request. Select `None` in the **Authentication** drop-down and use the `C.Secret` expression method to provide the value for the `Authentication` header. For example:
>
> ```
> `Bearer: ${C.Secret('<secretId>', 'text').value}`
> ```

5. Under **Retries** configure the following.
   - **Retry type**: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.
   - **Initial retry interval (ms)**: Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.
   - **Retry limit**: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.
   - **Backoff multiplier**: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
   - **Retry HTTP codes**: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.
   - **Honor Retry-After header**: If toggled on (default) and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. For details, see Honor Retry-After Header.
   - **Retry connection timeout**: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.
   - **Retry connection reset**: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.
6. Next, you can configure the following **Optional Settings**:
   - **Extra parameters**: Optional HTTP request parameters to append to the request URL. For details, see Extra Parameters below.

- **Extra headers**: Select **Add Header** to (optionally) add collection request headers as key-value pairs. For details, see Extra Headers below.
- In the **Request Timeout (secs)** field, you can set a maximum time period (in seconds) for an HTTP request to complete before Cribl Stream treats it as timed out. Defaults to `0`, which disables timeout metering. When running a real-time search you must update the **Request Timeout Parameter** to avoid having the Collector stuck in a forever running state. Updating the **Request Timeout Parameter** stops the search after the allocated period of time.
- **Round-robin DNS**: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.
- **Disable time filter**: Toggle on to bypass the collection job event timestamp filtering. By default, Collectors apply time filtering based on the configured time range. Events with timestamps outside this range are discarded. Enabling this setting disables timestamp-based filtering, allowing all returned events to be processed regardless of their timestamps. This is useful when the data source's timestamps do not align with the configured time range, causing unexpected filtering, or when you need to process all data regardless of their timestamps. If you encounter zero results despite the data source returning events, disabling the time filter can help isolate whether timestamp filtering is the issue. For details, see No Discover or Collector Event Results.
- **Reject unauthorized certificates**: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.
- **Preserve escaped characters**: Toggle on to preserve backslash-escaped characters in **Search** queries. This ensures complex search strings such as `| search "\"foo\" \"bar\""` execute as intended in Splunk by preventing the loss or alteration of escape sequences during processing. Defaults to toggled on.
- **Encoding**: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's **Manage Sources** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

7. Optionally, configure any Result, Result Routing, and Advanced settings outlined in the sections below.

Cribl offers a **Splunk Search Ruleset** Event Breaker that is designed for data from Splunk queries.

8. Select **Save**, then **Commit & Deploy**.
9. To verify that the Collector actually collects data, you can start a single run in the Preview mode.

The sections described below are spread across several tabs. Select the tab links at left to navigate among tabs.

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

Honor Retry-After Header

Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds. - Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms). - When toggled off, Cribl Stream ignores all `retry-after` headers.

Extra Parameters

The extra parameters refine or narrow the request. Select **Add Parameter** to add parameters as key-value pairs:

- **Field Name**: Unique name for the field you're adding.
- **Value**: JavaScript expression to compute the field's value (can be a constant).

Extra Headers

By adding the appropriate **Extra headers**, you can specify authentication based on API keys using secrets as an alternative to the **Authentication** options above. The key-value pairs are:

- **Name**: Header name.
- **Value**: JavaScript expression to compute the header's value (can be a constant).

Result Settings

The Result Settings determine how Cribl Stream transforms and routes the collected data.

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

Enabled: Defaults to toggled off. Toggle on to enable the custom command.

Command: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).

Arguments: Select **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

Event Breaker rulesets: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`. Cribl offers a **Splunk Search Ruleset** designed for data from Splunk queries.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Result Routing

Send to Routes: Toggle on (default) if you want Cribl Stream to send events to normal routing and event processing. Toggle off to select a specific Pipeline/Destination combination. Toggling off exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

Toggling on (default) exposes this field:

- **Pre-processing Pipeline**: Pipeline to process results before sending to Routes. Optional.

This field is always exposed:

- **Throttling**: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, and so on. (Example: `42 MB`.) Default value of `0` indicates no throttling.

You might toggle **Send to Routes** off when configuring a Collector that will connect data from a specific Source to a specific Pipeline and Destination. One use case might be a Splunk Search Collector that gathers a known, simple type of data. This approach keeps the Collector's configuration self-contained and separate from Cribl Stream's routing table for live data - potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional, and available only when **Send to Routes** is toggled on.

Throttling: Rate (in bytes per second) to throttle while writing to an output. Also takes values with multiple-byte units, such as `KB`, `MB`, `GB`, and so on. (Example: `42 MB`.) Default value of `0` indicates no throttling.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Time to live: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

Remove Discover fields: List of fields to remove from the Discover results. This is useful when discovery returns sensitive fields that should not be exposed in the Jobs user interface. You can specify wildcards (such as `aws*`).

Resume job on boot: Toggle on to resume ad hoc collection jobs if Cribl Stream restarts during the jobs' execution.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Collector:

- `__collectible` - This object's nested fields contain metadata about each collection job.
  - `collectorType`: Indicates the type of Collector used for the job.

    - `collectorId`: Represents the **Collector ID** of the Collector, as configured during setup.
- `__inputId` - Uniquely identifies the origin of data for a collection job. Its format varies depending on whether the job is ad hoc or scheduled:
    - Ad hoc jobs are formatted as `collection:<timestamp>.<randomId>.adhoc.<Collector ID>`.
        - `<timestamp>`: Timestamp when the job was initiated (in Unix time).
        - `<randomId>`: A random identifier to ensure uniqueness.
        - `adhoc`: Indicates the job was manually triggered.
        - `<Collector ID>`: The ID of the Collector.
    - Scheduled jobs are formatted as `collection:<Collector ID>`.
        - `<Collector ID>`: The ID of the Collector.

How Cribl Stream Pulls Data

This Collector will gather data from the specified **Search head** URL. If you enable scheduled collection, searches will repeat on the interval specified in the **Schedule** modal's **Cron schedule** field.

A single Worker executes each collection job. If the Leader goes down, search jobs in progress will complete, but future scheduled searches will not run until the Leader relaunches.

Mitigating Stuck-Job Problems

Occasionally, a scheduled job fails, but continues running for hours or even days, until someone intervenes and cancels it. If left alone, such a "stuck", "orphaned," or "zombie" job will never complete. This can cause missing events in downstream receivers, along with `HTTP timeout` or similar errors in Cribl Stream's logs.

To keep stuck jobs from running excessively long:

- First, try setting **Optional Settings** > **Request timeout (secs)** - whose default of `0` means "wait forever" - to a desired maximum duration.
- If adjusting **Timeout (secs)** does not fix the problem, try the global setting **Job Timeout** - whose default of `0` allows a job to run indefinitely - to a desired maximum duration. You'll find **Job Timeout** among the task manifest and buffering limits.

Using these settings in tandem works like this:

- **Timeout (secs)** limits the time that Cribl Stream will wait for an HTTP request to complete.
- Then, if a job gets stuck and keeps running beyond that limit, **Job Timeout** can catch and terminate the job, because it monitors the overall time the job has been running.

8.1.1.11. Manage Collector State In Database And REST API Collectors

Manage State in Database Collector

For the Database Collector, **state** identifies the last row that the Collector has read from the database, as expressed by that row's value in the tracking column. You can view, edit, refresh, or delete that state - and that is what we mean by managing state. Testing or debugging a Collector is the main reason to do this.

Manage State in REST Collector

The REST Collector tracks **state** according to configuration of the expressions **State update** and **State merge**. With state tracking enabled, these expressions default to track state by the greatest event `_time` value.

You can enable state tracking in the REST Collector in **State Tracking** when you schedule or perform a full run.

Manage State

The **Manage State** button is enabled whenever your Collector state is not empty, and is available from two locations:

- In the **Database** list view of your Database Collectors, at the end of each row. Select **Manage**, **Data**, **Manage Sources**, **Collectors**, **Database** to reach this UI.
- In the configuration modal for each Collector.

To view state, select **Manage State**. This opens the **Manage State** JSON editor, since state is recorded as a JSON object. For example, if you have chosen the `id` column in your database as your tracking column, and your Collector has just read 5 rows, the state object will look like this:

```
{
  "id": {
     "value": 5
  }
}
```

Copy

Suppose a particular job did not read the data you expected it to. Viewing state is the most basic troubleshooting step in this kind of situation.

If you know that state was updated because the Collector has run, but the state shown in the editor does not reflect the change you expected, select **Refresh State**.

If you want to start again with no state, select **Delete State**. To learn how and why to use this operation, see Deleting Collector State, below.

Edit Collector State

In the **Manage State** JSON editor, you can change state by editing the JSON, and then select **Save State** to persist the change.

To continue our example, if your Collector has read 1000 rows but you want it to repeat the last 500 rows on the next scheduled run, you would change the state from `1000` to `500`.

Delete Collector State

The **Delete State** and **Delete Collector** buttons are independent of each other. Deleting the Collector does **not** delete its saved state information. Cribl Stream decouples these two operations to provide benefits that we'll explore in this section.

To begin with, note that Collector state is stored on disk, not in Git. Each Collector state is stored separately.

Secondly, the uses of state-deleting differ depending on whether your Database Collector is on a single-instance deployment of Cribl Stream, or a distributed deployment.

Delete Collector State in a Single-Instance Deployment

When you first configure a Database Collector, you might need to experiment with SQL queries until you arrive at one that returns the desired results. In doing this, you might need the queries you're testing to always start from the same place in the database. If you're using a tracking column, this won't work, because each query will move you further along in the database. The solution is to delete state between iterations of the query you're building.

Delete Collector State in a Distributed Deployment

Just like in single-instance deployments, in distributed deployments it can be useful to delete Collector state when you're testing SQL queries for a Collector with state tracking enabled.

Beyond that, with distributed deployments it's important to know that Cribl Stream decouples the deleting of Collectors themselves from the deleting of their saved state, and to see the benefits of that decoupling.

Suppose you have been running a Collector with state tracking enabled, and it's working its way through a database. At some point you decide you're not happy with that Collector and you select the **Delete Collector** button. What happens next? There are at least three possibilities:

- If you're sure you want to delete the Collector, you should Commit and Deploy. Once you do that, the Leader pushes the change out to the Worker Nodes, and the Collector is gone (although you can recover it by rolling back to a previous commit in Git).
- If you realize that you made a mistake, and want to keep the Collector after all, then you should **not** Commit and Deploy. Instead, select the **Undo All Changes** button in the **Git Changes** modal - this un-deletes the Collector. And, since the Collector state remains on disk, the Collector will pick up where it left off.
- If you deleted the Collector because you want to create another Collector with the same ID and the same tracking column, but a different configuration, you should Commit and Deploy, then create your new Collector. Because the old Collector state remains on disk, the new Collector with the same ID picks up the saved state. When you begin using the new Collector, it will pick up where the old Collector left off. On the other hand, if you prefer the new Collector to start from zero, just select the **Delete State** button before you start using it.

In short, Cribl Stream decoupling of Collector deletion from state deletion gives you control, such that you can avert messy situations where your Collector cannot find the state it needs.

8.1.1.12. Schedule And Run Collector Jobs

Once you've configured a Collector, you can run it in one of two ways:

- **Ad hoc**: Execute a one-time collection job immediately. This is useful for testing configurations, previewing data, or replaying a specific dataset.
- **Scheduled**: Configure a job to run on a recurring interval, effectively turning a static data source into a continuous stream.

Before running a job, it's important to understand when your configuration edits are applied. Certain Collectors update automatically without a **Commit & Deploy**, while others require it. See Collector Changes for details.

Schedule Configuration

Select **Schedule** beside a configured Collector to display the **Schedule Collector** modal. This provides the following controls.

Enabled: Toggle on to enable this collection schedule.

The scheduled job will keep running on this schedule forever, unless you toggle **Enabled** off, which preserves the schedule's configuration but prevents its execution.

Cron schedule: A cron schedule on which to run this job.

- The **Estimated schedule** below this field shows the next few collection runs, as examples of the cron interval you've scheduled.

Concurrent run limit: Sets the maximum number of instances of this scheduled job that Cribl Stream will simultaneously run.

Skippable: Skippable jobs can be delayed up to their next run time if the system is hitting concurrency limits. Defaults to toggled on. Toggling off displays this additional option:

- **Resume missed runs**: Default is toggled off. When toggled on, if Cribl Stream's Leader (or single instance) restarts, it will run all missed jobs according to their original schedules.

Skippable Jobs and Concurrency Limits

If toggled on, the **Skippable** option obeys these concurrency limits configured separately in **Settings** > **Global** > **General Settings** > **Job Limits**:

- **Concurrent job limit**

- **Concurrent scheduled job limit**

See Job Limits for details on these and other limits that you can set in global **Settings**.

When the above limits delay a Skippable job:

- The Skippable job will be granted slightly higher priority than non-Skippable jobs.
- If the job receives resources to run before its next scheduled run, Cribl Stream will run the delayed job, then snap back to the original cron schedule.
- If resources do **not** free up before the next scheduled run: Cribl Stream will **skip** the delayed run, and snap back to the original cron schedule.

Toggle **Skippable** off if you absolutely must have all your data, for compliance or other reasons. In this case, Cribl Stream will build up a backlog of jobs to run.

Toggling **Skippable** off results in behavior more like the TCP protocol, and toggling **Skippable** on results in behavior more like UDP.

All collection jobs are constrained by the following options in **Settings** > **Global** > **General Settings** > **Job Limits**:

- **Concurrent task limit**
- **Task usage percentage limit**

Run Configuration and Shared Settings

Most of the remaining fields and options below are shared with the **Run Collector** modal, which you can open by selecting **Run** beside a configured Collector.

Mode

Depending on your requirements, you can schedule or run a Collector in these modes:

- Preview - Default for Run, but not offered for scheduled jobs.
- Discovery - Default for scheduled jobs.
- Full Run.

Preview

In the Preview mode, a collection job will return only **a sample subset** of matching results (for example, 100 events). This is useful in cases when users need a data sample to:

- Ensure that the correct data comes in.
- Iterate on Filter expressions.
- Capture a sample to iterate on Pipelines.

Schedule configuration omits the Preview option, because Preview is designed for immediate analysis and decision making. To configure a scheduled job with high confidence, you can first manually run Preview jobs with the same Collector, to verify that you're collecting the data you expect.

Preview Settings

In Preview mode, you can optionally configure these options:

Capture time (sec): Maximum time interval (in seconds) to collect data.

Capture up to N events: Maximum number of events to capture.

Where to capture: Select the stage in the event processing order where events should be captured during the Preview run. During a Preview run, the first two of the four capture options are available.

- `Before pre-processing Pipeline` (Default): Captures events immediately after they're received.
- `Before the Routes`: Captures events after pre-processing but before they're routed.

Discovery

In Discovery mode, a collection job will return only **the list of objects/files** to be collected, but none of the data. This mode is typically used to ensure that the Filter expression and time range are correct before a Full Run job collects unintended data.

Send to Routes

In Discovery mode, this toggle enables you to send discovery results to Cribl Stream Routes. Default is toggled off.

This setting overrides the Collector configuration's **Result Routing** > **Send to Routes** setting.

Full Run

In Full Run mode, the collection job is fully executed by Worker Nodes, and will return all data matching the Run configuration. Optionally, for the Database Collector, you can also configure state tracking.

Time Range

Set an **Absolute** or **Relative** time range for data collection. The **Relative** option is the default, and is particularly useful for configuring scheduled jobs.

In Discovery and Full Run modes, time range filters are applied before events enter the pre-processing Pipeline.

For Preview mode, while the default is to filter before pre-processing, you can select alternative capture points with the **Where to capture** setting.

Absolute

Select **Absolute** to set fixed collection boundaries. Use the **Earliest** and **Latest** controls to set the start date/time and end date/time. Provide the correct time zone in the **Range timezone** drop-down.

Relative

Select **Relative** to set dynamic collection boundaries. For ad hoc runs, these are relative to the current Unix time. For scheduled jobs, boundaries are calculated based on the job's scheduled start time, not the current time, ensuring the collection window remains consistent even if the job is delayed. Next, use the **Earliest** and **Latest** fields to set start and end times like these:

- **Earliest** example values: `-1h, -42m, -42m@h`
- **Latest** example values: `now, -20m, +42m@h`

By default, Cribl Stream uses **Earliest** and **Latest** as time-based tokens, which it evaluates in two ways:

- By comparison to time values specified in the directory path.
- By comparison to the `_time` field.

The Google Cloud Storage, REST/API, and Splunk Search Collectors provide a **Disable time filter** option in their **Collector Settings**. When toggled on, scheduling or running these Collectors against a date range will cause Cribl Stream to ignore event time-based filtering entirely.

Relative Time Syntax

For Relative times, the **Earliest** and **Latest** controls accept the following syntax:

`[+|-]<time_integer><time_unit>@<snap-to_time_unit>`

To break down this syntax:

Syntax Element	Values Supported
Offset	Specify: `-` for times in the past, `+` for times in the future, or omit with `now`.
<time_integer>	Specify any integer, or omit with `now`.

Syntax Element	Values Supported
<time_unit>	Specify the `now` constant, or one of the following abbreviations: `s[econds]`, `m[inutes]`, `h[ours]`, `d[ays]`, `w[eeks]`, `mon[ths]`, `q[uarters]`, `y[ears]`.
@<snap-to_time_unit>	Optionally, you can append the `@` modifier, followed by any of the above `<time_unit>`s, to round down to the nearest instance of that unit. (See the next section for details.)

Cribl Stream validates relative time values using these rules:

- **Earliest** must not be later than **Latest**.
- Values without units get interpreted as seconds. For example, `-1 = -1s`.

Snap-to-Time Syntax

The `@` snap modifier always rounds **down** (backwards) from any specified time. This is true even in relative time expressions with `+` (future) offsets. For example:

- `@d` snaps back to the beginning of today, 12:00 AM (midnight).
- `+128m@h` looks forward 128 minutes, then snaps back to the nearest round hour. (If you specified this in the `Latest` field, and ran the Collector at 4:20 PM, collection would end at 6:00 PM. The expression would look forward to 6:28 PM, but snap back to 6:00 PM.)

Other options:

- `@w` or `@w7` to snap back to the beginning of the week - defined here as the preceding Sunday.
- To snap back to other days of a week, use `w1` (Monday) through `w6` (Saturday).
- `@mon` to snap back to the 1st of a month.
- `@q` to snap back to the beginning of the most recent quarter - Jan. 1, Apr. 1, Jul. 1, or Oct. 1.
- `@y` to snap back to Jan. 1.

Filter

This is a JavaScript filter expression that is evaluated against token values in the provided Collector path (see below), and against the events being collected. The **Filter** value defaults to `true`, which matches all data, but this value can be customized almost arbitrarily.

For example, if a File System or S3 Collector is run with this Filter:

```
host=='myHost' && source.endsWith('.log') || source.endsWith('.txt')
```

...then only files/objects with `.log` or `.txt` extensions will be fetched. And, from those, only those events with host field `myHost` will be collected.

At the **Filter** field's right edge are a Copy button, an Expand button to open a validation modal, and a History button. For more extensive options, see Tokens for Filtering below.

Advanced Settings

Log level: Level at which to set task logging. More-verbose levels are useful for troubleshooting jobs and tasks, but use them sparingly.

Lower task bundle size: Limits the bundle size for small tasks. For example, bundle five 200KB files into one 1MB task bundle. Defaults to `1MB`.

Upper task bundle size: Limits the bundle size for files above the **Lower task bundle size**. For example, bundle five 2MB files into one 10MB task bundle. Files greater than this size will be assigned to individual tasks. Defaults to `10MB`.

Reschedule tasks: Whether to automatically reschedule tasks that failed with non-fatal errors. Defaults to toggled on; does not apply to fatal errors.

Task reschedule limit: Maximum number of times a task can be rescheduled. Defaults to `1`.

Job timeout: Maximum time this job will be allowed to run. Units are seconds, if not specified. Sample values: `30`, `45s`, or `15m`. Minimum granularity is 10 seconds, so a `45s` value would round up to a 50-second timeout. Defaults to `60m`. Enter '0' for unlimited time (no timeout).

State Tracking

Enabled: Toggle on to enable state tracking when running scheduled or full run jobs.

Subsequent options depend on the type of Collector you're working with.

For Database Collectors, see Working with a Tracking Column to learn how to craft a Collector query for state tracking.

For REST Collectors, see Working with State Tracking to learn how enable and use state in your collection configuration.

State tracking is available only for Database and REST Collectors.

Cribl recommends that when you enable state tracking on a Collector, you configure the schedule to allow only one job to run at a time.

> This prevents one run from starting before another has finished, which avoids the collection of duplicate records and/or incorrect state tracking.

Tokens for Filtering

Let's look at the options for path-based (basic) and time-based token filtering.

Basic Tokens

In collectors with paths, such as Filesystem or S3, Cribl Stream supports path filtering via token notation. Basic tokens' syntax follows that of JS template literals: `${<token_name>}` - where `token_name` is the field (name) of interest.

For example, if the path was set to `/var/log/${hostname}/${sourcetype}/`, you could use a Filter such as `hostname=='myHost' && sourcetype=='mySourcetype'` to collect data only from the `/var/log/myHost/mySourcetype/` subdirectory.

Time-based Tokens

In paths with time partitions, Cribl Stream supports further filtering via time-based tokens. This has a direct effect with earliest and latest boundaries. When a job runs against a path with time partitions, the job traverses a minimal superset of the required directories to satisfy the time range, before subsequent event `_time` filtering.

About Partitions and Tokens

Cribl Stream processes time-based tokens as follows:

- For each path, time partitions must be notated in descending order. So Year/Month/Day order is supported, but Day/Month/Year is not.
- Paths may contain more than one partition. For example, `/my/path/2020-04/20/`.
- In a given path, each time component can be used only once. So `/my/path/${_time:%Y}/${_time:%m}/${_time:%d}/...` is a valid expression format, but `/my/path/${_time:%Y}/${_time:%m}/${host}/${_time:%Y}/...` (with a repeated `Y`) is not supported.
- For each path, all extracted dates/times are considered in UTC.

The following strptime format components are allowed:

- `'Y'`, `'y'`, for years
- `'m'`, `'B'`, `'b'`, `'e'`, for months
- `'d'`, `'j'`, for days

- `'H'`, `'I'`, for hours
- `'M'`, for minutes
- `'S'`, for seconds

Token Syntax

Time-based token syntax follows that of a slightly modified JS template literal: `${_time:<some_strptime_format_component>}`. Examples:

Filter	Matches
`/my/path/${_time:%Y}/${_time:%m}/${_time:%d}/...`	`/my/path/2020/04/20`
`/my/path/${_time:year=%Y}/${_time:month=%m}/${_time:day=%d}/...`	`/my/path/year=2020/`
`/my/path/${_time:%Y-%m-%d}/...`	`/my/path/2020-05-20`

Autoscale with Collection Jobs

Cribl Stream supports autoscaling for streaming data, but not for collection jobs.

If you run a collection job using Workers in an autoscaling group (such as an AWS instance), you must adequately size your Worker Group or Worker Groups and provision the Worker Nodes before you run or schedule a collection job. Otherwise, a race condition may occur between task scheduling and configuration download from the Leader.

How Time Filtering Works for Scheduled Jobs

- In **Discovery** and **Full Run** modes, time range filters apply before events enter the pre-processing Pipeline.
- In **Preview** mode, filtering defaults to pre-processing, but you can adjust this using the **Where to capture** setting.

Arrays

If a Collector returns events as an array, make sure you configure your **Event Breaker** correctly. For JSON arrays, you'll need a **JSON array Event Breaker** to separate individual events for proper filtering. You can see examples of this with the REST Collector in Parse Discover Returns With Arrays as JSON Objects.

REST and Database Collectors

REST and Database Collectors support the `${earliest}` and `${latest}` variables. These variables are automatically populated with the Unix time (in seconds) that represents the lower and upper bounds of the time range you specify when running or scheduling a job.

- **REST Collector**: Use these variables in request URLs, parameters, and headers. See REST Collector - Time Range Variables for more information.
- **Database Collector**: Use these variables directly in your SQL queries. See an SQL query example.

Important Considerations:

- For millisecond resolution, multiply these variables by 1,000.
- To get human-readable timestamps, use a function like `C.Time.strftime()`. Otherwise, you'll just get a number.
- Reference the API documentation to match the required time format.

Other Collectors

Other Collectors do not support `${earliest}` and `${latest}` variables. They use time-based tokens like `${_time:%Y}` in the **Path** or environment variables.

Job Interruption and Recovery

The behavior of a collection job during a Cribl Stream shutdown depends on how the job was started and how the shutdown was initiated.

Explicit Restart or Stop: If you explicitly restart or stop Cribl Stream, this will cancel any currently running jobs. This applies to executing the `./cribl restart` or `./cribl stop` CLI commands, as well as to selecting **Settings** > **Global** > **Controls** > **Restart**.

Unintended Shutdowns:

- **Ad hoc** jobs can be configured to automatically resume if they were interrupted by an unexpected shutdown.
- **Scheduled** jobs will **not** resume automatically after any type of shutdown. They will wait for their next scheduled run time.

Troubleshoot

No Discover or Collector Event Results

Sometimes after you configure a Collector you might not see any results even after you verified those results exist outside of Cribl Stream and should have appeared in the run results.

Potential cause: One potential cause is the way time filtering works in Cribl Stream. There are two instances when time filters impact your data:

- When you submit a call for data within a specific time range.
- When you compare the data within a specific time range in Cribl Stream.

When you gather data from a Collector and it returns time-bound data, Cribl Stream runs Event Breakers on the data to break the incoming data into discrete data events. Each discrete event has its own timestamp. If any of those events are outside the `${earliest}` and `${latest}` variables as defined for the run, Cribl Stream is

8.1.1.13. Job Limits

You can configure global limits that optimize the execution of all Collectors and scheduled jobs (including Cribl Stream system tasks). To access and set these limits, follow the steps for your deployment type:

- Single-instance deployments: **Settings** > **System** > **General Settings** > **Limits** > **Jobs**.
  These limits apply to all Worker Processes.
- Distributed deployments: From the Leader, [select a Worker Group] > **Group Settings** > **System** > **General Settings** > **Limits** > **Jobs**.
  These limits are applied at the Worker Group level (except where noted), and trickle down to individual Worker Processes in the Group.

Limits Available

The following controls are available at the UI locations listed above.

Job Limits

Disable jobs/tasks: Toggled off by default in Stream Groups. When toggled on, Nodes running 4.5.0 and newer will not poll the Leader for jobs/tasks, and Nodes will not use the settings below (even if they're set).

Nodes running 4.4.4 and older still upgrade via jobs and will honor the Jobs settings, even with **Disable jobs/tasks** toggled on.

Concurrent job limit: The total number of jobs that can run concurrently. Defaults to `10`.

Concurrent system job limit: The total number of **system** jobs that can run concurrently. Defaults to `10`. Minimum `1`.

Concurrent scheduled job limit: The total number of **scheduled** jobs that can run concurrently. This limit is set as an offset relative to the **Concurrent job limit**. Defaults to `-2`.

Skipped jobs indicate that a Group's **Concurrent job limit** has been reached or exceeded. Increase this limit to reduce the number of skippable jobs. For resource-intensive jobs, this might require deploying more Worker Nodes.

Task Limits

Concurrent task limit: The total number of tasks that a Worker Process can run concurrently. Defaults to `2`. Minimum `1`.

Concurrent system task limit: The number of system tasks that a Worker Process can run concurrently. Defaults to `1`. Minimum `1`.

Task usage percentage limit: Value, between `0` and `1`, representing the percentage of total tasks on a Worker Process that any single job may consume. Defaults to `0.5` (i.e., 50%).

Task poll timeout: The number of milliseconds that a Worker's task handler will wait to receive a task, before retrying a request for a task. Defaults to `60000` (i.e., 60 seconds). Minimum `10000` (10 seconds).x

Completion Limits

Artifact reaper period: Interval on which Cribl Stream attempts to reap jobs' stale disk artifacts. Defaults to `30m`.

Finished job artifacts limit: Maximum number of finished job artifacts to keep on disk. Defaults to `100`. Minimum `0`.

Finished task artifacts limit: Maximum number of finished task artifacts to keep on disk, per job, on each Worker Node. Defaults to `500`. Minimum `0`.

Task Manifest and Buffering Limits

Manifest flush period: The rate (in milliseconds) at which a job's task manifest should be refreshed. Defaults to `100` ms. Minimum `100`, maximum `10000`.

Manifest buffer size limit: The maximum number of tasks that the task manifest can hold in memory before flushing to disk. Defaults to `1000`. Minimum `100`, maximum `10000`.

Manifest reader buffer size: The number of bytes that the task manifest reader should pull from disk. Defaults to `4kb`.

Job dispatching: The method by which tasks are assigned to Worker Processes. Defaults to `Least In-Flight Tasks`, to optimize available capacity. `Round Robin` is also available.

Job timeout: Maximum time a job is allowed to run. Defaults to `0`, for unlimited time. Units are seconds if not specified. Sample entries: `30`, `45s`, `15m`.

Task heartbeat period: The heartbeat period (in seconds) for tasks to report back to the Leader/API. Defaults to `60` seconds. Minimum `60`.

8.1.1.14. Using Collectors

Learn how to replay events using collectors gathering data from various senders.

- Using S3 Storage and Replay
- Using REST/API Collectors
- Lacework API Collection
- Microsoft Graph API Collection
- ServiceNow API Collection
- Creating a Custom Collector

8.1.1.14.1. Configure A Database Connection

Database Connections are reusable configuration objects in Cribl Stream that enable Database Collectors to access and retrieve data from a MySQL, Oracle, Postgres, or SQL Server database. You can reference the same Database Connection in multiple Database Collectors, which helps streamline Database Collector setup.

Overview of Database Collector Configuration

The workflow for setting up a new Database Collector has these phases:

1. Create a Connection String or Configuration Object: Before you start, you need to gather access and authentication information for your database. Using that information, build a connection string or configuration object that you will use to establish a valid connection to the database.
2. Configure a Database Connection: Database Collectors rely on Cribl Stream Database Connections, which are reusable objects that you can re-use with multiple Database Collectors to authenticate to the same database.
3. Test the Connection: Ensure the Database Connection is configured correctly and that data is flowing.
4. Configure a Database Collector: Configure a Database Collector to input data from an upstream database. You can then route this data to a Pipeline for further data processing.

Create a Connection String or Configuration Object

To get started with a Database Collector, you need to gather important details and authentication information about your target database. Using that information, you can build a **connection string**. A connection string is a structured text string that contains all the required parameters for Cribl Stream to establish a valid connection to the database.

In addition to connection strings, SQL Server databases can also accept a configuration object, which is a structured data format (such as JSON) that organizes the required database access into defined fields.

Connection strings should be properly escaped, especially when your password might contain a special character. For example, characters like `@`, `:`, `/`, `?`, `#`, or `%` may be interpreted as delimiters inside a URI. If they aren't escaped, the connection string can break or fail to authenticate. Use percent-encoding (URL encoding) for these characters. For example, a password of `MyP@ssw0rd!` would be encoded into `MyP%40ssw0rd%21`.

For information about creating connection strings, jump to the section for your database type:

- MySQL, Oracle, and Postgres

- SQL Server

MySQL, Oracle, and Postgres

Select the appropriate tab for information about building a connection string for these database types:

MySQL | Oracle | Postgres

A MySQL connection string requires this information:

- **Protocol:** Verify that your database is using a secure SSL or TLS connection and that you have the required certificates. Otherwise, your connection will need to use a non-secure protocol, which is not recommended. Append an `s` to your connection string header to enforce the secure SSL/TLS connection protocol. For example: `mysqls://`.
- **Username:** The database user account that has the necessary permissions to read tables and run queries. As a best practice, ensure that this account follows the principle of least privilege and has the minimum permissions required by Cribl Stream.
- **Password:** The password associated with the specified database user account.
- **Host:** The hostname or IP address where the MySQL database server is running.
- **Port:** MySQL uses a default port of `3306`, but you should confirm this with your database administrator. It is possible that your deployment runs on a non-standard port.
- **Database name:** The specific name of the database (schema) on the server that contains the tables or data to collect.
- **Optional parameters:** Any additional parameters that are needed for extra settings or security, such as the SSL mode. For example: `?sslmode=require` or `?sslmode=verify-full&sslrootcert=/path/to/ca.crt`.

Use that information to build a connection string. MySQL connection strings use this syntax:

```
mysqls://username:password@host:port/databasename?option=value
```

Copy

Examples:

```
mysqls://audit_user:L0gM3In#@mysql-staging.net:3307/staging_logs
mysqls://cribl_reader:S3cur3R3@d3r!@prod_mysql_db.company.net:3306/production_logs
mysql://dev_read:devpass123@192.168.1.5:3306/dev_data?sslmode=verify-full&sslrootc
```

Copy

SQL Server

The necessary requirements for SQL Server depend on how you will authenticate Cribl Stream with the server. Select the appropriate tab for your authentication type:

Service Principal | NTLM

If your SQL Server authenticates with a service principal, you must first register an application with Microsoft Entra ID (formerly known as Azure Active Directory) as a prerequisite. To prevent connection errors, you will need to use its ID and credentials in your configuration. To register an application with Microsoft Entra ID:

1. Register an application using the instructions from Microsoft: Register an application in Microsoft Entra ID. You can use the default settings as needed.
2. Generate credentials for the application: Add and manage application credentials in Microsoft Entra ID.

```
CREATE USER [<app_display_name>] FROM EXTERNAL PROVIDER;
ALTER ROLE db_owner ADD MEMBER [<app_display_name>];
```

Copy

3. After registering the application, get the following information from your Azure Portal:
    - **Server Name:** The Fully Qualified Domain Name (FQDN) or IP address of the machine hosting the SQL Server instance.
    - **Database Name:** The name of the specific logical database catalog on the SQL Server instance that contains the tables or data you want to collect.
    - **Port:** The TCP/IP port number on which the SQL Server is listening for connections. The default port is `1433`, but it is possible that your deployment runs on a non-standard port.
    - **Tenant ID**: The Directory (tenant) ID for your Azure instance.
    - **Client ID**: The Application (client) ID for your app registration.
    - **Client Secret**: The secret value you generated for your service principal.

Use that information to build a configuration object. SQL Server configuration objects that authenticate with a service principal use this syntax:

```
{
  "server":  "<your-server-name.database.windows.net>",
  "database": "<your-database-name>",
  "port": 1433,
  "authentication": {
    "type": "azure-active-directory-service-principal-secret",
    "options": {
      "tenantId": "<your-tenant-id>",
      "clientId": "<your-application-id>",
      "clientSecret": "<your-client-secret-value>"
    }
  },
  "options": {
    "encrypt": true,
    "trustServerCertificate": false
  }
}
```

Copy

Example:

```
{
  "server": "prod-sql-metrics.database.windows.net",
  "database": "Application_Telemetry",
  "port": 1433,
  "authentication": {
    "type": "azure-active-directory-service-principal-secret",
    "options": {
      "tenantId": "8e20f01a-1e4b-4c5d-b0a9-1f2a3c4d5e6f",
      "clientId": "1c3b5d7e-9f0a-2b3c-4d5e-6f7a8b9c0d1e",
      "clientSecret": "V3ryS3cr3tT0k3n@2025!"
    }
  },
  "options": {
    "encrypt": true,
    "trustServerCertificate": false
  }
}
```

Copy

Configure a Database Connection

After creating a connection string or configuration object for your database type, the next step is to configure a Database Connection for the target database. Database Connections are reusable Knowledge objects that you can use and re-use to connect the same database to multiple Database Collectors as needed.

When you change a Database Connection, it impacts all Database Collectors that use that Database Connection. Make changes with caution.

To configure or update a Database Connection:

1. Select **Worker Groups** from the sidebar, and choose a Worker Group. Then, on the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Database Connections**. Select **Add Database Connection**.

   You can also create a new Database Connection directly from a Database Collector. While configuring a Database Collector, select **Create** next to the **Connection** field to open the **New Database Connection** modal.

2. In the **New Database Connection** modal, configure the following settings:
   - **ID**: Enter a unique ID for this connection. A descriptive ID will help all users recognize the connection by purpose.
   - **Database type**: Select your database type: `MySQL`, `SQL Server`, `Postgres`, or `Oracle`.
   - **Authentication method:** The available options available depend on your database type:

Authentication Method	Description
Connection String	Enter the connection string you created in the previous Create a Connection String or Configuration Object step.
Connection String Secret	Rather than storing the connection string in the Database Connection object, you can add it to secret store and then select the stored connection string. Select **Create** to add it to the secret store.
Config (SQL Server only)	Enter the JSON connection configuration object you created in the previous Create a Connection String or Configuration Object step. The object uses the Tedious driver and the node-mssql client.

Authentication Method	Description
Stored Credentials (Oracle only)	Use the secret store for both your database login and connection details. Select a username and password secret for the **Credentials secret** field and a text secret for the **Connection string secret** field to ensure secure, centralized management.

Connection strings should be properly escaped, especially when your password might contain a special character. For example, characters like `@`, `:`, `/`, `?`, `#`, or `%` may be interpreted as delimiters inside a URI. If they aren't escaped, the connection string can break or fail to authenticate. Use percent-encoding (URL encoding) for these characters. For example, a password of `MyP@ssw0rd!` would be encoded into `MyP%40ssw0rd%21`.

3. Optionally, configure the **Additional Settings**:
    - **Connection timeout (ms)**: How long, in milliseconds, Cribl Stream should wait before assuming that a connection has failed. This defaults to `10000` (10 sec.) for MySQL databases, and to `15000` (15 sec.) for SQL Server databases. For both, the minimum allowed value is `1000` (1 sec.), and the maximum is `60000` (1 minute).
    - **Request timeout (ms)**: This setting appears only where **Database type** is set to `SQL Server`. Defines how long Cribl Stream should wait before assuming that a request has failed. If a query requires a longer timeout than the default `15000` (15 seconds), be sure that you know why. Long-running queries can affect other parts of the system. The minimum is `1000` (1 second).
4. Select **Save** to save your changes.

Validate that your Database Connection is working as expected and that data is flowing by testing your connection, as explained in the next section.

Test the Connection

You can test the connection to the database inside the Database Connection by selecting the **Test Connection** button inside the Database Connection object. This initiates an API call that pings the database to check whether the connection is valid.

Be aware that when you select **Test Connection** on the Leader Node, the system only tests the database connection for the Leader Node itself. However, the Leader Node is usually not the node that needs to connect to the database while collecting data. Usually that operation is performed by a Worker Node.

To test the connection from a specific Worker Node:

1. Ensure the Database Connection is committed and deployed.
2. Teleport to the Worker Node and open the Database Connection on that Worker Node.

3. Select **Test Connection**.
    - **Success:** If the test is successful, a success message displays in the lower right corner of the screen.
    - **Failure:** If the test fails, an error message display at the top of the screen. Connection test failures are logged in the logs for the Worker Group

8.1.1.14.2. Using S3 Storage And Replay

Cribl Stream's Replay options offer organizations fundamentally new ways to manage data, by providing an easy way to selectively ingest, **and re-ingest**, data into systems of analysis. Let's walk through how to use this feature, step by step.

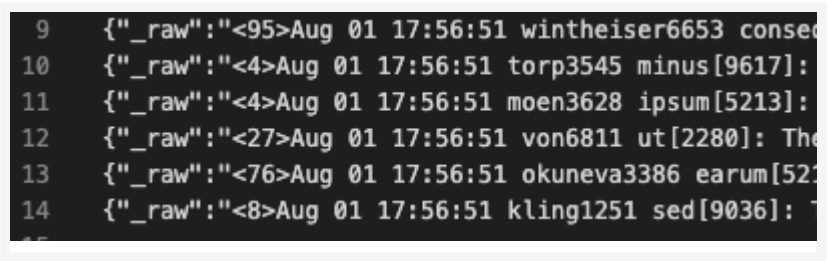
For simplicity, we'll treat the storage destination here as Amazon S3, although it could just as easily be MinIO, or any of several other options.

Choosing JSON vs. Raw Format

You can write data out of Cribl Stream in either of two formats - JSON or raw.

JSON

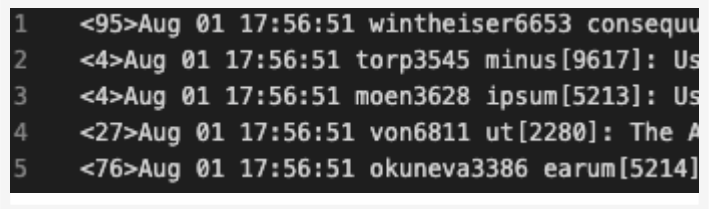
With this option, the parsed event, with all metadata and modifications it contains at the time it reaches the Destination step, will be wrapped in a JSON object. Each event is one line. This is newline-delimited JSON (abbreviated NDJSON). For example, here is syslog data using the JSON format option:



JSON-formatted event

Raw

With this option, the contents of the event's `_raw` field - unparsed, at the time it reaches the Destination step - are written out in plaintext. Each event is one line. For example, here's the same syslog data, except unmodified and unparsed, as written out with the raw option:



Raw-formatted

Which Format?

Cribl recommends using the default JSON format. Here, we expect that timestamp extractions and other vital enhancements have already been performed. Reusing that information makes sense, and will make your replay simpler.

Notice that in the screen capture above, the **raw** format simply contains the original data. There might be cases where you want this; but usually, exporting the preprocessed event is more desirable.

Cribl Stream supports replaying data that has been exported as Parquet, through any of the following: Amazon S3 Collector, Azure Blob Storage Collector, Google Cloud Storage Collector, or Filesystem Collector.

Writing the Data Out

The Worker Nodes stage files until certain limits are reached: Time open, idle time, size, or number of files. These settings are available on the Destination configuration modal's **Advanced Settings** tab (see S3 details here).

Once any of the configured limits is reached, the Worker gzips the file and drops it into the object store. If you reach the open-file limit, the oldest file will be targeted.

The S3 Destination's settings also allow you to define how the uploaded files are partitioned. Host, time, sourcetype, source - all the metadata is available to you for this purpose. When you're replaying data from the store, these partitions will be handy, to make your replay searches faster.

We can map segments of the path back to variables (including `time`) that you can use to zero in on the exact logs you need to replay, without requiring checking `_raw`.

For examples of the expressions Cribl recommends, see the example Destination definition below.

Retrieving the Events

With events stored in an object store, we can now point Cribl Stream to that store to Replay selected events back through the system. We want to use data in the file path as an initial level of filtering, to exclude as much data as we can from download. Object retrieval and unpacking imposes a big resource hit in the Replay process, so minimize your impact radius. Searching against `_raw` data is also possible, but should be secondary to `_time`, `sourcetype`, `index`, `host`, etc.

Retrieval details: The Leader picks one Node to do the discovery exploration, to find the potential objects that are in play. That list of targets is then doled out to the Worker Group to actually pull down the objects, and to examine them for content matches, before executing final delivery. All Worker Nodes share the workload of retrieving and re-injecting the data.

Finally, we need to process the data coming back to extract each event. As with any incoming data stream on a compatible Source, Cribl Stream can use default, or custom, Event Breaker definitions. In this case, we recommended above to use JSON as the format of the events when we write to disk, so we'll use the **Cribl** Event Breaker ruleset on this Source. This Event Breaker contains a newline-delimited-JSON definition.

Setting Up and Running Replay

With the above concepts established, let's put them to work. We'll round-trip data through our Destination and replay it in Cribl Stream.

The Store

Object storage, or any shared storage, will work. As long as all the Cribl Stream Nodes can see it, we're ready to go. For the purposes of this post, let's stick with an S3-compatible store.

You'll need all the credentials, keys, secret keys, endpoints, etc., required to access the bucket that you intend to use in your object store. (For details on cross-account access, see this blog post.) Obviously, the bucket should be able to grow to the size intended for your long-term archival needs.

Storage Class Compatibility

Cribl Stream's Collectors and Replay feature do not support the S3 Glacier, S3 Deep Glacier, or Azure archive tier, due to long retrieval times for those storage classes.

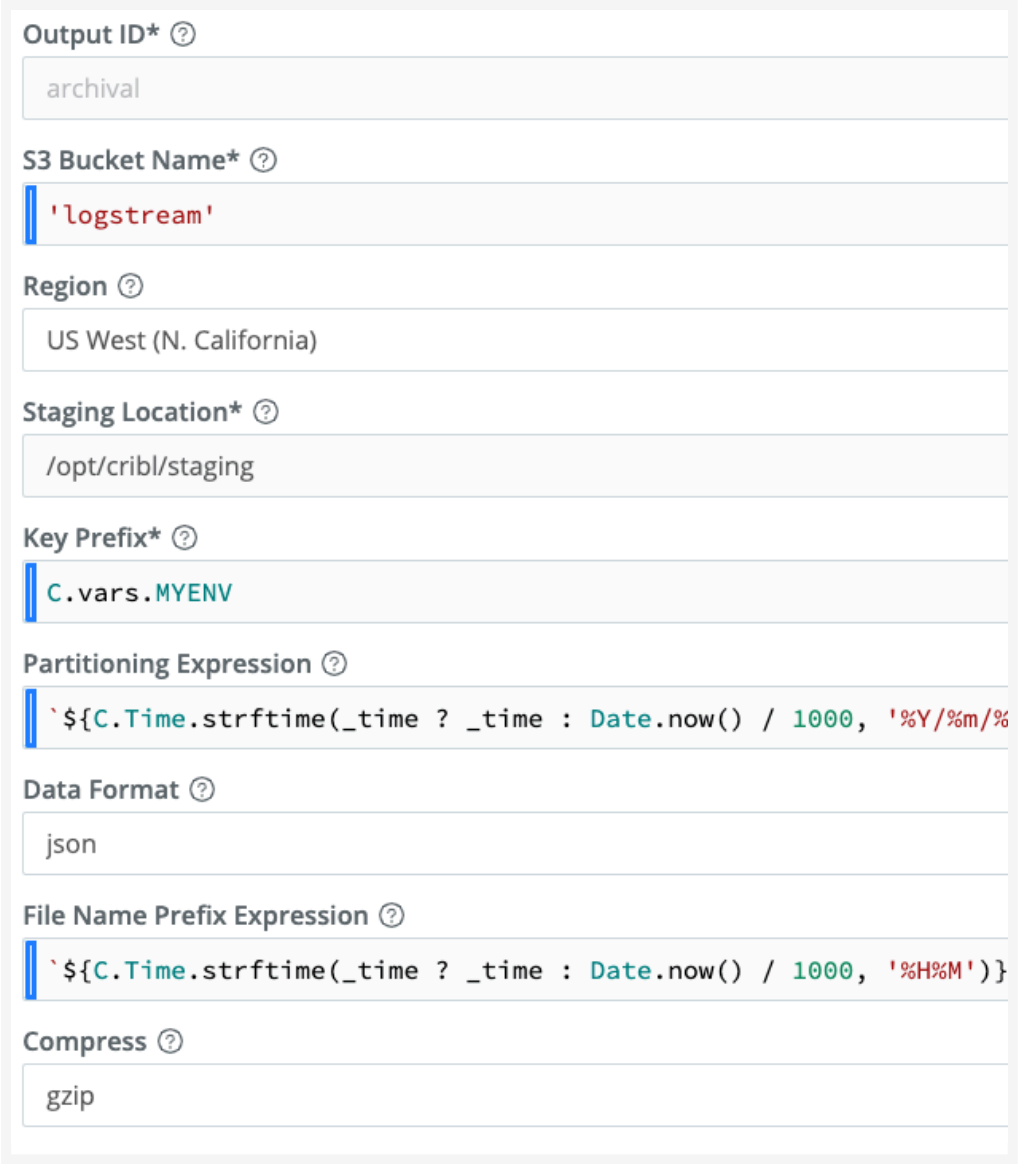
Collectors and Replay **do** support S3 Glacier Instant Retrieval when you're using the S3 Intelligent-Tiering storage class.

The Destination Definition

In your Cribl Stream Worker Group config, create a new S3 Destination. The screenshot below is an example built for this demo.

Use the Destination's **Advanced Settings** tab to adjust the limits, if needed. The defaults are fine for most situations, but depending on your partitioning scheme, we could be talking about 100+ files. So **make sure your staging area has enough space**.

In particular, set the **Max open files** option appropriately. It overrides the size and time limits. We recommend that the staging area be its own volume, so that you don't fill up a more-vital volume by mistake.



Partitioning and filename expressions

The screen capture above includes the partitioning and filename prefix expressions. Below is the full text of each expression. In a nutshell, we're using time and other metadata to construct a path in the object store, which will be useful to us at replay time:

```
Year/Month/Day/index/host/sourcetype/HHMM-foobar.gz
```

Copy

Partitioning (one line):

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d')}/${index ? index :
```

Copy

Filename:

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%H%M')}`
```

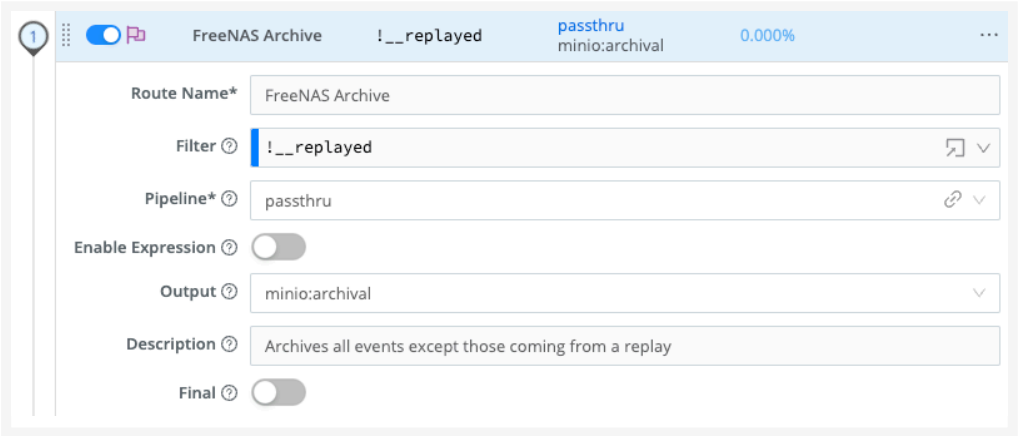
Copy

We've also included a Key Prefix from Cribl Stream's **Processing** > **Knowledge** > **Global Variables**. You could use this to partition your logs by environment, or any other qualifier.

The Archival Route

Create a new Cribl Stream Route called `Archival` that matches everything you want to archive. In this case, we've set it to `!__replayed`, and set this internal field in the Source (see the next step). Any event that is **not** coming from the defined Replay process will be archived.

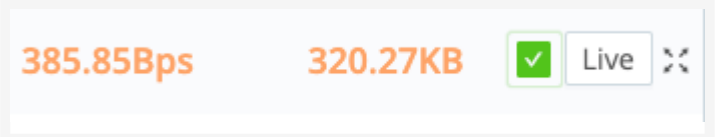
Select the `passthru` Pipeline, or create an empty Pipeline and use that. Select the S3 Destination you created above. And finally, make sure **Final** is **not set**. We want data to flow through this Route, not stop at it. So, position the `Archival` Route at or near the top of the Routing table. Save your work, commit, and deploy.



Pipeline configuration

Once saved and deployed, data should be flowing to your object store. Check your work: Go to the Monitoring dashboard and select **Data** > **Destinations** at the top. You want to see a green checkmark on the right side for your S3 Destination. If it's not green, find out why.

You can also use an S3-capable browser, like Cyberduck, to check the files more manually. (Specific troubleshooting steps are outside the scope of this doc.)



A green (healthy) destination

The Replay Collector

From the top nav of your Cribl Stream Worker Group or single instance, select **Data** > **Sources** > **Collectors** > **S3**, and create a new S3 Collector with the ID `Replay`. Use the **Auto-populate from** option to pull in the configs from your S3 Destination.

In the **Path** field, we need to establish the tokens to extract from the path on the S3 store. If you used the recommended partitioning scheme in the above example Destination definition, we recommend specifying these tokens here:

```
/${MYENV}/${_time:%Y}/${_time:%m}/${_time:%d}/${index}/${host}/${sourcetype}/${_time:
```

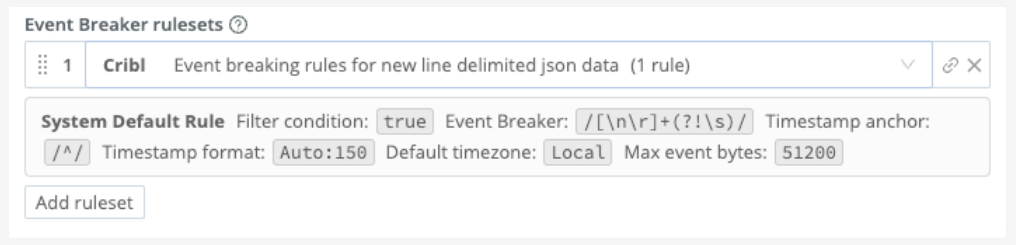
Copy

If you chose to leave out `C.Vars.MYENV`, exclude the first segment. It is **vital** that this scheme match your partitioning scheme in the Destination definition.

With these tokens defined, you can now define filters that exclude and include relevant files before Cribl Stream is required to download and open them. This cuts down on the work required, by orders of magnitude.

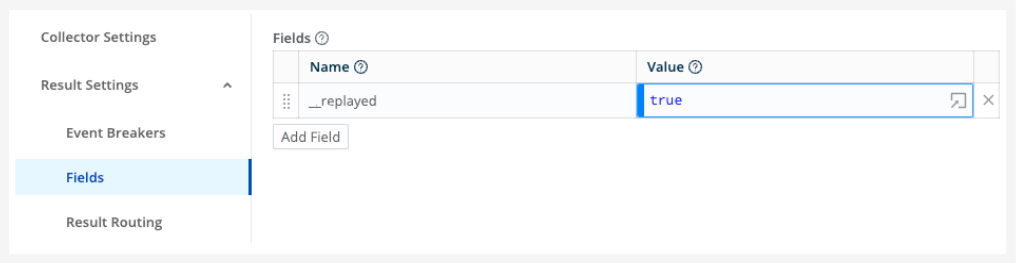
> This is a core concept. We want to avoid having to open files to check for matches, by instead relying on key data in the path. Cribl Stream can immediately exclude, without downloading, the files that have no chance of matching our target events.
>
> Using `_time`, `sourcetype`, `host`, and `index`, it can accurately zero in on the target files. After this high-level filtering, Cribl Stream will download and interrogate the contents of what's left, and will send the matches along to the Routing table.

Finally, under **Result Settings** > **Event Breakers**, select the `Cribl` Ruleset. This Ruleset understands how to parse the JSON packaged events stored by the Archive Destination.



Cribl Event Breaker Ruleset

To make it easier to identify events that have been replayed, create a field (**Result Settings** > **Fields**) named `__replayed` and set it to `true`. You could filter on this field in Routes and Pipelines later. Because it's a double-underscore field, it's internal-only, and won't be passed on to your final destinations. In a previous step, we used it to prevent replayed events from being re-archived:



__replayed field

Now save your Collector, and commit and deploy.

Testing Replay

After you've accumulated some data in your S3 store, head over to Pipelines and start a capture. For the filter, use `__replayed`, and run it for 300 seconds, 10 max events.

Once it's running, in a new browser window, navigate back to Collectors, and run your Replay Collector. Filter on `true`, and set the **Earliest** time to `-1h` and the **Latest** to `now`.

You can run it in Preview mode to make sure you get results, and then come back and do a full run. (Or you can just do a full run right off the bat if you're confident.)

With a running job, you can click on the job ID to follow its progress. You can also pop back over to the Capture browser window or tab, and you should see events there.

In a full run, the events will proceed through your Routes as normal, and will land wherever your original Routes and Pipelines dictate. In this case, they landed in Splunk, and we could easily see duplicate events - that

8.1.1.14.3. Using REST/API Collectors

The REST/API Endpoint Collector is powerful, but complex. This use case demonstrates several examples of building and running REST Collectors to pull data from public and simulated REST endpoints.

Check out the example REST Collector configurations in Cribl's Collector Templates repository. For many popular Collectors, the repo provides configurations (with companion Event Breakers, and event samples in some cases) that you can import into your Cribl Stream instance, saving the time you'd have spent building them yourself.

To configure a Collector for Netskope Events and Alerts, see this Netskope Community topic.

1. Basic HTTP GET

This example performs an HTTP GET operation against an external Joke API. This API uses a license key header to authenticate the user.

Discover type: None

Collect URL: `'https://matchilling-chuck-norris-jokes-v1.p.rapidapi.com/jokes/random'`

Collect parameters: None

Collect headers:

```
accept: 'application/json'

x-rapidapi-key: 'e4068647ffmsh65536596798f49dp17e998jsn342bac862377'

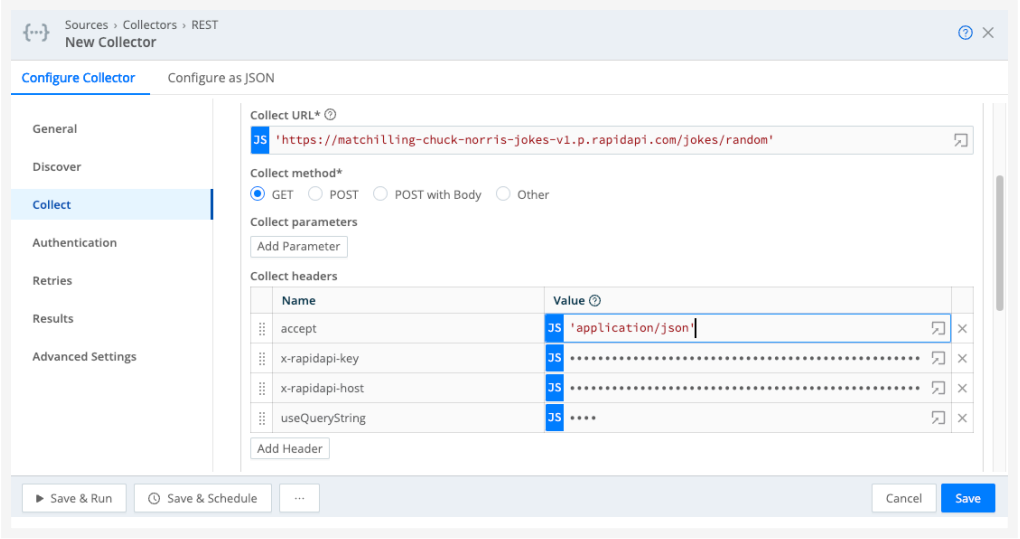
x-rapidapi-host: 'matchilling-chuck-norris-jokes-v1.p.rapidapi.com'

useQueryString: true
```

Pagination: None

Authentication: None

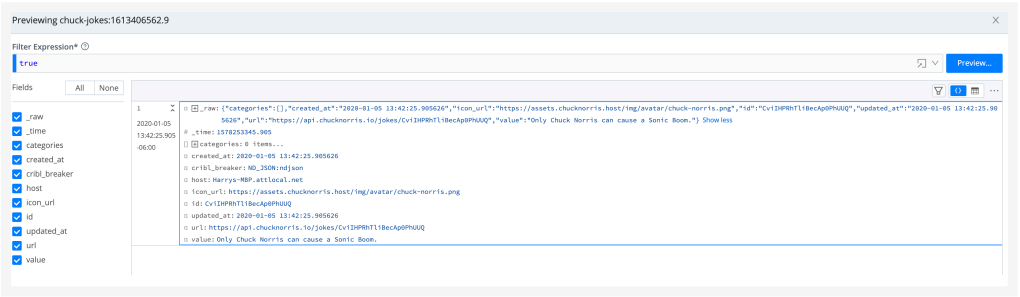
Event Breaker: JSON Newline Delimited - use Cribl Stream's built-in **Cribl > ndjson** rule, and associate it with the Collector to parse the JSON document.



Collector configuration for basic HTTP GET

Results

When run (in Preview mode), the Collector should return a single JSON record. If the Collector is set up with an NDJSON Event Breaker, it will look like this:



Returned event

2. HTTP GET with Pagination via URL Attribute

The REST Collector's Pagination feature allows collection to retrieve 1-*N* pages of data, using attributes returned in either the response body or response header. The returned attribute can either be a URL (referencing the next page), or a token that can be added to subsequent request headers or parameters.

In this example, a returned response-body attribute contains a URL that references the next page. Pagination will continue until either the Collector's **Page limit** setting is reached, or no more pages are present (that is, the returned attribute is not present in the response body).

This example's API retrieves near-Earth asteroid data from NASA. The example uses a JSON Array Event Breaker to extract individual records from an array attribute in the response.

Discover type: None

Collect URL: `'http://www.neowsapp.com/rest/v1/neo/browse?api_key=oDa6w0fjsKEb1N3bMA5dMLhatMJ4WC5XtOBTrLrk'`

Collect parameters: None - Parameters in this example are added to the header. Static parameters (that is, parameters that don't reference variables) can safely be added to the URL. Any parameters that do reference variables should always be added in the **Collect parameters** section, to allow filtering of values that evaluate as undefined.

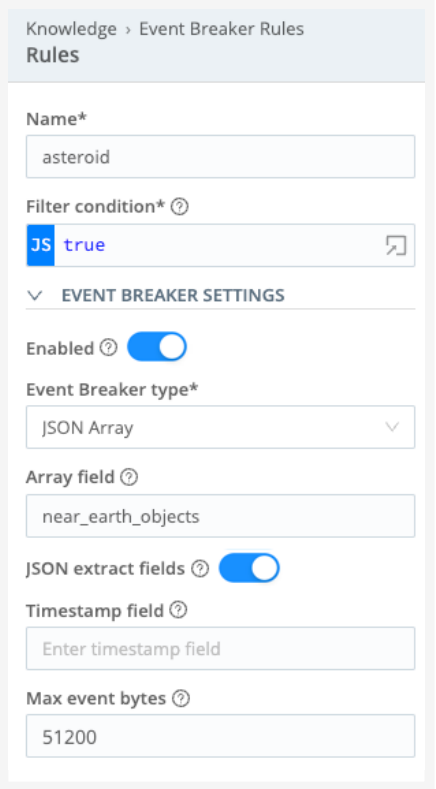
Collect headers: None

Pagination: Response Body Attribute

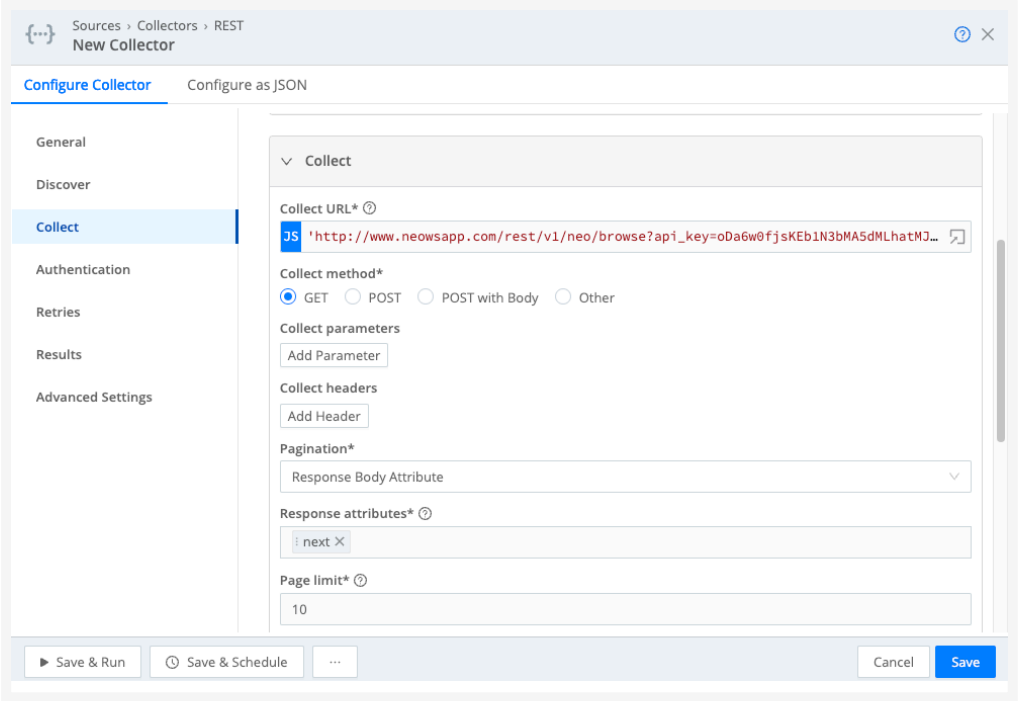
Response attributes: `next`

Authentication: None

Event Breaker: JSON Array

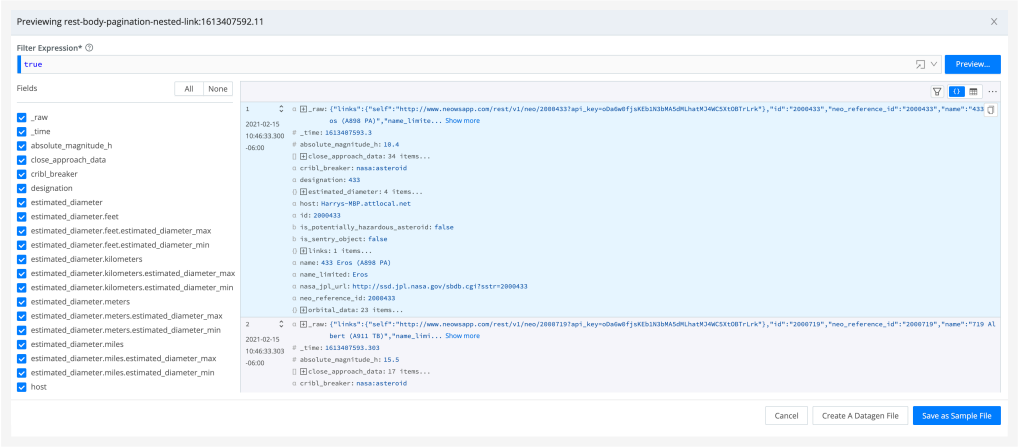


Event Breaker Configuration (Pagination via URL Attribute)



Collector configuration for HTTP GET, paginated via URL Attribute

When run (in Preview mode), the Collector should return multiple records extracted from the Event Breaker. In this example, we limited output to 10 pages of data. This particular dataset has over 1,000 total pages, so it's a good idea to limit output to avoid a job that runs too long.



Paginated events

This API allows a certain number of calls/month. Cribl recommends that you not schedule this Collector - run it ad-hoc, for testing only.

3. HTTP GET with Pagination via Response Body Attribute

This example uses Response Body Attribute pagination, which returns a token that is passed as a request parameter to retrieve subsequent pages of data. The only difference between this example and Example 2 is how the Response Body Attribute is used.

To authenticate against the GreyNoise endpoint used in this example, set up a trial account according to GreyNoise's Setting Up a Trial Account documentation.

Discover type: None

Collect URL: `'https://api.greynoise.io/v2/experimental/gnql'`

Collect method: `GET`

Collect parameters:

```
query: 'last_seen:1d'
```

```
scroll: `${scroll}`
```

Collect headers:

```
accept: 'application/json'
```

```
key: '<your-GreyNoise-API-key-here>'
```

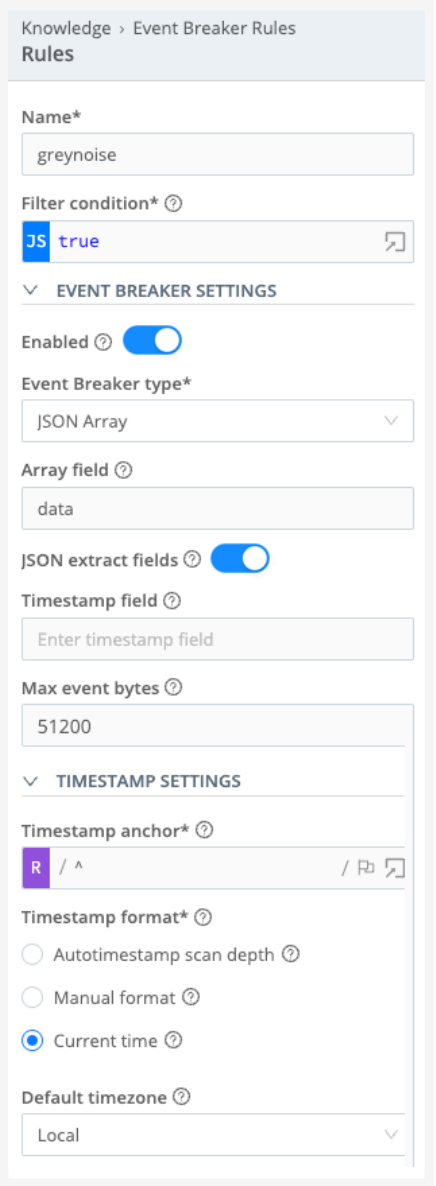
Pagination: Response Body Attribute

Response attributes: `scroll`

Page limit: `10` (or `0` to pull all data)

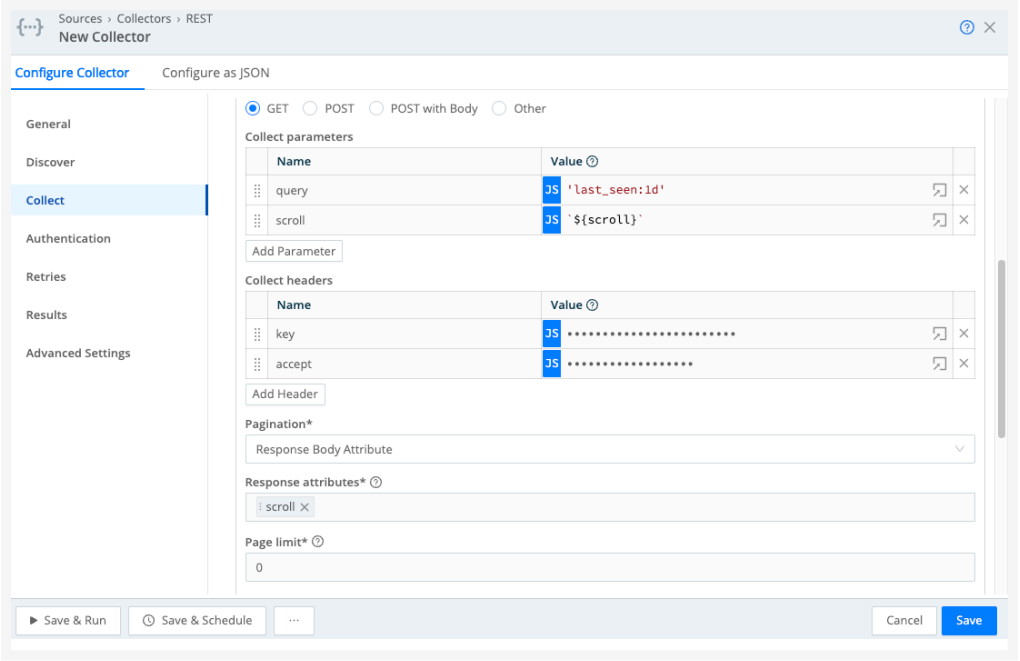
Authentication: None

Event Breaker: JSON Array - use the configuration shown here:



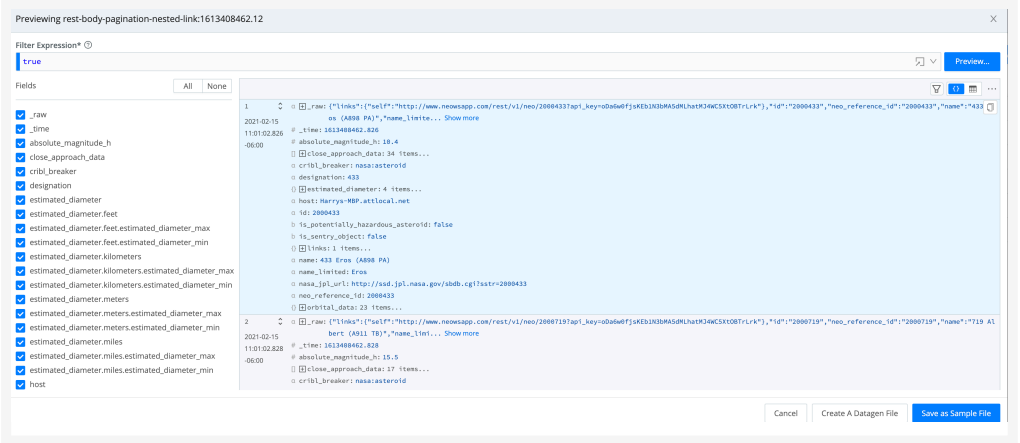
Event Breaker Configuration (Pagination via Response Body Attribute)

In this example, the response body returns an attribute named `scroll`, which is a token that references the next page of data to fetch. We reference the attribute in **Collect parameters** using the JavaScript expression: `` `${scroll}` ``. If present, this will be passed to retrieve subsequent pages of data, until either the Collector's **Page limit** setting is reached, or no more pages are present.



Collector configuration for HTTP GET, paginated via Response Body Attribute

Collector Output



Paginated events

This API allows a certain number of calls/month. Cribl recommends that you not schedule this Collector - run it ad-hoc, for testing only.

For a more detailed use case around this particular API, see Cribl's Enrichment at Scale! blog post.

4. Pagination via Nested Response Body Attributes

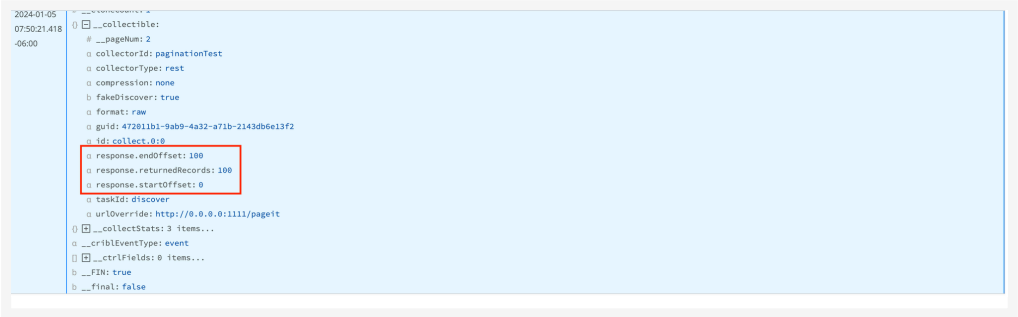
When using Response Body Attribute pagination with nested attributes for determining the next page, the extracted attributes from the response will have dots (`.`) included in their names within `__collectible`. This

requires different syntax for accessing these attributes within the **Collect URL**, **Collect parameters**, **Collect POST body**, **Collect headers**, and **Last-page expression** fields.

For example, the following shows attributes two levels deep:

Attributes two levels deep

The attributes are added to `__collectible` with dots (`.`) in the name:

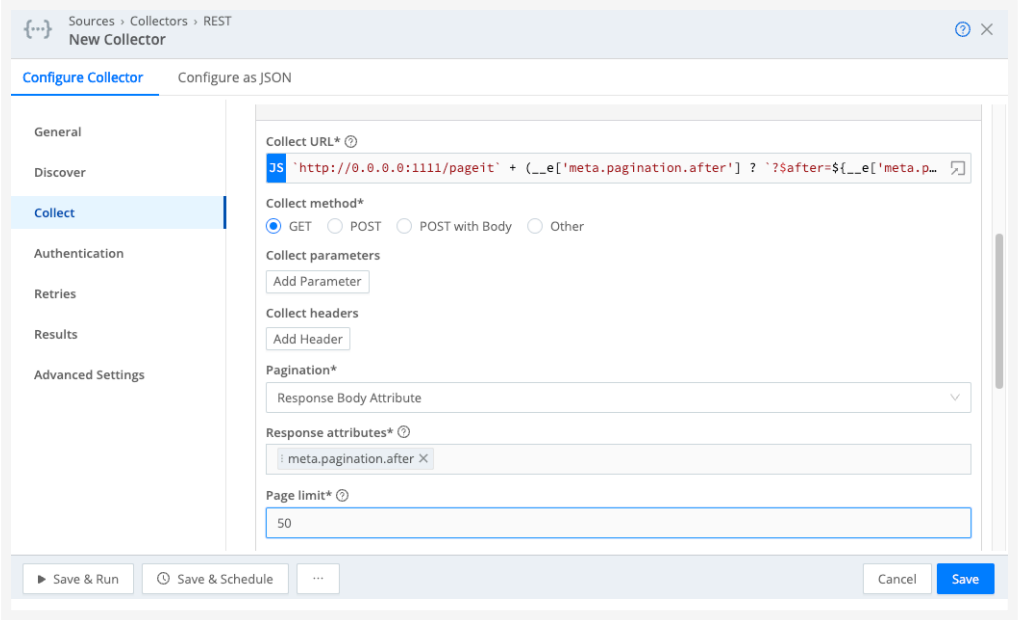


Attributes added to `__collectible`

You need special syntax to reference the attributes because they have dots in the name: `__e['attribute.name.with.dots']`.

The same syntax referenced in example 3 above, `__e['response.endOffset']`, can be used in the **Collect URL**, **Collect parameters**, **Collect POST body**, **Collect headers**, and **Last-page expression** fields. Here are examples of each.

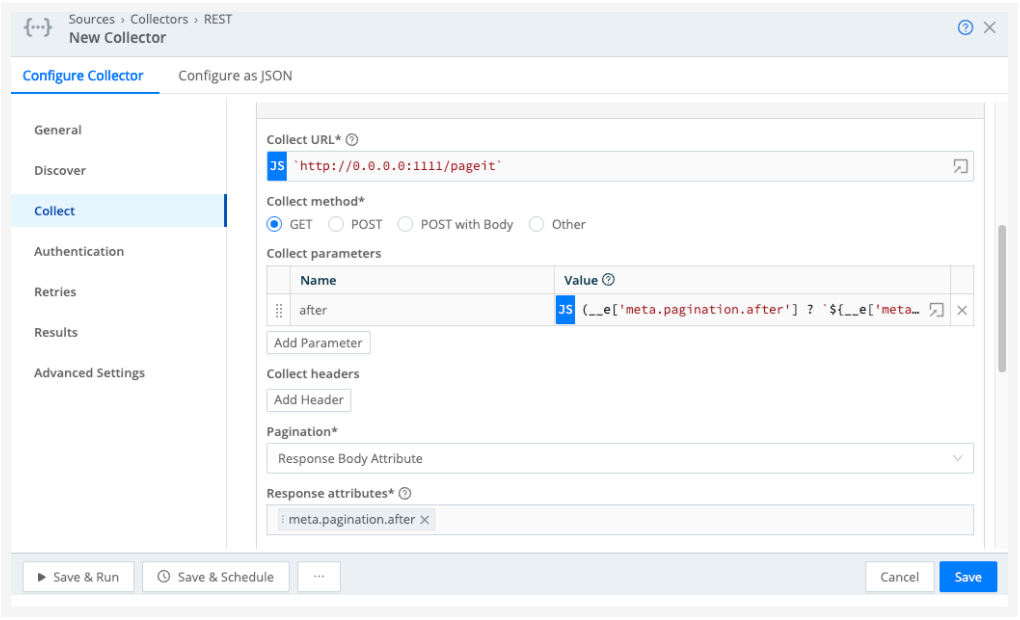
Collect URL: `` `http://0.0.0.0:1111/pageit` + (__e['meta.pagination.after'] ? `?$after=${__e['meta.pagination.after']}` : '') ``



Collect URL

Collect parameters:

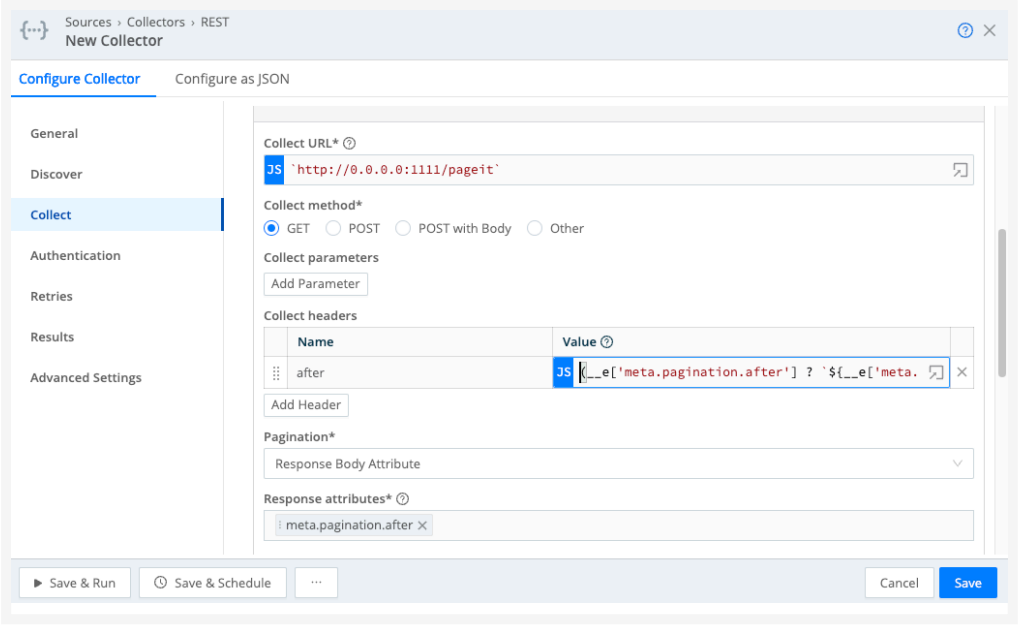
- **Name**: `after`
- **Value**: `` `${__e['meta.pagination.after'] ? __e['meta.pagination.after'] : ''}` ``



Collect parameters

Collect headers:

- **Name**: `after`
- **Value**: `` `${__e['meta.pagination.after'] ? __e['meta.pagination.after'] : ''}` ``

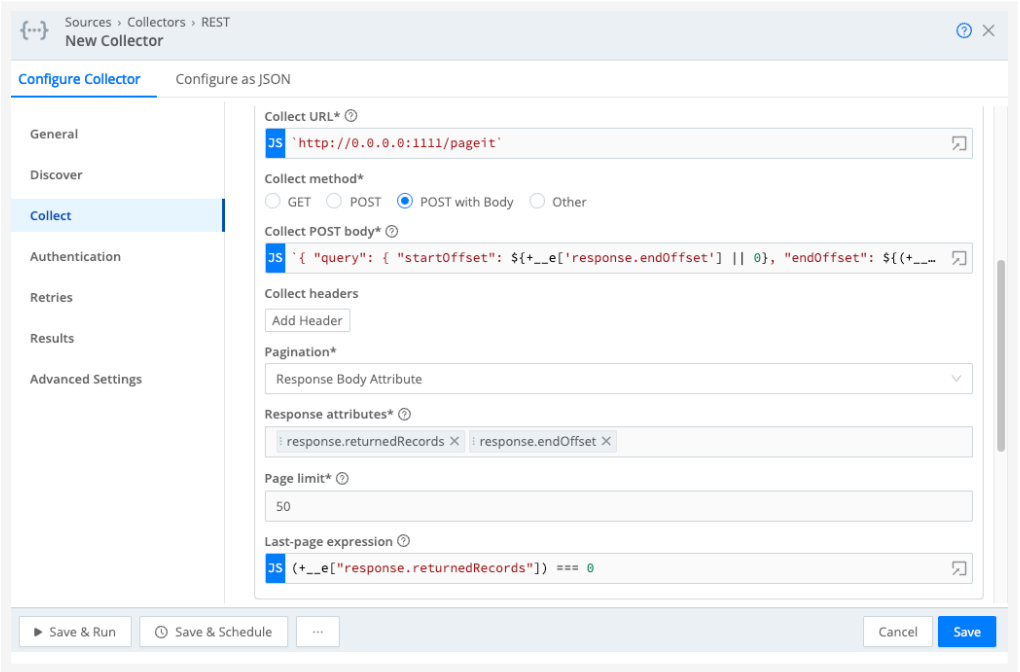


Collect headers

Collect POST body and **Last-page expression**:

- **Collect POST body**: `` `{ "query": { "startOffset": ${+__e['response.endOffset'] || 0}, "endOffset": ${(+__e['response.endOffset'] || 0) + 100} } }` ``
- **Last page expression**: `(+__e["response.returnedRecords"]) === 0`

Here the API passes the `startOffset` and `endOffset` parameters in the POST body to page through data until `response.returnedRecords == 0`. The **Last page expression** field is used to determine when all the available data has been consumed.



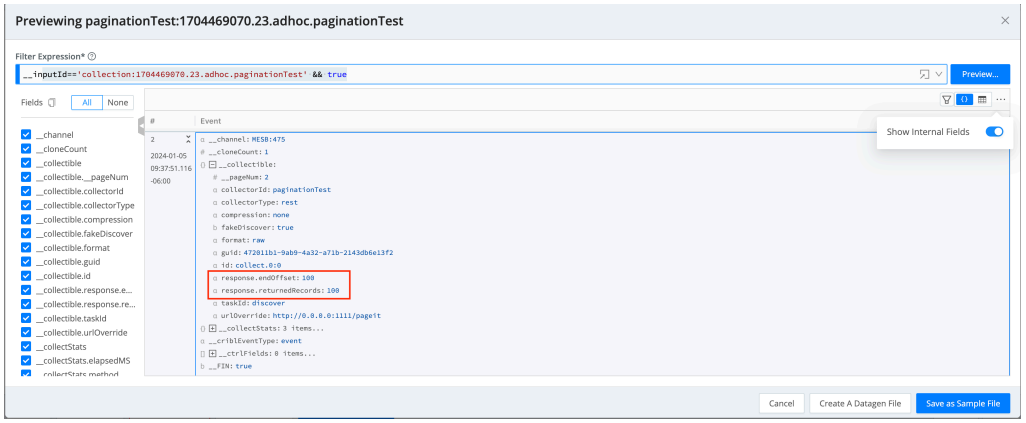
Collect POST body and Last-page expression

Here's sample output from the API showing partial results returned for the first page of data:

```
{
  "response": {
    "totalRecords": "1275",
    "returnedRecords": "100",
    "startOffset": "0",
    "endOffset": "100",
    "data": [
      {
        "one": 1
      },
      {
        "two": 2
      },
      ...
    ]
  }
}
```

Copy

Notice that the Collector Response attributes use dotted notation to access `endOffset` and `returnedRecords` as `response.endOffset` and `response.returnedRecords` respectively. This tells the Collector that the attributes of interest are named `response.endOffset` (dot in name) or nested JSON attributes: `{ response : { endOffset: 100 ... } }`. However, the response is structured (dots in attribute name or nested) the Collector will add the attributes to the second page's `__collectible` attribute using dots in the attribute name, for example:



`__collectible` second page of attributes

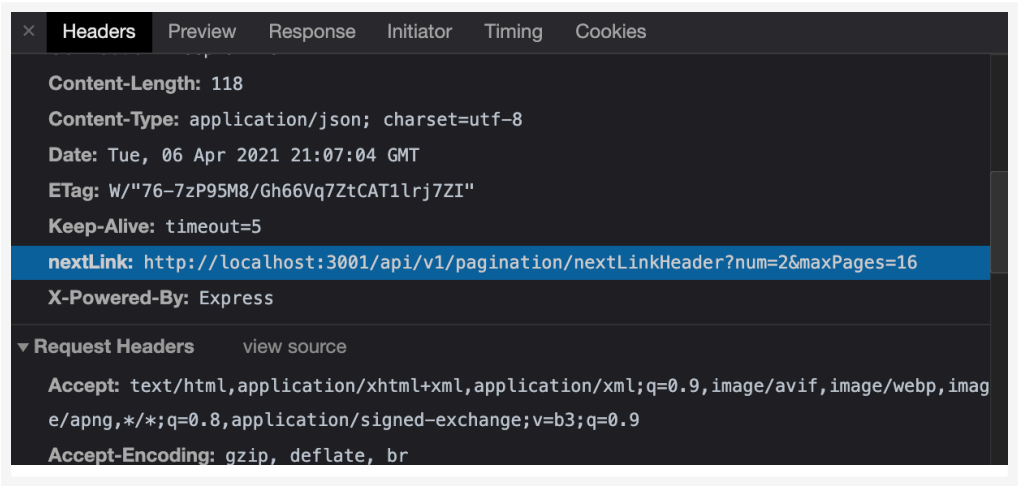
Now that you have the attributes available in `__collectible`, you can use them in the **Collect URL**, **Collect parameters**, **Collect POST body**, **Collect headers**, and **Last-page expression** fields. The syntax is the same across all these fields. You'll use a special variable, `__e`, which acts like a shorthand for `__collectible` (since you can't use it directly here). Here's how to access specific attributes:

- `response.endOffset`: Use `__e['response.endOffset']`
- `response.returnedRecords`: Use `__e['response.returnedRecords']`

5. HTTP GET with Pagination via Response Header URL

This example leverages pagination using a Response Header Attribute value. The value returned can be either a URL (of the next page) or a token value (a request attribute that is passed to retrieve the next page of data).

This example is based around a local Web server on port 3001. The server returns a response header when another page of data is available, and the header contains the URL of the next page. Here's how the header looks in developer tools:



Next-page URL passed as Response Header Attribute

Collector Configuration

Discover type: None

Collect URL: `'http://localhost:3001/api/v1/pagination/nextLinkHeader?num=1&maxPages=16'`

Collect parameters: None

Collect headers: None

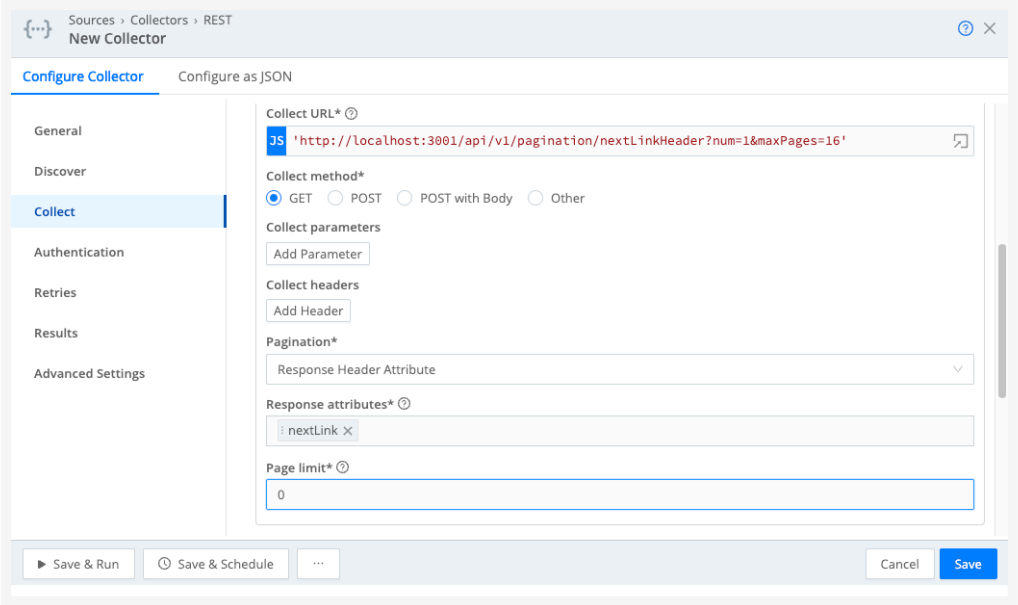
Pagination: Response Header Attribute

Response attributes: `nextLink`

Authentication: None

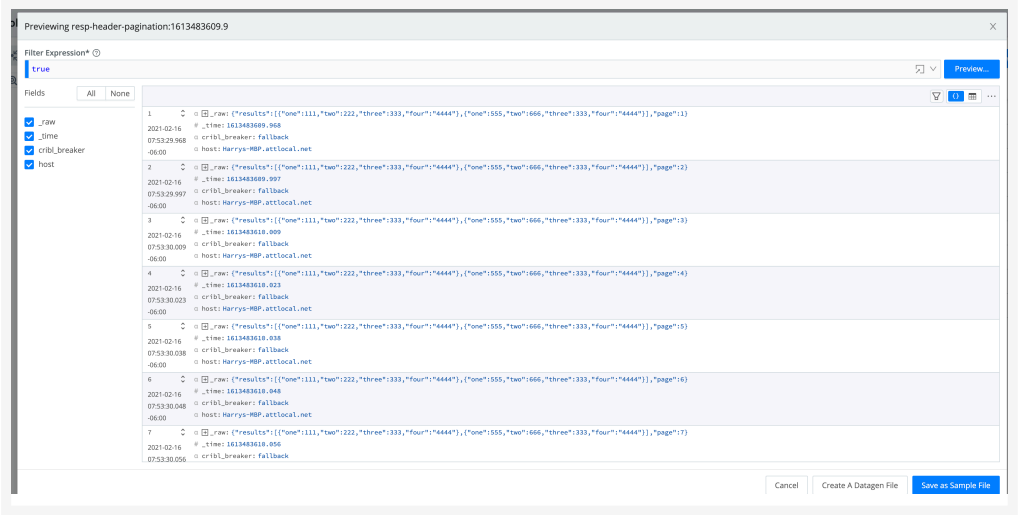
Event Breaker: None

You can modify the `maxPages` URL parameter to control how many pages this call returns.



Collector configuration for HTTP GET, paginated via Response Header URL

Collector Output



Paginated events

6. HTTP Discover and Collect with Login Authentication

In some cases, you must run an HTTP Request discovery to identify the items to collect. This example will do the following:

1. Perform a Login (POST with body containing the login credentials), to obtain an auth token that will passed in the Authorization header in all subsequent REST calls.
2. Run a REST call to discover items to be collected - in this case, log files.
3. For each log file discovered, collect the contents of that file.
4. We'll also demonstrate URL-encoding of a path element. You'd need to manually encode part of the URL in cases where unsafe ASCII characters might be present in the path element (for example, space, `$`, `/`, or `=`).

Discover type: `HTTP Request`

Discover URL: `'http://localhost:9000/api/v1/system/logs'`

Discover method: GET

Discover parameters: None

Discover headers: None

Discover data field: `items`

Collect URL: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)`

Collect method: GET

Collect parameters: None

Collect headers: None

Pagination: None

Authentication: Login

Login URL: `'http://localhost:9000/api/v1/auth/login'`

Login username: `admin` (or other user)

Login password: `admin` (or other user's corresponding password)

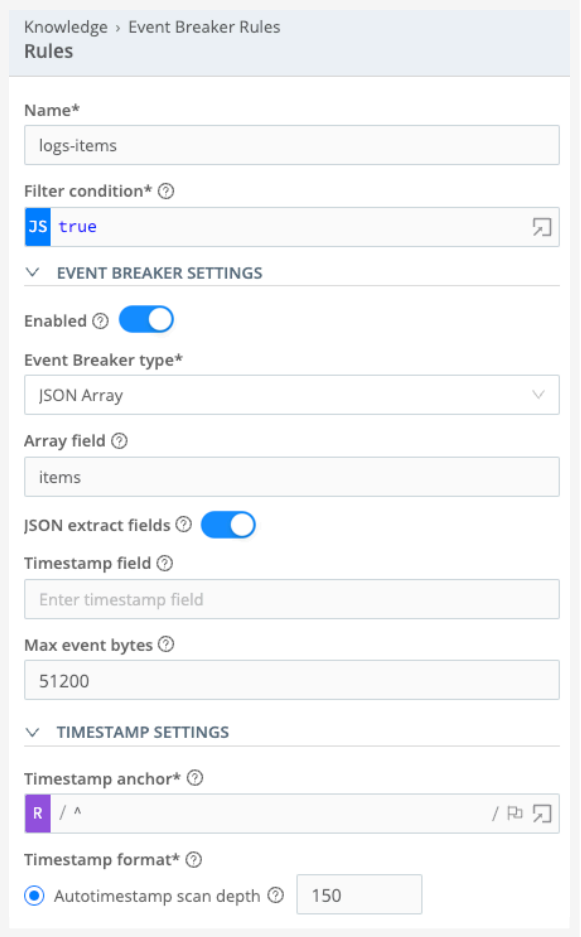
[Authentication] POST Body: `` `{ "username": "${username}", "password": "${password}" }` ``

Token attribute: `token`

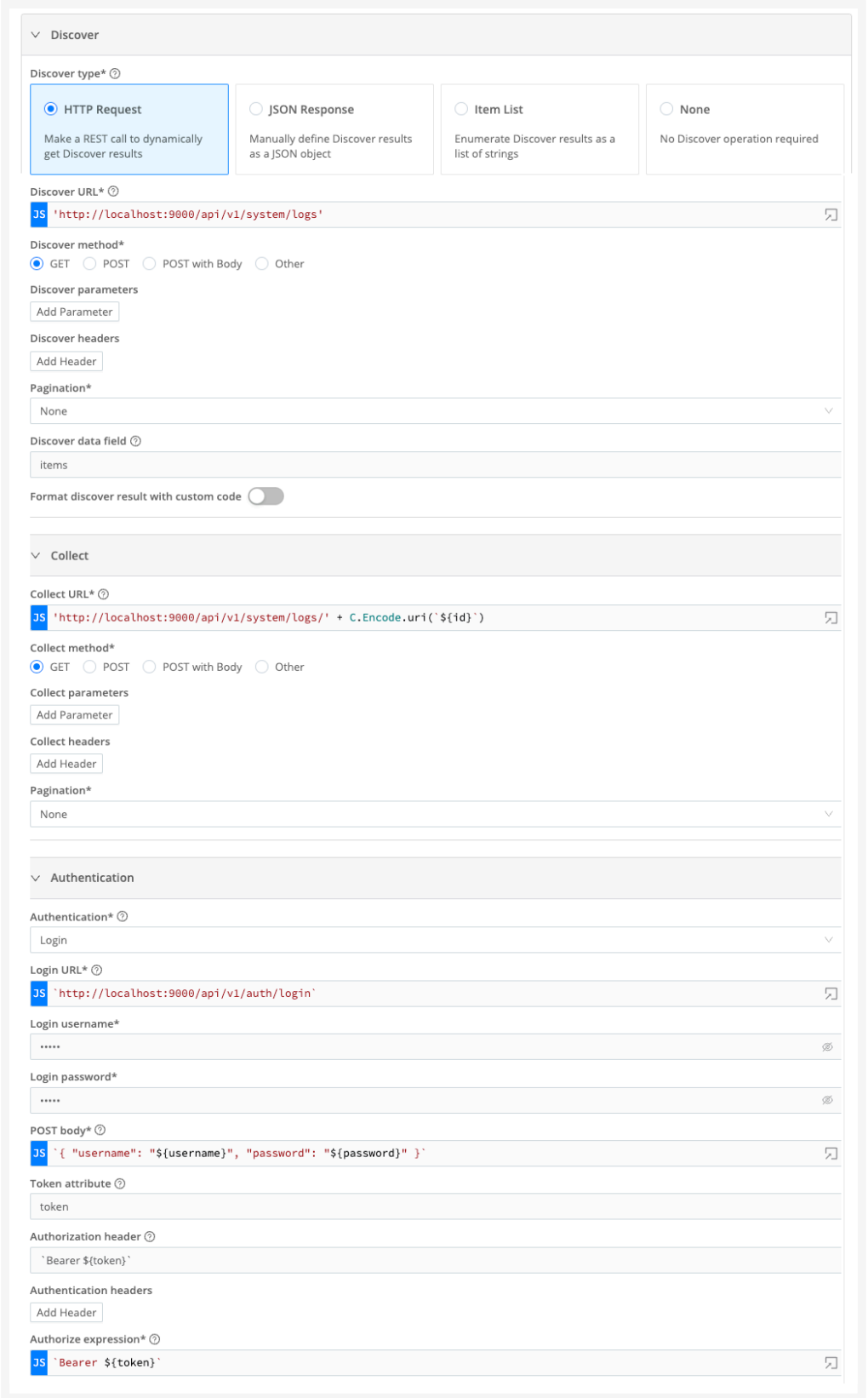
Authorize expression: `` `Bearer ${token}` ``

Event Breaker: JSON Array

- **Array Field**: `items.events`



Event Breaker Configuration (Discover and Collect with Login Authentication)



Collector configuration for HTTP Discover and Collect with Login authentication

Login

The login call sends a POST to the login URL, passing the string derived from the POST Body JavaScript expression. Note that the variables `${username}` and `${password}` are available to this call, and are taken from the `username` and `password` text fields.

Upon successful login (`200` response code), the login token will be extracted from the response body's token attributes, as specified by the **Token attribute** field.

Finally, the value derived from the **Authorize expression** field will be added to the Authorization header for all subsequent calls (here, both Discover and Collect). Set this to `${token}` to reference the token obtained from the login POST request.

Discover

The Discover call here is used to discover the list of log files that can be collected. The data returned by this call has this format:

```
{
  "count": 0,
  "items": [
        {
        "id": "logFileName",
        "path": "pathToFile"
        }
  ]
}
```

Copy

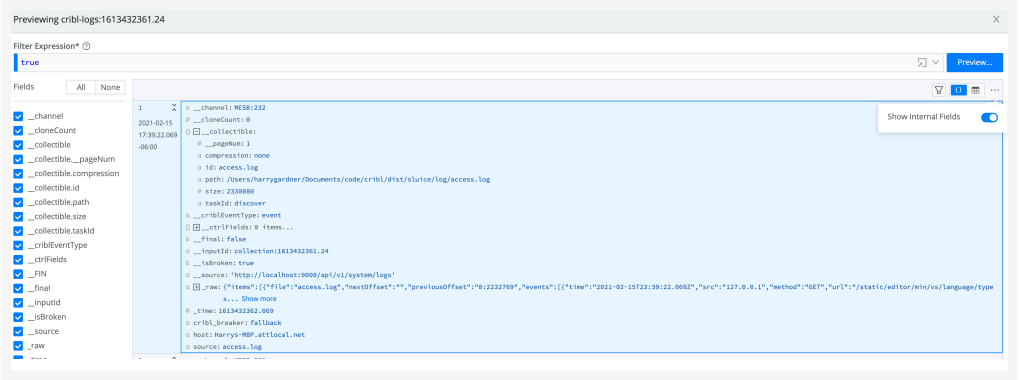
The **Discover Data Field** is used to define the array in Discover results that contains the list of items to discover. Here, each item is an object, with an attribute ID that is referenced in the Collect calls. So the Discover call generates a list of items for which Collect tasks will be created.

Collect

From the Discover task's returned list of items, each item will cause one Collect task to be created and run. An object containing the Discover item (along with some internal variables) will be passed to the Collect task.

You can reference this object's attributes as variables in the Collect task's URL, request parameters, and request headers. When running a preview, you can see the object's contents in the `__collectible` internal variable. (Enable **Show Internal Fields**, and expand `__collectible` to view the variables available).

For example, here's one of the events returned by this example's Collect operation. The `__collectible` attribute contains details identifying the page number and the URL used to obtain the data:



__collectible internal variable, expanded to show its contents

As you can see, `__collectible` contains a `__pageNum` variable, which shows which page of data the event was received in. Also, `__collectible` contains an `id` variable, available for use in the Collect operation. Here's how this variable is referenced in the Collect operation's URL:

```
'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)
```

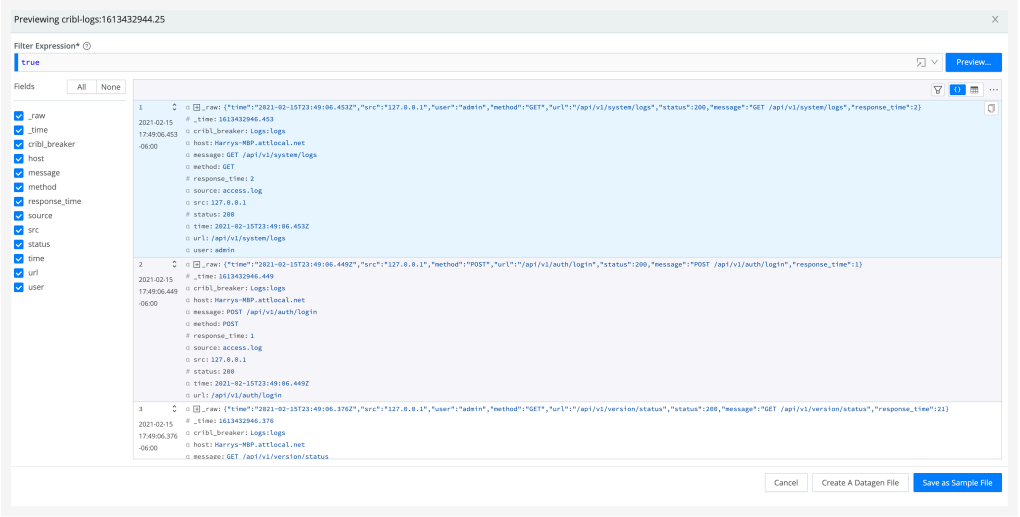
Because the variable is used in the path, and it might contain unsafe ASCII characters (specifically, space), we need to URL-encode the variable. This is the only case where the REST Collector requires URI encoding - variables that are defined directly as part of the URL. (Request parameters, not contained directly in the URL, are automatically encoded.)

The data returned by the Collect call has the following format:

```
{
  "items": [
        {
        "file": "access.log",
        "nextOffset": "",
        "previousOffset": "0:2236637",
        "events": [
        {
        "time": "2021-02-15T23:39:23.043Z",
        "src": "127.0.0.1",
        "user": "admin",
        "method": "GET",
        "url": "/api/v1/jobs/1613432361.24",
        "status": 200,
        "message": "GET /api/v1/jobs/1613432361.24",
        "response_time": 2
        },
        {
        "time": "2021-02-15T23:39:22.366Z",
        "src": "127.0.0.1",
        "user": "admin",
        "method": "GET",
        "url": "/api/v1/system/logs/worker%2F7%2Fcribl.log",
        "status": 200,
        "message": "GET /api/v1/system/logs/worker%2F7%2Fcribl.log",
        "response
...
```


Copy

The real data that we want to access is located at `items.events`. We can use a JSON Array event breaker to convert data from `events.items` into individual events that will be sent to Routes and processed by Cribl Stream. The output looks like this in Preview:



Collected data

If this example fails with errors of the form `statusCode: 429...Too many requests` - see Common Errors and Warnings to resolve this by relaxing the login rate limit.

7. Item List Discovery

This example demonstrates situations where the Item List discovery mechanism is useful: enabling collection based on a predefined list of items. Here, we want to collect weather information for a static list of states - each returned from Discover results as a single collection task.

Let's assume we are interested in weather for the following U.S. locations: Nashville, Dallas, and Denver. When the Discover operation runs, it will return a `__collectible` object for each location (each representing its own collection task): `{ id: ''}, {id: 'TX'}, {id: 'TN'}`.

Discover type: `Item List`

Discover items: `Nashville, Dallas, Denver`

Collect URL: `'https://community-open-weather-map.p.rapidapi.com/find'`

Collect parameters:

```
type: 'link'
```

```
units: 'imperial'
```

```
q: `${id}`
```

Collect headers:

```
x-rapidapi-host: 'community-open-weather-map.p.rapidapi.com'
```

`x-rapidapi-key: '78934c846cmsh70cb53f75a8a54bp119d21jsn29df549b4fd6'`

`useQueryString: true`

Pagination: None

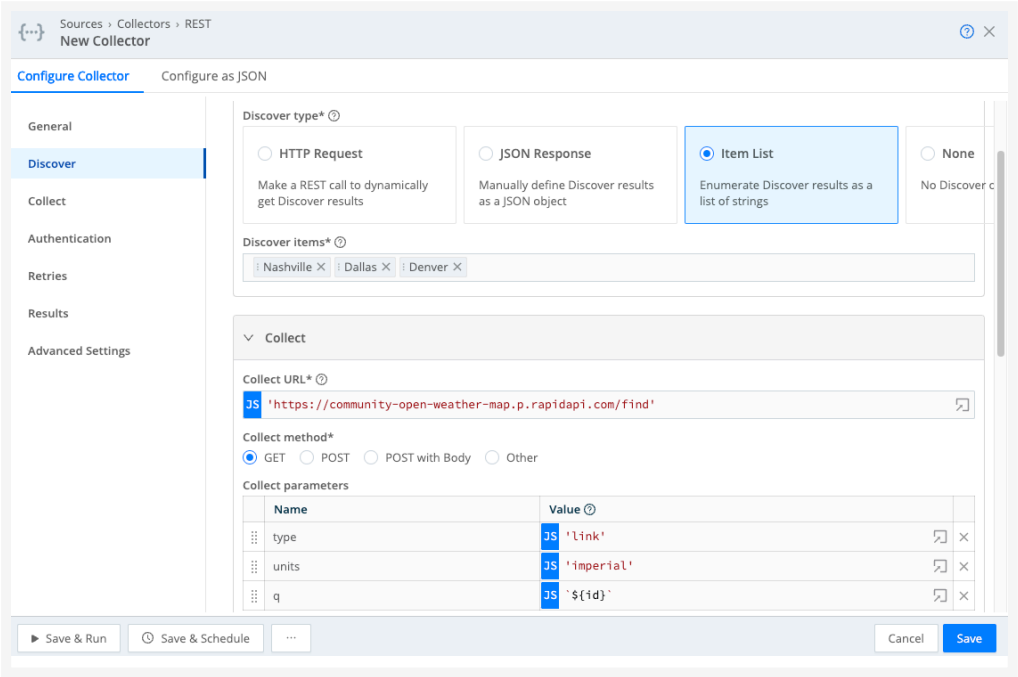
Authentication: None

Event Breaker: JSON Newline Delimited - Use a rule like **Cribl > ndjson** to parse each event and extract fields.

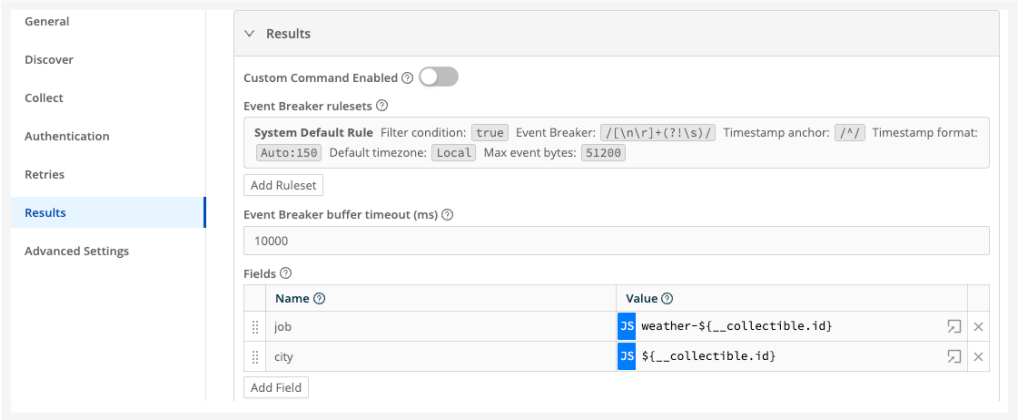
Fields:

`job: weather-${__collectible.id}`

`city: ${__collectible.id}`



Collector configuration for Discovery via Item List

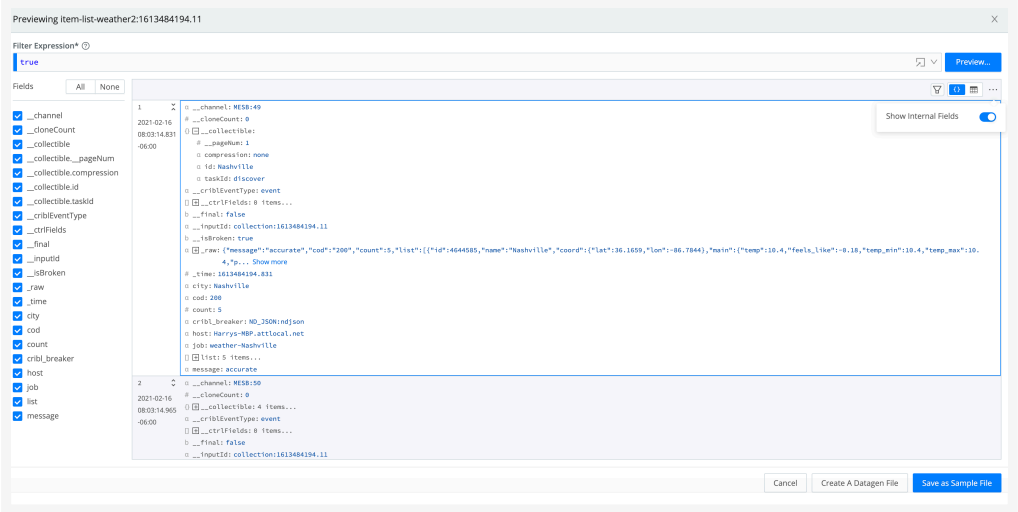


Fields configuration

Collector Output

One interesting thing about this example is the addition of **Fields** to each event, using content from the internal `__collectible` attribute. This `__collectible` attribute contains results from the Discover operation, and is available in each event collected.

This demonstrates how information from the Discover operation can be transferred to events generated during the Collect operation. Note the attributes `__collectible`, `city`, and `job` in the Collector output below:



Collected events

This API allows a certain number of calls/month. Cribl recommends that you not schedule this Collector - run it ad-hoc, for testing only.

8. JSON Response Discovery

Like Item List discovery, **Discover type: JSON Response** allows you to discover a predefined, static list of items. JSON Response's advantage is its ability to return an object containing more than one attribute that the Collect operation can use.

Sticking with our weather example above, imagine that we needed to use both longitude and latitude (instead of just city or state) when performing collection. This is the perfect use case for JSON Response discovery.

Discover type: `JSON Response`

Discover result: `{"items": [{"city": "Nashville", "lat": 36.174465, "lon": 86.767960}, {"city": "Dallas", "lat": 32.779167, "lon": -96.808891}, {"city": "Denver", "lat": 39.742043, "lon": -104.991531}] }`

Discover data field: `items`

Collect URL: `'http://api.openweathermap.org/data/2.5/weather'`

Collect headers: None

Collect parameters:

```
lat: `${lat}`
```

```
lon: `${lon}`
```

```
appid: '438d61a1db9e713240b30140e9ddfea2'
```

Pagination: None

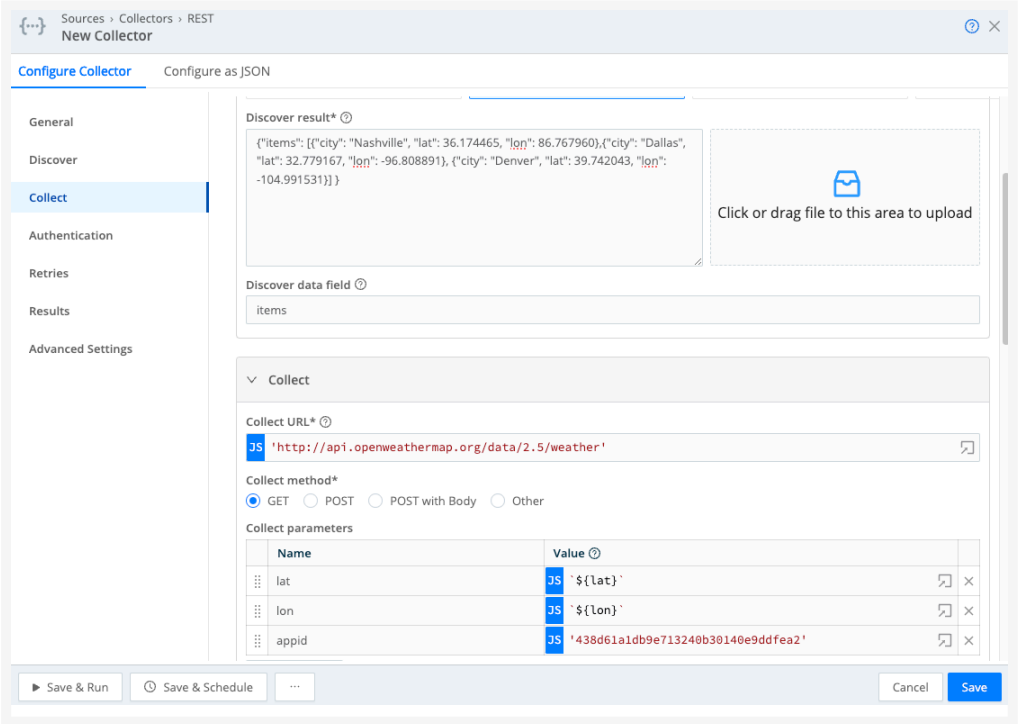
Authentication: None

Event Breaker: JSON Newline Delimited - Use a rule like **Cribl > ndjson** to parse each event and extract fields.

Fields:

```
job: `weather-${__collectible.city}`
```

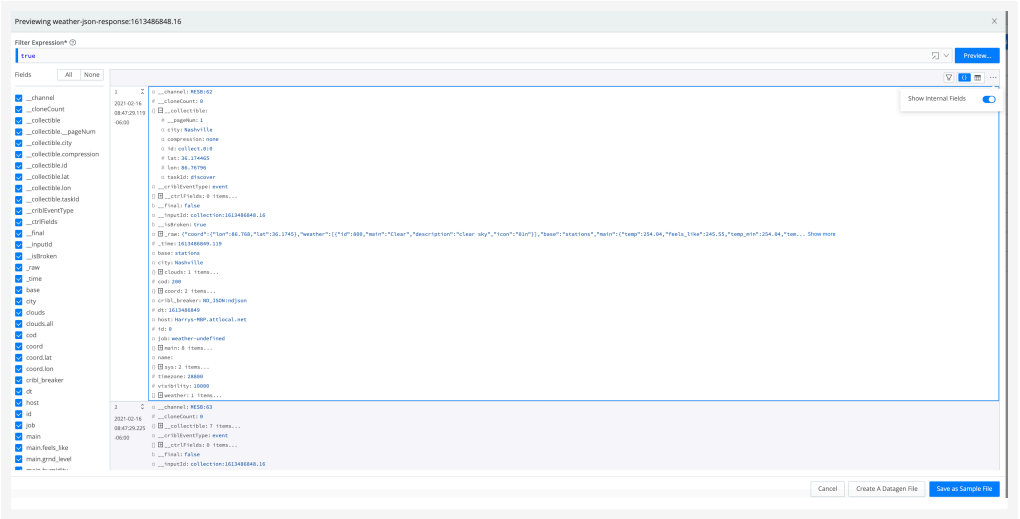
```
city: `${__collectible.city}`
```



Collector configuration for JSON Response Discovery

Notice how attributes present in the **Discover Result** JSON object's `items` array (`` `${lat}` ``, `` `${lon}` ``, `` `city` ``) are used in **Collect Request Parameters**, and in metadata **Fields**. Any other attribute present in the `items` array can similarly be referenced in the URL, request parameters, or request headers.

Collector Output



Item List preview

This API allows a certain number of calls/month. Cribl recommends that you not schedule this Collector - run it ad-hoc, for testing only.

9. HTTP Response Discover Result with Custom Code

Assume the Discover REST API returns a list of individual record IDs matching search criteria, such as:

```
`{ "ids": [1,2,3,4,5,6,7,8,...,49,50] }`
```

The collect API can accept a list of 1 to 10 individual IDs to collect data: `http://abc.com/collect?Ids=1,2,3,4,5,6,7,8,9,10`

By default, using the `ids` attribute from the Discover call results in 50 individual collect tasks, each retrieving data for a single ID. However, the API supports 1 to 10 IDs at a time. Using the **Format discover results** option, you can manipulate results to use 10 collect tasks, each retrieving data for 10 IDs. Here's how:

```
const pageSize = 10;
let arr = [];
__e['resultOut'] = []; // Store results in the original Discover results object.
__e['ids'].forEach(id => {
  if (arr.length === pageSize) {
    __e['resultOut'].push({ "ids": arr.join(',') });
    arr = [];
  }
  arr.push(id);
});
if (arr.length) __e['resultOut'].push({ "ids": arr.join(',') }); // Add last batch
```

Copy

Given 50 unique IDs, the output stores results in `resultOut`:

```
{
  "ids": [1, 2, 3, ..., 49, 50],
  "resultOut": [
    { "ids": "1,2,3,4,5,6,7,8,9,10" },
    { "ids": "11,12,13,14,15,16,17,18,19,20" },
    { "ids": "21,22,23,24,25,26,27,28,29,30" },
    { "ids": "31,32,33,34,35,36,37,38,39,40" },
    { "ids": "41,42,43,44,45,46,47,48,49,50" }
  ]
}
```

Copy

Set the **Discover data field** on the REST Collector to `resultOut` to use the array of 5 items during the collect phase, resulting in 5 collect tasks. The collect URL or request parameters can include the `ids` attribute from each array element: `http://abc.com/collect?Ids=${ids}`

10. State Tracking by Latest Time

REST Collector state tracking can help prevent both duplicate data and gaps in data for subsequent collection runs. To demonstrate how to configure a Collector to use state tracking, we'll access Stream's `/system/metrics` endpoint. This guide uses an out-of-the-box Stream configuration - you may need to update endpoints and authentication parameters to match your instance's configuration.

Configure the Event Breaker

For this example we'll use an Event Breaker from Cribl's Collector Templates repository. For more information on Event Breakers, check out the documentation.

1. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Event Breaker Ruleset**.
2. Select **Add Ruleset**.
3. Select **Manage as JSON** to open a text editor view.
4. Paste in the breaker config found here.
5. Select **OK**.
6. Select **Save**.

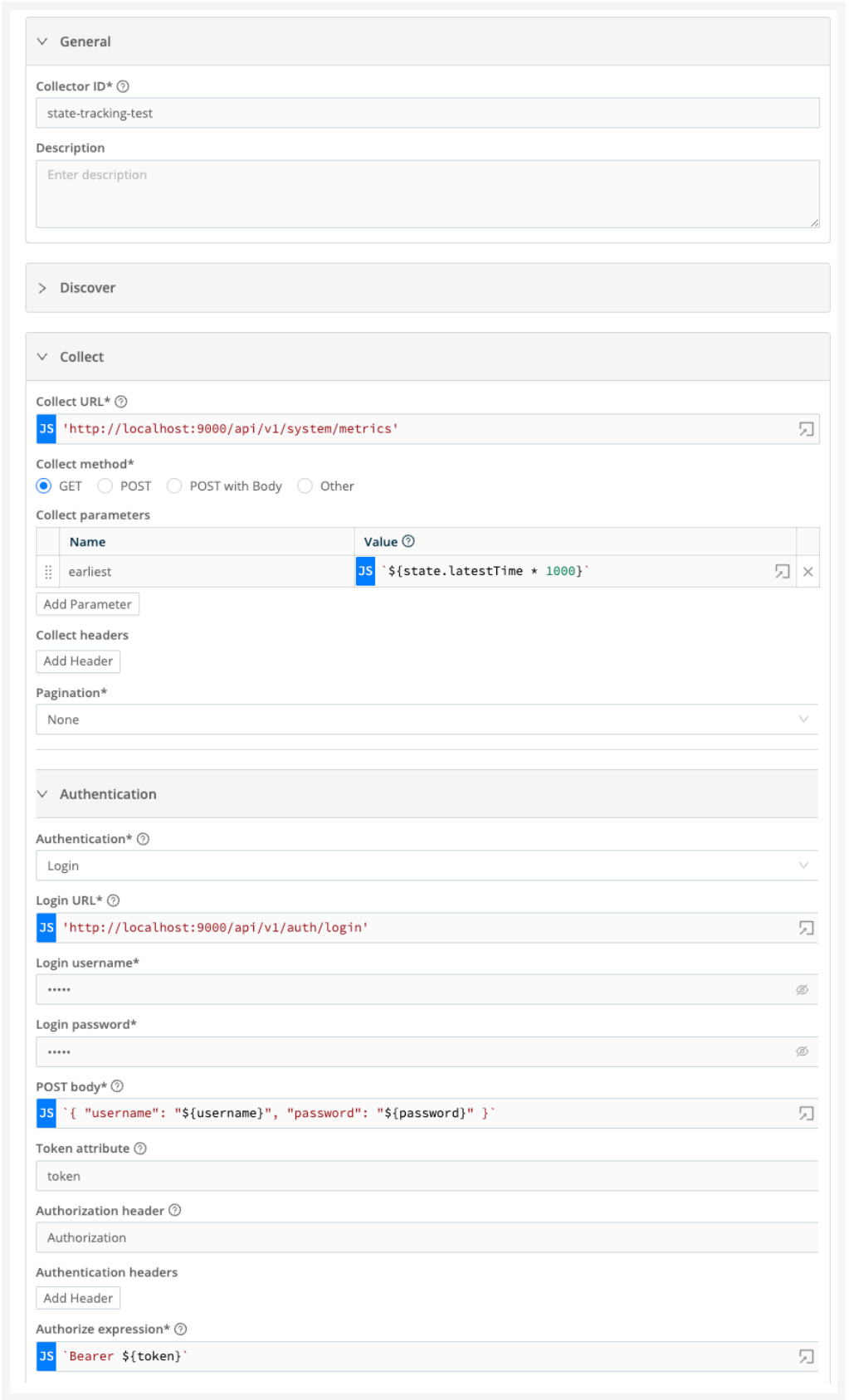
Configure the Collector

Set the following fields in the `Collector Settings` section

- **Collector ID**: `state-tracking-test`
- **Collect URL**: `'http://localhost:9000/api/v1/system/metrics'`
- **Collect parameters**:
  - **Name**: `earliest` | **Value**: `` `${state.latestTime * 1000}` ``
- **Authentication**: `Login`
- **Login URL**: `'http://localhost:9000/api/v1/auth/login'` (note that this is updated from the default `https` value)
- **Login username**: `admin`
- **Login password**: `admin`
- **Token attribute**: `token`

Select **Result Settings**, then **Event Breakers**. Then select **Add ruleset** and choose the `criblApi` ruleset from the earlier step.

At this point, your configuration should look like this:


General
Collector ID*
state-tracking-test
Description
Enter description
Discover
Collect
Collect URL*
JS 'http://localhost:9000/api/v1/system/metrics'
Collect method*
GET POST POST with Body Other
Collect parameters
Name
Value
earliest
JS `${state.latestTime * 1000}`
Add Parameter
Collect headers
Add Header
Pagination*
None
Authentication
Authentication*
Login
Login URL*
JS 'http://localhost:9000/api/v1/auth/login'
Login username*
Login password*
POST body*
JS `{ "username": "${username}", "password": "${password}" }`
Token attribute
token
Authorization header
Authorization
Authentication headers
Add Header
Authorize expression*
JS `Bearer ${token}`

8.1.1.14.4. Generate HMAC Functions For REST Collector Requests

HMAC (Hash-based Message Authentication Code) Functions are keyed hashing functions that you use with REST Collectors in Cribl, enabling you to ensure the integrity and authenticity of messages.

Practical Uses for HMAC Functions

HMAC Functions are used in REST Collectors. They are configured to generate a signature for each request, which is then included in the request headers. The server receiving the request can then verify the signature to ensure the request's integrity and authenticity.

- **Security**: HMAC Functions provide a way to verify the integrity and authenticity of messages, ensuring that data has not been altered in transit.
- **Authentication**: They help authenticate the source of the message, ensuring that it comes from a trusted entity.
- **Compliance**: Using HMAC Functions can help meet security compliance requirements by adding an additional layer of security to your data collection processes.
  For a practical example of how to use HMAC functions, refer to the Cribl REST Collector Configuration for the Duo API in Cribl's Collector Templates repository.

Create an HMAC Function

You can create an HMAC Function in the Knowledge library:

1. In the sidebar, select **Worker Groups** and choose a Worker Group.
2. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **HMAC Functions**.
3. Select **Add HMAC Function**.
4. Fill out the **New HMAC Function** form:

Field	Description	Notes
ID	A unique identifier for this HMAC Function.	None
Description	A description.	Optional

Field	Description	Notes
Tags	Used to filter and group HMAC Functions.	Optional
HMAC signature strings	A list that Cribl evaluates and concatenates to form the HMAC signature string. This specifies the components on the request to include in the HMAC signature.	You might include the HTTP method, URL, and certain headers to ensure that the signature is unique and secure. Expressions returned as undefined or null are ignored and not added to the final signature string. Each string can access the following variables: ◦ `method`: Uppercase HTTP verb used for the request: GET, PUT, POST, DELETE, and so on. ◦ `urlObj`: A standard JS URL object containing details of the URL and query string passed with the request, along with object properties. ◦ `headers`: An alphabetically ordered list of headers (by name) and values to send with the request. You can access the header elements by name, as in a JS array. ◦ `body`: The contents of the HTTP body (if present) passed with the request.
Signature string delimiter	A character to use when you join the HMAC signature strings for evaluation.	For example, `':'`, `'-'`, and `'\n'`. The value must be a JavaScript expression (which can evaluate to a constant value), enclosed in quotes or backticks. Leave blank to join without spaces.
Authorization header name	Contains the HMAC signature. Defaults to `signature`. This ensures the server knows where to find the signature in the request headers.	None
Authorization header expression	Forms the HMAC signature.	The expression should incorporate the following: ◦ `signatureString`: This contains the concatenated result of the HMAC signature strings. ◦ HMAC generation: Use the `C.Crypto.createHmac` function to calculate the HMAC signature based on the `signatureString`, a specified secret (either as a text secret or inline), the SHA256 algorithm, and hexadecimal encoding.

Field	Description	Notes

		For example:
		`${C.Crypto.createHmac(`${signatureString}`,`
		`C.Secret('myTextSecret', 'text').value`

8.1.1.14.5. Lacework API Collection

You can configure the Lacework v2 API within Stream. This enables you to collect data from the Lacework API without introducing custom scripts or add-ons into your Lacework environment.

Your workflow will be based on the Discover and Collect pattern that's standard for REST/API Collectors. In the Lacework API variation described here, the Discover job will generate the access token that the collection job uses in requests to the API. As an example, we'll configure Cribl Stream to collect Host Vulnerability data, using optional filters available in the Lacework API.

Before you begin, make sure you have a Lacework API `KeyID` and `secretKey`, or create them as described in the Lacework docs.

Configuring the REST Collector

Collector Sources currently cannot be selected or enabled in the QuickConnect UI.

For additional details about all the configuration options specified here, see our REST/API Endpoint and Scheduling and Running topics.

1. Navigate to **Products** > **Stream** > **Worker Groups**.
2. Select a Worker Group, then go to **Data** > **Sources**. Choose the Collector and select **Add Collector**.
3. Enter a **Collector ID**, then complete the following options and fields.

Discover Settings

From the **Discover type** drop-down, select `HTTP Request`. Then complete the **Discover** settings as follows.

Discover URL: Enter the URL at which Cribl Stream should access the Lacework `v2/access/tokens` endpoint.

Discover method: From the drop-down, select `POST with Body`.

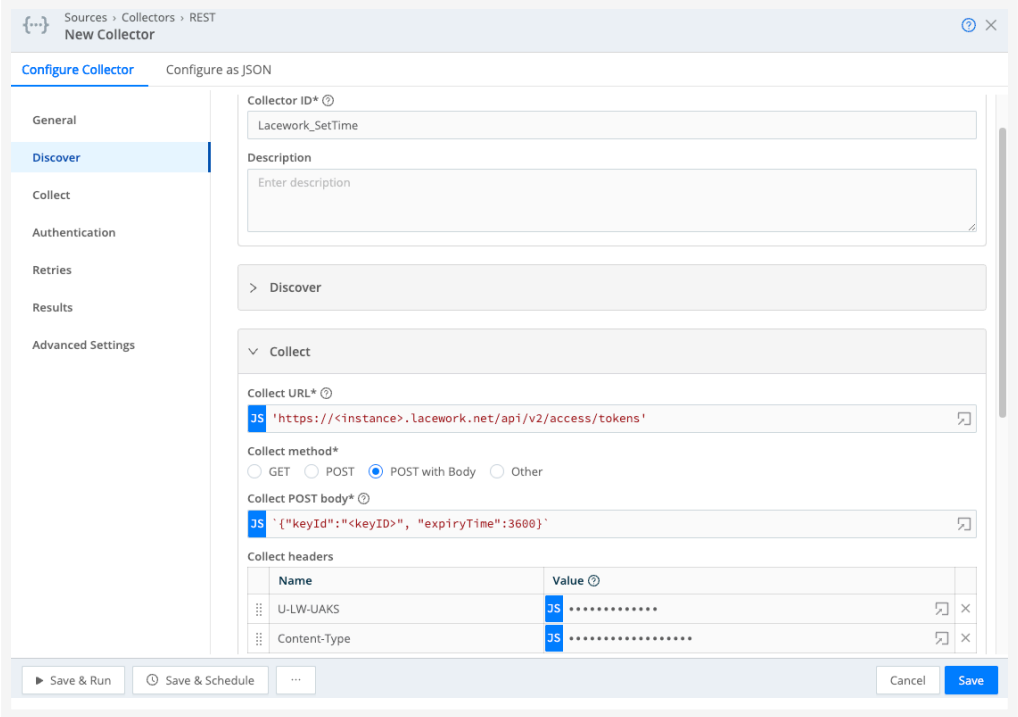
Discover POST Body: Enter `` `{"keyId":"<keyID>", "expiryTime":3600}` ``, substituting your Lacework API `KeyID` for the placeholder.

Discover headers: Create the two headers specified in the table below, substituting your Lacework API `secretKey` for the placeholder.

Name	Value
`U-LW-UAKS`	`'<secretKey>'`

Name	Value
`Content-Type`	`'application/json'`

Your Collector configuration should look similar to this:



Discover Settings

Collect Settings

Collect URL: Enter the URL at which Cribl Stream should access the Lacework `v2/Vulnerabilities/Hosts/search` endpoint.

Collect method: From the drop-down, select `POST with Body`.

Collect POST body: Enter the following request body.

```
`{"timeFilter": { "startTime": "${C.Time.strftime(earliest || new Date().getTime()/1
```

Copy

Here's what's happening in the request body example:

In our example, we've chosen to collect data only about hosts where one particular vulnerability, `CVE-2018-11233`, was detected. To accomplish this, we've added a `filters` element that specifies that the value of the `vulnId` field must equal `CVE-2018-11233`.

Note the `earliest` and `latest` variables. Later, in the **Run Collector** modal, you'll set **Time Range** values that will populate these variables when the Collector runs. Both variables are formatted as Unix time, in seconds units. (When using them in contexts that require milliseconds resolution, multiply them by 1,000 to convert to ms.) If you omit these variables, jobs will run for a period of 24 hours.

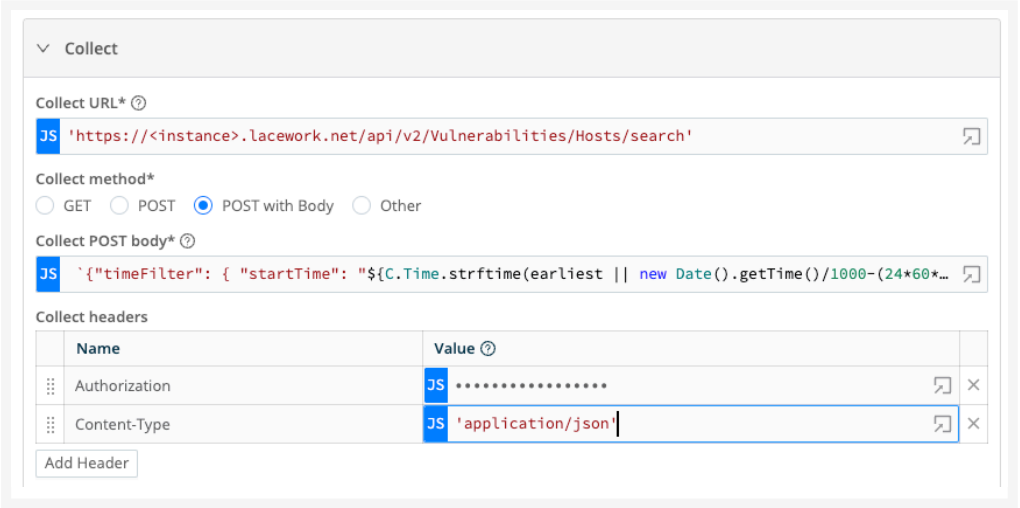
Collect Headers

We know that the Discover job, which we will run before the Collect job, will make a request to the Lacework API `v2/access/tokens` endpoint, and that the Lacework API's response body will include an access token. That means that the access token will be available to the Collect job, to pass in a Collect header.

Create the two **Collect header**s specified in the table below. The first will convey the access token.

Name	Value
`Authorization`	`` `Bearer ${token}` ``
`Content-Type`	`'application/json'`

Your **Collect** settings should look similar to this:

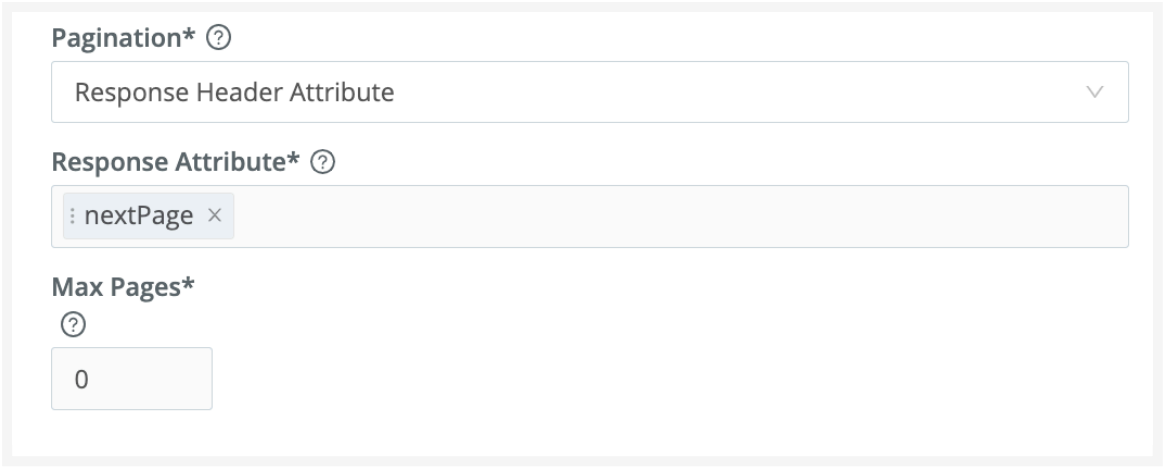


Adding a Collect header

Pagination

From the **Pagination** drop-down, select `Response Header Attribute`. In the **Response Attribute** field, enter `nextPage`. This configures your Collector to work with the Lacework API response body, which includes nested fields of pagination metadata, such that `urls` contains `nextPage`, whose value is the Next Page URL.

You should see something like this:



Configuring pagination

The Lacework API authentication mechanism requires HTTP header parameters. Since Cribl Stream does not (currently) support header parameters for authentication, we cannot use that authentication method, and you should skip the **Authentication** settings. This is why we use the Discover job to obtain an access token, and then include that token in the Collect job.

Additional Settings

Tags: Optionally, add tags that you can use for filtering and grouping in Cribl Stream. Use a tab or hard return between (arbitrary) tag names.

Result Settings

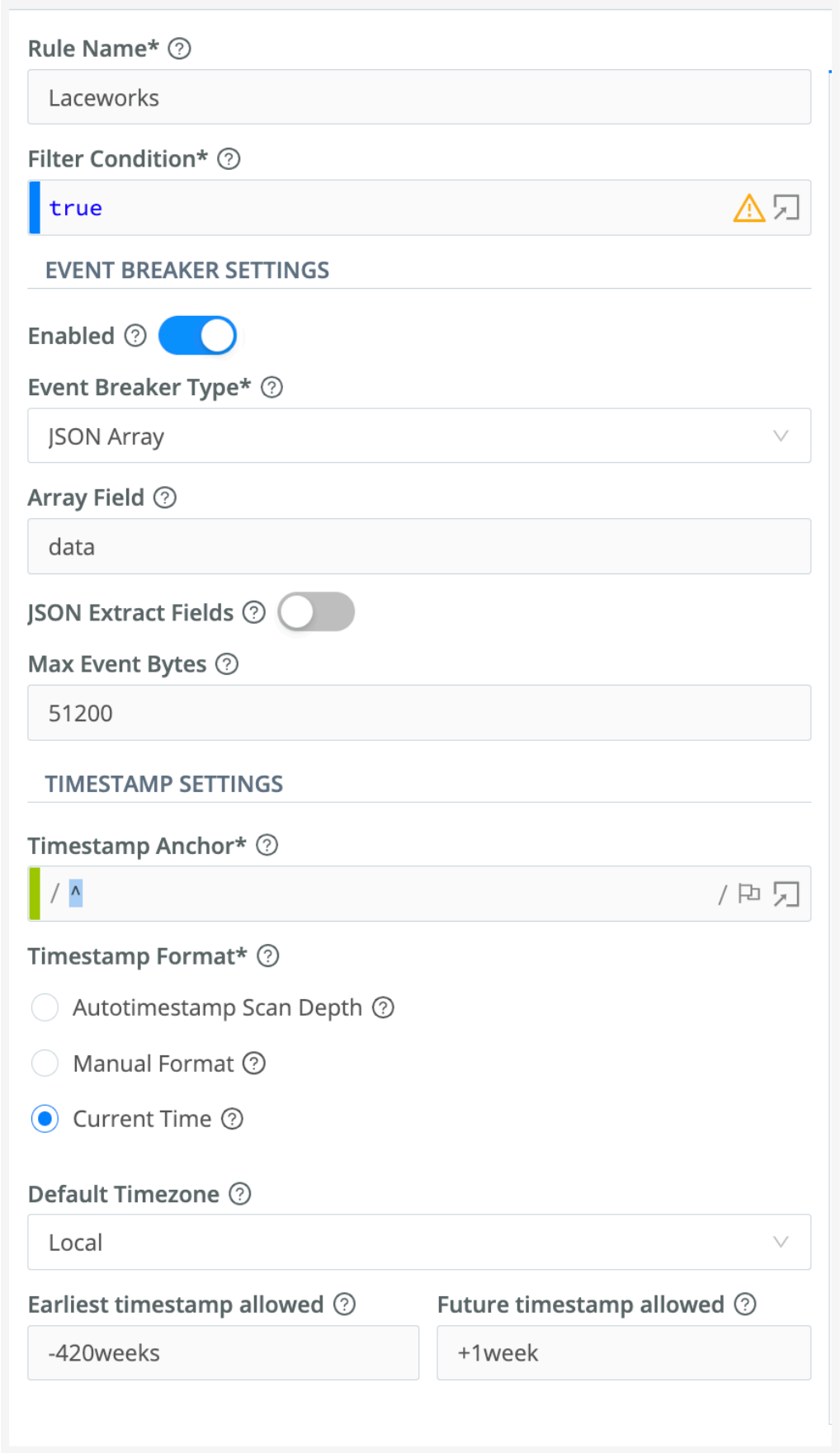
Every call to the Lacework API `v2/Vulnerabilities/Hosts/search` endpoint returns data formatted as a single event. Every event contains multiple nested JSON arrays, where each array is a `data` element. We'll now create an Event Breaker that parses the larger structure into individual events - one for each `data` element.

1. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Event Breaker Rules**.
2. From the resulting **Event Breaker Rulesets** form, select **New Ruleset** to open the **New Ruleset** modal.
3. Enter an **ID** for the new ruleset, and optionally add a **Description** and **Tags**.
4. Select **Add Rule**.

In the resulting **Rules** modal, name the Rule, and configure it as follows:

- **Filter Condition**: Enter `true`.
- **Enabled**: Toggle on.
- **Event Breaker Type**: Select `JSON Array`.
- **Array Field**: Enter `data`.

Optionally, you can use the **Timestamp Settings** to return events whose timestamp (`_time`) matches the **startTime** or **endTime** field defined in the Lacework data.



Creating a ruleset

Select **OK** to return to the previous **New Ruleset** modal, then select **Save**.

Then return to your REST Collector's configuration modal, and:

1. Select the **Results** section.
2. Under **Event Breaker rulesets**, select **Add Ruleset**.
3. From the drop-down that now appears above the **System Default Rule**, select your new ruleset.
4. Select **Save**.

Discovering and Collecting

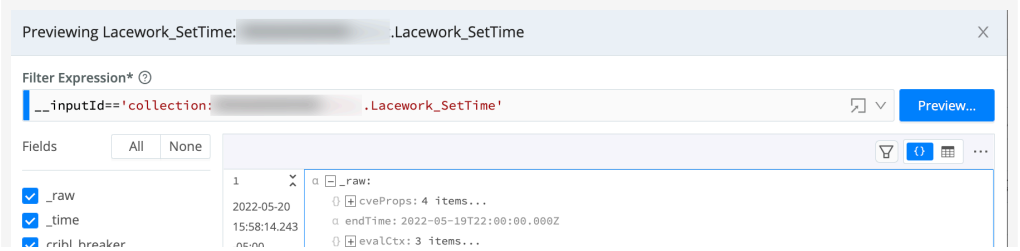
We'll start with the Discovery run:

1. On the **REST** page, select **Run** beside your new Collector.
2. In the resulting **Run Collector** modal, select **Mode** > **Discovery**.
3. Configure a **Time range**, if desired.
4. Select **Run** to retrieve Discovery results.

After inspecting these results, launch the Collector run:

1. Back on the **REST** page, again select **Run** beside your new Collector.
2. In the resulting **Run Collector** modal, this time select **Mode** > **Full Run**.
3. Configure a **Time range**, if desired.
4. Select **Run**.

Once the Lacework API responds, you should see data similar to this:


Previewing Lacework_SetTime: .Lacework_SetTime
Filter Expression*
__inputId=='collection: .Lacework_SetTime'
Preview...
Fields
All
None
_raw
_time
cribl_breaker
1
2022-05-20
15:58:14.243
_raw:
cveProps: 4 items...
endTime: 2022-05-19T22:00:00.000Z
evalCtx: 3 items...

8.1.1.14.6. Microsoft Graph API Collection

The Microsoft Graph Source is now available, so you no longer need to use a REST Collector to pull Message Trace data via the Microsoft Graph API.

The Microsoft Graph API provides access to data in the Microsoft Cloud. This page explains how to configure a Cribl Stream REST/API Collector to ingest data using the Microsoft Graph API.

Before you start, you'll need to do the following in the Azure portal:

- Register the app you'll use to interact with Graph API.
- Generate a **Client Secret** for the app.
- Write down the **Application ID**, **Tenant ID**, and **Secret Value** defined for the app.
- Decide which API calls you expect to use, then, for each of those API calls:
    - Configure the corresponding API permissions for your app.
    - Examine how the Graph API structures events, and decide whether you want to change that structure using an Event Breaker.

For example, if you plan to call the Audit Logs API (as demonstrated in this walkthrough), do the following:

- Configure your app with `AuditLog.Read.All` API permissions, which Audit Logs API calls require.
- Notice that the API structures Microsoft Entra ID Audit Logs data as a single JSON object with events nested under the `value` field. With an Event Breaker, you can split this into individual events for each AD Audit Log activity.

How the REST Collector Interacts with Microsoft APIs

To retrieve data using the Microsoft Graph API, your Collector first obtains a Bearer token by sending an HTTP `POST` request to the Microsoft identity platform. Once it has the Bearer token, your Collector can send an HTTP `GET` request to the Graph API, which responds with the data you requested. Configuring your Collector will be more intuitive if you keep this pattern in mind.

Configuring a Graph API REST Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**.
2. Select a Worker Group, then go to **Data** > **Sources**. Choose the Collector and select **Add Collector**.

The sections described below are spread across several tabs. Select the tab links to navigate among tabs. Select **Save** when you've configured your Collector.

Collector Settings

These settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. Such as `ms_graph_42`.

Collect Settings

Collect URL: An expression that produces a URL for the collect operation. The URL can be any of the Graph API endpoints. In this example, we want to retrieve Microsoft Entra ID Audit logs, so we use the Audit Logs endpoint:

`` `https://graph.microsoft.com/v1.0/auditLogs/directoryAudits` ``

Collect method: `GET`.

Collect headers: Add the following header.

- **Name**: `content-type`.
- **Value**: `application/x-www-form-urlencoded`

To authenticate against the Microsoft Graph API, the REST Collector uses a POST call whose body is a template. The `content-type` header tells the Graph API that the template is a URL-encoded (not JSON-encoded) form. See **Authentication Settings** > **POST body** below.

Authentication Settings

Use the **Authentication method** buttons to select **Login**, then enter the following information:

Login URL: The token API endpoint for the Microsoft identity platform. Use the following string substituting your Microsoft Entra ID tenant ID for `<tenant_id>`:

```
`https://login.microsoftonline.com/<tenant_id>/oauth2/v2.0/token`
```

Copy

Login username: The **Application (Client) ID** listed on your app's **Overview** page in Azure.

Login password: The **Secret Value** listed on your app's **Certificate & Secret** page in Azure.

POST body: The template for the token request that the Collector sends to the Microsoft identity platform. Use the following string:

```
`client_secret=${password}&scope=https://graph.microsoft.com/.default&client_id=${use
```

Copy

When the Collector sends the request, it will populate the template with the **Username** and **Password** that you entered above.

Token attribute: The field that contains the Bearer token, within the Microsoft identity platform's response to the token request. Use `access_token`.

Authorize expression: The JavaScript expression that computes the `Authorization` header the Collector uses in calls to the Graph API. Use the default `` `Bearer ${token}` ``. The Collector will populate the `${token}` variable with the token obtained from the Microsoft identity platform.

Result Settings

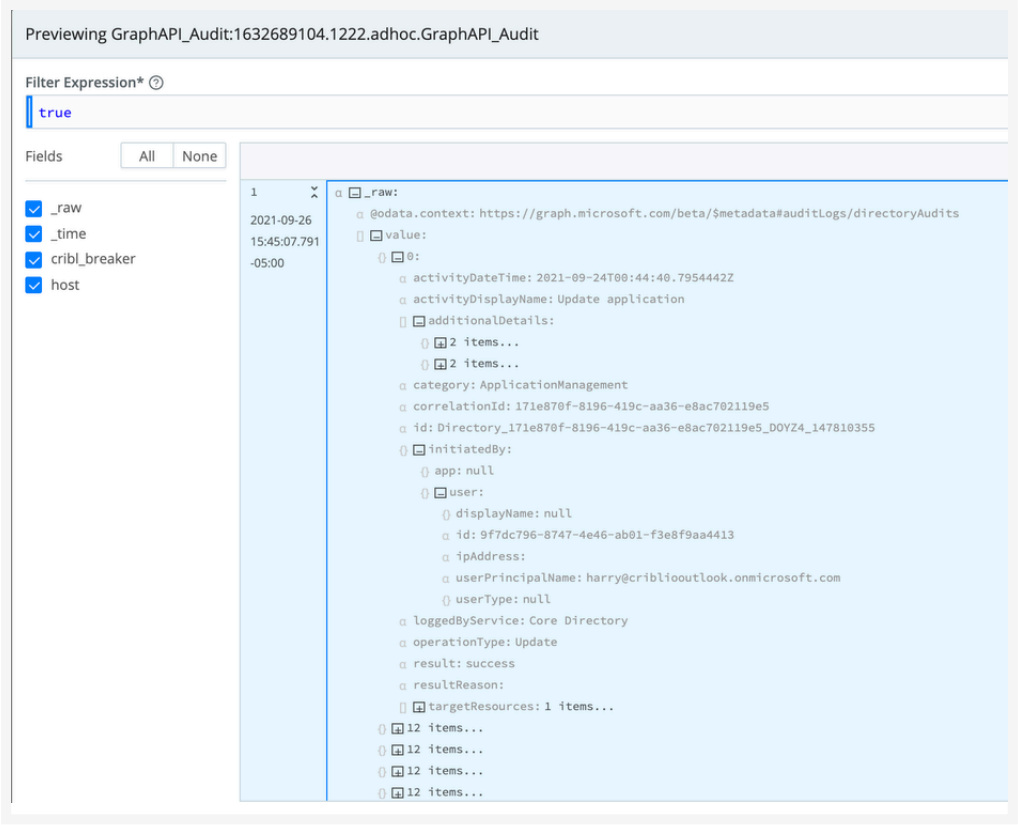
As explained above, the Microsoft Graph API delivers Microsoft Entra ID Audit Logs data as a single JSON object, with events nested under the `value` field. Cribl recommends using an Event Breaker to split this into separate name/value pairs.

You'll create the Event Breaker in the **Results** section. But first, you must save a sample file, and create a ruleset for the Event Breaker to use.

Saving a Sample File

On the **REST** page, select **Run** beside the REST Collector you configured. This opens the **Run configuration** modal, in **Preview** mode. Select **Run** to open the **Capture Sample Data** modal.

When data appears, notice that it's a large JSON object with multiple events nested under the `value` field:



Audit Logs data in original form

Select **Save as Sample File** and note the file name and location.

Creating an Event Breaker Ruleset

1. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**, then **Event Breaker Ruleset**.
2. Select **Add Ruleset**.
3. Enter an ID and a Description for the ruleset, then select **Rules** > **Add Rule** to open the **Rules** modal.

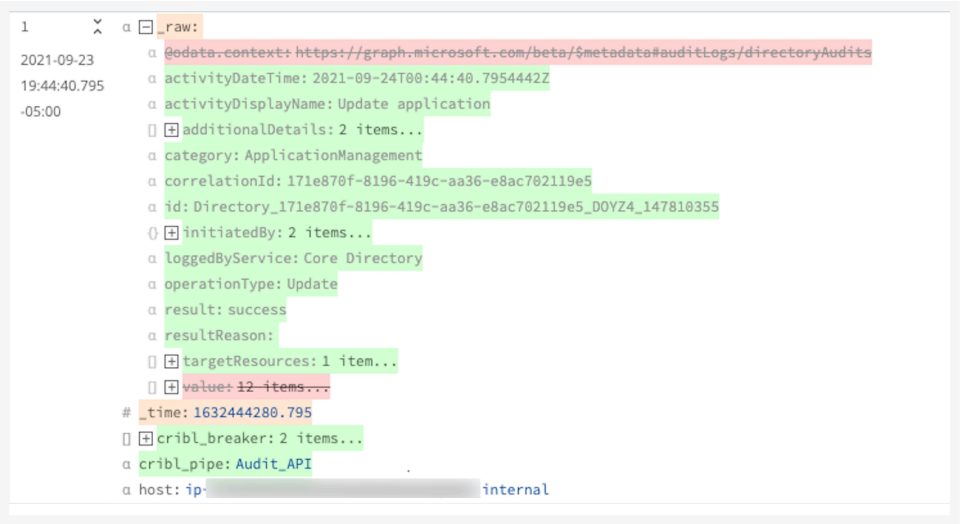
Here's what we want the ruleset to do:

1. Remove the leading `@odata.context` statement. For this example, we'll assume that OData is not important for your use case.
2. Structure each of the fields nested within the `value` element as a separate event, where the value of the `activityDateTime` field becomes the value of the `_timestamp` field.

Name the ruleset, then:

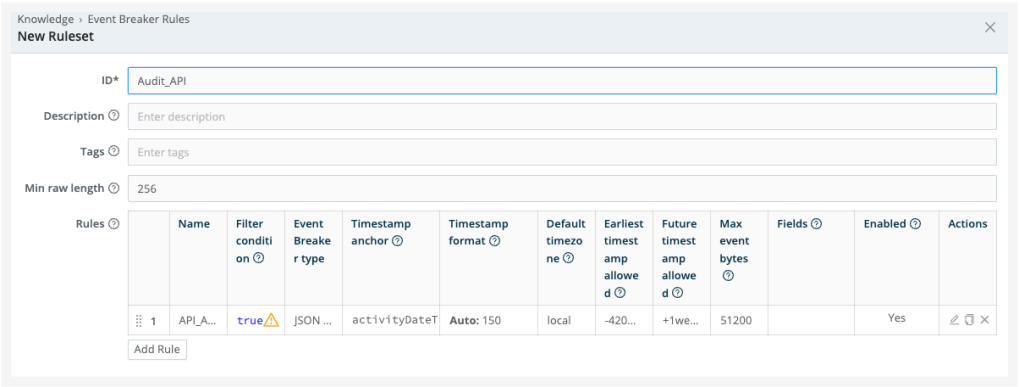
- From the **Event Breaker type** drop-down, choose **JSON Array**. This detects that the `value` element is a JSON array, and breaks it into key-value pairs. It also discards the `@odata.context` element, because that falls outside the detected JSON array.
- In the **Timestamp anchor** field, enter `activityDateTime` between the slashes.

- Upload your sample file.
- Compare the **In** and **Out** panes. You should see output similar to this:



Audit Logs data broken into individual events

- Select **OK** to return to the **New Ruleset** modal.
- You should see a result similar to the screenshot below. Select **Save**.



Creating an Event Breaker Ruleset for Audit Logs data

Event Breakers

In the Collector's **Results** section:

- From the **Event Breakers** drop-down, select the ruleset that you created and saved.
- Select **Save**.

Run the Collector again. Now you should see an individual JSON-formatted event for each AD Audit Log activity.

8.1.1.14.7. ServiceNow API Collection

This topic covers how to configure Cribl Stream REST Collectors to gather data via ServiceNow (SNOW) REST APIs and then enrich the data using Pipelines and the Redis Function.

From among the 100-or-so SNOW REST APIs, we've chosen the CMDB Instance for this tutorial. You can adapt this material for use with other SNOW REST APIs. You'll need to create a separate Collector for each SNOW API you connect to.

The prerequisites for this setup are a ServiceNow instance, which you can obtain here, and the URL for the Redis store you wish to use.

Using the Discover and Collect Pattern

A common pattern in REST APIs is to expose two related endpoints:

- One endpoint takes a category identifier as a URL parameter, and returns a list of instances of the category, with an individual identifier for each item in the list.
- A second endpoint takes the instance identifier as a URL parameter, and returns full details about the instance.

The SNOW CMDB Instance REST API follows this pattern. The API is called "CMDB" because it exposes a Configuration Management Database which contains records that describe how pieces of hardware are configured. The categories for different kinds of hardware are specified by "classnames," and the description of an individual piece of hardware is called a Configuration Item (CI) record. The relevant API calls look like this:

- The `GET /now/cmdb/instance/{classname}` call takes a `classname` (for example, `cmdb_ci_appl`, `cmdb_ci_linux_server`, `cmdb_ci_apache_web_server`) as a URL parameter, and returns a list of instances of the classname. Each instance has a `sys_id` and a `name`.
- The `GET /now/cmdb/instance/{classname}/{sys_id}` call takes a `sys_id` as a URL parameter, and returns the detailed CI record for the instance that the `sys_id` refers to.

In this tutorial, you'll create a Collector that uses the first API call to **discover** the systems for a given classname. Then you'll iterate the second API call over the list of `sys_ids` we discovered, to **collect** the CI records of interest. This **discover and collect** pattern works for any Cribl Stream Collector.

You'll have the Collector store the CI data in Redis. Then whatever Source you choose that has events to enrich, can do that by pulling the collected CI record data from Redis. That's the goal: enriching data.

How might this be useful? Here's one scenario: Suppose you have log data in which server names appear. Assuming that the servers in question have records in your CMDB, you can enrich the events with data about the server, such as its `sys_id` or the name of the user who updated it last.

Preparing Pipelines

Before creating the SNOW Collector itself, you'll set up the two Pipelines that the Collector needs. Both pipelines use the Cribl Stream Redis Function.

- The **process** Pipeline attaches to the Collector and performs a `set` to add events to Redis.
- The **enrichment** Pipeline performs a Redis `get` to retrieve the values with which we'll enrich events in the Collector.

Create the Process Pipeline

Create a Pipeline called `SNOW_Instance_Process` with the following Functions and values:

Parser Function

- **Operation Mode**: `Extract`
- **Type**: `JSON Object`
- **Source Field**: `_raw`

Note that each event this Function will parse is the reponse body from a `GET /now/cmdb/instance/{classname}/{sys_id}` call–i.e., a `result` object. That object has an attribute called `attributes` whose value is itself an object containing dozens of key-value pairs. To enrich our data, we'll treat one of those (`name`, meaning server name) as a key whose value will be two more (`sys_id` and `sys_updated_by`), comma-separated. We'll specify all three as **Evaluate Fields** in the next Function.

Eval Function

Evaluate Fields:

Name	Value Expression
`name`	`result.attributes.name`
`sys_id`	`result.attributes.sys_id`
`sys_updated_by`	`result.attributes.sys_updated_by`

Redis Function

The Redis function stores our fields in the key/value relationship described earlier.

- **Command**: `set`.
- **Key**: `` `${name}` ``.

- **Args**: `` `${sys_updated_by},${sys_id}` ``.
- **Redis URL**: The URL for your Redis store, for example, `redis://10.0.0.1:6379`.

Create the Enrichment Pipeline

Create a Pipeline called `SNOW_Instance_Enrich` with the following Functions and values:

Redis Function

- **Command**: `get`.
- **Result field**: `Args`.
- **Key**: `` `${name}` ``.
- **Args**: Leave blank.
- **Redis URL**: The URL for your Redis store, for example, `redis://10.0.0.1:6379`.

Parser Function

- **Operation Mode**: `Extract`.
- **Type**: `CSV`.
- **Source Field**: `Args`.
- **List of Fields**: `sys_updated_by sys_id`.

Eval Function

- **Remove Fields**: `Args`.

Configuring a SNOW Collector

1. Navigate to **Products** > **Stream** > **Worker Groups**.
2. Select a Worker Group, then go to **Data** > **Sources**. Choose the Collector and select **Add Collector**.

The sections described below are spread across several tabs. Select the tab links to navigate among tabs. Select **Save** when you've configured your Collector.

Collector Settings

These settings determine how data is discovered and collected before processing. You'll see the power of the **discover and collect** pattern here: from the results of a single Discover API call, we will generate an entire set of

Collect API calls. There is more about this pattern in the REST Collector documentation.

Collector ID: Unique ID for this Collector. For example, `snow_42-a`.

Discover Settings

Discover type: `HTTP Request`

Discover URL: An expression that produces a URL for the discover operation. We can enter this URL as a constant, for example:

```
'https://dev111111.service-now.com/api/now/cmdb/instance/cmdb_ci_appl'
```

Discover method: `GET`

Discover data field: `result`

We specify `result` for the **Discover data field** because that is the key of the JSON array that our Discover call retrieves. For example:

```
"result": [
  {
    "sys_id": "3a290cc60a0a0bb400000bdb386af1cf",
    "name": "PS LinuxApp01"
  },
  {
    "sys_id": "3a5dd3dbc0a8ce0100655f1ec66ed42c",
    "name": "PS LinuxApp02"
  }
]
```

Copy

This array becomes the list of items for which Collect tasks will be created. We'll reference one of its attributes, namely `sys_id`, as a variable in the Collect task's URL.

Collect Settings

Collect URL: An expression that produces a URL for the collect operation. We enter the first part of the URL as a constant, then for the final URL parameter, we use an expression to reference the `sys_id` returned by the Discover call:

```
'https://dev111111.service-now.com/api/now/cmdb/instance/cmdb_ci_appl'+ C.Encode.uri(
```

Copy

We use the C.Encode.uri Cribl expression to encode the `sys_id` in case any `sys_id` contains unsafe characters.

Collect method: `GET`.

Authentication

Use the **Authentication method** drop-down to select one of these options:

- **None**: Don't use authentication.
- **Basic**: Use HTTP token authentication.
- **Basic (credentials secret)**: Select a stored text secret in the resulting drop-down, or select **Create** to configure a new secret.
- **Login**: This option requires you to provide:
  - The **Login username** and **Login password** fields for your HTTP Basic authentication credentials.
  - The **Login URL** that SNOW should use to connect to Cribl Stream.
  - A **POST body** template for the request SNOW uses when logging in. You must edit this if your credentials' location differs from that specified by default.
  - The **Token attribute**, which is the path to the token attribute in login response body. Nested attributes are OK.
  - The **Authorize expression**, a JavaScript expression that computes the `Authorization` header to pass in Discover and Collect calls. The value `${token}` references the token obtained from login.
- **Login (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret. You must also provide the **Login URL**, **POST body**, **Token attribute**, and **Authorize expression**, as in the plain **Login** option.

Result Settings

These settings enable the Collector you've been configuring to use your **process** Pipeline.

Result Routing

Toggle **Send to Routes** off.

Choose the `SNOW_Instance_Process` Pipeline from the dropdown.

Choose whatever **Destination** suits your purposes. A DevNull Destination would make sense given that the purpose of this Collector is simply to get data into Redis.

Adding a Source with Data to Enrich

At this point, you have a Collector which stores CI record data in Redis. Assuming that you have a Source with events you want to enrich, you should now configure that Source to use the **enrich** Pipeline you created earlier.

In **Processing Settings** > **Pre-Processing**, choose `SNOW_Instance_Enrich` from the **Pipeline** drop-down.

Verifying the Enrichment of Events

This procedure should demonstrate that your setup is working, or help you troubleshoot.

1. Send a request to the SNOW Instance API to verify that it returns data.
    - Try the query builder.
2. Run the Collector once. Since you have attached the **process** Pipeline to the Collector, it should update Redis with `set` calls.
    - When you're ready to move towards production, you can run the Collector on a schedule, to update Redis periodically.
3. Run a `get` command on the Redis command line, using the key for your chosen API, to verify that Redis was updated as expected.
4. Check the Source whose data you are enriching, to verify that the fields you added are showing up.

8.1.1.14.8. Creating A Custom Collector

Cribl's flagship product introduced Collectors in Logstream 2.2, and they have since evolved to become a critical part of the platform. You can think of Collectors as jobs that retrieve data from an external service. You can also schedule them to run periodically, like a Linux cron job.

We built the Collection framework to be extensible like Cribl Stream Functions. This means that you can create new Collectors on the fly. This page covers the Collection process, schema files, and their implementation - to give you some context - before walking you though the steps of creating your own Collector.

Collection Process

At its simplest, collection is a two-step process: Discover, and Collect.

Discover

This step identifies the items to collect. The Discover call generates a list of items, and Collect tasks will be created for those items. Note that:

- The Discover call can return zero to many items. The Collection phase will run only if Discover returned at least one item.
- Your Collector's purpose (and definition) determines the actual data that Discover will return.
- If the Collector pulls data from files on disk, the Discover call lists the files in the directory to determine how many items match the criteria (matching by filter and/or date range). Assuming that there are 100 files to collect from, the Discover call will return 100 items (each specifying a file path and size).

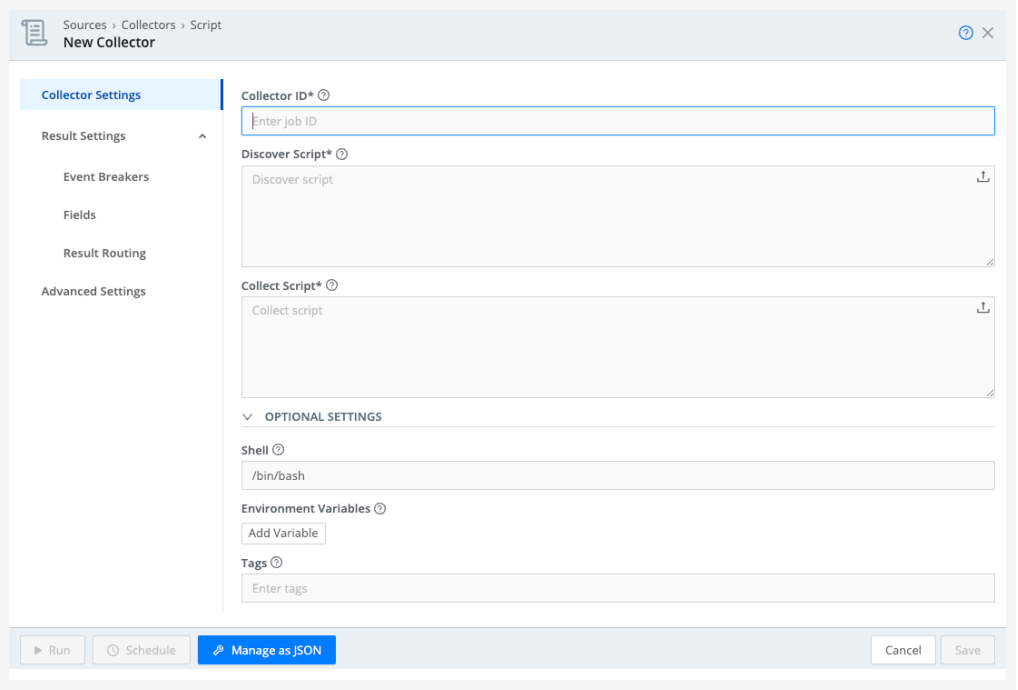
Collect

Each item, from the Discover phase's returned list of items, will trigger one Collect task for Cribl Stream to create and run. More on Collect:

- Generally, there is a single Collect task per item returned by Discover.
- In distributed deployments, you configure Collectors at the Worker Group level, and Worker Nodes execute the tasks. However, the Leader Node oversees the task distribution, and tries to maintain a fair balance across jobs.

Collector Schema Files

The Collector schema files, `conf.schema.json` and `conf.ui-schema.json`, describe the structure of the UI for configuring a new or existing Collector. As an example, here's a Script Collector's Configuration modal:



Script Collector Configuration Screen

The `conf.schema.json` below defines all the fields displayed on that **Collector Settings** form:

```
{
  "type": "object",
  "title": "",
  "required": ["discoverScript", "collectScript"],
  "properties": {
    "discoverScript": {
      "type": "string",
      "title": "Discover Script",
      "minLength": 1,
      "description": "Script to discover what to collect. Should output one task per
    },
    "collectScript": {
      "type": "string",
      "title": "Collect Script",
      "minLength": 1,
      "description": "Script to run to perform data collections. Task passed in as $(
    },
    "shell": {
      "type":"string",
      "title": "Shell",
      "description": "Shell to use to execute scripts.",
      "default": "/bin/bash"
    }
  }
}
```

Copy

The `conf-ui-schema.json` file further refines the field specifications in `conf.schema.json`. In the example below, it specifies that the `discoverScript` and `collectScript` fields should use the `TextareaUpload` widget with 5 rows. This file also gives each text field some placeholder (ghost) text, to display when no data is present in the field.

```
{
  "discoverScript": {
    "ui:widget": "TextareaUpload",
    "ui:options": {
      "rows": 5
    },
    "ui:placeholder": "Discover script"
  },
  "collectScript": {
    "ui:widget": "TextareaUpload",
    "ui:options": {
      "rows": 5
    },
    "ui:placeholder": "Collect script"
  }
```

Copy

Schemas can be simple, like the Script Collector, or complex like the REST Collector. This guide doesn't cover all the possibilities of working with schemas, which do get complex. However, you can check out other existing Collectors as examples of how to apply schemas for your own use case.

Collector Implementation

The Collector's implementation logic is part of `index.js`, and resides in the same directory as the Collector schema files. You must define the following attributes and methods for the Collector:

```
// Jobs can import anything from the C object, to see what's available use the
// Stream UI and an Eval Function to discover options.
const { Expression, PartialEvalRewrite } = C.expr;
const { httpSearch, isHttp200, RestVerb, HttpError, wrapExpr, DEFAULT_TIMEOUT_SECS }

exports.name = 'Weather';
exports.version = '0.1';
exports.disabled = false; // true to disable the collector
exports.destroyable = false;
exports.hidden = false; // true to hide collector in the UI

// Define Collector instance variables here: i.e.:
// let myVar; // Initialize in init, discover, or collect.

// init is called before the collection job starts. Gives the Collector a chance
// to validate configs and initialize internal state.
exports.init = async (opts) => {
  // validate configs, throw Error if a problem is found.
  // Initialize internal attributes here
}

// The Discover task's main job is to determine 'what' to collect. Each item
// to collect (i.e. a file, an API call, etc) is reported via the job object
// and will execute a collection task.
// Note that different instances of the Collector will be used for the Discover
// and Collect operations. Do not set internal Collector state in Discover
// and expect it to be present in Collect. The _init method will be called
// prior to Discover orCcollect.
exports.discover = async (job) => {
  // Job is an object that the collector can interact with, for example
  // we can use the job to access a logger.
  job.logger().info('Discover called');

  // In this case, reporting 2 hard coded items to be collected. Normally,
  // collectors dynamically report items to collect based on input from
  // an API or library call.
  await job.addResults([{"city": "San Francisco"},{"city": "Denver"}]);
  // Can also add results one at a time using await job.addResult("city": "San Franci
}

// One invocation of the Collect method is made for each item reported by the
// discover method. The collectible object contains the data reported by discover.
```

```
// In our example collect will be called twice, once for each item returned
// by discover's addResults call:
// Invocation 1: {"city": "San Francisco"}
// Invocation 2: {"city": "Denver"}
exports.collect = async (collectible, job) => {
   job.logger().info('In collect', { collectible });
   try {
     // Do actual data collection here. In this case we might make a REST API call
     // to retrieve current weather conditions for the city in collectible.
     const myReadableStream = doGetWeather(collectible.city);
   } catch (error) {
     // If the collector encounters a fatal error, pass the error to the job. This
     // will make the error visible in the Job inspector UI.
     job.reportError(error)
     return;
   }
   // Return result of the collect operation should be Promise<Readable>
   // which will be piped to routes or to the configured pipeline and destination.
   return Promise.resolve(myReadableStream)
}
```

Copy

Set Up a New Collector

Now to the exciting part. Here's an overview of how to add a Collector:

1. Create a directory where Collector files will reside. All files associated with the Collector should be in this directory.
2. Create the schema files and add them to the directory. The following schema files describe the structure of the UI for configuring a new or existing Collector:
   - `schema.json`
   - `ui-schema.json`
3. Create a JavaScript file named `index.js`, with naming attributes and required methods.
4. Test and validate the Collector.
5. Install the Collector in Cribl Stream.

Before You Start

A few things to note before we start the process of adding a custom Collector:

1. For a standalone install of a running Cribl Stream instance, add new directories and files to the `$CRIBL_HOME/local/cribl/collectors` directory (example: `/opt/cribl`).
2. The Collector will show up in the UI only if all schema files and the `index.js` file successfully compile. If the Collector is not showing up in the UI, check whether there was a problem compiling one of the files. You can check the errors through the API Server Logs.
3. Develop your Collector in a local test environment, as a best practice. After making changes to the Collector, you might need to restart/and or deploy your system.
4. Collectors can access all Node.js built-in modules, using the `require` directive to import each module.
5. You can include third-party Node.js modules into a custom Collector by installing them in the Collector's home directory.
6. Cribl Stream ships with out-of-the-box Collectors which reside in the `$CRIBL_HOME/default/cribl/collectors` directory. Feel free to copy contents from one of the existing Collectors to your `$CRIBL_HOME/local/cribl/collectors` (local directory) as a starting point/example of how to build more-complex schemas.

Never modify a Collector in the default directory (`$CRIBL_HOME/default/cribl/collectors/*`). Because:

1. Doing so changes the behavior of a Collector in your installed system.
2. Contents of the default directory will be overwritten during upgrades.

When creating your own Collector, always make a copy to a directory in the local `$CRIBL_HOME/local/cribl/collector/<yourCollector>`

Sample Collector Requirements

For this example, our sample Collector's goal is to generate events - containing a random quote obtained from an internal list - for a list of users. Here is a breakdown of this Collector's requirements:

1. Provide a random quote to a predefined or random list of usernames.
2. Accept the names of users for whom to generate quotes.
3. Accept randomly generated usernames, by specifying the number of usernames to generate.
4. Generate a single event containing a random quote for each user.
5. Pick the random quotes from a hard-coded list, or optionally, use a REST API instead.

For this sample Collector, we'll reference a third-party Node module to generate random usernames, and to give you a basic understanding of how to reference external code packages.

Set Up Your Environment

For this guide, we'll build a custom Collector in a standalone Cribl Stream install running in Docker. To adapt the steps for a distributed environment:

1. Build the Collector on the Leader Node.
2. The Collector directory will reside in the filesystem at: `$CRIBL_HOME/groups/<workerGroup>/local/cribl/collectors/quote_generator`. For example, to build in the default Worker Group, work in the directory: `$CRIBL_HOME/groups/default/local/cribl/collectors/quote_generator`.
3. Before running the Collector, commit and deploy changes for the parent Worker Group.

Deploy a Single Instance of Cribl Stream in Docker

1. Start the Docker instance by running the following command:
   `docker run -d -p 19000:9000 cribl/cribl:latest`
2. List Docker containers by running `docker ps`, as shown here:

```
$ docker ps
CONTAINER ID   IMAGE                    COMMAND                  CREATED
544370698fb5   cribl/cribl:3.4.1-RC1    "/sbin/entrypoint.sh..."   4 minutes ago
```

Copy

3. Access the Cribl Stream UI at port: `http://localhost:19000`
4. Connect to the container:
   `docker exec -it <Container ID> bash`
   (example: `docker exec -it 544370698fb5 bash`)
5. Update the apt installer:
   `apt update`
6. Install Vim (or an editor of your choice):
   - vim: `apt install vim`
   - nano: `apt install nano`

After this, remain connected to the Docker container, and follow the steps below to create the Collector.

Remember to leave the Docker container running while you work on the Collector. Also, remember to explicitly back up your files before stopping the Docker container.

Build a Collector (Single-Instance Environment)

In your terminal, type the following commands:

1. `cd /opt/cribl/local/cribl`
2. `mkdir collectors`
3. `cd collectors`
4. `mkdir quote_generator`
5. `cd quote_generator`
6. `cp ../../../../default/cribl/collectors/script/* .`
7. `chmod +w index.js *.json`
8. Edit `index.js` to change the following, to assign a unique name to the Collector:

```
exports.name = 'Quote Generator';
```

Copy

9. Edit `properties.type` within `conf.schema.json` file to remove `flag` keyword and change the `enum` to `["quote_generator"]`. You should end up with something like this:

```
{
  "type": "object",
  "title": "",
  "required": ["discoverScript", "collectScript"],
  "properties": {
    "type": {
      "type": "string",
      "enum": ["quote_generator"]
    },
    (...)
```

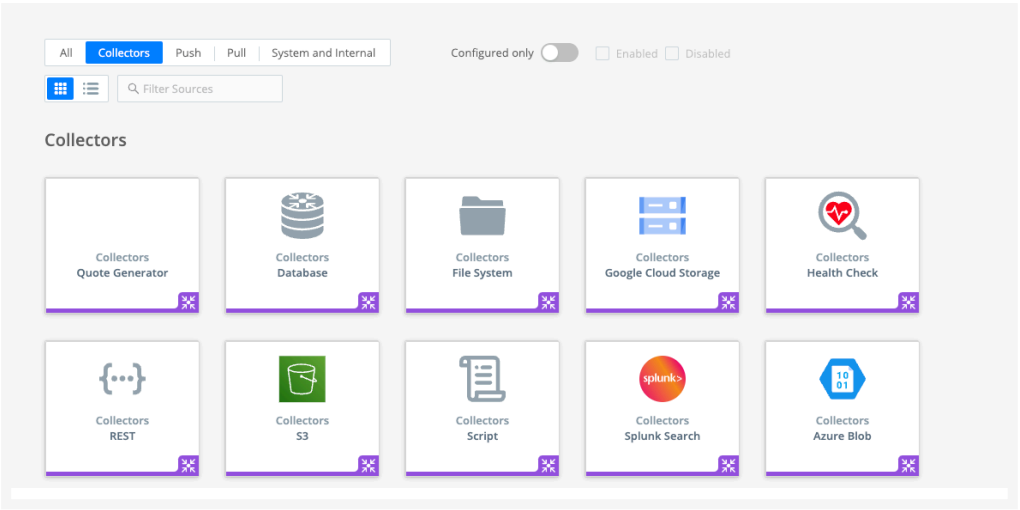
Copy

10. Next, restart Cribl Stream with the following command:
    `$ /opt/cribl/bin/cribl restart`
11. After Cribl Stream restarts, you must log out and log back in for the new Collector tile to display under Sources.

Configure the Collector in Cribl Stream

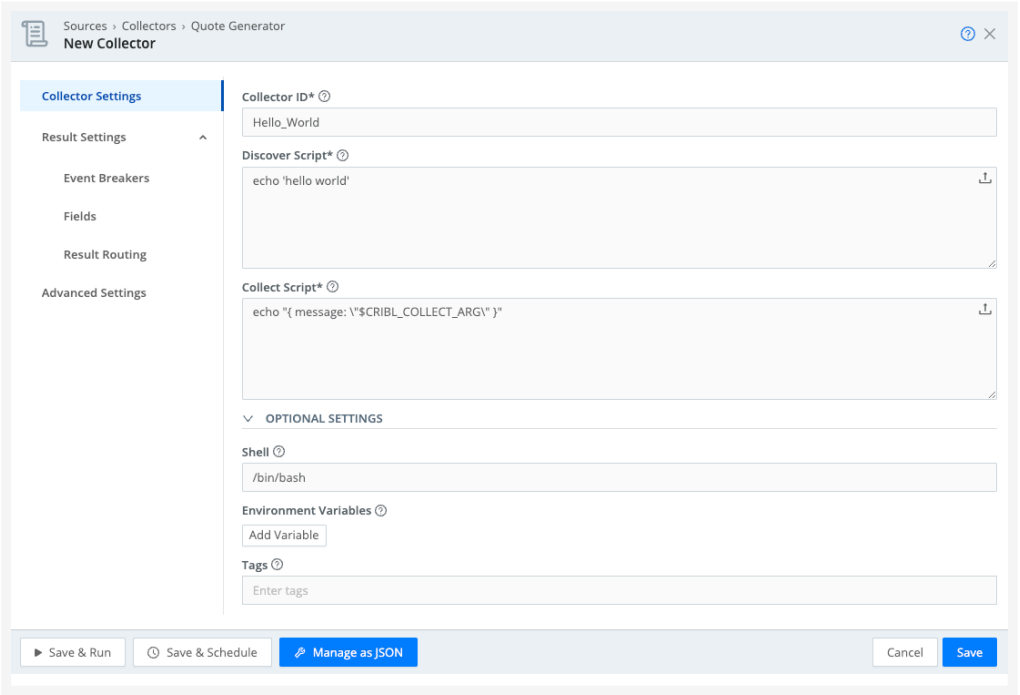
Now you'll configure the custom Quote Generator Collector that you built in the previous section.

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Sources**. Select the **Collectors** button to display the Collectors tiles.



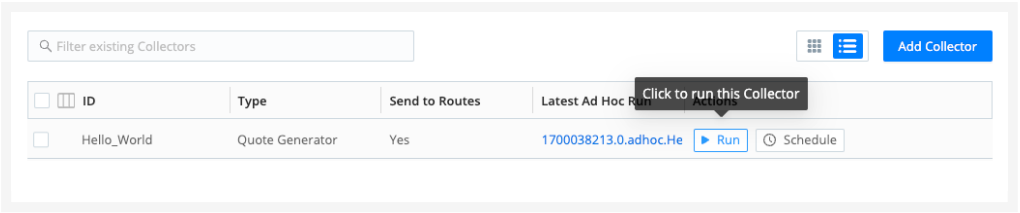
Quote Generator Collector (Missing an Icon)

1. Select the **Quote Generator** > **New Collector** modal.
2. Enter the following in the **Collector Settings** tab, then select **Save**:
    - **Collector ID**: `Hello_World`
    - **Discover Script**: `echo "hello world"`
    - **Collect Script**: `echo "{ message: \"$CRIBL_COLLECT_ARG\" }"`



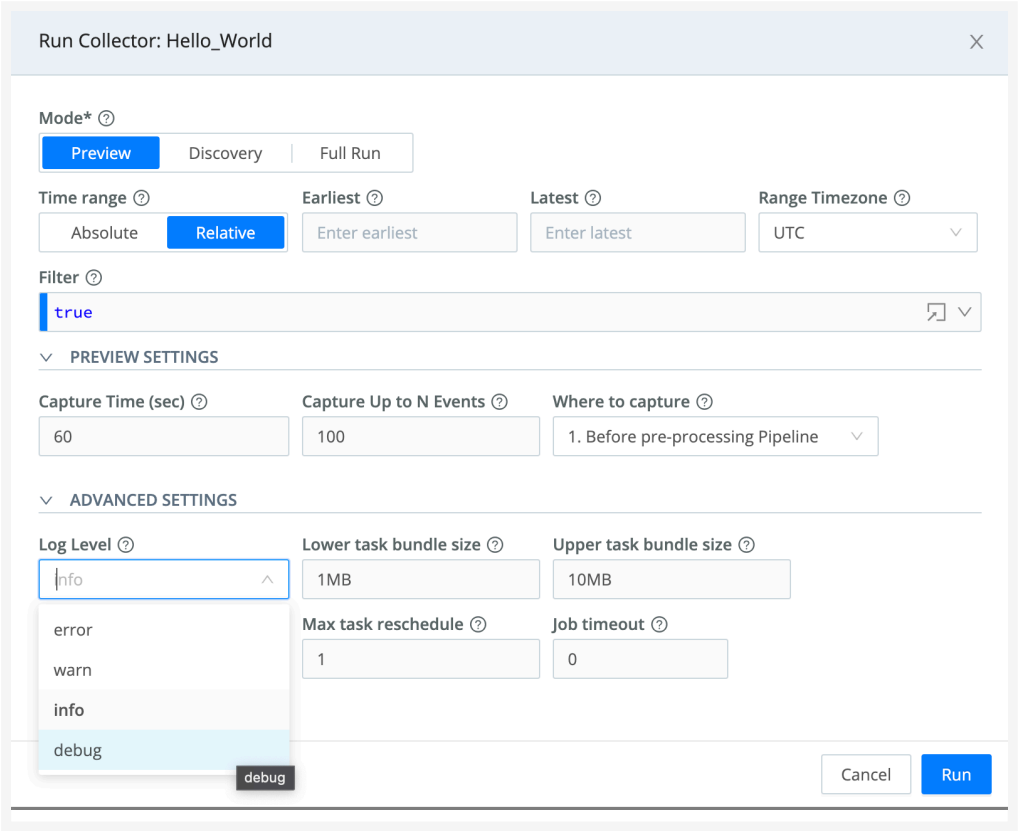
Collector Settings Tab

3. On the **Manage Quote Generator Collectors** page, select **Run** next to the Collector.



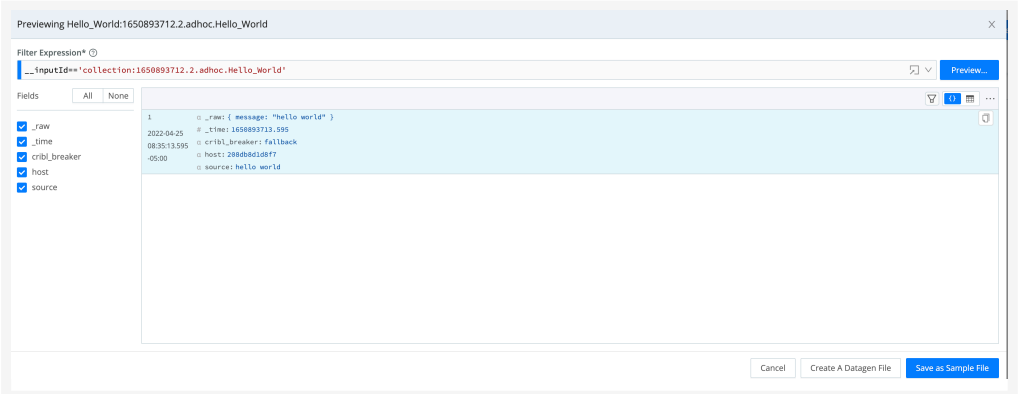
Run the Collector

4. In the **Run configuration** modal, select **Run** again. Note that you can set the Collector's debug level in the modal's **Advanced Options** section. For details, see Run Configurations and Shared Settings.



Run the Collector

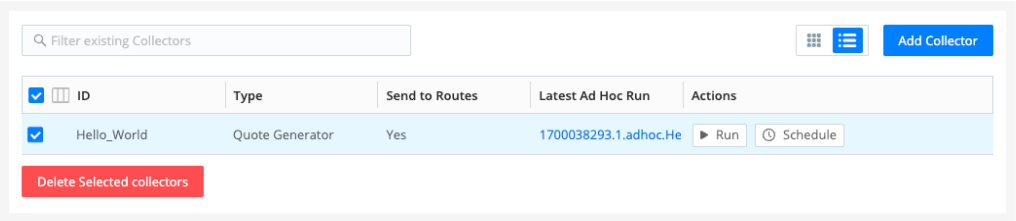
5. After the Collector runs, the following event will be generated:



Run the Collector

Now that we have a basic shell for our Collector, let's change the UI and `index.js` to add our custom Collector logic. Before making any additional changes, delete the Collector we just created.

6. On the **Manage Quote Generator Collectors** page, select the check box next to your Collector, then select **Delete selected Collectors**.



Delete the Collector

Edit the Schema Files

1. Replace the contents of `conf.schema.json` with the following:

```
{
  "type": "object",
  "title": "",
  "required": ["autoGenerateNames"],
  "properties": {
    "autoGenerateNames": {
      "type": "boolean",
      "title": "Auto generate names",
      "description": "Turn on to autogenerate names, use num names to specify ho
      "default": false
    }
  },
  "dependencies": {
    "autoGenerateNames": {
      "oneOf": [
        {
          "required": ["numNames"],
          "properties": {
            "autoGenerateNames": { "enum": [true] },
            "numNames": {
              "type": "number",
              "title": "Num names",
              "minimum": 1,
              "maximum": 1000,
              "description": "The number of names to auto generate, each name wi
            }
          }
        },
        {
          "required": ["names"],
          "properties": {
            "autoGenerateNames": { "enum": [false] },
            "names": {
              "type": "array",
              "title": "Names",
              "minLength": 1,
              "description": "List of user names to retrieve quotes for.",
              "items": {"type": "string"}
            }
          }
        }
      ]
    `
```

```
      }
    }
  }
```

Copy

Two interesting things to note in the `conf.schema.json` file:

- When you disable - set to `false` - the boolean toggle `autoGenerateNames`, the UI displays a **Names** field requesting a list of user names for which to retrieve quotes.
- When you enable - set to `true` - the boolean toggle `autoGenerateNames`, the UI displays a **Number of names to enter** field requesting the number of names to auto-generate.

The dependency allows dynamic onscreen behavior based on the field's value.

2. Replace the contents of `conf.ui-schema.json` with the following:

```
{
  "names": {
    "ui:field": "Tags",
    "ui:placeholder": "Enter names",
    "ui:options": {
      "separator": ","
    }
  }
}
```

Copy

The UI schema, in this case, further refines the widget ("Tags") used for the Names field. We will see this in action a bit later.

3. Finally, replace the contents of `index.js` with the following:

```
exports.name = 'Quote Generator';
exports.version = '0.1';
exports.disabled = false;
exports.destroyable = false;

const Readable = require('stream').Readable;
const os = require('os');
const host = os.hostname();
const logger = C.util.getLogger('myCollector'); // For use in debugging init, se

let conf;
let batchSize;
let filter;
let autoGenerate = false;
let numNames = 0;
let names;

exports.init = async (opts) => {
  conf = opts.conf;
  //logger.info('INIT conf', { conf });
  batchSize = conf.maxBatchSize || 10;
  filter = conf.filter || 'true';
  autoGenerate = conf.autoGenerateNames
  names = conf.names || [];
  numNames = conf.numNames ?? 3;
  //logger.info('INIT VALUES', { autoGenerate, names, numNames });
  return Promise.resolve();
};

exports.discover = async (job) => {
  for (let i = 0; i < names.length; i++) {
    job.addResult({"name": names[i]});
  }
}

exports.collect = async (collectible, job) => {
  job.logger().debug('Enter collect', { collectible });
  const quote = "Carpe Diem!"; // Hard coded quote for now.

  // Must return a readable stream from the collect method. Here we are returnin
  // the result string wrapped in a Readable.
  const s = new Readable();
```

```
    s.push(JSON.stringify({ host, name: collectible.name, quote }));
    s.push(null);

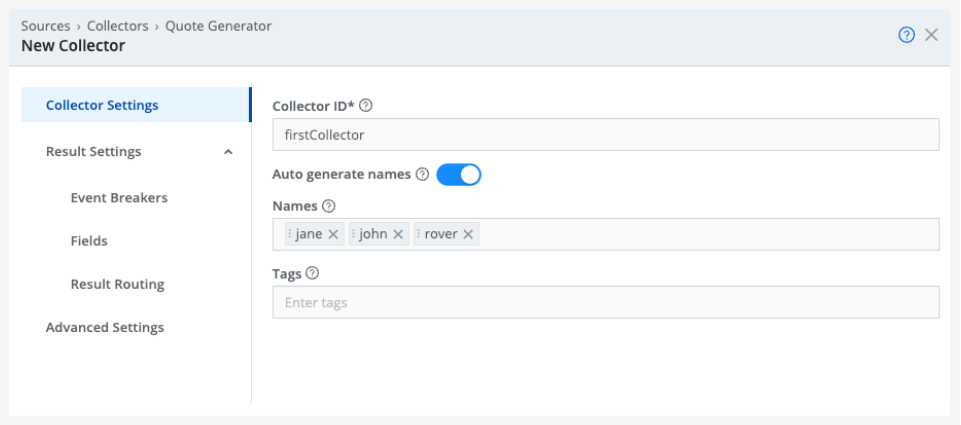
    // Return readable to stream collected data
    return Promise.resolve(s)
  };
```

Copy

4. Next, restart Cribl Stream with the following command:
   `$ /opt/cribl/bin/cribl restart`
5. After Cribl Stream restarts, you must log out and log back in for the new Collector tile to display under Sources.

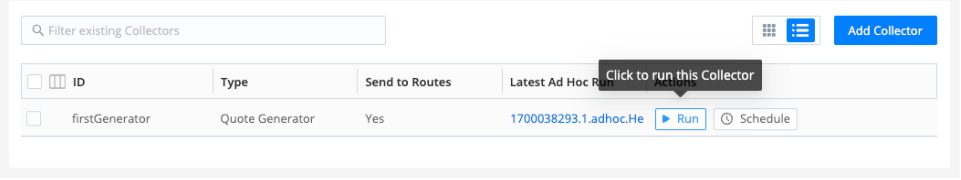
Configure a New Collector Instance

1. In Cribl Stream, navigate to the Quote Generator Collector and select **Add New** to open the **Quote Generator > New Collector** modal.
2. Type the following into the **Collector Settings** tab, then select **Save**:
   - **Collector ID**: `firstCollector`
   - **Auto generate names**: Toggle off.
   - **Names**: `Jane, John, Rover`



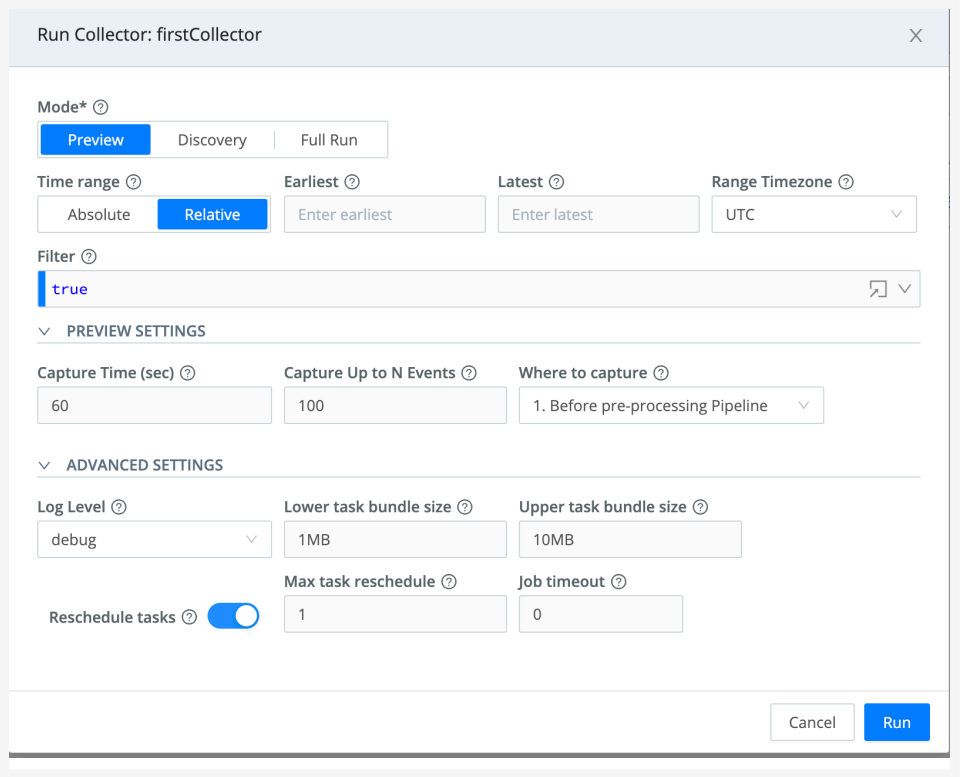
New Collector Settings

3. On the **Manage Quote Generator Collectors** page select **Run** next to the Collector.



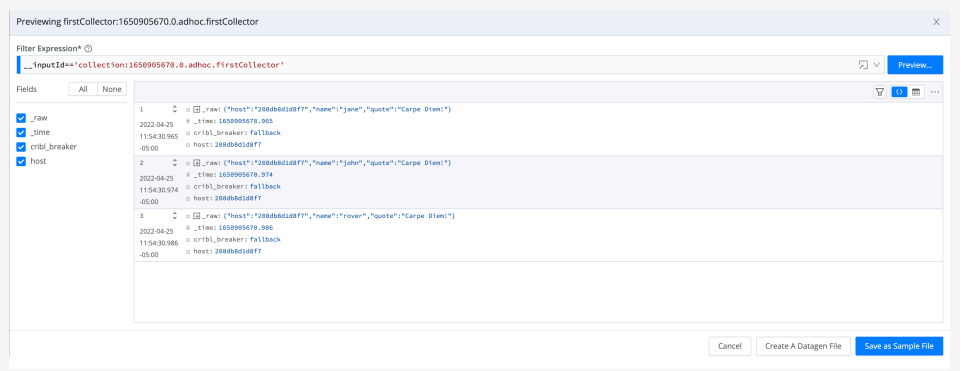
Run the Collector

4. In the **Run configuration** modal:
    - For **Mode**, select **Preview**.
    - For **Log Level**, select **debug**.
    - Select **Run** again.



Run Collector Settings

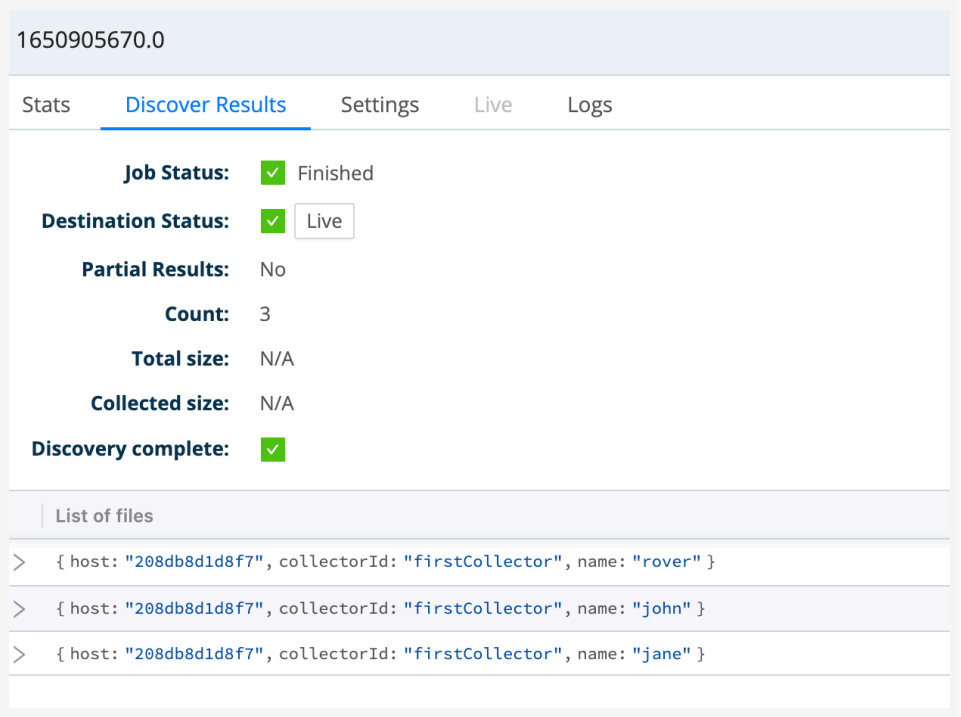
5. If the Collector works, a successful run displays a results dialog, including an event for each name that was added.



Collector Events

6. Close the **Preview** dialog.
7. On the **Manage Quote Generator Collectors** page, select **Latest ad hoc run**.
8. This will open a dialog from the Job Inspector with information about the run, including job status, items returned by the Discover call, and logs. Select the **Discover Results** tab to see data returned from the

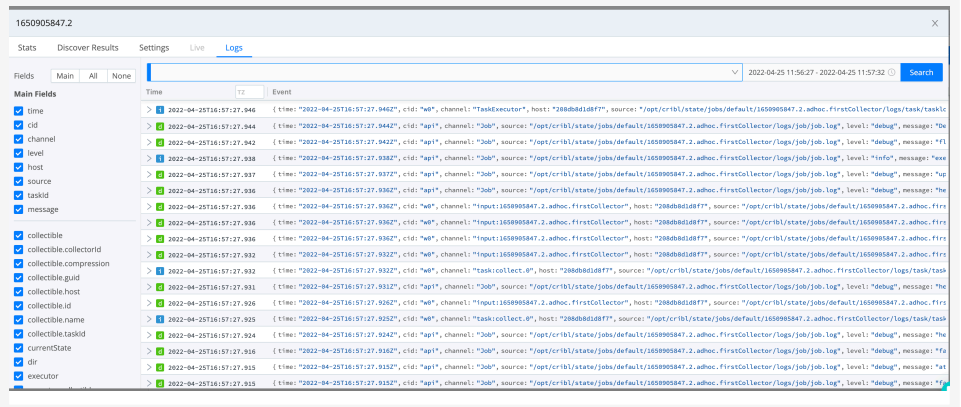
Collector's Discover call:



Discover Results

The Collector method separately. invokes each item returned The Collector argument is the content from each row in the table.

9. Select the **Logs** tab to view logs from the run. This is where you can locate anything logged by `job.logger`.



View Logs

The exceptions are anything logged by our Collector in `init`. Notice that `index.js` defined another logger to use when debugging `init`:

```
const logger = C.util.getLogger('myCollector');
```

You can view the relevant Worker Process' logs on the Monitoring page.

Integrate a Third-Party Package

Next, we'll integrate a third-party package to randomly generate names in the Discover method.

1. In your terminal, follow these steps to install npm (Node Package Manager) into the virtual machine running Cribl Stream:
    - `apt update`
    - `apt install npm`
    - When prompted, select a `Region`.
    - When prompted, select a `Timezone`.
2. Run the following commands in your Collector directory:
    - `cd /opt/cribl/local/cribl/collectors/quote_generator`
    - `npm init`: Answer all questions with default answers. The values are not important for our purposes.
    - `npm install username-generator --save`

    This will install the third-party username-generator package in `/opt/cribl/local/cribl/collectors/quote_generator/node_modules`.
3. Next, update the Collector's `index.js` with the following content, to use this new package to auto-generate names:

```
exports.name = 'Quote Generator';
exports.version = '0.1';
exports.disabled = false;
exports.destroyable = false;

const UsernameGenerator = require('username-generator');
const Readable = require('stream').Readable;
const os = require('os');
const host = os.hostname();
const logger = C.util.getLogger('myCollector'); // For use in debugging init, se

let conf;
let batchSize;
let filter;
let autoGenerate = false;
let numNames = 0;
let names;

exports.init = async (opts) => {
  conf = opts.conf;
  //logger.info('INIT conf', { conf });
  batchSize = conf.maxBatchSize || 10;
  filter = conf.filter || 'true';
  autoGenerate = conf.autoGenerateNames
  names = conf.names || [];
  numNames = conf.numNames ?? 3;
  //logger.info('INIT VALUES', { autoGenerate, names, numNames });
  return Promise.resolve();
};

exports.discover = async (job) => {
  if (autoGenerate) {
    // Auto generate usernames using 3rd party package.
    names = [];
    for (let i = 0; i < numNames; i++) {
      names.push(UsernameGenerator.generateUsername('_'));
    }
    job.logger().info('Successfully generated usernames', { numGenerated: names.
  }
  for (let i = 0; i < names.length; i++) {
    job.addResult({"name": names[i]});
  }
`
```

```
}

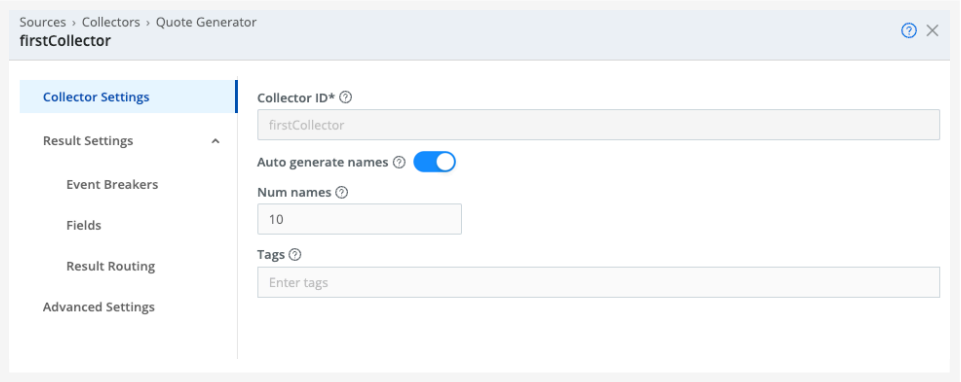
exports.collect = async (collectible, job) => {
  job.logger().info('Enter collect', { collectible });
  const quote = "Carpe Diem!"; // Hard coded quote for now.

  // Must return a readable stream from the collect method. Here we are returnin
  // the result string wrapped in a Readable.
  const s = new Readable();
  s.push(JSON.stringify({ host, name: collectible.name, quote }));
  s.push(null);

  // Return readable to stream collected data
  return Promise.resolve(s)
};
```

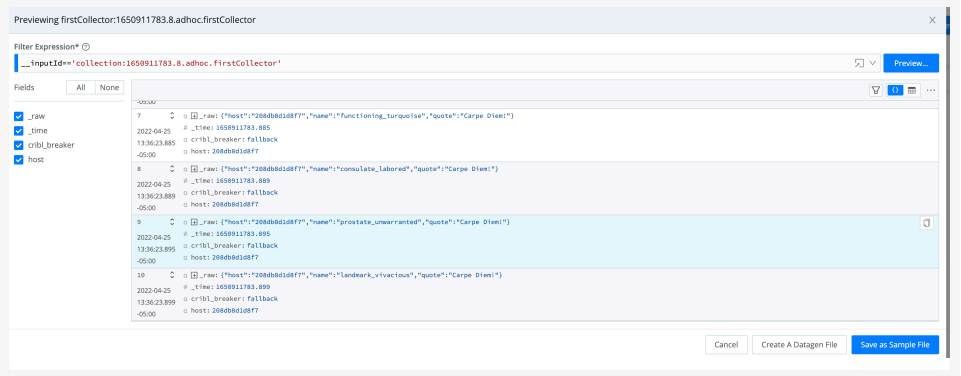
Copy

4. Now, in Cribl Stream, update the existing Collector to use the auto-generate feature. Add these settings, then select **Save**:
   - **Auto generate names**: Toggle on.
   - **Num names**: `10`



Update Existing Collector

5. Run the Collector, notice that 10 Events are now displayed and each username is randomized:



6. Next, update the Collector's `index.js` with the following content to create a static list of random quotes in the Collect method. You can optionally retrieve a random quote from a REST API instead.

```
exports.name = 'Quote Generator';
exports.version = '0.1';
exports.disabled = false;
exports.destroyable = false;

const UsernameGenerator = require('username-generator');
const Readable = require('stream').Readable;
const os = require('os');
const host = os.hostname();
```

8.1.2. Amazon

You can use the following Sources to receive data from Amazon services.

Source	Description
Amazon Data Firehose	Receive Amazon Data Firehose streams via HTTP endpoint
Amazon Kinesis Data Streams	Receive data records from Amazon Kinesis Data Streams
Amazon S3	Receive data from Amazon S3 buckets
Amazon Security Lake	Configuring Amazon Security Lake Source
Amazon SQS	Receive events from Amazon Simple Queuing Service

8.1.2.1. Amazon Data Firehose Source

Cribl Stream supports receiving data from Amazon Data Firehose delivery streams via the **HTTP endpoint** destination option.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

AWS Configuration

Configuring the endpoint and port settings for integrating Cribl with Amazon Data Firehose varies depending on whether you're using Cribl.Cloud or a non-Cribl.Cloud deployment.

- **Cribl.Cloud**:
  - Configure the AWS destination **HTTP endpoint URL** to your Cribl.Cloud endpoint `https://default.main.<organizationId>.cribl.cloud` on port `443`. Replace `<organizationId>` with your actual Organization ID.
  - Configure the Cribl Stream Amazon Data Firehose Source to listen on port `10443` internally.
- **Non-Cribl.Cloud**:
  - Configure the AWS destination **HTTP endpoint URL** to the public URL of your Cribl Stream instance.
  - Configure the Cribl Stream Amazon Data Firehose Source to either listen on port `443` with elevated privileges or use a reverse proxy to forward traffic from port `443` to a non-privileged port, like `10443`.

Configure Cribl Stream to Receive Data over HTTP(S) from Amazon Data Firehose

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
   - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
   - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.

2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Address to bind on. Defaults to `0.0.0.0` (all addresses).
    - **Port**: Enter the port number to listen on. In Cribl.Cloud, use port `10443`, as this is where the data is processed internally.
3. Under **Authentication**, enter the **Auth tokens**:
    - **Token**: Shared secrets to be provided by any client (Authorization: <token>). Click **Generate** to create a new secret. If empty, unauthorized access **will be permitted**.
4. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled off.

- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts

to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.

- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

> Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; etc.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

> The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.

- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and functions can use them to make processing decisions.

Fields for this Source:

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__inputId`
- `__raw`
- `__srcIpPort` - See details below.

Records also include the following metadata specific to the Amazon Data Firehose service:

- `__firehoseArn`: The Amazon Resource Name (ARN) of the Amazon Data Firehose delivery stream that served as the data source.
- `__firehoseReqId`: A unique identifier assigned to the specific data ingestion request processed by Amazon Data Firehose.
- `__firehoseEndpoint`: The service endpoint URL of the Amazon Data Firehose service that received the data.
- `__firehoseToken`: An authentication token associated with the Amazon Data Firehose data ingestion request, used for source validation.

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Amazon Data Firehose client sending data to this Source.

When any proxies (including load balancers) lie between the Amazon Data Firehose client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Limitations

If you set the optional `IntervalInSeconds` and/or `SizeInMBs` parameters in the Amazon Data Firehose `BufferingHints` API, beware of selecting extreme values (toward the ends of the API's supported ranges). These can send more bytes than Cribl Stream can buffer, causing Cribl Stream to send HTTP 500 error responses to Amazon Data Firehose.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

Dropping request because token invalid",“authToken”: “Bas...Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.2.2. Amazon Kinesis Data Streams Source

Cribl Stream supports receiving data records from Amazon Kinesis Data Streams.

Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **No** | Uses Key-Value Store

Configure Cribl Stream to Receive Data from Amazon Kinesis Data Streams

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Kinesis Stream Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Stream name**: Enter the name of the Kinesis Data Stream (not ARN) to read data from.
    - **Region**: Select the AWS Region where your Kinesis Data Stream is located. If your Region isn't listed, you can manually enter it. Additionally, you can specify the service endpoint in the **Endpoint** field under Advanced Settings.
3. Next, you can configure the following **Optional Settings**:
    - **Shard iterator start**: Location from which a newly created Source should start reading any shard, including shards that are added later. Defaults to `Earliest Record`. For details on Leader dependency, see Leader Dependency below.
    - **Record data format**: Format of data inside the Kinesis Stream records. Gzip compression is automatically detected. For details, see Record Data Format Options.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Authentication, AssumeRole, Processing and Advanced settings, or Connected Destinations outlined in the sections below.

5. Select **Save**, then **Commit & Deploy**.

Leader Dependency

The Amazon Kinesis Data Streams Source relies on the Leader Node for proper operation. If the Leader Node becomes unavailable, the Source's performance becomes degraded. In this state:

- Existing shard readers will continue to process data.
- New shard readers will not start.

This means that while your current Kinesis data streams will continue to be processed, any changes that require new shard readers (such as increasing the number of shards in your Kinesis stream) will be delayed until the Leader Node is restored. Your Leader Node must be stable for optimal performance.

Record Data Format Options

In **Record data format**, options include:

- Cribl (the default): Use this option if Cribl Stream wrote data to Kinesis in this format. This is a type of NDJSON.
- Newline JSON: Use if the records contain newline-delimited JSON (NDJSON) events - for example, Kubernetes logs ingested through Kinesis. This is a good choice if you don't know the records' format.
- CloudWatch Logs: Use if you've configured CloudWatch to send logs to Kinesis.
- **Event per line**: NDJSON can use this format when it fails to parse lines as valid JSON.

Authentication

Use the **Authentication method** drop-down to select an AWS authentication method:

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC - for example, those running in a private cloud.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key.

IAM Role Authentication

When using an IAM role to authenticate with Kinesis Streams, the IAM policy statements must include the following Actions:

- `kinesis:GetRecords`
- `kinesis:GetShardIterator`
- `kinesis:ListShards`

For details, see AWS' Actions, resources, and condition keys for Amazon Kinesis Data Streams documentation.

Manual Authentication

The **Manual** option exposes these additional fields:

Access key: Enter your AWS access key. If not present, will fall back to `env.AWS_ACCESS_KEY_ID`, or to the metadata endpoint for IAM role credentials.

Secret key: Enter your AWS secret key. If not present, will fall back to `env.AWS_SECRET_ACCESS_KEY`, or to the metadata endpoint for IAM credentials.

Secret Authentication

The **Secret** option exposes this additional field:

Secret key pair: Use the drop-down to select a secret key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for Kinesis Streams: Whether to use Assume Role credentials to access Kinesis Streams. Default is toggled off.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume. For Cribl.Cloud Worker Groups that need to read from Kinesis streams in a different AWS account, configure the target role's trust policy to trust your Cribl Worker role ARN. To get the role ARN for Workers in a Worker Group, see Worker Group Details.

External ID: Enter the External ID to use when assuming role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Shard selection expression: A JavaScript expression to be called with each `shardId` for the stream. The shard will be processed if the expression evaluates to a truthy value. Defaults to `true`. Avoid trying to use this expression to distribute data evenly across Worker Processes - the preferred way to do that is to use round robin load balancing.

Service period: Time interval (in minutes) between consecutive service calls that check for configuration updates and perform maintenance. Defaults to `1` minute. This differs from polling frequency, which determines how often the Source fetches new data. Adjust the service period to balance the need for quick configuration updates (use a shorter interval) with the frequency of maintenance tasks (use a longer interval for less frequent tasks).

Endpoint: Kinesis stream service endpoint. If empty, the endpoint will be automatically constructed from the Region.

Signature version: Signature version to use for signing Kinesis Stream requests. Defaults to `v4`.

Verify KPL checksums: Enable this setting to verify Kinesis Producer Library (KPL) event checksums. This setting is not available when Cribl Stream is deployed in FIPS mode.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority - self-signed certificates, for example. Defaults to toggled on.

Avoid duplicate records: If toggled on, this Source will always start streaming at the next available record in the sequence. (This can cause data loss after a Worker Node's unexpected shutdown or restart.) If toggled off (default), the Source will reread the last two batches of events at startup. (This prevents data loss, but can ingest duplicate events.)

Records limit per call: In v.4.0.4 and newer, sets the maximum number of records ingested, per shard, in each getRecords call. You can acclerate data collection by increasing the default/minimum `5000` limit, to as high as `10000` records.

Total records limit: In v.4.0.4 and newer, sets the maximum number of records, across all shards, to pull down at once per Worker Process. Default/minimum value is `20000` records. Beware of very high values that could excessively consume Workers' memory.

Shard load balancing: The load-balancing algorithm to use for spreading out shards across Workers and Worker Processes. Options include:

- Consistent Hashing (default)
- Round Robin

For the Round Robin option, both the Leader and the Worker Groups must be on Cribl Stream 4.1 or later.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Field for this Source:

- `__checksum`
- `__encryptionType`
- `__final`
- `__inputID`
- `__isKPL`
- `__partitionKey`
- `__sequenceNumber`

Apart from the internal fields, one important non-internal field that this Source adds is `_raw`.

Principles for Configuration Decisions

To get the best results, apply these principles when configuring the Source and Amazon Kinesis Data Streams itself:

The number of shards should be a multiple of the number of Worker Processes in the Worker Group. A ratio of one shard to one Worker Process yields the fastest possible processing.

Distributing data evenly across Worker Processes yields the best performance. To achieve this, choose **Advanced Settings > Shard Load Balancing > Round Robin**. This helps avoid the situation where some Worker Processes are overburdened while others sit idle. Avoid trying to use the **Advanced Settings > Shard selection expression** for this purpose.

How the Source and Kinesis Interact

The Leader Node for the Kinesis Streams Source stores shard state on disk, so that the Source can pick up where it left off across restarts. The state file is located in the `$CRIBL_HOME/state/` directory. The path format looks like this:

```
.../state/kvstore/<group-id>/input_kinesis_<input-id>_<stream-name>/state.json
```

For example:

```
state/kvstore/default/input_kinesis_kinesisIn_just-a-test/state.json
```

Worker Processes become Kinesis Consumers and get a list of available shards from Kinesis. They then contact the Leader Node to request the sequence numbers that the Leader has stored for those shards. Each sequence number references a specific record.

Every five minutes, each Worker Process takes the latest sequence numbers for the shards it is responsible for, and forwards those sequence numbers to the Leader Node, enabling the Leader Node to update the state that it is storing. The Leader Node persists the `shardId` > `sequenceNumber` mapping to disk using a key-value store where the keys are Shard IDs and the values are sequence numbers.

In situations where Worker Processes must reload state (by retrieving it from the key-value store on the Leader Node), each Worker Process will resume reading the same shard it was reading before, provided that neither the number of shards nor the number of Worker Processes have changed.

Polling Frequency

The polling frequency for the Kinesis Streams Source is adaptive. By default, if the Source has caught up to the latest records, it sets a delay of `1` second. If it has not caught up, it immediately requests the next batch of records without waiting. This ensures efficient data consumption without unnecessary delays.

Setting Shard iterator start

Under certain circumstances, the Leader Node will not have any sequence numbers for a given shard and the Worker Process will be reading from that shard for the first time. (These conditions include when the shard is newly added to the Kinesis stream and/or when the Source is newly created.)

When this happens, the Worker Process then does what is specified in **Optional Settings** > **Shard iterator start**:

- When **Shard iterator start** is set to `Earliest record`, the Worker starts reading from the beginning.
- When it is set to `Latest record`, the Worker Process resumes reading the shard from the point where Cribl Stream previously left off.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.2.3. Amazon S3 Source

Cribl Stream supports receiving data from Amazon S3 buckets. The Amazon S3 Source can receive data using either of these methods:

- S3 Bucket event notifications through Amazon Simple Queue Service (SQS)
- Amazon Security Lake event subscriptions

These two notifications use the same method of sending data objects to a queue, but the underlying data format from these two methods differ. Cribl Stream supports both data structures.

Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **YES**

Cribl Stream running on Linux (only) can use this Source to read Parquet files, identified by a `.parquet`, `.parq`, or `.pqt` filename extension.

For configuration examples, see our Amazon S3 Better Practices and Using S3 Storage and Replay guides.

S3 Setup Strategy

The source S3 bucket must be configured to send `s3:ObjectCreated:*` events to an SQS queue, either directly (easiest) or via SNS (Amazon Simple Notification Service). See the event notification configuration guidelines below.

SQS messages will be deleted after they're read, unless an error occurs, in which case Cribl Stream will retry. This means that although Cribl Stream will ignore files not matching the **Filename Filter**, their SQS events/notifications will still be read, and then deleted from the queue (along with those from files that match).

These ignored files will no longer be available to other S3 Sources targeting the same SQS queue. If you still need to process these files, we suggest one of these alternatives:

- Using a different, dedicated SQS queue. (Preferred and recommended.)
- Applying a broad filter on a single Source, and then using pre-processing Pipelines an/or Route filters for further processing.

Compression

Cribl Stream can ingest compressed S3 files if they meet one of the following conditions:

- Compressed with the `x-gzip` MIME type.

- Ends with the `.gz`, `.tgz`, `.tar.gz`, `.tgz`, or `.tar` extension.
- Can be uncompressed using the `zlib.gunzip` algorithm.

Storage Class Compatibility

Cribl Stream does **not** support data preview, collection, or replay from S3 Glacier or S3 Deep Glacier storage classes, whose stated retrieval lags (variously minutes to 48 hours) cannot guarantee data availability when the Collector needs it.

Cribl Stream **does** support data preview, collection, and replay from S3 Glacier Instant Retrieval when you're using the S3 Intelligent-Tiering storage class.

Configure Cribl Stream to Receive Data from Amazon S3

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this S3 Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Queue**: The name, URL, or Amazon Resource Name (ARN) of the SQS queue to read events from. When specifying a non-AWS URL, you must use the format: `{url}/<queueName>`. (For example: `https://host:port/<queueName>`.) This value must be a JavaScript expression (which can evaluate to a constant), enclosed in single quotes, double quotes, or backticks.
3. Next, you can configure the following **Optional Settings**:
    - **Filename filter**: Regex matching file names to download and process. Defaults to `.*`, to match all characters. This regex will be evaluated against the S3 key's full path.
    - **Region**: AWS Region where the S3 bucket and SQS queue are located. Required, unless the **Queue** entry is a URL or ARN that includes a Region.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. In the **Authentication** settings, select one of the following options from the menu:

Option	Description
Auto	This default option uses the AWS SDK for JavaScript to check for credentials in a specific order by priority. It first checks for IAM Roles for Amazon EC2, then a shared credentials file, then environmental variables, and so forth. See Authentication Settings for more information.
Manual	If not running on AWS, select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, such as those running a private cloud. See Authentication Settings for more information.
Secret	If not running on AWS, select this option to supply a stored secret that references an AWS access key and secret key. See Authentication Settings for more information.

5. If needed, configure the **AssumeRole** settings. You can use these settings to access resources in a different Region than Cribl Stream. See Assume Role Settings for more information.
6. Optionally, refer to the following sections for the remaining settings:
    - Processing Settings
    - Advanced Settings
    - Connected Destinations
7. Select **Save**, then **Commit & Deploy**.
8. Verify that data is flowing correctly to Cribl Stream by viewing the **Live Data** feed from the Source.

Authentication

Use the **Authentication method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, such as those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

The values for **Access key** and **Secret key** can be a constant or a JavaScript expression (such as `${C.env.MY_VAR}`). Enclose values containing special characters (like, `/`) or using environment variables in single quotes or backticks.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select a secret key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for Amazon S3: Whether to use Assume Role credentials to access S3. Defaults to toggled on.

Enable for Amazon SQS: Whether to use Assume Role credentials when accessing SQS. Default is toggled off.

AWS account ID: SQS queue owner's AWS account ID. Leave empty if the SQS queue is in the same AWS account.

AssumeRole ARN: Enter the ARN of the role to assume. When you run this Source on Cribl.Cloud Worker Groups, configure your AWS role's trust policy to allow the Worker role ARN to assume it. To get the role ARN

for Workers in a Worker Group, see Worker Group Details.

External ID: Enter the External ID to use when assuming role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

Enabled: Default is toggled off. Toggle on to enable the custom command.

Command: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).

Arguments: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

This section defines event breaking rulesets that will be applied, in order.

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Include notification metadata: Toggle on to add the S3 event notification payload to a `__sqsMetadata` field on every event from the corresponding S3 file.

This metadata is useful for several purposes. You can measure processing latency by comparing the object's `eventTime` to the event's `_time`, or implement region-based routing using the `awsRegion` field in a Pipeline. The full payload also serves as a definitive record for auditing the origin of the data.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Checkpoint Settings

Enable checkpointing: When toggled off (default) Cribl Stream will always start processing a file from the beginning. Toggle on if, after an error or restart occurs, you want Cribl Stream to resume processing a given file from the point where it was interrupted, as explained below. When turned on, the following additional setting becomes available:

- **Retries**: The number of times you want Cribl Stream to retry processing a file after an error or restart occurs. If **Skip file on error** is enabled, this setting is ignored.

Other Settings

Endpoint: S3 service endpoint. If empty, defaults to AWS's Region-specific endpoint. Otherwise, used to point to an S3-compatible endpoint. To access the AWS S3 endpoints, use the path-style URL format. You don't need to specify the bucket name in the URL, because Cribl Stream will automatically add it to the URL path. For details, see AWS' Path-Style Requests topic.

Signature version: Signature version to use for signing SQS requests. Defaults to `v4`.

Message limit: The maximum number of messages that SQS should return in a poll request. Amazon SQS never returns more messages than this value. (However, fewer messages might be returned.) Acceptable values: 1 to 10. Defaults to `1`.

Visibility timeout seconds: The duration (in seconds) that the received messages are hidden from subsequent retrieve requests, after being retrieved by a ReceiveMessage request. Defaults to `600`.

Cribl Stream will automatically extend this timeout until the initial request's files have been processed - notably, in the case of large files that require additional processing time.

Number of receivers: The number of receiver processes to run. The higher the number, the better the throughput, at the expense of CPU overhead. Defaults to `1`.

Poll timeout (secs): How long to wait for events before polling again. Minimum `1` second; default `10`; maximum `20`. Short durations increase the number - and thus the cost - of requests sent to AWS. (The UI will show a warning for intervals shorter than `5` seconds.) Long durations increase the time the Source takes to react to configuration changes and system restarts.

Socket timeout: Socket inactivity timeout (in seconds). Increase this value if retrievals time out during backpressure. Defaults to `300` seconds.

Parquet chunk size limit (MB): Maximum size for each Parquet chunk. Defaults to `5` MB. Valid range is `1` to `100` MB. Cribl Stream stores chunks in the location specified by the `CRIBL_TMP_DIR` environment variable. It removes the chunks immediately after reading them.

Parquet chunk download timeout (seconds): The maximum time to wait for a Parquet file's chunk to be downloaded. If a required chunk cannot be downloaded within this time limit, processing will end. Defaults to `600` seconds. Valid range is `1` second to `3600` seconds (1 hour).

Skip file on error: Toggle on to skip files that trigger a processing error (for example, corrupted files). Default is toggled off, which enables retries after a processing error.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Encoding: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.

Tag after processing: Toggle on to tag S3 objects once processed, allowing you to leverage tools like S3 Lifecycle to automatically delete, archive, or further process only those objects confirmed as ingested by Cribl. This requires the `s3:GetObjectTagging` and `s3:PutObjectTagging` S3 permissions.

Tag key and **Tag value** support the Cribl expression syntax. These expressions are evaluated once when the S3 Source is initialized, not for each file processed. The resulting static key and value are then applied to all processed objects for the entire lifespan of the Source. See Amazon's documentation on how to categorize your storage using tags for details and requirements.

- **Tag key**: The key for the S3 object tag. The key identifies the tag in AWS and must comply with S3 tag key requirements (maximum 128 Unicode characters, no spaces, case-sensitive).
- **Tag value**: The value for the S3 object tag. The value is paired with the key and must comply with S3 tag value requirements (maximum 256 Unicode characters, case-sensitive).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Checkpointing

When Workers restart, checkpointing can prevent the Source from losing data or collecting duplicate events. Among the situations where this works well:

- When Cribl Stream pushes a new configuration: the Leader pushes the configuration to the Workers, and restarts them.
- When a Worker has gotten into a bad state, and you want to restart it.

Checkpointing allows Cribl Stream to resume processing a file at the point where an error or restart occurred, minimizing the number of duplicate events sent downstream in the event of an interruption. To accomplish this, Cribl Stream keeps track of how many events have been processed for a given file. Resuming processing after an interruption, the Source skips the first *N* events it has seen and starts processing from event *N*+1.

The Source manages checkpoint information in memory, but writes it out to disk periodically (once per second, and on Worker shutdown) so that the information can survive a Worker restart. Cribl Stream keeps checkpoint information at the Worker Process level and does not share it between Worker Processes or Worker Nodes.

Checkpointing is a feature to keep in mind when you do disaster recovery (DR) planning.

Caveats to using checkpointing include: Whenever Worker Node state is destroyed (that is, when a Worker ceases to exist), checkpoint information will be lost. In Cribl.Cloud deployments, this can happen during upgrades. Also, because checkpoints are not written to disk on **every** event, the checkpoints written to disk will lag behind those that the S3 Source manages in memory. If the Worker Process stops abruptly without a clean shutdown, checkpoints not yet written to disk will be lost. When processing resumes, the Source will start from just after the last checkpoint written to disk, producing duplicate events corresponding to the lost checkpoints.

Located in the UI at **Advanced Settings** > **Checkpoint Settings** > **Enable checkpointing**, this feature is turned off by default.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__inputId`
- `__source`
- `_time`

How to Configure S3 to Send Event Notifications to SQS

1. Create a Standard SQS Queue. Note its ARN.
2. Replace its access policy with one similar to the examples below. To do so, select the queue; and then, in the **Permissions** tab, click: **Edit Policy Document (Advanced)**. (These examples differ only at line 9, showing public access to the SQS queue versus S3-only access to the queue.)
3. In the Amazon S3 console, add a notification configuration to publish events of the `s3:ObjectCreated:*` type to the SQS queue.

Permissive SQS access policy | Restrictive SQS access policy

```
{
 "Version": "2012-10-17",
 "Id": "example-ID",
 "Statement": [
  {
   "Sid": "<SID name>",
   "Effect": "Allow",
   "Principal": {
    "AWS":"*"
   },
   "Action": [
    "SQS:SendMessage"
   ],
   "Resource": "example-SQS-queue-ARN",
   "Condition": {
      "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:example-bucket-name" }
   }
  }
 ]
}
```

Copy

S3 and SQS Permissions

The following permissions are required on the S3 bucket:

- `s3:GetObject`
- `s3:ListBucket`

Needed when **Tag after processing** is toggled on:

- `s3:GetObjectTagging`
- `s3:PutObjectTagging`

The following permissions are required on the SQS queue:

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`

- `sqs:ChangeMessageVisibility`
- `sqs:GetQueueAttributes`
- `sqs:GetQueueUrl`

To collect encrypted data, such as CloudTrail logs, you'll need to add access to the KMS resources in your KMS Key Policy. Add the IAM user or role to the **Principal** section of your Key Policy and provide the `kms:Decrypt` **Action**. See Amazon's documentation on Key Policies for more information, including how to view and change your policy.

Temporary Access via SSO Provider

You can use Okta or SAML to provide access to S3 buckets using temporary security credentials.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

How Cribl Stream Pulls Data

Workers poll message from SQS. The call will return messages if they are available, or will time out after 1 second if no messages are available.

Each Worker gets its share of the load from S3. By default, S3 returns a maximum of 1 message in a single poll request. You can change this default in **Max messages**.

Best Practices

Beyond these basics, also see our Amazon S3 Better Practices and Using S3 Storage and Replay guides:

- When Cribl Stream instances are deployed on AWS, use IAM Roles whenever possible.
    - Not only is this safer, but it also makes the configuration simpler to maintain.
- Although optional, we highly recommend that you use a **Filename Filter**.
    - This will ensure that Cribl Stream ingests only files of interest.
    - Ingesting only what's strictly needed improves latency, processing power, and data quality.
- If higher throughput is needed, increase **Advanced Settings** > **Number of Receivers** and/or **Max messages**. However, do note:
    - These are set at `1` by default. As a result, **each** Worker Process, in **each** Cribl Stream Worker Node, will run one receiver consuming one message (that is, S3 file) at a time.

    - Total S3 objects processed at a time per Worker Node = Worker Processes x Number of Receivers x Max Messages
    - Increased throughput implies additional CPU utilization.
- When ingesting large files, tune up the **Visibility Timeout**, or consider using smaller objects.
    - The default value of `600s` works well in most cases, and while you certainly can increase it, we suggest that you also consider using smaller S3 objects.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Cribl University offers an Advanced Troubleshooting > Source Integrations: S3 short course. To follow the direct course link, first log into your Cribl University account. (To create an account, select the **Sign up** link. You'll need to click through a short **Terms & Conditions** presentation, with chill music, before proceeding to courses - but Cribl's training is always free of charge.) Once logged in, check out other useful Advanced Troubleshooting short courses and Troubleshooting Criblets.

VPC Endpoints

VPC endpoints for SQS and for S3 might need to be set up in your account. Check with your administrator for details.

Connectivity Issues

If you're having connectivity issues, but having no problems with the CLI, see if the AWS CLI proxy is in use. Check with your administrator for details.

Common Issues

"Inaccessible host: 'sqs.us-east-1.amazonaws.com'. This service may not be available..."

Full Error Text: `"message":"Inaccessible host: 'sqs.us-east-1.amazonaws.com'. This service may not be available in the 'us-east-1' region.","stack":"UnknownEndpoint: Inaccessible host: 'sqs.us-east-1.amazonaws.com'. This service may not be available in the 'us-east-1' region.`

If this is persistent rather than intermittent then it could be caused by TLS negotiation failures. For example, AWS SQS currently does not support TLS 1.3. If intermittent then a network-related issue could be occurring such as DNS-related problems.

“Missing credentials in config” or “stack:Error: connect ETIMEDOUT 169.254.169.254:80”

Can occur when **Authentication** is set to **Auto**, but no IAM role is attached.

Recommendation

Change to Manual Authentication; attach an IAM role; or if using IMDv2, switch to IMDv1 (if possible).

Missing credentials in config

“Bucket does not exist - self signed certificate”

Example Error Text:

8.1.2.4. Amazon Security Lake Source

Cribl Stream supports receiving data from Amazon Security Lake. Amazon Security Lake aggregates, normalizes, and stores security-related data from an AWS system. It outputs data using the Open Cybersecurity Schema Framework (OCSF). It's important not to confuse it with Amazon Security Hub, which is a different product.

Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **YES**

Cribl Stream running on Linux (only) can use this Source to read Parquet files, identified by a `.parquet`, `.parq`, or `.pqt` filename extension.

You can use the Amazon Security Lake Source in Cribl Stream to:

- Transform, trim, or shape incoming data from Amazon Security Lake and then send it to a downstream Destination, such as your preferred security analytics tool.
- Collect security data from multiple AWS cloud environments, different Regions, or different services in the Amazon ecosystem.

Relationship Between Amazon Security Lake and S3

The Amazon Security Lake Source functions nearly identically to the Amazon S3 Source. If needed, you can refer to the Amazon S3 Source documentation for more in-depth coverage of additional best practices and troubleshooting workarounds.

Because they are similar, you can use both the Amazon S3 Source and the Amazon Security Lake Source interchangeably as needed. Consider using both Sources together if your organization uses these different tools for different purposes and it is easier to manage them as separate and distinct data flows.

Both the Amazon Security Lake Source and the Amazon S3 Source can receive data using either of these methods:

- Amazon Security Lake event subscriptions
- S3 Bucket event notifications through Amazon Simple Queue Service (SQS)

These two notifications use the same method of sending data objects to a queue, but the underlying data format from these two methods differ. Cribl Stream supports both data structures. However, the Amazon Security Lake Source does not currently support pulling data from HTTPS endpoints.

You can customize the data template by configuring the default input transformer associated with Amazon Security Lake. However, changing the way Amazon Security Lake represents data could cause

> issues with the ability for Cribl Stream to consume data from Amazon Security Lake. Only advanced AWS administrators should change the default input transformer.

How the Amazon Security Lake Source Handles Data

When Amazon Security Lake collects security data, it pushes the data object (a parquet file) to a queue located in an S3 bucket. Amazon Security Lake then sends an event notification or a subscriber notification to Cribl Stream. The notification contains the object location data in the S3 bucket. Cribl Stream pulls the data object and deserializes it for further data processing.

Amazon Security Lake uses Amazon S3 as its backend service. For that reason, see the Amazon S3 Source documentation for additional information about how the Amazon Security Lake Source handles data, including SQS messages.

Configure an Amazon Security Lake Source

In Cribl Stream, to set up an Amazon Security Lake Source:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this S3 Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Queue**: The name, URL, or Amazon Resource Name (ARN) of the SQS queue to read events from. When specifying a non-AWS URL, you must use the format: `{url}/<queueName>`. (For example: `https://host:port/<queueName>`.) This value must be a JavaScript expression (which can evaluate to a constant), enclosed in single quotes, double quotes, or backticks.
3. Next, you can configure the following **Optional Settings**:
    - **Filename filter**: Regex matching file names to download and process. Defaults to `.*`, to match all characters. This regex will be evaluated against the S3 key's full path.
    - **Region**: AWS Region where the S3 bucket and SQS queue are located. Required, unless the **Queue** entry is a URL or ARN that includes a Region.

- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. In the **Authentication** settings, select one of the following options from the menu:

Option	Description
Auto	This default option uses the AWS SDK for JavaScript to check for credentials in a specific order by priority. It first checks for IAM Roles for Amazon EC2, then a shared credentials file, then environmental variables, and so forth. See Authentication Settings for more information.
Manual	If not running on AWS, select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, such as those running a private cloud. See Authentication Settings for more information.
Secret	If not running on AWS, select this option to supply a stored secret that references an AWS access key and secret key. See Authentication Settings for more information.

5. If needed, configure the **AssumeRole** settings. You can use these settings to access resources in a different Region than Cribl Stream. See Assume Role Settings for more information.
6. Optionally, refer to the following sections for the remaining settings:
   - Processing Settings
   - Advanced Settings
   - Connected Destinations
7. Select **Save**, then **Commit & Deploy**.
8. Verify that data is flowing correctly to Cribl Stream by viewing the **Live Data** feed from the Source.

Authentication Settings

Use the **Authentication method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.

5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, such as those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

The values for **Access key** and **Secret key** can be a constant or a JavaScript expression (such as `${C.env.MY_VAR}`). Enclose values containing special characters (like, `/`) or using environment variables in single quotes or backticks.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select a secret key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role Settings

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for Amazon S3: Whether to use Assume Role credentials to access S3. Defaults to toggled on.

Enable for Amazon SQS: Whether to use Assume Role credentials when accessing SQS. Default is toggled off.

AWS account ID: SQS queue owner's AWS account ID. Leave empty if the SQS queue is in the same AWS account.

AssumeRole ARN: Enter the ARN of the role to assume. When this Source runs on Cribl.Cloud Worker Groups and uses Assume Role to access Amazon Security Lake buckets or queues in another AWS account, configure the target IAM role's trust policy to allow your Cribl Worker role ARN to assume it. To get the role ARN for Workers in a Worker Group, see Worker Group Details.

External ID: Enter the External ID to use when assuming role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

Enabled: Default is toggled off. Toggle on to enable the custom command.

Command: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).

Arguments: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

This section defines event breaking rulesets that will be applied, in order.

Event Breaker Rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.

- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Include notification metadata: Toggle on to add the S3 event notification payload to a `__sqsMetadata` field on every event from the corresponding S3 file.

This metadata is useful for several purposes. You can measure processing latency by comparing the object's `eventTime` to the event's `_time`, or implement region-based routing using the `awsRegion` field in a Pipeline. The full payload also serves as a definitive record for auditing the origin of the data.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Checkpoint Settings

Enable checkpointing: When toggled off (default) Cribl Stream will always start processing a file from the beginning. Toggle on if, after an error or restart occurs, you want Cribl Stream to resume processing a given file from the point where it was interrupted, as explained below. When turned on, the following additional setting becomes available:

- **Retries**: The number of times you want Cribl Stream to retry processing a file after an error or restart occurs. If **Skip file on error** is enabled, this setting is ignored.

Other Settings

Endpoint: S3 service endpoint. If empty, defaults to AWS's Region-specific endpoint. Otherwise, used to point to an S3-compatible endpoint. To access the AWS S3 endpoints, use the path-style URL format. You don't need to specify the bucket name in the URL, because Cribl Stream will automatically add it to the URL path. For details, see AWS' Path-Style Requests topic.

Signature version: Signature version to use for signing SQS requests. Defaults to `v4`.

Message limit: The maximum number of messages that SQS should return in a poll request. Amazon SQS never returns more messages than this value. (However, fewer messages might be returned.) Acceptable

values: 1 to 10. Defaults to `1`.

Visibility timeout seconds: The duration (in seconds) that the received messages are hidden from subsequent retrieve requests, after being retrieved by a ReceiveMessage request. Defaults to `600`.

Cribl Stream will automatically extend this timeout until the initial request's files have been processed - notably, in the case of large files that require additional processing time.

Number of receivers: The number of receiver processes to run. The higher the number, the better the throughput, at the expense of CPU overhead. Defaults to `1`.

Poll timeout (secs): How long to wait for events before polling again. Minimum `1` second; default `10`; maximum `20`. Short durations increase the number - and thus the cost - of requests sent to AWS. (The UI will show a warning for intervals shorter than `5` seconds.) Long durations increase the time the Source takes to react to configuration changes and system restarts.

Socket timeout: Socket inactivity timeout (in seconds). Increase this value if retrievals time out during backpressure. Defaults to `300` seconds.

Parquet chunk size limit (MB): Maximum size for each Parquet chunk. Defaults to `5` MB. Valid range is `1` to `100` MB. Cribl Stream stores chunks in the location specified by the `CRIBL_TMP_DIR` environment variable. It removes the chunks immediately after reading them.

Parquet chunk download timeout (seconds): The maximum time to wait for a Parquet file's chunk to be downloaded. If a required chunk cannot be downloaded within this time limit, processing will end. Defaults to `600` seconds. Valid range is `1` second to `3600` seconds (1 hour).

Skip file on error: Toggle on to skip files that trigger a processing error (for example, corrupted files). Default is toggled off, which enables retries after a processing error.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Encoding: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Checkpointing

When Workers restart, checkpointing can prevent the Source from losing data or collecting duplicate events. Among the situations where this works well:

- When Cribl Stream pushes a new configuration: the Leader pushes the configuration to the Workers, and restarts them.
- When a Worker has gotten into a bad state, and you want to restart it.

Checkpointing allows Cribl Stream to resume processing a file at the point where an error or restart occurred, minimizing the number of duplicate events sent downstream in the event of an interruption. To accomplish this, Cribl Stream keeps track of how many events have been processed for a given file. Resuming processing after an interruption, the Source skips the first *N* events it has seen and starts processing from event *N*+1.

The Source manages checkpoint information in memory, but writes it out to disk periodically (once per second, and on Worker shutdown) so that the information can survive a Worker restart. Cribl Stream keeps checkpoint information at the Worker Process level and does not share it between Worker Processes or Worker Nodes.

Checkpointing is a feature to keep in mind when you do disaster recovery (DR) planning.

Caveats to using checkpointing include: Whenever Worker Node state is destroyed (that is, when a Worker ceases to exist), checkpoint information will be lost. In Cribl.Cloud deployments, this can happen during upgrades. Also, because checkpoints are not written to disk on **every** event, the checkpoints written to disk will lag behind those that the S3 Source manages in memory. If the Worker Process stops abruptly without a clean shutdown, checkpoints not yet written to disk will be lost. When processing resumes, the Source will start from just after the last checkpoint written to disk, producing duplicate events corresponding to the lost checkpoints.

Located in the UI at **Advanced Settings** > **Checkpoint Settings** > **Enable checkpointing**, this feature is turned off by default.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__inputId`

- `__source`
- `_time`

Troubleshooting

Navigate to and open the Amazon Security Lake Source, then use the Source's **Job Inspector** tab to see the results of recent collection runs.

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

For More Information

The Amazon Security Lake Source functions nearly identically to the Amazon S3 Source. If needed, you can refer to the Amazon S3 Source documentation for information about:

- How to configure S3 to send event notifications to SQS.
- How Cribl Stream pulls data.
- Best practices.

8.1.2.5. Amazon SQS Source

Cribl Stream supports receiving events from Amazon Simple Queuing Service.

Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **No**

Configure Cribl Stream to Receive Data from Amazon SQS

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - ***Input ID**: Enter a unique name to identify this SQS Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Queue**: The name, URL, or ARN of the SQS queue to read events from. This value must be a JavaScript expression (which can evaluate to a constant), enclosed in single quotes, double quotes, or backticks. To specify a non-AWS URL, use the format: `'{url}/<queueName>'`. (For example, `':port/<myQueueName>'`.)
    - **Queue type**: The queue type used (or created). Defaults to `Standard`. `FIFO` (First In, First Out) is the other option.
3. Next, you can configure the following **Optional Settings**:
    - **Create queue**: If toggled on, Cribl Stream will create the queue if it does not exist.
    - **Region**: AWS Region where the SQS queue is located. Required, unless the **Queue** entry is a URL or ARN that includes a Region.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Authentication, AssumeRole, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Authentication method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select a secret key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for Amazon SQS: Whether to use Assume Role credentials to access SQS. Default is toggled off.

AWS account ID: SQS queue owner's AWS account ID. Leave empty if SQS queue is in same AWS account.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume. If this Source runs on Cribl.Cloud Worker Groups and reads from an SQS queue in another AWS account, configure the target role's trust policy to allow your Cribl Worker role ARN to assume it. To get the role ARN for Workers in a Worker Group, see Worker Group Details.

External ID: Enter the external ID to use when assuming role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Endpoint: SQS service endpoint. If empty, the endpoint will be automatically constructed from the AWS Region.

Signature version: Signature version to use for signing SQS requests. Defaults to `v4`; `v2` is also available.

Message limit: The maximum number of messages that SQS should return in a poll request. Amazon SQS never returns more messages than this value. (However, fewer messages might be returned.) Acceptable values: `1` to `10`. Defaults to `10`.

Visibility timeout seconds: The duration (in seconds) that the received messages are hidden from subsequent retrieve requests, after they're retrieved by a `ReceiveMessage` request. Defaults to `600`.

Number of receivers: The number of receiver processes to run. The higher the number, the better the throughput, at the expense of CPU overhead. Defaults to `3`.

Poll timeout (secs): How long to wait for events before polling again. Minimum `1` second; default `10`; maximum `20`. Short durations increase the number - and thus the cost - of requests sent to AWS. (The UI will show a warning for intervals shorter than `5` seconds.) Long durations increase the time the Source takes to react to configuration changes and system restarts.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to reject certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on, the restrictive option.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__inputId`
- `__raw`
- `__receivedTs`
- `__sqsMessageId`
- `__sqsReceiptHandle`
- `_time`

SQS Permissions

The following permissions are needed on the SQS queue:

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`
- `sqs:GetQueueUrl`
- `sqs:CreateQueue` (optional, if and only if you want Cribl Stream to create the queue)

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

VPC endpoints for SQS might need to be set up in your account. Check with your administrator for details.

Common Issues

“Inaccessible host: ‘sqs.us-east-1.amazonaws.com’. This service may not be available…”

Full Error Text: `"message":"Inaccessible host: 'sqs.us-east-1.amazonaws.com'. This service may not be available in the 'us-east-1' region.","stack":"UnknownEndpoint: Inaccessible host: 'sqs.us-east-1.amazonaws.com'. This service may not be available in the 'us-east-1' region.`

If this is persistent rather than intermittent then it could be caused by TLS negotiation failures. For example, AWS SQS currently does not support TLS 1.3. If intermittent then a network-related issue could be occurring such as DNS-related problems.

How Cribl Stream Pulls Data

Workers poll messages from SQS. The call will return a message if one is available, or will time out after 1 second if no messages are available.

Each Worker gets its share of the load from SQS, and it receives a notification of a file newly added to an S3 bucket. By default, SQS returns a maximum of 10 messages in a single poll request.

8.1.3. Azure

You can use the following Sources to receive data from Azure services.

Source	Description
Azure Blob Storage	Receive data from Azure Blob Storage buckets
Azure Event Hubs	Receive data records from Azure Event Hubs

8.1.3.1. Azure Blob Storage Source

Cribl Stream supports receiving data from Azure Blob Storage buckets. Cribl Stream uses Azure Event Grid to receive notifications, via a queue, when new blobs are added to a storage account.

Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **YES** Available in: Cribl Stream (LogStream) 2.4.4 and above.

Cribl Stream running on Linux (only) can use this Source to read Parquet files, identified by a `.parquet`, `.parq`, or `.pqt` filename extension.

For configuration examples, see Resources below.

Restrictions

Cribl Stream supports data ingestion from Azure's **hot** and **cool** access tiers, but not from the **archive** tier - whose stated retrieval lag, up to several hours, cannot guarantee data availability.

This Source supports block blobs, but not append blobs, which can change after they are initially created and the create message is sent. Consider using a Cribl Stream Azure Event Hubs Source if you need to ingest changeable Azure data.

Configure Cribl Stream to Receive Data from Azure Blob Storage

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Azure Blob Storage Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Queue**: The queue name from which to read Blob notifications. Value must be a JavaScript expression (which can evaluate to a constant value), enclosed in quotes or backticks. Can be

evaluated only at init time. For example, referencing a Global Variable: `myQueue-${C.vars.myVar}`.

3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Use this default option to enter your Azure Storage connection string directly. Exposes a **Connection string** field for this purpose. (If left blank, Cribl Stream will fall back to `env.AZURE_STORAGE_CONNECTION_STRING`.)
    - **Secret**: This option exposes a **Connection string (text secret)** drop-down, in which you can select a stored secret that references an Azure Storage connection string. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need a new secret. For details, see Connection String Format.
    - **Client secret**: This option allows you to use the Azure service principal's client secret to authenticate, and exposes Azure service principal settings.
    - **Certificate**: This option allows you to use a certificate registered with the Azure service principal to authenticate, and exposes Azure service principal settings.
4. Next, you can configure the following **Optional Settings**:
    - **Filename filter**: Regex matching file names to download and process. Defaults to `.*`, to match all characters.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Processing and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Connection String Format

Manual and **Secret** authentication methods use an Azure Storage connection string in this format:

```
DefaultEndpointsProtocol=[http|https];AccountName=<your-account-name>;AccountKey=<you
```

Copy

A fictitious example, using Microsoft's recommended HTTPS option, is:

```
DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=12345678...32;Enc
```

Copy

Service Principal Settings

Client secret and **Certificate** authentication methods expose the following additional settings:

- **Storage account name**: Enter the name of your Azure Storage Account.
- **Tenant ID**: Enter the service principal's tenant ID.
- **Client ID**: Enter the service principal's client ID.
- **Client secret (text secret)** (with **Client secret** method selected): Select a text secret containing the client secret, or create a new one.
- **Certificate** (with **Certificate** method selected): Select the certificate you registered as credentials for your app in the Azure portal, or create a new one.
- **Azure Cloud**: Select the Azure cloud environment to use, `Azure Public Cloud` (default), `Azure China`, or `Azure Government`. The **Endpoint suffix** advanced setting is available if you need to override the default values.
    - Selecting `Azure China` sets the authority host to `login.chinacloudapi.cn` and the endpoint suffix to `core.chinacloudapi.cn`.
    - Selecting `Azure Government` sets the authority host to `login.microsoftonline.us` and the endpoint suffix to `core.usgovcloudapi.net`.

Processing Settings

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

Enabled: Default is toggled off. Toggle on to enable the custom command.

Command: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).

Arguments: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

This section defines event breaking rulesets that will be applied, in order.

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Pipelines. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

For processing Azure Virtual Network flow data through this Source, use the specialized Azure VNet Flow Event Breaker to automatically unroll and structure nested flow records.

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Message limit: The maximum number of messages to return in a poll request. Azure queues never return more messages than this value (although they might return fewer messages). Acceptable values: `1` to `32`.

Visibility timeout (secs): The duration (in seconds) that the received messages are hidden from subsequent retrieve requests, after being retrieved by a `ReceiveMessage` request. Defaults to `600` seconds. Maximum allowed value is `604800` seconds (7 days).

Cribl Stream will automatically extend this timeout until the initial request's files have been processed - notably, in the case of large files that require additional processing time.

Number of receivers: The number of receiver processes to run. The higher the number, the better the throughput, at the expense of CPU overhead. Defaults to `1`.

Service period (secs): The interval (in seconds) at which pollers should be validated, and restarted if exited. Defaults to `5` seconds.

Skip file on error: Toggle on to skip files that trigger a processing error (for example, corrupted files). Default is toggled off, which enables retries after a processing error.

Parquet chunk size limit: Maximum size for each Parquet chunk. Defaults to `5` MB. Valid range is `1` to `100` MB. Cribl Stream stores chunks in the location specified by the `CRIBL_TMP_DIR` environment variable. It removes the chunks immediately after reading them.

Parquet chunk download timeout (seconds): The maximum time to wait for a Parquet file's chunk to be downloaded. If a required chunk cannot not be downloaded within this time limit, processing will end. Defaults to `600` seconds. Valid range is `1` second to `3600` seconds (1 hour).

Endpoint suffix: (Optional) Overrides the default endpoint suffix set by the **Azure Cloud** setting. Use this only if you need to connect to a custom or non-standard Azure Blob Storage endpoint.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Azure Permissions

When configuring an Azure Source in Cribl Stream, you must grant the necessary permissions for it to interact with your Azure environment. Cribl recommends creating an Azure service principal with the minimum required permissions using Azure role-based access control (Azure RBAC). This approach ensures Cribl Stream has a dedicated identity with precisely the access it needs, following the principle of least privilege.

The minimum permissions required for Sources are:

- **Storage Blob Data Reader**
- **Storage Blob Data Contributor**

See Permissions for calling data operations in the official Microsoft documentation for more information.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__accountName`
- `__final`
- `__inputId`
- `__source`
- `__topic`
- `_time`

The remainder of this topic covers required Azure-side configuration.

Configuring Azure Blob Notifications

This Source needs to receive Azure Event Grid notifications, via a queue, when new blobs are added to a storage account. This queue approach enables Cribl Stream to manage backpressure conditions and retries upon errors.

You will therefore need to enable notifications in the Azure portal. The basic flow is:

File upload → Blob container → Blob Created notification → Azure Queue Storage queue

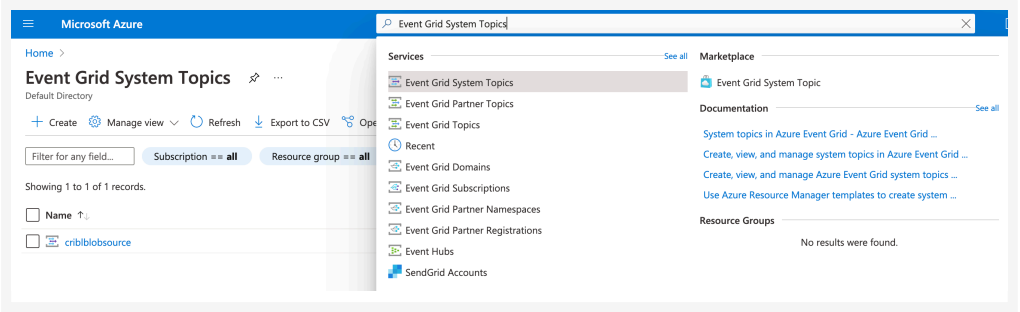
To configure notifications from the Blob storage account in the Azure backend, there are three major steps, outlined below:

1. Create an Event Grid system topic.
2. Create a queue.
3. Configure the generation of storage account notifications when new blobs are uploaded to the queue.

Azure's UI will change over time. Please fall back to Microsoft's Azure Event Grid documentation for up-to-date instructions and screenshots.

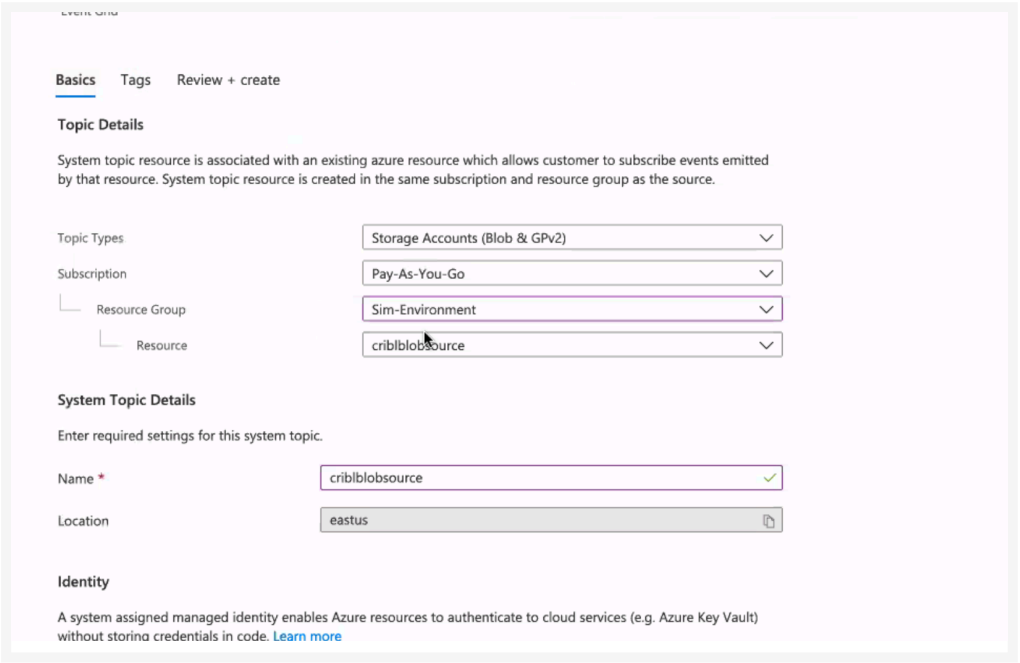
1. Create System Topic

First, you must create a system topic, to which Azure will publish notifications. In the Azure portal, start at **Event Grid System Topics**:



Azure portal > System topics

Select **+Create** to create a new system topic, then set the **Topic Type** to **Storage Account (Blob)**:



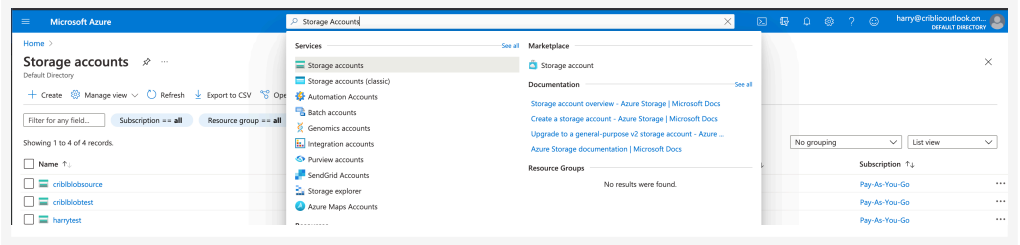
Creating a system topic

In **Subscription > Resource Group > Resource**, reference the storage account where you want to generate notifications.

Give the topic an arbitrary name that is meaningful to you. (In this example, the name is the same as the storage account.)

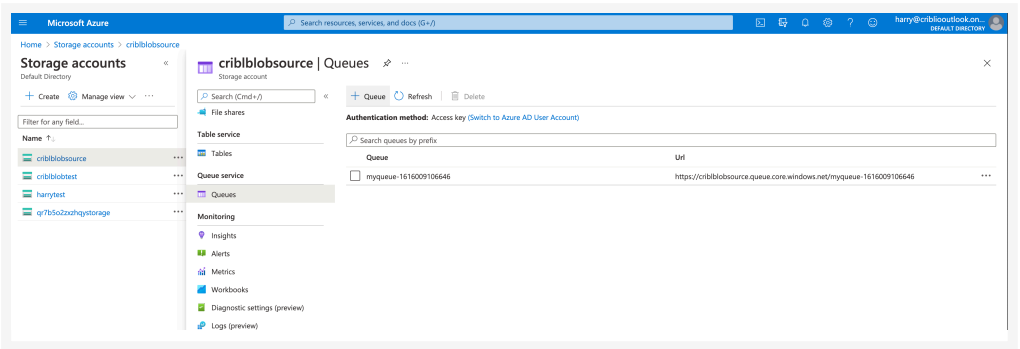
2. Create Storage Queue

Next, navigate to your storage account to create a queue.



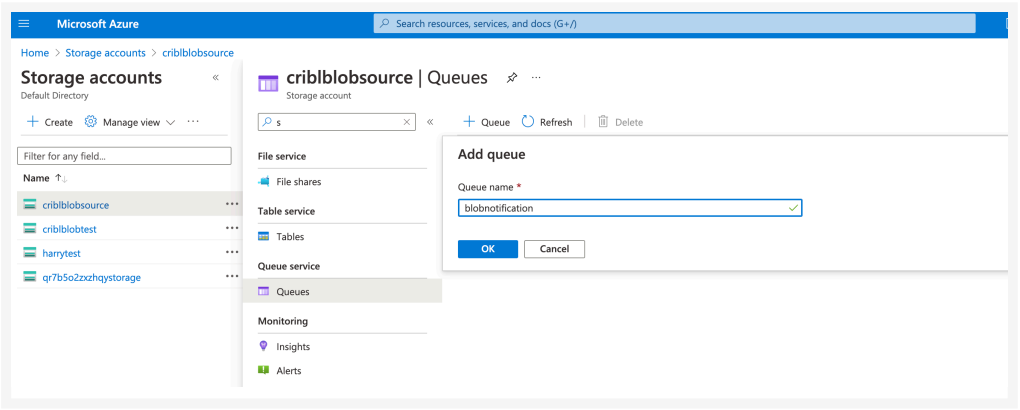
Accessing your storage account

Select the storage account for which you would like to set up notifications. Then, in the submenu, select **Queue service > Queues**:



Accessing queues

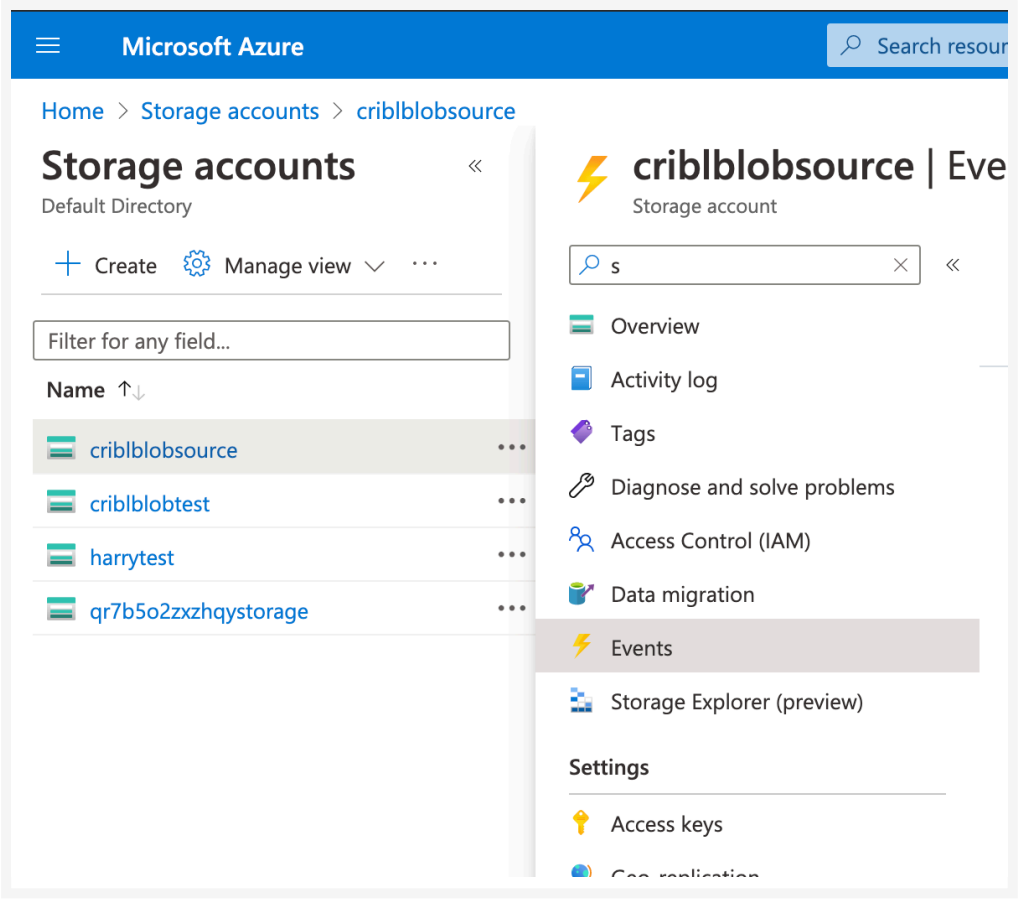
Select **Create queue**, and give the queue a name that is meaningful to you.



Adding a queue

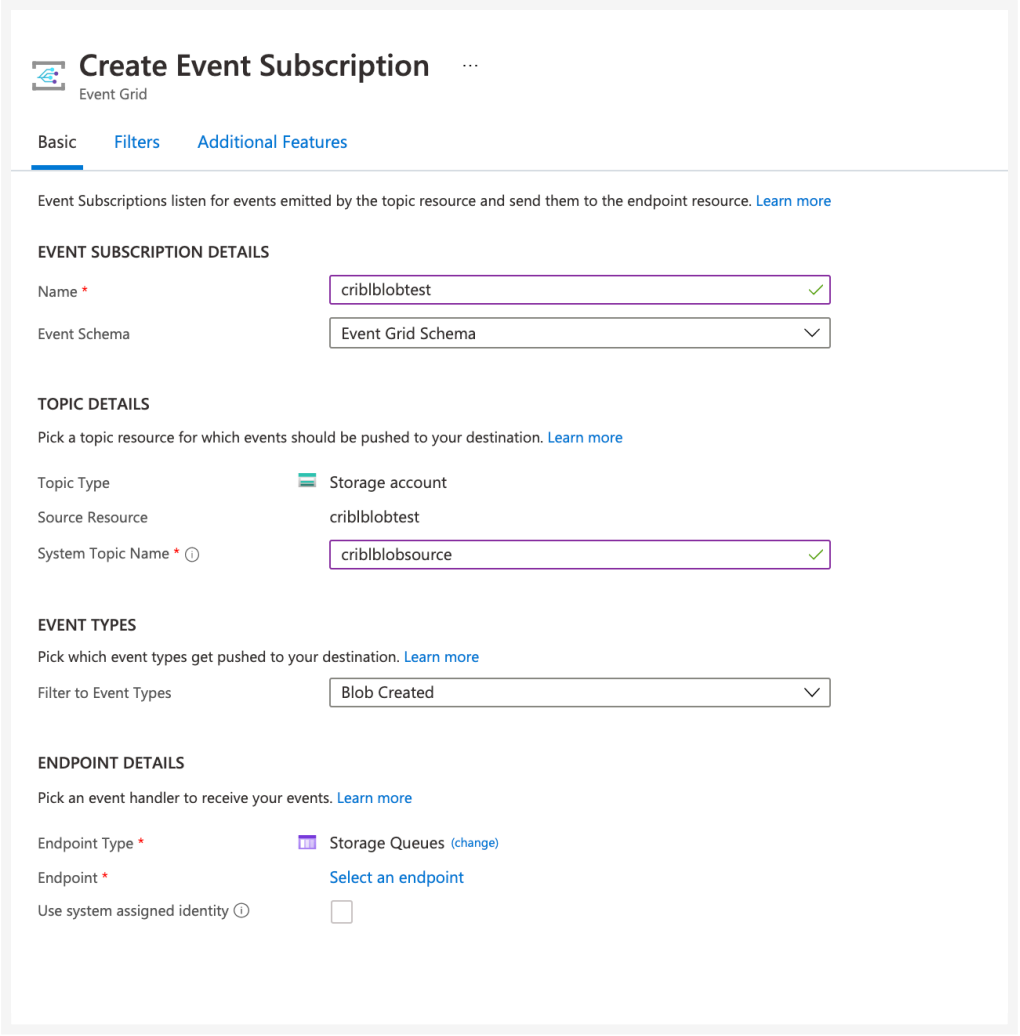
3. Configure Storage Account Notifications

Next, set up the storage account that will publish **Blob Create** notifications to the queue, using the system topic. From the **Storage Accounts** menu, select **Events**:



Accessing your storage account

Then click **+ Event Subscription** to proceed:

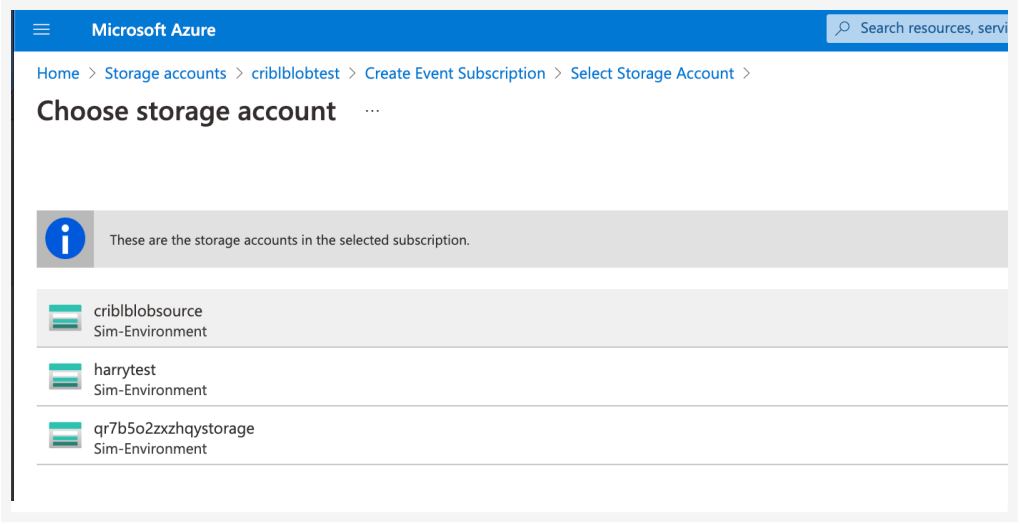


Creating a subscription.

There are a few things to configure here:

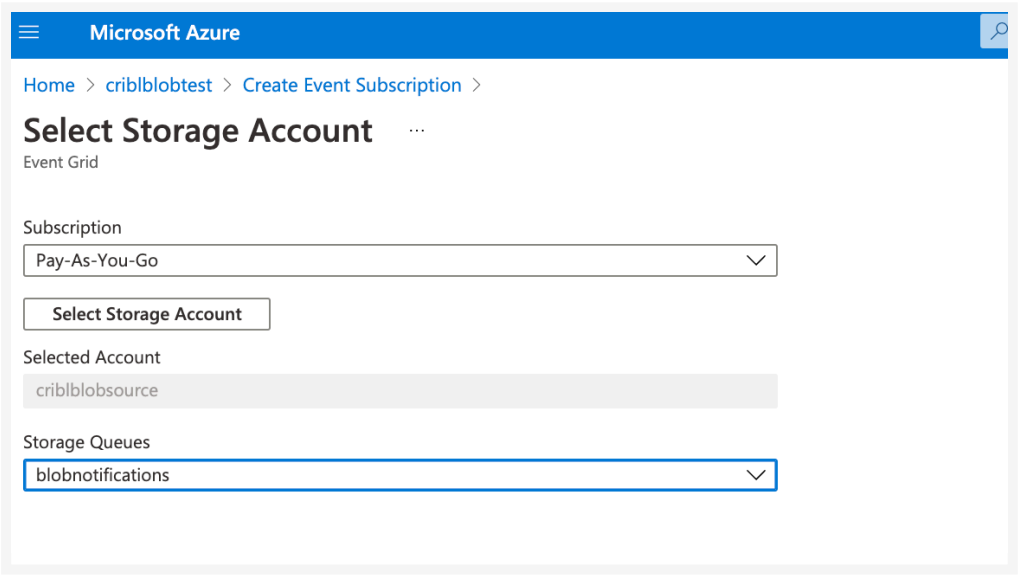
- Enter a **Name** for the subscription.
- In **System Topic Name**, enter the name of the system topic you created in 1. Create System Topic above.
- In **Event Types**, select **Blob Created**, and deselect **Blob Deleted**.
- As the **Endpoint Type**, select **Storage Queues**.
- Click **Select an endpoint**, and click the subscription to use (**Pay-As-You-Go**).

Next, select the storage account on which to add the subscription:



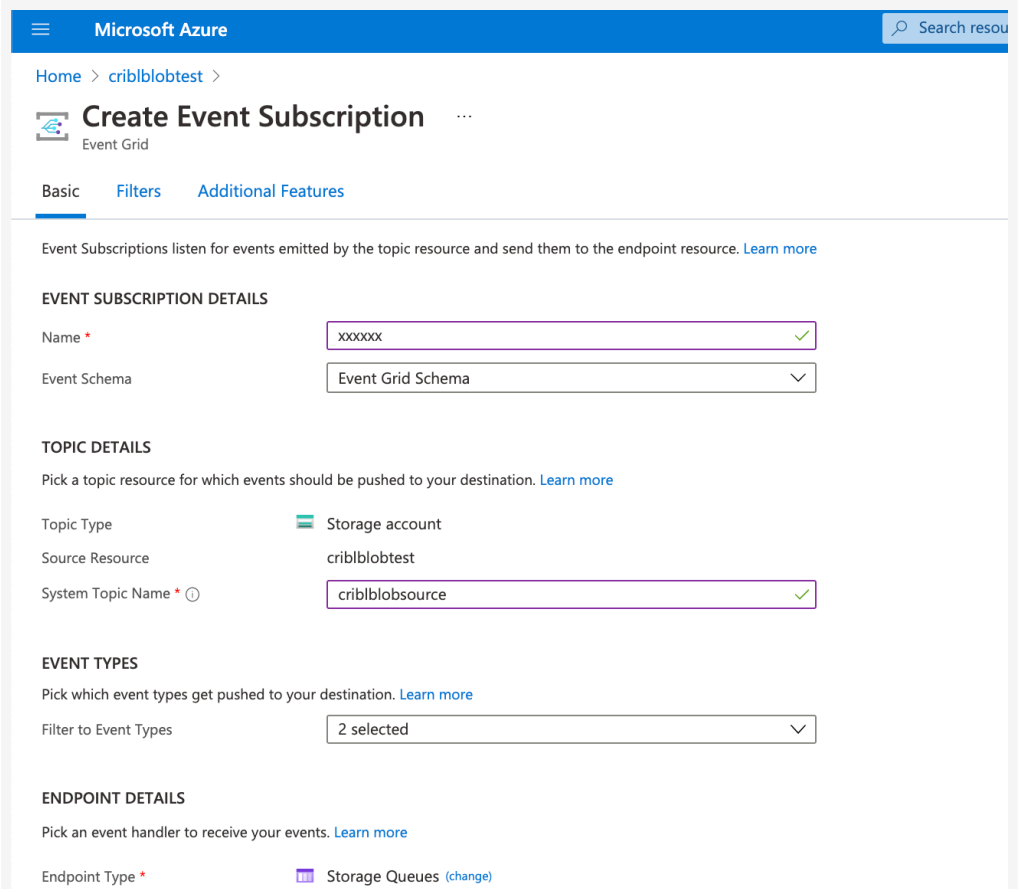
Choosing the storage account

Select the queue you created in Create Storage Queue above, and click **Confirm Selection** to save the settings.



Selecting the storage account

To complete the process, click **Create**.


Microsoft Azure
Search resou
Home
criblblobtest
Create Event Subscription
Event Grid
Basic
Filters
Additional Features
Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. Learn more
EVENT SUBSCRIPTION DETAILS
Name *
xxxxxx
Event Schema
Event Grid Schema
TOPIC DETAILS
Pick a topic resource for which events should be pushed to your destination. Learn more
Topic Type
Storage account
Source Resource
criblblobtest
System Topic Name *
criblblobsource
EVENT TYPES
Pick which event types get pushed to your destination. Learn more
Filter to Event Types
2 selected
ENDPOINT DETAILS
Pick an event handler to receive your events. Learn more
Endpoint Type *
Storage Queues (change)

8.1.3.2. Azure Event Hubs Source

Cribl Stream supports receiving data records from Azure Event Hubs.

Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **NO**

Azure Event Hubs uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must receive events directly from senders. You might need to adjust your firewall rules to allow this traffic.

For configuration examples, see Resources below.

Configure Cribl Stream to Receive Data from Azure Event Hubs

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Brokers**: List of Event Hubs Kafka brokers to connect to - for example, `yourdomain.servicebus.windows.net:9093`. Get the hostname from the host portion of the primary or secondary connection string in Shared Access Policies. If omitted, the port will default to `9093`.
    - **Event Hub name**: The name of the Event Hub (a.k.a. Kafka Topic) to subscribe to.
3. Next, you can configure the following **Optional Settings**:
    - **Group ID**: The name of the consumer group that includes this Cribl Stream instance. Defaults to `Cribl`.

To prevent excessive Kafka rebalancing and reduced throughput, each **Group ID** that you specify here should be subscribed to only one Kafka Topic - that is, only to the single Topic you specify in **Event Hub name**. This has a few implications:

- The **Group ID** should be something other than `$Default`, especially if Event Hubs are stored In shared accounts, where the `$Default` group might be subscribed to other Topics.
- The **Group ID** should also be unique for each of your Azure Event Hubs, Kafka, and Confluent Cloud Sources.
- You should configure a **separate** Azure Event Hubs Source for each Group:Topic pair whose events you want to subscribe to.

For details, see Controlling Rebalancing.

- **From beginning**: Whether to start reading from the earliest available data. Relevant only during initial subscription. Defaults to toggled on.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the TLS, Authentication, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Client Side)

Enabled: Defaults to toggled on.

Validate server certs: Whether to reject connections to servers without signed certificates. Defaults to toggled on.

Authentication Settings

Enabled: If toggled on (default), this section's remaining settings are displayed, and all are required settings.

SASL mechanism: SASL (Simple Authentication and Security Layer) authentication mechanism to use with Kafka brokers. Defaults to `PLAIN`, which exposes Basic Authentication options that rely on Azure Event Hubs connection strings. Select `OAUTHBEARER` to enable OAuth Authentication via a different set of options.

Basic Authentication

Selecting the `PLAIN` SASL mechanism provides the options listed in this section.

Username: The username for authentication. For Event Hubs, this should always be `$ConnectionString`.

Authentication method: Use the buttons to select one of these options:

- **Manual**: Use this default option to enter your Event Hubs connection string's primary or secondary key from the Event Hubs workspace. Exposes a **Password** field for this purpose.
- **Secret**: This option exposes a **Password (text secret)** drop-down, in which you can select a stored secret that references an Event Hubs connection string. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need a new secret.

Connection String Format

Either authentication method above uses an Azure Event Hubs connection string in this format:

```
Endpoint=sb://<FQDN>/;SharedAccessKeyName=<your-shared-access-key-name>;SharedAccessKey=<your-shared-access-key-value>
```

A fictitious example is:

```
Endpoint=sb://dummynamespace.servicebus.windows.net/;SharedAccessKeyName=DummyAccessKeyN
```

OAuth Authentication

Selecting the `OAUTHBEARER` SASL mechanism provides the options listed in this section.

Microsoft Entra ID authentication endpoint: Specifies the Microsoft Entra ID endpoint from which to acquire authentication tokens. Defaults to `https://login.microsoftonline.com`. You can instead select `https://login.microsoftonline.us` or `https://login.partner.microsoftonline.cn`.

Client ID: Enter the `client_id` to pass in the OAuth request parameter.

Tenant identifier: Enter your Microsoft Entra ID subscription's directory ID (tenant ID).

Scope: Enter the scope to pass in the OAuth request parameter. This will be of the form: `https://<Event-Hubs-Namespace-Host-name>/.default`. (For example, for an Event Hubs Namespace > Host name: `goatyoga.servicebus.windows.net`, **Scope**: `https://goatyoga.servicebus.windows.net/.default`.)

Authentication method: Use the buttons to select one of these options:

- **Manual**: This default option exposes a **Client secret** field, in which to directly enter the `client_secret` to pass in the OAuth request parameter.
- **Secret**: Exposes a **Client secret (text secret)** drop-down, in which you can select a stored secret that references the `client_secret`. A **Create** link is available to define a new secret.
- **Certificate**: Exposes a **Certificate name** drop-down, in which you can select a stored certificate. A **Create** link is available to define a new cert.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries,

use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Use these settings to fine-tune Cribl Stream's integration with Event Hubs Kafka brokers. For details, see Azure Event Hubs' recommended configuration documentation. If you are unfamiliar with these parameters, contact Cribl Support to understand the implications of changing the defaults.

Heartbeat interval (ms): Expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. (Corresponds to `heartbeat.interval.ms` in the Kafka domain.) Value must be lower than `sessionTimeout`, and typically should not exceed 1/3 of the `sessionTimeout` value. Defaults to `3000` ms (3 seconds).

Session timeout (ms): Timeout used to detect client failures when using Kafka's group management facilities. (Corresponds to `session.timeout.ms` in the Kafka domain.) If the client sends the broker no heartbeats before this timeout expires, the broker will remove this client from the group, and will initiate a rebalance. Value must be lower than `rebalanceTimeout`. Defaults to `30000` ms (30 seconds).

Rebalance timeout (ms): Maximum allowed time for each worker to join the group after a rebalance has begun. (Corresponds to `rebalance.timeout.ms` in the Kafka domain.) If this timeout is exceeded, the coordinator broker will remove the worker from the group. Defaults to `60000` ms (1 minute).

Connection timeout (ms): Maximum time to wait for a successful connection. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour).

Request timeout (ms): Maximum time to wait for a successful request. Defaults to `60000` ms (1 minute).

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Source from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Source to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Offset commit interval (ms): How often, in milliseconds, to commit offsets. If both this field and the **Offset commit threshold** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Offset commit threshold: The number of events that will trigger an offset commit. If both this field and the **Offset commit interval** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Byte limit, per partition: The maximum amount of data that the server will return per partition. Must equal or exceed the maximum message size the server allows. (Otherwise, the producer will be unable to send messages larger than the consumer can fetch.) If not specified, defaults to `1048576`.

Byte limit: Maximum amount of bytes to accumulate in the response. The default is `10485760 (10 MB)`.

Error limit, per socket: Maximum number of consecutive request errors that can occur on a single socket connection before the connection is discarded and reestablished. This mitigates issues with idle connections, particularly those used for sending heartbeats, thereby reducing the number of rebalances in a consumer group. Default is `0`, which disables this feature. Accepts values `1-100`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Minimize duplicates: Toggle on to start only one consumer for each topic partition (optional). This reduces duplicates.

If you observe an excessive number of group rebalances, and/or you observe consumers not regularly pulling messages, try increasing the values of **Heartbeat interval**, **Session timeout**, and **Rebalance timeout**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Azure Permissions

When configuring an Azure Source in Cribl Stream, you must grant the necessary permissions for it to interact with your Azure environment. Cribl recommends creating an Azure service principal with the minimum

required permissions using Azure role-based access control (Azure RBAC). This approach ensures Cribl Stream has a dedicated identity with precisely the access it needs, following the principle of least privilege.

The minimum permissions required for Sources are:

- **Azure Event Hubs Data Receiver**

See Authorize access to Azure Event Hubs resources using Microsoft Entra ID in the official Microsoft documentation for more information.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`
- `__topicIn` (indicates the Kafka topic that the event came from)
- `__partition`
- `__schemaId` (when using Azure Schema Registry)
- `__key` (when using Schema Registry)
- `__headers` (when using Schema Registry)
- `__keySchemaIdIn` (when using Schema Registry)
- `__valueSchemaIdIn` (when using Schema Registry)

How Cribl Stream Pulls Data

Azure Event Hubs treat all the Worker Nodes as members of a Consumer Group, and each Worker gets its share of the load from Azure Event Hubs. This is the same process as normal Kafka. By default, Workers will poll every 5 seconds. In the case of Leader failure, Worker Nodes will continue to receive data as normal.

Randomized Partition Assignment

In Cribl Stream 4.8.2 and newer, partition assignment is randomized. The Worker Processes selected to consume data during partition assignment are selected randomly, reducing the possibility of uneven load distribution when there are fewer partitions than Worker Processes. Note that *random* assignment is not the same as *balanced* assignment, and even distribution of work is not guaranteed.

Controlling Rebalancing

When you configure multiple Sources that subscribe to different topics, but all belong to the same consumer group, a state change affecting any Source in this consumer group will affect all the other Sources. Examples of state changes include: deploying new configs, adding or removing Worker Processes, or Worker Processes crashing.

Here's an example - three Sources, three different topics, all in one consumer group:

- `Source_1 - Topic_1 - ConsumerGroup1`
- `Source_2 - Topic_2 - ConsumerGroup1`
- `Source_3 - Topic_3 - ConsumerGroup1`

Imagine that Source 1 undergoes a state change event, such as a Worker Process crash. Source 2 and Source 3 will rebalance - stopping data flow until the rebalance completes.

Shared Worker Group Mitigation

If Sources that share a consumer group all deploy as part of the same Worker Group, changes will have smaller side effects than when Sources are spread across different Worker Groups. (Conversely, imagine a configuration where deploying new configs for Worker Group 1 caused rebalancing of topics in worker Worker Group 2. This spillover would be especially undesirable.)

Bottom Line

Changes to any member of a consumer group affect all other members of that consumer group. To prevent this undesired behavior, make sure to use a unique **Group ID** for each Kafka, Confluent Cloud, and Azure Event Hubs Source.

Resources

For examples of configuring Cribl Stream to interoperate with Azure services, see these guides:

- Azure Event Hubs Integrations.
- SSO with Microsoft Entra ID (Cribl.Cloud).
- SSO with Microsoft Entra ID (On-Prem).

Also see these Packs on the Cribl Packs Dispensary, which provide processing Pipelines to transform and optimize incoming Azure data. You can directly import these Pipelines and adapt them to your needs:

- Microsoft Azure NSG Source to Common Schema.
- Azure NSG Flow Logs to OCSF.
- Azure Audit Logs to OCSF.
- Microsoft Office Activity & Azure Logs.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

KafkaJSProtocolError: Not authorized to access topics: [Topic authorization failed]

The username does not have read permissions for the specified topic.

8.1.4. Google Cloud

You can use the following Sources to receive data from Google Cloud services.

Source	Description
Google Cloud Pub/Sub	Receive data records from Google Cloud Pub/Sub

8.1.4.1. Google Cloud Pub/Sub Source

Cribl Stream supports receiving data records from Google Cloud Pub/Sub, a managed real-time messaging service for sending and receiving messages between applications.

Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **No**

Prerequisites

When working with Google Cloud Pub/Sub, you need to know:

- The project where your credentials originated.
- The project that contains any topic and subscription you want to work with.

If the desired topic and subscription reside in the same project where your credentials originated, describe them using either short names **or** fully-qualified names.

If the desired topic and subscription reside in a different project than the one where your credentials originated, you **must** describe them using fully-qualified names.

Here's an example of a fully-qualified topic name: `projects/my-project-id/topics/my-topic-id`.

The short name for the same topic would be: `my-topic-id`.

Configure Cribl Stream to Receive Data from Pub/Sub

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Pub/Sub Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.

- **Topic ID**: ID of the Pub/Sub topic from which to receive events. If **Monitor subscription for new messages** is enabled, this setting is set to a default value and ignored. Otherwise, it is required for topic or subscription creation.
- **Subscription ID**: ID of the subscription to use when receiving events. When **Monitor subscription for new messages** is enabled, the fully qualified subscription name must be entered. For example: `projects/myProject/subscriptions/mySubscription'`

3. Next, you can configure the following **Optional Settings**:
   - **Monitor subscription for new messages**: Toggle on to monitor and pull messages directly from the specified Pub/Sub subscription, without requiring a **Topic ID**. Use this when you only have access to a subscription or do not need to manage the topic. When enabled, **Create topic** and **Create subscription** are not relevant and are disabled.
   - **Create topic**: If toggled on, Cribl Stream will create the topic on Pub/Sub if it does not exist. Disabled when **Monitor subscription for new messages** is enabled.
   - **Create subscription**: If toggled on (default), Cribl Stream will create the subscription on Pub/Sub if it does not exist. Disabled when **Monitor subscription for new messages** is enabled.
   - **Ordered delivery**: If toggled on, Cribl Stream will receive events in the order that they were added to the queue. For ordered delivery to work correctly, ensure that both the process receiving events and the subscription from which messages are read have ordered delivery enabled.
   - **Region**: Region to retrieve messages from. Select `default` to allow Google to auto-select the nearest region. (If you've enabled **Ordered delivery**, the selected region must be allowed by message storage policy.)
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Authentication, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Google authentication method** drop-down to select a Google authentication method:

- **Auto**: This option uses the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to provide the location of a credential JSON file. The environment variable can also be set directly in JSON format. When this option is set, Cribl Stream will first check whether the `GOOGLE_APPLICATION_CREDENTIALS` environment variable exists and properly set with valid credentials. If this environmental variable is missing or incorrect, Cribl Stream checks for credentials in two additional fallback locations, as described in the official Google Cloud documentation: How Application Default Credentials works. In addition to using the `GOOGLE_APPLICATION_CREDENTIALS` environment variable, you can use credentials attached to your Google Cloud instance.

- **Manual**: With this default option, you use the **Service account credentials** field to enter the contents of your service account credentials file (a set of JSON keys), as downloaded from Google Cloud. To insert the file itself, click the upload button at this field's upper right. As an alternative, you can use environment variables, as outlined here.
- **Secret**: Use the drop-down to select a key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Backlog limit: When the Destination exerts backpressure, this setting limits the number of events that Cribl Stream will queue for processing before it stops retrieving further events. Defaults to `1000` events.

Number of concurrent streams: How many streams to pull messages from at one time. Doubling the value doubles the number of messages this Source pulls from the topic (if available), while consuming more CPU and memory. Defaults to `5`.

Request timeout (ms): Pull request timeout, in milliseconds. Defaults to `60000` ms (i.e., 1 minute).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__messageId` - ID of the message from Google.
- `__projectId` - ID of the Google project from which the data was received.
- `__publishTime` - Time at which the event was originally published to the Pub/Sub topic.
- `__subscriptionIn` - The subscription from which the event was received.
- `__topicIn` - The topic from which the event was received.

Google Cloud Roles and Permissions

Your Google Cloud service account should have at least the following roles on subscriptions:

- `roles/pubsub.subscriber`
- `roles/pubsub.viewer` or `roles/viewer`

As an alternative, you can rely on the following discrete permissions, which are included in the above roles:

- `pubsub.snapshots.seek`
- `pubsub.subscriptions.consume`
- `pubsub.topics.attachSubscription`

To enable Cribl Stream's **Create topic** and/or **Create subscription** options, your service account should have one of the following (or higher) roles:

- `roles/pubsub.editor`
- `roles/editor`

Either `editor` role confers multiple permissions, including those from the lower `viewer`, `subscriber`, and `publisher` roles. For additional details, see the Google Cloud Access Control topic.

How Cribl Stream Pulls Data

Pub/Sub treats all the Worker Nodes as members of a Consumer Group, and each Worker gets its share of the load from Pub/Sub. This is the same process as normal Kafka. By default, Workers will poll every 1 minute. In the case of Leader failure, Worker Nodes will continue to receive data as normal.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.5. KAFKA

You can use the following Sources to receive data from Kafka services.

Source	Description
Kafka	Receive data records from a Kafka cluster
Confluent Cloud	Receive Kafka topics from Confluent Cloud
Amazon MSK	Receive data records from an Amazon MSK cluster

8.1.5.1. Kafka Source

Cribl Stream supports receiving data records from a Kafka cluster. This Source automatically detects compressed data in `Gzip`, `Snappy`, `LZ4`, or `ZSTD` format, and automatically decompresses the data upon ingesting it.

Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **No**

Kafka uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must receive events directly from senders. You might need to adjust your firewall rules to allow this traffic.

Configure Cribl Stream to Receive Data from Kafka Topics

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Bootstrap servers**: Enter each Kafka bootstrap server you want to use. Specify the hostname and port (such as `mykafkabroker:9092`) or just the hostname (in which case Stream will assign port `9092`).
    - **Topic**: Enter the name(s) of topics to subscribe to. Press `Enter/Return` between multiple entries.

    To optimize performance and prevent excessive rebalancing, Cribl recommends subscribing each Kafka Source to only one topic. To subscribe to multiple topics, consider creating a dedicated Kafka Source for each one.

    For the same reason, the **Group ID** (below) should be unique for each of your Kafka, Confluent Cloud, and Azure Event Hubs Sources. For details, see Controlling Rebalancing.
3. Next, you can configure the following **Optional Settings**:

- **Group ID**: The name of the consumer group to which this Cribl Stream instance belongs.
- **From beginning**: Whether to start reading from the earliest available data. Relevant only during initial subscription. Defaults to toggled on.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the TLS, Authentication, Schema Registry, Processing, Retries, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Client Side)

Use TLS: Toggled off by default. When toggled on:

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled off.

Server name (SNI): Leave this field blank. See Connecting to Kafka below.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Certificate: The name of the predefined certificate.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Authentication

This section details the SASL (Simple Authentication and Security Layer) and OAuth authentication settings for connecting to bootstrap servers. For enhanced security, using TLS is highly recommended.

To begin, toggle **Enabled** on (the default is off). This presents you with options to configure authentication via `SASL` or `OAuth`.

SASL Authentication

Use the **SASL mechanism** dropdown to select your authentication method from the available options: `PLAIN`, `SCRAM-256`, `SCRAM-512`, and `GSSAPI/Kerberos`. The controls displayed will vary based on your selection.

PLAIN, SCRAM-256, or SCRAM-512

For these methods, you can provide credentials either through:

- **Manual**: Displays **Username** and **Password** fields to enter your Kafka credentials directly.
- **Secret**: This option exposes a **Credentials secret** drop-down in which you can select a stored text secret that references your Kafka credentials. A **Create** link is available to store a new, reusable secret.

GSSAPI/Kerberos

Kerberos authentication is not supported for Cribl-managed Workers in Cribl.Cloud, but it is enabled for customer-managed (hybrid or on-prem) Worker Groups, whether the Leader is based in Cribl.Cloud or on-prem.

If you select Kerberos, you must provide the following:

Keytab location: Enter the location of the keytab file for the authentication principal.

Principal: Enter the authentication principal, for example: `kafka_user@example.com`.

Broker service class: Enter the Kerberos service class for Kafka brokers, for example: `kafka`.

You will also need to set up your environment and configure the Cribl Stream host for use with Kerberos. For details, see Kafka Authentication with Kerberos.

Enable OAuth Authentication

Enabling OAuth will replace any existing authentication settings with placeholder values. Old configurations will continue to work without modification after an upgrade, and disabling OAuth will restore your original settings.

Toggle **Enable OAuth** on to configure this method. To configure, you must provide the following:

- **Token URL**: URL of the token endpoint to use for OAuth authentication.
- **Client ID**: ID to use for OAuth authentication.
- **Client secret (text secret)**: Select or create a stored text secret.
- **Add OAuth Parameters**: Optionally, click **Add parameter** for each additional OAuth paramater. These are parameters sent to the authentication provider in the body of the token request. They are often used

for `scope` (for example, in Amazon Cognito) or `audience` (for example, in `Auth0`).

In the resulting table, enter the **Name** of each field per row. The corresponding Value should be a literal value (such, a string, a number, or a boolean).

- **Add SASL Extension Fields**: Optionally, click **Add Field** for each additional SASL extension fields. These are fields that are set as extensions when authenticating with the Kafka cluster. For example, Confluent Cloud uses fields like `logicalCluster` or `identityPoolId` for granular access control. You do not need to include the `extension_ prefix` when configuring these fields in Cribl.

In the resulting table, enter the **Name** of each field per row. The corresponding Value should be a literal value (such, a string, a number, or a boolean).

Schema Registry

This section governs Kafka Schema Registry Authentication for data encoded with Avro or JSON schemas, with schemas stored in the Confluent Schema Registry. Cribl Stream determines the schema type by inspecting the schema registered in the Confluent Schema Registry for the provided key and value schema IDs.

Enabled: Default is toggled off. When toggled on, displays the following controls:

Schema Registry URL: URL for access to the Confluent Schema Registry (for example, `http://<hostname>:8081`).

Connection timeout (ms): Specifies the maximum time allowed for establishing a successful connection to the schema registry. Defaults to 30000 ms (30 seconds). Valid range is 1000 ms - 60000 ms (or 1 sec to 60 sec).

Request timeout (ms): Maximum time to wait for a successful request from the schema registry. Defaults to 60000 ms (or 1 minute).

Retry limit: Max number of times to retry requests to the schema registry before dropping the outgoing message. Defaults to 1; enter 0 to not retry at all.

Authentication enabled: Toggle on if your schema registry requires authentication.

- **Credentials secret**: Select or **Create** a stored text secret containing the username and password provided by your schema registry administrator.

TLS enabled: When toggled on, displays the following TLS settings for the Schema Registry.

These have the same format as the TLS Settings (Client Side) above.

- **Validate server certs**: Require client to reject any connection that is not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled off.

- **Server name (SNI)**: Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.
- **Minimum TLS version**: Optionally, select the minimum TLS version to use when connecting.
- **Maximum TLS version**: Optionally, select the maximum TLS version to use when connecting.
- **Certificate**: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.
- **CA certificate path**: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.
- **Private key path (mutual auth)**: Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
- **Certificate path (mutual auth)**: Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
- **Passphrase**: Passphrase to use to decrypt private key.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Use these settings to fine-tune Cribl Stream's integration with Kafka topics. If you are unfamiliar with these parameters, contact Cribl Support to understand the implications of changing the defaults.

Heartbeat interval (ms): Expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Value must be lower than `sessionTimeout`, and typically should not exceed 1/3 of the `sessionTimeout` value. Defaults to `3000` ms (3 seconds). For details, see the Kafka documentation.

Session timeout (ms): Timeout used to detect client failures when using Kafka's group management facilities. If the client sends the broker no heartbeats before this timeout expires, the broker will remove this client from the group, and will initiate a rebalance. Value must be between the broker's configured `group.min.session.timeout.ms` and `group.max.session.timeout.ms`. Defaults to `30000` ms (30 seconds). For details, see the Kafka documentation.

Rebalance timeout (ms): Maximum allowed time for each worker to join the group after a rebalance has begun. If the timeout is exceeded, the coordinator broker will remove the worker from the group. Defaults to `60000` ms (1 minute). For details, see the Kafka documentation.

Connection timeout (ms): Maximum time to wait for a successful connection. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour). For details, see the Kafka documentation.

Request timeout (ms): Maximum time to wait for a successful request. Defaults to `60000` ms (1 minute). For details, see the Kafka documentation.

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Source from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Source to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Offset commit interval (ms): How often, in milliseconds, to commit offsets. If both this field and the **Offset commit threshold** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Offset commit threshold: The number of events that will trigger an offset commit. If both this field and the **Offset commit interval** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Byte limit, per partition: The maximum amount of data that the server will return per partition. Must equal or exceed the maximum message size the server allows. (Otherwise, the producer will be unable to send messages larger than the consumer can fetch.) If not specified, defaults to `1048576`.

Byte limit: Maximum amount of bytes to accumulate in the response. The default is `10485760 (10 MB)`.

Error limit, per socket: Maximum number of consecutive request errors that can occur on a single socket connection before the connection is discarded and reestablished. This mitigates issues with idle connections, particularly those used for sending heartbeats, thereby reducing the number of rebalances in a consumer group. Default is `0`, which disables this feature. Accepts values `1-100`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

If you observe an excessive number of group rebalances, and/or you observe consumers not regularly pulling messages, try increasing the values of **Heartbeat interval**, **Session timeout**, and **Rebalance timeout**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Connecting to Kafka

> **Leave the TLS Settings > Server name (SNI) field blank**
>
> In Cribl Stream's Kafka-based Sources and Destinations (including this one), the Kafka library that Cribl Stream uses manages SNI (Server Name Indication) without any input from Cribl Stream. Therefore, you should leave the **TLS Settings** > **Server name (SNI)** field blank.
>
> Setting this field in the Cribl Stream UI can cause traffic to be routed to the wrong brokers, because it interferes with the Kafka library's operation.

Connecting to a Kafka cluster involves using hostnames for two key components: bootstrap servers and brokers.

Brokers are servers that comprise the storage layer in a Kafka cluster. Bootstrap servers handle the initial connection to the Kafka cluster, and then return the list of brokers. A broker list can run into the hundreds. Every Kafka cluster has a `bootstrap.servers` property, defined as either a single `hostname:port` K-V pair, or a list of them. If Cribl Stream tries to connect via one bootstrap server and that fails, Cribl Stream then tries another one on the list.

In the **General Settings** > **Bootstrap servers** list, you can enter either the hostnames of brokers that your Kafka server has been configured to use, or, the hostnames of one or more bootstrap servers. If Kafka returns a list of brokers that's longer than the list you entered, Cribl Stream keeps the full list internally. Cribl Stream neither saves the list nor makes it available in the UI. The connection process simply starts at the beginning whenever the Source or Destination is started.

Here's an overview of the connection process:

1. From the **General Settings** > **Bootstrap servers** list - where each broker is listed as a hostname and port - Cribl Stream takes a hostname and resolves it to an IP address.
2. Cribl Stream makes a connection to that IP address. Notwithstanding the fact that Cribl Stream resolved **one** particular hostname to that IP address, there may be **many** services running at that IP address - each with its own distinct hostname.
3. Cribl Stream establishes TLS security for the connection.

Although SNI is managed by the Kafka library rather than in the Cribl Stream UI, you might want to know how it fits into the connection process. The purpose of the SNI is to specify one hostname - that is, service - among many that might be running on a given IP address within a Kafka cluster. Excluding the other services is one way that TLS makes the connection more secure.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__headers` (when present in the record)
- `__inputId`
- `__key` (when using Schema Registry)
- `__keySchemaIdIn` (when using Schema Registry)
- `__origTime`
- `__partition`
- `__raw`
- `__schemaId` (when using Schema Registry)
- `__topicIn` (indicates the Kafka topic that the event came from; see `__topicOut` in our Kafka Destination documentation)
- `__valueSchemaIdIn` (when using Schema Registry)
- `_raw`
- `_time`

How Cribl Stream Pulls Data

Kafka treats all the Worker Nodes as members of a Consumer Group, and Kafka manages each Node's data load. By default, Workers will poll every 5 seconds. In the case of Leader failure, Worker Nodes will continue to receive data as normal.

How Cribl Stream Handles the `_time` Field

Events that the Kafka Source emits always contain a `_time` field, and sometimes also an `__origTime` field. Here's how Cribl Stream determines what to send:

- If the incoming Kafka message contains no `_time` field, the message's timestamp becomes the value of the emitted event's `_time` field.
- If the incoming Kafka message contains a `_time` field whose value is a timestamp in Unix time, that timestamp becomes the value of the emitted event's `_time` field.
- If the incoming Kafka message contains a `_time` field whose value is **not** a timestamp in Unix time (such as an ISO or UTC timestamp), that becomes the value of the emitted event's `__origTime` field, **and** the message's timestamp becomes the value of the emitted event's `_time` field.

Randomized Partition Assignment

In Cribl Stream 4.8.2 and newer, partition assignment is randomized. The Worker Processes selected to consume data during partition assignment are selected randomly, reducing the possibility of uneven load distribution when there are fewer partitions than Worker Processes. Note that *random* assignment is not the same as *balanced* assignment, and even distribution of work is not guaranteed.

Controlling Rebalancing

When you configure multiple Sources that subscribe to different topics, but all belong to the same consumer group, a state change affecting any Source in this consumer group will affect all the other Sources. Examples of state changes include: deploying new configs, adding or removing Worker Processes, or Worker Processes crashing.

Here's an example - three Sources, three different topics, all in one consumer group:

- `Source_1 - Topic_1 - ConsumerGroup1`
- `Source_2 - Topic_2 - ConsumerGroup1`
- `Source_3 - Topic_3 - ConsumerGroup1`

Imagine that Source 1 undergoes a state change event, such as a Worker Process crash. Source 2 and Source 3 will rebalance - stopping data flow until the rebalance completes.

Shared Worker Group Mitigation

If Sources that share a consumer group all deploy as part of the same Worker Group, changes will have smaller side effects than when Sources are spread across different Worker Groups. (Conversely, imagine a configuration where deploying new configs for Worker Group 1 caused rebalancing of topics in worker Worker Group 2. This spillover would be especially undesirable.)

Bottom Line

Changes to any member of a consumer group affect all other members of that consumer group. To prevent this undesired behavior, make sure to use a unique **Group ID** for each Kafka, Confluent Cloud, and Azure Event

Hubs Source.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

KafkaJSProtocolError: Not authorized to access topics: [Topic authorization failed]

The username does not have read permissions for the specified topic.

8.1.5.2. Confluent Cloud Source

Cribl Stream supports receiving Kafka topics from the Confluent Cloud managed Kafka platform. As of Cribl Stream 3.3, this Source automatically detects compressed data in `Gzip`, `Snappy`, `LZ4`, or `ZSTD` format.

Your Confluent Cloud permissions should include `Consumer Group: Read`.

Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **No**

Confluent Cloud uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must receive events directly from senders. You might need to adjust your firewall rules to allow this traffic.

Ingest Kafka Topics from Confluent Cloud

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Bootstrap servers**: List of Confluent Cloud bootstrap servers to use, for example, `myAccount.confluent.cloud:9092`.
    - **Topics**: Enter the name(s) of topics to subscribe to. Press `Enter/Return` between multiple entries.

    To optimize performance and prevent excessive rebalancing, Cribl recommends subscribing each Confluent Cloud Source to only one topic. To subscribe to multiple topics, consider creating a dedicated Confluent Cloud Source for each one.

    For the same reason, the **Group ID** (below) should be unique for each of your Confluent Cloud, Kafka, and Azure Event Hubs Sources. For details, see Controlling Rebalancing.
3. Next, you can configure the following **Optional Settings**:

- **Group ID**: The name of the consumer group to which this Cribl Stream instance belongs.
- **From beginning**: Whether to start reading from the earliest available data. Relevant only during initial subscription. Defaults to toggled on.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the TLS, Authentication, Schema Registry, Processing, Retries, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Client Side)

Enabled: Defaults to toggled off. When toggled on:

Autofill?: This setting is experimental.

Validate client certificates: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Defaults to toggled on.

Server name (SNI): Leave this field blank. See Connecting to Kafka below.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Authentication

This section details the SASL (Simple Authentication and Security Layer) and OAuth authentication settings for connecting to bootstrap servers. For enhanced security, using TLS is highly recommended.

To begin, toggle **Enabled** on (the default is off). This presents you with options to configure authentication via `SASL` or `OAuth`.

SASL Authentication

Use the **SASL mechanism** dropdown to select your authentication method from the available options: `PLAIN`, `SCRAM-256`, `SCRAM-512`, and `GSSAPI/Kerberos`. The controls displayed will vary based on your selection.

PLAIN, SCRAM-256, or SCRAM-512

For these methods, you can provide credentials either through:

- **Manual**: Displays **API key** and **Secret** fields to enter your Kafka credentials directly.
- **Secret**: This option exposes a **Credentials secret** drop-down in which you can select a stored text secret that references your Kafka credentials. A **Create** link is available to store a new, reusable secret.

GSSAPI/Kerberos

Kerberos authentication is not supported for Cribl-managed Workers in Cribl.Cloud, but it is enabled for customer-managed (hybrid or on-prem) Worker Groups, whether the Leader is based in Cribl.Cloud or on-prem.

If you select Kerberos, you must provide the following:

Keytab location: Enter the location of the keytab file for the authentication principal.

Principal: Enter the authentication principal, for example: `kafka_user@example.com`.

Broker service class: Enter the Kerberos service class for Kafka brokers, for example: `kafka`.

You will also need to set up your environment and configure the Cribl Stream host for use with Kerberos. For details, see Kafka Authentication with Kerberos.

Enable OAuth Authentication

Enabling OAuth will replace any existing authentication settings with the word `placeholder`.

Toggle **Enable OAuth** on to configure this method. To configure, you must provide the following:

- **Token URL**: URL of the token endpoint to use for OAuth authentication.
- **Client ID**: ID to use for OAuth authentication.
- **Client secret (text secret)**: Select or create a stored text secret.
- **Add OAuth Parameters**: Optionally, click **Add parameter** for each additional OAuth paramater. These are parameters sent to the authentication provider in the body of the token request. They are often used

for `scope` (for example, in Amazon Cognito) or `audience` (for example, in `Auth0`).

In the resulting table, enter the **Name** of each field per row. The corresponding Value should be a literal value (such, a string, a number, or a boolean).

- **Add SASL Extension Fields**: Optionally, click **Add Field** for each additional SASL extension fields. These are fields that are set as extensions when authenticating with the Kafka cluster. For example, Confluent Cloud uses fields like `logicalCluster` or `identityPoolId` for granular access control. You do not need to include the `extension_ prefix` when configuring these fields in Cribl.

In the resulting table, enter the **Name** of each field per row. The corresponding Value should be a literal value (such, a string, a number, or a boolean).

Schema Registry

This section governs Kafka Schema Registry Authentication for data encoded with Avro or JSON schemas, with schemas stored in the Confluent Schema Registry. Cribl Stream determines the schema type by inspecting the schema registered in the Confluent Schema Registry for the provided key and value schema IDs.

Enabled: Defaults to toggled off. When toggled on, displays the following controls:

Schema registry URL: URL for access to the Confluent Schema Registry (for example, `http://<hostname>:8081`).

Connection timeout (ms): Specifies the maximum time allowed for establishing a successful connection to the schema registry. Defaults to 30000 ms (30 seconds). Valid range is 1000 ms - 60000 ms (or 1 sec to 60 sec).

Request timeout (ms): Maximum time to wait for a successful request from the schema registry. Defaults to 60000 ms (or 1 minute).

Retry Limit: Max number of times to retry requests to the schema registry before dropping the outgoing message. Defaults to 1; enter 0 to not retry at all.

Authentication enabled: Toggle on if your schema registry requires authentication.

- **Credentials secret**: Select or **Create** a stored text secret containing the username and password provided by your schema registry administrator.

TLS enabled: defaults to toggled off. When toggled on, displays the following TLS settings for the Schema Registry (in the same format as the TLS Settings (Client Side) above):

- **Validate server certs**: Require client to reject any connection that is not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled off.
- **Server name (SNI)**: Leave this field blank. See About Kafka and TLS below.
- **Minimum TLS version**: Optionally, select the minimum TLS version to use when connecting.

- **Maximum TLS version**: Optionally, select the maximum TLS version to use when connecting.
- **Certificate**: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.
- **CA certificate path**: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.
- **Private key path (mutual auth)**: Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
- **Certificate path (mutual auth)**: Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
- **Passphrase**: Passphrase to use to decrypt private key.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Use these settings to fine-tune Cribl Stream's integration with Kafka topics. If you are unfamiliar with these parameters, contact Cribl Support to understand the implications of changing the defaults.

Heartbeat interval (ms): Expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Value must be lower than `sessionTimeout`, and typically should not exceed 1/3 of the `sessionTimeout` value. Defaults to `3000` ms (3 seconds). For details, see the Kafka documentation.

Session timeout (ms): Timeout used to detect client failures when using Kafka's group management facilities. If the client sends the broker no heartbeats before this timeout expires, the broker will remove this client from the group, and will initiate a rebalance. Value must be between the broker's configured `group.min.session.timeout.ms` and `group.max.session.timeout.ms`. Defaults to `30000` ms (30 seconds). For details, see the Kafka documentation.

Rebalance timeout (ms): Maximum allowed time for each worker to join the group after a rebalance has begun. If the timeout is exceeded, the coordinator broker will remove the worker from the group. Defaults to `60000` ms (1 minute). For details, see the Kafka documentation.

Connection timeout (ms): Maximum time to wait for a successful connection. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour). For details, see the Kafka documentation.

Request timeout (ms): Maximum time to wait for a successful request. Defaults to `60000` ms (1 minute). For details, see the Kafka documentation.

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Source from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Source to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Offset commit interval (ms): How often, in milliseconds, to commit offsets. If both this field and the **Offset commit threshold** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Offset commit threshold: The number of events that will trigger an offset commit. If both this field and the **Offset commit interval** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Max bytes per partition: The maximum amount of data that the server will return per partition. Must equal or exceed the maximum message size the server allows. (Otherwise, the producer will be unable to send messages larger than the consumer can fetch.) If not specified, defaults to `1048576`.

Max bytes: Maximum amount of bytes to accumulate in the response. The default is `10485760 (10 MB)`.

Maximum number of errors per socket: Maximum number of consecutive request errors that can occur on a single socket connection before the connection is discarded and reestablished. This mitigates issues with idle connections, particularly those used for sending heartbeats, thereby reducing the number of rebalances in a consumer group. Default is `0`, which disables this feature. Accepts values `1-100`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

If you observe an excessive number of group rebalances, and/or you observe consumers not regularly pulling messages, try increasing the values of **Heartbeat interval**, **Session timeout**, and **Rebalance timeout**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Connecting to Kafka

> **Leave the TLS Settings > Server name (SNI) field blank**
>
> In Cribl Stream's Kafka-based Sources and Destinations (including this one), the Kafka library that Cribl Stream uses manages SNI (Server Name Indication) without any input from Cribl Stream. Therefore, you should leave the **TLS Settings** > **Server name (SNI)** field blank.
>
> Setting this field in the Cribl Stream UI can cause traffic to be routed to the wrong brokers, because it interferes with the Kafka library's operation. This is especially important for Confluent Cloud Dedicated clusters, which rely on SNI - as managed by the Kafka library - for routing.

Connecting to a Kafka cluster involves using hostnames for two key components: bootstrap servers and brokers.

Brokers are servers that comprise the storage layer in a Kafka cluster. Bootstrap servers handle the initial connection to the Kafka cluster, and then return the list of brokers. A broker list can run into the hundreds. Every Kafka cluster has a `bootstrap.servers` property, defined as either a single `hostname:port` K-V pair, or a list of them. If Cribl Stream tries to connect via one bootstrap server and that fails, Cribl Stream then tries another one on the list.

In the **General Settings** > **Bootstrap servers** list, you can enter either the hostnames of brokers that your Kafka server has been configured to use, or, the hostnames of one or more bootstrap servers. If Kafka returns a list of brokers that's longer than the list you entered, Cribl Stream keeps the full list internally. Cribl Stream neither saves the list nor makes it available in the UI. The connection process simply starts at the beginning whenever the Source or Destination is started.

Here's an overview of the connection process:

1. From the **General Settings** > **Bootstrap servers** list - where each broker is listed as a hostname and port - Cribl Stream takes a hostname and resolves it to an IP address.
2. Cribl Stream makes a connection to that IP address. Notwithstanding the fact that Cribl Stream resolved **one** particular hostname to that IP address, there may be **many** services running at that IP address - each with its own distinct hostname.
3. Cribl Stream establishes TLS security for the connection.

Although SNI is managed by the Kafka library rather than in the Cribl Stream UI, you might want to know how it fits into the connection process. The purpose of the SNI is to specify one hostname - that is, service - among many that might be running on a given IP address within a Kafka cluster. Excluding the other services is one way that TLS makes the connection more secure.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`
- `__topicIn` (indicates the Confluent Cloud topic that the event came from; see `__topicOut` in our [Confluent Cloud Destination](destinations-Confluent Cloud) documentation)
- `__partition`
- `__schemaId` (when using Schema Registry)

How Cribl Stream Pulls Data

Confluent Cloud treats all the Worker Nodes as members of a Consumer Group, and Confluent Cloud manages each Node's data load. By default, Workers will poll every 5 seconds. In the case of Leader failure, Worker Nodes will continue to receive data as normal.

How Cribl Stream Handles the `_time` Field

Events that the Kafka Source emits always contain a `_time` field, and sometimes also an `__origTime` field. Here's how Cribl Stream determines what to send:

- If the incoming Kafka message contains no `_time` field, the message's timestamp becomes the value of the emitted event's `_time` field.
- If the incoming Kafka message contains a `_time` field whose value is a timestamp in Unix time, that timestamp becomes the value of the emitted event's `_time` field.
- If the incoming Kafka message contains a `_time` field whose value is **not** a timestamp in Unix time (for example, an ISO or UTC timestamp), that becomes the value of the emitted event's `__origTime` field, **and** the message's timestamp becomes the value of the emitted event's `_time` field.

Randomized Partition Assignment

In Cribl Stream 4.8.2 and newer, partition assignment is randomized. The Worker Processes selected to consume data during partition assignment are selected randomly, reducing the possibility of uneven load

distribution when there are fewer partitions than Worker Processes. Note that *random* assignment is not the same as *balanced* assignment, and even distribution of work is not guaranteed.

Controlling Rebalancing

When you configure multiple Sources that subscribe to different topics, but all belong to the same consumer group, a state change affecting any Source in this consumer group will affect all the other Sources. Examples of state changes include: deploying new configs, adding or removing Worker Processes, or Worker Processes crashing.

Here's an example - three Sources, three different topics, all in one consumer group:

- `Source_1 - Topic_1 - ConsumerGroup1`
- `Source_2 - Topic_2 - ConsumerGroup1`
- `Source_3 - Topic_3 - ConsumerGroup1`

Imagine that Source 1 undergoes a state change event, such as a Worker Process crash. Source 2 and Source 3 will rebalance - stopping data flow until the rebalance completes.

Shared Worker Group Mitigation

If Sources that share a consumer group all deploy as part of the same Worker Group, changes will have smaller side effects than when Sources are spread across different Worker Groups. (Conversely, imagine a configuration where deploying new configs for Worker Group 1 caused rebalancing of topics in worker Worker Group 2. This spillover would be especially undesirable.)

Bottom Line

Changes to any member of a consumer group affect all other members of that consumer group. To prevent this undesired behavior, make sure to use a unique **Group ID** for each Kafka, Confluent Cloud, and Azure Event Hubs Source.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

KafkaJSProtocolError: Not authorized to access topics: [Topic authorization failed]

The username does not have read permissions for the specified topic.

8.1.5.3. Amazon MSK Source

Cribl Stream supports receiving data records from an Amazon Managed Streaming for Apache Kafka (MSK) cluster. This Source automatically detects compressed data in `Gzip`, `Snappy`, `LZ4`, or `ZSTD` format.

Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **No**

Kafka uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must receive events directly from senders. You might need to adjust your firewall rules to allow this traffic.

Configure Cribl Stream to Receive Data from Kafka Topics

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Bootstrap servers**: List of Kafka bootstrap servers to connect to, for example, `kafkaBrokerHost:9092`. Specify hostname and port - for example, `mykafkabroker:9092` - or just hostname, in which case Stream will assign port 9092.
    - **Topics**: Enter the name(s) of topics to subscribe to. Press `Enter`/`Return` between multiple entries.
    - **Region**: Select the name of the AWS Region where your Amazon MSK cluster is located.

To optimize performance and prevent excessive rebalancing, Cribl recommends subscribing each Kafka Source to only one topic. To subscribe to multiple topics, consider creating a dedicated Kafka Source for each one.

For the same reason, the **Group ID** (below) should be unique for each of your Kafka, Confluent Cloud, and Azure Event Hubs Sources. For details, see Controlling Rebalancing.

3. Next, you can configure the following **Optional Settings**:
    - **Group ID**: The name of the consumer group to which this Cribl Stream instance belongs.
    - **From beginning**: Toggle this off if you want Cribl Stream, upon subscribing to a topic for the first time, to read new messages only. Otherwise Cribl Stream will read all messages available on the server, starting from the oldest.
    - **Tags**: Optionally, add tags that you can use for filtering and grouping in Cribl Stream's UI. Use a tab or hard return between (arbitrary) tag names. These tags aren't added to processed events.
4. Optionally, you can adjust the TLS, Authentication, Assume Role, Processing, Retries, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Client Side)

For Amazon MSK Sources and Destinations:

- IAM is the only type of authentication that Cribl Stream supports.
- Because IAM auth requires TLS, TLS is automatically enabled.

Validate server certs: Reject certificates that are **not** authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Defaults to toggled on.

Server name (SNI): Leave this field blank. See Connecting to Kafka below.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Authentication

Use the **Authentication method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, such as those running in a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for MSK: Toggle on to use Assume Role credentials to access MSK.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role. This is required only when assuming a role that requires this ID in order to delegate third-party access. For details, see AWS' documentation.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Use these settings to fine-tune Cribl Stream's integration with Kafka topics. If you are unfamiliar with these parameters, contact Cribl Support to understand the implications of changing the defaults.

Heartbeat interval (ms): Expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Value must be lower than `sessionTimeout`, and typically should not exceed 1/3 of the `sessionTimeout` value. Defaults to `3000` ms (3 seconds). For details, see the Kafka documentation.

Session timeout (ms): Timeout used to detect client failures when using Kafka's group management facilities. If the client sends the broker no heartbeats before this timeout expires, the broker will remove this client from the group, and will initiate a rebalance. Value must be between the broker's configured `group.min.session.timeout.ms` and `group.max.session.timeout.ms`. Defaults to `30000` ms (30 seconds). For details, see the Kafka documentation.

Rebalance timeout (ms): Maximum allowed time for each Worker to join the Group after a rebalance has begun. If the timeout is exceeded, the coordinator broker will remove the Worker from the Group. Defaults to `60000` ms (1 minute). For details, see the Kafka documentation.

Connection timeout (ms): Maximum time to wait for a connection to complete successfully. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour). For details, see the Kafka documentation.

Request timeout (ms): Maximum time to wait for Kafka to respond to a request. Defaults to `60000` ms (1 minute). For details, see the Kafka documentation.

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Source from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Source to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Offset commit interval (ms): How often, in milliseconds, to commit offsets. If both this field and the **Offset commit threshold** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Offset commit threshold: How many events are needed to trigger an offset commit. If both this field and the **Offset commit interval** are empty, Cribl Stream will commit offsets after each batch. If both fields are set, Cribl Stream will commit offsets when either condition is met.

Max bytes per partition: Maximum amount of data that Kafka will return per partition, per fetch request. Must equal or exceed the maximum message size (maxBytesPerPartition) that Kafka is configured to allow. Otherwise, Cribl Stream can get stuck trying to retrieve messages. Defaults to `1048576 (1 MB)`.

Max bytes: Maximum number of bytes that Kafka will return per fetch request. Defaults to `10485760 (10 MB)`.

Maximum number of errors per socket: Maximum number of consecutive request errors that can occur on a single socket connection before the connection is discarded and reestablished. This mitigates issues with idle connections, particularly those used for sending heartbeats, thereby reducing the number of rebalances in a consumer group. Default is `0`, which disables this feature. Accepts values `1-100`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

If you observe an excessive number of group rebalances, and/or you observe consumers not regularly pulling messages, try increasing the values of **Heartbeat interval**, **Session timeout**, and **Rebalance timeout**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Connecting to Kafka

Leave the TLS Settings > Server name (SNI) field blank

In Cribl Stream's Kafka-based Sources and Destinations (including this one), the Kafka library that Cribl Stream uses manages SNI (Server Name Indication) without any input from Cribl Stream. Therefore, you should leave the **TLS Settings** > **Server name (SNI)** field blank.

Setting this field in the Cribl Stream UI can cause traffic to be routed to the wrong brokers, because it interferes with the Kafka library's operation.

Connecting to a Kafka cluster involves using hostnames for two key components: bootstrap servers and brokers.

Brokers are servers that comprise the storage layer in a Kafka cluster. Bootstrap servers handle the initial connection to the Kafka cluster, and then return the list of brokers. A broker list can run into the hundreds. Every Kafka cluster has a `bootstrap.servers` property, defined as either a single `hostname:port` K-V pair, or a list of them. If Cribl Stream tries to connect via one bootstrap server and that fails, Cribl Stream then tries another one on the list.

In the **General Settings** > **Bootstrap servers** list, you can enter either the hostnames of brokers that your Kafka server has been configured to use, or, the hostnames of one or more bootstrap servers. If Kafka returns a list of brokers that's longer than the list you entered, Cribl Stream keeps the full list internally. Cribl Stream neither saves the list nor makes it available in the UI. The connection process simply starts at the beginning whenever the Source or Destination is started.

Here's an overview of the connection process:

1. From the **General Settings** > **Bootstrap servers** list - where each broker is listed as a hostname and port - Cribl Stream takes a hostname and resolves it to an IP address.

2. Cribl Stream makes a connection to that IP address. Notwithstanding the fact that Cribl Stream resolved **one** particular hostname to that IP address, there may be **many** services running at that IP address - each with its own distinct hostname.
3. Cribl Stream establishes TLS security for the connection.

Although SNI is managed by the Kafka library rather than in the Cribl Stream UI, you might want to know how it fits into the connection process. The purpose of the SNI is to specify one hostname - that is, service - among many that might be running on a given IP address within a Kafka cluster. Excluding the other services is one way that TLS makes the connection more secure.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`
- `__topicIn` (indicates the Kafka topic that the event came from; see `__topicOut` in our Kafka Destination documentation)
- `__partition`
- `__schemaId` (when using Schema Registry)
- `__key` (when using Schema Registry)
- `__headers` (when using Schema Registry)
- `__keySchemaIdIn` (when using Schema Registry)
- `__valueSchemaIdIn` (when using Schema Registry)

How Cribl Stream Pulls Data

Kafka treats all the Worker Nodes as members of a Consumer Group, and Kafka manages each Node's data load. By default, Workers will poll every 5 seconds. In the case of Leader failure, Worker Nodes will continue to receive data as normal.

How Cribl Stream Handles the `_time` Field

Events that the Kafka Source emits always contain a `_time` field, and sometimes also an `__origTime` field. Here's how Cribl Stream determines what to send:

- If the incoming Kafka message contains no `_time` field, the message's timestamp becomes the value of the emitted event's `_time` field.

- If the incoming Kafka message contains a `_time` field whose value is a timestamp in Unix time, that timestamp becomes the value of the emitted event's `_time` field.
- If the incoming Kafka message contains a `_time` field whose value is **not** a timestamp in Unix time (for example, an ISO or UTC timestamp), that becomes the value of the emitted event's `__origTime` field, **and** the message's timestamp becomes the value of the emitted event's `_time` field.

Randomized Partition Assignment

In Cribl Stream 4.8.2 and newer, partition assignment is randomized. The Worker Processes selected to consume data during partition assignment are selected randomly, reducing the possibility of uneven load distribution when there are fewer partitions than Worker Processes. Note that *random* assignment is not the same as *balanced* assignment, and even distribution of work is not guaranteed.

Controlling Rebalancing

When you configure multiple Sources that subscribe to different topics but all belong to the same consumer group, a state change affecting any Source in this consumer group will affect all the other Sources. Examples of state changes include: deploying new configs, adding or removing Worker Processes, and Worker Processes crashing.

Here's an example - three Sources, three different topics, all in one consumer group:

- `Source_1 - Topic_1 - ConsumerGroup1`
- `Source_2 - Topic_2 - ConsumerGroup1`
- `Source_3 - Topic_3 - ConsumerGroup1`

Imagine that Source 1 undergoes a state change event, such as a Worker Process crash. Source 2 and Source 3 will rebalance - stopping data flow until the rebalance completes.

Shared Worker Group Mitigation

If Sources that share a consumer group all deploy as part of the same Worker Group, changes will have smaller side effects than when Sources are spread across different Worker Groups. (Conversely, imagine a configuration where deploying new configs for Worker Group 1 caused rebalancing of topics in Worker Worker Group 2. This spillover would be especially undesirable.)

Bottom Line

Changes to any member of a consumer group affect all other members of that consumer group. To prevent this undesired behavior, make sure to use a unique **Group ID** for each Kafka, Confluent Cloud, Amazon MSK, and Azure Event Hubs Source.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

KafkaJSProtocolError: Not authorized to access topics: [Topic authorization failed]

8.1.6. Office 365

You can use the following Sources to receive data from Office 365 services.

Source	Description
Office 365 Activity	Receive data from the Office 365 Management Activity API
Office 365 Message Trace	Receive Office 365 Message Trace data
Office 365 Services	Receive data from the Microsoft Graph service communications API

8.1.6.1. Office 365 Activity Source

Cribl Stream supports receiving data from the Office 365 Management Activity API. This facilitates analyzing actions and events on Microsoft Entra ID, Exchange, and SharePoint, along with global auditing and Data Loss Prevention data.

Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **YES**

TLS is enabled via the HTTPS protocol on this Source's underlying REST API.

Prerequisites

You need to have a registered application with the Microsoft identity platform to provide Cribl Stream access to make requests to the Microsoft Graph API.

From the registered application, you'll need its **Application (client) ID**, **Directory (tenant) ID**, and credentials to configure in the Cribl Stream Source.

For a complete, step-by-step walkthrough, see our Knowledge Base article Office 365 App Registration for Cribl Sources.

Microsoft Permissions

The application representing your Cribl Stream instance needs the following application permissions to pull data. The permission **Type** for both must be `Application` - `Delegated` is not sufficient:

- `ActivityFeed.Read` - Required for all Content Types except `DLP.All`.
- `ActivityFeed.ReadDlp` - Required for the `DLP.All` Content Type.

Office 365 Subscriptions

Cribl Stream does not support starting/stopping Office 365 subscriptions. You can start subscriptions either via another Office 365 API client, or simply via `curl` commands. We document the `curl` command method below in Starting Content Subscriptions.

Configure Cribl Stream to Receive Data from the Activity API

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:

- To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
- To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.

2. In the **New Source** modal, configure the following under **General Settings**:
   - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
   - **Description**: Optionally, enter a description.
   - **Tenant ID**: Enter the Office 365 Azure **Directory (tenant) ID**.
   - **App ID**: Enter the Office 365 Azure **Application (client) ID**.
   - **Subscription Plan**: See Subscription Plan below.
3. Under **Authentication**, select the **Authentication method** from the dropdown:
   - **Manual**: This default option provides a **Client secret** field, where you directly enter the required Office 365 Azure client secret.
   - **Secret**: This option instead exposes a **Client secret (text secret)** drop-down, from which you select a stored text secret to authenticate with. Click **Create** to configure a new secret.
4. Next, you can configure the following **Optional Settings**:
   - **Publisher identifier**: Use in API requests as described here. If not defined, defaults to Microsoft Office 365 tenant ID.
   - **Content Types**: See the Content Types section below.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Processing, Retries, and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Subscription Plan

Select the Office 365 subscription plan for your Organization. Options include:

- **Office 365 Enterprise**
- **Office 365 GCC**
- **Office 365 GCC High**
- **Office 365 DoD**

Content Types

Here, you can configure polling independently for the following types of audit data from the Office 365 Management Activity API:

- **Active Directory**
- **Exchange**
- **SharePoint**
- **General**: All workloads not included in the above content types
- **DLP.All**: Data Loss Prevention events only, for all workloads

For each of these content types, the **Content Types** table provides the following controls:

Interval Description: This column is informational only.

Interval: Optionally, override the default polling interval. See About Polling Intervals below.

Log Level: Set the verbosity level to one of `debug`, `info` (the default), `warn`, or `error`.

Enabled: Toggle on for each service that you want to poll.

About Polling Intervals

To poll the Office 365 Management Activity API, Cribl Stream uses the **Interval** field's value to establish the search date range and the cron schedule (for example: `*/${interval} * * * *`).

Therefore, intervals set in minutes must divide evenly into 60 minutes to create a predictable schedule. Dividing 60 by intervals like `1`, `2`, `3`, `4`, `5`, `6`, `10`, `12`, `15`, `20`, or `60` itself yields an integer, so you can enter any of these values.

Cribl Stream will reject intervals like `23`, `42`, or `45`, or `75` - which would yield non-integer results, meaning unpredictable schedules.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

Retry type: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.

Initial retry interval (ms): Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.

Retry limit: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.

Backoff multiplier: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

Retry HTTP codes: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.

Honor Retry-After header: When toggled on (the default) and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds.

- Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
- When toggled off, Cribl Stream ignores all `retry-after` headers.

Retry connection timeout: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.

Retry connection reset: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.

Advanced Settings

Ingestion lag (minutes): How far back into the past the Office 365 Activity API should look for events to retrieve. The default value of `0` means don't look back, and can often lead to missed events due to Microsoft's delays. See Ingestion Delays for information about how to determine the appropriate setting for this field.

Request timeout (secs): The maximum time period for an HTTP request to complete before Cribl Stream treats it as timed out. Defaults to `300` (5 minutes). Enter `0` to disable timeout metering.

Job timeout: Maximum time the job is allowed to run (for example, `30`, `45s`, or `15m`). Units are seconds, if not specified. Enter `0` for unlimited time. Defaults to `0`.

Time to live: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__collectible`
- `__final`
- `__inputId`
- `__isBroken`
- `__source`

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Starting Content Subscriptions

Content subscriptions (a different concept from the O365 subscription plans) are required in order for Cribl Stream to be able to begin retrieving O365 data. There is a separate subscription required for each Content Type. If you are using an existing Azure-registered application ID that already has subscriptions started, then you can ignore this section. But if you are:

- Using a newly registered application ID, and therefore never had any subscriptions started, or
- Reusing an application ID that had subscriptions started, but are currently stopped

...then you will need to use this procedure to manually start the necessary subscriptions. Follow either of the two methods below, using (respectively) PowerShell or `curl`.

Using PowerShell

This sample PowerShell script will enable all subscriptions for you. Update the appropriate variables as required:

```
# Create app of type Web app / API in Microsoft Entra ID, generate a Client Secret, a
# Get the tenant GUID from Properties | Directory ID under the Microsoft Entra ID sec
$AppID = "<APP_ID>"
$ClientSecret = "<CLIENT_SECRET>"
$TenantID = "<TENANT_ID>"
$loginURL = "https://login.microsoftonline.com/"

# For $resource, use one of these endpoint values based on your subscription plan:
# * Enterprise - manage.office.com
# * GCC - manage-gcc.office.com
# * GCC High - manage.office365.us
# * DoD - manage.protection.apps.mil
$resource = "https://manage.office.com"

$body = @{grant_type="client_credentials";resource=$resource;client_id=$AppID;client_
$oauth = Invoke-RestMethod -Method Post -Uri $loginURL/$TenantID/oauth2/token?api-ver
$headerParams = @{'Authorization'="$($oauth.token_type) $($oauth.access_token)"}

Invoke-WebRequest -Headers $headerParams -Uri "$resource/api/v1.0/$TenantID/activity/

Invoke-WebRequest -Method Post -Headers $headerParams -Uri "$resource/api/v1.0/$Tenar
Invoke-WebRequest -Method Post -Headers $headerParams -Uri "$resource/api/v1.0/$Tenar
Invoke-WebRequest -Method Post -Headers $headerParams -Uri "$resource/api/v1.0/$Tenar
Invoke-WebRequest -Method Post -Headers $headerParams -Uri "$resource/api/v1.0/$Tenar
Invoke-WebRequest -Method Post -Headers $headerParams -Uri "$resource/api/v1.0/$Tenar
```

Copy

Using curl

This process requires two `curl` commands. You need the same information (client secret, application ID, and tenant ID) that you use to set up this Source in Cribl Stream to run these two commands. Replace those three variables as appropriate in the commands below.

The first command obtains an auth token:

```
curl -X POST "https://login.windows.net/<tenantId>/oauth2/token" \
-d "client_secret=<client secret>&resource=https://manage.office.com&client_id=<app i
```

Copy

The second command uses the auth token from the response to the first command to actually start the subscription:

```
curl -X POST "https://manage.office.com/api/v1.0/<tenantId>/activity/feed/subscriptic
-H "Authorization: Bearer <token>" \
-d ""
```

Copy

Here is an example of Command #1 and the expected response:

Command #1 Example

```
curl -X POST "https://login.windows.net/12345678-aaaa-4233-cccc-160c6c30154a/oauth2/t
-d "client_secret=abcdefghijklmnopqrstuvwxyz12345678&resource=https://manage.office.c
```

Copy

Command #1 Example Response

```
{"token_type":"Bearer","expires_in":"3599","ext_expires_in":"3599","expires_on":"162
```

Copy

Here is an example of Command #2 and the expected response:

Command #2 Example

```
curl -X POST "https://manage.office.com/api/v1.0/12345678-aaaa-4233-cccc-160c6c30154a
-H "Authorization: Bearer eyJ0...longJWT...MRDvw" \
-d ""
```

Copy

Command #2 Example Response

```
{"contentType":"Audit.AzureActiveDirectory","status":"enabled","webhook":null}
```

Copy

Note there is no output when executing this second command with a `stop` operation.

You'll need to execute the second command for each Content Type whose logs you wish to collect. Use the exact strings below to specify Content Types in that command:

- `Audit.AzureActiveDirectory`
- `Audit.Exchange`
- `Audit.SharePoint`
- `Audit.General`
- `DLP.All`

How Cribl Stream Pulls Data

The Office 365 Activity Source retrieves data using Cribl Stream scheduled Collection jobs, which include Discover and Collection phases. The Discover phase task returns the URL of the content to collect.

In the Source's **General Settings** > **Content Types** > **Interval** column, you configure the polling schedule for each Content Type independently. Each Content Type that you enable gets its own separate scheduled job.

The job scheduler spreads the Collection tasks across all available Workers. The collected content is paginated, so the collection phase might include multiple calls to fetch data.

Viewing Scheduled Jobs

Once you've configured and saved the Source, you can view the Collection jobs' results by reopening the Source's config modal and clicking its **Job Inspector** tab.

Each content type that you enabled gets its own separate scheduled job.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring** > **System** > **Job Inspector** > **Currently Scheduled** tab.

Mitigating Stuck-Job Problems

Occasionally, a scheduled job fails, but continues running for hours or even days, until someone intervenes and cancels it. If left alone, such a "stuck", "orphaned," or "zombie" job will never complete. This can cause missing

events in downstream receivers, along with `HTTP timeout` or similar errors in Cribl Stream's logs.

To keep stuck jobs from running excessively long:

- First, try setting **Advanced** > **Timeout (secs)** to a duration shorter than the default of `300` seconds (5 minutes).
- If adjusting **Timeout (secs)** does not fix the problem, try the global setting **Job Timeout** - whose default of `0` allows a job to run indefinitely - to a desired maximum duration. You'll find **Job Timeout** among the task manifest and buffering limits.

Using these settings in tandem works like this:

- **Timeout (secs)** limits the time that Cribl Stream will wait for an HTTP request to complete.
- Then, if a job gets stuck and keeps running beyond that limit, **Job Timeout** can catch and terminate the job, because it monitors the overall time the job has been running.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Ingestion Delays

Office 365 activity events are not delivered in real time, and can be delayed by tens of minutes to several hours before they become available to be collected. In some cases, delays of 24 hours or more have been observed.

These delays in delivery can cause recent events to appear missing in Cribl Stream, and create event-count discrepancies compared to other tools (such as Splunk or Microsoft-native views).

To compensate for Microsoft's delivery delay, Cribl Stream uses the **Ingestion lag (minutes)** setting to configure how far back into the past the Office 365 Activity API should look for events to retrieve. The maximum value for this setting is `7200` minutes (`120` hours).

When the ingestion lag is longer than the delivery delay, events still fall within the time window and can be collected.

To recover such events, you must run a separate backfill process that specifically queries the affected historical time range.

Choose an Ingestion Lag Value

You can configure **Ingestion lag (minutes)** under **Advanced Settings**.

The default value of `0` means don't look back. This assumes events are available as soon as they occur, and in real environments often leads to missed events due to Microsoft's delays.

Use the following guidelines when setting **Ingestion lag (minutes)**:

- **Start with a conservative buffer.** Many environments see delays between 60–90 minutes or longer, so values in the 90–360 minute range are common starting points when it is important to avoid data loss.
- **Tune based on your observed behavior.** If you still see gaps in events when comparing Cribl to Microsoft's own views, gradually increase the ingestion lag and test again over longer time windows (for example several hours or a full day) until gaps in events disappear.

Note that while larger ingestion lag values provide better protection against delayed events, they also increase the amount of data the API returns. This increases the chance of overlapping or duplicate events if you run multiple jobs.

See Also

The Community-contributed Microsoft Office Activity & Azure Logs Pack provides sample Pipelines to parse and streamline ingested Office 365 and EntraID/Azure AD data. You can import these and adapt them to your own needs.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

Dropping request because token invalid",“authToken”: “Bas…Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.6.2. Office 365 Message Trace Source

Microsoft has announced deprecation of the legacy Message Trace API and Reporting Web Service endpoint. The new Microsoft Graph Source is available, and legacy Message Trace support is scheduled for deprecation on April 8, 2026. Plan to migrate to the Microsoft Graph Source before that date to avoid data loss.

Cribl Stream supports receiving Office 365 Message Trace data. This mail-flow metadata can be used to detect and report on malicious activity including bulk emails, spoofed-domain emails, and data exfiltration.

Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **YES**

TLS is enabled via the HTTPS protocol on this Source's underlying REST API.

Office 365 Setup

At a minimum, your Office 365 service account should include a role with `Message Tracking` and `View-Only Recipients` permissions, assigned to the Office 365 user that will integrate with Cribl Stream. Assign these permissions in the Exchange admin center (https://admin.exchange.microsoft.com).

For a complete, step-by-step walkthrough, see our Knowledge Base article Office 365 App Registration for Cribl Sources.

Modern Authentication (OAuth 2.0) Setup

If you plan to use OAuth, OAuth Secret, or OAuth Certificate authentication, you must configure an Application and its associated Service Principal within Microsoft Entra ID as part of your Office 365 setup.

1. In the Microsoft Azure portal, create a Microsoft Entra ID App Registration. This will automatically create a corresponding Service Principal in your tenant. For details, see the Microsoft Azure documentation.
2. Grant your Application the API permissions it needs to access Office 365 Message Trace data.
    - On your newly created App Registration, navigate to **API permissions** and select **Add a permission**.
    - On the **Microsoft APIs** tab, select **Office 365 Exchange Online**. If it's not immediately visible, search for `Office 365 Exchange Online`.
    - Under **Application permissions**, add the `ReportingWebService.Read.All` permission and consent to the changes.

For detailed steps on adding API permissions, see Microsoft's documentation on how to add permissions to access web APIs.

3. Assign the Service Principal (your registered Application's identity in the tenant) at least one Azure AD role that will enable it to access the Reporting Web Service:
    - Global Reader
    - Global Administrator
    - Exchange Administrator

    For detailed steps on assigning Azure AD roles to a Service Principal (Application), see Microsoft's documentation on how to assign Microsoft Entra roles. When assigning the role, you will search for your Application name (the Service Principal) to select it as a member.

If you encounter an error with `statusCode: 401` and details showing `host:"reports.office365.com"` and `path:"/ecp/reportingwebservice/reporting.svc/MessageTrace"`, this indicates the App Registration (Service Principal) is not correctly authorized to access the Office 365 Message Trace service.

Configure Cribl Stream to Receive Office 365 Message Trace Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Report URL**: Enter the URL to use when retrieving report data. Defaults to: `https://reports.office365.com/ecp/reportingwebservice/reporting.svc/MessageTrac`
    - **Poll interval**: How often (in minutes) to run the report. Must divide evenly into 60 minutes to create a predictable schedule, or Save will fail. See About Polling Intervals below.

3. Under **Authentication**, select the **Authentication method** from the dropdown:
    - In the **Authentication** section, use the buttons to select an **Authentication method**: Basic, Basic (credentials secret), OAuth, OAuth (text secret), or OAuth Certificate. The default is **OAuth**. All OAuth options rely on the OAuth 2.0 protocol.
4. Next, you can configure the following **Optional Settings**:
    - Use the **Date range start** and **Date range end** settings to specify the time window for data collection. These settings help compensate for delays and gaps in Message Trace data. Use relative time formats for ongoing data collection and exact epoch values to backfill specific time gaps.
    - **Date range start**: Backward offset for the head of the search date range. Supports relative time format like `-3h@h` or exact epoch values like `1720536960`.
    - **Date range end**: Backward offset for the tail of the search date range. Supports relative time format like `-2h@h` or exact epoch values like `1720533360`.
    - **Log level**: For data collection's runtime log, set the verbosity level to one of `debug`, `info`, `warn`, or `error`. (If not selected, defaults to `info`.)
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Processing, Retries, and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Basic Authentication

Selecting **Basic** exposes **Username** and **Password** fields, where you directly enter the HTTP Basic credentials to use on Message Trace API calls.

Basic Secret Authentication

Selecting **Basic (credentials secret)** exposes a **Credentials secret** drop-down, where you select an existing stored secret that references your credentials on the Message Trace API. You can use the adjacent **Create** button to store a new, reusable secret.

OAuth Authentication

The default **OAuth** authentication method exposes the following fields, all required:

- **Client secret**: Directly enter the `client_secret` to pass in the OAuth request parameter.
- **Tenant identifier**: Directory ID (tenant identifier) in Microsoft Entra ID.
- **Client ID**: The `client_id` to pass in the OAuth request parameter.
- **Resource**: Resource parameter to pass in the OAuth request parameter. Defaults to: `https://outlook.office365.com`.

OAuth Secret Authentication

Selecting **OAuth (text secret)** exposes three of the same controls as the default OAuth method, but - as you'd expect - you instead enter the **Client secret** by reference:

- **Client secret**: Use the drop-down to select an existing stored `client_secret` to pass in the OAuth request parameter. You can use the adjacent **Create** button to store a new, reusable secret.
- **Tenant identifier**: Directory ID (tenant identifier) in Microsoft Entra ID.
- **Client ID**: The `client_id` to pass in the OAuth request parameter.
- **Resource**: Resource parameter to pass in the OAuth request parameter. Defaults to: `https://outlook.office365.com`.

OAuth Certificate Authentication

Selecting **OAuth (certificate)** exposes two of the same controls as the above OAuth methods (**Tenant identifier** and **Client ID**). But this authentication method - available in Cribl Stream 4.1.3 and newer - foregrounds controls to define the certificate you're using to authenticate. Treat all fields here as required, except for the optional **Passphrase**:

- **Certificate name**: Use the drop-down to select an existing certificate to pass in the OAuth request parameter. You can use the adjacent **Create** button to store a new, reusable cert.
- **Private key path**: Path to the private key to use. The key should be in PEM format. Can reference `$ENV_VARS`.
- **Passphrase**: If your private key requires a passphrase to decrypt it, enter that passphrase here.
- **Certificate path**: Path to the certificate to use. The cert should be in PEM format. Can reference `$ENV_VARS`.
- **Tenant identifier**: Directory ID (tenant identifier) in Microsoft Entra ID.
- **Client ID**: The `client_id` to pass in the OAuth request parameter.
- **Resource**: Resource parameter to pass in the OAuth request parameter. Defaults to: `https://outlook.office365.com`.
- **Subscription Plan**: Select the Office 365 subscription plan for your organization. Options include:
  - **Office 365 Enterprise**
  - **Office 365 GCC**
  - **Office 365 GCC High**
  - **Office 365 DoD**

About Polling Intervals

To poll the Office 365 Message Trace API, Cribl Stream uses the **Poll interval** field's value to establish the cron schedule. For example, `*/${interval} * * * *`.

Because the interval is set in minutes, it must divide evenly into 60 minutes to create a predictable schedule. Dividing 60 by intervals like `1`, `2`, `3`, `4`, `5`, `6`, `10`, `12`, `15`, `20`, or `60` itself yields an integer, so you can enter any of these values.

Cribl Stream will reject intervals like `23`, `42`, or `45`, or `75` - which would yield non-integer results, meaning unpredictable schedules.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

Retry type: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.

Initial retry interval (ms): Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.

Retry limit: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.

Backoff multiplier: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

Retry HTTP codes: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.

Honor Retry-After header: When toggled on (the default) and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds.

- Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
- When toggled off, Cribl Stream ignores all `retry-after` headers.

Retry connection timeout: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.

Retry connection reset: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.

Advanced Settings

Request timeout (secs): Maximum time to wait for an individual Message Trace API request to complete. Defaults to `300` seconds (5 minutes). Enter `0` to disable metering, allowing unlimited response time. Because there is a single request to the Message Trace API per page of data, this timeout is applied at the page (request) level.

Reschedule tasks: Whether to automatically reschedule tasks that failed with non-fatal errors. Defaults to toggled on; does not apply to fatal errors.

Task reschedule limit: Maximum number of times a task can be rescheduled. Defaults to `1`.

Job timeout: Maximum time a job is allowed to run. Defaults to `0`, for unlimited time. Units are seconds if not specified. Sample entries: `30`, `45s`, `15m`.

Time to live: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

Disable time filter: Disables Collector event time filtering when a date range is specified in General Settings. Toggle off to allow filtering.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__inputId`
- `__isBroken`
- `__source`

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

How Cribl Stream Pulls Data

The Office 365 Message Trace Source uses a scheduled REST Collector. It runs one collection task every **Poll interval**, and a single Worker will process the collection. The data is paginated, so the Worker might make multiple calls to fetch the data.

Viewing Scheduled Jobs

This Source executes Cribl Stream's scheduled collection jobs. Once you've configured and saved the Source, you can view those jobs' results by reopening the Source's config modal and clicking its **Job Inspector** tab.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring** > **System** > **Job Inspector** > **Currently Scheduled** tab.

Mitigating Stuck-Job Problems

Occasionally, a scheduled job fails, but continues running for hours or even days, until someone intervenes and cancels it. If left alone, such a "stuck", "orphaned," or "zombie" job will never complete. This can cause missing events in downstream receivers, along with `HTTP timeout` or similar errors in Cribl Stream's logs.

To keep stuck jobs from running excessively long:

- First, try setting **Advanced** > **Timeout (secs)** to a duration shorter than the default of `600` seconds (10 minutes).
- If adjusting **Timeout (secs)** does not fix the problem, try setting **Advanced** > **Job Timeout** - whose default of `0` allows a job to run indefinitely - to a desired maximum duration.

Using these settings in tandem works like this:

- **Timeout (secs)** limits the time that Cribl Stream will wait for an HTTP request to complete.
- Then, if a job gets stuck and keeps running beyond that limit, **Job Timeout** can catch and terminate the job, because it monitors the overall time the job has been running.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

See Also

The Community-contributed Microsoft Office Activity & Azure Logs Pack provides sample Pipelines to parse and streamline ingested Office 365 and EntraID/Azure AD data. You can import these and adapt them to your own needs.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

Job Fails After One Hour with statusCode: 401

A collection job fails after one hour with an HTTP error, `statusCode: 401`. This error could indicate an authentication failure, like an expired access token.

The cause could be from the temporary access token provided by Microsoft during the OAuth 2.0 authentication process. This token has a one-hour time to live (TTL) that is not configurable. Any collection job that exceeds this one-hour window is unable to refresh its authentication and fails.

To prevent job failure due to token expiration, you can:

- Limit the time range of the collection job to ensure it completes within the one-hour timeframe.
- Configure multiple Office 365 Message Trace Sources, each with a distinct, non-overlapping time range to sequentially collect the data in increments. This approach could duplicate events if the time ranges are

not carefully managed.

Dropping request because token invalid",“authToken”: “Bas...Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.6.3. Microsoft Graph Source

Microsoft has announced deprecation of the legacy Message Trace API and Reporting Web Service endpoint. The new Microsoft Graph Source is available, and legacy Message Trace support is scheduled for deprecation on April 8, 2026. Plan to migrate to this Source before that date to avoid data loss.

Cribl Stream supports receiving Microsoft 365 Message Trace data via the Microsoft 365 Graph Source. This data can be used to detect and report on malicious activity, including bulk emails, spoofed-domain emails, and data exfiltration.

The Microsoft 365 Graph Source uses a scheduled REST Collector under the hood. It runs one collection task every Poll interval, and a single Worker will process the collection. The data is paginated, so the Worker might make multiple calls to fetch the data.

Type: **Pull** | TLS Support: **Yes** | Event Breaker Support: **Yes**

TLS is enabled via HTTPS on this Source's underlying Microsoft Graph REST API.

Prerequisites and Microsoft 365 Setup

At a minimum, your Microsoft 365 service account or application must have permissions to access Message Trace data via Microsoft Graph and be assigned to a tenant that contains Exchange Online.

For detailed prerequisites and Microsoft Graph permission requirements, follow Microsoft's Graph-based message trace API onboarding guide. This guide walks you through registering an app, configuring `ExchangeMessageTrace.Read.All` application permissions, granting admin consent, and provisioning the required Service Principal in your tenant.

You must also ensure that your account or application has appropriate roles for message tracking and mail flow reporting in Exchange Online and Microsoft Entra ID,

While the exact roles and Graph permissions can vary by environment and Microsoft cloud, make sure to:

- Assign a role that allows access to message tracking/reporting.
- Ensure that the account or application is scoped appropriately for your tenant and compliance requirements.
- Confirm that all required Microsoft Graph application permissions have admin consent granted for your tenant.

To validate your configuration and experiment with queries and permissions, you can use Microsoft Graph Explorer. Graph Explorer lets you sign in to your tenant, test the Graph-based Message Trace APIs, and verify that your app registration and permissions return the data you expect before wiring them into this Source.

Modern Authentication (OAuth 2.0) Setup

To use OAuth, OAuth Secret, or OAuth Certificate authentication, you must configure an Application and its associated Service Principal within Microsoft Entra ID as part of your Microsoft 365 setup.

1. In the Microsoft Azure portal, create a Microsoft Entra ID App Registration for Cribl Stream. This will automatically create a corresponding Service Principal in your tenant. For details, see the Microsoft Azure documentation.
2. Grant your Application the Microsoft Graph permissions required for Message Trace.
    - Navigate to **API permissions** on your App Registration and select **Add a permission**.
    - On the **Microsoft APIs** tab, select **Microsoft Graph** and grant the appropriate Application permissions (`ExchangeMessageTrace.Read.All`).
3. Assign the Service Principal a Microsoft Entra ID role that enables it to access the Graph-based Message Trace data.

Configure Cribl Stream to Receive Microsoft Graph Message Trace Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original Input ID.
    - **Description**: Optionally, enter a description.
    - **Endpoint**: The Microsoft Graph API endpoint URL (for example, `https://graph.microsoft.com/v1.0/admin/exchange/tracing/messageTraces`).
    - **Poll interval**: How often (in minutes) to run the report. Must divide evenly into `60` minutes to create a predictable schedule, or **Save** will fail. The default value of `60` means the Source will run the collection job every 60 minutes.

3. Under **Authentication**:
    - **Authentication method**: Select an authentication method: OAuth, OAuth (text secret), or OAuth Certificate. The default is `OAuth`. All OAuth options rely on the OAuth 2.0 protocol.
    - **Client secret**: `client_secret` to pass in the OAuth request parameter.
    - **Tenant identifier**: Directory ID (tenant identifier) in Azure Active Directory.
    - **Client ID**: `client_id` to pass in the OAuth request parameter.
    - **Resource**: Resource to pass in the OAuth request parameter.
    - **Subscription plan**: Your Office 365 subscription plan for your organization, typically `Office 365 Enterprise`.
4. Next, you can configure the following **Optional Settings**:
    - Use the **Date range start** and **Date range end** settings to specify the time window for data collection. These settings help compensate for delays and gaps in Message Trace data. Use relative time formats for ongoing data collection and exact epoch values to backfill specific time gaps.
    - **Date range start**: Backward offset for the head of the search date range. Supports relative time format like `-3h@h` or exact epoch values like `1720536960`.
    - **Date range end**: Backward offset for the tail of the search date range. Supports relative time format like `-2h@h` or exact epoch values like `1720533360`.
    - **Log level**: For data collection's runtime log, set the verbosity level to one of `debug`, `info`, `warn`, or `error`. If not selected, the value defaults to `info`.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Processing, Retries, and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Authentication

OAuth Authentication

The default OAuth authentication method exposes the following fields, all required:

- **Client secret**: Directly enter the `client_secret` to pass in the OAuth request parameter.
- **Tenant identifier**: Directory ID (tenant identifier) in Microsoft Entra ID.
- **Client ID**: The `client_id` to pass in the OAuth request parameter.
- **Resource**: Resource parameter to pass in the OAuth request parameter. Defaults to: `https://graph.microsoft.com`.

OAuth Secret Authentication

Selecting **OAuth (text secret)** exposes three of the same controls as the default OAuth method, but you instead enter the **Client secret** by reference:

- **Client secret**: Use the drop-down to select an existing stored `client_secret` to pass in the OAuth request parameter. You can use the adjacent **Create button** to store a new, reusable secret.
- **Tenant identifier**: Directory ID (tenant identifier) in Microsoft Entra ID.
- **Client ID**: The `client_id` to pass in the OAuth request parameter.
- **Resource**: Resource parameter to pass in the OAuth request parameter. Defaults to: `https://graph.microsoft.com`.

OAuth Certificate Authentication

Selecting OAuth (certificate) exposes two of the same controls as the above OAuth methods (**Tenant identifier** and **Client ID**). But this authentication method - available in Cribl Stream 4.1.3 and newer - foregrounds controls to define the certificate you're using to authenticate. Treat all fields here as required, except for the optional **Passphrase**:

- **Certificate name**: Use the drop-down to select an existing certificate to pass in the OAuth request parameter. You can use the adjacent **Create button** to store a new, reusable cert.
- **Private key path**: Path to the private key to use. The key should be in PEM format. Can reference `$ENV_VARS`.
- **Passphrase**: If your private key requires a passphrase to decrypt it, enter that passphrase here.
- **Certificate path**: Path to the certificate to use. The cert should be in PEM format. Can reference `$ENV_VARS`.
- **Tenant identifier**: Directory ID (tenant identifier) in Microsoft Entra ID.
- **Client ID**: The `client_id` to pass in the OAuth request parameter.
- **Resource**: Resource parameter to pass in the OAuth request parameter. Defaults to: `https://graph.microsoft.com`.
- **Subscription Plan**: Select the Office 365 subscription plan for your organization. Options include:
    - **Office 365 Enterprise**
    - **Office 365 GCC**
    - **Office 365 GCC High**
    - **Office 365 DoD**

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's Pipeline drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

Retry type: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`. The default is set to `Backoff`, meaning Cribl Stream will use an exponential backoff for retries.

Initial retry interval (ms): Time interval between failed request and first retry (kickoff). Maximum allowed value is `20,000 ms` (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**. The default is set to `1000`.

Retry limit: Maximum number of times to retry a failed HTTP request. Maximum: `20`. A value of `0` means don't retry at all. The default value is `5`.

Backoff multiplier: Base for exponential backoff. The default value is `2` and means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

Retry HTTP codes: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429`, `500`, and `503`). Cribl Stream does not retry codes in the `200` series.

Honor Retry-After header: When toggled on (the default) and the retry-after header is present, Cribl Stream honors any retry-after header that specifies a delay, up to a maximum of `20` seconds. Cribl Stream always ignores retry-after headers that specify a delay longer than `20` seconds.

- Cribl Stream will log a warning message with the delay value retrieved from the retry-after header (converted to ms).
- When toggled off, Cribl Stream ignores all retry-after headers.

Retry connection timeout: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.

Retry connection reset: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.

Advanced Settings

Request timeout (seconds): Max time to wait for an individual Graph Message Trace request. The default is `300` seconds.

Reschedule tasks and Task reschedule limit: Automatically reschedule tasks that fail with non-fatal errors. The default for **Task reschedule limit** is `1`.

Job timeout: Max time a job can run. A value of `0` means the time is unlimited.

Time to live: How long to keep job artifacts on disk and in **Job Inspector**. The default is `4h`, meaning Cribl Stream keeps the collector's job artifacts on disk and visible in **Job Inspector** for 4 hours, then automatically cleans them up.

Disable time filter: Controls Collector event time filtering when a date range is specified.

Environment: For GitOps workflows, optionally restrict this configuration to a single Git branch; leave empty to enable everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are not part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__inputId`
- `__isBroken`
- `__source`

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Viewing Scheduled Jobs

After saving the Source:

- Reopen the Source's configuration modal and select the **Job Inspector** tab to view results for this Source's scheduled jobs.
- Or go to **Monitoring > System > Job Inspector > Currently Scheduled** to see jobs across all Collectors and Sources.

Mitigating Stuck-Job Problems

Occasionally, a scheduled job fails but continues running for a long time ("stuck", "orphaned", or "zombie" jobs). This can cause missing events downstream and HTTP timeouts in logs.

To reduce risk:

1. Lower **Request timeout (secs)** to a value shorter than the default (for example, under `300` seconds).
2. Set **Job timeout** to a maximum duration so stuck jobs are terminated automatically.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

See Also

The Community-contributed Microsoft Office Activity & Azure Logs Pack provides sample Pipelines to parse and streamline ingested Office 365 and EntraID/Azure AD data. You can import these and adapt them to your own needs.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

Job fails with statusCode: 401

If a collection job fails with HTTP 401, check for:

- Incorrect or missing Microsoft Graph permissions.
- Expired or invalid OAuth access tokens.
- Misconfigured Resource or scope values in the OAuth request.

8.1.6.4. Office 365 Services Source

Cribl Stream supports receiving data from the Microsoft Graph service communications API. This facilitates analyzing the status and history of service incidents on multiple Microsoft cloud services, along with associated incident and Message Center communications. For details, see Microsoft's Overview of the Graph API.

Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **YES**

TLS is enabled via the HTTPS protocol on this Source's underlying REST API.

Use a REST/API Collector to collect from other Microsoft Graph API resources.

Prerequisites

You need to have a registered application with the Microsoft identity platform to provide Cribl Stream access to make requests to the Microsoft Graph API.

From the registered application, you'll need its **Application (client) ID**, **Directory (tenant) ID**, and credentials to configure in the Cribl Stream Source.

For a complete, step-by-step walkthrough, see our Knowledge Base article Office 365 App Registration for Cribl Sources.

Microsoft Permissions

The application representing your Cribl Stream instance needs the following application permissions to pull data. The permission **Type** for both must be `Application` - `Delegated` is not sufficient:

- `ServiceHealth.Read.All`
- `ServiceMessage.Read.All`

For the full list of permissions exposed by Microsoft Graph, see the Microsoft Graph permissions reference.

Configure Cribl Stream to Receive Data from the Service API

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.

- To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.

2. In the **New Source** modal, configure the following under **General Settings**:
   - **Input ID**: Enter a unique name to identify this Office 365 Services definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
   - **Description**: Optionally, enter a description.
   - **Tenant ID**: Enter the Office 365 Azure **Directory (tenant) ID**.
   - **App ID**: Enter the Office 365 Azure **Application (client) ID**.
3. Under **Authentication**, select the **Authentication method** from the dropdown:
   - **Manual**: This default option provides a **Client secret** field, where you directly enter the required Office 365 Azure client secret.
   - **Secret**: This option instead exposes a **Client secret (text secret)** drop-down, from which you select a stored text secret to authenticate with. Select **Create** to configure a new secret.
4. Next, you can configure the following **Optional Settings**:
   - **Subscription Plan**: See Subscription Plan below.
   - **Content Types**: See the Content Types section below.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Processing, Retries, and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Subscription Plan

Select the Office 365 subscription plan for your Organization. Options include:

- **Office 365 Enterprise**
- **Office 365 GCC**
- **Office 365 GCC High**
- **Office 365 DoD**

Content Types

Here, you can configure polling separately for the following types of data from the Office 365 Service Communications API:

- **Current Status**: Get a real-time view of current and ongoing service incidents.
- **Messages**: Find incident and Message Center communications.

As of this revision, this Microsoft API provides data for Office 365, Yammer, Dynamics CRM, and Microsoft Intune cloud services. For each of these content types, this section provides the following controls:

Enabled: Toggle on for each service that you want to poll.

Interval: Optionally, override the default polling interval. See About Polling Intervals below.

Log level: Set the verbosity level to one of `debug`, `info` (the default), `warn`, or `error`.

About Polling Intervals

To poll the Office 365 Service Communications API, Cribl Stream uses the **Interval** field's value to establish the search date range and the cron schedule, for example: `*/${interval} * * * *`

Therefore, intervals set in minutes - those for **Current Status** - must divide evenly into 60 minutes to create a predictable schedule. Dividing 60 by intervals like `1`, `2`, `3`, `4`, `5`, `6`, `10`, `12`, `15`, `20`, or `60` itself yields an integer, so you can enter any of these values.

Cribl Stream will reject intervals like `23`, `42`, or `45`, or `75` - which would yield non-integer results, meaning unpredictable schedules.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

Retry type: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.

Initial retry interval (ms): Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.

Retry limit: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.

Backoff multiplier: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

Retry HTTP codes: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.

Honor Retry-After header: When toggled on (the default) and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds.

- Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
- When toggled off, Cribl Stream ignores all `retry-after` headers.

Retry connection timeout: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.

Retry connection reset: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.

Advanced Settings

Request timeout (secs): The maximum time period for an HTTP request to complete before Cribl Stream treats it as timed out. Defaults to `300` (5 minutes). Enter `0` to disable timeout metering.

Job timeout: Maximum time the job is allowed to run (for example, `30`, `45s`, or `15m`). Units are seconds, if not specified. Enter `0` for unlimited time. Defaults to `0`.

Time to live: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__inputId`
- `__isBroken`
- `__source`

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

How Cribl Stream Pulls Data

The Office 365 Services Source retrieves data using Cribl Stream scheduled Collection jobs, which include Discover and Collection phases. The Discover phase task returns the URL of the content to collect.

In the Source's **General Settings** > **Content Types** > **Interval** column, you configure the polling schedule for each Content Type independently.

The job scheduler spreads the Collection tasks across all available Workers. The collected content is paginated, so the collection phase might include multiple calls to fetch data.

Viewing Scheduled Jobs

This Source executes Cribl Stream's scheduled collection jobs. Once you've configured and saved the Source, you can view those jobs' results by reopening the Source's config modal and selecting its **Job Inspector** tab.

Each content type that you enabled gets its own separate scheduled job.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring** > **System** > **Job Inspector** > **Currently Scheduled** tab.

Mitigating Stuck-Job Problems

Occasionally, a scheduled job fails, but continues running for hours or even days, until someone intervenes and cancels it. If left alone, such a “stuck”, “orphaned,” or “zombie” job will never complete. This can cause missing events in downstream receivers, along with `HTTP timeout` or similar errors in Cribl Stream’s logs.

To keep stuck jobs from running excessively long:

- First, try setting **Advanced** > **Timeout (secs)** to a duration shorter than the default of `300` seconds (5 minutes).
- If adjusting **Timeout (secs)** does not fix the problem, try the global setting **Job Timeout** - whose default of `0` allows a job to run indefinitely - to a desired maximum duration. You’ll find **Job Timeout** among the task manifest and buffering limits.

Using these settings in tandem works like this:

- **Timeout (secs)** limits the time that Cribl Stream will wait for an HTTP request to complete.
- Then, if a job gets stuck and keeps running beyond that limit, **Job Timeout** can catch and terminate the job, because it monitors the overall time the job has been running.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

See Also

The Community-contributed Microsoft Office Activity & Azure Logs Pack provides sample Pipelines to parse and streamline ingested Office 365 and EntraID/Azure AD data. You can import these and adapt them to your own needs.

Troubleshooting

The Source’s configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

Dropping request because token invalid",“authToken”: “Bas...Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.7. Prometheus

You can use the following Sources to receive data from Prometheus services.

Source	Description
Prometheus Scraper	Receive batched data from Prometheus targets
Prometheus Remote Write	Receive metric data from Prometheus via the remote write protocol
Grafana	Receive metric and log data from Grafana Agent via Prometheus remote write
Loki	Receive log data from Grafana Loki

8.1.7.1. Prometheus Scraper Source

Cribl Stream supports receiving batched data from Prometheus targets. This is a pull Source; to ingest Prometheus streaming data, see Prometheus Remote Write.

Type: **Pull** | TLS Support: Configurable | Event Breaker Support: **No**

This Source does not support Prometheus metadata.

Configure Cribl Stream to Scrape Prometheus Data

As of version 4.2, we've removed this Source from Cribl Edge. Please use the Edge-specific Prometheus Edge Scraper Source instead.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Discovery type**: Use this drop-down to select a discovery mechanism for targets. See Discovery Type below for the options.

Some **Discovery type** options replace the **Targets** field with additional controls below the **Poll interval** and **Log level** fields - while also adding an **Assume Role** and/or **Target Discovery** left tab to the modal.

3. Next, you can configure the following **Optional Settings**:
    - **Extra dimensions**: Specify the dimensions to include in events. Defaults to `host` and `source`.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Discovery Type

Use this drop-down to select a discovery mechanism for targets. To manually enter a targets list, use Static (the default). To enable dynamic discovery of endpoints to scrape, select DNS or AWS EC2. Each selection exposes different controls and/or tabs, listed below.

Static Discovery

The `Static` option adds a **General Settings** > **Targets** field, in which you enter a list of specific Prometheus targets from which to pull metrics.

Values can be in URL or `host[:port]` format, for example: `http://localhost:9090/metrics`, `localhost:9090`, or `localhost`.

If you specify only `host[:port]`, the endpoint will resolve to: `http://host[:port]/metrics` - that is, the protocol will be prepended and the `/metrics` path will be appended in all cases, even if you add a path. If you need a nonstandard path, you must enter the full URL (including protocol) with path. For example: `http://localhost:9090/my/custom/path`.

DNS Discovery

The `DNS` option adds a Target Discovery tab to the modal, and adds two extra fields to its **General Settings** tab:

- **DNS names**: Enter a list of DNS names to resolve.
- **Record type**: Select the DNS record type to resolve. Defaults to `SRV` (Service). Other options are `A` or `AAAA`.

AWS EC2 Discovery

The `AWS EC2` option adds Assume Role and Target Discovery tabs to the modal, and adds one extra field to the **Optional Settings** tab:

- **Region**: The AWS Region in which to discover EC2 instances with metrics endpoints to scrape.

This option also adds controls in the **Advanced Settings** tab, as described below.

Authentication (Prometheus)

Use the **Authentication method** drop-down to select one of these authentication options for Prometheus:

- **Manual**: In the resulting **Username** and **Password** fields, enter Basic authentication credentials corresponding to your Prometheus targets.

- **Secret**: This option exposes a **Secret** drop-down, in which you can select a stored secret that references your credentials described above. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. Click **Create** if you need to configure a new secret.

Assume Role

With the AWS EC2 target discovery type, you can configure AssumeRole behavior on AWS.

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

- **Enable for EC2**: Toggle on if you want to use `AssumeRole` credentials to access EC2.
- **AssumeRole ARN**: Enter the Amazon Resource Name (ARN) of the role to assume.
- **External ID**: Enter the External ID to use when assuming the role.
- **Duration (seconds)**: Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Target Discovery

Setting the General Settings > Discovery type drop-down to DNS or AWS EC2 exposes this tab. These two discovery types expose different controls here.

Target Discovery for DNS

Setting the Discovery type drop-down to DNS exposes the following **Target Discovery** fields.

Metrics protocol: Select `http` (the default) or `https` as the protocol to use when collecting metrics.

Metrics path: Specify a path to use when collecting metrics from discovered targets. Defaults to `/metrics`.

Target Discovery for AWS

Setting the Discovery type drop-down to AWS EC2 exposes the following **Target Discovery** controls. The first controls is a special case:

- **Authentication method**: Select the **Auto**, **Manual**, or **Secret** button to determine how Cribl Stream will authenticate against AWS. Each selection changes the fields displayed on this tab - see AWS Authentication Options for details.

These remaining controls are displayed for all **Authentication method** selections:

- **Metrics protocol**: Select `http` (the default) or `https` as the protocol to use when collecting metrics.

- **Metrics port**: Specify the port number to append to the metrics URL for discovered targets. Defaults to `9090`.
- **Metrics path**: Specify a path to use when collecting metrics from discovered targets. Defaults to `/metrics`.
- **Use public IP**: Toggle on (default) to use the public IP address for discovered targets. Toggle off to use a private IP address.
- **Search filter**: Click **Add filter** to apply filters when searching for EC2 instances. Each filter row provides two columns:
    - **Filter name**: Select standard attributes from the drop-down, or type in custom attributes.
    - **Filter values**: Enter values to match within this row's attribute, Press `Enter` between values. (If you specify no values, the search will return only `running` EC2 instances.)

AWS Authentication Options

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. This option displays the same fields as Auto, plus:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. This option displays the same fields as Auto, plus:

- **Secret key pair**: Use the drop-down to select a secret key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS. Click **Create** if you need to configure a key pair.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Reject unauthorized certificates: Whether to accept certificates (such as self-signed certificates, for example) that cannot be verified against a valid Certificate Authority. Defaults to toggled on.

HTTP Connection Timeout: The maximum time (in seconds) allowed to establish an HTTP connection. Enter `0` to disable the timeout and wait indefinitely. Defaults to `30` seconds for new Collectors, while existing Collectors default to `0` to maintain legacy behavior.

Job timeout: Maximum time a job is allowed to run. Defaults to `0`, for unlimited time. Units are seconds if not specified. Sample entries: `30`, `45s`, `15m`.

Time to live: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Advanced Settings for AWS

These two additional settings appear only when **Optional Settings** > **Discovery Type** is set to `AWS EC2`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to reject certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on, the restrictive option.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__source`
- `__isBroken`
- `__inputId`
- `__final`
- `__criblMetrics`
- `__channel`
- `__cloneCount`

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

How Cribl Stream Pulls Data

The Prometheus Source retrieves data using Cribl Stream scheduled Collection jobs. You determine the schedule using your **Poll interval** entry.

With the `DNS` or `AWS EC2` **Discovery Type**, these jobs include both Discover and Collection phases. The Discover phase runs on a single Worker, and returns 1 collection task per discovered target.

The job scheduler spreads the Collection tasks across all available Workers.

Viewing Scheduled Jobs

This Source executes Cribl Stream's scheduled collection jobs. Once you've configured and saved the Source, you can view those jobs' results by reopening the Source's config modal and clicking its **Job Inspector** tab.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring** > **System** > **Job Inspector** > **Currently Scheduled** tab.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

Dropping request because token invalid",“authToken”: “Bas...Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.7.2. Prometheus Remote Write Source In Cribl Stream

Cribl Stream supports receiving metric data from Prometheus instances that are configured to send data via the remote write protocol.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

This Source assumes that incoming data is snappy-compressed.

Looking for the Prometheus Remote Write Source in **Cribl Search**? See Add a Prometheus Remote Write Source for Cribl Search.

Configure Cribl Stream to Receive Metrics from Prometheus Remote Write Sources

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP to listen to. Defaults to `0.0.0.0`.
    - **Port**: Enter the port number to listen on.
    - **Remote Write API endpoint**: Enter the absolute path on which to listen for Prometheus requests. Defaults to `/write`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/write`.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, TLS, Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header, or select **Generate** if you need a new token.
- **Auth token (text secret)**: Provide an HTTP token referenced by a secret. Select a stored text secret in the resulting drop-down, or select **Create** to configure a new secret.
- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials. Click **Generate** if you need a new password.
- **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled off.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter:

`worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always On`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; etc.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

> The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__criblMetrics`
- `__final`
- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.

- `__inputId`
- `__srcIpPort` - See details below.
- `_time`
- `_value`

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Prometheus Remote Write client sending data to this Source.

When any proxies (including load balancers) lie between the Prometheus Remote Write client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Detecting Metrics' Types

Because Prometheus remote write requests don't specify metrics' types, Cribl Stream applies the following rules to determine the type as we ingest them:

- If the metric's name ends with `_total`, `_sum`, `_count`, or `_bucket`, the type is set to `counter`.
- Otherwise, the metric's type is set to `gauge`.

This is consistent with the type detection practiced by other services implementing the remote write protocol. See, for example, New Relic's and Elastic's documentation.

Note that Cribl Stream supports the `timer` type in addition to `counter` and `gauge`.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

Dropping request because token invalid",“authToken”: “Bas...Njc=”

8.1.7.3. Grafana Source

Cribl Stream supports receiving metric and log data from Grafana Agent instances via the Prometheus remote write specification. The Grafana Agent uses Prometheus for metrics collection and Grafana Loki for log collection.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

This Source assumes that incoming data is snappy-compressed.

Configure Cribl Stream to Receive Metrics and Logs from Grafana Agent Sources

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP to listen to. Defaults to `0.0.0.0`.
    - **Port**: Enter the port number to listen on.
3. Next, you can configure the following **Optional Settings**. However, at least one endpoint is required.
    - **Remote Write API endpoint**: Absolute path on which to listen for Grafana Agent's remote write requests. Defaults to `/api/prom/push`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/api/prom/push`.
    - **Logs API endpoint**: Absolute path on which to listen for Loki logs requests. Defaults to `/loki/api/v1/push`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/loki/api/v1/push`.

- **Enable structured metadata**: Toggle on to parse and extract structured metadata from incoming Loki log payloads. Structured metadata lets you attach arbitrary key-value pairs (beyond labels) to each log entry, supporting high-cardinality fields without polluting the label set. Extracted metadata is stored in the event field `__structuredMetadata` to be forwarded through your Pipeline.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, TLS, Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

The **Authentication** tab provides separate **Loki** and **Prometheus** sections, enabling you to configure these inputs separately. The two sections provide identical options.

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Enter the bearer token that must be included in the authorization header.
- **Auth token (text secret)**: Provide an HTTP token referenced by a secret. Select a stored text secret in the resulting drop-down, or select **Create** to configure a new secret.
- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled off.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.
>
> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.

- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; and so forth.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__inputId`
- `__labels` - For log events only - will contain all the labels found in each event's corresponding Loki stream.
- `__structuredMetadata` - Added only when **Enable structured metadata** is toggled on.
- `__srcIpPort` - See details below.

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Grafana client sending data to this Source.

When any proxies (including load balancers) lie between the Grafana client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Detecting Metrics' Types

Because Prometheus remote write requests don't specify metrics' types, Cribl Stream applies the following rules to determine the type as we ingest them:

- If the metric's name ends with `_total`, `_sum`, `_count`, or `_bucket`, the type is set to `counter`.
- Otherwise, the metric's type is set to `gauge`.

This is consistent with the type detection practiced by other services implementing the remote write protocol. See, for example, New Relic's and Elastic's documentation.

Note that Cribl Stream supports the `timer` type in addition to `counter` and `gauge`.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

Dropping request because token invalid",“authToken”: “Bas...Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.7.4. Loki Source

Cribl Stream supports receiving log data from Grafana Loki via an adaptation of the Protobuf (Protocol Buffers) specification.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

This Source assumes that incoming data is snappy-compressed.

Configure Cribl Stream to Receive Loki Logs Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP to listen to. Defaults to `0.0.0.0`.
    - **Port**: Enter the port number to listen on.
    - **Logs API endpoint**: Absolute path on which to listen for Loki logs requests. Defaults to `/loki/api/v1/push`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/loki/api/v1/push`.
    - **Enable structured metadata**: Toggle on to parse and extract structured metadata from incoming Loki log payloads. Structured metadata lets you attach arbitrary key-value pairs (beyond labels) to each log entry, supporting high-cardinality fields without polluting the label set. Extracted metadata is stored in the event field `__structuredMetadata` to be forwarded through your Pipeline.
3. Next, you can configure the following **Optional Settings**: - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or

hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, TLS, Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Authentication type** drop-down to specify how Loki's Promtail agent will authenticate against Cribl Stream:

- **None**: Don't use authentication.
- **Auth token**: Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Auth token (text secret)**: Provide an HTTP token referenced by a secret. Select a stored text secret in the resulting drop-down, or select **Create** to configure a new secret.
- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate server certs**: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters.

(For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always On`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; and so forth.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

> The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__final`
- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__inputId`

- `__labels` - Will contain all the labels found in each event's corresponding Loki stream.
- `__structuredMetadata` - Added only when **Enable structured metadata** is toggled on.
- `__srcIpPort` - See details below.
- `_time`

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Loki client sending data to this Source.

When any proxies (including load balancers) lie between the Loki client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

Dropping request because token invalid",“authToken”: “Bas…Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.8. Splunk

You can use the following Sources to receive data from Splunk services.

Source	Description
Splunk HEC	Receive data over HTTP/S using the Splunk HEC
Splunk Search	Ingest data by executing Splunk search queries
Splunk TCP	Receive Splunk data from Universal or Heavy Forwarders

8.1.8.1. Splunk HEC Source In Cribl Stream

Cribl Stream supports receiving data over HTTP/S using the Splunk HEC (HTTP Event Collector).

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

For examples of optimizing incoming Splunk events for downstream services, see our Splunk Stream to Cribl Stream, Splunk to Elasticsearch, and Splunk to Exabeam guides.

Also examine several Splunk Forwarder/UF Packs on the Cribl Packs Dispensary. These provide processing Pipelines to optimize Splunk Windows events and Universal Forwarder internal events. You can directly import these and adapt them to your needs.

Looking for the Splunk HEC Source in **Cribl Search**? See Add a Splunk HEC Source for Cribl Search.

Configure Cribl Stream to Receive Data over Splunk HEC

Cribl Stream ships with a Splunk HEC Source preconfigured to listen on Port 8088. You can clone or directly modify this Source to further configure it, and then enable it.

In Cribl Stream, configure Splunk HEC:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. Configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Splunk HEC Source definition. The default Source is prefilled with the value `in_splunk_hec`, which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.

- **Description**: Optionally, enter a description.
- **Address**: Enter the hostname/IP on which to listen for HTTP(S) data. Such as: `localhost` or `0.0.0.0`. Supports IPv4 and IPv6 addresses.
- **Port**: Enter the port number.
- **Splunk HEC endpoint**: Absolute path on which to listen for the Splunk HTTP Event Collector API requests. Defaults to `/services/collector`.
  - This single endpoint supports JSON events via `/event`, raw events via `/raw`, and Splunk S2S events via `/s2s`. See the Splunk REST API endpoints documentation and the following examples. The Source will automatically detect where to forward the request.

3. Next, you can configure the following **Optional Settings**:
   - **Allowed Indexes**: List the values allowed in the HEC event index field. Allows wildcards. Leave blank to skip validation.
   - **Enable Splunk HEC acknowledgements**: Whether to enable Splunk HEC acknowledgments. Default is toggled off. See Working with HEC Acks below to learn about context and limitations.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing, Auth tokens and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Working with HEC Acks

Cribl Stream sends a `200` or similar HTTP response code when it receives an event.

Some senders also send **ack requests**, which differ from events. This type of request asks Cribl Stream to acknowledge delivery of an array of event IDs. (A sender might **require** HEC acks to be enabled. Cribl does not maintain a comprehensive list of senders that require acks - please refer to your sender's documentation.)

How Cribl Stream responds depends on **Optional Settings** > **Enable Splunk HEC acknowledgements**:

- When **Enable Splunk HEC acknowledgements** is toggled on, Cribl Stream will respond to an ack request with an acknowledgement affirming that Cribl Stream received each of the events whose IDs were listed in the array.
- When **Enable Splunk HEC acknowledgements** is toggled off, Cribl Stream will respond to an ack request with a `400` error and text saying that `ACK` is disabled.

It makes sense to turn on Splunk HEC acknowledgements for senders that keep TCP connections open while waiting for an ack, because this behavior can exhaust available file descriptors.

There is a caveat to how HEC acks work in Cribl Stream: The ack response simply cites whatever event IDs appeared in the ack request, regardless of whether or not those events were ever received or processed by Cribl Stream. In that sense, what Cribl Stream sends is a "fake ack:" unlike acknowledgements in some other systems, the ack in itself is meaningless. It mainly serves to spare a sender from keeping TCP connections open needlessly.

To determine whether Cribl Stream successfully ingested an event, pay no attention to acks. Instead, verify that the sender received a `200` or similar HTTP response from the Splunk HEC source.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as, the system's CA). Default is toggled off.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Event Breakers

Splunk HEC Source Event Breakers only function with the `/raw` endpoint. The `/event` and `/s2s` endpoints operate on the assumption that the data they receive is already properly formatted into individual events and therefore do not utilize event breakers.

- `/raw`: Event Breakers split incoming raw, unstructured data into distinct events before processing them through Pipelines.
- `/event`: Event Breakers are not applied since data is already segmented in JSON format.
- `/s2s`: Event Breakers are not applied since data is already structured in the Splunk-to-Splunk format.

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can add Fields to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to determine field's value (can be a constant).

Fields specified on the **Fields** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

In particular, where incoming events have no `index` field, this Source adds one with the literal value `default`. You can override this value by using **Add Field** to specify an `index` field, and then setting its **Value** to an expression of the following form: `index == 'default' ? 'myIndex' : index`

Fields here are evaluated and applied **after** any fields specified in the Auth Tokens section.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Auth Tokens

Provides the authentication token required to access this Source's data. Ensure the token is valid and has the necessary permissions for data retrieval.

Not providing a token allows **unauthorized access**.

If you exclusively use secret tokens for your Splunk HEC Source, be aware that Cribl Stream versions older than 4.8.2 will not recognize them. This can lead to unintended consequences, such as unsecured data ingestion. To ensure compatibility and security, configure both regular HEC tokens and secret tokens, especially in environments with mixed versions.

Authentication method: Choose between manually entering a **Token** or selecting a stored **Secret**. If you opt for manual entry, type the token directly or **Generate** a new one. For added security, use the secrets store to select a stored **Token (text secret)** or **Create** a new one.

Enable token: Controls the status of the specified authentication token. When toggled on, the specified auth token is active and can be used for client access. If toggled off, the token is disabled, causing it to be skipped and ignored during initialization. This means that any client attempting to use a disabled token will not be granted access.

Disabling a token can be helpful when you need to temporarily stop receiving data from a particular service or application. Disabling a token prevents the service or application from sending data to your endpoint without having to completely delete the token.

See also Periodic Logging for information on how auth tokens affect product logging.

Description: Optional (but recommended), the description for this token. Although this field is optional, the token description becomes the display name for the token. You can select this display name to expand or collapse the token details.

Allowed indexes: Optionally, specify which indexes are permissible for events ingested using this token. Supports multiple index values and wildcards.

Fields: Fields to add to events referencing this token. Each field is a **Name/Value** pair.

Fields specified on the **Auth Tokens** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

In particular, where incoming events have no `index` field, this Source adds one with the literal value `default`. You can override this value by using **Add Field** to specify an `index` field, and then setting its **Value** to an expression of the following form: `index == 'default' ? 'myIndex' : index`

Fields here are evaluated and applied **before** any fields specified in the Fields section.

Advanced Settings

Use Universal Forwarder time zone (S2S only): Toggle on (default) to have Event Breakers determine the time zone for events based on Universal Forwarder-provided metadata when the time zone can't be inferred from the raw event.

Drop control fields: Toggle on (default) to drop Splunk control fields such as `crcSalt` and `savedPort`. Toggle off if you want to retain these fields and store the data in the `__ctrlFields` internal field.

Extract metrics (S2S only): Toggle on if you're using Splunk's S2S protocol. When enabled, Cribl will identify and handle Splunk metric events, ensuring that metric events generated by Splunk are processed and forwarded as metrics.

Emit per-token request metrics: Toggle on to emit per-token `<prefix>.http.perToken` and summary `<prefix>.http.summary` request metrics. This enables Periodic Logging.

CORS allowed origins: If you need to enable cross-origin resource sharing with Splunk senders, use this field to specify the HTTP origins to which Cribl Stream should send `Access-Control-Allow-*` headers. You can enter multiple domains and use wildcards. This and its companion **CORS allowed headers** option should seldom be needed - see Working with CORS Headers below. Both options are available in Cribl Stream 4.1.2 and newer.

CORS allowed headers: List HTTP headers that Cribl Stream will send in a CORS preflight response to your configured (above) **CORS allowed origins** as `Access-Control-Allow-Headers`. Enter `*` to allow all headers.

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; and so forth.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request

> frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is

done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__hecToken`
- `__inputId`
- `__ctrlFields`
- `__s2sVersion` - value can be either `v3` or `v4`. This field is present only when the Source is receiving S2S-encoded payloads.
- `__srcIpPort` - See details below.
- `__TZ`

Universal Forwarder Time Zone

The `__TZ` field uses the universal forwarder time zone to mitigate cases where incoming events have timestamp strings but no time zone information. Event Breakers use the `__TZ` field to derive time zone information, enabling them to set the `_time` field correctly. See Using the UF Time Zone for additional information.

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Splunk HEC client sending data to this Source.

When any proxies (including load balancers) lie between the Splunk HEC client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Working with CORS Headers

The **CORS allowed origins** and **CORS allowed headers** settings are needed **only** when you want JavaScript code running in a web browser to send events to the Splunk HEC Source. In this (uncommon) scenario, the code in question sends a preflight request that includes CORS (Cross-Origin Resource Sharing) headers, and the Splunk HEC Source includes CORS headers in its response.

If the settings (and thus the CORS headers in the response) are correct, the browser will allow the code in question to complete requests to the Splunk HEC Source - that is, allow the code to read the Source's responses. (These settings are optional because for some web applications it's sufficient to send requests **without** reading responses.)

The CORS header dance begins when the web browser sends a request with a header stating its origin. What happens next depends on how you've configured the CORS header settings, as explained in the tables below. (For these examples, we'll assume an origin of `https://mycoolwebapp:3001`.)

CORS allowed origins value	What Splunk HEC Source does	Does Browser then allow code to read response?
`*`	Send `Access-Control-Allow-Origin: *` header.	Yes
Multiple wildcard values	If origin value matches any wildcard, send `Access-Control-Allow-Origin: https://mycoolwebapp:3001` header.	Yes
Nothing (not configured)	Send neither an `Access-Control-Allow-Origin` nor an `Access-Control-Allow-Headers` header.	No

The web browser's request might also include an `Access-Control-Request-Headers` header that lists one or more headers that it wants to use in a subsequent `POST` request (that is, the "actual" as opposed to preflight request). The Splunk HEC Source can then respond with an `Access-Control-Allow-Headers` header **if and only if** it's also sending the `Access-Control-Allow-Origin` header.

CORS allowed headers value	What Splunk HEC Source does	Does Browser then allow code to read response?
*	Send `Access-Control-Allow-Headers` header with the same list the client sent.	Yes
Multiple wildcard values	If any headers from the client's list match wildcards, send `Access-Control-Allow-Headers` header listing the matching headers.	Yes
Nothing (not configured)	Do not send an `Access-Control-Allow-Headers` header.	Probably not

Format and Endpoint Examples

Splunk HEC to Cribl Stream

- Configure Cribl Stream to listen on port `8088` with an auth token of `myToken42`.
- Send a payload to your Cribl Stream receiver.

Note: Token specification can be either `Splunk <token>` or `<token>`.

Splunk HEC - JSON Event Examples | Splunk HEC - Raw Event Example | Splunk HEC - Health Event Example

```
curl -k "http://<myCriblHost>:8088/services/collector/event" \
-H "Authorization: myToken42" \
-d '{"event":"this is a sample event ", "host":"myHost", "source":"mySource", "fie


curl -k "http://<myCriblHost>:8088/services/collector" \
-H "Authorization: myToken42" \
-d '{"event":"this is a sample event ", "host":"myHost", "source":"mySource", "fie

# Multiple Events
curl -k "http://<myCriblHost>:8088/services/collector" \
-H "Authorization: myToken42" \
-d '{"event":"this is a sample event ", "host":"myHost", "source":"mySource", "fie

# Metrics Events
curl -k "http://<myCriblHost>:8088/services/collector/event" \
-H "Authorization: myToken42" \
-d '{"event":"metric", "host":"myHost", "fields":{"_value":3850,"metric_name":"ker

curl -k "http://<myCriblHost>:8088/services/collector/event" \
-H "Authorization: myToken42" \
-d '{"host":"myHost", "fields":{"_value":3850,"metric_name":"kernel.entropy_avail"

# Send the auth token as a query parameter, with no additional configuration
curl -k "http://<myCriblHost>:8088/services/collector/event?token=mToken42" \
-d '{"event":"this is a sample event ", "host":"myHost", "source":"mySource", "fie
```

Copy

Splunk HEC to Cribl.Cloud

- Navigate to Cribl.Cloud's Splunk HEC Source > **Auth Tokens** tab.
- Copy your token out of the **Token** field.
- From the command line, use `https`, your Cribl.Cloud portal's Ingress Address and port, and the token's value.

Splunk HEC > Cribl Cloud endpoint

```
curl -k "https://default.main.<organizationId>.cribl.cloud:8088/services/collector" \
-H "Authorization: <token>" \
-d '{"event": "Goats are better than ponies."}{"event": "Goats are better climbers."}
```

Copy

With a Cribl.Cloud Enterprise plan, generalize the above URL's `default.main` substring to `<groupName>.<workspaceName>` when sending to other Worker Groups and/or Workspaces.

Periodic Logging

Cribl Stream emits metrics about incoming requests when **Advanced Settings > Emit per-token request metrics** is enabled. These metrics can be routed to a Destination by enabling the Cribl Metrics Source. There are two categories of metrics: per-token metrics provide details for each configured token, while summary metrics aggregate data across all tokens. If no tokens are configured, only summary metrics will be emitted.

Per-token metric names will be prefixed by `<basePrefix>.http.perToken.`, where `<basePrefix>` is the configured base prefix for the Cribl Metrics Source. The available metric names are listed below, excluding the prefix. As these are counter-type metrics, they will not be emitted if the value is zero.

- `bytesReceived`
- `numRequests`
- `numEvents`
- `numErrors` *Note: this is cumulative of specific error types below as well as a catch-all for other types*
- `numParserErrors`
- `numToDisabledToken`

The summary metric names will be prefixed by `<basePrefix>.http.summary.` where `<basePrefix>` is the configured base prefix for the Cribl Metrics Source. The available metric names are listed below, excluding the prefix. As these are counter-type metrics, they will not be emitted if the value is zero.

- `bytesReceived`
- `numRequests`
- `numEvents`
- `numErrors` *Note: this is cumulative of specific error types below as well as a catch-all for other types*
- `numParserErrors`
- `numToDisabledToken`
- `numAuthFailures`

- `numToWrongUrl`
- `numAckReqs`
- `numReqsAcked`

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

"Malformed HEC event"

The event is missing both `event` and `fields` fields.

"Invalid token"

Auth token(s) are defined, but the token specified in the HEC request doesn't match any defined tokens, therefore it's invalid.

"Invalid index"

8.1.8.2. Splunk Search Source

Cribl Stream supports receiving Splunk search data from Splunk Search.

Type: **Pull** | TLS Support: **Yes** | Event Breaker Support: **YES**

Configure Cribl Stream to Receive Splunk Search Data

In Cribl Stream, configure Splunk Search Data:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Splunk Search Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Cron schedule**: Enter a cron expression to define the schedule on which to run this job. Defaults to one run every 15 minutes. The **Estimated Schedule** below this field shows the next few collection runs, as examples of the cron interval you've scheduled. You enter the **Cron schedule** expression in UTC time, but the **Estimated Schedule** examples are displayed in local time.
3. Under **Search**, configure the following:
    - **Search**: Enter the Splunk query. For example: `index=myAppLogs level=error channel=myApp OR | mstats avg(myStat) as myStat WHERE index=myStatsIndex`.
    - **Search head**: Enter the search head base URL. The default is `https://localhost:8089`.
    - **Earliest**: You can enter the earliest time boundary for the search. This maybe be an exact or relative time. For example: `2022-01-14T12:00:00Z` or `-16m@m`.
    - **Latest**: You can enter the latest time boundary for the search. This maybe be an exact or relative time. For example: `2022-01-14T12:00:00Z` or `-16m@m`.
4. Next, you can configure the following **Optional Settings**:

- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Under **Authentication**, select the **Authentication type** from the dropdown:
    - **None**: Don't use authentication. Compatible with REST servers like AWS, where you embed a secret directly in the request URL.
    - **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
    - **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or click **Create** to configure a new secret.
    - **Bearer Token**: Provide the `token` value configured and generated in Splunk.
    - **Bearer Token (text secret)**: Provide the Bearer Token referenced by a secret. Select a stored text secret in the resulting **Token (text secret)** drop-down, or click **Create** to configure a new secret.
6. Optionally, you can adjust the Processing, Retries, and Advanced settings outlined in the sections below.
7. Select **Save**, then **Commit & Deploy**.

Processing Settings

Event Breakers

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to the `Splunk Search Ruleset`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.

- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

Retry type: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.

Initial retry interval (ms): Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.

Retry limit: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.

Backoff multiplier: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, etc.

Retry HTTP codes: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.

Honor Retry-After header: When toggled on (the default) and the `retry-after` header is present, Cribl Stream honors any `retry-after` header that specifies a delay, up to a maximum of 20 seconds. Cribl Stream always ignores `retry-after` headers that specify a delay longer than 20 seconds.

- Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
- When toggled off, Cribl Stream ignores all `retry-after` headers.

Retry connection timeout: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.

Retry connection reset: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.

Advanced Settings

Reject unauthorized certificates: Whether to accept certificates (such as self-signed certificates, for example) that cannot be verified against a valid Certificate Authority. Defaults to toggled on.

Search endpoint: Rest API used to conduct a search. Defaults to `/services/search/v2/jobs/export`.

Output mode: Format of the returned output. Defaults to JSON format.To parse the returned JSON, add the Cribl event breaker which parses newline delimited events in the Event Breakers tab.

Events returned from Splunk search can also be returned in the more compact CSV format. To use CSV format, set the **Output mode** to CSV and specify the CSV event breaker in the Event Breakers tab.

Endpoint parameters: Optional HTTP request parameters to append to the request URL. These refine or narrow the request. Click **Add Parameter** to add parameters as key-value pairs:

- **Field Name**: Unique name for the field you're adding.
- **Value**: JavaScript expression to compute the field's value (can be a constant).

Endpoint headers:: Click **Add Header** to (optionally) add request headers to send to the endpoint, as key-value pairs:

- **Name**: Header name.
- **Value**: JavaScript expression to compute the header's value, normally enclosed in backticks (for example, `` `${earliest}` ``). Can also be a constant, enclosed in single quotes (`'earliest'`). Values without delimiters (for example, `earliest`) are evaluated as strings.

Log level: Set the verbosity level for the data collection's runtime log.

Job timeout: Maximum time a job is allowed to run. Defaults to `0`, for unlimited time. Units are seconds if not specified. Sample entries: `30`, `45s`, `15m`.

Time to live: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

Request Timeout (secs): Here, you can set a maximum time period (in seconds) for an HTTP request to complete before Cribl Stream treats it as timed out. Defaults to `0`, which disables timeout metering.

Encoding: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`
- `__outputMode`

How Cribl Stream Pulls Data

This Collector-based Source will gather data from the specified **Search head** URL repeatedly, on the interval specified in the **Cron schedule** field. A single Worker executes each collection job.

If the Leader goes down, search jobs in progress will complete, but future scheduled searches will not run until the Leader relaunches.

Mitigating Stuck-Job Problems

Occasionally, a scheduled job fails, but continues running for hours or even days, until someone intervenes and cancels it. If left alone, such a "stuck", "orphaned," or "zombie" job will never complete. This can cause missing events in downstream receivers, along with `HTTP timeout` or similar errors in Cribl Stream's logs.

To keep stuck jobs from running excessively long:

- First, try setting **Advanced** > **Request timeout (secs)** - whose default of `0` means "wait forever" - to a desired maximum duration.
- If adjusting **Timeout (secs)** does not fix the problem, try setting **Advanced** > **Job Timeout** - whose default of `0` allows a job to run indefinitely - to a desired maximum duration.

Using these settings in tandem works like this:

- **Timeout (secs)** limits the time that Cribl Stream will wait for an HTTP request to complete.
- Then, if a job gets stuck and keeps running beyond that limit, **Job Timeout** can catch and terminate the job, because it monitors the overall time the job has been running.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.8.3. Splunk TCP Source In Cribl Stream

Cribl Stream supports receiving Splunk data from Universal or Heavy Forwarders.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

For examples of optimizing incoming Splunk events for downstream services, see our Splunk Stream to Cribl Stream, Splunk to Elasticsearch, and Splunk to Exabeam guides.

Also examine several Splunk Forwarder/UF Packs on the Cribl Packs Dispensary. These provide processing Pipelines to optimize Splunk Windows events and Universal Forwarder internal events. You can directly import these and adapt them to your needs.

Looking for the Splunk TCP Source in **Cribl Search**? See Add a Splunk TCP Source for Cribl Search.

Configure Cribl Stream to Receive Splunk TCP Data

Cribl Stream ships with a Splunk TCP Source preconfigured to listen on Port 9997. You can clone or directly modify this Source to further configure it, and then enable it.

In Cribl Stream, configure Splunk TCP:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. Configure the following under **General Settings**:
    - **Input ID**: Enter a unique name. The default Source is prefilled with the value `in_splunk_tcp`, which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter hostname/IP to listen for Splunk data. For example, `localhost` or `0.0.0.0`.

- **Port**: Enter port number.

3. Next, you can configure the following **Optional Settings**:
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing, Auth tokens and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Event Breakers

Event Breakers are applied only to raw, unparsed data. Data originating from Splunk Heavy Forwarders or Universal Forwarders with sourcetypes configured for event breaking or indexed extractions bypasses Event Breakers, as it is already segmented into discrete events.

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.

- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Auth Tokens

Add Token: Click to add authorization tokens. Each token's section provides the fields listed below. If no tokens are specified, unauthorized access **will be permitted**.

Token: Shared secrets to be provided by any Splunk forwarder (`Authorization: <token>`). Click **Generate** to create a new secret.

Description: Optional description for this token.

Advanced Settings

Enable proxy protocol: Toggle on if the connection is proxied by a device that supports Proxy Protocol v1 or v2.

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` (such as, all IPs).

Active connection limit: Maximum number of active connections allowed per Worker Process. Defaults to `1000`. Set a lower value if connection storms are causing the Source to hang. Set `0` for unlimited connections.

Socket idle timeout (seconds): The duration that Cribl Stream will wait for activity on an idle TCP socket before closing the connection. Disabled when set to `0`, the default.

Forced socket termination timeout (seconds): The extra time the server waits before forcibly closing a socket that has been idle (**TCP socket idle timeout**) or exceeded its maximum lifespan (**TCP socket max lifespan**) but has not yet properly closed. This prevents resource leaks caused by unresponsive clients or network issues. Configure based on network latency and client behavior. Default: `30` seconds. Set to `0` to disable.

Socket max lifespan (seconds): The duration that a socket is allowed to remain open, regardless of activity. This setting prevents resource exhaustion (such as TCP pinning) by limiting the lifespan of connections.

Configure based on expected connection durations and resource availability. Disabled when set to `0`, the default.

Max S2S version: The highest version of the Splunk-to-Splunk protocol to expose during handshake. Defaults to `v4`; `v3` is also available.

Use Universal Forwarder time zone: Displayed (and enabled by default) only when **Max S2S version** is set to `v4`. Provides Event Breakers with a `__TZ` field, which derives events' time zone from UF-provided metadata. See Using the UF Time Zone and Configuring a Splunk Forwarder, below.

Drop control fields: Toggle on (default) to drop Splunk control fields such as `crcSalt` and `savedPort`. Toggle off if you want to retain these fields and store the data in the `__ctrlFields` internal field.

Extract metrics: Toggle on if you're using Splunk's S2S protocol. When enabled, Cribl will identify and handle Splunk metric events, ensuring that metric events generated by Splunk are processed and forwarded as metrics.

Compression: This setting controls how Cribl Stream reacts to incoming compressed Splunk-to-Splunk `v4` data. Select the **Compression** type from the drop-down list:

- `Automatic`: Cribl Stream will automatically decompress incoming `S2S v4` data.
- `Disabled`: Cribl Stream will reject compressed `S2S v4` connections, closing them immediately. This is the default setting for all Sources, both pre-existing (in versions 4.11 and older) and newly created.

This setting is not available when **Max S2S version** is set to `v3`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Using the UF Time Zone

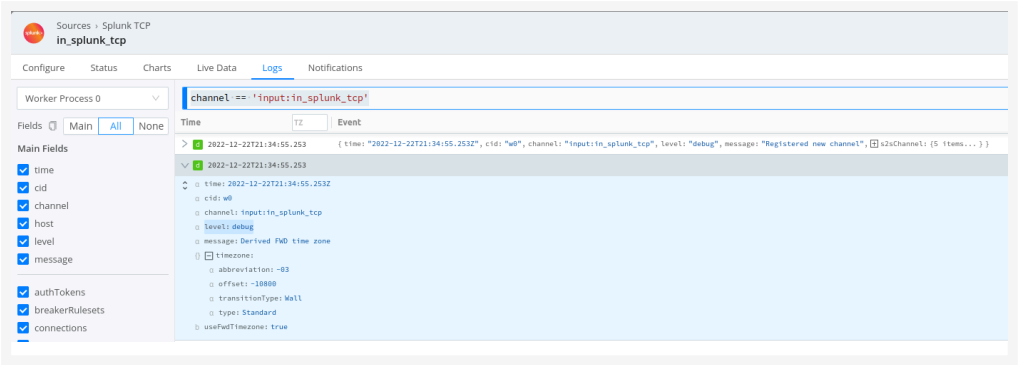
Under Advanced Settings, the **Use Universal Forwarder time zone** toggle mitigates cases where incoming events have timestamp strings but no time zone information. For example:

```
12-15-2022 14:57:22.080 WARN TcpOutputFd [1607 TcpOutEloop] - Connect to 172.17.0.1:9
```

Copy

This gap can be problematic, especially if the originating Universal Forwarder is in a different time zone from the processing Worker Node.

The `__TZ` field is the solution. Event Breakers use the `__TZ` field to derive time zone information, enabling them to set the `_time` field correctly. Derived time zone information will appear in Cribl Stream's own logs as shown below:



Time zone information in logs

Setting the Log Level for Connection Messages

When a Splunk forwarder connects to Cribl Stream, Cribl Stream logs the following message at the `debug` level: `Connection with forwarder has been established successfully.`

To see this message, set the `:forwarders` level to `debug`.

Each message contains details specific to the forwarder, such as the protocol, Splunk version, or remote host, to name a few.

In some situations, logging each incoming connection can produce many messages, which can make it hard to find other messages.

You can adjust the level of these connection messages. To do so, follow these steps:

1. Select **Settings** > **Global Settings** > **Logging** > **Levels**.
2. Search for the channel that logs the connection messages. It will have a name in the form `input:<source-id>:forwarders`. For example: `input:in_splunk_tcp:forwarders`.
3. Set the channel to the log level you prefer, such as `debug` or `silly`.
4. Click **Save** to save your setting.
5. Commit and deploy the change.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`
- `__ctrlFields`
- `__s2sVersion` - value can be either `v3` or `v4`
- `__source`
- `__srcIpPort`
- `__TZ` - see above

Configuring a Splunk Forwarder

To configure a Splunk forwarder (UF, HF) use the following sample outputs.conf stanzas:

outputs.conf (on-prem) | outputs.conf (Cribl Cloud)

```
[tcpout]
disabled = false
defaultGroup = cribl, <optional_clone_target_group>, ...
enableOldS2SProtocol = true

[tcpout:cribl]
server = [<cribl_ip>|<cribl_host>]:<port>, [<cribl_ip>|<cribl_host>]:<port>, ...
compressed = false
sendCookedData = true
# As of Splunk 6.5, using forceTimebasedAutoLB is no longer recommended. Ensure th
# forceTimebasedAutoLB = false
```

Copy

With a Cribl.Cloud Enterprise plan, generalize the above URL's `default.main` substring to `<groupName>.main` when sending to other Worker Groups.

If your use case requires compression, use useClientSSLCompression vs compressed on forwarder to compress the data stream.

Preventing Data Loss with v3

If you set **Max S2S version** to `v3` and are using Splunk 9.1.0 or later, Cribl recommends that you use the `enableOldS2SProtocol = true` setting shown above to avoid data loss. If you are working with `v3` and a Splunk version earlier than 9.1.0, you should use `negotiateProtocolLevel = 0`. Depending on your environment, enabling `negotiateProtocolLevel` with a non-`0` value could cause Cribl Stream to not accept data from the forwarder.

If you set **Max S2S version** to `v4`, these settings are not necessary. The Splunk receiver will detect which version is in use and automatically use the correct handler.

See Internal Fields for information on the `__s2sVersion` field.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Splunk Forwarder Performance Issues

If you encounter performance issues with a Splunk Forwarder, Cribl recommends increasing the number of parallel ingestion Pipelines or increasing forwarder throughput. You can experiment with either or both of these settings.

To increase the number of parallel ingestion Pipelines, adjust the setting for `parallelIngestionPipelines` in `server.conf`. Experiment with values ranging from `2-4`.

To adjust forwarder throughput, increase the `maxKBps` value in `limits.conf`. The default value is `256`. A value of `0` removes all throttling from the forwarder.

"Connection rejected" with a `reason` of "Too many connections"

The maximum number of active Splunk TCP connections has been exceeded per Worker Process. The default is 1000.

Recommendation

In the Splunk TCP input's Advanced Settings configuration, increase the Max Active Connections value, set it to 0 for unlimited, and/or increase the # of Worker Processes the Worker Node(s) are using.

8.1.9. Internal

You can use the following Sources to receive data from internal senders.

Source	Description
Cribl Internal	Capture Cribl Stream's internal logs and metrics and send them through Routes and Pipelines
Cribl HTTP	Receive data from peer Nodes managed by the same Leader when raw TCP traffic is not an option
Cribl TCP	Send data between Worker Nodes/Edge Nodes connected to the same Leader

8.1.9.1. Cribl Internal Source

The Cribl Internal Source enables you to capture and send Cribl Stream's own internal logs and metrics through Routes and Pipelines.

Type: **Internal** | TLS Support: **N/A** | Event Breaker Support: **No**

You can't configure custom Cribl Internal Sources.

Scope and Purpose

In Distributed mode, this Source's CriblLogs option can process internal logs only from Worker Processes. (Logs on the Leader remain on the Leader, because the Leader Node is not part of any processing path.)

In both Distributed and Single-instance mode, this Source's CriblLogs option omits API Process logs, meaning that it omits telemetry/license-validation traffic. You can, however, use a Script Collector to check for API Server (or Worker Group) events.

This Source's CriblMetrics option offers Cribl's most up-to-date and precise view of event throughput and transformation, aggregated every 2 seconds at the Worker Process level. The CriblLogs option is less granular - tracking logs that are written once per minute - so CriblLogs might fail to reflect Worker Process crashes or restarts.

In Cribl.Cloud, on Cribl-managed Worker Groups in Cribl.Cloud, the CriblLogs internal Source contains only logs related to Sources and Destinations.

Configure Cribl Internal Logs/Metrics as a Data Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, then choose **Select Existing**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want.
2. Next, select either the default `CriblLogs` or `CriblMetrics` to open the configuration modal.
3. Configure the **General Settings** for your selected Source.

- For Cribl Logs, configure the following:
  - **Enabled**: This duplicates the parent page's **Enabled** toggle. Toggle on to enable Cribl logs as a Source.
  - **Input ID**: This is prefilled with the default value `CriblLogs`, which cannot be changed.
  - **Description**: Optionally, enter a description.
- For Cribl Metrics, configure the following:
  - **Enabled**: This duplicates the parent page's **Enabled** toggle. Toggle on to enable Cribl metrics as a Source.
  - **Input ID**: This is prefilled with the default value `CriblMetrics`, which cannot be changed.
  - **Description**: Optionally, enter a description.

4. Next, you can configure the following **Optional Settings** for your selected Source:
   - For Cribl Logs, configure the following:
     - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
   - For Cribl Metrics, configure the following:
     - **Metric name prefix**: Enter an optional prefix that will be applied to metrics provided by Cribl Stream. The prefix defaults to `cribl.logstream.`
     - **Full fidelity**: Toggle off to reduce CPU load by excluding metrics for any fields listed in **Settings** > **Global** > **System** > **General Settings** > **Limits** > **Metrics** > **Disable field metrics**. See Controlling Metrics Volume for details. By default, these include:
       - `cribl.logstream.host.(in_bytes,in_events,out_bytes,out_events)`
       - `cribl.logstream.index.(in_bytes,in_events,out_bytes,out_events)`
       - `cribl.logstream.project.(in_bytes, in_events, out_bytes, out_events)`
       - `cribl.logstream.source.(in_bytes,in_events,out_bytes,out_events)`
       - `cribl.logstream.sourcetype.(in_bytes,in_events,out_bytes,out_events)`
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, configure any Processing and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Reporting Metrics Less Frequently

By default, Cribl Stream generates internal metrics every 2 seconds. To consume metrics at longer intervals, you can use or adapt the `cribl_metrics_rollup` Pipeline that ships with Cribl Stream.

Attach this Pipeline to your **Cribl Internal** Source as a pre-processing Pipeline. The Pipeline's **Rollup Metrics** Function has a default **Time Window** of 30 seconds, which you can adjust to a different granularity as needed. This provides a second lever to reduce granularity, in addition to the Full fidelity toggle described above.

Omitting `sourcetype`

You can easily drop the `sourcetype` attribute from metrics events, leaving only `event_sourcetype`. This will prevent duplicate `sourcetype` events from being routed to Destinations.

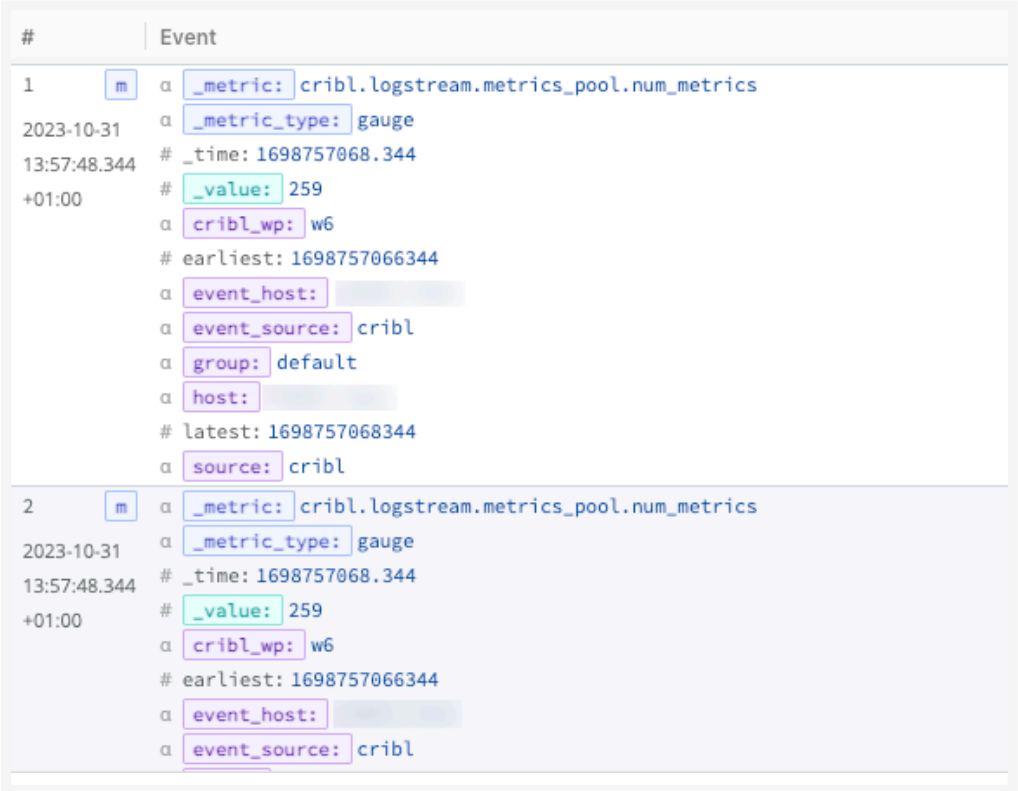
To do this: In the same `cribl_metrics_rollup` pre-processing Pipeline (or a clone) that you attach to your Source, enable the final Eval Function, which applies this **Filter** expression to remove the `sourcetype` field: `_metric && _metric.startsWith('cribl.logstream.sourcetype.')`

Duplicated Fields/Dimensions

The CriblMetrics Source operates on metrics that Worker Nodes report to their Leader Nodes. Typically included are `source` and `host` fields.

Sending metrics from this Source to Splunk is one common use case. Because Splunk might overwrite these two fields, the Source copies their values into new dimensions with added `event_` prefixes: `event_source` and `event_host`. This way, if Splunk does overwrite `source` and/or `host`, their original values remain intact in the new dimensions with `event_` prefixes.

Here's an example of how the added dimensions look in the **Live Capture** window:



Doubled fields

If you are not sending to a downstream service that overwrites `source` or `host` fields, you can use an appropriate Function to drop the added dimensions. You also have the option to suppress these fields entirely by specifying them in **Settings** > **Global** > **System** > **General Settings** > **Limits** > **Metrics** > **Disable field metrics**.

Internal Fields

The following fields will be added to all events/metrics:

- `source`: set to `cribl`.
- `host`: set to the hostname of the Cribl instance.

Use these fields to guide these events/metrics through Cribl Routes.

All Cribl internal fields are subject to change and modification. Cribl provides them to assist with analytics and diagnostics, but does not guarantee that they will remain available.

8.1.9.2. Cribl HTTP Source In Cribl Stream

The Cribl HTTP Source receives data from a Cribl HTTP Destination. Relaying data between a Cribl HTTP Destination and Cribl HTTP Source pair prevents double-billing: You won't incur any additional charges or credit usage after initial ingest to the first Worker Group (depending on your Leader setup - see Transfer Data Between Workspaces or Environments for more details).

It's common to send data from Cribl Edge to a Cribl Stream Worker Group using this kind of pairing.

Type: **Internal** | TLS Support: **YES** | Event Breaker Support: **No**

For guidance on when to choose this Source versus the Cribl TCP Source, see Interoperation Protocols.

Within Cribl Stream, the Cribl HTTP Source is available only in Distributed deployments. In single-instance mode, or for testing, you can use the Raw HTTP/S Source instead. However, this substitution will not facilitate sending all internal fields, as described below.

Looking for the Cribl HTTP Source in **Cribl Search**? See Add a Cribl HTTP Source for Cribl Search.

How It Works

You can use the Cribl HTTP Source to transfer data between Worker Nodes and Workers. If the Cribl HTTP Source receives data from its Cribl HTTP Destination counterpart on another Worker, you're billed for ingress only once - when Cribl first receives the data.

Using a Cribl HTTP Destination/Source pair, you're not charged for subsequent data sent to:

- Workers that share a Leader.
- Workers that don't share a Leader, but are configured in Cribl.Cloud Workspaces within the same Organization.
- Workers that do not share a Leader, but that do have the exact same on-prem license installed.

This use case is common in hybrid Cribl.Cloud deployments, where a customer-managed (hybrid) Worker Node sends data to a Worker in Cribl.Cloud for additional processing and routing to Destinations. However, the Cribl HTTP Destination/Source pair can similarly reduce your metered data ingress in other scenarios, such as on-prem Edge to on-prem Stream.

As one usage example, assume that you want to send data from one Node deployed on-prem, to another that is deployed in Cribl.Cloud. You could do the following:

- Create an on-prem File System Collector (or whatever Collector or Source is suitable) for the data you want to send to Cribl.Cloud.
- Create an on-prem Cribl HTTP Destination.
- Create a Cribl HTTP Source, on the target Stream Worker Group or Edge Fleet in Cribl.Cloud.
- For an on-prem Node configure a File System Collector to send data to the Cribl HTTP Destination, and from there to the Cribl HTTP Source in Cribl.Cloud.
- On Cribl-managed Nodes in Cribl.Cloud, make sure that TLS is either disabled on both the Cribl HTTP Destination and the Cribl HTTP Source it's sending data to, or enabled on both. Otherwise, no data will flow. On Cribl.Cloud instances, the Cribl HTTP Source ships with TLS enabled by default.

Configuration Requirements

The key points about configuring this architecture are:

- The Destination's **Cribl endpoint** field must point to the **Address** and **Port** you've configured on its peer Cribl HTTP Source(s).
- Cribl 3.5.4 was a breakpoint in Cribl HTTP Leader/Worker communications. Nodes running the Cribl HTTP Source on Cribl 3.5.4 and newer can send data only to Nodes running v.3.5.4 and newer. Nodes running the Cribl HTTP Source on Cribl 3.5.3 and older can send data only to Nodes running v.3.5.3 and older.

Configure the Cribl HTTP Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Cribl HTTP Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the address to bind on. Defaults to `0.0.0.0` (all addresses).
    - **Port**: Enter the port number to listen on, for example, `10200`.
3. Next, you can configure the following **Optional Settings**:

- **Tags**: Optionally, add tags that you can use for filtering and grouping in the Cribl Stream UI. Use a tab or hard return between (arbitrary) tag names. These tags aren't added to processed events.

4. Optionally, configure any TLS settings, Persistent Queue, Processing, Advanced settings and Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled Default is toggled off. When toggled on, exposes this section's remaining fields.

Certificate name: Select a predefined certificate from the drop-down. A **Create** button is available to create a new certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on, exposes these two additional fields:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled off.
- **Common name**: Regex matching subject common names in peer certificates allowed to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

Enable Persistent Queue: Defaults to toggled off. When toggled on:

Mode: Choose a mode from the drop-down:

- With `Smart` mode, PQ will write events to the filesystem only when it detects backpressure from the processing engine.
- With `Always On` mode, PQ will always write events directly to the queue before forwarding them to the processing engine.

Buffer size limit: Maximum number of events to hold in-memory before dumping them to disk.

Commit frequency: Number of events to send before committing that Stream has read them.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, Cribl Stream stops queueing and applies the fallback **Queue-full behavior**. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this value, Cribl Stream will append `/<worker-id>/<output-id>`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Auth Token

This section is only available for Cribl.Cloud environments. It is intended for use when migrating data from an on-prem environment to a Cribl.Cloud environment.

If you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from on-prem Worker Nodes to their counterpart Cribl.Cloud Worker Nodes. See Migrate Data from On-Prem to Cribl.Cloud for implementation details.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; etc.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

The Cribl HTTP Source (and the Cribl TCP Source) treat internal fields differently than other Sources do. That's because of the difference in the way that incoming data originates.

Other Sources ingest data that's not coming from Cribl Edge or Stream, meaning that no Cribl internal fields can be present in that data when it arrives at the Source, and the Source is free to add internal fields without clobbering (overwriting) anything that existed already.

By contrast, the Cribl HTTP Source and the Cribl TCP Source ingest data that's coming from a Cribl HTTP or Cribl TCP Destination. That data **can** contain internal fields when it arrives at the Source. This means that if the Source adds internal fields, those could potentially clobber what existed before.

To avoid this problem, the Cribl HTTP Source and the Cribl TCP Source add a unique `__forwardedAttrs` (i.e., "forwarded attributes") field. The nested structure of the `__forwardedAttrs` field contains any of the following fields that are present in the arriving data:

Internal Fields
`__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
`__inputId`

Internal Fields
`__outputId`
`fleet/ group`
`__srcIpPort` - See details below.

Other Fields
`cribl_breaker`
`cribl_pipe`

These fields are **copied** into `__forwardedAttrs`, not moved there. As the data (apart from `__forwardedAttrs`) moves through the Source and any Pipelines, the values of these fields can be overwritten. But the copies of these fields in `__forwardedAttrs` remain unchanged, so you can retrieve them as necessary.

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the HTTP client sending data to this Source.

When any proxies (including load balancers) lie between the HTTP client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Troubleshooting

Dropping request because token invalid",“authToken”: “Bas…Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.9.3. Cribl TCP Source

The Cribl TCP Source receives data from a Cribl TCP Destination. Relaying data between a Cribl TCP Destination and Cribl TCP Source pair prevents double-billing: You won't incur any additional charges or credit usage after initial ingest to the first Worker Group (depending on your Leader setup - see Transfer Data Between Workspaces or Environments for more details).

It's common to send data from Cribl Edge to a Cribl Stream Worker Group using this kind of pairing.

Type: **Internal** | TLS Support: **YES** | Event Breaker Support: **No**

For guidance on when to choose this Source versus the Cribl HTTP Source, see Interoperation Protocols.

Within Cribl Stream, the Cribl TCP Source is available only in Distributed deployments. In single-instance mode, or for testing, you can use the TCP JSON Source instead. However, this substitution will not facilitate sending all internal fields, as described below.

How It Works

You can use the Cribl TCP Source to transfer data between Worker Nodes and Workers. If the Cribl TCP Source receives data from its Cribl TCP Destination counterpart on another Worker, you're billed for ingress only once - when Cribl first receives the data.

Using a Cribl HTTP Destination/Source pair, you're not charged for subsequent data sent to:

- Workers that share a Leader.
- Workers that don't share a Leader, but are configured in Cribl.Cloud Workspaces within the same Organization.
- Workers that do not share a Leader, but that do have the exact same on-prem license installed.

This use case is common in hybrid Cribl.Cloud deployments, where a customer-managed (hybrid) Worker Node sends data to a Worker in Cribl.Cloud for additional processing and routing to Destinations. However, the Cribl TCP Destination/Source pair can similarly reduce your metered data ingress in other scenarios, such as on-prem Cribl Edge to on-prem Cribl Stream.

As one usage example, assume that you want to send data from one Node deployed on-prem, to another that is deployed in Cribl.Cloud. You could do the following:

- Create an on-prem File System Collector (or whatever Collector or Source is suitable) for the data you want to send to Cribl.Cloud.
- Create an on-prem Cribl TCP Destination.

- Create a Cribl TCP Source, on the target Cribl Stream Worker Group or Cribl Edge Fleet in Cribl.Cloud.
- For an on-prem Node, configure a File System Collector to send data to the Cribl TCP Destination, and from there to the Cribl TCP Source in Cribl.Cloud.
- On Cribl-managed Nodes in Cribl.Cloud, make sure that TLS is either disabled on both the Cribl TCP Destination and the Cribl TCP Source it's sending data to, or enabled on both. Otherwise, no data will flow. On Cribl.Cloud instances, the Cribl TCP Source ships with TLS enabled by default.

Configuration Requirements

The key points about configuring this architecture are:

- When you configure the Cribl TCP Destination, its **Address** and **Port** values must point to the **Address** and **Port** you've configured on the Cribl TCP Source.
- Cribl 3.5.4 was a breakpoint in Cribl TCP Leader/Worker communications. Nodes running the Cribl TCP Source on Cribl 3.5.4 and newer can send data only to Nodes running v.3.5.4 and newer. Nodes running the Cribl TCP Source on Cribl 3.5.3 and older can send data only to Nodes running v.3.5.3 and older.

Configure the Cribl TCP Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Cribl TCP Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter hostname/IP to listen for TCP JSON data. For example, `localhost` or `0.0.0.0`.
    - **Port**: Enter the port number to listen on, for example, `10300`.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use for filtering and grouping in the Cribl Stream UI. Use a tab or hard return between (arbitrary) tag names. These tags aren't added to processed events.

4. Optionally, configure any TLS settings, Persistent Queue, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled off.
- **Common name**: Regex matching subject common names in peer certificates allowed to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

Enable persistent queue: Defaults to toggled off. When toggled on:

Mode: Choose a mode from the drop-down:

- With `Smart` mode, PQ will write events to the filesystem only when it detects backpressure from the processing engine.

- With `Always On` mode, PQ will always write events directly to the queue before forwarding them to the processing engine.

Buffer size limit: Maximum number of events to hold in memory before dumping them to disk. Defaults to `1000`.

Commit frequency: Number of events to send before committing that Cribl Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, and so on. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, Cribl Stream stops queueing and applies the fallback **Queue-full behavior**. Enter a numeral with units of KB, MB, and so on.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this value, Cribl Stream will append `/<worker-id>/<output-id>`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Auth Token

This section is only available for Cribl.Cloud environments. It is intended for use when migrating data from an on-prem environment to a Cribl.Cloud environment.

If you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from on-prem Worker Nodes to their counterpart Cribl.Cloud Worker Nodes. See Migrate Data from On-Prem to Cribl.Cloud for implementation details.

Advanced Settings

Enable proxy protocol: Toggle on if the connection is proxied by a device that supports proxy protocol v1 or v2.

Enable load balancing (Cribl Stream only): Toggle on to distribute the data from incoming TCP connections across multiple Worker Processes. This improves performance compared to the default of having all data from an incoming connection processed on a single Worker Process.

- This spins up an extra Worker Process (named `wLB` in **Settings** > **Processes**) on the Worker Node, which handles splitting data from incoming TCP connections across all other Worker Processes.
- When enabled, the option to include common fields in the JSON header is not supported. This means fields specified under `fields` in the JSON header are not automatically added to all events.

Load balancing is available in Cribl Stream single-instance and Distributed deployments when a TCP port is configured.

Active connection limit: Maximum number of active connections allowed per Worker Process. Defaults to `1000`. Set a lower value if connection storms are causing the Source to hang. Set to `0` for unlimited connections.

Socket idle timeout (seconds): The duration that Cribl Stream will wait for activity on an idle TCP socket before closing the connection. Disabled when set to `0`, the default.

Forced socket termination timeout (seconds): The extra time the server waits before forcibly closing a socket that has been idle (**TCP socket idle timeout**) or exceeded its maximum lifespan (**TCP socket max lifespan**) but has not yet properly closed. This prevents resource leaks caused by unresponsive clients or network issues. Configure based on network latency and client behavior. Default: `30` seconds. Set to `0` to disable.

Socket max lifespan (seconds): The duration that a socket is allowed to remain open, regardless of activity. This setting prevents resource exhaustion (such as TCP pinning) by limiting the lifespan of connections.

Configure based on expected connection durations and resource availability. Disabled when set to `0`, the default.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the configuration will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

The Cribl TCP Source (and the Cribl HTTP Source) treat internal fields differently than other Sources do. That's because of the difference in the way that incoming data originates.

Other Sources ingest data that's not coming from Cribl Edge or Cribl Stream, meaning that no Cribl internal fields can be present in that data when it arrives at the Source, and the Source is free to add internal fields without clobbering (overwriting) anything that existed already.

By contrast, the Cribl TCP Source and the Cribl HTTP Source ingest data that's coming from a Cribl TCP or Cribl HTTP Destination. That data **can** contain internal fields when it arrives at the Source. This means that if the Source adds internal fields, those could potentially clobber what existed before.

To avoid this problem, the Cribl TCP Source and the Cribl HTTP Source add a unique `__forwardedAttrs` (that is, "forwarded attributes") field. The nested structure of the `__forwardedAttrs` field contains any of the following fields that are present in the arriving data:

Internal Fields
`__srcIpPort`
`__inputId`
`__outputId`
`fleet / group`

Other Fields
`cribl_breaker`
`cribl_pipe`

These fields are **copied** into `__forwardedAttrs`, not moved there. As the data (apart from `__forwardedAttrs`) moves through the Source and any Pipelines, the values of these fields can be overwritten. But the copies of these fields in `__forwardedAttrs` remain unchanged, so you can retrieve them as necessary.

Periodic Logging

Cribl Stream logs metrics about incoming requests and ingested events once per minute.

These logs are stored in the `metrics.log` file. To view them in the UI, open the Source's **Logs** tab and choose **Worker Process X Metrics** from the drop-down, where **X** is the desired Worker Process.

This kind of periodic logging helps you determine whether a Source is in fact still healthy even when no data is

8.1.10. System

You can use the following Sources to receive data from internal senders.

Source	Description
Datagen	Generate data from datagen files
Exec	Execute a command periodically, and collect its stdout output
File Monitor	Collect log files and generate events from the file content
Journal Files	Collects data from systemd's journald service
System Metrics	Collect metrics from a host, and populate dashboards
System State	Collects the system's current state on a configurable schedule, relaying corresponding events to downstream systems.

8.1.10.1. Datagen Source

Cribl Stream supports generating data from datagen files, as detailed in Using Datagens. When a datagen is enabled, each Worker Process uses the specified data generator file to generate events. These events proceed through Routes and Pipelines, or through a QuickConnect configuration, to configured Destinations. Whichever Worker Process generated an event from the file will also send the same event.

Type: **System** | TLS Support: **N/A** | Event Breaker Support: **No**

Configure Cribl Stream to Generate Sample Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Datagens**: List of datagens. For details, see Datagen Fields below.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use for filtering and grouping in the Cribl Stream UI. Use a tab or hard return between (arbitrary) tag names. These tags aren't added to processed events.
4. Optionally, configure any Processing and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Datagen Fields

To configure the list of datagens, include the following:

- **Data generator file**: Name of the datagen file.

- **Events per second per Worker Node**: Maximum number of events to generate per second, per Worker Node/Edge Node. Defaults to `10`.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These “meta” fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`

8.1.10.2. Exec Source

The Exec Source enables you to periodically execute a command and collect its `stdout` output. This is typically used in cases when Cribl Stream cannot accomplish collection with native Collectors or other Sources.

Type: **System** | TLS Support: **N/A** | Event Breaker Support: **Yes**

This Source is available in Cribl.Cloud on customer-managed hybrid Workers, but not on Cribl-managed Workers in Cribl.Cloud.

This Source allows you to run **almost anything** on the host system. Make sure you understand the security and other impacts of commands you plan to execute, before proceeding. To disable it on your Worker Nodes, see Disable the Exec Source below.

Especially for monitoring or polling, you'll need to receive the command or script's output periodically. For this reason, the Exec Source lets you specify when the command or script should run, by time interval or by cron-style schedule.

Here are a few examples of what you can run from an Exec Source:

- Database queries.
- Custom commands to poll databases or apps.
- `ping` to check latency.
- `ps -ax` to get a list of running processes.
- `SNMP GET` requests to query an SNMP agent about network entities.
- `netstat -natp` to get lists of sockets and TCP ports, and what they're doing.
- `mtr -c 1 --report --json 8.8.8.8` to run a traceroute on a location (here, `8.8.8.8`), and packaging up a report in JSON format.

Configure an Exec Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.

    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Defaults to toggled on.
    - **Input ID**: Enter a unique name to identify this Exec Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Command**: Command to execute; supports Bourne shell syntax.
    - **Schedule type**: Use the buttons to select either `Interval` or `Cron`. For details, see Schedule Type below.
3. Next, you can configure the following **Optional Settings**:
    - **Script**: Optional multiline script to send to the defined command. The content of this field will be piped into the `stdin` of the command. The `stdin` stream is closed after adding the script. See Run Custom Scripts for more information.
    - **Retry limit**: Maximum number of retry attempts in the event that the command fails.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Schedule Type

Customize the behavior in the corresponding field below the button.

- **Interval**: Specify how often, in seconds, the command should run. Defaults to `60`.
- **Schedule**: Enter a cron expression. (You enter the cron expression in UTC time, but the resulting **Estimated Schedule** displays in local time.)

Processing Settings

Event Breakers

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

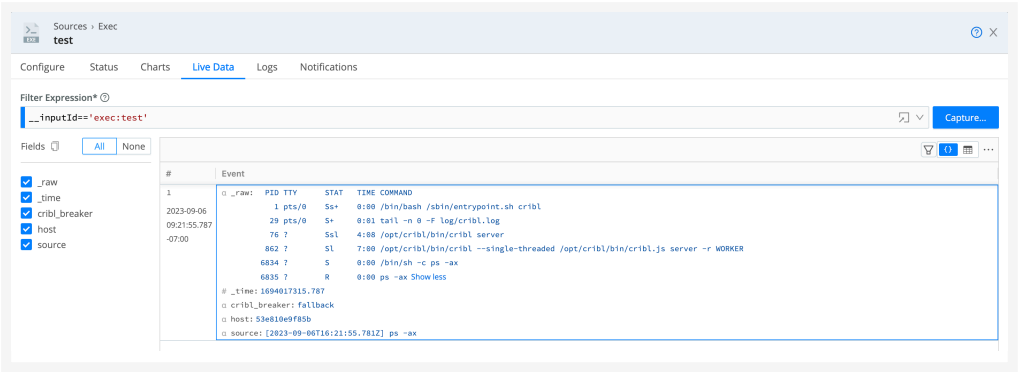
Internal Fields

Cribl Stream uses internal fields to assist in forwarding data to a Destination.

You can find the following event fields on the **Live Data** tab of the selected Exec Source, including the hostname and source of the event:

- `_raw`
- `_time`

- `cribl_breaker`
- `host`
- `source`



Event fields

Troubleshooting

If a command isn't running as expected, the **Logs** tab can help you identify issues by displaying the command executed, its elapsed time, and its exit code.

Event fields

Run Custom Scripts

You can use the Exec Source to run custom multiline scripts, for example in Bash or Python. To do so, enter the command in the **Command** field and the script contents in the **Script** field.

For example, to run a Bash script, enter:

Command: `bash -`

Script:

```
#!/usr/bin/env bash
USER_NAME=$(whoami)
DATE=$(date)
echo "Hi $USER_NAME, today is $DATE"
```

To run a Python script, enter:

Command: `python3 -`

Script:

```
#!/usr/bin/env python3
import getpass
from datetime import datetime
user_name = getpass.getuser()
date_now = datetime.now()
print(f"Hi {user_name}, today is {date_now}")
```

Execute Windows PowerShell Commands

On a Windows Edge Node, the Exec Source command is executed from a Windows command prompt, not from a PowerShell prompt. However, you can still run PowerShell commands in it with some modification.

> The execution of PowerShell is resource-intensive and should be used with caution. On limited resource systems (e.g., domain controllers) it is best to run Exec with PowerShell carefully as the Source uses the host's CPU.

To run a PowerShell command in the Exec Source, you need to prepend the **Command** field with `powershell`, for example:

PowerShell Prompt Command	Cribl Exec Source configuration
`Get-Process`	`powershell Get-Process`
`# cd c:\mydir` `# .\myps-script.ps1`	`powershell c:\mydir\myps-script.ps1`

You can handle a PowerShell command with parameters by placing the command and the parameters in double quotes (`""`), for example:

PowerShell Prompt Command	Cribl Exec Source configuration
`Get-CimInstance win32_process | select ProcessId, Name, commandline, HandleCount, WorkingSetSize, VirtualSize`	`powershell [-command] "Get-CimInstance win32_process | select ProcessId, Name, commandline, HandleCount, WorkingSetSize, VirtualSize"`

(Include the `-command` parameter if your PowerShell version requires it.)

Disable the Exec Source on a Node

To disable the Exec Source on a specific Worker Node, set the `CRIBL_NOEXEC` environment variable. This will override any configuration pushed from the Leader Node, ensuring the Exec Source remains disabled on that particular node.

The `CRIBL_NOEXEC` environment variable must be set outside of the Cribl product to disable the Exec Source. This variable cannot be modified within the product's interface.

How to Disable the Exec Source on Different Platforms

To disable the Exec Source, follow the platform-specific instructions below. Once you've set the `CRIBL_NOEXEC` environment variable, the Exec Source will be disabled. To re-enable it, remove or unset the variable.

Disable on Linux Platform

To disable the Exec Source on a Linux system running as a Systemd service, edit the appropriate service definition file:

- Cribl Edge: `/etc/systemd/system/(cribl|cribl-edge).service`
- Cribl Stream: `/etc/systemd/system/cribl.service`

To ensure the `CRIBL_NOEXEC` environment variable is set when the Cribl Edge service starts, add `CRIBL_NOEXEC=1` to the service's configuration. If the service is currently running, a restart is required for the change to take effect.

Disable on Docker

To disable the Exec Source on Docker, modify the `docker run` command (or equivalent) to include the `-e CRIBL_NOEXEC=1` flag.

Disable on Kubernetes/Helm

To disable the Exec Source on Kubernetes, add the `--set env.CRIBL_NOEXEC=1` flag to your `Helm install` command or modify the `values.yml` file to include this setting.

Disable on Windows (For Cribl Edge)

To add the `CRIBL_NOEXEC` environment variable to Cribl Edge on Windows, use the command prompt to execute the following command:

```
setx CRIBL_NOEXEC "1" /M
```

This command sets the environment variable for the current user account. The Cribl Edge service runs under `LocalSystem` user account by default.

8.1.10.3. File Monitor Source

The File Monitor Source generates events from text files, compressed, archived, and (some) binary log files, based on lines and records extracted from the content.

Type: **System** | TLS Support: **N/A** | Event Breaker Support: **Yes**

This Source is available in Cribl.Cloud on customer-managed hybrid Workers, but not on Cribl-managed Workers in Cribl.Cloud.

As of v.4.2.x, this Source can process the following file formats:

- **Compressed/archived**: `zip`, `gzip`, `zstd`, and `tar`. Cribl Stream only supports DEFLATE compression for `zip` files.
- **UTF-8**: `text`.
- **Non-identifiable binary**: Includes all file formats that aren't in the Cribl Edge-supported compressed, archived, and text categories.

Binary files are broken into base64-encoded chunks and streamed bypassing Event Breakers. Text files are processed normally through Event Breakers. (Before v.4.2.x, the File Monitor Source could handle only text files.) Also as of v.4.2.x, the File Monitor Source no longer excludes `*.gz` files by default.

File Monitoring and One-Time Ingestion

Cribl Edge offers two options for ingesting file data:

- **File Monitor Source**: This is ideal for ongoing log file monitoring. By default it will prioritize processing the latest data in newly discovered files upon startup ("Collect from end"). Subsequently, it automatically monitors and ingests new content as it's added.

To automatically discover and ingest new log files, Cribl Edge requires privileged access for file scanning and reading. For alternative approaches, see Running Edge as an Unprivileged User.

- **One-time File Ingest**: This feature complements the File Monitor by allowing a one-time ingest of any file. It's useful for quickly processing specific files or initial data loads. If **Explore** is enabled, as it might be restricted in some environments for security reasons.

Discovering and Filtering Files to Monitor

To produce its initial list of files to monitor, the File Monitor Source runs a discovery procedure at a configurable **Polling interval**. The Source then applies a **Filename allowlist** to filter the initial list down into its final form. Then, for each file on the list, the Source compares current state with previously-stored state. This comparison determines whether the File Monitor Source will actually watch a given file for a given polling interval, or just ignore the file.

In the simplest case, the Source discovers a file for which it has no stored state. This means that the file has just been created, and needs to be monitored. See the Examples for other possibilities, along with a description of the **Status** tab, which displays state information for all files being monitored.

How does the File Monitor Source discover files in the first place? You have the choice of two **Discovery Modes**: **Auto** or **Manual**.

- In **Auto** mode, the Source automatically discovers files that running processes have open for writing. Auto mode collects logs from the discovered active files that match the allowlist.
  Auto mode is useful to detect which files are being written to. For example, if you enter an allowlist of `*log`, Auto mode will find all the active logs within that directory, no matter what's writing to it, and it will exclude any rotated logs.

Cribl Edge currently supports Auto mode only when running on Linux, not on Windows or macOS. Windows and macOS deployments require using the Manual option. For a Windows-specific configuration, see Collect Logs with File Monitor.

- In **Manual** mode, the Source discovers files based on the specified directory and depth.
  Manual mode is often used to collect logs in a folder where the process that creates them rotates them when they get large. For example, if you specify a path of `/var/log` and an allowlist of `*/messages*`, Manual mode will capture the current `/var/log/messages`, as well as older logs from `/var/log/messages*`.
  Manual is the default discovery mode for new File Monitor Sources.

If you are using a tool like `rsync` to transfer files in chunks, the File Monitor Source might start collecting files before the transfer is complete. To prevent this, configure your tool to transfer files serially. (For example, use: `rsync --append`.)

Configure a File Monitor Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:

- To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
- To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.

2. In the **New Source** modal, configure the following under **General Settings**:
   - **Enabled**: Toggle on to enable the Source.
   - **Input ID**: Enter a unique name to identify this File Monitor Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
   - **Description**: Optionally, enter a description.
   - **Discovery Mode**: Select either **Auto** or **Manual**. See Discovering and Filtering Files to Monitor for details on how each mode works.
     - **Auto**: Tells the Source to automatically discover files that running processes have open for writing.
     - **Manual**: Tells the Source to discover files within a single directory that you specify as the **Search path**, down to the **Max depth**.
       - **Search path**: Enter an absolute path (no wildcards). Defaults to the `/var/log` directory.
       - **Max depth**: Controls how many levels deep the Source will search within the **Search path**. If you leave the **Max depth** field empty, the Source will search subdirectories, and their subdirectories, and so on, without limit. If you specify `0`, the Source will discover only the top-level files within the **Search path**. For `1`, the Source will discover files one level down.
       - **Suppress errors when search path does not exist**: Toggle on to hide `Path does not exist` errors when a directory doesn't exist. When enabled, missing directories will not affect the File Monitor Source health status. This is useful when you know the search path contains nonexistent directories for some hosts, and you don't want to see the error messages in the File Monitor status and logs.
       - **Delete files**: Whether to delete the source file after it has been idle for the specified **Idle timeout** period. This toggle is under Advanced Settings.

3. Next, you can configure the following **Optional Settings**:
   - **Polling interval**: Interval of time in seconds to run the discovery procedure. Defaults to `10` seconds.
   - **Filename allowlist**: Controls which files the Source discovers and monitors. Defaults to `*/log/*` and `*log`. You can use wildcards (`*`) to match any sequence of characters and the exclamation mark (`!`) for negation. These patterns are matched against the full path of the file. For example:

- Wildcard (`*`): `*DhcpSrvLog-*.log` collects file `/var/log/DhcpSrvLog-05-22-2025.log`.
- Negation `!`: `!*cribl*access.log` prevents the Source from discovering Cribl Stream's own log files.

If you're using **Manual** discovery mode, always think through how your **Search path** and **Filename allowlist** settings interact. If you're not careful, you can inadvertently create overlapping monitor groups that deliver duplicates of some files. See Using Allowlists Effectively for details.

- **Apply filename allowlist internal to archive files**: Toggle on if you want the filename allowlist to apply to files inside archives such as `tar` or `zip`.
- **Minimum age duration**: Optionally, specify a minimum age of files to monitor. This Source will only start ingesting a file once its modification time exceeds the configured duration. Enter a duration in a format like 30s, 15m, 1h. Defaults to an empty field, which applies no age filters.
- **Maximum age duration**: Optionally, specify a maximum age of files to monitor. This Source will filter events with timestamps newer than the configured duration. (Where files don't have a parsable timestamp, it will set their events' timestamp to `Now`.) Enter a duration in a format like `30s`, `4h`, `3d`, or `1w`. Defaults to an empty field, which applies no age filters.
- **Check file modification times**: Toggle on if you want this Source to skip files with modification times older than the configured **Maximum age duration**. When you set an **Maximum age duration** threshold, the Source will open every file and read through all its contents to seek a timestamp. For best performance, also toggle **Check file modification times** on skip older files.
- **Collect from end**: When toggled on (default) this Source prioritizes processing the latest data by jumping to the end of newly discovered files on Cribl Stream's startup. This allows focusing on new content in existing log files. However, this behavior only applies during the first encounter with a file. For existing files after the initial scan and all subsequent discoveries, Cribl Stream will switch to reading from the beginning to capture ongoing updates.
- **Enable binary files**: Toggle on if you want this Source to report the file as binary and stream it in Base64-encoded chunks. By default, the Source ignores binary (non-text) files such as JPEG images, MP3 audio files, or some binary data files.
- **Force text format**: Toggle on if you want this Source to display ingested data as text in the event (when the data isn't in a compressed or archived file). This option is helpful if you have files that contain binary mixed with text where the data should be streamed as all text (for example, AuditD logs can contain this type of data).
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Processing Settings

Event Breakers

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec.), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Idle timeout: Time, in seconds, before an idle file is closed. Defaults to `300` sec. (5 minutes).

Hash length: How many file header bytes to use in a hash for identifying a file uniquely. Defaults to `256` bytes. For details, see Configuring Hash Lengths.

Delete files: Default is toggled off. When toggled on: Cribl Stream will delete the source file after it has been idle for the specified **Idle timeout** period. This setting is only available in `Manual` mode. In the confirmation modal, select **Enable** to proceed. Files that fail to delete will be reported as errors on the **Status** tab for up to one hour.

Files might not be deleted if:

- The Cribl Service doesn't have proper permissions to access and delete the files.
- They are currently in use by other applications on Windows.
- They are being moved or renamed while Cribl Stream is collecting them.
- Their filenames are not on the allowlist.

To make sure files are deleted properly:

- Verify and grant necessary permissions to the Cribl Service. For details, see Configure Cribl Stream for Non-Root User and Running Edge as an Unprivileged User.
- Consider access patterns and how files are being used.
- Adjust system settings if necessary (such as, file sharing, antivirus exclusions).

Salt file hash: Default is toggled off. Toggle on to salt the hash with the source file name. Use this option for files with the same header (for example, CSV files) to ensure that each has a different hash and can be ingested correctly. A file will get a new hash and be re-ingested if you move or rename it. Additionally, toggling this setting on after the Source has started operating will cause all files to be re-ingested as well.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

This Source defaults to QuickConnect.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are not part of an event, but they are accessible, and Functions can use them to make processing decisions.

For the File Monitor Source, you can use internal fields to enrich events with container and host metadata. The following internal fields are available:

- `__baseFilename`

- `__source / source`
- `__raw`
- `__inputId`
- `container_id`
- `container_path`
- `host_path`
- `__unresolved_source` (This internal field represents the unresolved path for the symbolic link configured in the File Monitor Source.)

Monitoring Renamed Files' State

The File Monitor Source uses a simple scheme to find "backlog" files corresponding to a matching file: Tell it to tail `foo.log`, and it will look for `foo.log.[0-9]` and scrape those, too.

This Source keeps hashes that correspond to the start point within the file, and to the last-read point. This way, if Cribl Stream is stopped, files are rotated, and Cribl Stream is restarted, the File Monitor Source can find the resume point in those backlog files.

Similarly, if you rename the file **within the same backlog scheme** (for example, `foo.log` to `foo.log.0`), the File Monitor Source can resume at the right place. However, if you renamed `foo.log` to `bar.log` (a deviation from the expected naming scheme), this Source could not find its resume point.

File Tracking with Hashes

The File Monitor Source doesn't track state using filenames. Instead, it uses hashes of the head and tail of the file to identify it later so it can resume where it left off.

To make sure hashes are generated correctly to ingest the files you want, follow these better practices.

Configure Hash Lengths

For files with identical first lines, configure the hash length to be longer than the first line to ensure that the header includes something unique. It is common for a batch of `CSV` files or `IIS` logs to have identical first lines listing columns that appear in subsequent lines.

If you don't adjust the hash length, all files with headers larger than `256 bytes` will appear to File Monitor as one file. As long as the hash length is greater than the header length, and the subsequent lines include something unique, the File Monitor Source will identify them as different files.

Salt the Hashes

If you want to ingest files with identical first lines (such as CSV files), and configuring hash length does not yield satisfying results, you can ensure different hashes for the files by toggling the **Salt file hash** setting. See Salt file hash for more details.

Avoid Race Conditions

The hashing logic isn't instantaneous, so it's possible for files with the same header and the same hash to be both collected. For example, quickly copying the same example file to separate `eg1.log` and `eg2.log` files in the Source's path will usually result in both files being collected even though they share the same hash. However, copying it to `eg3.log` shortly (not immediately) after will result in that third file being ignored because its hash matches an existing entry in the state-store.

Don't Ingest Multiple Copies of the Same File

Creating copies of the same file in the Source's path will not result in the file being ingested multiple times. The head-hashing logic used by the Source to determine what it's seen before and how far it's collected already prevents you from successfully testing multiple copies of the same.

Don't Point Multiple File Monitor Sources at the Same File

Multiple File Monitor Sources set to collect the same file will result in duplicate events. Sources run independently and don't coordinate with each other.

Using Allowlists Effectively

Done properly, file monitoring collects exactly the files you need to see, without duplication. You want to avoid inadvertently creating overlapping monitor groups that deliver duplicates, because this can "run the meter" twice for affected files, potentially causing issues with data ingestion volume limits.

The key is knowing how to craft allowlists.

Crafting Allowlists: Principles

To get the results you want from file monitoring, always apply the following principles.

Write allowlists to explicitly include the full path of files you want to collect.

The search path itself is **not** excluded from the match. For example, if you want to collect the file `/var/log/messages` when you've set your search path to `/var/log`, your filename allowlist needs to be either `/var/log/messages`, `/var/*/messages`, or `*/log/messages`.

Exclusions (`!`) also need to address the full file path. For example, if you want to collect everything from `/var/log/*` **except for** `/var/log/apache/*`, your allowlist must include one of the following:

- `!/var/log/apache/*`
- `!*/apache/*`
- `!*/log/apache/*`

Wildcards (`*`) match **any** part of the file path:

- `*log` matches `/var/foo/bar/myfile.log`.
- `*.log` matches anything ending in `.log`.
- `/var/log/*` matches anything in `/var/log`, subject to the **Max Depth** setting.

Put explicit matches (and exclusions) at the beginning of the allowlist, ahead of wildcard matches.

For example, an allowlist of `/var/log/*` followed by `!/var/log/apache/*` will fail to exclude the file `/var/log/apache/foo` because that file matches the first allowlist entry.

In allowlists, order matters!

Be as specific as possible, whenever possible, to prevent accidental matches.

Each of the following examples shows how to avoid matching unwanted files within the directories that you're monitoring.

To collect `/var/log/boot.log` (and older versions of `boot.log.*`) as well as `/var/log/messages`:

Search path	Filename allowlist
`/var/log`	`/var/log/boot.log`, `/var/log/messages`

To collect everything in `/var/log` while excluding `/var/log/apache/*`:

Search path	Filename allowlist
`/var/log`	`!/var/log/apache/*`, `/var/log/*`

Examples of Allowlists

Below are typical scenarios where good allowlist technique will save you from duplication headaches.

Example: Excluding the System's Own Logs

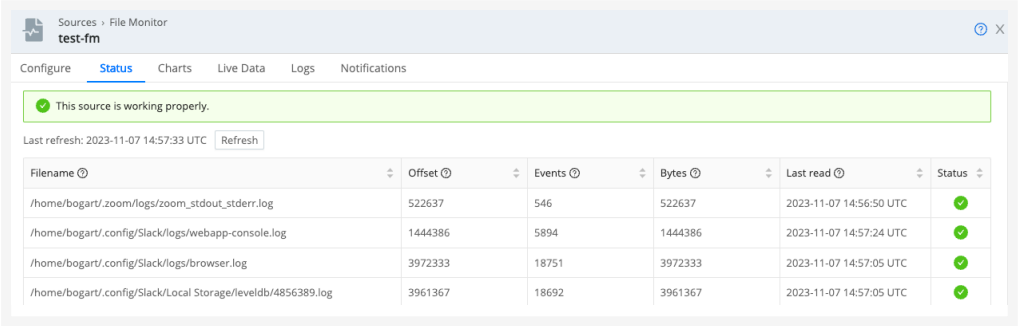
Recall that you'll often create a set of File Monitor Sources, each of which monitors one file. This is **not** the right approach in an example like this one, where **Discovery Mode** is set to **Auto**. Using **Auto** mode in a file monitor along with literally any other file monitor can cause unintended, duplicate collection. Use **Auto** mode carefully!

Here's the example:

Suppose you added a File Monitor Source on a Linux machine, set your **Discovery Mode** to **Auto**, and specified your home directory as the **Search path** (for example, `/home/bogart/`).

You do not want to monitor Cribl Stream's own log files, or any log files generated by Chrome, the web browser you're using. To exclude them, you add `!*cribl*access.log` and `!*chrome*` to your **Allowlist**. You **do** want to monitor any other log files that the Source can discover, so you also add `*log`.

After awhile, the Status tab looks like this:



The Status tab

Example: Fixing Overlapping Monitor Groups

Here's an example of the kind of overlapping monitor groups you should avoid:

Search path	Max depth	Filename allowlist
`/var`	(none)	`*/log/*`
`/var/log/apache`	`0`	`*`

The above configuration would collect all the files in `/var/log/apache/` twice. A better solution would be the following:

Search path	Max depth	Filename allowlist
`/var/log`	(none)	`!*/apache/*`
`/var/log/apache`	`0`	`*`

Troubleshooting

Common mistakes to avoid when exploring the File Monitor Source include the following.

Don’t Manually Edit Ingested Files

Do not test the File Monitor Source by editing and saving files using a text editor (such as vi or Notepad). Under the hood, these editors operate on a temporary file that they move into place when saved. As a result, the Source sees files get replaced and moved rather than appended to. This confuses the Source’s logic that assumes logs are appended to the existing file.

Instead, to test the Fole Monitor Source, run the command below to append logs to files: `echo "{ \"time\":\"$(date -Is -u)\", \"message\":\"...\" }" >> eg.log`

You end up with lines like this in the logs:

```
{ "time":"2023-09-19T20:08:42+00:00", "message":"..." }
```

Test File Ingestion with Event Breakers

To make sure that incoming files are split into individual events as expected, test sample file content with Event Breakers. When you configure the Event Breaker Ruleset, configure the timestamp correctly and verify that it works. The Event Breaker will default to “Now” if it can’t find a timestamp in the raw log, and it saves the cursor state at that point.

If the Event Breaker encounters another log later in the file with a timestamp before “Now”, it will ignore it. For

8.1.10.4. Journal Files Source

The Journal Files Source collects data from systemd's centralized logging, which is called the journald service. This service is a centralized location for all messages logged by different components in a systemd-enabled Linux system. This includes messages from kernel, boot, syslog, or other services. This Source is currently configured to configured to collect data from the `user` journal, as `system` is privileged. This Source is disabled by default.

Type: **System** | TLS Support: **N/A** | Event Breaker Support: **No**

The Journal Files Source is available in Cribl.Cloud on customer-managed hybrid Workers, but not on Cribl-managed Workers in Cribl.Cloud.

Discovering and Filtering Journal Files to Monitor

To determine all the names of all journals to read, the Journal Files Source runs a discovery procedure at a configurable **Polling interval**. The Source then applies an **Allowed list** to filter the initial list down into its final form. Then, for each journal on the list, the Source compares current state with previously stored state. This comparison determines whether the Journal Files Source will actually stream a given journal for a given polling interval. The Source then runs the files through the **Filter Rules** which allows you to set additional criteria.

Cribl Stream ships with a single, disabled instance of the Journal Files Source with a default configuration. The search path is configured to point to `$CRIBL_EDGE_FS_ROOT/var/log/journal/$MACHINE_ID` and defaults to streaming the primary `system.journal` it finds there. `$MACHINE_ID` is an environment variable that you can insert into the Source's search path. If you use the environment variable but don't assign a value, the Source will look up the local host `machine id` from `/etc/machine-id` and use that value.

Currently, this Source does not support all of the compression options that the journald service supports. The parser this Source uses to read the files can handle `LZ4` and `ZSTD` compression. When the parser encounters compressed fields it can't unpack, an error message emits in the logs. Below is an example of the error message:

```
Error reading journald event, dropping offset=31678152 evt="Mar 02 21:29:54
goats-xps15 multipassd: undefined"
```

Configure a Journal Files Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Source configuration. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Search path**: Specify the directory to search journals on. Accepts environment variables. The default value for this field varies depending on the product:
    - In Cribl Edge, it defaults to `$CRIBL_EDGE_FS_ROOT/var/log/journal/$MACHINE_ID`.
    - In Cribl Stream, it defaults to `/var/log/journal/$MACHINE_ID`.
    - **Journal allowlist**: The full path of each discovered journal is matched against this wildcard list. Defaults to `system`.
3. Next, you can configure the following **Optional Settings**:
    - **Polling interval**: How often, in seconds, to collect metrics. If not specified, defaults to `10` seconds.
    - **Filter Rules**: Optionally, specify which journal objects to allow. Events are generated if all the rules' expressions evaluate to true, or if no rules are specified.
    - **Filter expression**: JavaScript expression to filter Journal objects. Returns `true` to include it. For details, see Filter Expressions below.
    - **Description**: Optional description of the rule.
    - **Current boot only**: Skip events that are not part of the current boot session.
    - **Age duration limit**: Optionally, specify a maximum age of journal objects to stream. This Source will filter events with timestamps newer than the configured duration. Enter a duration in a format like `30s`, `4h`, `3d`, or `1w`. Defaults to an empty field, which applies no age filters.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Filter Expressions

The default **Filter expression** is `severity <= 4` with a **Description** of `Allow events having 'emergency', 'alert', 'critical', 'error', or 'warning' priority.`

`The screenshot below shows other examples you can use:`

Filter expressions

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings {advanced-tab}

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

In Cribl Edge, this Source defaults to QuickConnect. While, in Cribl Stream, it defaults to Send to Routes.

8.1.10.5. System Metrics Source

Cribl Stream instances can collect metrics from their Linux hosts, and can populate some standard metrics dashboards right out of the box. To see all of the metrics this Source supports, check out Linux System Metrics Details.

Type: **System** | TLS Support: **N/A** | Event Breaker Support: **No**

Cribl Edge Workers support this System Metrics Source only when running on Linux. To collect metrics from Windows hosts, use the Windows Metrics Source.

This Source is available in Cribl.Cloud on customer-managed hybrid Workers, but not on Cribl-managed Workers in Cribl.Cloud.

Configure Cribl Stream to Collect System Metrics

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. Configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name. The default Source is prefilled with the value `in_system_metrics`, which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
3. Next, you can configure the following **Optional Settings**:
    - **Polling interval**: How often to collect metrics, in seconds. Defaults to `10`.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can configure metrics for Host, Process, Container, or adjust Processing, Advanced settings and Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Host Metrics

Use the buttons to select a level of detail.

- **Basic** enables minimal metrics, averaged or aggregated.
- **All** enables full, detailed metrics, specified for individual CPUs, interfaces, and so on.
- **Custom** displays sub-menus and buttons from which you can choose a level of detail (**Basic**, **All**, **Custom**, or **Disabled**) for each type of event.
- **Disabled** means that no metrics will be generated.

The meaning of **All** and **Disabled** are self-evident. **Basic** and **Custom** have different meanings depending on event type, as follows:

System

Basic level captures load averages, uptime, and CPU count.

Custom toggles **Process** metrics on or off; these are metrics for the numbers of processes in various states.

CPU

Basic level captures active, user, system, idle, iowait percentages over all CPUs.

Custom toggles the following on or off: **Per CPU metrics**, **Detailed metrics** (meaning, for all CPU states), and **CPU time metrics** (meaning raw, monotonic CPU time counters).

Memory

Basic level captures captures total, used, available, `swap_free`, and `swap_total`.

Custom toggles **Detailed metrics** on or off. Detailed means for all memory states.

Network

Basic level captures bytes, packets, errors, connections over all interfaces.

Custom exposes the following:

- The **Interface filter**, which specifies which network interfaces to include or exclude (all are included if the filter is empty).
- **Per-interface metrics**, which toggle on or off.

- **Detailed metrics**, which toggle on or off. If on, the **Protocol metrics** toggle appears, allowing you to choose whether to generate metrics for ICMP, ICMPMsg, IP, TCP, UDP, and UDPLite.

Disk

Basic level captures disk used in percent, bytes read and written, and read and write operations, over all mounted disks.

Custom exposes the following:

- The **Device filter**, which specifies which block devices to include or exclude (all are included if the filter is empty). Wildcards and ！(not) operators are supported.
- The **Mountpoint filter**, which specifies which filesystem mountpoints to include or exclude (all are included if the filter is empty). Wildcards and ！(not) operators are supported.
- The **Filesystem type filter**, which specifies which filesystem types to include or exclude (all are included if the filter is empty). Wildcards and ！(not) operators are supported.
- **Per-device metrics**, which toggle on or off.
- **Detailed metrics**, which toggle on or off. If on, the **Enable inode metrics** toggle appears, allowing you to choose whether to generate metrics for filesystem inodes.

Process Metrics

With Process Metrics enabled, Cribl Stream captures process-specific metrics from Linux servers and reports them as events. This allows you to monitor specific processes on Cribl.Cloud instances. You can generate events for any process object.

To collect a process metrics event, create a Process Set and add a filter expression. Processes that match the filter are returned as individual events. See Collecting Metrics. Process Sets are separate from aggregate and host-wide metrics.

Process-specific metrics are **not** affected by the **Host Metrics** detail setting.

Adding a Process Set

To add a Process Set:

1. Open the System Metrics Source and access the **Configure** tab.
2. Select the **Process Metrics** menu, then **Add Process Set**.
3. Configure the details:
    - **Set Name**: The name for this process set.
    - **Filter Expression**: The JavaScript expression that will filter the processes.

- **Include Child Processes**: When toggled on, the processes that match the filter include metrics for child processes.

Filtering Processes

You can filter processes using the field names and values from each process object.

Tip

To see all processes currently running on an Edge Node, select **Explore** in the submenu and open the Processes Tab.

Example filters

This filter looks for Java processes in the sleep state (an interruptible wait) that are using more than 100% of the CPU or more than 25% of memory:

```
cmdline.args[0] == 'java' && status == 'S' && (cpu > 100 || memory_percent > 25)
```

This filter looks for running NGINX web servers that have more than 12 open file descriptors:

```
service == 'nginx' && fds > 12
```

Here are some more Linux processes you could find metrics for:

- `cmdline.args[0].endsWith('/java') && cmdline.args[1] == 'nifi'`
- `exe.path == '/usr/bin/sshd'`
- `uid == 2`
- `cgroup.0 == /user.slice/user-1000.slice/`
- `stat.status == R`
- `stat.starttime > 943517`
- `cpu > 30`
- `memory_percent > 30`
- `stat.num_threads > 10`

The **Filter expression** field expects a specific syntax in order to apply the filter to the processes. These expressions evaluate to either `true` or `false`. See Build Custom Logic to Route and Process Your Data: Filter Expressions for more information.

Collecting Metrics

Once you have at least one Process Set created and saved, Cribl Stream will begin collecting process metrics at the interval defined in the **Polling interval** field (**General Settings** menu).

To view the status of the Collector and total events collected, click the **Status** tab in the Linux Metrics Source.

To view a live capture of individual metrics gathered from processes that match the Process Set, click the **Live Data** tab. The Process Set that owns each metric is represented by the `__process_set` internal field.

Supported Processes

To view the complete table of supported Linux process metrics, their descriptions, types, and dimensions, see Process-Specific Metrics.

Using Advanced Mode

To test your filter expression against a sample input, click the **Advanced mode** button within the **Filter expression** field. The **Filter expression** modal will open. See Test and Validate Expressions in Advanced Mode for more information.

Container Metrics

Use the buttons to select a level of detail.

- **Basic** generates the server event and per running container events.
- **All** adds per-device and detailed metrics.
- **Custom** provides controls for customizing which containers to generate metrics from.
- **Disabled** means that no metrics will be generated.

The **Custom** buttons displays additional controls, as follows:

- **Container filters**: Enter a filter expression to govern which containers to generate metrics from. Leave empty (the default) to generate metrics from all containers.
- **All containers**: Toggle on to include stopped and paused containers.
- **Per-device metrics**: Toggle on to generate separate metrics for each device.
- **Detailed metrics**: Toggle on to generate full container metrics.

The **Basic**, **All**, and **Custom** buttons provide the following **Advanced Settings**, which are different from those in the main **Advanced Settings** tab:

- **Docker socket**: Enter the full path(s) for Docker's UNIX domain socket. Defaults to `/var/run/docker.sock` and `/run/docker.sock`.
- **Docker timeout**: Timeout, in seconds, for the Docker API. Defaults to `5`.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

If desired, choose a **Pipeline** from the drop-down if you want to process data from this Source before sending it through the Routes.

Disk Spooling

Enable disk spooling: Whether to save metrics to disk. When toggled on, exposes this section's remaining fields.

Bucket time span: The amount of time that data is held in each bucket before it's written to disk. The default is 10 minutes (`10m`).

Data size limit: Maximum disk space the persistent metrics can consume. Once reached, Cribl Stream will delete older data. Example values: `420 MB`, `4 GB`. Default value: `100 MB`.

Data age limit: How long to retain data. Once reached, Cribl Stream will delete older data. Example values: `2h`, `4d`. Default value: `24h` (24 hours).

Data compression format: Optionally compress the data before sending. Defaults to `gzip` compression. Select `none` to send uncompressed data.

Path location: Path to write metrics to. Default value is `$CRIBL_HOME/state/system_metrics`.

Advanced Settings

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Populating Dashboards with System Metrics

Cribl has configured a Prometheus dashboard to display Cribl Stream System Metrics, and shared the dashboard in the Grafana library, which is public. This is a relatively simple dashboard suitable for showing aggregate metrics. Try this dashboard if you prefer **Basic** mode for most of your metrics.

Another useful dashboard is the one that's commonly used with Prometheus and their Node Exporter agent, found here. This dashboard can handle the highly detailed metrics. To use this dashboard, attach the `prometheus_metrics` pre-processing pipeline, and choose **All** mode.

For details on how populate your Grafana Cloud dashboards with system metrics, see System Metrics to Grafana.

8.1.10.6. System State Source

The System State Source collects snapshots of the host system's current state, on a configurable schedule, and sends them out as events to downstream systems for operational and security analytics.

Type: **System** | TLS Support: **N/A** | Event Breaker Support: **No**

This Source is available in Cribl.Cloud on customer-managed hybrid Workers, but not on Cribl-managed Workers in Cribl.Cloud. Cribl Edge on macOS offers support for only a subset of System State collectors (see Cribl Edge on MacOS).

Configure Cribl Stream to Collect System State Events

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. Configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name. The default Source is prefilled with the value `in_system_state`, which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
3. Next, you can configure the following **Optional Settings**:
    - **Polling interval**: How often to collect metrics, in seconds. Defaults to `300`. Each run of the state collector generates events with identical `_time` values.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Collector, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.

5. Select **Save**, then **Commit & Deploy**.

Collector Settings

Cribl Stream contains the following System State collectors:

- Host Info
- Disks and File Systems
- DNS
- Firewall
- Hosts File
- Interfaces
- Listening Ports
- Logged-In Users
- Routes
- Services
- Users and Groups

Host Info

With the **Enabled** toggle on (the default), Cribl Stream will create events based on entries collected from the hosts file.

A partial list of Host Info events includes:

- `host`
- `hostOs` (when Cribl Edge is running in a container)
- `timestamp`
- `cribl.version`
- `cribl.mode`
- `cribl.group`
- `cribl.config_version`
- `os.arch`
- `os.cpu_count`

See the **Explore** > **System State** > **Host Info** UI for a full list of events.

The System State Source provides live system information, with most tabs updating periodically based on the **Polling Interval** configured in the Source. However, the data in the **Host Info** tab is collected once when the Source is started and does not update dynamically.

Disks and File Systems

With the **Enabled** toggle on (the default), Cribl Stream will collect entries on physical disks, partitions, and mounted filesystems. These entries can help you monitor drive status, including available space, percent utilization, inode availability, wait times, and number of blocks free.

On Cribl Edge, this option is available for Windows and Linux. You must create a different Worker Group for each OS. For details see, Mapping Edge Nodes to Fleets.

For each physical disk, Cribl Stream will collect:

- name
- size (in blocks)
- blockSize (in bytes)
- model
- serial
- firmware
- partitions (count)
- interface type

For each partition on a physical disk, Cribl Stream will collect:

- disk (name)
- disk ID (number)
- type (primary or logical)
- label
- offset
- size (in blocks)
- blockSize (in bytes)
- filesystem (mount point)

For each mounted filesystem, Cribl Stream will collect:

- mount point (path)
- disk (name)
- partition (ID)
- filesystem type
- size (in blocks)
- blockSize (in bytes)
- blocksFree
- inodes
- inodesFree
- flags

Mount point refers to the directory where a filesystem is currently mounted. This information reflects the active filesystems, not entries in the `/etc/fstab` file.

DNS

With the **Enabled** toggle on (the default), Cribl Stream will create events for DNS resolvers and search entries. The event field mapping depends on the operating system.

When Cribl Edge is deployed in a container, the **DNS** Collector does not collect any events.

Linux

Cribl Stream emits events with the following fields:

Event Field	Description
`nameServer`	Name server entries from the file.
`search`	Search entries from the file.
`source`	The source of the data used to populate the event.

Windows (Cribl Edge Only)

Cribl Edge on Windows emits events with the following fields:

Event Field	Description
`ifAlias`	Interface alias name (if present).

Event Field	Description
`ifIndex`	Interface index (if present).
`addressFamily`	IPv4 or IPv6.
`nameServer`	Array of name servers.
`search`	Array of search suffixes .
`source`	The source of the data used to populate the event.
`dnsClient`	Reports `MSFT_DnsClient` when the native WMI method is used (default). Reports `Get-DnsClient` when the **Use Windows Tools** toggle is enabled in **Advanced Settings**.

Hosts File

With the **Enabled** toggle on (the default), Cribl Stream will collect entries from the `hosts` file and emit them as events.

Examples of entries in the `hosts` file are:

- `ip`: IP address of the host.
- `hostnames`: list of hostnames per IP address.

Firewall

With the **Enabled** toggle on (the default), Cribl Stream will collect events from host's defined firewall rules.

Chain Policy/References Rule(s) Event Fields

Cribl Stream emits events with the following fields:

- `chain`: Name.
- `policy`: Policy name.
- `pkts`: Total bytes packets processed by the chain.
- `bytes`: Total bytes processed by the chain.
- `references`: Number of chains referencing the current chain.

Chain Rule(s) Event Fields

Cribl Stream emits events with the following fields:

- `chain`: Name.

- `num`: Rule sequence number.
- `pkts`: Number of matched messages processed.
- `bytes`: Cumulative packet size processed (bytes).
- `target`: If the message matches the rule, the specified target is executed.
- `prot`: The specified protocol can be one of `tcp`, `udp`, `udplite`, `icmp`, `icmpv6`, `esp`, `ah`, `sctp`, `mh`, or the special keyword `all`.
- `opt`: Rarely used, this column is used to display IP options.
- `in`: Inbound network interface.
- `out`: Outbound network interface.
- `src`: The source IP address or subnet of the traffic, the latter being anywhere.
- `dest`: The destination IP address or subnet of the traffic, or anywhere.
- `match`: `ctstate`, `state`, `ADDRTYPE` info added to a state field.
- `comment`: Comment defined on the rule.
- `prot_family`: `IPv4` or `IPv6`.

Windows (Cribl Edge Only)

On Cribl Edge Windows, this collector emits events with the following fields:

- `ruleName`: Rule Name.
- `displayName`: Display Name.
- `direction`: Specifies which direction of traffic to match with this rule. The acceptable values for this parameter are `Inbound` or `Outbound`.
- `enabled`: `True` or `False`.
- `action`: Specifies the action to take on traffic that matches this rule. The acceptable values for this parameter are `Allow` or `Block`.
- `group`: Specifies the rule group (if applicable).
- `icmpType`: Specifies the `ICMP` type used.
- `policyStoreSource`: Contains a path to the policy store where the rule originated.
- `profile`: Profile conditions associated with a rule.
- `protocol`: protocols associated with firewall and `IPsec` rules.
- `localPort`: Rules for local-only port mapping.
- `remotePort`: Rules for remote port mapping.

Interfaces

With the **Enabled** toggle on (the default), Cribl Stream will create events for each of the host's network interfaces.

Cribl Stream will create separate events for each entry. All events will have identical timestamp values.

Linux IPV4/IPV6

On Linux, Cribl Stream creates `interface` events with the following fields:

- `ifIndex`: Interface index identification numbers.
- `ifName`: The name of the network interface.
- `flags`: Flags.
- `mtu`: Maximum transmission unit (MTU) in bytes for packets sent on this interface.
- `operstate`: Operational State.
- `linkType`: Physical link type.
- `macAddress`: Media Access Control address.
- `broadcast` Broadcast address.
- `addrInfo`: Array of `{family, ipAddress, mask, prefix}`.

Windows (Cribl Edge Only): IPV4/IPV6

On Windows, Cribl Edge creates `interface` events with the following fields:

- `ifIndex`: Interface index identification numbers.
- `interface`: The name of the network interface.
- `mtu`: Maximum transmission unit (MTU) in bytes for packets sent on this interface.
- `flags`: Flags.
- `macAddress`: Media Access Control address.
- `addrInfo`: Array of `{family, ipAddress, mask, prefix}`.

Listening Ports

With the **Enabled** toggle on (the default), Cribl Stream will create events from the list of listening ports and their associated process identifier (pid).

Cribl Stream will create separate events for each entry. All events will have identical timestamp values.

Sample entries include:

- `Protocol Family`: `ipv4` or `ipv6`.
- `Protocol`: `TCP`, `UDP`, or `UNIX`.
- `Socket`: `${addr}:${port}`, or `${path}` for UNIX sockets.
- `Process id`: The process identifier.
- `Program`: The program listening on the port.
- `listeningPorts`: In Cribl Edge (Windows), reports `MSFT_NetTCPConnection` and `MSFT_NetUDPEndpoint` when the native WMI method is used (default). Reports `Get-NetTCPConnection` and `Get-NetUDPEndpoint` when the **Use Windows Tools** toggle is enabled in **Advanced Settings**.

Logged-In Users

With the **Enabled** toggle on (the default), Cribl Stream will collect entries from currently logged-in users.

Linux

Cribl Stream emits events with the following fields:

- `uid`: User ID.
- `Last Login date`: Last login date.
- `terminal`: Type of the terminal device.
- `hostname`: Display the hostname of the system, if the user is connected from a remote computer.
- `userName`: Logon name of the user.

Windows (Cribl Edge Only)

Cribl Edge Windows emits events with the following fields:

- `userName`: Login name of the user.
- `sessionName`: `rdp` (Remote desktop/terminal services login session) or `console` (Direct login session).
- `id`: Session ID.
- `state`: `active` (Active session) or `disc` (Disconnected/inactive session).
- `logonTime`: Date and time the user logged on.
- `source`: Set to `Query User`.

Routes

With the **Enabled** toggle on (the default), Cribl Stream will collect entries from host's network routes and emit them as events.

Sample entries include:

- `address_family`: `ipv4` or `ipv6`.
- `destination`: The destination network or host.
- `prefix`: Emits only from Linux IPv6.
- `flags`: Flags.
- `gateway`: The gateway address.
- `interface`: Interface to which packets for this route will be sent.
- `ifIndex`: Interface ID. Emits only from Windows.
- `mask`: The netmask for the destination net. Emits only from Linux IPv4.
- `metric`: The distance to the target.
- `sequence`: Sequence number in the table.
- `source`: The source of the data used to populate the event.
- `routes`: In Cribl Edge on Windows, reports as `MSFT_NetRoute` when the native WMI method is used (default). Reports `Get-NetRoute` when the **Use Windows Tools** toggle is enabled in **Advanced Settings**.

Services

With the **Enabled** toggle on (the default), Cribl Stream will create events for each configured service (for example `systemd` and `initd`) along with their enabled and running status.

Sample entries include:

- `Name`: Name of the service.
- `Unit Activation Status`: The high-level unit activation state, i.e. generalization of `SUB` for `systemctl`. For `sysV` system value can be `enabled` or `disabled`.
- `Unit Load Status`: Reflects whether the unit definition was properly loaded. (Only `systemctl`).
- `Sub Unit Activation Status`: The low-level unit activation state, values depend on unit type.
- `Description`: Description of the service.
- `windowsService`: In Cribl Edge on Windows, reports as `Win32_Service` when the native WMI method is used (default). Reports `Get-Service` when the **Use Windows Tools** toggle is enabled in **Advanced Settings**.

When Cribl Edge is deployed in a container, the **Services** Collector does not collect any events.

Users and Groups

With the **Enabled** toggle on (the default), Cribl Stream will create events for users and groups.

Linux

The Linux collector reports exclusively on local users and groups defined within `/etc/passwd` and `/etc/group`. It does not include users or groups from external directory services like LDAP.

On Linux, Cribl Stream creates `user` events with the following fields:

- `username`
- `fullname`
- `loginShell`
- `userHome`
- `userid`
- `primaryGroup` (name of primary group, not group ID)
- `groups` (list of group names)

Cribl Stream also creates `group` events with the following fields:

- `name`
- `groupId`
- `users`

Windows (Cribl Edge Only)

Cribl Edge on Windows reports exclusively on local users and groups. On Windows hosts that are members of a domain (but are not domain controllers), Cribl Edge will only list local accounts. On Windows domain controllers, the collector will not list any users or groups as the local account list is replaced by the domain accounts.

On Windows, Cribl Edge creates `user` events with the following fields:

- `userName`
- `fullname`
- `description`
- `groups`

Cribl Edge also creates `group` events with the following fields:

- `name`
- `description`
- `users`

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

Select a **Pipeline** (or Pack) from the drop-down to process this Source's data. Required to configure this Source via **Data Routes**; optional to configure via **Collect**/**QuickConnect**.

Disk Spooling

Enable disk spooling: Whether to save metrics to disk. When toggled on, exposes this section's remaining fields.

Bucket time span: The amount of time that data is held in each bucket before it's written to disk. The default is 10 minutes (`10m`).

Max data size: Maximum disk space the persistent metrics can consume. Once reached, Cribl Stream will delete older data. Example values: `420 MB`, `4 GB`. Default value: `100 MB`.

Max data age: How long to retain data. Once reached, Cribl Stream will delete older data. Example values: `2h`, `4d`. Default value: `24h` (24 hours).

Compression: Currently locked to `none`. Cribl plans to add (optional) disk-spooling compression in a future release.

Path location: Path to write metrics to. Default value is `$CRIBL_HOME/state/system_state`.

Advanced Settings

Use Windows Tools: Toggled off by default. This legacy setting will be removed in a future release. We highly recommend keeping this option disabled and using the newer native capabilities which are faster and more reliable.

This setting affects Windows Edge Nodes only, it has no effect on Linux Edge Nodes or in Cribl Stream.

- If toggled off, the System State Source interacts directly with the Windows Management Instrumentation (WMI) API, resulting in faster data collection.
- If toggled on, the System State Source will collect events using Powershell cmdlets.

Use legacy collection for LastLog: Toggled off by default. This legacy setting will be removed in a future release. We highly recommend keeping this option disabled and using the newer native capabilities which are faster and more reliable.

Toggle on if you encounter issues with collecting LastLog information to revert to the legacy collection method.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

8.1.11. AppScope Source

AppScope is deprecated and will be removed in a future release. Don't use AppScope in production and ensure it is not part of your data flow configuration. For details, please read the announcement in Cribl Community Slack.

AppScope is an open-source instrumentation utility from Cribl. It offers visibility into any Linux command or application, regardless of runtime, with no code modification. For details about configuring the AppScope CLI, loader, and library, see: https://appscope.dev/docs.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

Conceptual Walkthrough of AppScope

This two-minute video provides a conceptual walkthrough of Cribl open-source AppScope project.

Download AppScope

Cribl Stream must download the AppScope package from the Cribl CDN the first time it performs an operation on this Source. This will also update the AppScope package's version, if a newer one is available in the CDN.

If Cribl Stream cannot access the CDN (for example, if you are working on an airgapped instance), you should manually download the binary and place it in `$CRIBL_HOME/state/download/scope`.

Configure Cribl Stream to Receive AppScope Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this AppScope Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **UNIX domain socket**: For details, see Unix Domain Socket Fields below.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Use this default option to enter the shared secret clients must provide in the `authToken` header field. Select **Generate** if you need a new auth token. If empty, unauthorized access will be permitted.
    - **Secret**: This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the auth token described above. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. Click **Create** if you need to configure a new secret.
4. Next, you can configure the following **Optional Settings**:
    - **UNIX socket permissions**: Permissions to set for socket. For the preconfigured `in_appscope` source, defaults to `777`. When creating a new AppScope Source, you should set this to `777`. If empty, falls back to the runtime user's default permissions.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, configure any TLS settings, AppScope Rules, Persistent Queue, Processing, Advanced settings and Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Unix Domain Socket Fields

When toggled on, the **Unix doman socket** field exposes the following two fields to specify a file-backed UNIX domain socket connection to listen on.

- **UNIX socket path**: Path to the UNIX domain socket. Defaults to `$CRIBL_HOME/state/appscope.sock`.
- **UNIX socket permissions**: Permissions to set for this socket, for example, `777`. If empty, Cribl Stream will use the runtime user's default permissions. (Appears under **Optional Settings**)

When **UNIX domain socket** is toggled off, you instead see the following two fields to specify a network host and port.

- **Address**: Enter the hostname/IP on which to listen for AppScope data. (For example, `localhost`.) Defaults to `0.0.0.0`, meaning all addresses.
- **Port**: Enter the port number to listen on.

By default:

- In Cribl Stream, **UNIX domain socket** is toggled off, with default network connections (address and port) of `0.0.0.0:10090` for TCP, and `0.0.0.0:10091` for TLS, respectively.
- In Cribl Edge, **UNIX domain socket** is toggled on, with a UNIX socket path of `$CRIBL_HOME/state/appscope.sock`.

TLS Settings (Server Side)

This left tab is displayed only when the Optional Settings toggle **UNIX domain socket** is off. It provides the following options.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

AppScope Rules

The AppScope Rules settings are available:

- In Cribl Stream single-instance - but not Cribl.Cloud - deployments.
- In Cribl Edge, both single-instance and Cloud.

Rules: Click **Add Rule** to include processes to scope, and to link to an AppScope config. Once you have saved the configuration, and committed and deployed your changes, AppScope will instrument any process that matches a Rule, on any Edge Node in the Fleet.

(When no Rules are defined, you can still scope by PID in the Edge Processes page. Scope by PID only instruments a single process running in one Edge Node.)

- **Process name**: Matches if the string value you enter corresponds to the basename of the scoped process.
- **Process argument**: Matches if the string value you enter appears as a substring anywhere in the scoped process' full command line (including options and arguments).
- **AppScope config**: Select an AppScope config.

Transport override: Enter a URL to override aspects of the transport configuration, such as the hostname, port, or TLS settings. For details, see Transport Override Details.

Transport Override Details

In scenarios like the following, use the **Transport override** option to extend the defaults in AppScope's transport configuration:

- When this Source is set to **TCP** mode, it typically listens on the default address `0.0.0.0`. Scoped clients will need a specific `IP` or `hostname` to connect. In these cases, set **Transport override** URL to a specific

IP/hostname (example format: `tcp://my.host.name`). This Source will parse the URL and look for the `hostname` and `port`, then use those values to override what would otherwise be sent to the `scope start` command.

- When this Source is set to **UNIX domain socket**, it listens by default on `$CRIBL_HOME/state/appscope.sock`. The socket is often created on a mounted volume in a container. The path to that socket might be different outside the container, or when mounted into another container. In these cases, set the **Transport override** URL to specify an alternative path (example format: `unix:///some/other/volume/appscope.sock`).

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts

to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Event Breakers

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Disk Spooling

For Cribl Search to access the data that arrives at an AppScope Source, **Disk Spooling** must be enabled.

Enable disk persistence: Whether to save metrics to disk. When toggled on, exposes this section's remaining fields.

Bucket time span: The amount of time that data is held in each bucket before it's written to disk. The default is 10 minutes (`10m`).

Max data size: Maximum disk space the persistent metrics can consume. Once reached, Cribl Stream will delete older data. Example values: `420 MB`, `4 GB`. Default value: `100 MB`.

Max data age: How long to retain data. Once reached, Cribl Stream will delete older data. Example values: `2h`, `4d`. Default value: `24h` (24 hours).

Compression: Optionally compress the data before sending. Defaults to `gzip` compression. Select `none` to send uncompressed data.

Path location: Path to write metrics to. Default value is `$CRIBL_HOME/state/appscope.sock`.

Advanced Settings

Enable proxy protocol: Toggle on if the connection is proxied by a device that supports Proxy Protocol v1 or v2.

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection.

Active connection limit: Maximum number of active connections allowed per Worker Process. Defaults to `1000`; enter `0` to allow unlimited connections.

Socket idle timeout (seconds): The duration that Cribl Stream will wait for activity on an idle TCP socket before closing the connection. Disabled when set to `0`, the default.

Forced socket termination timeout (seconds): The extra time the server waits before forcibly closing a socket that has been idle (**TCP socket idle timeout**) or exceeded its maximum lifespan (**TCP socket max lifespan**) but has not yet properly closed. This prevents resource leaks caused by unresponsive clients or network issues. Configure based on network latency and client behavior. Default: `30` seconds. Set to `0` to disable.

Socket max lifespan (seconds): The duration that a socket is allowed to remain open, regardless of activity. This setting prevents resource exhaustion (such as TCP pinning) by limiting the lifespan of connections. Configure based on expected connection durations and resource availability. Disabled when set to `0`, the default.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

AppScope with Edge on Kubernetes

When Cribl Edge detects that a `scope`'d process is running inside a Kubernetes container, it reports the Kubernetes metadata as `kube_**` properties. It adds these to the incoming events and metrics, and you can view the combined events and metrics on the AppScope Source's **Live Data** tab.

For Cribl Edge to detect that it's running in a Kubernetes Pod, you must first set the `CRIBL_K8S_POD` environment variable.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`
- `__srcIpPort`

Examples

The following examples work only in Cribl Stream. You can vary the scoped commands (`ps -ef` and `curl`) as desired.

Cribl.Cloud - TLS

An `in_appscope_tls` TLS Source is preconfigured for you on Cribl.Cloud, using port `10090`. You can send it AppScope data using this command:

```
./scope run -c <Your-Ingress-Address>:10090 -- ps -ef
```

Copy

Cribl Cloud - TCP

An `in_appscope_tcp` TCP Source is preconfigured for you on Cribl.Cloud, using port `10091`. You can send it AppScope data using this command:

```
./scope run -c tcp://<Your-Ingress-Address>:10091 -- curl -so /dev/null \
https://wttr.in/94105
```

Copy

Periodic Logging

Cribl Stream logs metrics about incoming requests and ingested events once per minute.

These logs are stored in the `metrics.log` file. To view them in the UI, open the Source's **Logs** tab and choose **Worker Process X Metrics** from the drop-down, where **X** is the desired Worker process.

This kind of periodic logging helps you determine whether a Source is in fact still healthy even when no data is

8.1.12. Cloudflare Source

The Cloudflare Source receives logs and events directly from Cloudflare Logpush using the HTTP Event Collector (HEC) protocol. This Source is designed for seamless integration with the Cloudflare push API, supporting both structured and raw event formats.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

Configure a Cloudflare Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. Configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP on which to listen for HTTP(S) data. Such as: `localhost` or `0.0.0.0`. Supports IPv4 and IPv6 addresses.
    - **Port**: Enter the port number.
    - **HEC endpoint**: Absolute path on which to listen for the Cloudflare HTTP Event Collector API requests. Defaults to `/services/collector`. This single endpoint supports JSON events via `/event` and raw events via `/raw`.
3. Next, you can configure the following **Optional Settings**:
    - **Allowed Indexes**: List the values allowed in the HEC event index field. Allows wildcards. Leave blank to skip validation.
    - **Tags**: Add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing, Auth tokens, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled off.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Event Breakers

Cloudflare Source Event Breakers only function with the `/raw` endpoint. The `/event` endpoint operates on the assumption that the data they receive is already properly formatted into individual events and therefore do not utilize Event Breakers.

- `/raw`: Event Breakers split incoming raw, unstructured data into distinct events before processing them through Pipelines.
- `/event`: Event Breakers are not applied since data is already segmented in JSON format.

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can add Fields to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to determine the field's value (can be a constant).

Fields specified on the **Fields** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

In particular, where incoming events have no `index` field, this Source adds one with the literal value `default`. You can override this value by using **Add Field** to specify an `index` field, and then setting its **Value** to an expression of the following form: `index == 'default' ? 'myIndex' : index`

Fields here are evaluated and applied **after** any fields specified in the Auth Tokens section.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Auth Tokens

Provides the authentication token required to access this Source's data. Ensure the token is valid and has the necessary permissions for data retrieval.

Not providing a token allows **unauthorized access**.

Authentication method: Choose between manually entering a **Token** or selecting a stored **Secret**. If you opt for manual entry, type the token directly or **Generate** a new one. For added security, use the secrets store to select a stored **Token (text secret)** or **Create** a new one.

Enable token: Controls the status of the specified authentication token. When toggled on, the specified auth token is active and can be used for client access. If toggled off, the token is disabled, causing it to be skipped and ignored during initialization. This means that any client attempting to use a disabled token will not be granted access.

Disabling a token can be helpful when you need to temporarily stop receiving data from a particular service or application. Disabling a token prevents the service or application from sending data to your endpoint without having to completely delete the token.

See also Periodic Logging for information on how auth tokens affect product logging.

Description: Optional (but recommended), the description for this token. Although this field is optional, the token description becomes the display name for the token. You can select this display name to expand or collapse the token details.

Allowed indexes: Optionally, specify which indexes are permissible for events ingested using this token. Supports multiple index values and wildcards.

Fields: Fields to add to events referencing this token. Each field is a **Name/Value** pair.

Fields specified on the **Auth Tokens** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

In particular, where incoming events have no `index` field, this Source adds one with the literal value `default`. You can override this value by using **Add Field** to specify an `index` field, and then setting its **Value** to an expression of the following form: `index == 'default' ? 'myIndex' : index`

Fields here are evaluated and applied **before** any fields specified in the Fields section.

Advanced Settings

Emit per-token request metrics: Toggle on to emit per-token `<prefix>.http.perToken` and summary `<prefix>.http.summary` request metrics. This enables Periodic Logging.

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

CORS allowed origins: If you need to enable cross-origin resource sharing, use this field to specify the HTTP origins to which Cribl Stream should send `Access-Control-Allow-*` headers. You can enter multiple domains and use wildcards. This and its companion **CORS allowed headers** option should seldom be needed - see Working with CORS Headers for details.

CORS allowed headers: List HTTP headers that Cribl Stream will send in a CORS preflight response to your configured **CORS allowed origins** as `Access-Control-Allow-Headers`. Enter `*` to allow all headers.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; and so forth.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

> The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) by the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__hecToken`
- `__inputId`

- `__ctrlFields`
- `__srcIpPort` - See Overriding `__srcIpPort` with Client IP/Port for details.
- `__TZ`

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the client sending data to this Source.

When any proxies (including load balancers) lie between the client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Working with CORS Headers

The **CORS allowed origins** and **CORS allowed headers** settings are needed **only** when you want JavaScript code running in a web browser to send events to the Source. In this (uncommon) scenario, the code in question sends a preflight request that includes CORS (Cross-Origin Resource Sharing) headers, and the Source includes CORS headers in its response.

If the settings (and thus the CORS headers in the response) are correct, the browser will allow the code in question to complete requests to the Source - that is, allow the code to read the Source's responses. (These settings are optional because for some web applications it's sufficient to send requests **without** reading responses.)

The CORS header dance begins when the web browser sends a request with a header stating its origin. What happens next depends on how you've configured the CORS header settings, as explained in the following tables. (For these examples, we'll assume an origin of `https://mycoolwebapp:3001`.)

CORS allowed origins value	What Source does	Does Browser then allow code to read response?
`*`	Send `Access-Control-Allow-Origin: *` header.	Yes

CORS allowed origins value	What Source does	Does Browser then allow code to read response?
Multiple wildcard values	If origin value matches any wildcard, send `Access-Control-Allow-Origin: https://mycoolwebapp:3001` header.	Yes
Nothing (not configured)	Send neither an `Access-Control-Allow-Origin` nor an `Access-Control-Allow-Headers` header.	No

The web browser's request might also include an `Access-Control-Request-Headers` header that lists one or more headers that it wants to use in a subsequent `POST` request (that is, the "actual" as opposed to preflight request). The Source can then respond with an `Access-Control-Allow-Headers` header **if and only if** it's also sending the `Access-Control-Allow-Origin` header.

CORS allowed headers value	What Source does	Does Browser then allow code to read response?
`*`	Send `Access-Control-Allow-Headers` header with the same list the client sent.	Yes
Multiple wildcard values	If any headers from the client's list match wildcards, send `Access-Control-Allow-Headers` header listing the matching headers.	Yes
Nothing (not configured)	Do not send an `Access-Control-Allow-Headers` header.	Probably not

Periodic Logging

Cribl Stream emits metrics about incoming requests when **Advanced Settings > Emit per-token request metrics** is enabled. These metrics can be routed to a Destination by enabling the Cribl Metrics Source. There are two categories of metrics: per-token metrics provide details for each configured token, while summary metrics aggregate data across all tokens. If no tokens are configured, only summary metrics will be emitted.

Per-token metric names will be prefixed by `<basePrefix>.http.perToken.`, where `<basePrefix>` is the configured base prefix for the Cribl Metrics Source. The available metric names are listed below, excluding the prefix. As these are counter-type metrics, they will not be emitted if the value is zero.

- `bytesReceived`
- `numRequests`
- `numEvents`

- `numErrors` *Note: this is cumulative of specific error types below as well as a catch-all for other types*
- `numParserErrors`
- `numToDisabledToken`

The summary metric names will be prefixed by `<basePrefix>.http.summary.` where `<basePrefix>` is the configured base prefix for the Cribl Metrics Source. The available metric names are listed below, excluding the prefix. As these are counter-type metrics, they will not be emitted if the value is zero.

- `bytesReceived`
- `numRequests`
- `numEvents`
- `numErrors` *Note: this is cumulative of specific error types below as well as a catch-all for other types*
- `numParserErrors`
- `numToDisabledToken`
- `numAuthFailures`
- `numToWrongUrl`
- `numAckReqs`
- `numReqsAcked`

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate,

8.1.13. CrowdStrike FDR Source

Cribl Stream supports receiving data from the CrowdStrike Falcon Data Replicator (FDR) platform. CrowdStrike data can then be sent to SIEM, threat-hunting, and other security tools and platforms.

Type: **Pull** | TLS Support: **Yes** | Event Breaker Support: **Yes**

This page covers how to configure the Source. Because the FDR platform pulls data from Amazon S3 buckets maintained by CrowdStrike, some of the configuration described here actually involves S3.

For configuration examples that you can import and adapt to your needs, examine these resources on the Cribl Packs Dispensary:

- CrowdStrike FDR Pack.
- Exabeam Pack for CrowdStrike FDR.

Configure a CrowdStrike Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Unique ID for this Source. For example, `Endpoint42Investigation`.
    - **Description**: Optionally, enter a description.
    - **Queue**: The name, URL, or ARN of the SQS queue to read notifications from. When a non-AWS URL is specified, format must be: `'{url}/myQueueName'`. For example, `'https://host:port/myQueueName'`. The value must be a JavaScript expression (which can evaluate to a constant value), enclosed in quotes or backticks. The expression can only be evaluated at init time, for example when referencing a Global Variable like this: `https://host:port/myQueue-${C.vars.myVar}`.
3. Next, you can configure the following **Optional Settings**:
    - **Filename filter**: Regex matching file names to download and process. Defaults to: `.*`.

- **Region**: AWS Region where the S3 bucket and SQS queue are located. Required, unless **Queue** is a URL or ARN that includes a Region.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Assume Role, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the buttons to select an authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.

- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select a secret key pair that you've configured in Cribl Stream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for Amazon S3: Whether to use Assume Role credentials to access S3. Defaults to toggled on.

Enable for Amazon SQS: Whether to use Assume Role credentials when accessing SQS (Amazon Simple Queue Service). Default is toggled off.

AWS account ID: SQS queue owner's AWS account ID. Leave empty if the SQS queue is in the same AWS account.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role.

Processing Settings

Custom Command

In this section, you can pass the data from this input to an external command for processing, before the data continues downstream.

Enabled: Default is toggled off. Toggle on to enable the custom command.

Command: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).

Arguments: Click **Add Argument** to add each argument to the command. You can drag arguments vertically to resequence them.

Event Breakers

In this section, you can apply event breaking rules to convert data streams to discrete events.

Event Breaker rulesets: A list of event breaking rulesets that will be applied, in order, to the input data stream. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

Optionally, use the drop-down to select an existing Pipeline to process data from this Source before sending it through the Routes. Otherwise (by default), events will be sent to normal routing and event processing.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

Checkpoint Settings

Enable checkpointing: When toggled off (default) Cribl Stream will always start processing a file from the beginning. Toggle on if, after an error or restart occurs, you want Cribl Stream to resume processing a given file from the point where it was interrupted, as explained below. When turned on, the following additional setting becomes available:

- **Retries**: The number of times you want Cribl Stream to retry processing a file after an error or restart occurs. If **Skip file on error** is enabled, this setting is ignored.

Other Settings

Endpoint: The S3 service endpoint you want CrowdStrike to use. If empty, Cribl Stream will automatically construct the endpoint from the AWS Region. To access the AWS S3 endpoints, use the path-style URL format. You don't need to specify the bucket name in the URL, because Cribl Stream will automatically add it to the URL path. For details, see AWS' Path-Style Requests topic.

Signature version: Signature version to use for signing S3 requests. Defaults to `v4`; `v2` is also available.

Message limit: The maximum number of messages that SQS should return in a poll request. Amazon SQS never returns more messages than this value. (However, fewer messages might be returned.) Acceptable values: `1` to `10`. Defaults to `1`.

Visibility timeout seconds: After messages are retrieved by a `ReceiveMessage` request, Cribl Stream will hide those messages from subsequent retrieve requests for at least this duration. Default is `21600` seconds (6 hours); maximum allowed threshold is `43200` sec. (12 hours). Set to `0` for no message hiding.

Cribl Stream will automatically extend this timeout until the initial request's files have been processed - notably, for large files that require additional processing time. The generous 6-hour default allows each Worker Process time to decompress, stitch, and read large multi-part files, without duplicate processing.

Number of receivers: The number of receiver processes to run. The higher the number, the better the throughput, at the expense of CPU overhead. Defaults to `1`.

Poll timeout (secs): How long to wait for events before polling again. Minimum `1` second; default `10`; maximum `20`. Short durations increase the number - and thus the cost - of requests sent to AWS. (The UI will show a warning for intervals shorter than `5` seconds.) Long durations increase the time the Source takes to react to configuration changes and system restarts.

Socket timeout: Socket inactivity timeout (in seconds). Increase this value if retrievals time out during backpressure. Defaults to `300` seconds.

Skip file on error: Toggle on to skip files that trigger a processing error. (For example, corrupted files.) Default is toggled off, which enables retries after a processing error.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to reject certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on, the restrictive option.

Encoding: Character encoding to use when parsing ingested data. If not set, Cribl Stream will default to UTF-8 but might incorrectly interpret multi-byte characters. This option is ignored for Parquet files. UTF-16LE and Latin-1 are also supported.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Checkpointing

When Workers restart, checkpointing can prevent the Source from losing data or collecting duplicate events. Among the situations where this works well:

- When Cribl Stream pushes a new configuration: the Leader pushes the configuration to the Workers, and restarts them.
- When a Worker has gotten into a bad state, and you want to restart it.

Checkpointing allows Cribl Stream to resume processing a file at the point where an error or restart occurred, minimizing the number of duplicate events sent downstream in the event of an interruption. To accomplish this, Cribl Stream keeps track of how many events have been processed for a given file. Resuming processing after an interruption, the Source skips the first *N* events it has seen and starts processing from event *N*+1.

The Source manages checkpoint information in memory, but writes it out to disk periodically (once per second, and on Worker shutdown) so that the information can survive a Worker restart. Cribl Stream keeps checkpoint information at the Worker Process level and does not share it between Worker Processes or Worker Nodes.

Checkpointing is a feature to keep in mind when you do disaster recovery (DR) planning.

Caveats to using checkpointing include: Whenever Worker Node state is destroyed (that is, when a Worker ceases to exist), checkpoint information will be lost. In Cribl.Cloud deployments, this can happen during upgrades. Also, because checkpoints are not written to disk on **every** event, the checkpoints written to disk will lag behind those that the S3 Source manages in memory. If the Worker Process stops abruptly without a clean shutdown, checkpoints not yet written to disk will be lost. When processing resumes, the Source will start from just after the last checkpoint written to disk, producing duplicate events corresponding to the lost checkpoints.

Located in the UI at **Advanced Settings** > **Checkpoint Settings** > **Enable checkpointing**, this feature is turned off by default.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

How Cribl Stream Pulls Data

Worker Processes poll for messages from SQS, using the AWS SDK S3 APIs. Each polling call will return (by default) one message, if any are available. You can change this default using `Max messages`.

If no message is available, the call will time out after one second, and then polling will repeat. All Worker Processes participate in polling, so as to disribute files' collection and processing evenly across Workers.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate,

8.1.14. Datadog Agent Source In Cribl Stream

Datadog Agent is open-source software that monitors the host on which it runs. Acting as a DogStatsD server, Datadog Agent also aggregates metrics from other processes or containers on the host.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Looking for the Datadog Agent Source in **Cribl Search**? See Add a Datadog Agent Source for Cribl Search.

Cribl Stream can ingest the following from Datadog Agent, in bundled form:

- Logs.
- Metrics (gauge, rate, counter, and histogram).
- Service checks.
- Agent metadata and other events emitted from the `/intake/` endpoint.

APM traces: While the Datadog Agent typically sends APM data to Datadog by default, this Source cannot receive APM data routed through the Datadog Agent (via `DD_APM_DD_URL`). Instead, it can receive APM traces directly from applications that use the Datadog tracing libraries (`dd-trace`) when those applications are configured to send trace data to this Source using the Datadog trace API v0.3 or v0.4 endpoints. For details on instrumenting your applications, see the Datadog tracing library documentation.

On the system(s) that you want to monitor, you'll need to install Datadog Agent and configure it to send data to Cribl Stream. Cribl Stream can parse, filter, and enrich that data, and then send it to any supported Destination, including a Cribl Stream Datadog Destination. (By default, Datadog Agent sends data only to Datadog.)

For inbound log data, this Source supports gzip -compression when the `Content-Encoding: gzip` connection header is set. For other data types, it assumes that all inbound data is compressed using `deflate`.

Configure Datadog to use Datadog API v1

This Source communicates with the Datadog API. Although v2 is the latest version of this API, this Source uses v1, which Datadog still supports.

To configure your Datadog Agent to work with this Source, ensure that in the `datadog.yaml` file, `use_v2_api.series` is set to `false`. Otherwise, when Datadog Agent sends metrics, Cribl Stream will not receive them, and `API Key invalid, dropping transaction` errors will appear in the Datadog Agent logs.

> Potential Cribl Stream support for Datadog API v2 is being tracked as **CRIBL-18281** in Known Issues.

Configure Cribl Stream to Ingest Datadog Agent Output

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter hostname/IP to listen for Datadog Agent data. For example, `localhost` or `0.0.0.0`.
    - **Port**: Enter the port number to listen on.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.
>
> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

> `Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

> In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).
>
> You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests per Worker Process. Use `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

Extract metrics: Toggle on to extract each incoming metric to multiple events, one per data point. This works well when sending metrics to a `statsd`-type output. If sending metrics to DatadogHQ or any destination that accepts arbitrary JSON, leave toggled off (default).

Forward API key validation requests: Toggle on to send key validation requests from Datadog Agent to the Datadog API. If toggled off (the default), Stream handles key validation requests by always responding that the key is valid.

Reject unauthorized certificates: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Default is toggled on. Available when **Forward API key validation requests** is toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__agent_api_key`
- `__agent_event_type`
- `__final`

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__inputId`
- `__srcIpPort` - See details below.
- `_time`

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Datadog Agent sending data to this Source.

When any proxies (including load balancers) lie between the Datadog Agent and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Sending Datadog Agent Data to Cribl Stream

Before you begin this section, you should have Datadog Agent running on one or more hosts.

To enable Datadog Agent to send data to Cribl Stream, you'll set the following environment variables:

- `DD_DD_URL`: The URL or IP address and port of your Datadog Agent Source in Cribl Stream.
- `DD_LOGS_CONFIG_LOGS_DD_URL`: The same value as above, assuming log collection is enabled on Datadog Agent.
- `DD_LOGS_CONFIG_USE_HTTP`: Set to `true`. Cribl Stream ingests Datadog Agent logs over HTTP only; ingesting logs over TCP is not supported.

Set these environment variables in one of two ways: (1) through a `docker run` command, or (2) by editing the `datadog.yaml` configuration file that your Datadog Agent uses.

Using the `docker run` Command

Here's an example `docker run` that sets the environment variables described above, along with others required for Datadog Agent but not relevant to Cribl Stream. Replace the example values with values appropriate for your environment.

```
docker run --rm --name dd-agent
  -e DD_API_KEY=6d1ephx1w55p978i6d1ephx1w55p978i \
  -e DD_SKIP_SSL_VALIDATION=true \
  -e DD_DD_URL="http://0.0.0.0:8080" \
  -e DD_LOGS_ENABLED=true \
  -e DD_LOGS_CONFIG_LOGS_DD_URL="0.0.0.0:8080" \
  -e DD_LOGS_CONFIG_USE_HTTP=true \
  -e DD_LOGS_CONFIG_LOGS_NO_SSL=true \
  -e DD_LOGS_CONFIG_CONTAINER_COLLECT_ALL=true \
  -e DD_DOGSTATSD_NON_LOCAL_TRAFFIC="true" \
  gcr.io/datadoghq/agent:7
```

Copy

Using a Config File

Instead of using environment variables, you can set the Cribl Stream-related values in your Datadog Agent's `datadog.yaml` config file.

See the documentation links in the example config file that Datadog maintains online; and, the docs that describe filesystem locations relevant to getting Datadog Agent to configure itself with the desired `datadog.yaml` file.

Managing What Data Goes Where

The different kinds of information that Datadog Agents emit - logs, metrics, service checks, agent metadata, and APM data - are not completely independent when you send them to Cribl Stream Datadog Agent Source. The reference to "bundling" near the top of this page introduced this situation in simplified form. The table below explains the relevant constraints for each type of information that Datadog Agents emit, along with the Datadog environment variables that control them.

Type(s) of Information	Dependencies and/or Limitations	Environment Variable
Logs	Independent of other types.	`DD_LOGS_CONFIG_DD_URL`
Metrics, Events, Service Checks, and Metadata	Always sent together.	`DD_DD_URL`
APM Data (via Datadog	When configured with `DD_APM_DD_URL`, APM data can only be sent to Datadog (for example,	`DD_APM_DD_URL`

Type(s) of Information	Dependencies and/or Limitations	Environment Variable
Agent)	`https://trace.agent.datadoghq.com`) and cannot be sent to Cribl Stream, even when you send other data types here. To send APM traces to this Source, configure your Datadog tracing library (`dd-trace`) to send traces directly to Cribl Stream instead of via the Datadog Agent.	

Managing API Keys

Suppose your data flow runs from Datadog Agents, to Cribl Stream Datadog Agent Sources, to Cribl Stream Datadog Destinations, to Datadog accounts. You'll need to decide **how many** of each of these elements there are to define the data flow you want. You will also set (or override) **Datadog API keys** to support the desired data flow. For some data flows, you'll need the **General Settings** > **Allow API key from events** toggle in the Cribl Stream Datadog Destination.

Another possibility is that you want data to flow to some Destination other than a Cribl Stream Datadog Destination. If that's the case, try adapting the examples below to your use case. Please connect with us on the `Cribl Community` Slack if you have questions.

Data Flow Examples

The examples which follow start simple and become more complex in terms of desired data flow, and, consequently, of Cribl Stream configuration.

One Agent to One Account

- You're running one Datadog Agent on one host.
- You want the output of the Agent sent to a single Datadog account.
- You only need one API key.
- You configure that single API key on your Cribl Stream Datadog Destination.

Many Agents to One Account

- You're running Datadog Agents on a fleet of hosts.
- You want to consolidate all the output of the Agents into a single Datadog account.
- You only need one API key. This is no different from the simpler one-to-one scenario above.
- You configure that single API key on your Cribl Stream Datadog Destination.
- No matter which host the data originates from, your Cribl Stream Datadog Destination sends it to Datadog using that single API key.

Many Agents to Many Accounts

- You're running Datadog Agents on a fleet of hosts.
- You want the output of the Agents from some hosts to flow into one Datadog account, and the output from other hosts to flow into a different Datadog account.
    - This may need to scale up to multiple accounts, if, for example, you are managing data from several different customers, or from several different organizations within your company.
- You need one API key **for each** Datadog account.
- In the Cribl Stream Datadog Destination, toggle **General Settings** > **Allow API key from events** on.
- Here's what happens:
    - Datadog Agent always sends an API key when it communicates with the Cribl Stream Datadog Agent Source. Agents within different subgroups of hosts will send different API keys, as described above.
    - Cribl Stream Datadog Agent Source passes the API key from the Agent to the Cribl Stream Datadog Destination as an internal field.
    - Cribl Stream Datadog Destination uses that API key to direct its output to the correct Datadog account. If need be, you can configure the Destination to override the passed-in API key with a different one. This comes in handy when you know that your Agent(s) are using invalid API keys. You can even override **all** passed-in API keys.

Verifying that Data is Flowing

Once you have configured your Datadog Agent(s) as described above, and they have begun sending data:

- Try viewing the incoming data in the **Live Data** tab in your Cribl Stream Datadog Agent Source.
- If you have a Cribl Stream Datadog Destination set up, connect your Cribl Stream Datadog Agent Source to it via a **QuickConnect Passthru**, and verify that the same data shows up in the Destination's **Live Data** tab.
    - Another possibility is that you want your Cribl Stream Datadog Agent Source to connect to some Destination other than a Cribl Stream Datadog Destination. That's beyond the scope of this example, but verifying data flow should be similar.
- If the Cribl Stream Datadog Destination is configured to send data to a Datadog instance, verify that the same data shows up there, in the appropriate form, according to the transformations you've configured in Cribl Stream.
- If you have a many-to-many data flow as described above, verify that output is being correctly split among the possible Datadog accounts at the end of the data flow.

Assuming that data is behaving as expected, you can proceed to the best part, namely adding one or more Pipelines between Datadog Agent Source and Datadog Destination, to transform the data however you wish.

Given the varied nature of what Datadog Agents collect, there should be ample opportunities to make the data leaner and more usable.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

8.1.15. Elasticsearch API Source In Cribl Stream

Cribl Stream supports receiving data over HTTP/S using the Elasticsearch Bulk API.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

For examples of receiving data from popular senders via this API, see Configuring Elastic Beats below.

Looking for the Elasticsearch API Source in **Cribl Search**? See Add an Elasticsearch API Source for Cribl Search.

Configure Cribl Stream to Receive Elasticsearch Bulk API Data over HTTP(S)

Cribl Stream ships with an Elasticsearch API Source preconfigured to listen on Port 9200. You can clone or directly modify this Source to further configure it, and then enable it.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Enabled**: Toggle on to enable the Source.
    - **Input ID**: Enter a unique name to identify this Syslog Source definition. The default Source is prefilled with the value `in_elastic`. which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP on which to listen for Elasticsearch data. (For example, `localhost` or `0.0.0.0`.)
    - **Port**: Enter the port number to listen on.

- **Elasticsearch API endpoint** (for Bulk API): Absolute path on which to listen for Elasticsearch API requests. Defaults to `/`. Cribl Stream automatically appends `_bulk`, so (for example) `/myPath` becomes `/myPath/_bulk`. Requests could then be made to either `/myPath/_bulk` or `/myPath/<myIndexName>/_bulk`. Other entries are faked as success.

3. Next, you can configure the following **Optional Settings**:
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Authentication, TLS, Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

In the **Authentication type** drop-down, select one of the following options:

- **None**: Don't use authentication.
- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials. Click **Generate** if you need a new password.
- **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or select **Create** to configure a new secret.
- **Auth tokens**: Use HTTP token authentication. Click **Add Token** and, in the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header. Click **Generate** if you need a new token. Click **Add Token** to display additional rows to specify more tokens.

See also Periodic Logging for information on how auth tokens affect product logging.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.
>
> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; etc.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Enable proxy mode: If toggled on, see Proxy Mode below for the resulting options.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. By default, Cribl Stream's responses to HTTP requests include the `X-elastic-product` header, with an `Elasticsearch` value. (This is required by certain clients, including some Elastic Beats.)

API version: To upstream Elastic Beats, this Cribl Stream Source will appear as an Elasticsearch instance matching the version that you set in this drop-down:

- `6.8.4` - Retained for backward compatibility.
- `8.3.2` - This default entry matches Elasticsearch's current `8.3.x` versions.
- `Custom` - Opens an HTTP response object in the **Custom API Version** editor. This object replicates what an Elasticsearch server would send in its HTTP responses to a client. You can edit the `version : number`, and any other fields, as required to satisfy the Elastic Beat sending data to this Source. (This `Custom` option supports future Elasticsearch releases, as long as Elasticsearch keeps the same response-object structure.)

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Proxy Mode

Enabling proxy mode allows Cribl Stream to proxy non-Bulk API requests to a downstream Elasticsearch server. This can be useful when integrating with Elasticsearch API senders like Elastic Endgame agents, which send requests that Cribl Stream does not natively support. Toggle **Enable proxy mode** on to expose the following controls.

Proxy URL: URL of the Elasticsearch server that will proxy non-bulk requests, for example: `http://elastic:9200`.

Reject unauthorized certificates: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Default is toggled on.

Remove headers: Enter any headers that you want removed from proxied requests. Press `Tab` or `Return` to separate header names.

Proxy request timeout: How long, in seconds, to wait for a proxy request to complete before aborting it. Defaults to `60` seconds; minimum timeout is `1` second.

To understand how **Proxy request timeout** interacts with the `X-Forwarded-For` header, see Overriding `__srcIpPort` with Client IP/Port.

Authentication method: Select one of the following options.

- **None**: Don't use authentication.
- **Manual**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Secret**: This option exposes a **Credentials secret** drop-down, in which you can select a stored secret that references the credentials described above. A **Create** link is available to store a new, reusable secret.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Field Normalization

When ingesting data from Elasticsearch, this Source performs the following field normalization:

- Renames `@timestamp` to `_time`, maintaining millisecond precision.
- Renames `host.name` to `host`.
- Stores original `host.name` in `__host` to retain all host-related information.

The Elasticsearch Destination does the reverse, and it also recognizes the presence of `__host`.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__host`
- `__id`
- `__index`
- `__inputId`
- `__pipeline` - If present in the Elasticsearch `event.action` field of the event. See also how to override the pipeline attribute.
- `__srcIpPort` - See details below.
- `__type`

Overriding the Pipeline Attribute in the Elasticsearch and Elastic Cloud Destinations

The Elasticsearch Source will record the pipeline attribute received in a variable named `__pipeline`, if pipeline was presented in the Source event. If you want to forward the pipeline from the Source event to an Elasticsearch or Elastic Cloud Destination, set up the **Elastic pipeline** in one of the following ways.

This expression will use the value of `__pipeline` or default to `'myPipeline'` if `__pipeline` is missing:

```
__pipeline || 'myPipeline'
```

An alternative is to pass the pipeline if present, or otherwise omit it:

```
__pipeline ? __pipeline : undefined
```

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Elasticsearch client sending data to this Source.

When any proxies (including load balancers) lie between the Elasticsearch client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Configuring Elastic Beats

Beats are open-source data shippers that act as agents, sending data to Elasticsearch (or to other services, in this case Cribl Stream). The Beats most popular with Cribl users are Filebeat and Winlogbeat.

To set up a Beat to send data to Cribl Stream, edit the Beat's YAML configuration file: `filebeat.yml` for Filebeat, `winlogbeat.yml` for Winlogbeat, and so on. In the config file, you'll specify your Cribl Stream Elasticsearch Source endpoint as the Beat's Elasticsearch output. To the Beat, Cribl Stream will appear as an instance of Elasticsearch.

If you're using HTTP token authentication (which is disabled by default, both on-prem and on Cribl.Cloud): First, set the token. Then add the following to the Beat config file under `output.elasticsearch.headers`, substituting your token for `myToken42`:

```
output.elasticsearch:
  headers:
    Authorization: "myToken42"
```

Copy

Configuring an Elastic Agent

An Elastic Agent is single agent for logs, metrics, security data, and threat prevention that sends data to Elasticsearch (or to other services, in this case Cribl Stream).

When you are sending from an Elastic Agent to Cribl Stream, the `elastic-agent.yml` file doesn't pay any attention to the settings for `output.elasticsarch.allow_older_versions: true`. As a result, the Elasticsearch API Source will not get any data.

To set it up correctly, go to the **Advanced Settings** tab, and change the **API Version** to `Custom`. In the **Custom API Version** editor, edit the `version : number` to match the version of the Elastic Agent you are using. This should allow the data to start flowing to the Source.

Periodic Logging

Cribl Stream logs metrics about incoming requests and ingested events once per minute.

If one or more auth tokens are configured and enabled, Cribl Stream logs requests and events for each enabled auth token individually. Since the tokens themselves are redacted for security, Cribl Stream logs the initial text of the token description to help you identify which token a given log is for.

If no auth token is configured and enabled, Cribl Stream simply logs overall statistics about incoming requests and ingested events.

These logs are stored in the `metrics.log` file. To view them in the UI, open the Source's **Logs** tab and choose **Worker Process X Metrics** from the drop-down, where **X** is the desired Worker process.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

Dropping request because token invalid",“authToken”: “Bas...Njc=”

The specified token is invalid. Note that the above message is logged only at the debug level.

8.1.16. HTTP/S (Bulk API) Source

Cribl Stream supports receiving data over HTTP/S from Cribl Bulk API, Splunk HEC, and Elastic Bulk API endpoints.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

Configure Cribl Stream to Receive Data over HTTP(S)

Cribl Stream ships with an HTTP Source preconfigured to listen on Port 10080, and on several default endpoints. You can clone or directly modify this Source to further configure it, and then enable it.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name. The default Source is prefilled with the value `http`, which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP on which to listen for HTTP(S) data. (For example, `localhost` or `0.0.0.0`.)
    - **Port**: Enter the port number to listen on.
3. Under **Authentication**, enter the **Auth tokens**:
    - **Token**: Shared secrets to be provided by any client (`Authorization: <token>`). Click **Generate** to create a new secret. If empty, unauthorized access **will be permitted**.
    - **Description**: Optionally, enter a description.

- **Fields**: Fields to add to events referencing this token. Each field is a **Name/Value** pair, where **Value** is a JavaScript expression to compute field's value, enclosed in quotes or backticks. (Can evaluate to a constant.) For details, see Authentication Fields.

4. Next, you can configure the following **Optional Settings**:
    - **Cribl HTTP event API**: Base path on which to listen for Cribl HTTP API requests. To construct the actual endpoint, Cribl Stream will append `/_bulk` to this path. For example, with the default value of `/cribl`, your senders should send events to a `/cribl/_bulk` path. Use an empty string to disable.
    - **Elastic API endpoint** (for Bulk API): Base path on which to listen for Elasticsearch API requests. For details, see Elastic API Endpiont.
    - **Splunk HEC endpoint**: Absolute path on which to listen for Splunk HTTP Event Collector (HEC) API requests. Use an empty string to disable. Default entry is `/services/collector`. For details, see Splunk HEC Endpoint.
    - **Enable Splunk HEC acknowledgements**: Whether to enable Splunk HEC acknowledgements. Default is toggled off.
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Elastic API Endpoint

Currently, the only supported option is the default `/elastic`, to which Cribl Stream will append `/_bulk`. So, your senders should send events to an `/elastic/_bulk` path. Other entries are faked as success. Use an empty string to disable.

Cribl generally recommends that you use the dedicated Elasticsearch API Source instead of this endpoint. The Elastic API implementation here is provided for backward compatibility, and for users who want to ingest multiple inputs on one HTTP/S port.

Splunk HEC Endpoint

This Splunk HEC implementation is an **event** (such as, not **raw**) endpoint. For details, see Splunk's documentation. To send data to it from a HEC client, use either `/services/collector` or `/services/collector/event`. (See the examples below.)

The system default Event Breaker is automatically applied, which enforces a limit of 51,200 bytes per event.

Cribl generally recommends that you use the dedicated Splunk HEC Source instead of this endpoint. The Splunk HEC implementation here is provided for backward compatibility, and for users who want to ingest multiple

inputs on one HTTP/S port.

Authentication Fields

By default, fields specified here will override fields of the same name in incoming events. However, you can customize this behavior to allow event fields to take precedence. For example, to allow the event's `index` field to take precedence, you could use this expression: `` `event.index || 'defaultIndex'` ``.

Fields add context to events based on the token used. This can be useful for tagging events with metadata such as source or environment, enhancing security by avoiding direct use of authorization values in Routes and Pipelines.

See also Periodic Logging for information on how auth tokens affect product logging.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.
>
> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

> `Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may

result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.

- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; and so forth.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.

- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__inputId`
- `__hecToken`
- `__srcIpPort` - See details below.
- `__host` (Elastic In)
- `__id` (Elastic In)

- `__index` (Elastic In)
- `__type` (Elastic In)

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the client sending data to this Source.

When any proxies (including load balancers) lie between the HTTP client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original HTTP client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Format and Endpoint

Cribl Stream expects HTTP(S) events to be formatted as one JSON record per event. Here are two event records:

Sample Event Format

```
{"_time":1541280341, "_raw":"this is a sample event ", "host":"myHost", "source":"myS
{"_time":1541280341, "host":"myOtherHost", "source":"myOtherSource", "_raw": "{\"mess
```

Copy

Note 1: Events can be sent as separate POSTs, but Cribl **highly** recommends combining multiple events in newline-delimited groups, and POSTing them together.

Note 2: If an HTTP(S) source is routed to a Splunk destination, fields within the JSON payload are mapped to Splunk fields. Fields that do not have corresponding (native) Splunk fields become index-time fields. For example, let's assume we have a HTTP(S) event like this:

```
{"_time":1541280341, "host":"myHost", "source":"mySource", "_raw":"this is a sample event ", "fieldA":"valueA"}
```

Here, `_time`, `host` and `source` become their corresponding fields in Splunk. The value of `_raw` becomes the actual body of the event, and `fieldA` becomes an index-time field. (`fieldA::valueA`).

Examples

The examples in this section demonstrate sending HTTP data into a Bulk HTTP Source configured to listen on port `10080`. The token is set to `myToken42` and included in the `Authorization` header of requests.

Cribl Endpoint - Single Event

Cribl Endpoint, Single Event Example

```
curl -k "http://<myCriblHost>:10080/cribl/_bulk" \
-H "Authorization: myToken42" \
-d '{"_raw":"this is a sample event ", "host":"myHost", "source":"mySource", "fieldA'
```

Copy

Cribl Endpoint - Multiple Events

Cribl Endpoint, Multiple Events Example

```
curl -k "http://<myCriblHost>:10080/cribl/_bulk" \
-H "Authorization: myToken42" \
-d $'{"_raw":"this is a sample event ", "host":"myHost", "source":"mySource", "fieldA
{"_raw":"this is another sample event ", "host":"myOtherHost", "source":"myOtherSourc
```

Copy

Splunk HEC Event Endpoints

For Splunk HEC, the token specification can be either `Splunk <token>` or `<token>`.

Splunk HEC Event Endpoint, /services/collector/event Example

```
curl -k "http://<myCriblHost>:10080/services/collector/event" \
-H "Authorization: myToken42" \
-d '{"event":"this is a sample event ", "host":"myHost", "source":"mySource", "fieldA
```

Copy

Splunk HEC Event Endpoint, /services/collector Example

```
curl -k "http://<myCriblHost>:10080/services/collector" \
-H "Authorization: myToken42" \
-d '{"event":"this is a sample event ", "host":"myHost", "source":"mySource", "fieldA
```

Copy

Cribl.Cloud - Single Event

Replace `<organizationId>` with your Cribl.Cloud Organization ID.

Cribl.Cloud, Single Event Example

```
curl -k "https://default.main.<organizationId>.cribl.cloud:10080/cribl/_bulk" \
-H "Authorization: myToken42" \
-d '{"_raw":"this is a sample event ", "host":"myHost", "source":"mySource", "fieldA'
```

Copy

For Cribl.Cloud Enterprise plans, replace `default` in the request URL with the name of the Worker Group that you want to send to.

Periodic Logging

Cribl Stream logs metrics about incoming requests and ingested events once per minute.

If one or more auth tokens are configured and enabled, Cribl Stream logs requests and events for each enabled auth token individually. Since the tokens themselves are redacted for security, Cribl Stream logs the initial text of the token description to help you identify which token a given log is for.

If no auth token is configured and enabled, Cribl Stream simply logs overall statistics about incoming requests and ingested events.

These logs are stored in the `metrics.log` file. To view them in the UI, open the Source's **Logs** tab and choose **Worker Process X Metrics** from the drop-down, where **X** is the desired Worker process.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.17. Raw HTTP/S Source In Cribl Stream

Cribl Stream supports receiving raw HTTP data. The Raw HTTP Source listens on a specific port, captures every HTTP request to that port, and creates a corresponding event that it pushes to its configured Event Breakers.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

Looking for the Raw HTTP Source in **Cribl Search**? See Add a Raw HTTP Source for Cribl Search.

Configure Cribl Stream to Receive Raw HTTP Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Raw HTTP Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the address to bind on. Defaults to `0.0.0.0` (all addresses).
    - **Port**: Enter the port number to listen on. For Cribl.Cloud, use the `20000 - 20010` port range.
3. Under **Authentication**, enter the **Auth tokens**:
    - **Token**: Shared secrets to be provided by any client. Click **Generate** to create a new secret. If empty, permits open access.
    - **Description**: Optionally, enter a description.
    - **Fields**: Fields to add to events referencing this token. Each field is a **Name/Value** pair, where **Value** is a JavaScript expression to compute field's value, enclosed in quotes or backticks. (Can evaluate to a constant.) For details, see Authentication Fields.
4. Next, you can configure the following **Optional Settings**:

- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Authentication Fields

By default, fields specified here will override fields of the same name in incoming events. However, you can customize this behavior to allow event fields to take precedence. For example, to allow the event's `index` field to take precedence, you could use this expression: `` `event.index || 'defaultIndex'` ``.

Fields add context to events based on the token used. This can be useful for tagging events with metadata such as source or environment, enhancing security by avoiding direct use of authorization values in Routes and Pipelines.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the

Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Event Breakers

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Allowed URI paths: List of URI paths accepted by this input. Supports wildcards, for example, `/api/v*/hook`. Defaults to `*`, which allows all paths.

Allowed HTTP methods: List of HTTP methods accepted by this input. Supports wildcards, for example, `P*`, `GET`. Defaults to `*`, which allows all methods.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; etc.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and functions can use them to make processing decisions.

Fields for this Source:

- `__channel`
- `__headers` - Automatically includes any headers sent with request.
- `__inputId`
- `__srcIpPort` - See details below.

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the HTTP client sending data to this Source.

When any proxies (including load balancers) lie between the HTTP client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Examples

The examples in this section demonstrate sending HTTP data into a Raw HTTP Source. The token is set to `myToken42` and included in the `Authorization` header of requests.

Cribl Endpoint - Single Event

Replace `<myCriblHost>` with your **Public Ingress address**.

Cribl Endpoint, Single Event Example

```
curl -k "http://<myCriblHost>:10050/postPath" \
-H "Authorization: myToken42" \
-H "Content-Type: application/json" \
-d '{"_raw":"this is a sample event", "host":"myHost", "source":"mySource", "fieldA":
```

Copy

Cribl Endpoint - Multiple Events

Replace `<myCriblHost>` with your Public Ingress address.

Cribl Endpoint, Multiple Events Example

```
curl -k "http://<myCriblHost>:10050/postPath" \
-H "Authorization: myToken42" \
-H "Content-Type: application/json" \
-d $'{"_raw":"this is a sample event", "host":"myHost", "source":"mySource", "fieldA'
{"_raw":"this is another sample event", "host":"myOtherHost", "source":"myOtherSource
```

Copy

Cribl.Cloud - Single Event

Replace `<Your-Org-ID>` with your Cribl.Cloud Organization ID. Cribl.Cloud makes the `20000` - `20010` port range available for configuring Sources that don't have a default port, like the Raw HTTP Source.

Cribl.Cloud, Single Event Example

```
curl -k "http://default.main.<Your-Org-ID>.cribl.cloud:20001" \
-H "Authorization: myToken42" \
-d '{"_raw":"this is a sample event"}'
```

Copy

For Cribl.Cloud Enterprise plans, replace `default` in the request URL with the name of the Worker Group that you want to send to.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

8.1.18. Metrics Source

Cribl Stream supports receiving metrics in these wire formats/protocols: StatsD, StatsD Extended, and Graphite. Automatic protocol detection happens on the first line received over a TCP connection or a UDP packet. Lines not matching the detected protocol are dropped.

Type: **Push** | TLS Support: **No** | Event Breaker Support: **No**

Cribl Edge Workers support System Metrics only when running on Linux, not on Windows.

Configure Cribl Stream to Receive Metrics

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP to listen to. Defaults to `0.0.0.0`.
    - **UDP port**: Enter the UDP port number to listen on. Not required if listening on TCP.For detail, see UDP Port Considerations.
    - **TCP port**: Enter the TCP port number to listen on. Not required if listening on UDP.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

UDP Port Considerations

Sending large numbers of UDP events per second can cause Cribl.Cloud to drop some of the data. This results from restrictions of the UDP protocol. To minimize the risk of data loss, deploy a customer-managed (hybrid) Stream Worker Group with Worker Nodes as close as possible to the UDP senders. Cribl also recommends tuning the OS UDP buffer size.

TLS Settings (TCP Only)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always On`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable Proxy Protocol: Toggle on if the connection is proxied by a device that supports Proxy Protocol v1 or v2.

IP allowlist regex: Regex matching IP addresses that are allowed to send data. Defaults to `.*` (i.e., all IPs.)

Buffer size limit (events): Maximum number of events to buffer when downstream is blocking. Defaults to `1000`.

UDP socket buffer size (bytes): Optionally, set the SO_RCVBUF socket option for the UDP socket. This value tells the operating system how many bytes can be buffered in the kernel before events are dropped. Leave blank to use the OS default. Min: `256`. Max: `4294967295`.

It may also be necessary to increase the size of the buffer available to the `SO_RCVBUF` socket option. Consult the documentation for your operating system for a specific procedure.

Setting this value will affect OS memory utilization.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__srcIpPort`
- `__metricsInType`

Metric Event Schema and Destination Support

Metric data is read into the following event schema:

```
_metric - the metric name
_metric_type - the type of the metric (gauge, counter, timer)
_value - the value of the metric
_time - metric_time or Date.now()/1000
dim1 - value of dimension1
dim2 - value of dimension2
....
```

Copy

Cribl Stream places sufficient information into a field called `__criblMetric` to enable these events to be properly serialized out to any metric outputs (independent of the input type).

The following Destinations natively support the `__criblMetric` field:

- Splunk
- Splunk HEC
- InfluxDB
- Statsd
- Statsd Extended
- Graphite

Data Format/Protocol Examples

StatsD

Format: `MetricName:value|type`

StatsD Example

```
metric1:100|g
metric2:200|ms
metric.dot.3:300.16|c
```

Copy

See the StatsD repo.

StatsD Extended

Format: `MetricName:value|type|#dim=value,dim2=value`

StatsD Extended Example

```
metric1:100|g|#dim1:val1,dim2:val2,dim3:val3
metric2:200|ms|#dim1:val1,dim2:val2,dim3:val3
metric.dot.3:300.16|c|#dim1:val1,dim2:val2,dim3:val3
```

Copy

Graphite

Format: `MetricName[;dim1=val1[;dim2=val2]] value time`

Graphite Example with Dimensions

```
metric1;dim1=val1;dim2=val2 100 9999
metric2;dim1=val1;dim2=val2 200 9999
metric.dot.3;dim1=val1;dim2=val2 300.16 9999.16
```

Copy

Graphite Example without Dimensions

```
metric1 100 9999
metric2 200 9999
metric.dot.3 300.16 9999.16
```

Copy

See the Graphite (also known as Carbon) plaintext protocol.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

8.1.19. Model Driven Telemetry Source

Cribl Stream supports receiving network device metrics and events via Model Driven Telemetry (MDT).

Type: **Push** | TLS Support: **Yes** | Event Breaker Support: **No**

For observability of network devices, MDT compares favorably with the older SNMP-based approach because:

- MDT's messaging granularity is much finer than SNMP's.
- MDT's push-based model reduces load on network devices, while SNMP's pull-based model queues requests on devices.
- MDT's extensive ecosystem of data modules enables devices to collect many more kinds of data than they could with SNMP tools.

Prerequisites

When you configure MDT on a network device you choose among a variety of options and techniques. For this Source (in Cribl Stream version 4.5), you must configure your upstream sending device to use the following:

- MDT in dial-out mode.
- gRPC for the data transport mechanism.
- YANG models for data representation.
- Self-describing, also known as Key/Value Protocol Buffers (protobufs) for data encoding.

Configure Cribl Stream to Receive Model Driven Telemetry Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources**. Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.

- **Description**: Optionally, enter a description.
- **Address**: Enter the hostname/IP to listen to. Defaults to `0.0.0.0`.
- **Port**: Enter the port number to listen on. Defaults to port `57000`.

3. Under **Authentication**, enter the **Auth tokens**:
   - **Auth tokens**: Shared secrets to be provided by any client. Click **Generate** to create a new secret. If empty, permits open access.
   - **Description**: Optionally, enter a description.
   - **Fields**: Fields to add to events referencing this token. Each field is a **Name/Value** pair, where **Value** is a JavaScript expression to compute field's value, enclosed in quotes or backticks. (Can evaluate to a constant.) For details, see Authentication Fields.
4. Next, you can configure the following **Optional Settings**:
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter:

`worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may

result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.

- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Active connection limit: Maximum number of active connections allowed per Worker Process. Defaults to `1000`. Set a lower value if connection storms are causing the Source to hang. Set `0` for unlimited connections.

Shutdown timeout: Time to allow the server to shut down gracefully before forcing shutdown. Defaults to `5000` milliseconds (5 seconds).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of data from this Source via the Routing table.

Select **QuickConnect** to send data from this Source to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__srcIpPort`

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.20. NetFlow & IPFIX Source

Cribl Stream supports receiving NetFlow v5, v9, and IPFIX (v10) data via UDP.

Type: **Push** | TLS Support: **No** | Event Breaker Support: **No**

This Source ingests NetFlow records similarly to how it ingests events from other upstream senders: fields are broken out, and each record includes a message header. If you prefer to render NetFlow or IPFIX data as metrics, use a pre-processing Pipeline or a Route.

If you want to associate a Community ID with a network flow that passes through this Source, connect the Source to a Route containing a Function that supports expression evaluation; then, in the Function, use the C.Net.communityIDv1() method. You can even generate different CommunityIDs for different use cases within the same network flow - to do this, use multiple `communityIDv1()` methods, with the optional `seed` parameter.

Sending large numbers of UDP events per second can cause Cribl.Cloud to drop some of the data. This results from restrictions of the UDP protocol.

To minimize the risk of data loss, deploy a hybrid Stream Worker Group with customer-managed Worker Nodes as close as possible to the UDP sender. Cribl also recommends tuning the OS UDP buffer size.

Configure a NetFlow & IPFIX Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source.
    - **Description**: Optionally, enter a description.

- **Address**: Enter the hostname/IP on which to listen for data. For example: `localhost`, `0.0.0.0`, or `::`.
- **Port**: Enter the port number.
- **Enable pass-through**: Enable this setting to allow Cribl Stream to forward events to a NetFlow Destination. If enabled, Cribl Stream generates an additional event containing `__netflowRaw`, which directly routes the raw NetFlow packet data to a downstream NetFlow Destination. These additional events do not count against any ingest quotas.
- **Version Support**: Toggle which versions of NetFlow and IPFIX (NetFlow v10) data are processed. By default, all versions (`V5`, `V9`, and `IPFIX`) are enabled. You can disable support for specific versions if needed. Disabling a version prevents the Source from processing data for that version and treats any received packets of that version as unsupported messages.

3. Next, you can configure the following **Optional Settings**:
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Persistent Queue, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.
>
> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` (that is, all IPs).

IP denylist regex: Regex matching IP addresses whose messages you want this Source to ignore. Defaults to `^$` (that is, every specific IP address in the list). This takes precedence over the allowlist.

UDP socket buffer size (bytes): Optionally, set the `SO_RCVBUF` socket option for the UDP socket. This value tells the operating system how many bytes can be buffered in the kernel before events are dropped. Leave blank to use the OS default. Minimum: `256`. Maximum: `4294967295`.

It may also be necessary to increase the size of the buffer available to the `SO_RCVBUF` socket option. Consult the documentation for your operating system for a specific procedure.

Setting this value will affect OS memory utilization.

Template cache minutes (minutes): This setting only affects v9 flow records. Use this setting to specify the length of time (in minutes) that Cribl Stream caches NetFlow v9 templates before discarding them. For

example, setting the duration to `30` means Cribl Stream will cache templates for 30 minutes. If flow records are not refreshed by a new event, the templates are discarded.

The template cache minutes setting helps optimize the performance and reliability of NetFlow v9 data processing. This setting manages memory usage and ensures that outdated templates do not persist indefinitely. The goal is to balance system performance against memory use. Monitor your network and adjust this setting to find the right balance between template availability and system data overload. Some considerations when tuning your system:

- If your network's data patterns change often, consider using a shorter cache duration. Conversely, if the data records are stable and change less frequently, try using a longer duration.
- Monitor the status of this Source across your Worker/Edge nodes. If you notice Cribl Stream discards FlowSets, it may mean the cache duration is too short. Try increasing the cache duration and then observer whether there is a performance improvement. Alternatively, adjust your NetFlow exporters to send templates more frequently.
- If your system is using too much memory, it may mean that your cache duration is too long, especially if you have a lot of NetFlow exporters sending data to a single Source.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

What Fields to Expect

The NetFlow Source does minimal processing of the incoming UDP messages to remain consistent with the internal Cribl event model. For each UDP message received on the socket, you can expect an event with the following fields.

Field Representation for NetFlow v9 & IPFIX (v10)

For both NetFlow v9 and IPFIX (v10), fields map to their most natural representation in Cribl. Specifically:

- IPv4 addresses are represented as dotted octet strings.
- IPv6 as colon-separated octet strings.
- MAC addresses also as colon-separated strings.
- Numbers that are six bytes or less are represented as numbers.

- Numbers that are greater than six bytes are represented as strings.
- Registered field attribute names are the camelCase versions of the SCREAMING_CASE names in the Cisco whitepaper.

Cribl Stream collects and offers unregistered fields in an array of records consisting of their type codes and byte buffers. You can add Pipeline functions to change how these fields are processed by default.

Fields for NetFlow IPFIX (v10)

IPFIX fields are derived from the IANA IPFIX Information Elements list. These fields are predefined and included in the `ipfix-information-elements.yml` configuration file.

Fields for NetFlow v9

We have organized these fields into categories to make them easier to grasp; the categories are our own, and not from the NetFlow specifications. The field definitions are mostly copied from Cisco NetFlow documentation.

Not all NetFlow v9 fields will appear on all events, only those that are specified by the template for that specific data record. These fields are available for NetFlow v9:

- `inBytes`: The number of incoming Bytes.
- `inPkts`: The number of incoming packets.
- `flows`: The number of flows that were aggregated.
- `protocol`: The IP protocol.
- `srcTos`: The type of service setting when entering the incoming interface.
- `tcpFlags`: The accumulation of all the TCP flags seen in this flow.
- `l4SrcPort`: The TCP/UDP source port number.
- `ipv4SrcAddr`: The IPv4 source address.
- `srcMask`: The number of contiguous bits in the source subnet mask.
- `inputSnmp`: The input interface index.
- `l4DstPort`: The TCP/UDP destination port number.
- `ipv4DstAddr`: The IPv4 destination address.
- `dstMask`: The number of contiguous bits in the destination subnet mask.
- `outputSnmp`: The output interface index.
- `ipv4NextHop`: The IPv4 address of the next-hop router.
- `srcAs`: The source BGP autonomous system number. The length may be and defaults to 2.
- `dstAs`: The destination BGP autonomous system number. The length may be and defaults to 2.

- `bgpIpv4NextHop`: The IPv4 address of the next-hop router in the bgp domain.
- `mulDstPkts`: The number of IP multicast outgoing packets.
- `mulDstBytes`: The number of IP multicast outgoing bytes.
- `lastSwitched`: The system uptime in milliseconds at which the last packet of this flow was switched.
- `firstSwitched`: The system uptime in milliseconds at which the first package of this flow was switched.
- `outBytes`: The number of outgoing bytes.
- `outPkts`: The number of outgoing packets.
- `minPktLngth`: The minimum IP packet length on incoming packets.
- `maxPktLngth`: The maximum IP packet length on incoming packets.
- `ipv6SrcAddr`: The IPv6 source address.
- `ipv6DstAddr`: The IPv6 destination address.
- `ipv6SrcMask`: The number of contiguous bits in the IPv6 source mask.
- `ipv6DstMask`: The number of contiguous bits in the IPv6 destination mask.
- `ipv6FlowLabel`: The IPv6 flow label as per the RFC 2460 definition.
- `icmpType`: The ICMP packet type, reported as type * 256 + code.
- `mulIgmpType`: The IGMP packet type.
- `samplingInterval`: The rate at which packets are sampled; a value of 100 indicates that 1% of packets are sampled.
- `samplingAlgorithm`: The platform sampling algorithm; 1 is deterministic, 2 is random.
- `flowActiveTimeout`: Timeout value in seconds for active flow entries in the NetFlow cache.
- `flowInactiveTimeout`: Timeout value in seconds for inactive flow entries in the NetFlow cache.
- `engineType`: The type of flow switching engine; 0 is RP, 1 is VIP/Linecard.
- `engineId`: The ID number of the flow switching engine.
- `totalBytesExp`: The number of bytes exported by the observation domain.
- `totalPktsExp`: The number of packets exported by the observation domain.
- `totalFlowsExp`: The number of flows exported by the observation domain.
- `ipv4SrcPrefix`: The IPv4 source address prefix, specific for Catalyst architecture.
- `ipv4DstPrefix`: The IPv4 destination address prefix, specific for Catalyst architecture.
- `mplsTopLabelType`: The MPLS top label type; 0 is unknown, 1 is TE midpoint, 2 is AToM, 3 is VPN, 4 is BGP, and 5 is LDP.
- `mplsTopLabelIpAddr`: The IPv4 address of the forwarding equivalent class corresponding to the MPLS top label.

- `flowSamplerId`: The ID number of the flow sampler.
- `flowSampleMode`: The type of algorithm used for sample data.
- `flowSamplerRandomInterval`: The packet interval at which to sample.
- `minTtl`: Minimum TTL on incoming packets.
- `maxTtl`: Maximum TTL on incoming packets.
- `ipv4Ident`: The IPv4 identification field.
- `dstTos`: The type of service byte setting when exiting the outgoing interface.
- `inSrcMac`: The incoming source MAC address.
- `outDstMac`: The outgoing destination MAC address.
- `srcVlan`: The virtual LAN ID associated with the ingress interface.
- `dstVlan`: The virtual LAN ID associated with the egree interface.
- `ipProtocolVersion`: The IP version. The default is assumed to be 4 if unspecified.
- `direction`: The flow direction; 0 is ingress, 1 is egress.
- `ipv6NextHop`: The IPv6 address of the next-hop router.
- `bgpIpv6NextHop`: The IPv6 address of the next-hop router in the bgp domain.
- `ipv6OptionHeaders`: The IPv6 option headers found in the flow.
- `mplsLabel1`: The MPLS label at position 1 in the stack.
- `mplsLabel2`: The MPLS label at position 2 in the stack.
- `mplsLabel3`: The MPLS label at position 3 in the stack.
- `mplsLabel4`: The MPLS label at position 4 in the stack.
- `mplsLabel5`: The MPLS label at position 5 in the stack.
- `mplsLabel6`: The MPLS label at position 6 in the stack.
- `mplsLabel7`: The MPLS label at position 7 in the stack.
- `mplsLabel8`: The MPLS label at position 8 in the stack.
- `mplsLabel9`: The MPLS label at position 9 in the stack.
- `mplsLabel10`: The MPLS label at position 10 in the stack.
- `inDstMac`: The incoming destination MAC address.
- `outSrcMac`: The outgoing source MAC address.
- `ifName`: The shortened interface name.
- `ifDesc`: The full interface name.
- `samplerName`: The name of the flow sampler.

- `inPermanentBytes`: The running byte counter for a permanent flow.
- `inPermanentPkts`: The running packet counter for a permanent flow.
- `fragmentOffset`: The fragment-offset value from fragmented IP packets.
- `forwardingStatus`: The forwarding status.
- `mplsPalRd`: The MPLS pal route distinguisher.
- `mplsPrefixLen`: The number of consecutive bits in the MPLS prefix length.
- `srcTrafficIndex`: The BGP policy accounting source traffic index.
- `dstTrafficIndex`: The BGP policy accounting destination traffic index.
- `applicationDescription`: The application description.
- `applicationTag`: The application tag.
- `applicationName`: The name associated with a classification.
- `postIpDiffServeCodePoint`: The value of a differentiated services code point encoded in the differentiated services field, after modification.
- `replicationFactor`: The multicast replication factor.
- `layer2PacketSectionOffset`: The layer 2 packet section offset, potentially a generic offset.
- `layer2PacketSectionSize`: The layer 2 packet section size, potentially a generic size.
- `layer2PacketSectionData`: The layer 2 packet section data.

Option Data Records for NetFlow v9

Options data records report aggregate metadata about the Exporters. Like data records, they can include any or all of the previous fields. Additionally, they also have a `scopes` field with a value containing an array of records with a `type` field. The valid types are:

- `system`
- `interface`
- `line-card`
- `cache`
- `template`

Option data records also have a `value` field, which is a natural number. The JavaScript type is either a number or a string, depending on the size of the value.

Fields for NetFlow v5

General

- `_time`: Timestamp in Unix time (seconds) at which the message was received by Cribl Stream.
- `source`: A string in the form `udp|<remote IP address>|<remote port>`, indicating the remote sender.
- `host`: The hostname of the machine running Cribl Stream that ingested this event.
- `inputInt`: SNMP index of input interface; always set to zero.
- `outputInt`: SNMP index of output interface.
- `protocol`: IP protocol type (for example, TCP = 6; UDP = 17) of the observed network flow.
- `tcpFlags`: Cumulative logical OR of TCP flags in the observed network flow.
- `tos`: IP type of service; switch sets it to the ToS of the first packet of the flow.
- `icmpType`: Type of the ICMP message. Only present if the protocol is ICMP.
- `icmpCode`: Code of the ICMP message. Only present if the protocol is ICMP.

Because this Source reads input data directly from bytes in a compact format, there is nothing suitable to put in a `_raw` field, and the Source does not add a `_raw` field to events.

Flow Source and Destination

- `srcAddr`: Source IP address; in case of destination-only flows, set to zero.
- `dstAddr`: Destination IP address.
- `nextHop`: IP address of next hop router.
- `srcPort`: TCP/UDP source port number.
- `dstPort`: TCP/UDP destination port number. If this is an ICMP flow, this field is a combination of ICMP type and ICMP code, which are broken out separately as `icmpType` and `icmpCode` fields.
- `srcMask`: Source address prefix mask bits.
- `dstMask`: Destination address prefix mask bits.
- `srcAs`: Autonomous system number of the source, either origin or peer.
- `dstAs`: Autonomous system number of the destination, either origin or peer.

Flow Statistics

- `packets`: Packets in the flow.
- `octets`: Total number of Layer 3 bytes in the packets of the flow.
- `startTime`: System uptime, in milliseconds, at the start of the flow.

- `endTime`: System uptime, in milliseconds, at the time the last packet of the flow was received.
- `durationMs`: `endTime` minus `startTime`.

Header

The following are subfields within the `header` field:

- `count`: Number of flows exported in this flow frame (protocol data unit, or PDU).
- `sysUptimeMs`: Current time in milliseconds since the export device booted.
- `unixSecs`: Current seconds since 0000 UTC 1970.
- `unixNsecs`: Residual nanoseconds since 0000 UTC 1970.
- `flowSequence`: Sequence counter of total flows seen.
- `engineType`: Type of flow switching engine.
- `engineId`: ID number of the flow switching engine.
- `samplingMode`: The first two bits of what Cisco NetFlow documentation calls `SAMPLING_INTERVAL`. These bits specify a sampling mode.
- `samplingInterval`: The remaining 14 bits of what Cisco NetFlow documentation calls `SAMPLING_INTERVAL`. These bits specify a sampling interval.
- `version`: NetFlow export format version number.

Also, the internal fields listed below will be present.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and functions can use them to make processing decisions.

Fields accessible for this Source:

- `__netflowRaw` : Directly routes the raw NetFlow packet data to a downstream NetFlow Destination.

Considerations for Working with NetFlow Data FlowSets

For both NetFlow v5 and v9, Cribl Stream:

- Can forward NetFlow packets to other NetFlow destinations. However, it cannot modify the the contents of the incoming packet. In other words, Cribl Stream forwards the packets verbatim as they came in.
- Only routes NetFlow packets from upstream exporters and cannot generate its own NetFlow packets.
- Cannot forward non-NetFlow input data to NetFlow Destinations.

- Can forward NetFlow packets to non-NetFlow Destinations, such as Splunk, Syslog, S3, and others. To send NetFlow flow records to other non-NetFlow Destinations, use the other events generated on the NetFlow & IPFIX Source that do not include the `__netflowRaw` field. NetFlow Destinations discard those events.

Cribl Stream emits:

- Decoded Cribl events downstream for each data record and options data record.
- When you enable pass-through mode: an undecoded Cribl event for the entire packet.

Leader and Worker Node Interactions

The NetFlow & IPFIX Source uses the Leader Node to keep all Worker Nodes in sync with NetFlow v9 and IPFIX template information. If the Leader is unavailable, Worker Nodes can still process data using the templates they already have. However, new or updated templates for v9 flows will only be learned by the specific Worker Node receiving that traffic. Other Worker Nodes won't get these updates and will drop any flowsets that use those new templates.

High Availability and Disaster Recovery for NetFlow v9

NetFlow v9 Template Records are sent every 20 packets or every 30 minutes. Cribl Stream uses a key-value store to share these records across Worker Nodes in a Worker Group, ensuring availability for processing FlowSets.

- **Template Sharing**: Records are shared across Worker Nodes with a minimal delay, ensuring timely processing. The brief interval between receiving and sharing Template Records is designed to be negligible, given the low rate of Template Record changes.
- **Worker Coordination**: The key-value store allows uninterrupted processing during Worker Node failures.
- **Disaster Recovery**: Replication in the key-value store ensures Template Records are preserved, enabling seamless recovery.

UDP Tuning

Incoming UDP traffic is put into a buffer by the Linux kernel. Cribl will drain from that buffer as resources are available. At lower throughput levels, and with plenty of available processes, this isn't an issue. As you scale up, however, the default size of that buffer may be too small.

You can check the current buffer size with:

```
$ sysctl net.core.rmem_max
```

Copy

A typical value of about 200 KB is far too small for an even moderately busy syslog server. You can check the health of UDP with the following command. Check the `packet receive errors` line.

```
$ netstat -su
```

Copy

If `packet receive errors` is more than zero, you have lost events, which is a particularly serious problem if the number of errors is increasing rapidly. This means you need to increase your `net.core.rmem_max` setting (see earlier).

You can update the live settings, but you'll also need to change the boot-time setting so next time you reboot everything is ready to roll.

Live change, setting to 25 MB:

```
$ sysctl -w net.core.rmem_max=26214400
net.core.rmem_max = 26214400
$ sysctl -w net.core.rmem_default=26214400
net.core.rmem_default = 26214400
```

Copy

For the permanent settings change, add the following lines to `/etc/sysctl.conf`:

```
net.core.rmem_max=26214400
net.core.rmem_default=26214400
```

Copy

We recommend you track a few things related to UDP receiving:

- The `netstat -su` command, watching for errors.
- The **Status** tab in the UDP (Raw) Source. In particular, watch for dropped messages. They could indicate you need a bigger buffer under **Advanced Settings** (default: 1000 events). They could also indicate your Worker is encountering pressure further down the Pipeline.
- Especially if you increase your kernel receive buffer as above, watch your Worker processes' memory usage.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Data Collisions

When Cribl Stream receives a NetFlow v9 template, it identifies the template using three values:

- Template ID
- Source ID, which is unique to the upstream exporting device
- IP address of the upstream exporting device

Cribl Stream uses these three values to prevent data collisions. However, data collisions may occur under these two conditions:

- Your system has a heterogeneous variety of upstream exporter devices, AND
- The Exporter IP addresses are rewritten (such as by a proxy server) before they are sent downstream to Cribl Stream.

In that situation, it is possible for two exporter devices with the same IP address to send packets simultaneously and cause a collision. The workaround is to create unique IP addresses for the exporters or use the original IP addresses.

8.1.21. OpenAI Source

The OpenAI Source in Cribl Stream collects model invocation logs, audit logs, and other organization-level telemetry from the OpenAI platform. This lets you bring OpenAI activity into your pipelines for AI usage governance, security auditing, and cost/usage analysis.

Type: **Pull** | TLS Support: **Yes** | Event Breaker Support: **No**

TLS is enabled via HTTPS on this Source's underlying OpenAI REST APIs.

Prerequisites

Before you configure an OpenAI Source, you need:

- A Cribl Stream deployment (Cloud or self-hosted).
- An OpenAI organization with access to the relevant organization-level APIs (such as usage and audit logs).
- An OpenAI organization admin key (API key or equivalent) with permissions to call the endpoints you plan to enable. Visit OpenAI's organization admin keys page to create an organization admin key.
- Network connectivity from your Cribl Workers to the OpenAI API over HTTPS (direct or via an HTTP/S proxy).

For details on generating and managing OpenAI credentials, see the OpenAI API documentation.

How the OpenAI Source Works

When you enable specific content types (endpoints), the Source:

1. Polls each enabled OpenAI endpoint on its own schedule.
2. Uses a shared OpenAI organization auth token for all endpoint calls.
3. Tracks state independently for each endpoint.
4. Emits events into Cribl Stream, where you can route, filter, and enrich them before forwarding to Destinations.

Configuring an OpenAI Source

1. From the top nav, click **Manage**, then select a Worker Group to configure. Next, you have two options:
    - To configure via the graphical QuickConnect UI, click **Routing** then **QuickConnect**. Next, click **Add Source** and from the resulting drawer's tiles, select **OpenAI**. Next, click either **Add Source** or (if

displayed) **Select Existing**.

- To configure via the Routing UI, click **Data** then **Sources**. Select **OpenAI** from the list of tiles or the Sources left nav. Next, click **Add Source** to open the **Add Source** modal.

2. In the Source modal, configure the following under **General Settings**:
   - **Input ID**: Enter a unique name to identify this OpenAI Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
   - **Desctiption**: Optionally, enter a description.
   - **API Key**: Select or create a stored API key.
   - **OpenAI Organization**: Optionally, enter the `OpenAI-Organization` request header value. Typically `org-xxxxxxxxxxxxxxxxxxxxxxxx`.
   - **OpenAI Project**: Optionally, enter the `OpenAI-Project` request header value. Typically `proj_xxxxxxxxxxxxxxxxxxxxxxxx`.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
3. Under **Content Types**, enable the endpoints you'd like to collect data from. See the available Content Types below.
4. Optionally, configure any Processing Settings, Retries, and Advanced Settings.
5. Click **Save**, then **Commit & Deploy**.

Supported Content Types (#content-types)

The OpenAI Source exposes up to 10 OpenAI endpoints as individually toggleable content types. These content types include:

The available Content types are:

- Audit Logs
- Costs
- Users
- Projects
- Completions Usage Details
- Embeddings Usage Details
- Moderations Usage Details
- Images Usage Details
- Audio Speeches Usage Details
- Audio Transcriptions Usage Details

Each content type uses the same controls:

Enable content: Turn collection for that endpoint on or off.

Details: Add or edit query parameters to control what each request returns.

Pagination: Choose a pagination type and specify response attributes and any last-page expression.

Schedule: Adjust how often Cribl Stream polls that endpoint. See Scheduling.

State tracking: Customize how the Source tracks time or cursor state to avoid overlaps and gaps between jobs. See State Tracking.

Fields: Add fixed fields (for example, `env`, `tenant`, or a business-unit tag) that are attached to every event from that endpoint.

The **Scheduling** and **Query Parameters** are automatically configured for you.

Processing Settings

Fields: You can add Fields to each event, using Eval-like functionality. A field consists of a **Name** and Value pair. The **Value** is a JavaScript expression and can be a constant.

Pre-Processing: Select a **Pipeline** to process results before sending to Routes. Optional, and available only when **Send to Routes** is toggled on.

Fields

In this section, you can add Fields to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to determine field's value (can be a constant).

Fields specified on the **Fields** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Retries

Optionally adjust the default retry settings for failed collect jobs.

Retry type: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.

Initial retry interval (ms): Time interval between failed request and first retry (kickoff). Maximum allowed value is `20,000 ms` (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.

Retry limit: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.

Backoff multiplier: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so forth.

Retry HTTP codes: List of HTTP codes that trigger a retry. Leave unchanged to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.

Honor Retry-After header: When toggled on (the default), retry-after headers up to a maximum of 20 seconds are processed. Delays longer than 20 seconds are ignored.

- Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
- When toggled off, Cribl Stream ignores all `retry-after` headers.

Retry connection timeout: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.

Retry connection reset: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.

Advanced Settings

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default value is `300`. A value of `0` means wait indefinitely.

Time to live (seconds): Time to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in the **Job Inspector**. The default value of `4h` means Cribl Stream keeps the collector's job artifacts on disk and visible in **Job Inspector** for 4 hours, then automatically cleans them up.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Scheduling

When you enable a **Content type** under **General Settings** the following scheduling options are available:

The Cron schedule configures when collection requests are made.

Job timeout: Maximum time this job will be allowed to run. Units are seconds, if not specified. Sample values: `30`, `45s`, or `15m`. Minimum granularity is `10` seconds, so a `45s` value would round up to a 50-second timeout. Defaults to `0`, meaning unlimited time (no timeout).

Log Level: Level at which to set task logging. More verbose levels are useful for troubleshooting jobs and tasks, but use them sparingly.

The **Earliest time** and **Latest time** defines the time range of events to collect, based on the _time field. These fields accept the following syntax:

`[+|-]<time_integer><time_unit>@<snap-to_time_unit>`

To break down this syntax:

Syntax Element	Values Supported
Offset	Specify: `-` for times in the past, `+` for times in the future, or omit with `now`.
<time_integer>	Specify any integer, or omit with `now`.
<time_unit>	Specify the `now` constant, or one of the following abbreviations: `s[econds]`, `m[inutes]`, `h[ours]`, `d[ays]`, `w[eeks]`, `mon[ths]`, `q[uarters]`, `y[ears]`.
@<snap-to_time_unit>	Optionally, you can append the `@` modifier, followed by any of the above `<time_unit>`s, to round down to the nearest instance of that unit. (See the next section for details.)

Cribl Stream validates relative time values using these rules:

- **Earliest** must not be later than **Latest**.
- Values without units get interpreted as seconds. For example, `-1 = -1s`.

Snap-to-Time Syntax

The `@` snap modifier always rounds down (backwards) from any specified time. This is true even in relative time expressions with `+` (future) offsets. For example:

- `@d` snaps back to the beginning of today, 12:00 AM (midnight).
- `+128m@h` looks forward 128 minutes, then snaps back to the nearest round hour. (If you specified this in the **Latest** field, and ran the Source at 4:20 PM, collection would end at 6:00 PM. The expression would look forward to 6:28 PM, but snap back to 6:00 PM.)

Other options:

- `@w` or `@w7` to snap back to the beginning of the week, defined as the preceding Sunday.
- To snap back to other days of a week, use `w1` (Monday) through `w6` (Saturday).
- `@mon` to snap back to the 1st of a month.
- `@q` to snap back to the beginning of the most recent quarter: Jan. 1, Apr. 1, Jul. 1, or Oct. 1.
- `@y` to snap back to Jan. 1.

Working with State Tracking

You can configure the Source to track state, either by time or another arbitrary value. This can help prevent overlaps between jobs, where subsequent runs may return some of the same results as previous runs. Similarly, it can help prevent gaps in data by allowing a run to pick up from where the last run ended.

State update expression: JavaScript expression that defines how to update the state from an event. Use the event's data and the current state to compute the new state.

State merge expression: JavaScript expression that defines which state to keep when merging a task's newly reported state with the previously saved state. Evaluates prevState and newState variables, resolving to the state to keep.

The default values for these fields are configured to track state by the latest `_time` field found in events gathered in a collection run.

Understanding State Expression Fields

The State update and State merge expressions control how state is derived from a collection run and how it is merged with existing state, respectively. They're preconfigured to work with the common use case of tracking state by latest _time, but you may need to update them for other use cases. To understand what these fields do, let's break down the default values.

State update expression

This expression has a default value of:

```
__timestampExtracted !== false && {latestTime: (state.latestTime || 0) > _time ? state.latestTime : _time}
```

The `__timestampExtracted` field is set to false if the Event Breaker was unable to parse time for the event. If this is the case, don't want to update state (the event's `_time` value defaults to `Date.now()` if the Event Breaker was unable to parse out the correct time). If `__timestampExtracted` is false, take advantage of short-circuit evaluation to not update state.

State values must resolve to an object, such as:

```
{ "latestTime": 17122806161 }
```

If the expression does not resolve to an object, Cribl Stream will ignore the result.

`{latestTime: (state.latestTime || 0) > _time ? state.latestTime : _time}` - compare `state.latestTime` to the event's `_time` value, keeping whichever value is greater.

State Merge Expression

This expression has a default value of:

```
prevState.latestTime > newState.latestTime ? prevState : newState
```

It compares `prevState` (the state that was previously saved) to `newState` (the state reported from the most recent collection task), keeping the state with the greatest `latestTime` value.

Managing State

Select **Manage State** to view, modify, or delete a state. For more information, see Manage State.

The default values for these fields are configured to track state by the latest `_time` field found in events gathered in a collection run.

OpenAI API Rate Limits

OpenAI enforces per-organization and per-project rate limits on both the number of requests and the number of tokens you can send over a given interval. When the OpenAI Source calls the API, each response includes rate-limit headers that describe your current limits and remaining quota, for example:

- `x-ratelimit-limit-requests` / `x-ratelimit-remaining-requests`
- `x-ratelimit-limit-tokens` / `x-ratelimit-remaining-tokens`
- `x-ratelimit-reset-requests` / `x-ratelimit-reset-tokens`

If your schedules or enabled content types cause the Source to exceed these limits, OpenAI will return `429` responses. The Source relies on its retry settings and your polling schedule to avoid sustained rate-limit errors. If you see repeated `429s` in job logs, reduce the polling frequency or disable nonessential endpoints.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are not part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__collectible` - Contains metadata about each collection job.
- `__collectStats` - Contains per-request metadata.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the Live Data tab, click Start Capture to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Response Errors

The Source treats all non-200 responses from configured URL endpoints as errors. This includes `1xx`, `3xx`, `4xx`, and `5xx` responses.

A few exceptions are treated as non-fatal errors:

- Where a collect job launches multiple tasks, and only a subset of those tasks fail, Cribl Stream places the job in failed status, but treats the error as non-fatal. (Note that Cribl Stream does not retry the failed tasks.)
- Where a collect job receives a `3xx` redirection error code, it follows the error's treatment by the underlying library, and does not necessarily treat the error as fatal.

8.1.22. OpenTelemetry (OTel) Source In Cribl Stream

Cribl Stream supports receiving trace, metric, and log events from OTLP-compliant senders.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Looking for the OpenTelemetry Source in **Cribl Search**? See Add an OpenTelemetry Source for Cribl Search.

Supported and Unsupported Input Data

The OpenTelemetry (OTel) Source supports receiving:

- Compressed inbound data (with DEFLATE or gzip compression), as well as uncompressed data.
- Telemetry data over either of the transports that the OpenTelemetry Protocol (OTLP) describes: gRPC or HTTP. OTLP defines Protocol buffer (Protobuf) schemas for its payloads (requests and responses). With the HTTP transport, this Source supports Binary Protobuf payload encoding, but does not support JSON Protobuf.

The OpenTelemetry Project's Signals documentation provides these hierarchical definitions of Cribl Stream's supported trace and metric event types:

- A **trace** tracks the progression of a single request.
    - Each trace is a tree of **spans**.
    - A span object represents the work being done by the individual services, or components, involved in a request as that request flows through a system.
- A **metric** provides aggreggated statistical information.
    - A metric contains individual measurements called **data points**.
- A **log** provides a recording of an event.
    - See the OTLP Logs Function topic for an explanation of OTLP logs structure.

Breaking Changes Between OTLP 0.19.0 and 1.3.1

With version 0.19.0, the OTLP Protobuf spec introduced breaking changes in its field names. This means that if you created Routes or Pipelines that work with OTLP version 0.10.0, they may break when you use them with OTLP 1.3.1.

To use Routes or Pipelines created before Cribl Stream 4.7 with OTLP 1.3.1, you must check their filters and logic for any use of obsolete field names, and modify them accordingly,

Configure an OTel Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Unique ID for this Source. For example, `OTel042`.
    - **Description**: Optionally, enter a description.
    - **OTLP version**: The drop-down offers `0.10.0` and `1.3.1` (default).
    - **Protocol**: Use the drop-down to choose the protocol matching the data you will ingest: `gRPC` (default), or `HTTP`.
    - **Address**: Enter the hostname/IP to listen on. Defaults to `0.0.0.0` (all addresses, IPv4 format).
    - **Port**: By default, OTel applications send output to port `4317` when using the gRPC protocol, and port `4318` when using HTTP. This setting defaults to `4317` - you must change it if you set **Protocol** (below) to `HTTP`, or you want Cribl Stream to collect data from an OTel application that is using a different port.

      Port `4318` is not available on Cribl-managed Worker Groups in Cribl.Cloud.

    - The **Extract spans**, **Extract metrics**, and **Extract logs** settings are specific to the OpenTelemetry Source. By default, these options are toggled off, allowing Cribl Stream to act as a pass-through that generates a single event for each incoming OTel payload. This is useful when you want to forward complete OTel events to downstream systems, such as persistent storage, without breaking them apart. You can enable these settings to extract and process individual records from within OTel events:
        - **Extract spans**: Generates an individual event for each span in a trace. Traces typically contain multiple spans.

        - **Extract metrics**: Generates an individual event for each data point in a metric event. OTel metrics often contain multiple data points per event.
        - **Extract logs**: Available only when **OTLP version** is set to `1.3.1`. Generates an individual event for each log record. Cribl recommends enabling this option to simplify log transformation and manipulation.
    - **Tags**: Optionally, add tags to help filter and group Sources within Cribl Stream's UI. Tags are not included in the event data. Separate tag names using a tab or hard return.
3. Optionally, you can adjust the Authentication, TLS, Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the following sections.
4. Select **Save**, then **Commit & Deploy**.

Authentication

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Enter the bearer token that must be included in the authorization header.
- **Auth token (text secret)**: Provide an HTTP token referenced by a secret. Select a stored text secret in the resulting drop-down, or click **Create** to configure a new secret.
- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret)**: Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or click **Create** to configure a new secret.

TLS Settings (Server Side)

In OTel terminology, your Cribl Stream OTel Source will receive OTel data from a **Collector** running on a local **agent**. In Cribl Stream's terminology, the Collector is the **client** and the OTel Source is the **server**. This is why this Source's UI identifies the Source's TLS Settings as "server-side."

For more about this client-server relationship, see the TLS Configuration Example below.

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs (mTLS). Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Defines the oldest TLS version allowed for incoming connections. Connections using an earlier version will be rejected. Use this setting to enforce a baseline for secure communication.

Maximum TLS version: Defines the newest TLS version that can be negotiated during a connection. Use this setting to restrict connections to tested and supported TLS versions.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

When **General Settings** > **Protocol** is set to `gRPC`, the UI displays one additional setting:

- **Active connection limit**: Maximum number of active connections allowed per Worker Process. Defaults to `1000`. Set a lower value if connection storms are causing the Source to hang. Set `0` for unlimited connections.
- **Environment**: Optionally, specify a single Git branch on which to enable this configuration. If this field is empty, the config will be enabled everywhere.

When **General Settings** > **Protocol** is set to `HTTP`, the UI displays these additional settings:

- **Health check endpoint**: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:
    - `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
    - `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.
- **Active request limit (on-prem only)**: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

- **Requests-per-socket limit**: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.
- **Socket timeout (seconds)**: How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.
- **Request timeout (seconds)**: How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.
- **Keep-alive timeout (seconds)**: After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).
- **IP allowlist regex**: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).
- **IP denylist regex**: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).
- **Environment**: Optionally, specify a single Git branch on which to enable this configuration. If this field is empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

OTel internal fields are metadata fields added to events processed by the OTel Source. These fields provide essential context about the event's origin and processing details and can be referenced in filters, Routes, and Pipelines to apply specific logic based on the event's metadata. These fields are not included in the final output unless explicitly configured at the Pipeline or Destination stage.

Fields for this Source:

- `__otlp.version` - The OTLP version used to parse the event. For example, `0.10.0` or `1.3.1`. This helps ensure compatibility with upstream systems and configurations.
- `__otlp.type` - The type of telemetry data. For example, `logs`, `metrics`, or `traces`. This is crucial for distinguishing between different signal types in Pipelines.
- `__otlp.extracted` - A boolean flag that denotes whether individual records were extracted from a batch event. If `true`, the event represents an extracted record. This behavior is directly governed by the **Extract spans**, **Extract metrics**, and **Extract logs** general settings of the OTel Source.

TLS Configuration Example

Here's a simple example for using TLS to secure the communication between an OpenTelemetry client and your Cribl Stream OTel Source.

1. Choose or generate a certificate and key. If you need to generate a certificate/key pair, you can adapt the following OpenSSL command:

   ```
   openssl req -nodes -new -x509 -newkey rsa:2048 -keyout myKey.pem -out myCert.pem
   ```

   Copy

   This example command will generate both a self-signed cert named `myCert.pem` (certified for 420 days), and an unencrypted, 2048-bit RSA private key named `myKey.pem`.

2. Configure the **TLS Settings (Server Side)**. Toggle **Enabled** on, then:
   - Enter the appropriate values in the **Certificate name**, **Private key path**, and **Certificate path** fields. A **Create** link is available if you need a new certificate, and **Certificate name** also works as a drop-down to allow you to choose from any existing certificates.
   - Leave the **CA certificate path** field empty.
   - Leave **Authenticate client (mutual auth)** toggled off.
3. Configure the OTel client. See the OTel Collector TLS Configuration Settings README for an explanation of the relevant settings. The config file might be named `otel-config.yaml`, `otel-local-`

`config.yaml`, or just `config.yaml`, depending on your environment. This YAML file will have an `exporters` section, which you must edit to include an `otlp` sub-section, as follows:

- Add an `endpoint` whose value is the IP address of either (a) the Cribl Stream Worker Node on which your OTel Source is running, or (b) the IP address of the load balancer for the relevant Worker Group. In the example snippet below, this is the `<Cribl_IP_address>`. Specify the port on which Cribl Stream's OTel Source is listening; port `4317` is the default.
- Set `tls` > `insecure` to `false`. This matches your setting **TLS Settings (Server Side)** > **Enabled** to toggled on for the Cribl Stream OTel Source.
- Set `tls` > `insecure_skip_verify` to `true`. This matches your setting **TLS Settings (Server Side)** > **Authenticate client (mutual auth)** to toggled off for the Cribl Stream OTel Source. Setting `insecure_skip_verify` to `true` is also required if you're using a self-signed certificate.

Here's how the section you edited should look:

```
exporters:
  otlp:
    endpoint: "https://<Cribl_IP_address>:4317"
    tls:
      insecure: false
      insecure_skip_verify: true
```

Copy

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.23. SNMP Trap Source

Cribl Stream supports receiving data from SNMP Traps.

Type: **Push** | TLS Support: **NO** | Event Breaker Support: **No**

Configure Cribl Stream to Receive SNMP Traps

Cribl Stream, except in Cribl.Cloud, ships with an SNMP Trap Source preconfigured to listen on port `9162`. You can clone or directly modify this Source to further configure it, and then enable it.

In Cribl Stream, configure SNMP Traps:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Source definition. The default Source is prefilled with the value `in_snmp_trap`. which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Address to bind on. Defaults to `0.0.0.0` (all addresses).
    - **UDP Port**: Port on which to receive SNMP traps. Defaults to `162`.

      Sending large numbers of UDP events per second can cause Cribl.Cloud to drop some of the data. This results from restrictions of the UDP protocol.

      To minimize the risk of data loss, deploy a customer-managed (hybrid) Stream Worker Group with Worker Nodes as close as possible to the UDP senders. Cribl also recommends tuning the OS UDP buffer size.
3. Next, you can configure the following **Optional Settings**:

- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication Settings

SNMPv3 authentication provides secure access to devices through optional user authentication and decryption of incoming data packets. Cribl Stream lets you choose any of the following levels of security:

- User name checking without authentication.
- User authentication without privacy.
- User authentication with privacy.

SNMPv3 authentication: Toggle on to enable SNMPv3 authentication and reveal authentication parameters. Default is toggled off.

Allow unmatched traps: Toggle on to pass through traps that don't match any of the configured users. Cribl Stream will not attempt to authenticate or decrypt these traps. When toggled off (default), Cribl Stream drops traps without a correctly configured user name.

v3 Name: Enter the SNMPv3 user name (required). Multiple users are supported.

You must configure at least one user to enable SNMPv3 authentication. If desired, you can enter separate user credentials for each SNMPv3 user. The authentication protocol, authentication key, privacy protocol, and privacy key can be unique for each user.

For authentication to succeed, the user name and authentication protocol and key must match in the Source and the sending device configuration.

For decryption to succeed, the user name, authentication protocol and key, and the privacy protocol and key must match in the Source and the sending device configuration.

Authentication protocol: Select the authentication protocol required for your use case. If you select `None`, Cribl Stream will only check the user name without authenticating the user or decrypting the data. Otherwise, Cribl Stream uses the authentication protocol you specify and the key you provide to authenticate the user.

Options include:

- `None`
- `MD5`
- `SHA1`

- `SHA224`
- `SHA256`
- `SHA384`
- `SHA512`

v3 authentication key: Enter the authentication key.

Choosing an authentication protocol also allows you to select an optional privacy protocol to decrypt incoming packets.

Cribl Stream logs authentication failures at the `debug` level.

Privacy protocol: Select the privacy protocol required for your use case. If you select `None`, Cribl Stream authenticates the user without decrypting the incoming data. Otherwise, Cribl Stream uses the privacy protocol you specify and the key you provide to decrypt the data.

Options include:

- `None`
- `AES128`
- `AES256b (Blumenthal)`
- `AES2556r (Reeder)`

For every successfully decrypted trap, Cribl Stream adds `__didDecrypt: true` to the Event. Cribl Stream logs decryption failures at the `debug` level.

v3 privacy key: Enter the privacy key.

Add user: Click to add a new set of user credentials for SNMPv3 authentication.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Include varbind types: Toggle on to include varbind types in the `varbinds` field on each event. This is required for the SNMP Trap Serialize Function to serialize compliant events into SNMP traps before sending them to an SNMP Trap Destination. The field is parsed to an array of objects like, `[{oid: string, value: any, type: number}]`. Off by default.

IP allowlist regex: Regex matching IP addresses that are allowed to send data. Defaults to `.*`, that is, all IPs.

Buffer size limit (events): Maximum number of events to buffer when downstream is blocking. Defaults to `1000`.

UDP socket buffer size (bytes): Optionally, set the SO_RCVBUF socket option for the UDP socket. This value tells the operating system how many bytes can be buffered in the kernel before events are dropped. Leave blank to use the OS default. Min: `256`. Max: `4294967295`.

It may also be necessary to increase the size of the buffer available to the `SO_RCVBUF` socket option. Consult the documentation for your operating system for a specific procedure.

Setting this value will affect OS memory utilization.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__didDecrypt`: Set to `true` for every trap that is successfully decrypted.
- `__final`
- `__inputId`
- `__snmpRaw`: Buffer containing Raw SNMP packet
- `__snmpVersion`: Acceptable values are `0`, `2` , or `3`. These respectively indicate SNMP v1, v2c, and v3.
- `__srcIpPort` : In this particular Source, this field uses a pipe (|) symbol to separate the source IP address and the port, in this format: `event.__srcIpPort = ${rInfo.address}|${rInfo.port};`
- `_time`

Considerations for Working with SNMP Trap Data

- It's possible to work with SNMP metadata (that is, we'll decode the packet). Options include dropping, routing, and so on.
- SNMP packets can be forwarded to other SNMP destinations. However, the contents of the incoming packet **cannot** be modified - that is, we'll forward the packets verbatim as they came in.
- SNMP packets can be forwarded to non-SNMP destinations (for example, Splunk, Syslog, S3, and so on).
- Non-SNMP input data **cannot** be sent to SNMP destinations.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.24. Syslog Source In Cribl Stream

The Syslog Source receives syslog data (UDP/TCP) from various devices. Syslog messages are parsed into structured fields or stored in a raw format if unrecognized. For high-volume scenarios, TCP load balancing distributes data, optimizing performance and minimizing CPU strain.

Compatible RFCs:

- Supports structured syslog data defined by RFC 3164 and RFC 5424, which can include timestamps, facility codes, severities, and message content for detailed analysis.
- Processes message-length prefixes specified in RFC 5425 and RFC 6587, ensuring complete and accurate data transmission.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

For details on how to replace your syslog server with Cribl Stream, and for configuration examples, see these resources:

- Syslog Best Practices.
- Syslog to Cribl Stream Reference Architecture.
- Palo Alto Syslog to Cribl.Cloud.
- CEF/LEEF Syslog Pack.
- Exabeam Pack for Palo Alto Networks NGFW.

Looking for the Syslog Source in **Cribl Search**? See Add a Syslog Source for Cribl Search.

Configure Cribl Stream to Receive Data over Syslog

Cribl Stream ships with a Syslog Source preconfigured to listen for both UDP and TCP traffic on Port 9514. You can clone or directly modify this Source to further configure it, and then enable it.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.

- To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.

2. In the **New Source** modal, configure the following under **General Settings**:
   - **Input ID**: Enter a unique name to identify this Syslog Source definition. The default Source is prefilled with the value `in_syslog_default`. which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.

The default filter expression from the **Input ID** field, `__inputId.startsWith('syslog:in_syslog:')` includes a trailing colon, which is omitted in the **Preview Full** mode of Data Preview. Therefore, when testing Routes and Pipelines, adjust your filter by removing the colon to match **Preview Full**. This issue is limited to testing and does not affect normal data processing.

   - **Description**: Optionally, enter a description.
   - **Address**: Enter the hostname/IP on which to listen for data. For example, `localhost` or `0.0.0.0`. For details, see IP Address Considerations.
   - **UDP port**: Enter the UDP port number to listen on. Not required if listening on TCP. The maximum supported inbound UDP message size is 16,384 bytes. For details, see UDP Port Considerations.
   - **TCP port**: Enter the TCP port number to listen on. Not required if listening on UDP.
3. Next, you can configure the following **Optional Settings**:
   - **Fields to keep**: List of fields from source data to retain and pass through. Supports wildcards. Defaults to `*` wildcard, meaning keep all fields. Fields **not** specified here (by wildcard or specific name) will be removed from the event.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in the Cribl Stream UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the TLS, Persistent Queue, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

IP Address Considerations

Cribl recommends that you change the **Address** setting from its default `0.0.0.0` only if you know precisely why you're doing so. There are very few use cases where changing this broad default makes sense. All your Worker Nodes in each Worker Group receive this config, and no two Nodes should have the same IPs, other than `0.0.0.0` or `127.0.0.1`. The `0.0.0.0` tells the system to listen to all interfaces and addresses. A `127.0.0.1` entry means listen **only** on `localhost`.

UDP Port Considerations

Sending large numbers of UDP events per second can cause Cribl.Cloud to drop some of the data. This is due to the UDP protocol's restrictions. To minimize the risk of data loss, deploy a customer-managed (hybrid) Stream Worker Group with Worker Nodes as close as possible to the UDP senders. Cribl also recommends tuning the OS UDP buffer size.

Listening on Port 514

For Syslog devices that are hardcoded to send to port 514, you can listen on this port on Cribl.Cloud, which otherwise is normally available only with privileged access. To do this, enable the built-in `in_syslog_default` Syslog Source, which forwards traffic from port 514 to 10514.

Note that traffic on port 514 is unencrypted. This means that using this port may result in collecting fake or undesirable data. Enable the `in_syslog_default` Source only if you are aware of this risk.

TLS Settings (TCP Only)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may

result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always On`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

On this tab, you can enrich fields with added fields, and attach a pre-processing Pipeline to condition inbound syslog data.

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable proxy protocol: Toggle on if the connection is proxied by a device that supports PROXY protocol v1 or v2.

- **Enable enhanced TLS handshake for proxy protocol**: Toggle on to make TLS handshakes compatible with the PROXY protocol v1 and v2. When enabled, PROXY headers are parsed during the TLS handshake, reliably preserving the original source IP and port even for encrypted traffic routed through proxies or load balancers that prepend these headers.

Single msg per UDP: Toggle on to treat received UDP packet data as a full syslog message. If toggled off (default) Cribl Stream will treat newlines within the packet as event delimiters.

Allow non-standard app name: Toggle on to allow hyphens to appear in an RFC 3164-formatted Syslog message's `TAG` section. For details, see TAG Section Processing.

Enable TCP load balancing: Toggle on if you want the Source to load balance traffic across all Worker Processes, as explained in TCP Load Balancing. This setting is available only in Cribl Stream single-instance and Distributed deployments and when a **TCP port** is configured.

IP allowlist regex: Regex matching IP addresses that are permitted to send data. Defaults to `.*` (that is, all IPs).

Active connection limit: Maximum number of active connections allowed per Worker Process. Defaults to `1000`. Set a lower value if connection storms are causing the Source to hang. Set `0` for unlimited connections.

TCP socket idle timeout (seconds): The duration that Cribl Stream will wait for activity on an idle TCP socket before closing the connection. Disabled when set to `0`, the default.

TCP forced socket termination timeout (seconds): The extra time the server waits before forcibly closing a socket that has been idle (**TCP socket idle timeout**) or exceeded its maximum lifespan (**TCP socket max lifespan**) but has not yet properly closed. This prevents resource leaks caused by unresponsive clients or network issues. Configure based on network latency and client behavior. Default: `30` seconds. Set to `0` to disable.

TCP socket max lifespan (seconds): The duration that a socket is allowed to remain open, regardless of activity. This setting prevents resource exhaustion (such as TCP pinning) by limiting the lifespan of connections.

Configure based on expected connection durations and resource availability. Disabled when set to `0`, the default.

Buffer size limit (events): Maximum number of events to buffer when downstream is blocking. The buffer is only in memory. (This setting is applicable only to UDP syslog.) Events dropped because they exceed this buffer are logged as dropped in `stats` messages.

UDP socket buffer size (bytes): Optionally, set the SO_RCVBUF socket option for the UDP socket. This value tells the operating system how many bytes can be buffered in the kernel before events are dropped. Leave blank to use the OS default. Min: `256`. Max: `4294967295`.

You might also need to increase the size of the buffer available to the `SO_RCVBUF` socket option. Consult the documentation for your operating system for a specific procedure.

Setting this value will affect OS memory utilization.

Default timezone: Timezone to assign to timestamps that omit timezone info. Accept the default `Local` value, or use the drop-down list to select a specific timezone by city name or GMT/UTC offset.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

TCP Load Balancing

This feature is available for Cribl Stream deployments running in either single-instance or Distributed mode. The UI displays the setting when a **TCP port** is configured.

In single-instance mode, enabling TCP load balancing still starts a dedicated Load Balancer Worker Process. Ensure the host has enough CPU capacity to run the Load Balancer plus the configured Worker Processes.

When the Syslog Source receives a high volume of data over TCP, a single Worker Process can place excessive CPU load on its Cribl Stream Worker Node, while remaining Worker Processes on the same node mostly sit idle. This undesirable condition is called "TCP pinning" because the high volume of traffic "pins" a single TCP connection.

To alleviate TCP pinning, try toggling **Advanced Settings** > **Enable TCP load balancing** on. The Syslog Source will then load balance traffic across all Worker Processes.

When TCP load balancing is enabled, the Worker Node forks a special Worker Process dedicated to load balancing; creates sockets for communication between the load balancer and the regular Worker Processes; and, makes TCP load balancing metrics available.

The Load Balancing Process

The Worker Node forks a new, special **Load Balancer** Worker Process that distributes incoming syslog data among the other Worker Processes. For customer-managed (hybrid or on-prem) Worker Groups, consider reducing the Process count by 1, if possible. This will leave a core free to run the load balancer process.

If you want to verify that the load balancer Worker Process exists, teleport into the relevant Worker and in the **Worker Processes** tab you will see a Worker Process with `LB` in its name.

To view logs from the Leader UI, navigate to **Monitoring** > **Logs** > `<Worker Node ID>` > **Load Balancer**.

Worker Process Socket Files

The load balancer Worker Process sends data to the regular Worker Processes over inter-process communication (IPC) sockets. This means that in the `state` directory, there is a socket file for each load balancer Worker Process and each Worker Process.

By default, the socket files are linked to `state`, from the Cribl `tmp` directory. For customer-managed (hybrid or on-prem) Stream Worker Groups, you can specify a directory other than `tmp` for these links. Cribl recommends that you do **not** use this option, but if you find it necessary, navigate to **Worker Group/Fleet Settings** > **General Settings** > **Sockets** to configure as desired.

TCP Load Balancing Metrics

Metrics specific to TCP load balancing become available.

- `lb.bytes_out` - Total bytes processed by the load balancer Worker Process, by Source.
- `lb.writable_sockets` - Total unblocked Worker Process sockets, by Source.
- `lb.blocked_sockets` - Total blocked Worker Process sockets, by Source.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source data via the Routing table.

Select **QuickConnect** to send the data from this Source to one or more Destinations via independent, direct connections.

TAG Section Processing

Cribl Stream will try to parse `appname` out of a syslog message's `TAG` section, even when the message is not RFC 5424-formatted.

This practice means that the `TAG` section can contain any alphanumeric character or an underscore (`_`). When the **Allow-non-standard app name** option is enabled, Cribl Stream will also process hyphens that appear in an

RFC 3164-formatted Syslog message's `TAG` section. If Cribl Stream encounters a hyphenated `appname`, it will continue processing to find `procid`. (This setting has no effect on RFC 5424-formatted messages.)

If the `TAG` section contains any non-alphanumeric character, Cribl Stream will treat that character as the termination of the `TAG` section, and as the starting character of the `CONTENT` section.

`CONTENT` Section Processing

Cribl Stream will try to parse `procid` from the beginning of the `CONTENT` section if this section is directly after the `TAG` section, and if it either follows a `:` or is surrounded by `[]`. This process occurs regardless of the **Allow-non-standard app name** setting.

What Fields to Expect

To create fields, the Syslog Source requires messages to comply with RFC 3164 or RFC 5424. If messages sent to the Source comply with neither of these RFCs, you'll see the following fields:

- `_raw`: Contains the whole message, but no other fields broken out.
- `_time`: Contains the time the message was received by Cribl Stream.
- `__srcIpPort <udp|tcp>:<port>`: See Internal Fields.
- `__syslogFail`: See Internal Fields.
- `host`: IP address of the device sending the incoming message.

If messages sent to the Source comply with either RFC 3164 or RFC 5424, fields that the RFC deems guaranteed will always be there, but fields deemed optional might or might not be. Once Cribl Stream parses the required fields and any optional fields, what remains is the actual message.

To see this in real life, install the `cribl-syslog-input` Pack and preview the `RFC5424-RFC3164.log` sample file.

RFC Name for Field	Cribl Name for Field	Guaranteed?	Notes
`PRI`	`facility`, `facilityName`, `severity`, `severityName`	Yes	`PRI` is a bitwise representation of Facility and Severity, which is how Cribl Stream can derive values for all four of its fields.
`TIMESTAMP`	`_time`	Yes	This field's format differs depending on which RFC the messages adhere to. Cribl Stream converts it to Unix time.
`HOSTNAME`	`host`	Yes	

RFC Name for Field	Cribl Name for Field	Guaranteed?	Notes
APP-NAME	appname	No	
PROCID	procid	No	
MSGID	msgid	No	
STRUCTURED-DATA	structuredData	No	
MSG	message	Yes	

Event Size and Storage

By default, a Syslog Source produces eight parsed fields: `_time`, `appname`, `facility` (numeric), `facilityName` (text), `host`, `message`, `severity` (numeric), and `severityName` (text). This improves readability but does not reduce event volume. It also introduces redundancy, numeric and text pairs for `facility` and `severity`.

If you keep all of the default fields (the default behavior), events written to downstream storage are larger than the original raw syslog line because the original message is accompanied by additional fields. For short messages, this overhead can be significant relative to the raw size and could roughly double the stored size. For larger messages, the percentage increase is lower.

To control size growth, configure trimming at the Source or in a Pipeline:

- Adjust the Source **Fields to keep** setting to include only the fields you require. By default, all fields are kept.
- Use a Pipeline to remove redundant/unnecessary fields and, if appropriate, shrink outbound `_raw` to only the message component.

For a walk-through of optimization steps and expected results, see Syslog Best Practices: The Goal–Optimizing Syslog Events.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but are accessible and Functions can use them to make processing decisions.

Fields for this Source:

- `__inputId`

- `__srcIpPort <udp|tcp>:<port>`: Identifies the port of the sending socket.
- `__syslogFail: true` for data that fails RFC 3164/5424 validation as syslog format.

`stats` Log Message

In Cribl Stream 4.2.2 and newer, `stats` log messages report the number of events received, buffered, or dropped for exceeding the maximum Cribl buffer size. By default, these messages are logged every 60 seconds. These values also appear in **Sources** > **Syslog** > <Source_name> > **Status**, in the **UDP** row.

UDP Tuning

This section applies only to UDP-based syslog ingest with on-prem or hybrid Cribl Stream Worker Groups or Cribl Edge Fleets. It does not apply to TCP-based syslog, nor to Cribl-managed Stream Workers in Cribl.Cloud, where Cribl will manage UDP tuning for you.

Incoming UDP traffic is put into a buffer by the Linux kernel. Cribl will drain from that buffer as resources are available. At lower throughput levels, and with plenty of available processes, this isn't an issue. As you scale up, however, the default size of that buffer is very likely too small.

Monitoring Buffer Size and Packet Errors

You can check the current buffer size with the following command:

```
$ sysctl net.core.rmem_max
```

Copy

A typical value of about 200 KB is far too small for an even moderately busy syslog server.

You can check the health of UDP with the following command. Check the `packet receive errors` line.

```
$ netstat -su
```

Copy

If `packet receive errors` is more than zero, you have lost events, which is a particularly serious problem if the number of errors is increasing rapidly. This means you need to increase your `net.core.rmem_max` setting (covered later in this section).

Commands to Update Settings

You can update the live settings. But you'll also need to change the boot-time setting, so that next time you reboot, everything is ready to roll.

Live change, setting to 25 MB:

```
$ sysctl -w net.core.rmem_max=26214400
net.core.rmem_max = 26214400
$ sysctl -w net.core.rmem_default=26214400
net.core.rmem_default = 26214400
```

Copy

For the permanent settings change, add the following lines to `/etc/sysctl.conf`:

```
net.core.rmem_max=26214400
net.core.rmem_default=26214400
```

Copy

We recommend that you track a few things related to UDP receiving:

- The `netstat -su` command, watching for errors.
- The **Status** tab in the Syslog Source. In particular, watch for dropped messages. They could indicate you need a bigger buffer under **Advanced Settings** (default: 1000 events). They could also indicate your Worker is encountering pressure further down the Pipeline.
- Especially if you increase your kernel receive buffer as above, watch your Worker processes' memory usage.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.25. TCP JSON Source In Cribl Stream

Cribl Stream can receive newline-delimited JSON data over TCP.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Looking for the TCP JSON Source in **Cribl Search**? See Add a TCP JSON Source for Cribl Search.

Configure Cribl Stream to Receive TCP JSON Data

Cribl Stream ships with a TCP JSON Source preconfigured to listen on Port 10070. You can clone or directly modify this Source to further configure it, and then enable it.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. Configure the following under **General Settings**:
    - **Input ID**: Enter a unique name. The default Source is prefilled with the value `in_tcp_json`, which can't be changed via the UI. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter hostname/IP to listen for TCP JSON data. For example, `localhost` or `0.0.0.0`.
    - **Port**: Enter the port number to listen on.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Use this default option to enter the shared secret that clients must provide in the `authToken` header field. Exposes an **Auth token** field for this purpose. (If left blank, unauthorized access will be permitted.) A **Generate** link is available if you need a new secret.
    - **Secret**: This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the `authToken` header field value described above. The secret can

reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need a new secret.

4. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable proxy protocol: Toggle on if the connection is proxied by a device that supports proxy protocol v1 or v2.

Enable load balancing (Cribl Stream only): Toggle on to distribute the data from incoming TCP connections across multiple Worker Processes. This improves performance compared to the default of having all data from an incoming connection processed on a single Worker Process.

- This spins up an extra Worker Process (named `wLB` in **Settings** > **Processes**) on the Worker Node, which handles splitting data from incoming TCP connections across all other Worker Processes.
- When enabled, the option to include common fields in the JSON header is not supported. This means fields specified under `fields` in the JSON header are not automatically added to all events.

Load balancing is available in Cribl Stream single-instance and Distributed deployments when a TCP port is configured.

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` (such as, all IPs).

Active connection limit: Maximum number of active connections allowed per Worker Process. Defaults to `1000`. Set a lower value if connection storms are causing the Source to hang. Set to `0` for unlimited connections.

Socket idle timeout (seconds): The duration that Cribl Stream will wait for activity on an idle TCP socket before closing the connection. Disabled when set to `0`, the default.

Forced socket termination timeout (seconds): The extra time the server waits before forcibly closing a socket that has been idle (**TCP socket idle timeout**) or exceeded its maximum lifespan (**TCP socket max lifespan**) but has not yet properly closed. This prevents resource leaks caused by unresponsive clients or network issues. Configure based on network latency and client behavior. Default: `30` seconds. Set to `0` to disable.

Socket max lifespan (seconds): The duration that a socket is allowed to remain open, regardless of activity. This setting prevents resource exhaustion (such as TCP pinning) by limiting the lifespan of connections. Configure based on expected connection durations and resource availability. Disabled when set to `0`, the default.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the configuration will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Field for this Source:

- `__inputId`
- `__srcIpPort`

Format

Cribl Stream expects TCP JSON events in newline-delimited JSON format:

1. A header line. Can be empty - for example, `{}`. If **authToken** is enabled (see above) it should be included here as a field called `authToken`. When `authToken` is **not** set, the header line is **optional**. In this case, the first line will be treated as an event if does not look like a header record.
   In addition, if events need to contain common fields, they can be included here under `fields`. In the example below, `region` and `AZ` will be automatically added to all events.

   > When you enable **Enable load balancing (Cribl Stream only)** under Advanced Settings, the Source will no longer support common fields in the JSON header. Consequently, any fields you define in the JSON header `fields` object will not be automatically propagated to each event.

2. A JSON event/record per line.

   Sample TCP JSON Events

```
{"authToken":"myToken42", "fields": {"region": "us-east-1", "AZ":"az1"}}

{"_raw":"this is a sample event ", "host":"myHost", "source":"mySource", "fieldA
{"host":"myOtherHost", "source":"myOtherSource", "_raw": "{\"message\":\"Somethi
```

Copy

TCP JSON Field Mapping to Splunk

If a TCP JSON Source is routed to a Splunk destination, fields within the JSON payload are mapped to Splunk fields. Fields that do not have corresponding (native) Splunk fields become index-time fields. For example, let's assume we have a TCP JSON event as below:

```
{"_time":1541280341, "host":"myHost", "source":"mySource", "_raw":"this is a sample event ", "fieldA":"valueA"}
```

Here, `_time`, `host`, and `source` become their corresponding fields in Splunk. The value of `_raw` becomes the actual body of the event, and `fieldA` becomes an index-time field (`fieldA::`valueA``).

Examples

Testing TCP JSON In

This first example simply tests that data is flowing in through the Source:

1. Configure Cribl Stream to listen on port `10001` for TCP JSON. Set `authToken` to `myToken42`.
2. Create a file called `test.json` with the payload above.
3. Send it over to your Cribl Stream host: `cat test.json | nc <myCriblHost> 10001`

Cribl Stream to Cribl.Cloud

This second example demonstrates using TCP JSON to send data from one Cribl Stream instance to a downstream Cribl.Cloud instance. We assume that the downstream Cloud instance uses Cribl.Cloud's **default** TCP JSON Source configuration.

So all the configuration happens on the upstream instance's TCP JSON Destination. Replace the `<organizationId>` placeholder with the Org ID from your Cribl.Cloud portal.

TCP JSON Destination Configuration

On the upstream Cribl Stream instance's Destination, set the following field values to match the target Cloud instance's defaults:

General Settings

Address: `default.main.<organizationId>.cribl.cloud` - you can simply copy/paste your Cribl.Cloud portal's **Ingest Endpoint** here. With a Cribl.Cloud Enterprise plan, generalize the `default.main` substring in this URL to `<groupName>.main` when sending to other Worker Groups.

Port: `10070`

TLS Settings (Client Side)

Enabled: Toggle on

Validate server certs: Toggle on

Periodic Logging

Cribl Stream logs metrics about incoming requests and ingested events once per minute.

These logs are stored in the `metrics.log` file. To view them in the UI, open the Source's **Logs** tab and choose **Worker Process X Metrics** from the drop-down, where **X** is the desired Worker Process.

This kind of periodic logging helps you determine whether a Source is in fact still healthy even when no data is coming in.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.26. TCP (Raw) Source In Cribl Stream

Cribl Stream supports receiving of data over TCP. (See examples and header options below.)

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

Looking for the TCP (Raw) Source in **Cribl Search**? See Add a TCP (Raw) Source for Cribl Search.

Configure Cribl Stream to Receive TCP Data

Cribl Stream ships with a TCP Source preconfigured to listen on Port 10060. You can clone or directly modify this Source to further configure it, and then enable it.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. Configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this TCP Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter hostname/IP to listen for raw TCP data. For example, `localhost` or `0.0.0.0`.
    - **Port**: Enter port number.
    - **Enable header**: Toggle on to indicate that client will pass a header record with every new connection. The header can contain an `authToken`, and an object with a list of fields and values to add to every event. These fields can be used to simplify Event Breaker selection, routing, and so forth. Header format: `{ "authToken" : "myToken", "fields": { "field1": "value1", "field2": "value2" }}`.
    - **Authentication method**: Select **Manual** to enter an auth token directly, or **Secret** to use a text secret to authenticate.

- **Auth token**: If you selected **Manual**, enter an auth token, or select **Generate** to automatically generate a token. If empty, unauthorized access is permitted.
- **Auth token (text secret)**: If you selected **Secret**, enter a shared secret to be provided by any client (in `authToken` header field). Select **Create** to create a new secret. If empty, unauthorized access will be permitted.

3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the TLS Settings, Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

On Cribl.Cloud, you can enable TLS using Cribl-provided certficates if desired.

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may

result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Custom Command

In this section, you can pass the data from this input to an external command for processing before the data continues downstream.

Enabled: Default is toggled off. When toggled on:

Command: Enter the command that will consume the data (via `stdin`) and will process its output (via `stdout`).

Arguments: Click **Add Argument** to add each argument for the command. You can drag arguments vertically to resequence them.

Event Breakers

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable proxy protocol: Toggle on if the connection is proxied by a device that supports Proxy Protocol v1 or v2. When this setting is enabled, the `__srcIpPort` internal field will show the original source IP address and port. When it is disabled, the `__srcIpPort` field will show the IP address and port of the proxy that forwarded the connection.

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` (i.e,. all IPs).

Active connection limit: Maximum number of active connections allowed per Worker Process. Defaults to `1000`. Set a lower value if connection storms are causing the Source to hang. Set `0` for unlimited connections.

Socket idle timeout (seconds): The duration that Cribl Stream will wait for activity on an idle TCP socket before closing the connection. Disabled when set to `0`, the default.

Forced socket termination timeout (seconds): The extra time the server waits before forcibly closing a socket that has been idle (**TCP socket idle timeout**) or exceeded its maximum lifespan (**TCP socket max lifespan**) but has not yet properly closed. This prevents resource leaks caused by unresponsive clients or network issues. Configure based on network latency and client behavior. Default: `30` seconds. Set to `0` to disable.

Socket max lifespan (seconds): The duration that a socket is allowed to remain open, regardless of activity. This setting prevents resource exhaustion (such as TCP pinning) by limiting the lifespan of connections. Configure based on expected connection durations and resource availability. Disabled when set to `0`, the default.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and functions can use them to make processing decisions.

Fields accessible for this Source:

- `__inputId`
- `__srcIpPort`
- `__channel`

TCP Source Examples

Every new TCP connection may contain an **optional** header line, with an `authToken` and a list of fields and values to add to every event. To use the Cribl Stream Cloud sample, copy the `<token_value>` out of your Cribl Stream Cloud TCP Source.

Sample test.raw (on-prem) | Sample test.raw (Cribl.Cloud)

```
{"authToken":"myToken42", "fields": {"region": "us-east-1", "AZ":"az1"}}

this is event number 1
this is event number 2
```

Copy

Enabling the Example - Cribl Stream

1. Configure Cribl Stream to listen on port `7777` for raw TCP. Set `authToken` to `myToken42`.
2. Create a file called `test.raw`, with the payload above.
3. Send it over to your Cribl Stream host, using this command: `cat test.raw | nc <myCriblHost> 7777`

Enabling the Example - Cribl.Cloud

Use netcat with `--ssl` and `--ssl-verify`:

Command-line test

```
cat test.raw | nc --ssl --ssl-verify default.main.<organizationId>.cribl.cloud 10060
```

Copy

With a Cribl.Cloud Enterprise plan, generalize the above URL's `default.main` substring to `<groupName>.main` when sending to other Worker Groups.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

8.1.27. UDP (Raw) Source

Cribl Stream supports receiving raw, unparsed data via UDP.

Type: **Push** | TLS Support: **NO** | Event Breaker Support: **NO**

Configure Cribl Stream to Receive Raw UDP Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this raw UDP Source definition.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the hostname/IP to listen for raw UDP data. For example: `localhost`, `0.0.0.0`, or `::`.
    - **Port**: Enter the port number.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Persistent Queue Settings, Processing, and Advanced settings, or Connected Destinations outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Sending large numbers of UDP events per second can cause Cribl.Cloud to drop some of the data. This results from restrictions of the UDP protocol.

To minimize the risk of data loss, deploy a customer-managed (hybrid) Stream Worker Group with Worker Nodes as close as possible to the UDP sender. Cribl also recommends tuning the OS UDP buffer size.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Single msg per UDP: Toggle on if each UDP message should be treated as an independent event. Toggle off (default) if the message should be broken on newlines to create multiple events.

Ingest raw bytes: Toggle on to add a `__rawBytes` field to each event containing an array of the bytes received as the UDP message.

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` (i.e, all IPs).

Buffer size limit (events): Maximum number of events to buffer when downstream is blocking. The buffer is only in memory.

UDP socket buffer size (bytes): Optionally, set the SO_RCVBUF socket option for the UDP socket. This value tells the operating system how many bytes can be buffered in the kernel before events are dropped. Leave blank to use the OS default. Min: `256`. Max: `4294967295`.

It may also be necessary to increase the size of the buffer available to the `SO_RCVBUF` socket option. Consult the documentation for your operating system for a specific procedure.

Setting this value will affect OS memory utilization.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

What Fields to Expect

The Raw UDP Source does minimal processing of the incoming UDP messages to remain consistent with the internal Cribl event model. For each UDP message received on the socket, you can expect an event with the following fields:

- `_raw`: Contains the UTF-8 representation of the entire message received (if **Single msg per UDP** is toggled on), or of the given line that was split out of the message.
- `_time`: Timestamp in Unix time (seconds) at which the message was received by Cribl Stream.
- `source`: A string in the form `udp|<remote IP address>|<remote port>`, indicating the remote sender.
- `host`: The hostname of the machine running Cribl Stream that ingested this event.

Also, the internal fields listed below will be present.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and functions can use them to make processing decisions.

Fields accessible for this Source:

- `__inputId`
- `__srcIpPort`
- `__rawBytes`: When **Ingest raw bytes** is toggled on, this field will be an array containing the bytes of the UDP message.

UDP Tuning

Incoming UDP traffic is put into a buffer by the Linux kernel. Cribl will drain from that buffer as resources are available. At lower throughput levels, and with plenty of available processes, this isn't an issue. As you scale up, however, the default size of that buffer may be too small.

You can check the current buffer size with:

```
$ sysctl net.core.rmem_max
```

Copy

A typical value of about 200 KB is far too small for an even moderately busy syslog server. You can check the health of UDP with the following command. Check the `packet receive errors` line.

```
$ netstat -su
```

Copy

If `packet receive errors` is more than zero, you have lost events, which is a particularly serious problem if the number of errors is increasing rapidly. This means you need to increase your `net.core.rmem_max` setting (see earlier).

You can update the live settings, but you'll also need to change the boot-time setting so next time you reboot everything is ready to roll.

Live change, setting to 25 MB:

```
$ sysctl -w net.core.rmem_max=26214400
net.core.rmem_max = 26214400
$ sysctl -w net.core.rmem_default=26214400
net.core.rmem_default = 26214400
```

Copy

For the permanent settings change, add the following lines to `/etc/sysctl.conf`:

```
net.core.rmem_max=26214400
net.core.rmem_default=26214400
```

Copy

We recommend you track a few things related to UDP receiving:

- The `netstat -su` command, watching for errors.
- The **Status** tab in the UDP (Raw) Source. In particular, watch for dropped messages. They could indicate you need a bigger buffer under **Advanced Settings** (default: 1000 events). They could also indicate your Worker is encountering pressure further down the Pipeline.
- Especially if you increase your kernel receive buffer as above, watch your Worker processes' memory usage.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

8.1.28. Windows Event Forwarder Source In Cribl Stream

Cribl Stream supports receiving Windows events from the Windows Event Forwarding (WEF) mechanism built into modern versions of Microsoft Windows (including Windows 10, Windows Server 2012, and more-recent releases).

The integration between WEF and Cribl works by generating events on Windows systems, which are collected by Windows Event Collectors (WECs). These events are then forwarded to Cribl using WEF, secured by either mutual TLS or Kerberos authentication. Cribl receives the events through the Windows Event Forwarder Source, processes them according to configured pipelines, and sends the processed events to designated Destinations.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No** | Uses Key-Value Store

Looking for the Windows Event Forwarder Source in **Cribl Search**? See Add a Windows Event Forwarder Source for Cribl Search.

Prerequisites

- WEF is set up and enabled.
- WEF client(s) are pointed to WECs.
- You've configured either client certificate or Kerberos authentication.

On Cribl Edge, Kerberos authentication for this Source is supported only for Edge Nodes running on x86_64 Linux.

Configure Cribl Stream to Receive Windows Events

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing**.
    - To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.
2. In the Source modal, configure the following under **General Settings**:
    - **Input ID**: A unique name that identifies this Windows Event Forwarder Source.

- **Address**: The hostname/IP on which to listen for Windows events data. Example: `localhost` or `0.0.0.0`.
- **Port**: The port number. The default, `5986`, is the port used by Windows Event Collector for HTTPS-based subscriptions.
- **Authentication method**: The method for accepting incoming client connections, either `Client certificate` or `Kerberos`.

Kerberos authentication is not supported for Cribl-managed Workers in Cribl.Cloud, but it is enabled for customer-managed Worker Groups, whether on-prem or hybrid.

- For **Client Certificate**:
  - **Certificate name**: Choose the certificate you created.
  - **Private key path**: Server path containing the private key (in PEM format) to use. For Cribl.Cloud, the default path is `/opt/criblcerts/criblcloud.key` for a Cribl-managed Worker Group in Cribl.Cloud. Path can reference `$ENV_VARS`.
  - **Passphrase**: Passphrase to use to decrypt private key.
  - **Certificate path**: Server path containing certificates (in PEM format) to use. For Cribl.Cloud, the default path is `/opt/criblcerts/criblcloud.crt` for a Cribl-managed Worker Group in Cribl.Cloud. Path can reference `$ENV_VARS`.
  - **CA certificate path**: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`. If multiple certificates are present in a `.pem`, the first certificate in the chain should be the issuing CA certificate. If the first certificate is not the issuing CA, use the **CA fingerprint override** setting to specify the expected SHA1 fingerprint. This should match the order of traversing the chain from the host to the root CA cert.

The CA certificate that generated the Cribl Stream server certificate must match the root CA that signed all client certificates that will be forwarding Windows events to this Source. If the issuing CA is an intermediate CA, this CA must also match on the server and all client certs.

  - **Common name**: Regex that a peer certificate's subject attribute must be able to connect. Defaults to `.*`.
    - Matches on the substring after `CN=`.
    - As needed, escape regular expression (regex) tokens to match literal characters. For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.

        - If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.
    - Optionally, select the **Minimum TLS version** and **Maximum TLS version** to accept from connections.
    - **Verify certificate via OCSP**: If toggled on, Cribl Stream will use an OCSP (Online Certificate Status Protocol) service to check that client certificates presented in incoming requests have not been revoked. Exposes the **Strict validation** toggle.
    - **Strict validation**: If enabled, Cribl Stream will fail checks on **any** OCSP error. Otherwise, Cribl Stream will fail checks only when a certificate is revoked, and will ignore other errors (such as `OCSP server is unavailable` errors).
  - For **Kerberos**: Provide the SPN and keytab file location that you set in Keytab file.
    - **Service Principal Name**: The Service Principal Name (SPN) in this format: `HTTP/<fully qualified domain name>@REALM`. This identifies the service in the Kerberos realm. The service pricipal name is case-sensitive, and must be exactly in the form used to export the keytab.
    - **Keytab location**: The path to the keytab file containing the service principal credentials. Cribl Stream uses `/etc/krb5.keytab` by default. This file contains the keys used for Kerberos authentication.
  - Optionally, add **Tags** that you can use to filter and group Destinations in Cribl Stream's **Manage Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
3. **Subscriptions** define which logs to collect and how to filter them. You need to add at least one subscription. For comprehensive details on subscription settings and the query builder, refer to the subscriptions section.
4. Optionally, configure any Persistent Queue, Processing, and Advanced settings outlined in the below sections.
5. Click **Save**, then **Commit & Deploy**.

Subscriptions

Subscriptions control which events are collected from Windows endpoints and sent to the Cribl Stream Windows Event Forwarder Source. They filter the types of events to monitor, like security logs or application logs.

Due to the way subscription bookmarks are stored, Leader CPU usage is directly impacted by the number of subscriptions created for the Source and the number of clients receiving each subscription.

> To optimize performance, prefer to minimize the number of subscriptions by combining them where possible.

You can tailor your event collection to focus on the data that matters most to your Organization's security and monitoring needs. This helps reduce noise, save storage space, and ensure critical events are captured and analyzed.

Select **Add Subscription** and fill out the following fields to define at least one new, required subscription.

Subscription name: A friendly name for the subscription, which is only used to help you identify it.

Version: A read-only field that automatically updates whenever you save changes to the subscription. This version number reflects the specific configuration of the subscription and helps ensure that clients correctly interpret and process the subscription data.

Format: You can choose `Raw` to receive only XML data about the subscribed events, or `RenderedText` to additionally include amplifying information generated by the client about the event's contents.

Heartbeat: The maximum allowable time, in seconds, before the client will check in with Cribl Stream even if it has no new events to send.

Batch timeout: The maximum time, in seconds, that the client should batch new events before sending them to Cribl Stream.

> Windows Event Collector defines two delivery modes of "Event Delivery Optimization":
>
> - The "Minimize Bandwidth" mode corresponds to a **Heartbeat** and **Batch timeout** of 6 hours (21,600 seconds).
> - The "Minimize Latency" mode corresponds to a **Heartbeat** of 1 hour (3,600 seconds) and a **Batch timeout** of 30 seconds.

Locale: The language code to use for the events sent by the client, in RFC-3066 format (for example, `en-US`). The language code uses ISO 639, and the country code uses ISO 3166. Defaults to `en-US` when left blank.

Read existing events: Controls whether the client should send historical events when it first connects. The behavior of **Read existing events** is influenced by the **Use bookmarks** setting. See the table below for a complete overview.

Use bookmarks: Keeps track of which events it has received.

Setting	Behavior
Read existing events toggled off	Only new events generated after it receives the subscription are sent.

Setting	Behavior
Read existing events toggled on and **Use bookmarks** toggled off	All historical events are sent on the initial subscription and each time a client *forcibly* renews its subscription (for example, after group policy update), all events matching the query will be sent. Restarting Cribl Stream or restarting the client will not cause all events to be re-sent.
Read existing events toggled on and **Use bookmarks** toggled on	Historical events are sent on the initial subscription, but only new events (later than the saved bookmark) are sent on subsequent (even forced) subscription renewals.

Important Considerations

- **Subscription changes**: If you change and save a subscription, the saved bookmarks are no longer valid (they are tied to a specific subscription version), and all events matching the updated subscription will be sent.
- **Leader and Worker Node interactions**: Worker Nodes continue to accept new events even when the Leader is down. They sync with the Leader when it's back up. However, bookmarks stored in memory can be lost if a Worker Node restarts while the Leader is down.
- **Client balancing and bookmarks**: If a client balances to a different Worker Node while the Leader is down, it might receive a stale bookmark. Bookmarks are only sent to the client when it requests a subscription, not when it delivers events. This typically happens on the `Refresh=XX` interval from the GPO.
    - If the Leader is down when a client requests a subscription, it will receive the last known bookmark from the Worker Node, which could be outdated.
    - If a Worker Node restarts while the Leader is down, the client might not receive any bookmark at all, potentially leading to duplicate events.

Compression: Sets whether Windows clients compress the events sent to Cribl Stream, using the Streaming Lossless Data Compression (SLDC) algorithm. Defaults to toggled on.

Queries: See the explanation in Configuring Queries below.

Query builder mode: See the explanation in Configuring Queries below.

Targets: Set the DNS names of the clients that should receive this subscription. This field supports wildcard matching.

Fields: Use this setting to add fields exclusively to the events that WEF delivers **for the subscription you are defining**, using Eval-like functionality. To add fields to all incoming events regardless of subscription, use **Processing Settings** > **Fields** instead:

Optionally, you can create the field such that its **Value** is the value of a subscription metadata element. (See the available subscription metadata elements listed below.)

To do this, specify a **Value** of `__subscription.<metadata_element>`. The Source will then find the metadata element you specified in the special object `__subscription` and set the value to that.

For example, if you wanted all events for the subscription to be tagged with its **Locale**, you could set the **Name** to `Locale` (or any arbitrary name) and **Value** to `__subscription.locale`.

- **Field Name**: Unique name for the field you're adding.
- **Value**: JavaScript expression to compute field's value, enclosed in quotes or backticks. (Can evaluate to a constant.)

Subscription Metadata Elements

Metadata Element	Type	Corresponding UI Setting
`version`	string	**Version**
`subscriptionName`	string	**Name**
`contentFormat`	`Raw` or `RenderedText`	**Format**
`readExistingEvents`	boolean	**Read existing events**
`heartbeatInterval`	number	**Heartbeat**
`batchTimeout`	number	**Batch timeout**
`sendBookmarks`	boolean	**Use bookmarks**
`compress`	boolean	**Compression**
`queries`	array	see About the Query builder mode below

Change Subscriptions

If you change a subscription that is in use, Windows Event Forwarder will discover the new subscription ID. Once the `Refresh=XX` interval has elapsed, if any events were sent to the now-invalid previous subscription ID, Windows will resend them to the new one (assuming that these events remain valid under the updated subscription parameters). The Receiver Refuses to Accept Delivery of Events warnings you'll see in this scenario are normal.

About Query Builder Mode

This setting produces the `queries` array in one of two different forms, depending on whether you select its **Simple** button or its **Raw XML** button.

Simple produces an array whose elements each contain `path`, a string that corresponds to **Queries** > **Path**, and `queryExpression`, a string that corresponds to **Queries** > **Query expression**.

Raw XML produces an array whose elements each contain only the `xmlQuery` metadata element. This is the XML query you specified, in the form of a string, that corresponds to the **XML query** setting.

To access any element in the array, use array indexing. For example:

- `__subscription.queries[0].path` will access the path to the first query in an array created by **Query builder mode** > **Simple**.
- `__subscription.queries[0].xmlQuery` will access the first query in an array created by **Query builder mode** > **Raw XML**.

Configure Queries

Queries determine which events to send from the clients. **At least one query is required.** The format is derived from the XPath implementation used by Windows Event Collector.

While a single XPath query in a WEF subscription is limited to approximately 22 EventIDs, you can monitor more by creating a structured XML query with multiple `<Query>` elements within a single subscription. This is the recommended approach for performance and resource efficiency.

You have two **Query builder mode** options: **Simple** or **Raw XML**.

- **Raw XML** allows you to enter your XPath query into the **XML query** field.
- **Simple** allows you to manually build queries. Click **Add Query** to define each new query. A query has the following properties:
  - **Path**: Set this to the `Path` attribute of a `Select` XPath element. See the example below.
  - **Query expression**: Set this to the value inside a `Select` XPath element. See the example below.

With either a **Simple** or **Raw XML** query, you can use the **Targets** field to specify the DNS names of the endpoints that should forward these events. Supports wildcards (for example, `*.mydomain.com`). The default global wildcard (`*`) enables all endpoints.

If you have both **Simple** and **Raw XML** queries defined within a given Subscription, the **Query builder mode** button that you select when saving this Source's configuration determines which query/queries Cribl Stream will send. To send both **Simple** and **Raw XML** queries, use the **Add Subscription** button to define multiple subscriptions.

Example

When creating a subscription in a Windows Event Collector, you can view the generated XML/XPath query. Consider a subscription that returns all events from the Security log that are of severities Critical, Error, or Warning, and that occurred within the last 24 hours. This subscription would generate XML like this:

```
<QueryList>
  <Query Id="0" Path="Security">
    <Select Path="Security">*[System[(Level=1  or Level=2 or Level=3) and TimeCreated
  </Query>
</QueryList>
```

Copy

To use this subscription in a Cribl Stream Windows Event Forwarder Source, you would either use the **Raw XML** option and paste the above XML, or use **Simple** mode and set the following query properties:

- **Path**: `Security`
- **Query expression**: `*[System[(Level=1 or Level=2 or Level=3) and TimeCreated[timediff(@SystemTime) <= 86400000]]]`

You do not need to use the `<Query>` element's `Id` or `Path` attributes anywhere in the Cribl Stream subscription config.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

Use this setting to add fields to **all events coming into the Source**, using Eval-like functionality. (To add fields on a per-subscription basis, use **Subscriptions** > **Fields** instead.)

Field Name: Unique name for the field you're adding.

Value: JavaScript expression to compute field's value, enclosed in quotes or backticks. (Can evaluate to a constant.)

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Allow MachineID mismatch (client certificate only): Toggle on if you do not want to verify that the events sent by a client match the client's certificate Common Name (Subject CN). If toggled off, events where the `MachineID` (by default the client's machine name, like `CLIENT1.domainName.com`) do not match the CN will be rejected.

This setting applies only to client certificate authentication. If you use Kerberos authentication, this option will not appear and will be treated as if it is toggled off.

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source: `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.

- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Log CA fingerprint mismatch warning (client certificate only): Toggle on to log a warning message when the client certificate's issuing Certificate Authority (CA) fingerprint doesn't match the expected CA fingerprint configured in the Windows Event Forwarding Source. This mismatch prevents proper authentication and event forwarding. The log message is `Mismatched fingerprint` and includes both the expected `acceptedFingerprint` and actual `issuerFingerprint` fingerprints. Toggled off by default.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever. For Kerberos authentication, see how to adjust timeouts.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream how long to wait for additional data before closing the socket connection. Defaults to `90` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the timeout, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

For Kerberos authentication, see how to adjust timeouts.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

CA fingerprint override (client certificate only): Use this setting only if the first certificate in the configured CA chain does not match the SHA1 fingerprint that the WEF client expects, such as an intermediate certificate. If that is the case, enter the expected SHA1 fingerprint. The thumbprint specified here must match the `IssuerCA` parameter in the Subscription Manager template.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Timeout Considerations for Kerberos Authentication

If you use Kerberos authentication, there are some trade-offs you should consider when you select settings in **Subscriptions** > **Batch timeout** and **Advanced Settings** > **Keep-alive timeout (seconds)**.

Windows expects socket connections to be long-lived for Kerberos authentication so it can avoid the overhead of performing the two-step Kerberos authentication on each new batch delivery.

Set **Subscriptions** > **Batch timeout**, **Subscriptions** > **Heartbeat**, or both to be less than or equal to **Keep-alive timeout** (or the lowest **Batch timeout** if several subscriptions are used), especially when a load balancer is in front of the Worker Nodes.

Set **Socket timeout** higher than **Batch timeout** (or the lowest **Batch timeout** if several subscriptions are used). This ensures that connections aren't prematurely closed before batches are sent. If the **Socket timeout** has elapsed between event deliveries or heartbeats, Cribl Stream will still handle it gracefully, but incur an additional overhead of two network round trips.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Event Handling

The Windows Event Forwarder Source uses internal fields and a key-value store to manage and process events efficiently. Internal fields are metadata added to events for internal processing and should be treated as read-only to avoid unintended consequences. Key fields include:

- `__final`
- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__inputId`
- `__srcIpPort` - See details below.
- `__subscriptionName`
- `__subscriptionVersion`
- `_raw`
- `_time`

The key-value store uses the `machineId` as keys to store bookmark information when bookmarks are enabled. This ensures that events are processed correctly and efficiently by maintaining the state of event processing.

Together, the key-value store and internal fields work to ensure accurate and efficient event handling. The key-value store maintains the state of event processing, while internal fields provide necessary metadata for internal operations, enabling Cribl Stream to process and route events effectively.

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the WEF client sending data to this Source.

In the WEF Source configuration, there's an option to handle the `X-Forwarded-For` header, which is commonly used by load balancers to pass the original client's IP address. When any proxies (including load balancers) lie between the WEF client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Troubleshoot

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

High CPU Usage on Leader Node API Process

Problem: Experiencing high CPU usage on your Leader Node's API Process is often a symptom of having too many subscriptions (thousands). This can lead to performance issues and instability.

Solutions: First attempt to reduce or consolidate your subscriptions. This can often alleviate the problem without the need for additional configurations. However, since the process of consolidating subscriptions can vary significantly between environments, there is no one-size-fits-all approach.

If you are unable to reduce the number of subscriptions sufficiently, consider enabling Write-Ahead Log (WAL) mode for the key-value store (KV Store). We recommend working with Cribl Support to incorporate the new **KV Store** settings effectively. You can find these settings under **Settings** > **Global Settings** > **General Settings** > **Limits** or in the limits.yml file.

Security Event Log Forwarding Fails

Error: `Error "0x138C" Windows Event Forward plugin can't read any event from the query since the query returns no active channel.`

Solution: Enable the Network Service to read the security logs by configuring access for the Event Log Service.

When configuring a Group Policy Object (GPO) to add NETWORK SERVICE permissions, carefully review existing permissions to prevent unintended removal. Preserve all necessary permissions to avoid application issues.

Cannot Find the Certificate

Error: `The forwarder is having a problem communicating with subscription manager at address... The WS-Management service cannot find the certificate that was requested.`

For example:

```
The forwarder is having a problem communicating with subscription manager at address

Error code is 2150858882 and Error Message is <f:WSManFault xmlns:f=http://schemas.mi
```

Copy

Resolution:

- The `NETWORK SERVICE` user does not have permissions to use the private key of the certificate for authentication.
- Add the `NETWORK SERVICE` user to the list of users allowed to use the private key.

Receiver Refuses to Accept Delivery of Events

Warning: `The receiver refuses to accept delivery of events and requests that the subscription be canceled.`

If you make a change to a subscription that is in use, **and** if the Windows client tries to send events before the `Refresh=XX` interval has elapsed, the result is that these events are sent to the old subscription, which is now invalid because its query is outdated or because other subscription parameters have changed.

The warning message tells the WEF client to stop sending events to the now-invalid old subscription.

Resolution:

Windows Event Forwarder handles this scenario gracefully, and you do not need to take any action to resolve it:

- Once the `Refresh=XX` interval has elapsed, the WEF client will ask for and receive the updated subscription, and from then on should be able to successfully deliver events against that new subscription.
- Any events that were refused under the old/invalid subscription ID will be resent under the new one, assuming that they remain valid according to the new subscription parameters.

Subscription Cannot be Created

Error: `The subscription <subscription> can not be created. The error code is 5004.`

The error appears in the `Microsoft-Windows-Eventlog-ForwardingPlugin/Operational` log.

Resolution:

1. Apply the `O:BAG:SYD:...` permissions to make the log file readable.
2. Reboot the machine.

Verify that Client and Destination are Joined to a Domain

Error: `If Kerberos mechanism is used, verify that the client computer and the destination computer are joined to a domain.`

When you are using auto-enroll, the `Eventlog-ForwardingPlugin Operational` logs might display an error of this form.

Resolution:

In your auto-enroll template's **Client-Server Authentication Properties**, make sure the **Subject Name format** is set to **Common name**.

Resources

For processing Pipelines that you can import and adapt to your needs, examine these resources on the Cribl Packs Dispensary:

- Microsoft Windows Events - Shape events into JSON or Key=Value format, and dramatically reduce event sizes.
- Cribl Edge Windows for Common Schema - Transform Windows events into a Common Schema.
- Exabeam Pack for Windows - Forward specific Windows Event IDs to Exabeam.
- QRadar Pack for Windows Events - Optimize Windows events for delivery to QRadar.

Further Reading

8.1.28.1. Client Certificate Authentication For Windows Event Forwarder

This document covers the client certificate authentication setup steps you need to take before configuring a Windows Event Forwarder Source.

1. Set Up WEF

Follow this Microsoft guide for setting up Windows Event Forwarding (WEF) in a traditional Windows environment. Generally, you can follow the guide's "non-domain" section to correctly configure the endpoints/senders.

2. Configure Certificates

After setting up WEF, you need to configure certificates to ensure secure communication. Server certificate configuration varies depending on your deployment type. However, you'll always need to configure a client certificate.

On-prem deployments: Add the server certificate and CA chain to the Worker Group, see importing certificates for details. Add the client certificate to Windows either manually or with certificate auto-enrollment through Group Policy to automatically configure unique device certificates. For specific configuration options, consult with your Windows administrators.

Cribl.Cloud deployment: Upload the CA chain for your client certificate, see importing certificates for details. The actual certificate/key pair in the chain doesn't matter for this purpose. It's solely to satisfy the technical requirement of uploading a chain. Use Cribl's provided certificate and key:

- `/opt/criblcerts/criblcloud.crt`
- `/opt/criblcerts/criblcloud.key`

3. Set Client Certificate Permissions

Set the appropriate permissions for the `NETWORK SERVICE` user to use the private key of the certificate for authentication.

Open the Certificate Manager tool (`certlm.msc`).

- Right-click your client certificate and select **All Tasks** > **Manage Private Keys...**.
- Add the `NETWORK SERVICE` user.

4. Configure the Subscription Manager and Log Forwarding Group Policies

1. On a domain controller, open the Group Policy Management Console (GPMC).
2. Create a new Group Policy Object (GPO) or edit an existing one that is linked to the organizational unit (OU) containing the computers you want to configure.
3. Navigate to **Computer Configuration** > **Policies** > **Administrative Templates** > **Windows Components**. Select **Event Forwarding** to open the Local Group Policy Editor.
4. Select **Configure target Subscription Manager** and use one of the following templates depending on your deployment type, substituting the appropriate values for the placeholders.
   On-Prem:

   ```
   Server=https://<cribl-worker-or-lb>:<wef-source-port>/wsman/SubscriptionManager/
   ```

   Copy

   Cribl.Cloud:

   ```
   https://<groupName>.main.<org-name>.cribl.cloud:<wef-source-port>/wsman/Subscrip
   ```

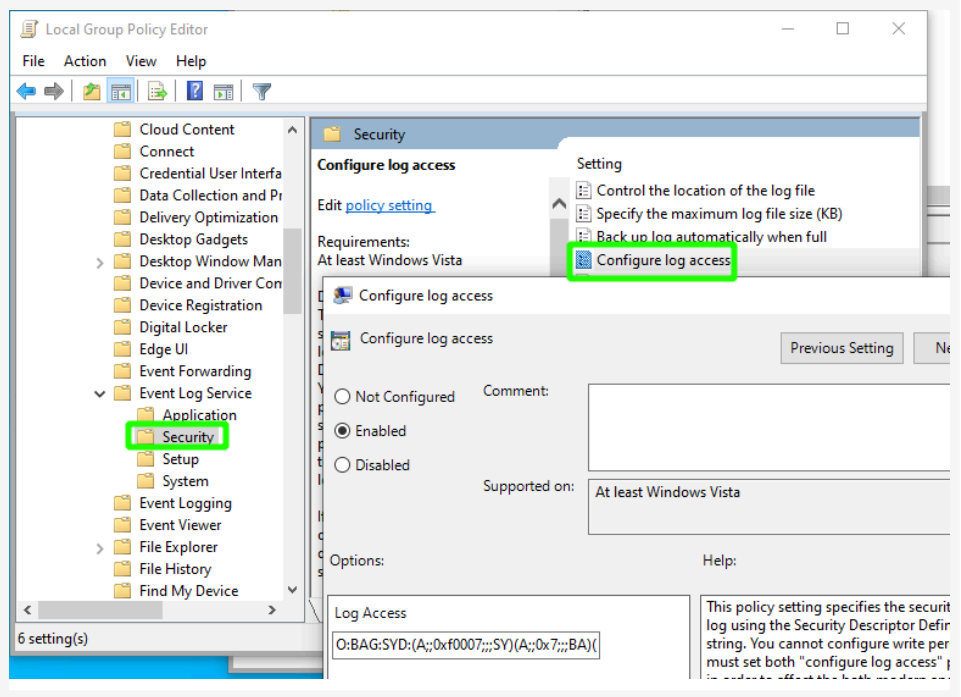
   Copy

   - The path portion `/wsman/SubscriptionManager/WEC` within the **Subscription Target URL** set using Group Policy Objects (GPOs) is the same as required for a WEC subscription and is strictly case-sensitive. If the capitalization is incorrect, WEF will encounter errors and fail to forward events to Cribl Stream.
   - The `IssuerCA` thumbprint is the SHA1 thumbprint of the issuing certificate, which could be an intermediate certificate. This must match the thumbprint of the first certificate in the certificate chain configured in the WEF Source's CA certificate path or the specified **CA fingerprint override**.
5. When complete, save the policy and apply it to affected endpoints. Link the GPO to the appropriate OU if it is not already linked.
6. To enable the Network Service to read the security logs, you'll need to configure access for the Event Log Service.

> When configuring a Group Policy Object (GPO) to add NETWORK SERVICE permissions, carefully review existing permissions to prevent unintended removal. Preserve all necessary permissions to avoid application issues.

1. Open the Local Group Policy Editor tool (`gpedit.msc`).

2. Navigate to **Computer Configuration** > **Policies** > **Administrative Templates** > **Windows Components** > **Event Log Service**. Double-click **Security**, then in the **Settings** pane, select **Configure log access**.



Configuring log access

3. In the resulting modal, under **Options** > **Log Access**, enter the following Log Access configuration:

```
O:BAG:SYD:(A;;0xf0007;;;SY)(A;;0x7;;;BA)(A;;0x1;;;BO)(A;;0x1;;;SO)(A;;0x1;;;S-1-
```

Copy

4. Then, reboot your Windows machine to apply this setting. (Source: Microsoft's Security event log forwarding fails... topic.)

7. Check that events are flowing into Cribl Stream now according to the configured subscriptions. On client machines, run `gpupdate /force` to apply the new policy immediately. If events are not flowing, verify all of the following:
   - The clients can reach the Cribl Stream instance through the network to port `5986` (or other configured port) via TLS/HTTP. If clients are connecting to Cribl Stream via a proxy, you may need to toggle **Show originating IP** on (in the Advanced section of the Source configuration). Ensure that the correct outbound firewall port is opened on the client.
   - The certificate chain is correct
   - The endpoints have a valid CRL encompassing the CA cert
   - The CA cert is a trusted root on the clients

- The server and client certs are issued by the same CA. The `CAPI2` Windows event log might reveal any errors here.
- Check Cribl Stream for any errors, as well as the `EventForwarding-Plugin` and `Windows Remote Management` event logs on the clients.

Now you can provide client certificate authentication to a Windows Event Forwarder Source.

8.1.28.2. Kerberos Authentication For Windows Event Forwarder

This document covers the Kerberos authentication setup steps you need to take before configuring a Windows Event Forwarder Source.

Kerberos authentication is only supported on x86_64 Linux systems.

Initial Assumptions

To ensure a smooth configuration process, review the following prerequisites:

- The clients that will be sending events are in a Windows domain. In this example, the domain is `contoso.com`, the NETBIOS name is `CONTOSO`, and the domain controller is `DC01.contoso.com`.
- Kerberos KDC is already configured and working on the domain, and WEC with Kerberos authentication already works on the domain.
- You have an x86_64 Worker Node to which you have shell access. This guide assumes a recent Ubuntu distribution.
- The fully qualified domain name (FQDN) of this example Worker Node is `worker1.contoso.com`.
- You are using a Kerberos credential cache of the `DIR` or `FILE` type.

1. Initial Setup for the Worker Node

To set up the Worker Node:

1. Ensure that the Worker Node can resolve the domain controller's FQDN. You can test it with: `ping DC01.contoso.com`.
2. Ensure that the Worker Node's `hostname` is set to the FQDN. If it's not, you can set it with `hostnamectl set-hostname worker1.contoso.com`.
3. Ensure the Worker Node has its clock synchronized with the domain controller.
4. Install the `krb5-user` package and configure Kerberos to correctly identify your domain. Here's an example configuration (`/etc/krb5.conf`).

```
[libdefaults]
default_realm = CONTOSO.COM

[realms]
  CONTOSO.COM = {
    kdc = DC01.contoso.com
    admin_server = DC01.contoso.com
  }

[domain_realm]
  .contoso.com = CONTOSO.COM
  contoso.com = CONTOSO.COM
```

Copy

2. Setup for the Domain Controller

Configure the domain controller to support Kerberos authentication for your Worker Node. This involves setting up DNS records and creating a domain user with the necessary permissions and encryption settings. Proper configuration ensures that the Worker Node can securely authenticate and communicate within your network.

To set up the domain controller:

1. Set up forward and reverse DNS for the Worker Node's FQDN (that is, both A and PTR records).
2. Both `nslookup worker1.contoso.com` and `nslookup <worker's IP>` should work.
3. Create a domain user for the Worker Node. This guide assumes the user is called `svc-stream-worker1`.
4. Set the user's password to never expire.
5. Under the user's **Properties** > **Account** tab, set both `This account supports Kerberos AES 128 bit encryption` and `This account supports Kerberos AES 256 bit encryption`.
6. Create a keytab file for the Worker Node. This file will contain the necessary credentials for the Worker Node to authenticate with the domain controller. See the Keytab Export section below for instructions on how to generate the keytab file.

Keytab Export

This section guides you through exporting a keytab file, which is essential for Kerberos authentication. The keytab file contains the necessary credentials for the Worker Node to authenticate with the domain controller. You will generate this file using the `ktpass` command and then transfer it to the Worker Node. This step is

crucial for enabling secure, authenticated communication between your Worker Node and the domain controller.

To export the keytab:

1. Export a keytab file for this user, which will also create the appropriate service principal name (SPN). The keytab will be exported to the root of the user profile running the command.
2. Open an elevated PowerShell prompt on the domain controller.
3. Run the following ktpass command to generate the keytab file: `ktpass /princ http/worker1.contoso.com@CONTOSO.COM /pass <password> /mapuser CONTOSO\svc-stream-worker1 /crypto AES256-SHA1 /ptype KRB5_NT_PRINCIPAL /out worker1.contoso.com.keytab`
    - `ktpass`: This is the command-line utility used to create keytab files for Kerberos authentication.
    - `/princ http/worker1.contoso.com@CONTOSO.COM`: Specifies the principal name. In this case, it is `http/worker1.contoso.com@CONTOSO.COM`. The principal name is a unique identifier for the service.
    - `/pass <password>`: Specifies the password for the user account. Replace `<password>` with the actual password for the `svc-stream-worker1` user.
    - `/mapuser CONTOSO\svc-stream-worker1`: Maps the principal name to the specified domain user account. Here, `CONTOSO\svc-stream-worker1` is the domain user created for the Worker Node.
    - `/crypto AES256-SHA1`: Specifies the encryption type to be used. `AES256-SHA1` is a strong encryption algorithm.
    - `/ptype KRB5_NT_PRINCIPAL`: Specifies the principal type. `KRB5_NT_PRINCIPAL` indicates that the principal is a Kerberos principal.
    - `/out worker1.contoso.com.keytab`: Specifies the output file name for the keytab. The keytab file will be saved as `worker1.contoso.com.keytab`.

Windows SPN compatibility

The SPN prefix you use in the `ktpass /princ` parameter must match what your Windows clients use when connecting. The example here uses `http/worker1.contoso.com@CONTOSO.COM`, which works in most environments, but some newer Windows versions (such as Windows Server 2025) require `host/worker1.contoso.com@CONTOSO.COM` instead.

To maximize compatibility across Windows versions, register both `http/worker1.contoso.com` and `host/worker1.contoso.com` SPNs on the account running the Worker Node. If Kerberos authentication fails with the `http/` example,

> regenerate the keytab using `/princ host/worker1.contoso.com@CONTOSO.COM` so it matches the SPN your clients require.

4. Note the SPN created here (in this example, `http/worker1.contoso.com@CONTOSO.COM`) as you will need it later. This SPN is case-sensitive.
5. Copy the keytab file to the Cribl Stream Worker Node. Carefully note the file's location on the Worker Node, because you will need it later. The keytab command and output will resemble this example:

```
PS C:\Users\Administrator> ktpass /princ http/wefkerb.weftest.local@WEFTEST.LOCAL /pa
Targeting domain controller: SUP-DC01.weftest.local
Using legacy password setting method

Successfully mapped http/wefkerb.weftest.local@WEFTEST.LOCAL to cribl.
Key created.

Output keytab to wefkerb.weftest.local.keytab:
Keytab version: 0x502
keysize 110 http/wefkerb.weftest.local@WEFTEST.LOCAL ptype 1 (KRB5_NT_PRINCIPAL) vno
```

Copy

> If you rotate passwords on this user service account, you'll need to export a new keytab file and merge or replace the keytab on each Worker Node. Microsoft describes a general method for this task. The Microsoft method is for a different service, but it explains the general principles. You could also use a tool like `msktutil`.

3. Setup for the Windows Event Forwarding Target

To set up the Windows event forwarding target, you'll need to create group policy objects (GPOs) for the subscription target, the WinRM service, and the NETWORK SERVICE.

Create a GPO for the Subscription Target

Create a Group Policy Object (GPO) to configure the subscription target for Windows Event Forwarding (WEF). This GPO will define the target server and the refresh interval for event subscriptions.

1. Access the UI **Computer Configuration** > **Administrative Templates** > **Windows Components** > **Event Forwarding** > **Configure target Subscription Manager**.
2. Set to `Enabled`.
3. Go to **Subscription Managers** > **Show**.

4. Add `Server=http://worker1.contoso.com:5985/wsman/SubscriptionManager/WEC,Refresh=60` to the list.

Create a GPO for the WinRM Service

Create a Group Policy Object (GPO) to ensure that the Windows Remote Management (WinRM) service is automatically started on client machines. This is a crucial step for enabling remote management and communication between the clients and the Worker Node. By setting the WinRM service to start automatically, you ensure that the necessary remote management capabilities are always available.

1. Access the UI **Computer Configuration** > **Policies** > **Windows Settings** > **Security Settings** > **System Services** > **Windows Remote Management (WS-Management)**.
2. Set **Startup type** to `Automatic`.

Create a GPO for the NETWORK SERVICE

Create a Group Policy Object (GPO) to grant the NETWORK SERVICE account the necessary permissions to read event logs. This is particularly important if you plan to subscribe to the Security event log. Properly configuring these permissions ensures that the `NETWORK SERVICE` account can access and forward the required event logs to the Worker Node.

1. Access the UI **Computer Configuration** > **Policies** > **Windows Settings** > **Security Settings** > **Restricted Groups**.
2. In the right pane, right-click and select **Add Group**.
3. Enter `Event Log Readers` as the group name.
4. Under `Members of this group`, click **Add**.
5. Enter `NETWORK SERVICE` as the member name.

8.1.29. Wiz

You can use the following Sources to receive data from Wiz.

Source	Description
Wiz API	Receive data from the Wiz cloud security platform.
Wiz Webhook	Receive Wiz Defend Alert data.

8.1.29.1. Wiz API

Cribl Stream supports collecting data from the Wiz cloud security platform. The Wiz API Source will communicate with APIs that your organization's Wiz portal exposes: Audit Logs, Configuration Findings (sometimes called Cloud Configuration), Issues, and Vulnerabilities.

Type: **Pull** | TLS Support: **No** | Event Breaker Support: **No**

The Wiz API Source polls the Wiz API using the `createdAt` timestamp to fetch new events. If an already ingested event, such as an Issue, is updated in Wiz, the Wiz API Source won't recognize these changes because it doesn't track the `updatedAt` timestamp. To capture these modifications, you can use a custom REST Collector. You can adapt the existing Wiz REST Collector template, ensuring you modify it to utilize the `updatedAt` field in your API calls and event breaking logic instead of relying on `createdAt`.

Prerequisites

Reference the Wiz integration documentation for gathering your Wiz account's unique GraphQL API endpoint URL and authentication information. You'll provide these in the Cribl Stream Wiz API Source configuration.

Configuring a Wiz API Source

1. From the top nav, click **Manage**, then select a **Worker Group** to configure. Next, you have two options:
    - To configure via the graphical QuickConnect UI, click **Routing** then **QuickConnect**. Next, click **Add Source** and from the resulting drawer's tiles, select **Wiz**. Next, click either **Add Source** or (if displayed) **Select Existing**.
    - To configure via the Routing UI, click **Data** then **Sources**. Select **Wiz** from the list of tiles or the **Sources** left nav. Next, click **Add Source** to open the **Add Source** modal.
2. In the Source modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Wiz API Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **GraphQL endpoint**: Enter the Wiz GraphQL API endpoint URL. The UI prefills with a URL that you need to update with the appropriate `<region>`, `https://api.<region>.app.wiz.io/graphql`. For example, `https://api.us1.app.wiz.io/graphql`.

- **Authentication URL**: Select the authentication URL to generate an OAuth token. For details, see Authentication URL below.
- **Client ID**: Enter the Wiz **Client ID** (53-character string).
- **Authentication method**: Enter the Wiz **Client secret** (64-character string) password credentials manually or with a secret. When choosing **Secret**, you'll select a stored text secret in the resulting **Client Secret (text secret)** drop-down, or click **Create** to configure a new secret.
- **Content types**: Toggle **Enable content** on for the API types you want to query. For details, see Content Types below.
- **Tags**: Optionally, add tags that you can use to filter and group Sources in in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

3. The remaining options, Processing Settings, Retries, and Advanced Settings, are already configured for Wiz data collection.
4. Click **Save**, then **Commit & Deploy**.
5. Verify that data is making it to Cribl Stream by viewing the **Live Data** feed from the Source.

Authentication URL

If you're using a custom or self-hosted identity provider, enter the specific URL provided by your organization.

Authentication audience: When using a custom **Authentication URL** you can define the audience to use when requesting an OAuth token. This is used to validate that the token is meant for your application. Enter the audience value required by your identity provider. Defaults to `wiz-api`.

Content Types

The **Content types** are:

- **Audit Logs**
- **Configuration Findings**
- **Issues**
- **Vulnerabilities**

The schedule and query are automatically configured for you. See scheduling options for details on how these options work.

Toggle **State tracking** on to track state by time, which prevents overlaps between collection jobs where subsequent jobs may return some of the same results as previous jobs. Similarly, it can help prevent gaps in data by allowing the next run to pick up from where the last run ended.

Processing Settings

- **Fields**: You can add Fields to each event, using Eval-like functionality. A field consists of a **Name** and **Value** pair. The **Value** is a JavaScript expression and can be a constant.
- **Pre-Processing**: Select a **Pipeline** to process results before sending to Routes. Optional, and available only when **Send to Routes** is toggled on.

Retries

Optionally adjust the default retry settings for failed collect jobs.

- **Retry type**: The algorithm to use when performing HTTP retries. Options include `Backoff` (the default), `Static`, and `Disabled`.
- **Initial retry interval (ms)**: Time interval between failed request and first retry (kickoff). Maximum allowed value is 20,000 ms (1/3 minute). A value of `0` means retry immediately until reaching the limit specified in **Retry limit**.
- **Retry limit**: Maximum number of times to retry a failed HTTP request. Defaults to `5`. Maximum: `20`. A value of `0` means don't retry at all.
- **Backoff multiplier**: Base for exponential backoff. A value of `2` (default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so forth.
- **Retry HTTP codes**: List of HTTP codes that trigger a retry. Leave empty to use the defaults (`429` and `503`). Cribl Stream does not retry codes in the `200` series.
- **Honor Retry-After header**: When toggled on (the default), `retry-after` headers up to a maximum of 20 seconds are processed. Delays longer than 20 seconds are ignored.
    - Cribl Stream will log a warning message with the delay value retrieved from the `retry-after` header (converted to ms).
    - When toggled off, Cribl Stream ignores all `retry-after` headers.
- **Retry connection timeout**: Toggle on to automatically retry a single connection attempt after a timeout (`ETIMEDOUT`) to ensure data continuity.
- **Retry connection reset**: Toggle on to automatically retry a connection after a peer reset (`ECONNRESET`) to maintain data flow.

Advanced Settings

Advanced Settings enable you to customize post-processing and administrative options.

- **Request timeout (seconds)**: The maximum time period for an HTTP request to complete before Cribl Stream treats it as timed out. Defaults to `0`, which disables timeout metering.
- **Time to live**: How long to keep the job's artifacts on disk after job completion. This also affects how long a job is listed in **Job Inspector**. Defaults to `4h`.

- **Environment**: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Connected Destinations

Connected Destinations mimics which method you used to open the Wiz configuration modal in step 1, either the QuickConnect UI or the Routes UI. **Send to Routes** enables conditional routing, filtering, and cloning of this Source's data via the Routing table. **QuickConnect** allows you to send this Source's data to one or more Destinations via independent, direct connections.

Scheduling

When you enable a **Content type** under **General Settings** the following scheduling options are available:

The Cron schedule configures when collection requests are made.

Job timeout: Maximum time this job will be allowed to run. Units are seconds, if not specified. Sample values: `30`, `45s`, or `15m`. Minimum granularity is 10 seconds, so a `45s` value would round up to a 50-second timeout. Defaults to `0`, meaning unlimited time (no timeout).

Log Level: Level at which to set task logging. More verbose levels are useful for troubleshooting jobs and tasks, but use them sparingly.

The **Earliest time** and **Latest time** defines the time range of events to collect, based on the `_time` field. These fields accept the following syntax:

```
[+|-]<time_integer><time_unit>@<snap-to_time_unit>
```

To break down this syntax:

Syntax Element	Values Supported
Offset	Specify: `-` for times in the past, `+` for times in the future, or omit with `now`.
<time_integer>	Specify any integer, or omit with `now`.
<time_unit>	Specify the `now` constant, or one of the following abbreviations: `s[econds]`, `m[inutes]`, `h[ours]`, `d[ays]`, `w[eeks]`, `mon[ths]`, `q[uarters]`, `y[ears]`.
@<snap-to_time_unit>	Optionally, you can append the `@` modifier, followed by any of the above `<time_unit>`s, to round down to the nearest instance of that unit. (See the next section for details.)

Cribl Stream validates relative time values using these rules:

- **Earliest** must not be later than **Latest**.

- Values without units get interpreted as seconds. For example, `-1 = -1s`.

Snap-to-Time Syntax

The `@` snap modifier always rounds **down** (backwards) from any specified time. This is true even in relative time expressions with `+` (future) offsets. For example:

- `@d` snaps back to the beginning of today, 12:00 AM (midnight).
- `+128m@h` looks forward 128 minutes, then snaps back to the nearest round hour. (If you specified this in the `Latest` field, and ran the Source at 4:20 PM, collection would end at 6:00 PM. The expression would look forward to 6:28 PM, but snap back to 6:00 PM.)

Other options:

- `@w` or `@w7` to snap back to the beginning of the week - defined here as the preceding Sunday.
- To snap back to other days of a week, use `w1` (Monday) through `w6` (Saturday).
- `@mon` to snap back to the 1st of a month.
- `@q` to snap back to the beginning of the most recent quarter - Jan. 1, Apr. 1, Jul. 1, or Oct. 1.
- `@y` to snap back to Jan. 1.

Working with State Tracking

You can configure the Source to track state, either by time or another arbitrary value. This can help prevent overlaps between jobs, where subsequent runs may return some of the same results as previous runs. Similarly, it can help prevent gaps in data by allowing a run to pick up from where the last run ended.

State update expression: JavaScript expression that defines how to update the state from an event. Use the event's data and the current state to compute the new state.

State merge expression: JavaScript expression that defines which state to keep when merging a task's newly reported state with the previously saved state. Evaluates `prevState` and `newState` variables, resolving to the state to keep.

The default values for these fields are configured to track state by the latest `_time` field found in events gathered in a collection run.

Understanding State Expression Fields

The **State update** and **State merge** expressions control how state is derived from a collection run and how it is merged with existing state, respectively. They're preconfigured to work with the common use case of tracking state by latest `_time`, but you may need to update them for other use cases. To understand what these fields do, let's break down the default values.

State update expression

This expression has a default value of:

```
__timestampExtracted !== false && {latestTime: (state.latestTime || 0) > _time ? stat
```

Copy

__timestampExtracted !== false - the `__timestampExtracted` field is set to false if the Event Breaker was unable to parse time for the event. If this is the case, we don't want to update state (the event's `_time` value defaults to `Date.now()` if the Event Breaker was unable to parse out the correct time). If `__timestampExtracted` is `false`, we take advantage of short-circuit evaluation to not update state.

State values must resolve to an object, such as:

```
{ "latestTime": 17122806161 }
```

Copy

If the expression does not resolve to an object, Cribl Stream will ignore the result.

{latestTime: (state.latestTime || 0) > _time ? state.latestTime : _time} - compare `state.latestTime` to the event's `_time` value, keeping whichever value is greater.

State merge expression

This expression has a default value of:

```
prevState.latestTime > newState.latestTime ? prevState : newState
```

Copy

It compares `prevState` (the state that was previously saved) to `newState` (the state reported from the most recent collection task), keeping the state with the greatest `latestTime` value.

Managing State

Click **Manage State** to view, modify, or delete a state. For more information, see Manage State.

Wiz API Results Limits

The table below shows the results limits for each Wiz API.

API	Limit (results)
Audit Logs	10,000
Cloud Configuration	10,000
Issues	no limit
Vulnerability	no limit

Once your query hits the limit of results for a given Wiz API, the API will send no further results.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Internal Fields

Cribl Stream uses a set of internal fields to assist in data handling. These "meta" fields are **not** part of an event, but they are accessible, and you can use them in Functions to make processing decisions.

Relevant fields for this Source:

- `__collectible` - This object's nested fields contain metadata about each collection job.
- `__collectStats` - This object's nested fields are `method`, `url`, and `elapsedMS`.

Troubleshooting

Navigate to and open the Wiz API Source, then use the Source's **Job Inspector** tab to see the results of recent collection runs.

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify ingestion issues. Analyze the graphs showing events and bytes in/out over time.

Response Errors

The Source treats all non-`200` responses from configured URL endpoints as errors. This includes `1xx`, `3xx`, `4xx`, and `5xx` responses.

A few exceptions are treated as non-fatal errors:

- Where a collect job launches multiple tasks, and only a subset of those tasks fail, Cribl Stream places the job in failed status, but treats the error as non-fatal. (Note that Cribl Stream does not retry the failed tasks.)
- Where a collect job receives a `3xx` redirection error code, it follows the error's treatment by the underlying library, and does not necessarily treat the error as fatal.

8.1.29.2. Wiz Webhook In Cribl Stream

Cribl Stream supports receiving Wiz Defend Alert data. Wiz Webhook, an HTTP-based Source, listens on a specific port, captures every HTTP request to that port, and creates a corresponding event that it pushes to its configured Event Breakers.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

Looking for the Wiz Webhook Source in **Cribl Search**? See Add a Wiz Webhook Source for Cribl Search.

Configure Cribl Stream to Receive Wiz Defend Alert Data

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Sources** (Stream). Select the Source you want. Next, select **Add Source**.
2. In the **New Source** modal, configure the following under **General Settings**:
    - **Input ID**: Enter a unique name to identify this Raw HTTP Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Enter the address to bind on. Defaults to `0.0.0.0` (all addresses).
    - **Port**: Enter the port number to listen on. For Cribl.Cloud, use the `20000 - 20010` port range.
3. Under **Authentication**, select **Add Token** to enter the **Auth tokens**:
    - **Token**: Shared secrets to be provided by any client. Click **Generate** to create a new secret. If empty, permits open access.
    - **Description**: Optionally, enter a description.
    - **Fields**: Fields to add to events referencing this token. Select **Add Field** to create a new field. Each field is a **Name/Value** pair, where **Value** is a JavaScript expression to compute field's value, enclosed in quotes or backticks. (Can evaluate to a constant.) For details, see Authentication Fields.
4. Next, you can configure the following **Optional Settings**:

- **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the TLS, Persistent Queue Settings, Processing and Advanced settings, or Connected Destinations outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Authentication Fields

By default, fields specified here will override fields of the same name in incoming events. However, you can customize this behavior to allow event fields to take precedence. For example, to allow the event's `index` field to take precedence, you could use this expression: `` `event.index || 'defaultIndex'` ``.

Fields add context to events based on the token used. This can be useful for tagging events with metadata such as source or environment, enhancing security by avoiding direct use of authorization values in Routes and Pipelines.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may

result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Event Breakers

Event Breaker rulesets: A list of event breaking rulesets that will be applied to the input data stream before the data is sent through the Routes. Defaults to `System Default Rule`.

Event Breaker buffer timeout: How long (in milliseconds) the Event Breaker will wait for new data to be sent to a specific channel, before flushing out the data stream, as-is, to the Routes. Minimum `10` ms, default `10000` (10 sec), maximum `43200000` (12 hours).

Fields

In this section, you can define new fields or modify existing ones using JavaScript expressions, similar to the Eval function.

- The **Field Name** can either be a new field (unique within the event) or an existing field name to modify its value.
- The **Value** is a JavaScript expression (enclosed in quotes or backticks) to compute the field's value (can be a constant). Select this field's advanced mode icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.

This flexibility means you can:

- Add new fields to enrich the event.
- Modify existing fields by overwriting their values.
- Compute logic or transformations using JavaScript expressions.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Advanced Settings

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

Health check endpoint: Toggle on to enable a health check endpoint specific to this Source, `http(s)://<host>:<port>/cribl_health`. A `200` HTTP response code is returned when the Source is healthy. Otherwise, two errors you could receive are:

- `ECONNRESET` where the Source failed to initialize due to not having listeners on the port.
- `503` or `Server is busy, max active connections reached` indicate there are too many connections per Worker Process.

Allowed URI paths: List of URI paths accepted by this input. Supports wildcards, for example, `/api/v*/hook`. Defaults to `*`, which allows all paths.

Allowed HTTP methods: List of HTTP methods accepted by this input. Supports wildcards, for example, `P*`, `GET`. Defaults to `*`, which allows all methods.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; etc.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and functions can use them to make processing decisions.

Fields for this Source:

- `__channel`
- `__headers` - Automatically includes any headers sent with request.
- `__inputId`
- `__srcIpPort` field.

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the HTTP client sending data to this Source.

When any proxies (including load balancers) lie between the HTTP client and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Examples

The examples in this section demonstrate sending HTTP data into a Raw HTTP Source. The token is set to `myToken42` and included in the `Authorization` header of requests.

Cribl Endpoint - Single Event

Replace `<myCriblHost>` with your **Public Ingress address**.

Cribl Endpoint, Single Event Example

```
curl -k "http://<myCriblHost>:10050/postPath" \
-H "Authorization: myToken42" \
-H "Content-Type: application/json" \
-d '{"_raw":"this is a sample event", "host":"myHost", "source":"mySource", "fieldA":
```

Copy

Cribl Endpoint - Multiple Events

Replace `<myCriblHost>` with your Public Ingress address.

Cribl Endpoint, Multiple Events Example

```
curl -k "http://<myCriblHost>:10050/postPath" \
-H "Authorization: myToken42" \
-H "Content-Type: application/json" \
-d $'{"_raw":"this is a sample event", "host":"myHost", "source":"mySource", "fieldA'
{"_raw":"this is another sample event", "host":"myOtherHost", "source":"myOtherSource
```

Copy

Cribl.Cloud - Single Event

Replace `<Your-Org-ID>` with your Cribl.Cloud Organization ID. Cribl.Cloud makes the `20000` - `20010` port range available for configuring Sources that don't have a default port, like the Raw HTTP Source.

Cribl.Cloud, Single Event Example

```
curl -k "http://default.main.<Your-Org-ID>.cribl.cloud:20001" \
-H "Authorization: myToken42" \
-d '{"_raw":"this is a sample event"}'
```

Copy

For Cribl.Cloud Enterprise plans, replace `default` in the request URL with the name of the Worker Group that you want to send to.

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate,

8.1.30. Zscaler Cloud NSS Source

The Zscaler Cloud NSS Source receives log data over HTTP/S from Zscaler Cloud Nanolog Streaming Service (NSS).

If you're sending from a Zscaler NSS Virtual Machine, use a Syslog Source over TCP.

Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

This Source supports gzip-compressed inbound data when the `Content-Encoding: gzip` connection header is set.

Zscaler Cloud NSS Feed

When adding a Zcaler Cloud NSS feed, configure the following:

- **API URL**: The URL varies depending on your Cribl environment. Use the same port and HEC endpoint you define in your Zscaler Cloud NSS Source.
    - **Cribl.Cloud**: Enter the Workspace **Public Ingress Address** in the following format: `https://<groupName>.<workspaceName>.<organizationId>.cribl.cloud:<port><hec-endpoint>`
    - **On-prem**: Enter the IP address or FQDN for either your Cribl Stream instance, or the load balancer you're using with your Cribl Stream instances.
- **HTTP Headers**:
    - **Key 1**: Enter `Authorization`.
    - **Value 1**: Enter the Auth Token from your Cribl Source.
    - **Key 2**: Enter `Content-Encoding`.
    - **Value 2**: Enter `gzip`.

Configure a Zscaler Cloud NSS Source

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Source** and select the Source you want from the list, choosing either **Select Existing** or **Add New**.

- To configure via the Routes, select **Data** > **Sources** (Stream) or **More** > **Sources** (Edge). Select the Source you want. Next, select **Add Source**.

2. Configure the following under **General Settings**:
   - **Input ID**: Enter a unique name to identify this Source definition. If you clone this Source, Cribl Stream will add `-CLONE` to the original **Input ID**.
   - **Description**: Optionally, enter a description.

   The next three settings configure the location where the Source listens for incoming HTTP(S) data.
   - **Address**: Enter the hostname/IP. Such as: `localhost` or `0.0.0.0`. Supports IPv4 and IPv6 addresses.
   - **Port**: Enter the network port.
   - **HEC endpoint**: Enter the absolute path. Defaults to `/services/collector`.
3. Next, you can configure the following **Optional Settings**:
   - **Allowed Indexes**: List the values allowed in the HEC event index field. Allows wildcards. Leave blank to skip validation.
   - **Zscaler HEC Acks**: Whether to respond with a `200` HTTP status code acknowledging receipt of the `ACK` request. If toggled off (default), the Source responds with a `400` code to indicate `ACK` processing is disabled.
   - **Tags**: Optionally, add tags that you can use to filter and group Sources in Cribl Stream's UI. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can enable TLS and persistent queue, then configure processing and advanced settings.
5. Create an Auth Token. You'll provide the token in an HTTP header with the key `Authorization`, see the Overview section for details.
6. Select **Save**, then **Commit & Deploy**.

TLS Settings (Server Side)

Enabled: Defaults to toggled off. When toggled on:

Certificate: Name of the predefined certificate.

Private key path: Server path containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path containing certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path containing CA certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

Authenticate client (mutual auth): Require clients to present their certificates. Used to perform mutual authentication using SSL certs. Default is toggled off. When toggled on:

- **Validate client certificates**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as, the system's CA). Default is toggled off.
- **Common name**: Regex that a peer certificate's subject attribute must match in order to connect. Defaults to `.*`. Matches on the substring after `CN=`. As needed, escape regex tokens to match literal characters. (For example, to match the subject `CN=worker.cribl.local`, you would enter: `worker\.cribl\.local`.) If the subject attribute contains Subject Alternative Name (SAN) entries, the Source will check the regex against all of those but ignore the Common Name (CN) entry (if any). If the certificate has no SAN extension, the Source will check the regex against the single name in the CN.

Minimum TLS version: Optionally, select the minimum TLS version to accept from connections.

Maximum TLS version: Optionally, select the maximum TLS version to accept from connections.

Persistent Queue Settings

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.
>
> The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

> `Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5 GB`. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

> In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).
>
> You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

Processing Settings

Fields

In this section, you can add Fields to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to determine field's value (can be a constant).

Fields specified on the **Fields** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

In particular, where incoming events have no `index` field, this Source adds one with the literal value `default`. You can override this value by using **Add Field** to specify an `index` field, and then setting its **Value** to an expression of the following form: `index == 'default' ? 'myIndex' : index`

Fields here are evaluated and applied **after** any fields specified in the Auth Tokens section.

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

Auth Tokens

Provides the authentication token required to access this Source's data. Ensure the token is valid and has the necessary permissions for data retrieval.

Not providing a token allows **unauthorized access**.

Authentication method: Choose between manually entering a **Token** or selecting a stored **Secret**. If you opt for manual entry, type the token directly or **Generate** a new one. For added security, use the secrets store to select a stored **Token (text secret)** or **Create** a new one.

Enable token: Controls the status of the specified authentication token. When toggled on, the specified auth token is active and can be used for client access. If toggled off, the token is disabled, causing it to be skipped and ignored during initialization. This means that any client attempting to use a disabled token will not be granted access.

Disabling a token can be helpful when you need to temporarily stop receiving data from a particular service or application. Disabling a token prevents the service or application from sending data to your endpoint without having to completely delete the token.

See also Periodic Logging for information on how auth tokens affect product logging.

Description: Optional (but recommended), the description for this token. Although this field is optional, the token description becomes the display name for the token. You can select this display name to expand or collapse the token details.

Allowed indexes: Optionally, specify which indexes are permissible for events ingested using this token. Supports multiple index values and wildcards.

Fields: Fields to add to events referencing this token. Each field is a **Name/Value** pair.

Fields specified on the **Auth Tokens** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

In particular, where incoming events have no `index` field, this Source adds one with the literal value `default`. You can override this value by using **Add Field** to specify an `index` field, and then setting its **Value** to an expression of the following form: `index == 'default' ? 'myIndex' : index`

Fields here are evaluated and applied **before** any fields specified in the Fields section.

Advanced Settings

Emit per-token request metrics: Toggle on to emit per-token `<prefix>.http.perToken` and summary `<prefix>.http.summary` request metrics. This enables Periodic Logging.

Show originating IP: Toggle on when clients are connecting through a proxy that supports the `X-Forwarded-For` header to keep the client's original IP address on the event instead of the proxy's IP address. This setting affects how the Source handles the `__srcIpPort` field.

Capture request headers: Toggle on to add request headers to events, in the `__headers` field.

CORS allowed origins: If you need to enable cross-origin resource sharing with Zscaler senders, use this field to specify the HTTP origins to which Cribl Stream should send `Access-Control-Allow-*` headers. You can enter multiple domains and use wildcards. This and its companion **CORS allowed headers** option should seldom be needed - see Working with CORS Headers below.

CORS allowed headers: List HTTP headers that Cribl Stream will send in a CORS preflight response to your configured (above) **CORS allowed origins** as `Access-Control-Allow-Headers`. Enter `*` to allow all headers.

Active request limit (on-prem only): Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Raising this limit can increase throughput by allowing more concurrent data requests, but increases resource usage and load on both your Cribl Stream infrastructure and on downstream Destinations. Before raising this limit, ensure:

- Your Cribl Stream deployment has sufficient capacity to support higher request concurrency, including CPU, memory, or number of Worker Processes.
- Downstream Destinations are correctly sized and tuned to accept the higher data ingest rate, preventing backpressure. See Manage Backpressure for more information.

Improper sizing on either side can result in dropped events, delayed processing, or overall system instability.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; and so forth.

Requests-per-socket limit: The maximum number of requests Cribl Stream should allow on one socket before instructing the client to close the connection. Defaults to `0` (unlimited). See Balancing Connection Reuse Against Request Distribution below.

Socket timeout (seconds): How long Cribl Stream should wait before assuming that an inactive socket has timed out. The default `0` value means wait forever.

Request timeout (seconds): How long to wait for an incoming request to complete before aborting it. The default `0` value means wait indefinitely.

Keep-alive timeout (seconds): After the last response is sent, Cribl Stream will wait this long for additional data before closing the socket connection. Defaults to `5` seconds; minimum is `1` second; maximum is `600` seconds (10 minutes).

The longer the **Keep-alive timeout**, the more Cribl Stream will reuse connections. The shorter the timeout, the closer Cribl Stream gets to creating a new connection for every request. When request frequency is high, you can use longer timeouts to reduce the number of connections created, which mitigates the associated cost.

IP allowlist regex: Grants access to requests originating from specific IP addresses that match a defined pattern. Unmatched requests are rejected with a 403 (Forbidden) status code. Defaults to `.*` (allow all).

IP denylist regex: Blocks requests originating from specific IP addresses that match a defined pattern, even if they would be allowed by default. Rejected requests receive a 403 (Forbidden) status code. Defaults to `^$` (allow all).

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Balancing Connection Reuse Against Request Distribution

Requests-per-socket limit allows you to limit the number of HTTP requests an upstream client can send on one network connection. Once the limit is reached, Cribl Stream uses HTTP headers to inform the client that it must establish a new connection to send any more requests. (Specifically, Cribl Stream sets the HTTP `Connection` header to `close`.) After that, if the client disregards what Cribl Stream has asked it to do and tries to send another HTTP request over the existing connection, Cribl Stream will respond with an HTTP status code of `503 Service Unavailable`.

Use this setting to strike a balance between connection reuse by the client, and distribution of requests among one or more Worker Node processes by Cribl Stream:

- When a client sends a sequence of requests on the same connection, that is called connection reuse. Because connection reuse benefits client performance by avoiding the overhead of creating new connections, clients have an incentive to **maximize** connection reuse.
- Meanwhile, a single process on that Worker Node will handle all the requests of a single network connection, for the lifetime of the connection. When receiving a large overall set of data, Cribl Stream performs better when the workload is distributed across multiple Worker Node processes. In that situation, it makes sense to **limit** connection reuse.

There is no one-size-fits-all solution, because of variation in the size of the payload a client sends with a request and in the number of requests a client wants to send in one sequence. Start by estimating how long connections will stay open. To do this, multiply the typical time that requests take to process (based on payload size) times the number of requests the client typically wants to send.

If the result is 60 seconds or longer, set **Requests-per-socket limit** to force the client to create a new connection sooner. This way, more data can be spread over more Worker Node processes within a given unit of time.

For example: Suppose a client tries to send thousands of requests over a very few connections that stay open for hours on end. By setting a relatively low **Requests-per-socket limit**, you can ensure that the same work is done over more, shorter-lived connections distributed between more Worker Node processes, yielding better performance from Cribl Stream.

A final point to consider is that one Cribl Stream Source can receive requests from more than one client, making it more complicated to determine an optimal value for **Requests-per-socket limit**.

Connected Destinations

Select **Send to Routes** to enable conditional routing, filtering, and cloning of this Source's data via the Routing table.

Select **QuickConnect** to send this Source's data to one or more Destinations via independent, direct connections.

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and Functions can use them to make processing decisions.

Fields for this Source:

- `__headers` - Added only when **Advanced Settings** > **Capture request headers** is toggled on.
- `__hecToken`
- `__inputId`
- `__ctrlFields`
- `__srcIpPort` - See details below.
- `__TZ`

Overriding `__srcIpPort` with Client IP/Port

The `__srcIpPort` field's value contains the IP address and (optionally) port of the Zscaler Cloud NSS sending data to this Source.

When any proxies (including load balancers) lie between the Zscaler Cloud NSS and the Source, the last proxy adds an `X-Forwarded-For` header whose value is the IP/port of the original client. With multiple proxies, this header's value will be an array, whose first item is the original client IP/port.

If `X-Forwarded-For` is present, and **Advanced Settings** > **Show originating IP** is toggled off, the original client IP/port in this header will override the value of `__srcIpPort`.

If **Show originating IP** is toggled on, the `X-Forwarded-For` header's contents will **not** override the `__srcIpPort` value. (Here, the upstream proxy can convey the client IP/port without using this header.)

Working with CORS Headers

The **CORS allowed origins** and **CORS allowed headers** settings are needed **only** when you want JavaScript code running in a web browser to send events to the Zscaler Cloud NSS Source. In this (uncommon) scenario, the code in question sends a preflight request that includes CORS (Cross-Origin Resource Sharing) headers, and the Zscaler Cloud NSS Source includes CORS headers in its response.

If the settings (and thus the CORS headers in the response) are correct, the browser will allow the code in question to complete requests to the Zscaler Cloud NSS Source - that is, allow the code to read the Source's responses. (These settings are optional because for some web applications it's sufficient to send requests **without** reading responses.)

The CORS header dance begins when the web browser sends a request with a header stating its origin. What happens next depends on how you've configured the CORS header settings, as explained in the tables below. (For these examples, we'll assume an origin of `https://mycoolwebapp:3001`.)

CORS allowed origins value	What Zscaler Cloud NSS Source does	Does Browser then allow code to read response?
`*`	Send `Access-Control-Allow-Origin: *` header.	Yes
Multiple wildcard values	If origin value matches any wildcard, send `Access-Control-Allow-Origin: https://mycoolwebapp:3001` header.	Yes
Nothing (not configured)	Send neither an `Access-Control-Allow-Origin` nor an `Access-Control-Allow-Headers` header.	No

The web browser's request might also include an `Access-Control-Request-Headers` header that lists one or more headers that it wants to use in a subsequent `POST` request (that is, the "actual" as opposed to preflight request). The Zscaler Cloud NSS Source can then respond with an `Access-Control-Allow-Headers` header **if and only if** it's also sending the `Access-Control-Allow-Origin` header.

CORS allowed headers value	What Zscaler Cloud NSS Source does	Does Browser then allow code to read response?
`*`	Send `Access-Control-Allow-Headers` header with the same list the client sent.	Yes
Multiple wildcard values	If any headers from the client's list match wildcards, send `Access-Control-Allow-Headers` header listing the matching headers.	Yes
Nothing (not configured)	Do not send an `Access-Control-Allow-Headers` header.	Probably not

Periodic Logging

Cribl Stream emits metrics about incoming requests when **Advanced Settings** > **Emit per-token request metrics** is enabled. These metrics can be routed to a Destination by enabling the Cribl Metrics Source. There are two categories of metrics: per-token metrics provide details for each configured token, while summary metrics aggregate data across all tokens. If no tokens are configured, only summary metrics will be emitted.

Per-token metric names will be prefixed by `<basePrefix>.http.perToken.`, where `<basePrefix>` is the configured base prefix for the Cribl Metrics Source. The available metric names are listed below, excluding the prefix. As these are counter-type metrics, they will not be emitted if the value is zero.

- `bytesReceived`

- `numRequests`
- `numEvents`
- `numErrors` *Note: this is cumulative of specific error types below as well as a catch-all for other types*
- `numParserErrors`
- `numToDisabledToken`

The summary metric names will be prefixed by `<basePrefix>.http.summary.` where `<basePrefix>` is the configured base prefix for the Cribl Metrics Source. The available metric names are listed below, excluding the prefix. As these are counter-type metrics, they will not be emitted if the value is zero.

- `bytesReceived`
- `numRequests`
- `numEvents`
- `numErrors` *Note: this is cumulative of specific error types below as well as a catch-all for other types*
- `numParserErrors`
- `numToDisabledToken`
- `numAuthFailures`
- `numToWrongUrl`
- `numAckReqs`
- `numReqsAcked`

Troubleshooting

The Source's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they are ingested. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the ingestion process, including any errors or warnings that may have occurred.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate.

8.2. Destinations

Cribl Stream can send transformed data to various Destinations, including Cribl HTTP, Cribl TCP, Cribl Lake, Elasticsearch, Amazon Kinesis, Amazon S3 and other object stores, Prometheus and compatible services, InfluxDB, Splunk, Snowflake, Databricks, TCP JSON, and many others.

Destinations can write data to either IPv4 or IPv6 addresses.

Destination are grouped into categories that define how they handle unreachable outputs (backpressure events) and their load-balancing capabilities.

Flow of data from Sources into Cribl Stream and then on to Destinations.

Destination Categories

Destinations can be divided into streaming and non-streaming: those that accept events in real time, and those that batch them from a staging directory.

Other than that, each Destination can belong to one of the following (non-exclusive) categories:

- Filesystem-based
- Load-balanced

Streaming and Non-Streaming Destinations

Streaming and non-streaming Destination differ in the way they receive events:

- Streaming Destinations accept events in real time.
- Non-streaming Destinations receive events in batches from a staging directory.

Non-Streaming Destinations

Non-streaming Destinations make use of the staging directory, and have specific behavior regarding batching events into files.

Staging Directory

With non-streaming Destinations, Cribl Stream uses a staging directory in the local filesystem to format and write outputted events before sending them to configured Destinations. After a set of conditions is met, data is compressed and then moved to the final Destination.

To reduce costs when the staging directory is also the final directory, Cribl Stream avoids iterating through all the files within a directory by keeping an inventory of open (in progress) files in the staging directory's root. At startup, Cribl Stream will check for any leftover files in progress from prior sessions, and will ensure that they're moved to their final Destination. The process of moving to the final Destination is delayed after startup (default delay: 30 seconds). Processing of these files is paced at one file per service period (which defaults to 1 second).

In Cribl.Cloud, using a staging directory is only available on customer-managed hybrid Workers.

Batching Conditions

In non-streaming delivery, a file is closed and rolled out when it reaches its configured maximum:

- Size
- Open time
- Idle time

If a new file needs to be open, Cribl Stream will enforce the maximum number of open files by closing files in the order in which they were opened.

Filesystem-based Destinations

Some Destinations are Filesystem-based, which means they receive files on disk from a staging directory and batch them in a queue. When a batch of events is ready for transmission, Cribl Stream closes the file, optionally compresses it, and transmits the file to the downstream service. Filesystem-based Destinations do not support persistent queues.

Load-balanced Destinations

Certain Destinations offer built-in load-balancing capabilities.

Available Destinations

Cribl Stream supports the following Destinations. You can configure proxy servers for all HTTP-based Destinations.

Destination	Protocol	Streaming	Filesystem-Based	Load-Balanced
Amazon S3 Compatible Stores	HTTP/S	Non-streaming		
Amazon CloudWatch Logs	HTTP/S	Streaming		
Data Lakes > Cribl Lake	HTTP/S	Non-Streaming		
Data Lakes > Amazon S3	HTTPS only	Non-Streaming		
Data Lakes > Amazon Security Lake	HTTP/S	Non-Streaming		
Amazon Kinesis Data Streams	HTTP/S	Streaming		
Amazon MSK	TCP	Streaming		
Amazon SQS	HTTP/S	Streaming		
Azure Blob Storage	HTTPS only	Non-Streaming		
Azure Data Explorer	HTTPS only	Streaming or non-streaming		
Azure Event Hubs	TCP	Streaming		
Azure Monitor Logs	HTTPS only	Streaming		
Microsoft Sentinel	HTTP/S	Streaming		
ClickHouse	HTTP/S	Streaming		
Cloudflare R2	HTTP/S	Non-Streaming		
Cortex XSIAM	HTTP/S	Streaming		
Confluent Cloud	TCP	Streaming		
CrowdStrike Falcon LogScale	HTTPS only	Streaming		
CrowdStrike Falcon Next-Gen SIEM	HTTPS only	Streaming		
Datadog	HTTPS only	Streaming		
Databricks		Non-Streaming		
Dynatrace HTTP	HTTP/S	Streaming		

Destination	Protocol	Streaming	Filesystem-Based	Load-Balanced
Dynatrace OTLP	HTTP/S	Streaming		
Elastic Cloud	HTTPS only	Streaming		
Elasticsearch	HTTP/S	Streaming		
Exabeam	HTTP/S	Non-Streaming		
Filesystem/NFS		Non-Streaming		
Google Cloud Chronicle API	HTTPS only	Streaming		
Google Cloud Logging	HTTPS only	Streaming		
Google Cloud Pub/Sub	HTTPS only	Streaming		
Google Cloud Storage	HTTPS only	Non-Streaming		
Google SecOps	HTTPS only	Streaming		
Grafana Cloud	HTTP/S	Streaming		
Graphite	TCP or UDP	Streaming		
Honeycomb	HTTPS only	Streaming		
InfluxDB	HTTP/S	Streaming		
Kafka	TCP	Streaming		
Loki	HTTP/S	Streaming		
Microsoft Fabric Real-Time Intelligence	TCP	Streaming		
MinIO	HTTP/S	Non-Streaming		
NetFlow	UDP	Streaming		
New Relic Events	HTTPS only	Streaming		
New Relic Logs & Metrics	HTTPS only	Streaming		
OpenTelemetry (OTel)	gRPC or HTTP/S	Streaming		

Destination	Protocol	Streaming	Filesystem-Based	Load-Balanced
Prometheus	HTTP/S	Streaming		
SentinelOne AI SIEM	HTTP/S	Streaming		
SentinelOne DataSet	HTTPS only	Streaming		
ServiceNow Cloud Observability	gRPC or HTTP/S	Streaming		
SignalFx	HTTPS only	Streaming		
SNMP Trap	UDP	Streaming		
Splunk HEC	HTTP/S	Streaming		
Splunk Load Balanced	TCP	Streaming		
Splunk Single Instance	TCP	Streaming		
StatsD	TCP or UDP	Streaming		
StatsD Extended	TCP or UDP	Streaming		
Sumo Logic	HTTP/S	Streaming		
Syslog	TCP or UDP	Streaming		(TCP only)
TCP JSON	TCP	Streaming		
Wavefront	HTTPS only	Streaming		
Webhook	HTTP/S	Streaming		
Wiz Defend	HTTP/S	Streaming		

You can adapt the Amazon S3 Compatible Stores Destination to send data to downstream services like Snowflake, for which Cribl Stream currently has no preconfigured Destination. For details, please contact Cribl Support.

Internal Destinations

Internal Destinations are special-purpose Destinations that route data within your Cribl Stream deployment, or among Workers across distributed or hybrid Cribl.Cloud deployments. The following internal Destinations are

available:

- Default: Specify a default output from among your configured Destinations.
- Output Router: A "meta-Destination." Configure rules that route data to multiple configured Destinations.
- DevNull: Simply drops events. Preconfigured and active when you install Cribl Stream, so it requires no configuration. Useful for testing.
- Cribl HTTP: Send data among peer Worker Nodes over HTTP. Streaming and load-balanced.
- Cribl TCP: Send data among peer Worker Nodes over TCP. Streaming and load-balanced.
- Cribl Search: Send data from Worker Groups directly to Cribl Search over HTTP. Streaming and load-balanced.
- **SpaceOut**: This experimental Destination is undocumented. Be careful!

Data Delivery to Unreachable Destinations

Cribl Stream attempts to deliver data to all Destinations that are configured to receive it at least once. When a Destination is unreachable, there are three possible behaviors:

- **Block** - Cribl Stream will block incoming events.
- **Drop** - Cribl Stream will drop events addressed to that Destination.
- **Queue** - To prevent data loss, Cribl Stream will write events to a **persistent queue** disk buffer, then forward them when a Destination becomes available. (Available on several streaming Destinations.)

For further information about backpressure (a situation when a Destination receives more data than it can send), see Destination Backpressure Triggers.

You can configure your desired behavior through a Destination's **Backpressure Behavior** drop-down. Where other options are not displayed, Cribl Stream's default behavior is **Block**. For details about all the above behaviors and options, see Persistent Queues.

How Destinations Count Bytes Sent

Cribl Destinations record bytes sent per event by setting an internal field, `__outLen`, at the point where the Destination serializes or writes the payload for delivery. The calculation varies by Destination type. `__outLen` reflects uncompressed bytes. Compression is applied after `__outLen` is set, so metrics represent logical (uncompressed) size rather than on-the-wire compressed size.

Some Destinations can operate in multiple transport/mode categories. For example, Azure Data Explorer supports both streaming and batching, and OpenTelemetry and Dynatrace OTLP support HTTP and gRPC. These Destinations have clarifications in parenthesis in the `Destinations` column of the Byte Calculations table.

Byte Calculations

Destination Category	Byte Calculations	Destinations
HTTP and API-based Destinations	Measures the size of the data payload sent in the HTTP request (the uncompressed event data), calculated either event-by-event or during batch creation.	Amazon CloudWatch Logs, Azure Data Explorer (streaming mode), CrowdStrike Next-Gen SIEM, Cribl HTTP, Dynatrace (HTTP), Dynatrace (OTLP HTTP), Elastic, Elastic Cloud, Exabeam, Grafana Cloud, Google Cloud Logging, Google Pub/Sub, Honeycomb, Humio (HEC), InfluxDB, Microsoft Fabric, Microsoft Sentinel, New Relic (Events), OpenTelemetry (HTTP), SentinelOne AI SIEM, ServiceNow, Splunk HEC, Subscription, Wavefront
Batched HTTP serialization	Measures the event size during batch assembly. The final byte count is assigned from the size of the event's data within the batched request body.	Azure Monitor Logs, ClickHouse, Datadog, DataSet, Google Cloud Chronicle, Grafana Loki, Palo Alto Cortex XSIAM, Sumo Logic, Webhook
Object storage	Measures the size of the content before compression and writing to the storage medium.	Amazon S3, Amazon Security Lake, Azure Blob Storage, Azure Data Explorer (batching mode), Cloudflare R2, Cribl Lake, Databricks, Disk Spool, DL S3, Filesystem, Google Cloud Storage, MinIO, Ring
TCP-based outputs	Measures the actual bytes written to the TCP network buffer for each event.	Cribl TCP, Graphite, Syslog (TCP), TCP JSON
Splunk Forwarder (S2S)	Measures the raw byte size for data events. Metric events are counted as a fixed 150 bytes each.	Splunk (Load Balanced), Splunk (S2S)
OpenTelemetry (OTLP/gRPC)	Measures the size of the data after it has been formatted (serialized) into the protobuf structure for gRPC transmission.	Dynatrace (OTLP gRPC), OpenTelemetry (OTLP gRPC)
Streaming and message queues	Measures the total size of the message, including the key, value (payload), and any headers.	Amazon Kinesis Data Streams, Amazon MSK, Amazon SQS, Apache Kafka, Azure Event Hubs, Confluent Cloud
Metrics line senders	Measures the byte size of the formatted metric line at the moment it is prepared for output.	StatsD Extended, StatsD

Destination Category	Byte Calculations	Destinations
Metrics platforms	Measures the size based on the metric components and series metadata, as defined by the specific platform.	New Relic (Metrics), Prometheus, SignalFx
UDP-based outputs	Measures the buffer size multiplied by the number of endpoints (since UDP sends copies to all	NetFlow, SNMP, Syslog (UDP)

8.2.1. Managing Destinations

For each Destination **type**, you can create multiple definitions, each with different settings, to handle delivering data to different outputs with differing requirements.

Once a Destination definition is created, enabled and set up to receive data, you can verify that data is flowing correctly, and monitor the Destination status to check for issues.

Create a Destination

You can create a new Destination in two ways: by using QuickConnect or Routing.

For configuration settings specific to each Destination, check that Destination's documentation.

Create a Destination with QuickConnect

To create a new Destination via QuickConnect:

1. If you are on a Distributed deployment, first select **Worker Groups** in the sidebar and choose a Worker Group.
2. Select **Routing**, then **QuickConnect**.
3. Select **Add Destination** at right, and select the Destination you want from the list.

For configuration settings specific to each Destination, check that Destination's documentation. After you've set up the Destination, you can verify its data flow.

Create a Destination with Routes

To create a new Destination via Routing:

1. If you are on a Distributed deployment, first select **Worker Groups** in the sidebar and choose a Worker Group.
2. Select **Data**, and then **Destinations**.
3. Select the Destination you want.
4. Select **Add Destination** to open a **New Destination** modal.

For configuration settings specific to each Destination, check that Destination's documentation. After you've set up the Destination, you can verify its data flow.

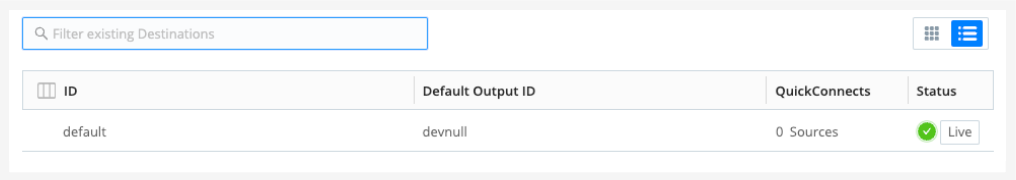
Edit Destination Settings with JSON

To edit any Destination's definition in a JSON text editor, select **Manage as JSON** at the bottom of the Destination modal. You can directly edit multiple values, and you can use the **Import** and **Export** buttons to copy and modify existing Destination configurations as `.json` files.

When JSON configuration contains sensitive information, it is redacted during export.

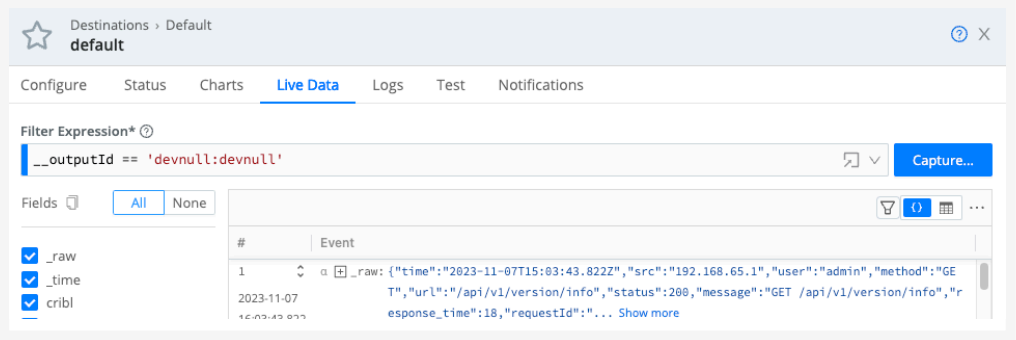
Capture Outgoing Data

To capture data from a single enabled Destination, you can bypass the Data Preview pane and instead capture directly from the list of Destinations. Select the **Live** button beside the Destination you want to capture.



Destination > Live button

You can also start an immediate capture from within an enabled Destination's config modal, by selecting the modal's **Live Data** tab.



Destination modal > Live Data tab

Verify Data Flow

Use the **Test** tab in the Destination's configuration modal to validate the configuration and connectivity of your Destination. A test sends sample events through the Destination to ensure it is correctly set up and can communicate with the downstream receiver.

You can choose from the available sample events or enter your own events, formatted as a JSON array.

Monitor Destination Status

You can get a quick overview of Destination health status by referring to their status icons.

Additionally, each Destination's configuration modal offers two tabs for monitoring: Status and Charts.

Destination Status Icons

Destination status icons are available on the **Data** > **Destinations** page and for each individual Destination in the list for a specific Destination type.

The icons have the following meanings:

Icon	Meaning
	Healthy. Operating correctly.
	Warning. Experiencing issues. The Destination is not functioning fully. Specific conditions will depend on the Destination type.
	Critical. Experiencing critical issues. Drill down to the Destination's Status tab to find out the details.
	Disabled. The Destination is configured, but not enabled.
	No health metrics available. This may mean that a Destination is enabled, but has not been deployed yet.
	Inactive. When using GitOps, a Destination appears `Inactive` if its **Environment** field (configured under **Advanced Settings**) does not match the currently active environment determined by the deployed Git branch. This ensures integrations only activate in their designated environments, preventing unintended data flow or misconfiguration.

You can also find status icons on the Cribl Stream front page on Worker Group tiles. or on any tile indicate that one or more Sources or Destinations in this Worker Group are experiencing issues. Hover over the icon to see more details.

Destination Status Tab

The **Status** tab provides details about the Workers in the group and their status. An icon shows whether the Worker is operating normally.

You can select each Worker's row to see specific information, for example, to identify issues when the Destination displays an error. The specific set of information provided depends on the Destination type. The data represents only process 0 for each Worker Node.

The content of the **Status** tab is loaded live when you open it and only displayed when all the data is ready. With a lot of busy Workers in a group, or Workers located far from the Leader, there may be a delay before you see any information.

The statistics presented are reset when the Worker restarts.

Destination Charts Tab

The **Charts** tab presents a visual representation of Destination activity using internal metrics. Unlike the status tab, this data is read almost instantly and does not reset when restarting a Worker.

At the top of the modal, a time range picker controls which time window the charts display:

- The picker offers the same presets as the main Monitoring time selector (for example, `5 min`, `15 min`, `1 hour`, `6 hours`, `1 day`).
- When you open the Destination I/O metrics modal from **Monitoring**, the charts inherit the current Monitoring time range.
- When you open the modal from other areas of the UI (for example from **Data** > **Destinations** or directly from a Destination configuration modal), the Charts tab defaults to `10 minutes`, matching the previous behavior.
- Changing the time range in the modal only affects that modal and is not persistent. It does not change the global time range on the originating page (for example, the **Monitoring** overview or the page-level time picker).

Use this picker to zoom in on short spikes or expand to longer windows when you're troubleshooting Destination performance or throughput issues.

Data	Metric
Total events out	`total.out_events`
Average throughput (events per second)	Derived from `total.out_events`
Total events dropped	`total.dropped_events`
Total bytes out	`total.out_bytes`
Average throughput (bytes per second)	Derived from `total.out_bytes`

Data	Metric
Backpressure	`backpressure.outputs` and `blocked.outputs`

The following charts are available on Cribl.Cloud deployments utilizing the `iometrics` integration metrics, requiring a metrics level.

Data	Metric	Level Required
Endpoint health	Averages the percentage of healthy endpoints from `endpoints_healthy_percentage`.	`Minimal`
P95 latency	`p95_duration_millis` - 95th percentile request latency in milliseconds.	`Detailed`
P99 latency	`p99_duration_millis` - 99th percentile request latency in milliseconds.	`Detailed`
Failure rate	Percentage of failed requests, calculated as `(failed_requests / total_requests) * 100`.	`Basic`

8.2.2. Amazon

You can send data from Cribl Stream to the following Amazon services:

Destination	Description
Amazon CloudWatch Logs	Send data to Amazon CloudWatch Logs
Amazon Kinesis Data Streams	Deliver data to an Amazon Kinesis Data Stream
Amazon S3 Compatible Stores	Send data to Amazon S3 or to an S3-compatible store
Amazon SQS	Send events to Amazon Simple Queuing Service

8.2.2.1. Amazon CloudWatch Logs Destination

Cribl Stream supports sending data to Amazon CloudWatch Logs. Cribl Stream does **not** have to run on AWS in order to deliver data to CloudWatch Logs.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

IAM Permissions

Ensure your AWS IAM user or role has the following permissions:

- `logs:CreateLogGroup` - Allows the creation of new log groups in CloudWatch Logs.
- `logs:CreateLogStream` - Permits the creation of new log streams within a log group.
- `logs:PutLogEvents` - Enables the insertion of log events into a log stream.

Configure Cribl Stream to Output to Amazon CloudWatch Logs

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this CloudWatch definition.
    - **Description**: Optionally, enter a description.
    - **Log group name**: CloudWatch log group to associate events with.
    - **Log stream prefix**: Prefix for CloudWatch log stream name. This prefix will be used to generate a unique log stream name per Cribl Stream instance. (For example, `myStream_myHost_myOutputId`.)
    - **Region**: AWS Region where the CloudWatch Logs group is located.
3. Next, you can configure the following **Optional Settings**:

- **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`. When this field is set to `queue`, see Persistent Queue Settings for details.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, configure any Authentication, Assume Role, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.

Mode	Description
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Authentication

Use the **Authentication method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS

VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for CloudWatch Logs: Toggle on to use Assume Role credentials to access CloudWatch Logs.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.

- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Endpoint: CloudWatch Logs service endpoint. If empty, defaults to AWS' Region-specific endpoint. Otherwise, use this field to point to a CloudWatchLogs-compatible endpoint.

To proxy outbound HTTP/S requests, see System Proxy Configuration.

Queue size limit: Maximum number of queued batches before blocking. Defaults to `5`.

Record size limit (KB, uncompressed): Maximum size of each individual record before compression. For non-compressible data, 1MB (the default) is the maximum recommended size.

Flush period (sec): Maximum time between requests. Low settings could cause the payload size to be smaller than its configured maximum.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

How the Amazon CloudWatch Logs Destination Batches Events

Amazon CloudWatch Logs specifies that a batch of log events in a single request cannot span more than 24 hours.

When the Amazon CloudWatch Logs Destination transmits a batch of events, it notes the most recent event in the batch (by date-time) and drops all events in the batch that are 24 hours older (or more) than the most recent event.

To monitor dropped events, check **Out of Range Dropped Event Count** under **Status**. Alternately, you can monitor a debug log action to flag how many events are being dropped in a batch.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate,

8.2.2.2. Amazon Kinesis Data Streams Destination

Cribl Stream can output events to **Amazon Kinesis Data Streams** records of up to 1MB uncompressed. Cribl Stream does **not** have to run on AWS in order to deliver data to a Kinesis Data Stream.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Meeting the Needs of Different Downstream Receivers

You can use this Destination to send data to different kinds of downstream receivers that have different expectations. In all cases what the Destination sends out are Kinesis Records. This works because the Amazon Kinesis Data Streams Service is flexible about what is actually in a Record, as the two following use cases show.

Sending Multiple Events within One Record

When used to feed data into Amazon Data Firehose, this Destination will start with multiple events as NDJSON, gzip-compress them together into a blob, and send that as a single Kinesis Record. This uses the Kinesis PutRecord API call to send the combined events in a single Record.

Amazon Data Firehose will gunzip-uncompress the Record and split the NDJSON out into individual events again. There are also other downstream receivers besides Amazon Data Firehose that can do the same thing.

This approach requires either:

- **Advanced Settings** > **Compression** set to `Gzip`, **or**
- **Advanced Settings** > **Send batched** toggled on.

Meanwhile, if you want Cribl Stream to be the consumer, on the Kinesis Streams Source, set **Record data format** to `Cribl`. See the format details below.

Sending One Event Per Record

In contrast to the first use case, some downstream receivers require a Kinesis Record to contain exactly one Cribl event. Here, this Destination uses the Kinesis PutRecords API call to send multiple events in a single API call, but where each event has a Record to itself. (Note the difference between `PutRecord` in the previous use case and `PutRecords` in this one.)

This approach requires:

- **Advanced Settings** > **Compression** set to `None`, **and**
- **Advanced Settings** > **Send batched** toggled off.

Meanwhile, if you want Cribl Stream to be the consumer, on the Kinesis Streams Source, set **Record data format** to `Newline JSON`. See the format details below.

Amazon Kinesis Data Streams Permissions

The following permissions are needed to write to an Amazon Kinesis Stream:

Shard Verification (Determining Stream Availability):

- `kinesis:ListShards` - Required by default to use the ListShards API. This API offers higher rate limits for shard verification.
- `kinesis:DescribeStream` - Required instead of `kinesis:ListShards` if the **ListShards API** advanced setting is toggled off.

Data Writing (Sending Data to the Stream):

- `kinesis:PutRecord` - Required for Gzip compression or batching (when **Advanced Settings > Compression** is set to `Gzip` or **Advanced Settings > Send batched** is toggled on).
- `kinesis:PutRecords` - Required for non-batched mode (when **Advanced Settings > Send batched** is toggled off).

Configure Cribl Stream to Output to Amazon Kinesis Data Streams

In Cribl Stream, configure Amazon CloudWatch Kinesis Data Streams:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Kinesis definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Stream name**: Enter the name of the Kinesis Data Stream (not ARN) to which to send events.

- **Region**: Select the AWS Region where your Kinesis Data Stream is located. If your Region isn't listed, you can manually enter it. Additionally, you can specify the service endpoint in the **Endpoint** field under Advanced Settings.

3. Next, you can configure the following **Optional Settings**:
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`. When this field is set to `queue`, see Persistent Queue Settings for details.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, configure any Authentication, Assume Role, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10

seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Authentication

Use the **Authentication Method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

For details, see AWS' Actions, resources, and condition keys for Amazon Kinesis Data Streams documentation.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for Kinesis Streams: Toggle on to use Assume Role credentials to access Kinesis Streams.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Endpoint: Kinesis Stream service endpoint. If empty, the endpoint will be automatically constructed from the Region.

To proxy outbound HTTP/S requests, see System Proxy Configuration.

Signature version: Signature version to use for signing Kinesis Stream requests. Defaults to `v4`.

Put request concurrency: Maximum number of ongoing put requests before blocking. Defaults to `5`.

Record size limit (KB, uncompressed): Maximum size of each individual record before compression. For non-compressible data, 1MB (the default) is the maximum recommended size.

Flush period (sec): Maximum time between requests. Low settings could cause the payload size to be smaller than its configured maximum.

Compression: Type of compression to use on records being sent downstream. Defaults to `Gzip`, and when set to `None`, displays the following control:

Send batched: Whether to send multiple events batched as a single NDJSON Kinesis record (the default) or send each event as its own Kinesis record, serialized as JSON. When toggled off, displays the following control:

- **Records-per-flush limit**: Maximum number of records to send in a single request to the Amazon Kinesis Data Streams Service PutRecords API endpoint.

ListShards API: The ListShards API is used by default for Kinesis Stream shard verification. This provides significantly higher rate limits (1,000 calls/second/stream), minimizing throttling and improving data delivery speed and reliability, particularly in high-throughput environments.

- Using ListShards requires your IAM policy to have the `kinesis:ListShards` action to grant access. See Controlling Access to Amazon Kinesis Data Streams Resources Using IAM.
- Toggle off to use the DescribeStream API, which has lower rate limits (10 calls/second/account) and increased susceptibility to throttling.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Record Formats in Detail

When sending multiple events within one record, the Record format will be as follows:

- A header line provides information about the payload (the one supported type is shown below).
- Each Kinesis record will contain multiple events serialized as Newline-Delimited JSON (NDJSON).
- Record payloads (including header and body) will be gzip-compressed if **Advanced Settings > Compression** is set to `Gzip`.

Sample Kinesis Record

```
{"format":"ndjson","count":8,"size":3960}
{"_raw":"07-03-2018 18:33:51.136 -0700 ERROR TcpOutputFd - Read error. Connection res
{"_raw":"07-03-2018 18:33:51.136 -0700 INFO  TcpOutputProc - Connection to 127.0.0.1:
...
```

Copy

When sending one event per Record, the Record format will be as follows:

- No header will be included.
- Each Kinesis record will contain a single event serialized as JSON (**not** NDJSON).
- Record payloads will not be compressed.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

8.2.2.3. Amazon S3 Compatible Stores Destination

S3 is a non-streaming Destination type. Cribl Stream does **not** have to run on AWS to deliver data to S3.

Type: Non-Streaming | TLS Support: Yes | PQ Support: No

Stores that are S3-compatible will also work with this Destination type. For example, the S3 Destination can be adapted to send data to Snowflake, for which Cribl Stream currently has no preconfigured Destination.

For guidance in configuring these adaptations, contact Cribl Support. For general guidance, see our Amazon S3 Better Practices guide. To replay data you've previously exported to S3-compatible stores, see our Using S3 Storage and Replay guide and our interactive Data Collection & Replay sandbox.

Configure Cribl Stream to Output to S3 Destinations

In Cribl Stream, configure Amazon S3:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this S3 definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **S3 bucket name**: Name of the destination S3 Bucket. This value can be a constant, or a JavaScript expression that will be evaluated only at init time. For example, referencing a Global Variable: `myBucket-${C.vars.myVar}`.

        Event-level variables are not available for JavaScript expressions. This is because the bucket name is evaluated only at Destination initialization. If you want to use event-level variables in file paths, Cribl recommends specifying them in the **Partitioning Expression** field (described below), because this is evaluated for each file.

- **Staging location**: Filesystem location in which to locally buffer files before compressing and moving to final destination. Cribl recommends that this location be stable and high-performance.

  The **Staging location** field is not displayed or available on Cribl.Cloud-managed Worker Nodes.

- **Key prefix**: Root directory to prepend to path before uploading. Enter either a constant, or a JS expression (enclosed in single quotes, double quotes, or backticks) that will be evaluated only at init time.
- **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (supported only on Linux, not Windows) exposes a **Parquet Settings** left tab, where you **must** configure certain options in order to export data in Parquet format.

3. Next, you can configure the following **Optional Settings**:

   - **Region**: Region where the S3 bucket is located.

     You must grant your IAM role or user the `GetBucketLocation` permission to the S3 bucket when no **Region** is selected. See Amazon S3 Permissions for an example.

   - **Partitioning expression**: JavaScript expression that defines how files are partitioned and organized. Default is date-based. If blank, Cribl Stream will fall back to the event's `__partition` field value (if present); or otherwise to the root directory of the **Output Location** and **Staging Location**.
   - **Compression**: Data compression format used before moving to final destination. Defaults to `gzip` (recommended). This setting is not available when **Data format** is set to `Parquet`.
   - **File name prefix expression**: The output filename prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
   - **File name suffix expression**: The output filename suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.

     To prevent files from being overwritten, Cribl appends a random sequence of 6 characters to the end of their names. This also applies to prefix and suffix expressions in file names.

     For example, if you set the **File name prefix expression** to `CriblExec` and set the **File name suffix expression** to `.csv`, the file name might display as `CriblExec-adPRWM.csv` with `adPRWM` appended.

   - **Backpressure behavior**: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.)

Defaults to `Block`.

- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, configure any Authentication, Assume Role, Processing, Parquet, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Authentication method** drop-down to select one of these options:

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.

- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

The values for **Access key** and **Secret key** can be a constant or a JavaScript expression (such as `${C.env.MY_VAR}`).

Enclose values that contain special characters (like `/`) or environment variables in single quotes or backticks.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for S3: Toggle on to use Assume Role credentials to access S3.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role. This is required only when assuming a role that requires this ID in order to delegate third-party access. For details, see AWS' documentation.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports `c*` wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.

- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.
- In Cribl Edge 4.3 and newer, the S3 Collector supports ingesting data in the Parquet format. Therefore, data that you export in Parquet format can be replayed via the Collector.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Set the target memory size for page segments. Generally, set lower values to improve reading speed, or set higher values to improve compression. Value must be a positive integer smaller than the **Row group size** value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Leave toggled on (the default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Leave toggled on (the default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When this limit is exceeded, on any individual Worker Process, Cribl Stream will close the oldest open files, and move them to the final output location. Defaults to `100`.

Cribl Stream will close files when **any** of the four above conditions is met.

Compression level: Specifies the level of compression to apply before moving files to the final destination. Higher compression levels reduce file size but increase CPU usage. Choose from `Best Speed`, `Normal`, or `Best Compression`. The default level, `Best Speed`, will prioritize faster compression over smaller file sizes.

Staging file limit: Maximum number of files that the Destination will allow to wait for upload before it applies backpressure. Defaults to `100`; minimum is `10`; maximum is `4200`. For context, see Troubleshooting below.

Concurrent file parts limit: Maximum number of parts to upload in parallel per file. A value of `1` tells the Destination to send the whole file at once. When set to `2` or above, IAM permissions must include those required for multipart uploads. Defaults to `4`; highest allowed value is `10`.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined **Min free disk space** amount.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

For a Destination originally configured in a Cribl Stream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this toggle's state. This is to avoid losing any files pending in the original staging directory, upon Cribl Stream upgrade and restart. To enable this option for such Destinations, Cribl's recommended migration path is:

- Clone the Destination.
- Redirect the Routes referencing the original Destination to instead reference the new, cloned Destination.

This way, the original Destination will process pending files (after an idle timeout), and the new, cloned Destination will process newly arriving events with **Add output ID** enabled.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Force close on shutdown: Toggle on to force all staged files to close during an orderly shutdown. This triggers immediate upload or finalization of any in-progress files, regardless of **Idle time**, **File open time**, or **File size** limits. Use this option to minimize data loss risk in ephemeral or auto-scaling environments. This could result in a higher number of small files, as files are closed before reaching their configured thresholds. Ensure your Worker Group or Fleet **Drain timeout** is set high enough to allow all staged data to be finalized before shutdown completes.

Enable dead-lettering: Toggle this on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Endpoint: S3 service endpoint. If empty, Cribl Stream will automatically construct the endpoint from the AWS Region. To access the AWS S3 endpoints, use the path-style URL format. You don't need to specify the bucket

name in the URL, because Cribl Stream will automatically add it to the URL path. For details, see AWS' Path-Style Requests topic.

Object ACL: Object ACL (Access Control List) to assign to uploaded objects.

Storage class: Select a storage class for uploaded objects. Defaults to `Standard`. The other options are: `Reduced Redundancy Storage`; `Standard, Infrequent Access`; `One Zone, Infrequent Access`; `Intelligent Tiering`; `Glacier Flexible Retrieval`; `Glacier Instant Retrieval`; or `Glacier Deep Archive`.

Server-side encryption: Encryption type for uploaded objects - used to enable encryption on data at rest. Defaults to no encryption; the other options are `Amazon S3 Managed Key` or `AWS KMS Managed Key`.

AWS S3 always encrypts data in transit using HTTPS, with default one-way authentication from server to clients. With other S3-compatible stores (such as our native MinIO Destination), use an `https://` URL to invoke in-transit encryption. Two-way authentication is not required to get encryption, and requires clients to possess a certificate.

Signature version: Signature version to use for signing S3 requests. Defaults to `v4`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Verify if bucket exists: Toggle off if you can access files within the bucket, but not the bucket itself. This resolves errors of the form: `error initializing...Bucket does not exist`. For context, see Troubleshooting below.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

IAM Permissions

The following permissions are needed to write to an Amazon S3 bucket:

- `s3:GetBucketLocation` - Needed if **Region** is not configured.
- `s3:ListBucket` - Needed unless **Verify if bucket exists** is toggled off.
- `s3:PutObject`

If your Destination needs to do multipart uploads to S3, two more permissions are needed:

- `kms:GenerateDataKey`

- `kms:Decrypt`

See the AWS documentation.

Temporary Access via SSO Provider

You can use Okta or SAML to provide access to S3 buckets using temporary security credentials.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

> Cribl University offers an Advanced Troubleshooting > Destination Integrations: S3 short course. To follow the direct course link, first log into your Cribl University account. (To create an account, select the **Sign up** link. You'll need to click through a short **Terms & Conditions** presentation, with chill music, before proceeding to courses - but Cribl's training is always free of charge.) Once logged in, check out other useful Advanced Troubleshooting short courses and Troubleshooting Criblets.

Common Issues

“Bucket does not exist - self signed certificate”

Example Error Text:

```
message: Bucket does not exist - self signed certificate
stack: Error: self signed certificate
    at TLSSocket.onConnectSecure (node:_tls_wrap_1535:34)
    at TLSSocket.emit (node:events:513:28)
    at TLSSocket.emit (node:events:489:12)
    at TLSSocket._fini... Show more
tmpPath: /opt/cribl/s3_bucket/cribl/2023/03/28/CriblOut-XTNGh5.0.json.tmp
```

Copy

Can occur when:

- the Destination uses a self-signed cert that has not yet been trusted by Cribl Stream.
- the user does not have appropriate permissions to view the bucket.
- the authentication token generated by your computer has a timestamp that is out of sync with the server’s time, resulting in an authentication failure.

Recommendation

Disable **Advanced Settings** > **Reject unauthorized certificates** or; get the sender’s cert and add it to the `NODE_EXTRA_CA_CERTS` path for validation; or verify that the user has appropriate permissions.

Misconfigured or Nonexistent Buckets

If the S3 bucket that the Destination is trying to send files to does not exist or is misconfigured, the Destination logs will reflect the following pattern:

1. Upon initialization, the Destination will log an error that says the bucket cannot be found. This error will not be logged if **Advanced Settings** > **Verify if bucket exists** is turned off.
2. Every time the Destination “rolls” a new file out of staging to try to upload it to S3, the Destination will log an error, until the number of files rolled exceeds the value of **Advanced Settings** > **Max files to stage**.

Once the Destination has begun to apply backpressure:

- The Destination will log a backpressure warning.
- Events will stop flowing through the Destination.
- The Destination will stop creating staging files.

- Although the Destination will periodically attempt to upload the staged files, it will not log any additional failures.

8.2.2.4. Amazon SQS Destination

Cribl Stream supports sending events to Amazon Simple Queuing Service.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Send Data to Amazon SQS

In Cribl Stream, configure Amazon SQS:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this SQS Destination.
    - **Description**: Optionally, enter a description.
    - **Queue name**: The name, URL, or ARN of the SQS queue to send events to. This value must be a JavaScript expression (which can evaluate to a constant), enclosed in single quotes, double quotes, or backticks. To specify a non-AWS URL, use the format: `'{url}/<queueName>'`. (For example, `':port/<myQueueName>'`.)
    - **Queue type**: The queue type used (or created). Defaults to `Standard`. `FIFO` (First In, First Out) is the other option.
3. Next, you can configure the following **Optional Settings**:
    - **Message group ID**: This parameter applies only to queues of type FIFO. Enter the tag that specifies that a message belongs to a specific message group. (Messages belonging to the same message group are processed in FIFO order.) Defaults to `cribl`. Use event field `__messageGroupId` to override this value.
    - **Create queue**: Specifies whether to create the queue if it does not exist. Defaults to toggled on.
    - **Region**: Region where SQS queue is located.
    - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.)

Defaults to `Block`. When this field is set to `queue`, see Persistent Queue Settings for details.

- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, configure any Authentication, Assume Role, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Authentication

Use the **Authentication method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.

- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for SQS: Toggle on to use Assume Role credentials to access SQS.

AWS account ID: Enter the SQS queue owner's AWS account ID. Leave empty if the SQS queue is in the same AWS account where this Cribl Stream instance is located.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Endpoint: SQS service endpoint. If empty, the endpoint will be automatically constructed from the Region.

Signature version: Signature version to use for signing SQS requests. Defaults to `v4`.

Queue size limit: Maximum number of queued batches before blocking. Defaults to `100`.

Record size limit (KB): Maximum size of each individual record. Per the SQS spec, the maximum allowed value is 256 KB. (the default).

Flush period (sec): Maximum time between requests. Low settings could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Concurrent request limit: The maximum number of in-progress API requests before backpressure is applied. Defaults to `10`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

SQS Permissions

The following permissions are needed to write to an SQS queue:

- `sqs:ListQueues`
- `sqs:SendMessage`
- `sqs:SendMessageBatch`
- `sqs:CreateQueue`
- `sqs:GetQueueAttributes`
- `sqs:SetQueueAttributes`
- `sqs:GetQueueUrl`

Internal Fields

Cribl Stream uses a set of internal fields to assist in handling of data. These "meta" fields are **not** part of an event, but they are accessible, and functions can use them to make processing decisions.

Fields for this Destination:

- `__messageGroupId`
- `__sqsMsgAttrs`
- `__sqsSysAttrs`

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.3. Azure

You can send data from Cribl Stream to the following Azure services;

Destination	Description
Azure Blob Storage	Deliver data to Azure Blog Storage or Azure Data Lake Storage Gen2
Azure Data Explorer	Send batched or streaming data to the Azure Data Explorer managed data analytics service
Azure Event Hubs	Send data to Azure Event Hubs
Azure Monitor Logs	Send data to Azure Monitor Logs
Microsoft Sentinel	Send Log and Metric Events to Microsoft Sentinel SIEM

8.2.3.1. Azure Blob Storage Destination

Cribl Stream supports sending data to (and replaying specific events from) both Azure Blob Storage, and Azure Data Lake Storage Gen2, which implements a hierarchical namespace over blob data. This Destination can deliver data to Azure whether Cribl Stream is running on Azure, another cloud platform, or on-prem.

To use Azure Data Lake Storage Gen2 with this Destination, configure an Azure Storage account with hierarchical namespace enabled and use the standard connection string format. No additional settings are required to target Azure Data Lake Storage Gen2.

Type: Non-Streaming | TLS Support: Yes | PQ Support: No

For configuration examples, see Resources below.

Configure Cribl Stream to Output to Azure Blob Storage

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination definition.
    - **Description**: Optionally, enter a description.
    - **Container name**: Specify the Azure Blob Storage container name. Containers organize blobs, similar to directories in a file system. Container names can include only lowercase letters, numbers, and hyphens.
        - For dynamic container names, provide a JavaScript expression within quotes or backticks, for example, `` `myContainer-${C.env["CRIBL_WORKER_ID"]}` ``. This expression is evaluated at initialization, resolving environment or runtime variables. Only simple concatenations and references to global variables are supported.
    - **Blob prefix**: Root directory to prepend to path before uploading.
    - **Staging location**: Local filesystem location in which to buffer files before compressing and moving them to the final destination. Cribl recommends that this location be stable and high-performance.

Default is `$CRIBL_HOME/state/outputs/staging`.

> In Cribl Stream, the **Staging location** field is not displayed or available on Cribl.Cloud-managed Worker Groups.

- **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (supported only on Linux, not Windows) exposes a **Parquet Settings** left tab, where you **must** configure certain options in order to export data in Parquet format.

3. Under **Authentication**, select an **Authentication method** from the dropdown:
   - **Manual**: Use this default option to enter your Azure Storage connection string directly. Exposes a **Connection string** field for this purpose. (If left blank, Cribl Stream will fall back to `env.AZURE_STORAGE_CONNECTION_STRING`.)
   - **Secret**: This option exposes a **Connection string (text secret)** drop-down, in which you can select a stored secret that references an Azure Storage connection string. A **Create** link is available to store a new, reusable secret. For details, see Connection String Format.
   - **Client secret**: This option allows you to use the Azure service principal's client secret to authenticate, and exposes Azure service principal settings.
   - **Certificate**: This option allows you to use a certificate registered with the Azure service principal to authenticate, and exposes Azure service principal settings.
4. Next, you can configure the following **Optional Settings**:
   - **Create container**: Toggle on to create the configured container in Azure Blob Storage if one does not already exist. Hierarchical namespaces are typically used when working with Azure Data Lake Storage Gen2. When the target storage account has hierarchical namespace enabled, any container created by this Destination will therefore be an ADLS Gen2 container by default.
   - **Partitioning expression**: JavaScript expression that defines how files are partitioned and organized. Default is date-based. If blank, Cribl Stream will fall back to the event's `__partition` field value (if present); or otherwise to the root directory of the **Output Location** and **Staging Location**.
   - **Compression**: Data compression format used before moving to final destination. Defaults to `gzip` (recommended). This setting is not available when **Data format** is set to `Parquet`.
   - **File name prefix expression**: The output file name prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
   - **File name suffix expression**: The output file name suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.

To prevent files from being overwritten, Cribl appends a random sequence of 6 characters to the end of their names. This also applies to prefix and suffix expressions in file names.

For example, if you set the **File name prefix expression** to `CriblExec` and set the **File name suffix expression** to `.csv`, the file name might display as `CriblExec-adPRWM.csv` with `adPRWM` appended.

- **Backpressure behavior**: Whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the Processing, Parquet Settings, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Connection String Format

Manual and **Secret** authentication methods use an Azure Storage connection string in this format:

```
DefaultEndpointsProtocol=[http|https];AccountName=<your-account-name>;AccountKey=<you
```

Copy

A fictitious example, using Microsoft's recommended HTTPS option, is:

```
DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=12345678...32;Enc
```

Copy

For Azure Data Lake Storage Gen2 accounts, use the same connection string pattern as for standard Azure Blob Storage.

Service Principal Settings

Client secret and **Certificate** authentication methods expose the following additional settings:

- **Storage account name**: Enter the name of your Azure Storage Account.
- **Tenant ID**: Enter the service principal's tenant ID.

- **Client ID**: Enter the service principal's client ID.
- **Client secret (text secret)** (with **Client secret** method selected): Select a text secret containing the client secret, or create a new one.
- **Certificate** (with **Certificate** method selected): Select the certificate you registered as credentials for your app in the Azure portal, or create a new one.
- **Azure Cloud**: Select the Azure cloud environment to use, `Azure Public Cloud` (default), `Azure China`, or `Azure Government`. The **Endpoint suffix** advanced setting is available if you need to override the default values.
    - Selecting `Azure China` sets the authority host to `login.chinacloudapi.cn` and the endpoint suffix to `core.chinacloudapi.cn`.
    - Selecting `Azure Government` sets the authority host to `login.microsoftonline.us` and the endpoint suffix to `core.usgovcloudapi.net`.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Set the target memory size for page segments. Generally, set lower values to improve reading speed, or set higher values to improve compression. Value must be a positive integer smaller than the **Row group size** value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Leave toggled on (the default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, and so on.

Write page indexes: Leave toggled on (the default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files reaching this size will be closed and moved to the final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Default: `30`.

Open file limit: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to final output location. Default: `100`.

Cribl Stream will close files when **either** of the `File size limit (MB)` or the `Max file open time (sec)` conditions are met.

Concurrent file parts limit: Maximum number of parts to upload in parallel per file. A value of `1` tells the Destination to send one part at a time - that is, to upload the file's contents sequentially. Defaults to `1`; highest allowed value is `10`.

Blob access tier: Select the access tier for storing your data in Azure Blob Storage. Defaults to `Default account access tier`. Options include:

- `Hot tier` for frequently accessed data.
- `Cool tier` for infrequent access.
- `Cold tier` for rarely accessed data.
- `Archive tier` for data that can tolerate retrieval delays.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined Min free disk space amount.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

For a Destination originally configured in a Cribl Stream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this toggle's state. This is to avoid losing any files pending in the original staging directory, upon Cribl Stream upgrade and restart. To enable this option for such Destinations, Cribl's recommended migration path is:

- Clone the Destination.
- Redirect the Routes referencing the original Destination to instead reference the new, cloned Destination.

This way, the original Destination will process pending files (after an idle timeout), and the new, cloned Destination will process newly arriving events with **Add output ID** enabled.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled,

exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Force close on shutdown: Toggle on to force all staged files to close during an orderly shutdown. This triggers immediate upload or finalization of any in-progress files, regardless of **Idle time**, **File open time**, or **File size** limits. Use this option to minimize data loss risk in ephemeral or auto-scaling environments. This could result in a higher number of small files, as files are closed before reaching their configured thresholds. Ensure your Worker Group or Fleet **Drain timeout** is set high enough to allow all staged data to be finalized before shutdown completes.

Enable dead-lettering: Toggle this on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Endpoint suffix: (Optional) Overrides the default endpoint suffix set by the **Azure Cloud** setting. Use this only if you need to connect to a custom or non-standard Azure Blob Storage endpoint.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Azure Permissions

When configuring an Azure Destination in Cribl Stream, you must grant the necessary permissions for it to interact with your Azure environment. Cribl recommends creating an Azure service principal with the minimum required permissions using Azure role-based access control (Azure RBAC). This approach ensures Cribl Stream has a dedicated identity with precisely the access it needs, following the principle of least privilege.

The minimum permissions required for Destinations are:

- **Storage Blob Data Contributor**

See Permissions for calling data operations in the official Microsoft documentation for more information.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Resources

For examples of configuring Cribl Stream to interoperate with Azure services, see these guides:

- Preparing the Azure Workspace.
- SSO with Microsoft Entra ID (Cribl.Cloud).
- SSO with Microsoft Entra ID (On-Prem).

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate,

8.2.3.2. Azure Data Explorer Destination

Cribl Stream supports sending data to the Azure Data Explorer (ADX) managed data analytics service; you can then run Kusto queries against the data. This Destination can deliver data to Azure whether Cribl Stream is running on Azure, another cloud platform, or on-prem.

ADX stores log data in databases, which in turn contain one or more of the tables defined in the Azure namespace. You must create a separate Cribl Stream ADX Destination for each table that will store your data.

Type: Non-Streaming **or** Streaming (configurable) | TLS Support: Yes | PQ Support: No

For configuration examples, see Resources below.

Prerequisites

Before you configure the Destination, you need to collect some information from your Azure deployment. Start from **Home** (`portal.azure.com`), then navigate to the locations described below and collect the items specified.

Information About Your ADX Cluster

From **Home**, navigate to:

Azure Data Explorer Clusters > `your_Azure_cluster_name` > **Overview** > **Essentials**

- Copy the **URI**. In Cribl Stream, this will be the value for **General Settings** > **Cluster base URI**. Make a copy of the URI with `/.default` appended - this will be the value of **Authentication Settings** > **Scope**.
- Copy the **Data Ingestion URI**. In Cribl Stream, this will be the value for **General Settings** > **Ingestion service URI**.

Information About Your Azure Database

From **Home**, navigate to:

Azure Data Explorer Clusters > `your_Azure_cluster_name` | **Databases** > `database_name` | **Query**

- Note the name of your desired target table - this table name will be the value of **General Settings** > **Table name**.

Information About Your Azure App

First, from **Home**, navigate to:

App registrations > `your_Azure_app_name` | **Overview** | **Essentials**

- Copy the **Application (client) ID** - this will be the value of **Authentication Settings** > **Client ID**.
- Copy the **Directory (tenant) ID** - this will be the value of **Authentication Settings** > **Tenant ID**.

Nex, from **Home**, navigate to:

App registrations > `your_Azure_app_name` | **Certificates & secrets**

Now what you do depends on which method for **Authentication** you plan to use in Cribl Stream.

To use the Client secret method

- In the **Client secrets** tab, click **New client secret** and copy the **Value** of the new secret. In Cribl Stream, this will be the value of **Client secret**.

To use the Client secret (text secret) method

- In the **Client secrets** tab, click **New client secret** and copy the **Value** of the new secret. In Cribl Stream, this will be the value of **Create new secret** > **Value**.

To use the Certificate method

After you have generated a certificate and keys (if you do this later on in the configuration process, return to this section then):

- In the **Certificates** tab, click **Upload Certificate**. In the resulting drawer, follow the prompts to upload your certificate, and click **Add** to close the drawer.

The chosen certificate will then appear in the **Certificates** tab's list.

Data Mapping Prerequisites

Finally, you might need to note or copy data mapping information to obtain values for some of the **General Settings**.

ADX uses a data mapping to map fields from incoming data to fields in the target table. To see the data mapping, you'll run a Kusto query against your target table.

Here's the query syntax:

```
.show table EntityName ingestion MappingKind mapping MappingName
```

Copy

Here's an example query:

```
.show table SyslogTable ingestion json mapping "SyslogMapping"
```

Copy

The Kusto query that originally created this mapping would look like this:

```
.create table SyslogTable ingestion json mapping "SyslogMapping"
```

Copy

In Cribl Stream, you can use the names of mappings like the above example in the **General Settings** > **Data mapping** field. Alternatively, you can create a mapping without naming it, then copy and paste it (as a JSON object) into the Cribl Stream UI. To do this, toggle **General Settings** > **Mapping object** on. This hides the default **Data mapping** text field and opens a text window that expects JSON.

Configure Cribl Stream to Output to Azure Data Explorer

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination definition.
    - **Description**: Optionally, enter a description.
    - **Ingestion mode**: Use the buttons to select **Batching** mode (the default) or **Streaming** mode. For details, see Ingestion Mode.
    - **Cluster base URI**: Enter the base URI for your ADX cluster.
    - **Ingestion service URI**: Displayed only in **Batching** mode. Enter the Ingestion Service URI for your ADX cluster.
    - **Database name**: Enter the name of the ADX database where the target table resides.
    - **Table name**: Enter the name of the target table.

- **Validate database settings**: When you save or start the Destination, validates the database name and credentials that you have entered in these settings. Also validates the table name, except when **Add mapping object** (below) is on. Defaults to toggled on. Toggle off if your Azure app does not have **both** the **Database Viewer** and the **Table Viewer** role.
- **Add mapping object**: Displayed only in **Batching** mode, this control is toggled off by default. Toggle on when you want to paste in a data mapping as a JSON object, instead of using a named data mapping. See Data Mapping Prerequisites above.
- **Data mapping**: If **Add mapping object** is off, enter the name of the desired data mapping. If **Add mapping object** is on, enter the desired data mapping as a JSON object.
- **Compression**: By default, this Destination compresses data using `gzip`; otherwise it sends the data uncompressed. This setting is not available when **Data format** is set to `Parquet`.
- **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (available only in **Batching** mode, and supported only on Linux, not Windows) exposes a **Parquet Settings** tab, where you select the Parquet schema.
- **Backpressure behavior**: Whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`. When **Ingestion mode** is set to **Streaming** mode, the **Persistent Queue** option becomes available. See **Persistent Queue Settings** below.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

3. Optionally, you can adjust the Authentication, Persistent Queue , Processing, Parquet, and Advanced settings outlined in the sections below.
4. Select **Save**, then **Commit & Deploy**.

Ingestion Mode

This Destination supports two ingestion modes:

- **Batching** (the default) stages data in a storage container from which ADX pulls batches of data, then writes them to the target ADX table. Batching works well when the Destination needs to ingest large amounts of data in short periods of time.
- **Streaming** sends data as HTTP request payloads, directly to the target ADX table, without staging them in a container first. Streaming can achieve lower latency than batching, as long as the data that the Destination ingests arrives in relatively small amounts. When in this mode, the Retries section (left tab) becomes available.

Authentication Settings

Microsoft Entra ID authentication endpoint: From the drop-down, select the Microsoft Entra ID endpoint that provides authentication tokens. To understand the available choices, see the Microsoft Entra documentation.

Tenant ID: Directory ID (tenant identifier) in Microsoft Entra ID.

Client ID: `client_id` to pass in the OAuth request parameter.

Scope: Scope to pass in the OAuth request parameter.

Use the **Authentication method** buttons to select one of these options:

- **Client secret**: Use this default option to enter the client secret that you generated for your app in the Azure portal.
- **Client secret (text secret)**: Use this option to select or create a stored text secret.
- **Certificate**: Use this option to select the certificate whose public key you register (or will register) for your app in the Azure portal. Or, create a new one.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when **Ingestion mode** is set to **Streaming** and the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10

seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: The target memory size for page segments. Generally, lower values improve reading speed, while higher values improve compression. Must be a positive integer smaller than the **Row group size** value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Leave toggled on (the default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Leave toggled on (the default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Retries

This tab is displayed when **General Settings** > **Ingestion mode** is set to **Streaming** mode.

Honor Retry-After header: Whether to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream limits the delay to 180 seconds even if the `Retry-After`

header specifies a longer delay. When enabled, any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. When disabled, all `Retry-After` headers are ignored.

Settings for failed HTTP requests: When you want to automatically retry requests that receive particular HTTP response status codes, use these settings to list those response codes.

For any HTTP response status codes that are not explicitly configured for retries, Cribl Stream applies the following rules:

Status Code	Action
`408, 429, 503, 504, 520.`	Retry the request.
Any other status code.	Drop the request.

Upon receiving a response code that's on the list, Cribl Stream first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream continues retrying the request without increasing the time between retries any further.

By default, `429 (Too Many Requests)` is the only response code configured for automatic retries. For each response code you want to add to the list, click **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: When you want to automatically retry requests that have timed out, toggle this control on to display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3

minutes).

Advanced Settings

The controls on this tab vary depending on the **Ingestion mode** selected on the **General Settings** tab. Below you'll find one dedicated section for the Batching, and another for the Streaming controls.

Only the **Environment** control, at the bottom of the tab, is common to both:

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Advanced Settings for Batching Mode

Flush immediately: Toggle on to bypass the data management service's aggregation mechanism. See the ADX documentation.

Retain blob on success: By default, Cribl Stream deletes each data blob once ingestion is complete. Toggle this on if you prefer to retain blobs instead.

Extent tags: Optionally, add strings or tags to partitions of the target table - Azure calls these extents or data shards.

Enforce uniqueness via tag values: Use the **Add value** button to specify a list of `ingest-by` values. Cribl Stream will then check the `ingest-by` tags of incoming extents and discard those whose values match a listed value. This mechanism for avoiding ingesting duplicate extents uses the `ingestIfNotExists` property, as described in the ADX documentation.

Report level: Level for ingestion status reporting. Options are `DoNotReport`, `FailuresOnly` (the default), and `FailuresAndSuccesses`.

Report method: Target for ingestion status reporting. Options are `Queue` (the default), `Table`, and `QueueAndTable`.

Additional fields: Optionally, enter additional configuration properties to send to the ingestion service.

Staging location: Local filesystem location in which to buffer files before compressing and moving them to the final destination. Cribl recommends that this location be stable and high-performance. Defaults to `/tmp`.

File name suffix expression: The output filename suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks.

Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.

To prevent files from being overwritten, Cribl appends a random sequence of 6 characters to the end of their names. This also applies to prefix and suffix expressions in file names.

For example, if you set the **File name prefix expression** to `CriblExec` and set the **File name suffix expression** to `.csv`, the file name might display as `CriblExec-adPRWM.csv` with `adPRWM` appended.

File size limit (MB): Maximum uncompressed output file size. Files reaching this size will be closed and moved to the storage container. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to the storage container. Defaults to `300` seconds (5 minutes).

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to the storage container. Default: `30`.

Open file limit: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to the storage container. Default: `100`.

Concurrent file parts limit: Maximum number of parts to upload in parallel per file. A value of `1` tells the Destination to send one part at a time - that is, to upload the file's contents sequentially. Defaults to `1`; highest allowed value is `10`.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined Min free disk space amount.

Add output ID: Toggle on if you want Cribl Stream to append the Destination name to staging directory pathnames. This can make it easier to organize and troubleshoot when you have multiple Destinations populating staging directories.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Enable dead-lettering: Toggle this on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Advanced Settings for Streaming Mode

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low settings could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Validate server certs: When toggled on (default) Cribl Stream will reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (such as the system's CA, for example).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle this off if you want Cribl Stream to close the connection immediately after sending a request.

Azure Permissions

When configuring an Azure Destination in Cribl Stream, you must grant the necessary permissions for it to interact with your Azure environment. Cribl recommends creating an Azure service principal with the minimum required permissions using Azure role-based access control (Azure RBAC). This approach ensures Cribl Stream has a dedicated identity with precisely the access it needs, following the principle of least privilege.

The minimum permissions required for Destinations are:

- **Database Ingestor:** For the database being written to.
- **Database Viewer:** Required if **Validate database settings** is enabled.

OR

- **Table Ingestor:** For the table being written to.
- **Database Viewer:** Required if **Validate database settings** is enabled.

See Manage Azure Data Explorer cluster permissions and Manage Azure Data Explorer database permissions in the Azure portal in the official Microsoft documentation for more information.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Resources

For examples of configuring Cribl Stream to interoperate with Azure services, see these guides:

- Preparing the Azure Workspace.
- SSO with Microsoft Entra ID (Cribl.Cloud).
- SSO with Microsoft Entra ID (On-Prem).

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly

8.2.3.3. Azure Event Hubs Destination

Cribl Stream supports sending data to Azure Event Hubs.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Azure Event Hubs uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must send events directly to receivers. You might need to adjust your firewall rules to allow this traffic.

For configuration examples, see Resources below.

See Microsoft's Compare Azure Event Hubs Tiers topic to configure tiers whose features and quotas match your desired data volume and throughput.

Configuring Cribl Stream to Output to Azure Event Hubs

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Azure Event Hubs definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Brokers**: List of Event Hub Kafka brokers to connect to. (For example, `yourdomain.servicebus.windows.net:9093`.) Find the hostname in Shared Access Policies, in the host portion of the primary or secondary connection string. If omitted, the port will default to `9093`.
    - **Event Hub name**: The name of the Event Hub (a.k.a., Kafka Topic) on which to publish events. Can be overwritten using the `__topicOut` field.
3. Next, you can configure the following **Optional Settings**:
    - **Acknowledgments**: Control the number of required acknowledgments. Defaults to `Leader`. Setting this control to `Leader` or `All` may significantly slow throughput. If the impact is too great

for your use case, set this control to `None`.

- **Record data format**: Format to use to serialize events before writing to the Event Hub Kafka brokers. Defaults to `JSON`.
  - `JSON` serializes the entire event from your Pipeline, including all metadata and enriched fields, into a single JSON object.
  - `Field _raw` sends only the contents of the `_raw` field as the payload. This option provides maximum throughput and minimal overhead.
- **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the TLS, Authentication, Persistent Queue, Processing, Parquet, Retries and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Client Side)

Enabled: Defaults to toggled on.

Validate server certs: Defaults to toggled on.

Authentication

Authentication parameters to use when connecting to brokers. Using TLS is highly recommended.

Enabled: If toggled on (default), this section's remaining settings are displayed, and all are required settings.

SASL mechanism: SASL (Simple Authentication and Security Layer) authentication mechanism to use with Kafka brokers. Defaults to `PLAIN`, which exposes Basic Authentication options that rely on Azure Event Hubs connection strings. Select `OAUTHBEARER` to enable OAuth Authentication via a different set of options.

Basic Authentication

Selecting the `PLAIN` SASL mechanism provides the options listed in this section.

Username: The username for authentication. For Event Hubs, this should always be `$ConnectionString`.

Authentication method: Use the buttons to select one of these options:

- **Manual**: Use this default option to enter your Event Hubs connection string's primary or secondary key from the Event Hubs workspace. Exposes a **Password** field for this purpose.

- **Secret**: This option exposes a **Password (text secret)** drop-down, in which you can select a stored secret that references an Event Hubs connection string. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need a new secret.

Connection String Format

Either authentication method above uses an Azure Event Hubs connection string in this format:

```
Endpoint=sb://<FQDN>/;SharedAccessKeyName=<your-shared-access-key-name>;SharedAccessKey=<your-shared-access-key-value>
```

A fictitious example is:

```
Endpoint=sb://dummynamespace.servicebus.windows.net/;SharedAccessKeyName=DummyAccessKeyN
```

OAuth Authentication

Selecting the `OAUTHBEARER` SASL mechanism provides the options listed in this section.

Microsoft Entra ID authentication endpoint: Specifies the Microsoft Entra ID endpoint from which to acquire authentication tokens. Defaults to `https://login.microsoftonline.com`. You can instead select `https://login.microsoftonline.us` or `https://login.partner.microsoftonline.cn`.

Client ID: Enter the `client_id` to pass in the OAuth request parameter.

Tenant identifier: Enter your Microsoft Entra ID subscription's directory ID (tenant ID).

Scope: Enter the scope to pass in the OAuth request parameter. This will be of the form: `https://<Event-Hubs-Namespace-Host-name>/.default`. (For example, for an Event Hubs Namespace > Host name: `goatyoga.servicebus.windows.net`, **Scope**: `https://goatyoga.servicebus.windows.net/.default`.)

Authentication method: Use the buttons to select one of these options:

- **Manual**: This default option exposes a **Client secret** field, in which to directly enter the `client_secret` to pass in the OAuth request parameter.
- **Secret**: Exposes a **Client secret (text secret)** drop-down, in which you can select a stored secret that references the `client_secret`. A **Create** link is available to define a new secret.
- **Certificate**: Exposes a **Certificate name** drop-down, in which you can select a stored certificate. A **Create** link is available to define a new cert.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination

has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high

as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Record size limit (KB, uncompressed): Maximum size (KB) of each record batch before compression. Setting should be < `message.max.bytes` settings in Kafka brokers. Defaults to `768`.

Events-per-batch limit: Maximum number of events in a batch before forcing a flush. Defaults to `1000`.

Flush period (sec): Maximum time between requests. Low settings could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Connection timeout (ms): Maximum time to wait for a successful connection. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour).

Request timeout (ms): Maximum time to wait for a successful request. Defaults to `60000` ms (1 minute).

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Destination from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Destination to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Working with Event Timestamps

By default, when an incoming event reaches this Destination, the underlying Kafka JS library adds a `timestamp` field set to the current time at that moment. The library gets the current time from the `Date.now()` JavaScript method. Events that this Destination sends to downstream Kafka receivers include this `timestamp` field.

Some production situations require the `timestamp` value to be the time an incoming event was originally created by an upstream sender, not the current time of its arrival at this Destination. For example, suppose the events were originally created over a wide time range, and arrive at the Destination hours or days later. In this case, the original creation time might be important to retain in events Cribl Stream sends to downstream Kafka receivers.

Here is how to satisfy such a requirement:

1. In a Pipeline that routes events to this Destination, use an Eval Function to add a field named `__kafkaTime` and write the incoming event's original creation time into that field. The value of

`__kafkaTime` **must** be in Unix time (either seconds or milliseconds - both will work). For context, see this explanation of the related fields `_time` and `__origTime`.

2. This Destination will recognize the `__kafkaTime` field and write its value into the `timestamp` field. (This is similar to the way you can use the `__topicOut` field to overwrite a topic setting, as described above.)

If the `__kafkaTime` field is **not** present, the Destination will apply the default behavior (using `Date.now()`) described previously.

Azure Permissions

When configuring an Azure Destination in Cribl Stream, you must grant the necessary permissions for it to interact with your Azure environment. Cribl recommends creating an Azure service principal with the minimum required permissions using Azure role-based access control (Azure RBAC). This approach ensures Cribl Stream has a dedicated identity with precisely the access it needs, following the principle of least privilege.

The minimum permissions required for Destinations are:

- **Azure Event Hubs Data Sender**

See Authorize access to Azure Event Hubs resources using Microsoft Entra ID in the official Microsoft documentation for more information.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__topicOut`
- `__key`
- `__headers`
- `__keySchemaIdOut`
- `__valueSchemaIdOut`

Resources

For examples of configuring Cribl Stream to interoperate with Azure services, see these guides:

- Preparing the Azure Workspace.
- SSO with Microsoft Entra ID (Cribl.Cloud).
- SSO with Microsoft Entra ID (On-Prem).

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Cribl University offers an Advanced Troubleshooting > Destination Integrations: Azure Event Hubs short course. To follow the direct course link, first log into your Cribl University account. (To create an account, select the **Sign up** link. You'll need to click through a short **Terms & Conditions** presentation, with chill music, before proceeding to courses - but Cribl's training is always free of charge.) Once logged in, check out other useful Advanced Troubleshooting short courses and Troubleshooting Criblets.

8.2.3.4. Azure Monitor Logs Destination

The Azure Monitor Log Destination is now deprecated. The replacement for routing data to the Microsoft Log Analytics platform is the Microsoft Sentinel Destination. This change aligns with the Microsoft recommendation to use the Azure Monitor Logs Ingestion API, which the Microsoft Sentinel Destination is built upon.

Microsoft is retiring the legacy HTTP Data Collector API on September 14, 2026. For details, see the Microsoft deprecation notice: Send log data to Azure Monitor by using the HTTP Data Collector API (deprecated).

Cribl Stream supports sending data to Azure Monitor Logs.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

For configuration examples, see Resources below.

Configure Cribl Stream to Output to Azure Monitor Logs

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Azure Monitor Logs definition.
    - **Description**: Optionally, enter a description.
    - **Log type**: The Record Type of events sent to this LogAnalytics workspace. Defaults to `Cribl`. Use only letters, numbers, and `_` characters. (See Microsoft's Azure Monitor documentation.) Can be overwritten by an event's `__logType` field.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Displays fields in which to enter your Azure Log Analytics **Workspace ID** and your Primary or Secondary Shared **Workspace key**. See the Azure Monitor documentation.

- **Secret**: This option exposes a **Secret key pair** drop-down, in which you can select a stored secret that references the credentials described above. A **Create** link is available to store a new, reusable secret.

4. Next, you can configure the following **Optional Settings**:
   - **DNS name of API endpoint**: Enter the DNS name of the Log API endpoint that sends log data to a Log Analytics workspace in Azure Monitor. Defaults to: `.ods.opinsights.azure.com.` Cribl Stream will add a prefix and suffix around this DNS name, to construct a URI in this format: `https://<Workspace_ID><your_DNS_name>/api/logs?api-version=<API version>`.
   - **Resource ID**: Resource ID of the Azure resource to associate the data with. This populates the `_ResourceId` property, and allows the data to be included in resource-centric queries. (Optional, but if this field is not specified, the data will not be included in resource-centric queries.)
   - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Persistent Queue, Processing, Retries and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a

> hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

> If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new

requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `1024` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low settings could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Azure Monitor Limitations

The Azure Monitor Logs architecture limits the number of columns per table, characters per column name, and other parameters. For details, see Microsoft's Azure Monitor Service Limits topic.

Azure will drop logs if your data exceeds these limits. To diagnose this, you can search in the Azure Data Explorer console with a query like this:

```
Operation | summarize count() by Detail
```

...for error messages of this form:

```
Data of type <type> was dropped: The number of custom fields <number> is above the limit of 500 fields per data type.
```

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new one will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If keepalives are not supported by the server (or if the server closes a pooled connection while idle), a new connection will be established for the next request.
- When resolving the Destination's hostname, Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between destination cluster nodes.

Azure Permissions

Azure Monitor Logs does not require any specific permissions.

Resources

For examples of configuring Cribl Stream to interoperate with Azure services, see these guides:

- Preparing the Azure Workspace.
- SSO with Microsoft Entra ID (Cribl.Cloud).
- SSO with Microsoft Entra ID (On-Prem).

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate,

8.2.3.5. Microsoft Sentinel Destination

Cribl Stream can send log and metric events to the Microsoft Sentinel SIEM. This Destination encrypts and sends events via the HTTPS protocol. The supported environments include:

- Azure Public Cloud
- Azure US Government Cloud
- Microsoft Azure operated by 21Vianet

Type: Streaming | TLS Support: No | PQ Support: Yes

For a walk-through of configuring this Destination with Microsoft Sentinel, see our Microsoft Sentinel SIEM Integration guide. Also see these Packs on the Cribl Packs Dispensary, which provide processing Pipelines that you can import and adapt to your needs:

- Microsoft Sentinel Pack.
- Microsoft Sentinel Syslog Pack.
- Microsoft Sentinel Common Security Log Pack.

Ensure that the field names in your data match the schema defined in the Data Collection Rule (DCR) for Microsoft Sentinel. The DCR specifies the expected structure, including field names and types. Mismatched field names can cause dropped events. This Destination sends out individual top-level fields, not a combined `_raw` (or similar) field.

Prerequisites

Before configuring your Microsoft Sentinel Destination(s), you have to prepare the Azure workspace, create data collection rules (DCRs), and obtain the ingestion URL that will receive the data from your Destination(s).

Complete these preparatory steps, which are described in Microsoft Sentinel Integration, before configuring the Microsoft Sentinel Destination. That topic explains the overall Sentinel/Cribl workflow, provides necessary context, and shows you how to obtain the values you'll need to configure the Destination.

If you intend to send multiple event types, or tables, to Sentinel, you'll need a separate Destination for each event type. There are three approaches to configure this:

- **Multiple Destinations**: Configure separate Microsoft Sentinel Destinations, each with its own URL, and use Routes to direct data to the appropriate Destination.

- **Multiple Destinations and a Output Router**: Use an Output Router to split events from a single Source into multiple Destinations, each corresponding to a specific table.
- **Single Destination and overwriting the URL in a Pipeline**: Configure a single Microsoft Sentinel Destination and dynamically overwrite the URL within Pipelines using the `__url` internal field to direct events to the correct table.

Behind the scenes, this Destination uses the Azure Monitor Logs Ingestion API.

Configure Cribl Stream to Output to Microsoft Sentinel

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this HTTP output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Endpoint configuration**: Use the buttons to select **URL** or **ID**. This specifies the method for configuring the endpoint. For details, see Endpoint Configuration Options
3. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Authentication Settings, Processing, Retries and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Endpoint Configuration Options

For Syslog, Cribl recommends starting from the Syslog DCR template in the Cribl-Microsoft GitHub repository. This template defines a stream name of `Custom-Syslog` and routes data into the Sentinel Syslog table.

After you create a data collection rule (DCR) from this template in Azure, you will use its values when configuring the Microsoft Sentinel Destination.

Note that the Syslog template is provided under the `DataCollectionRules` path and is intended for the **DCR kind:direct** approach, matching the corresponding method described in preparing the Azure workspace for Cribl integrations. If your deployment instead uses a Data Collection Endpoint (DCE), use the DCE-based template from the same repository and follow the DCE path in the guide.

The **Endpoint configuration** setting offers the following options:

- **URL** - lets you directly enter the data collection endpoint. This method is the simplest way of configuring the endpoint. See Obtaining a URL for more information.
- **ID** - lets you enter individual IDs that Cribl Stream uses to create the URL used as the data collection endpoint.

Selecting `URL` exposes the following field.

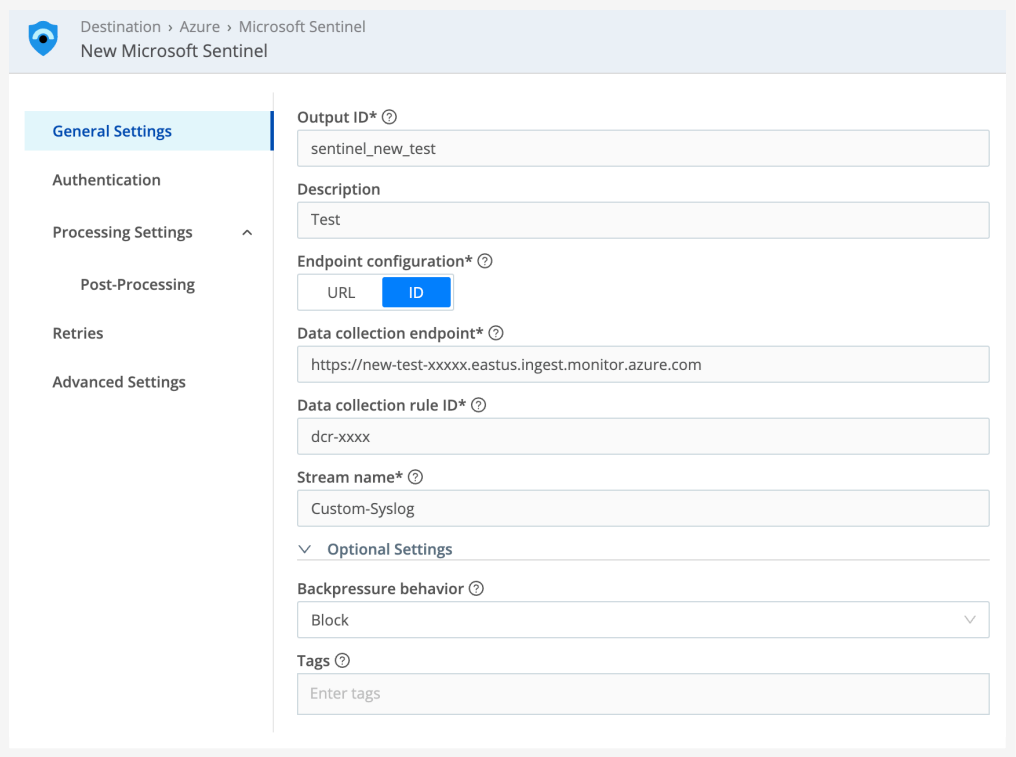
URL: Endpoint URL to send events to. The internal field `__url`, where present in events, will override the `URL` and `ID` values. See Internal Fields below.

Selecting `ID` exposes the following fields.

Data collection endpoint: The Azure `LogsIngestion` URL in the format `https://<endpoint-name>.<identifier>.<region>.ingest.monitor.azure.com`.

Data collection rule ID: Immutable ID for the DCR.

Stream name: Name of the Sentinel table in which to store events. For DCR-based ingestion, this value must exactly match the stream/table name defined in the Azure DCR template, not the friendly table name shown in the Sentinel UI. For the Cribl Syslog template, the correct stream/table name is `Custom-Syslog`. If Stream name does not exactly match the table name in the template, data may fail to land in the expected Sentinel table.



Microsoft Sentinel Destination settings

Optional Settings

Authentication Settings

In Cribl Stream, open your **Microsoft Sentinel Destination** and select **Authentication Settings**. Fill in the fields as follows:

Login URL: Endpoint for the OAuth API call. Use the OAuth2 token endpoint that matches your cloud shown in the Azure **Endpoints** view for your Microsoft Entra app.

Typical formats are:

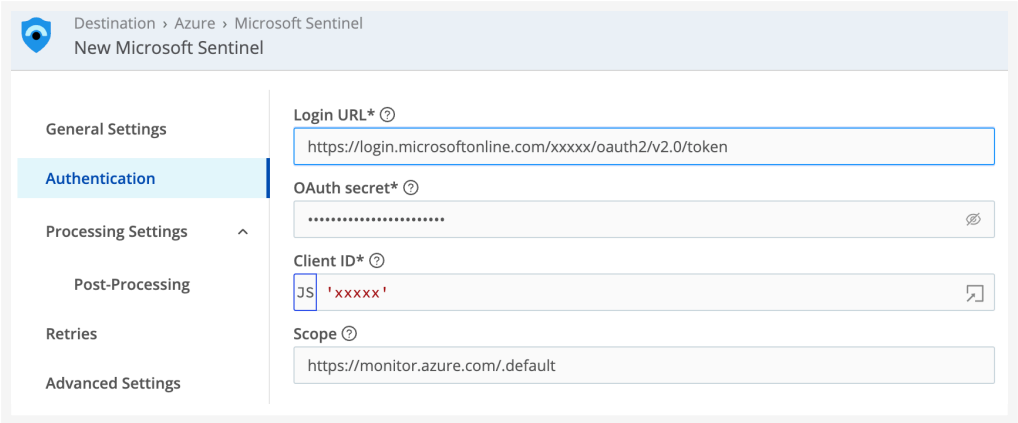
- Azure Public Cloud: `https://login.microsoftonline.com/<tenant-id>/oauth2/v2.0/token`
- Azure US Government Cloud: `https://login.microsoftonline.us/<tenant-id>/oauth2/v2.0/token`
- Microsoft Azure operated by 21Vianet: `https://login.chinacloudapi.cn/<tenant-id>/oauth2/v2.0/token`

OAuth secret: Secret parameter value to pass in the API call's request body. Secret parameter value to pass in the API call's request body. Use the client secret value you created for the app.

Client ID: JavaScript expression used to compute the application (client) ID for the Azure application. Can be a constant.

Scope: The OAuth scope to the appropriate endpoint URL for your Azure environment. This ensures that your OAuth tokens are valid and that your data is correctly ingested into Microsoft Sentinel. This setting should match the specific endpoint for your Azure environment:

- Azure Public Cloud (default): `https://monitor.azure.com/.default`
- Azure US Government Cloud: `https://monitor.azure.us/.default`
- Microsoft Azure operated by 21Vianet: `https://monitor.azure.cn/.default`



Microsoft Sentinel Destination auth settings

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx, 3xx` series	Drop the request
`408, 429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Toggle off if you want Cribl Stream to accept such certificates. Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

By default, Cribl Stream enables gzip-compress to compress the payload body before sending. Toggle this off if you want Cribl Stream not to gzip-compress the payload body.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle this off if you want Cribl Stream to close the connection immediately after sending a request.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `1000` KB, the maximum allowed size of API call. Be aware that high values can cause high memory usage per Worker Node, especially if a dynamically constructed URL (see Internal Fields) causes this Destination to send events to more than one URL. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass to all events as additional HTTP headers. Values will be sent encrypted. You can also add headers dynamically on a per-event basis in the `__headers` field. See Internal Fields below.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__criblMetrics`
- `__url`
- `__headers`

If an event contains the internal field `__criblMetrics`, Cribl Stream will send it to the HTTP endpoint as a metric event. Otherwise, Cribl Stream will send it as a log event.

If an event contains the internal field `__url`, that field's value will override the **General Settings** > **URL** value. This way, you can determine the endpoint URL dynamically.

For example, you could create a Pipeline containing an Eval Function that adds the `__url` field, and connect that Pipeline to your Microsoft Sentinel Destination. Configure the Eval Function to set `__url`'s value to a URL that varies depending on a global variable, or on some property of the event, or on some other dynamically-generated value that meets your needs.

If an event contains the internal field `__headers`, that field's value will be a JSON object containing Name-value pairs, each of which defines a header. By creating Pipelines that set the value of `__headers` according to conditions that you specify, you can add headers dynamically.

For example, you could create a Pipeline containing Eval Functions that add the `__headers` field, and connect that Pipeline to your Microsoft Sentinel Destination. Configure the Eval Functions to set `__headers` values to Name-value pairs that vary depending on some properties of the event, or on dynamically-generated values that meet your needs.

Here's an overview of how to add headers:

- To add "dynamic" (a.k.a. "custom") headers to some events but not others, use the `__headers` field.
- To define headers to add to **all** events, use **Advanced Settings** > **Extra HTTP Headers**.
- An event can include headers added by both methods. So if you use `__headers` to add `{ "api-key": "foo" }` and **Extra HTTP Headers** to add `{ "goat": "Kid A" }`, you'll get both headers.
- Headers added via the `__headers` field take precedence. So if you use `__headers` to add `{ "api-key": "foo" }` and **Extra HTTP Headers** to add `{ "api-key": "bar" }`, you'll get only one header, and that will be `{ "api-key": "foo" }`.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, it will throw away the connection and attempt a new one. This is to prevent sticking to a particular destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), Cribl Stream will establish a new connection for the next request.
- When resolving the Destination's hostname, Cribl Stream will pick the first IP in the list for use in the next connection. Enable **Round-robin DNS** to better balance distribution of events among destination cluster nodes.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

8.2.4. Data Lakes

You can send data from Cribl Stream to the following data lakes:

Destination	Description
Cribl Lake	Send data to Cribl Lake
Amazon S3	Send Cribl Search-formatted data to Amazon S3
Amazon Security Lake	Send OCSF data to Amazon Security Lake in Parquet format

8.2.4.1. Cribl Lake Destination

The **Cribl Lake** Destination delivers data to Cribl Lake and automatically selects a partitioning scheme that works well with Cribl Search.

Type: Non-Streaming | TLS Support: Yes | PQ Support: No

Requirements

The Cribl Lake Destination is available only in Cribl.Cloud.

This Destination can receive data from both Cribl-managed Cloud and customer-managed hybrid Stream Worker Groups. Hybrid Worker Groups must be running Cribl Stream version 4.8 or later.

On hosts for hybrid Worker Groups, this Destination requires outbound HTTP/S access to port 443. This Destination does not allow selecting a different port. For details, see Troubleshoot Hybrid Access to Cribl Lake.

Prepare Data for Use with Lakehouse

Storing data in regular Lake Datasets does not require setting a schema. However, if you want to make full use of Cribl Lakehouse functionality, you need to make sure that events sent from Cribl Stream are parsed into distinct fields before sending to Cribl Lake/Lakehouse.

To do this, use the Stream Parser Function to ensure that all events have named fields.

Configure a Cribl Lake Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect**. Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations**. Select the Destination you want. Next, select **Add Destination**.
2. In the Destination modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Cribl Lake Destination.
    - **Lake dataset**: Select Cribl Lake Dataset to send data to.

You can't target the built-in `cribl_logs` and `cribl_metrics` Datasets with this Destination.

3. Next, you can configure the following **Optional Settings** that you'll find across many Cribl Destinations:
    - **Backpressure behavior**: Whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, customize the settings outlined in Post-Processing below.
5. Click **Save**, then **Commit & Deploy**.
6. Verify that data is searchable in Cribl Lake.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports `c*` wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

How Cribl Stream Handles the `_time` Field

The Cribl Lake Destination requires `_time` to be a number and not `null`. If an incoming `_time` value is `null` or a string, this Destination will convert the value to `Date.now() / 1000`. This creates a time-based partitioning scheme that is readily searchable.

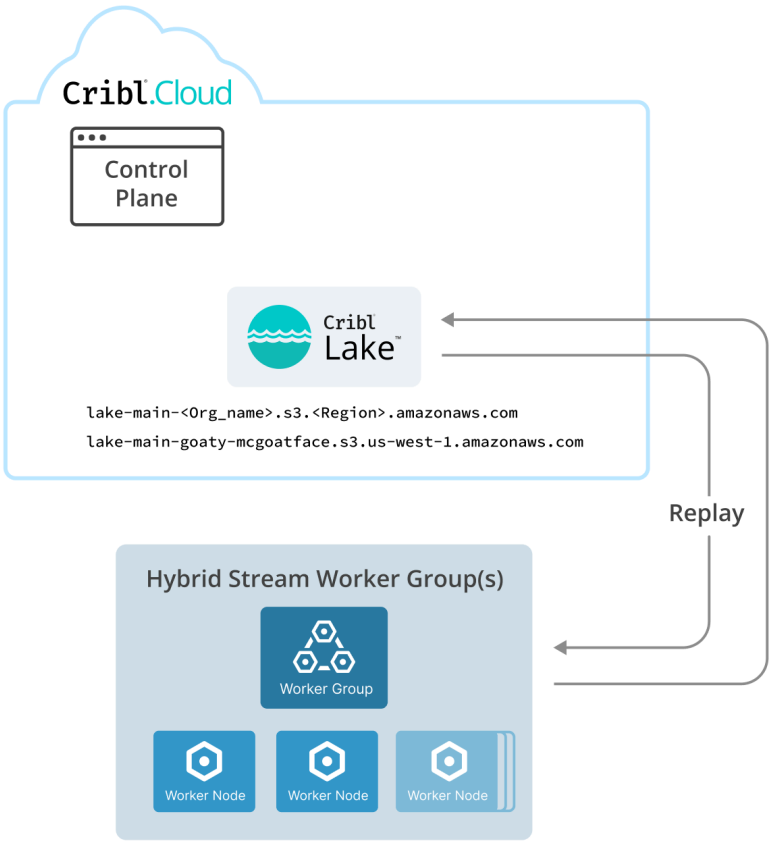
Troubleshoot Cribl Lake Hybrid Access

When running the Cribl Lake Destination on hybrid Worker Groups, you might receive SSL errors, or errors about being unable to connect to a host. This indicates that these Groups have restricted access to the internet through a firewall or proxy that performs SSL inspection. To reach the Cribl.Cloud Organization that hosts your Cribl Lake instance, you can remedy this as follows:

1. At the lower left of your Cribl.Cloud Organization, select **Organization Details**.
2. In the resulting fly-out, note the **Organization ID** and (AWS) **Region**.
3. Swap those two strings for the two placeholders in this fully qualified domain (FDQN) format: `lake-main-<organizationId>.s3.<region>.amazonaws.com`.

   With a fictional **Organization ID**, your FQDN might be: `lake-main-goaty-mcgoatface.s3.us-west-1.amazonaws.com`.

4. Provide the resulting FQDN to your Security team, asking them to add a corresponding exception on the firewall or proxy.
5. Verify that outbound port 443 is open for HTTP/S - this is required to access the FQDN.
6. Test the Cribl Lake Destination: Ensure that the target URL resolves, and that Cribl Stream can successfully send data to the corresponding Cribl Lake Dataset.



Firewall exception for hybrid output to Cribl Lake

8.2.4.2. Amazon S3 Destination

The **Data Lake Amazon S3 Destination** sends data to Amazon Simple Storage Service (**Amazon S3**) and uses a partitioning scheme that is designed to work with Cribl Search. You can organize your bucket structure by event fields for referencing in Cribl Search. See the Cribl Search setup for Data Lake Amazon S3 for details.

Type: Non-Streaming | TLS Support: Yes | PQ Support: No

Configuring Cribl Stream to Output to S3 Destinations

From the top nav, click **Manage**, then select a **Worker Group** to configure. Next, you have two options:

To configure via the graphical QuickConnect UI. Next, click **Add Destination** at right. From the resulting drawer's tiles, select **Data Lakes** > **Amazon S3**. Next, click either **Add Destination** or (if displayed) **Select Existing**. The resulting drawer will provide the options below.

Or, to configure via the Routing UI, click **Data** > **Destinations**. From the resulting page's tiles or the **Destinations** left nav, select **Data Lakes** > **Amazon S3**. Next, click **Add Destination** to open a **New Destination** modal that provides the options below.

General Settings

Output ID: Enter a unique name to identify this S3 definition.

S3 bucket name: Name of the destination S3 Bucket. This value can be a constant, or a JavaScript expression that will be evaluated only at init time. For example, referencing a Global Variable: `myBucket-${C.vars.myVar}`.

Optional Settings

Partition by fields: List of fields to partition the path by, in addition to time (which is included automatically).

The partitioning scheme is designed to work with Cribl Search. This scheme optimizes how objects are searched relative to the search's time range.

In AWS S3, bucket content is returned in ascending alphanumeric order, by default, they are sorted "oldest-first". This Destination uses alpha characters to prefix objects, so ascending sort returns them in "newest-first" order, allowing Search to find files that will likely have the requested objects soonest.

The effective partition will be `<4-letters>-YYYY/<2-letters>-MM/<2-letters>-DD/<2-letters>-HH/<list/of/fields>`. For example, `hjhh-2023/ag-07/ai-25/aj-14/<list/of/fields>`.

In Cribl Search, if you create a corresponding Data Lake Amazon S3 Dataset, the UI will automatically render the path with wildcards and tokens in this format: `<path>/${*}-${_time:%Y}/${*}-${_time:%m}/${*}-${_time:%d}/${*}-${_time:%H}/<list/of/fiel`

Backpressure behavior: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.

Tags: Optionally, add tags that you can use to filter and group Destinations in Cribl Stream's **Manage Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

Authentication

Use the **Authentication method** drop-downs to select one of these options:

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

The values for **Access key** and **Secret key** can be a constant, or a JavaScript expression (such as `${C.env.MY_VAR}`) enclosed in quotes or backticks, which allows configuration with environment variables.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for S3: Toggle on to use Assume Role credentials to access S3.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role. This is required only when assuming a role that requires this ID in order to delegate third-party access. For details, see AWS' documentation.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports `c*` wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.

- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When this limit is exceeded, on any individual Worker Process, Cribl Stream will close the oldest open files, and move them to the final output location. Defaults to `100`.

Cribl Stream will close files when **any** of the four above conditions is met.

Concurrent file parts limit: Maximum number of parts to upload in parallel per file. A value of `1` tells the Destination to send the whole file at once. When set to `2` or above, IAM permissions must include those required for multipart uploads. Defaults to `4`; highest allowed value is `10`.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined **Min free disk space** amount.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

For a Destination originally configured in a Cribl Stream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this toggle's state. This is to avoid losing any files pending in the original staging directory, upon Cribl Stream upgrade and restart. To enable this option for such Destinations, Cribl's recommended migration path is:

- Clone the Destination.
- Redirect the Routes referencing the original Destination to instead reference the new, cloned Destination.

This way, the original Destination will process pending files (after an idle timeout), and the new, cloned Destination will process newly arriving events with **Add output ID** enabled.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Enable dead-lettering: Toggle this on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Endpoint: S3 service endpoint. If empty, Cribl Stream will automatically construct the endpoint from the AWS Region. To access the AWS S3 endpoints, use the path-style URL format. You don't need to specify the bucket name in the URL, because Cribl Stream will automatically add it to the URL path. For details, see AWS' Path-Style Requests topic.

Object ACL: Object ACL (Access Control List) to assign to uploaded objects.

Storage class: Select a storage class for uploaded objects. Defaults to `Standard`. The other options are: `Reduced Redundancy Storage`; `Standard, Infrequent Access`; `One Zone, Infrequent Access`; `Intelligent Tiering`; `Glacier`; or `Deep Archive`.

Server-side encryption: Encryption type for uploaded objects - used to enable encryption on data at rest. Defaults to no encryption; the other options are `Amazon S3 Managed Key` or `AWS KMS Managed Key`.

AWS S3 always encrypts data in transit using HTTPS, with default one-way authentication from server to clients. With other S3-compatible stores (such as our native MinIO Destination), use an `https://` URL to invoke in-transit encryption. Two-way authentication is not required to get encryption, and requires clients to possess a certificate.

Signature version: Signature version to use for signing S3 requests. Defaults to `v4`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

IAM Permissions

The following permissions are always needed to write to an Amazon S3 bucket:

- `s3:ListBucket`
- `s3:GetBucketLocation`
- `s3:PutObject`

If your Destination needs to do multipart uploads to S3, two more permissions are needed:

- `kms:GenerateDataKey`
- `kms:Decrypt`

See the AWS documentation.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

8.2.4.3. Amazon Security Lake Destination

This Destination delivers data to Amazon Security Lake, a fully-managed security data lake service backed by Amazon S3 buckets.

Type: Non-Streaming | TLS Support: Yes | PQ Support: No

This Destination is unavailable on Windows and macOS.

Security Lake ingests events described by the Open Cybersecurity Schema Framework (OCSF) and conveyed as Parquet files. This Destination sends the Parquet files in batches. Cribl Stream does not read or replay the data sent to Security Lake.

A Complete Setup Scenario

Cribl recommends that you start with the Amazon Security Lake Integration topic, which will guide you through the following steps in the correct order:

1. Set up Amazon Security Lake.
2. Set up Cribl Stream.
3. Create an Amazon Security Lake custom source, This is an S3 bucket dedicated to a single OCSF event class.
4. Create a Cribl Amazon Security Lake Destination.
5. Connect a Cribl Source to the Amazon Security Lake Destination.
6. Send data to Amazon Security Lake.

As outlined above, you'll create and configure this Destination as the fourth of the six overall steps. The Integration topic covers the key points about configuring this Destination. Refer back to this reference topic, as needed, for details about any given setting. Some of the instructions below assume that you have recorded IDs, values, etc., from procedures you've already completed according to the Integration topic.

Although Cribl Stream does **not** need to be deployed in Cribl.Cloud in order to deliver data to Amazon Security Lake, setup procedures on Cribl>Cloud will differ from those documented here. To learn more, please contact Cribl via the `#packs` channel of Cribl Community Slack.

Configure Cribl Stream to Output to Amazon Security Lake

In Cribl Stream, configure Amazon Security Lake:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:

    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.

2. In the **New Destination** modal, configure the following under **General Settings**: -**Output ID**: Enter a unique name to identify this Amazon Security Lake Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.

    - **Description**: Optionally, enter a description.
    - **S3 bucket name**: Name of the destination S3 Bucket you created when setting up Amazon Security Lake. This value can be a constant, or a JavaScript expression that will be evaluated only at init time. For example, referencing a Global Variable: `myBucket-${C.vars.myVar}`.

      Event-level variables are not available for JavaScript expressions. This is because the bucket name is evaluated only at Destination initialization.

    - **Region**: Region where the Amazon Security Lake is located.
    - **Account ID**: The ID of the AWS Account whose admin enabled Amazon Security Lake.
    - **Custom source name**: The name of the custom source you created in Amazon Security Lake to receive data from this Destination.
    - **Staging location**: Filesystem location in which to locally buffer files before compressing and moving to final destination. Cribl recommends that this location be stable and high-performance.

      The **Staging location** field is not displayed or available on Cribl.Cloud-managed Worker Nodes.

3. Next, you can configure the following **Optional Settings**:

    - **File name prefix expression**: The output filename prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
    - **Backpressure behavior**: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, configure any Authentication, Assume Role, Processing, Parquet, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

For the **Authentication Method**, you **must** leave the default **Auto** method selected.

This method uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants Cribl Stream Worker Nodes access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.

Assume Role

For these settings, you'll enter values that you noted when creating the custom source in Security Lake.

AssumeRole ARN: The custom source's **Provider role ARN**.

External ID: In the custom source, this is the value of the `ExternalId` element of the trust relationship JSON object.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output. Use the drop-down to select the Pack you installed when setting up Cribl Stream.

System fields: Remove any fields specified here.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

- In Cribl Edge 4.3 and newer, the S3 Collector supports ingesting data in the Parquet format. Therefore, data that you export in Parquet format can be replayed via the Collector.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

Parquet schema: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Ideal memory size for page segments. For example, 1 MB or 128 MB. Generally, lower values improve reading speed, while higher values improve compression. Imposes a target, not a strict limit; the final size of a row group may be larger or smaller. Defaults to 1 MB.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Leave toggled on (the default) if you have Parquet tools configured to view statistics – these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Leave toggled on (the default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When this limit is exceeded, on any individual Worker Node Process, Cribl Stream will close the oldest open files, and move them to the final output location. Defaults to `100`.

Cribl Stream will close files when **any** of the four above conditions is met.

Concurrent file parts limit: Maximum number of parts to upload in parallel per file. A value of `1` tells the Destination to send the whole file at once. When set to `2` or above, IAM permissions must include those required for multipart uploads. Defaults to `4`; highest allowed value is `10`.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Enable dead-lettering: Toggle this on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Endpoint: Security Lake service endpoint. If empty, defaults to AWS' Region-specific endpoint. Otherwise, it must point to Security Lake-compatible endpoint.

Object ACL: Object ACL (Access Control List) to assign to uploaded objects.

Storage class: Select a storage class for uploaded objects. Defaults to `Standard`. The other options are: `Reduced Redundancy Storage`; `Standard, Infrequent Access`; `One Zone, Infrequent Access`; `Intelligent Tiering`; `Glacier`; or `Deep Archive`.

Server-side encryption: Encryption type for uploaded objects - used to enable encryption on data at rest. Defaults to no encryption; the other options are `Amazon S3 Managed Key` or `AWS KMS Managed Key`.

Amazon S3 always encrypts data in transit using HTTPS, with default one-way authentication from server to clients. Two-way authentication is not required to get encryption, and requires clients to possess a certificate.

Signature version: Signature version to use for signing Security Lake requests. Defaults to `v4`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

IAM Permissions

The following permissions are always needed to write to an Amazon S3 bucket:

- `s3:ListBucket`
- `s3:GetBucketLocation`
- `s3:PutObject`

If your Destination needs to do multipart uploads to S3, two more permissions are needed:

- `kms:GenerateDataKey`
- `kms:Decrypt`

See the AWS documentation.

Creating and Modifying Parquet Schemas

When you need to add a new Parquet schema or modify an existing one:

1. Navigate to the Parquet Schema Library, located at **Processing** > **Knowledge** > **Parquet Schemas**.
2. Create or modify the schema, as described in Parquet Schemas.

Cribl strongly recommends that to modify an existing schema, you first clone it and give the clone its own distinct name.

3. Return to this Destination, and select the new or modified schema in the **Parquet schema** drop-down.
4. Click **Save**.

Cloning First is Safer Than Just Modifying

Modifying an existing schema in the Schema Library does **not** propagate your modifications to the Destination's copy of the schema.

- Cloning and renaming schemas is the safest approach, because in Step 3 above, it removes any doubt that your Destination will now use the newly-modified version of your schema.
- If you do **not** clone and rename the schema (i.e., you leave the schema name unchanged), you still must re-select the schema in the Destination's **Parquet schema** drop-down, and click **Save**, to bring the modified version into the Destination.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Troubleshooting Resources

Cribl University offers an Advanced Troubleshooting > Destination Integrations: S3 short course. Since Amazon Security Lake relies on S3 buckets, what you learn from the course can augment your Security Lake skills.
To follow the direct course link, first log into your Cribl University account. (To create an account, select the **Sign up** link.
You'll need to click through a short **Terms & Conditions** presentation, with chill music, before proceeding to courses - but Cribl's training is always free of charge.)
Once logged in, check out other useful Advanced Troubleshooting short courses and Troubleshooting Criblets.

8.2.5. Elastic

You can send data from Cribl Stream to the following Elastic services:

Destination	Description
Elasticsearch	Send events to an Elasticsearch cluster using the Bulk API
Elastic Cloud	Send events to Elastic Cloud

8.2.5.1. Elasticsearch Destination

Cribl Stream can send events to an Elasticsearch cluster using the Bulk API. As of v.3.3, Cribl Stream supports Elastic data streams.

Use the Elastic Cloud Destination if you are exclusively targeting Elastic Cloud and prefer a simpler, optimized configuration process.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

For an example of using Cribl Stream to parse and enrich inbound events for Elasticsearch, see our Splunk to Elasticsearch topic.

Configure Cribl Stream to Output to Elasticsearch

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Elasticsearch Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Load balancing**: Toggle on (default) to specify multiple bulk API URLs and load weights. If toggled off and you notice that Cribl Stream is not sending data to all possible IP addresses, enable **Advanced Settings** > **Round-robin DNS**.
    - **Bulk API URL or Cloud ID**: Specify either an Elasticsearch cluster or Elastic Cloud deployment to send events to. For an Elasticsearch cluster, enter a URL (for example, `http://<myElasticCluster>:9200/_bulk`). For an Elastic Cloud deployment, enter its Cloud ID. This setting is not available when **Load balancing** is enabled.
    - **Index or data stream**: Enter a JavaScript expression that evaluates to the name of the Elastic data stream or Elastic index where you want events to go. The expression is evaluated for each event,

can evaluate to a constant value, and must be enclosed in quotes or backticks. An event's `__index` field can overwrite the index or data stream name.

3. Under **Authentication**, toggle **Authentication enabled** on to display **Authentication method** dropdown. Select one of these options:
    - **Manual**: Enter your credentials directly in the resulting **Username** and **Password** fields.
    - **Secret**: Exposes a **Credentials secret** drop-down, in which you can select a stored secret that references the credentials described above. A **Create** link is available to store a new, reusable secret.
    - **Manual API Key**: Exposes an **API key** field to directly enter your Elasticsearch API key.
    - **Secret API Key**: Exposes an **API key (text secret)** drop-down, in which you can select a stored text secret that references your Elasticsearch API key. A **Create** link is available to store a new, reusable secret.
4. Next, you can configure the following **Optional Settings**:
    - **Exclude current host IPs**: This toggle appears when **Load balancing** is toggled on. It determines whether to exclude all IPs of the current host from the list of any resolved hostnames. Default is toggled off, which keeps the current host available for load balancing.
    - **Type**: Specify document type to use for events. An event's `__type` field can overwrite this value.
    - **Backpressure behavior**: Specify whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Bulk API URLs

Use the **Bulk API URLs** table to specify a known set of receivers on which to load-balance data. To specify more receivers on new rows, click **Add URL**. Each row provides the following fields:

URL: Specify the URL to an Elastic node to send events to - for example, `http://elastic:9200/_bulk`

Load weight: Set the relative traffic-handling capability for each connection by assigning a weight (> `0`). This column accepts arbitrary values, but for best results, assign weights in the same order of magnitude to all connections. Cribl Stream will attempt to distribute traffic to the connections according to their relative weights.

The final column provides an `X` button to delete any row from the table.

For details on configuring all these options, see About Load Balancing.

When you first enable load balancing, or if you edit the load weight once your data is load-balanced, give the logic time to settle. The changes might take a few seconds to register.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older

than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request

Status Code	Action
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are **not** authorized by a trusted CA (for example, the system's CA). Default is toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts. (This option is visible only when the **General Settings** > **Load balancing** option is toggled off.)

Compress: Compresses the payload body before sending. Default is toggled on (recommended).

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (s): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Extra parameters: Name-value pairs to pass as additional parameters. If you are using Elastic ingest pipelines, specify an extra parameter whose name is `pipeline` and whose value is the name of your pipeline, similar to these examples.

Elastic version: Determines how to format events. For Elastic Cloud, you must explicitly set version `7.x`. For other Elasticsearch clusters, the `Auto` default will discover the downstream Elasticsearch version automatically, but you have the option to explicitly set version `6.x` or `7.x`.

Elastic pipeline: To send data to an Elastic Ingest pipeline, optionally enter that pipeline's name as a constant. Or, enter a JavaScript expression that evaluates outgoing events and sends matching events to the desired Elastic Ingest pipeline(s). Cribl Stream will first interpret your entry as a constant and try to find a matching value in the event. If it finds no matching value, it will evaluate your **Elastic pipeline** entry as an expression.

For example, the expression `sourcetype=='access_common'?'cribl_pipeline':undefined` matches events whose `sourcetype` is `access_common`, and sends them to an Elastic Ingest pipeline named `cribl_pipeline`.

You can also specify the name of the pipeline in an event field. For example, `myPipelineField` would use the value from the event's `myPipelineField` property (if present) to identify the Elastic Ingest pipeline to process the event. Alternately, the expression `myPipelineField != null ? myPipelineField : undefined` would also identify this field. If the event does not contain such a field, `myPipelineField != null ? myPipelineField : 'theDefaultIndex'` will provide a default index. With this approach, you can override the Elastic Ingest pipeline at the event level.

See also the Elasticsearch Source documentation.

The next two fields appear only when you toggle **Load balancing** on in the **General Settings**.

DNS resolution period (seconds): Re-resolve any hostnames each time this interval recurs, and pick up destinations from the A records. Defaults to `600` seconds.

Load balance stats period (seconds): Lookback traffic history period. Defaults to `300` seconds.

Include document_id: Toggle off to omit the `document_id` field when sending events to an Elastic TSDS (time series data stream).

Write action: Action to use when writing events. Set this option to `Create` (default) when writing to a data stream, which is append-only. Set this option to `Index` only when you write directly to an index and need to update existing records. `Index` will fail if you use it to write to a data stream.

Failed request logging mode: Determines which data is logged when a request fails. Use the drop-down to select one of these options:

- `None` (default).

- `Payload`.
- `Payload + Headers`. Use the **Safe Headers** field below to specify the headers to log. If you leave that field empty, all headers are redacted, even with this setting.

Safe headers: List the headers you want to log, in plain text.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Field Normalization

When sending data to Elasticsearch, this Destination applies the following field normalization to align with the Elasticsearch schema:

- Renames `_time` to `@timestamp`, maintaining millisecond precision.
- Renames `host` to `host.name`.

See also our Elasticsearch Source documentation's **Field Normalization** section.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__id`
- `__type`
- `__index`
- `__host`

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new one will be reattempted. This is to prevent sticking to a particular destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), a new connection will be established for the next request.
- When resolving the Destination's hostname with load balancing disabled, Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between Elasticsearch cluster nodes.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.5.2. Elastic Cloud Destination

Cribl Stream can send events to Elastic Cloud.

Use the Elasticsearch Destination if you need flexibility to support both self-hosted and Elastic Cloud deployments and require advanced configuration options.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

For an example of using Cribl Stream to parse and enrich inbound events for Elasticsearch, see our Splunk to Elasticsearch topic.

Configure Cribl Stream to Output to Elastic Cloud

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Elastic Cloud Destination definition.
    - **Description**: Optionally, enter a description.
    - **Cloud ID**: Enter the Cloud ID of the Elastic Cloud environment where you want to send events. You'll find it on the Deployments overview page of your Elastic Cloud environment.
    - **Data stream or index**: Enter a JavaScript expression that evaluates to the name of the Elastic data stream or Elastic index where you want events to go. The expression is evaluated for each event, can evaluate to a constant value, and must be enclosed in quotes or backticks. An event's `__index` field can overwrite the index or data stream name.
    - **Elastic pipeline**: Enter the name of the Elastic ingest pipeline (optional). You can use an expression to override the pipeline attribute received from the Elasticsearch Source.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Enter your credentials directly in the resulting **Username** and **Password** fields.

- **Secret**: Exposes a **Credentials secret** drop-down, in which you can select a stored secret that references the credentials described above. A **Create** link is available to store a new, reusable secret.
- **Manual API Key**: Exposes an **API key** field to directly enter your Elasticsearch API key.
- **Secret API Key**: Exposes an **API key (text secret)** drop-down, in which you can select a stored text secret that references your Elasticsearch API key. A **Create** link is available to store a new, reusable secret.

4. Next, you can configure the following **Optional Settings**:
   - **Backpressure behavior**: Specify whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Persistent Queue, Processing, Retries and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is

influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (s): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Extra parameters: Name-value pairs to pass as additional parameters. If you are using Elastic ingest pipelines, specify an extra parameter whose name is `pipeline` and whose value is the name of your pipeline, similar to these examples.

Include document_id: Toggle off to omit the `document_id` field when sending events to an Elastic TSDS (time series data stream).

Failed request logging mode: Determines which data is logged when a request fails. Use the drop-down to select one of these options:

- `None` (default).
- `Payload`.
- `Payload + Headers`. Use the **Safe Headers** field below to specify the headers to log. If you leave that field empty, all headers are redacted, even with this setting.

Safe headers: List the headers you want to log, in plain text.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Field Normalization

This Destination normalizes the following fields:

- `_time` becomes `@timestamp` at millisecond resolultion.
- `host.name` is set to `host`.

See also our Elasticsearch Source documentation's **Field Normalization** section.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__id`
- `__type`
- `__index`

- `__host`

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new one will be reattempted. This is to prevent sticking to a particular destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), a new connection will be established for the next request.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.6. Google Cloud

You can send data from Cribl Stream to the following Google Cloud services:

Destination	Description
Google SecOps	Send data to Google Security Operations (SecOps)
Google Cloud Chronicle API	Send to Google Cloud Chronicle API
Google Cloud Logging	Send data to Google Cloud Logging
Google Cloud Pub/Sub	Send data to Google Cloud Pub/Sub
Google Cloud Storage	Send data to Google Cloud Storage

8.2.6.1. Google Security Operations (SecOps) Destination

Cribl Stream supports sending data to Security Operations (SecOps), a cloud service for retaining, analyzing, and searching enterprise security and network telemetry data. This service was formerly known as Google Chronicle.

If you plan to use **V2** of the Google Security Operations API, you need your Google service account credentials to define a Google SecOps Destination.

If you plan to use **V1** of the Google Security Operations API, Cribl recommends using Google service account credentials for authentication, but alternatively, you can opt to use an API key from Google.

To add labels to individual events instead of at the batch level, you can use the Google Cloud Chronicle API Destination.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Options for Sending Data to SecOps

When sending data to SecOps, you have two primary methods, each with its own advantages:

- **Sending Unstructured Data for SecOps Parsing**: This is ideal for well-known data types. You send your raw, unstructured data, and SecOps uses its built-in parsers to transform it into the Unified Data Model (UDM). However, this approach can lead to parsing errors and troubleshooting if you're dealing with less common or proprietary data types.
- **Sending Pre-Structured UDM Data**: This method bypasses SecOps's parsing process entirely. It's especially beneficial for less-usual or unique, proprietary data types that might confuse SecOps's parsers. By sending data that's already structured as UDM, you gain greater control and flexibility over how your data lands in SecOps. The trade-off is that you "own" the data transformation process, using Pipelines and Functions to convert your initial unstructured data to UDM before sending it to SecOps.
- **Sending Pre-Structured UDM Entity Data**: This is an extension of the **Pre-Structured UDM Data** option, specifically for sending entity data to the `v2/entities:batchCreate` endpoint. This data is used to enrich logs and provide additional context on events. Like sending UDM logs, you must format your events to conform to the UDM entity schema before sending them to SecOps.

Do not place UDM fields like `metadata`, `principal`, and `network` inside `_raw`. These fields must exist as top-level fields in the event payload.

Working with Unstructured Log Types

Google SecOps supports a dynamic list of unstructured log types that evolves over time. There are two configuration settings for managing log type assignments:

- **Default log type**: Cribl Stream periodically updates its **Default log type** dropdown with the latest list from Google.
- **Custom log types**: If a log type you need isn't in the default dropdown, or if you're using an older, removed log type, you can define and manage it in the **Custom log types** table under **Optional Settings**.

For Google's latest list of supported log types, see the Google SecOps SIEM reference or send a `GET` request to `https://malachiteingestion-pa.googleapis.com/v1/logtypes`.

How Log Types are Assigned

- **Event-specific**: When processing an event, Cribl Stream first checks if the event has a `__logType` field. If this field is present, its value determines the log type for that event.
- **Default**: If the event lacks a `__logType` field, Cribl Stream assigns the log type specified in the **Default log type** drop-down.
- **Custom**: If the **Default log type** is set to `Custom`, Cribl Stream will use the first custom log type listed in the **Custom log types** table. Reorder the custom log types in the table using the grab handles to prioritize which custom log type should be used as the default.
    - If the **Default log type** is set to a specific log type (not `Custom`), the custom log types defined in the **Custom log types** table will not be used as defaults. However, you can still use these custom log types by explicitly setting the `__logType` field in your events to one of the custom log types.

Configure Cribl Stream to Output to SecOps

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this SecOps output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.

- **API version**: The version of the Google Security Operations Ingestion API you want the Destination to use when communicating with SecOps. Defaults to `V2`.
- **Send events as**: Choose how you want the Destination to send data to SecOps. The options are: **Unstructured** and **UDM**. See Options for Sending Data to SecOps to learn about the differences between working with `Unstructured` and already-structured `UDM` data.

3. Next, configure the settings that appear based on your **API version** and **Send events** as selections. For detail, see the API Version and Output Format Options section.
4. Next, you can configure the following **Optional Settings**: (The following settings are available regardless of how **General Settings** > **Send events as** is set.)
   - **Region**: The Google SecOps regional endpoint to send events to.
   - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Authentication, Persistent Queue, Processing, Retries and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

API Version and Output Format Options

The settings that appear in the **New Destination** modal depend on your selections for **API version** and **Send events as**. The following sections describe how these choices affect the available configuration options.

For Google SecOps unstructured parsers, set the Destination **Log text field** to `_raw`. This ensures that the payload body is the raw log line and that fields like `message_trace_id` are at the payload root.

API version: V2 (Default)

The API version `V2` is the recommended version for most use cases.

- If you choose **Send events as**: `UDM`, an additional **UDM type** option displays. You must select either `Logs` for standard log data or `Entities` for entity-specific data.
- If you choose **Send events as**: `Unstructured`, the following settings appear under **General Settings**:
  - **Default log type**: Select a log type to assign to events that lack a `__logType` field.
  - **Customer ID**: A unique identifier (UUID) corresponding to a particular SecOps instance.

  The following are **Optional Settings**:

- **Custom log types**: For each log type you want to define, select **Add type** and enter values for **Log type** and **Description** in the resulting table row. See Working with Unstructured Log Types above for more details.
- **Namespace**: Configure an environment namespace to identify logs' originating data domain. You can use this as a tag when indexing and/or enriching data. The `__namespace` event field, if present, will overwrite this.
- **Custom labels**: Define labels for the Destination to add to events. These are used by Google SecOps for features such as data RBAC and filtering. See how to configure data RBAC for reference lists. Select **Add label** to create a new label and specify it as a key-value pair.
- **Log text field**: Set to the event field containing the raw log string (commonly `_raw`). If left blank, the Destination sends a JSON representation of the whole event. Use this when you want Google to parse the raw line itself rather than JSON you provide.

API version V1

THe API version `V1` is a legacy version. We recommend using `V2` for all new deployments, but `V1` is available for backward compatibility with existing configurations or for environments that have specific requirements that prevent them from using `V2`.

- If you choose **Send events as**: `UDM`, no additional settings appear, but the **Custom labels** setting is available.
- If you choose **Send events as**: `Unstructured`, the following settings appear under **General Settings**:
  - **Default log type**: Select a log type for events that lack a `__logType` field.

  The following are **Optional Settings**:
  - **Custom log types**: For each log type you want to define, select **Add type** and enter values for **Log type** and **Description** in the resulting table row. See Working with Unstructured Log Types above for more details.
  - **Namespace**: Configure an environment namespace to identify logs' originating data domain. You can use this as a tag when indexing and/or enriching data. The `__namespace` event field, if present, will overwrite this.
  - **Customer ID**: A unique identifier (UUID) corresponding to a particular SecOps instance.
  - **Custom labels**: Define labels for the Destination to add to events. These are used by Google SecOps for features such as data RBAC and filtering. See how to configure data RBAC for reference lists. Select **Add label** to create a new label and specify it as a key-value pair.
  - **Log text field**: Set to the event field containing the raw log string (commonly `_raw`). If left blank, the Destination sends a JSON representation of the whole event. Use this when you want Google to parse the raw line itself rather than JSON you provide.

Authentication

To complete this part of the Destination definition, you will need either a Google Security Operations API key or your Google SecOps service account credentials. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.

Use the **Authentication method** to select an option. Whether you have set **General Settings** > **API version** to `V2` or `V1` determines what options are displayed.

For API version `V2`, the options are:

- **Service Account Credentials**: This option exposes a field where you can enter your Google SecOps service account directly.
- **Service Account Credentials Secret**: This option exposes a drop-down, in which you can select a stored secret that references your Google SecOps service account. A **Create** link is available to store a new, reusable secret.

For API version `V1`, the options include the two listed above plus two more:

- **API key**: This option exposes a field where you can enter your Google Security Operations API key.
- **API Key Secret**: This option exposes a drop-down, in which you can select a stored secret that references your Google Security Operations API key. A **Create** link is available to store a new, reusable secret.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: In this section's **Pipeline** drop-down list, you can select a single existing Pipeline or Pack to process data from this input before the data is sent through the Routes.

System fields: Specify any fields you want Cribl Stream to automatically add to events using this output. Wildcards are supported. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Whether to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. When enabled, any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. When disabled, all `Retry-After` headers are ignored.

Settings for failed HTTP requests: When you want to automatically retry requests that receive particular HTTP response status codes, use these settings to list those response codes.

For any HTTP response status codes that are not explicitly configured for retries, Cribl Stream applies the following rules:

Status Code	Action
Any in the `1xx`, `3xx`, or `4xx` series	Drop the request
Any in the `5xx` series	Retry the request

Upon receiving a response code that's on the list, Cribl Stream first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

By default, `429 (Too Many Requests)` and `503 (Service Unavailable)` are the only response codes configured for automatic retries. For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `30000` (30 seconds).
- **Backoff multiplier**: The base for the exponential backoff algorithm; defaults to `1`. A value of `2` means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Defaults to `30000` (30 seconds); minimum is `10,000` (10 seconds); maximum is `180,000`

(180 seconds, or 3 minutes).

Retry timed-out HTTP requests: When you want to automatically retry requests that have timed out, toggle this control on to display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are **not** authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Enter an amount of time, in seconds, to wait for a request to complete before aborting it. Defaults to `90`.

Request concurrency: Enter the maximum number of ongoing requests to allow before blocking.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `1024` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to five times the max body size (if set). If `0`, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Enter the maximum number of events to include in the request body. Defaults to `0` (unlimited).

Flush period (sec): Enter the maximum time to allow between requests. Be aware that small values could cause the payload size to be smaller than the configured **Body size limit**.

Extra HTTP Headers: Select **Add Header** to insert extra headers as **Name/Value** pairs. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.6.2. Google Cloud Chronicle API Destination

The **Google Cloud Chronicle API** Destination sends data to Google Cloud Chronicle using the v1alpha `ImportLogs` ingestion method. The Destination sends your raw, unstructured data, and Chronicle API uses its built-in parsers to transform it into the Unified Data Model (UDM).

Using the **Google Cloud Chronicle API** Destination, in contrast to the Google Security Operations (SecOps) Destination, lets you add labels to individual events, not only at the batch level, and take advantage of the improved performance and scaling offered by Chronicle API.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Working with Unstructured Log Types

Google Cloud Chronicle API supports a dynamic list of unstructured log types that evolves over time.

The **Default log type** setting lets you select the log type to send. To send a log of an unsupported type, type your custom log type into this field.

For Google's latest list of supported log types, see the Supported log types and default parsers in Google Cloud documentation to learn more about the specific log types.

How Log Types are Assigned

- **Event-specific**: When processing an event, Cribl Stream first checks if the event has a `__logType` field. If this field is present, its value determines the log type for that event.
- **Default**: If the event lacks a `__logType` field, Cribl Stream assigns the log type specified in the **Default log type** drop-down.
- **Custom**: If you entered a custom log name into **Default log type**, Cribl Stream will use this type.

Configure a Google Cloud Chronicle API Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.

- To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.

2. In the **New Destination** modal, configure the following under **General Settings**:
   - **Output ID**: Enter a unique name to identify this Destination. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
   - **Description**: Optionally, enter a description.
   - **Default log type**: Select or type in the default log type value to assign to events that lack a `__logType` field. See Working with Unstructured Log Types for more information.
   - **GCP project ID**: Enter the Google Cloud Platform (GCP) project ID to send events to. You can find the project ID on the Google Cloud Welcome page.
   - **GCP instance**: Enter ID of the GCP instance to send events to. This is the Chronicle customer UUID.
   - **Region**: Select the regional endpoint to send events to.
   - **Log text field**: Optionally, select the name of the event field that contains the log text to send. If not specified, Cribl Stream sends a JSON representation of the whole event.
3. Next, you can configure the following **Optional Settings**:
   - **Namespace**: User-configured environment namespace to identify the data domain from which the logs originated. This namespace is used as a tag to identify the appropriate data domain for indexing and enrichment functionality. The `__namespace` event field, if present, will overwrite this.
   - **Custom labels**: Custom labels to be added to every event. Custom labels are a Google SecOps feature used for data role-based access control (data RBAC) and other purposes. See the Google SecOps documentation. Click **Add label** to create a new label and specify it as a key-value pair.
     - **Enable RBAC**: Toggle on to designate a the custom label for RBAC and filtering, allowing granular multi-tenant data management directly within Chronicle. The label is flagged in the event payload, and Chronicle enforces access boundaries based on its value. This enables per-event RBAC, so you can dynamically tag events and restrict visibility to authorized roles without duplicating Destinations or relying on parser extensions. The label must resolve for all events in a batch.
   - **Backpressure behavior**: How to handle events when all receivers are exerting backpressure.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Under **Authentication**, select an **Authentication method** from the dropdown:
   - **Service Account Credentials**: This option exposes a field where you can enter your Google SecOps service account directly.
   - **Service Account Credentials Secret**: This option exposes a drop-down, in which you can select a stored secret that references your Google SecOps service account. A **Create** link is available to

store a new, reusable secret.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request

Status Code	Action
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Endpoint: The Chronicle API service endpoint. If empty, this defaults to the endpoint in the **Region** field. Otherwise, it must point to a Chronicle API-compatible endpoint (for example, https://custom-endpoint.googleapis.com).

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`. Each request can potentially hit a different HEC receiver.

Body size limit (KB): Maximum size, in KB, of the request body. Defaults to `5120`. Lowering the size can potentially result in more parallel requests and also cause outbound requests to be made sooner.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `5`.

- Retries happen on this flush interval.
- Any HTTP response code in the `2xx` range is considered success.
- Any response code in the `5xx` range is considered a retryable error, which will not trigger persistent queue (PQ) usage.
- Any other response code will trigger PQ (if PQ is configured as the Backpressure behavior).

Extra HTTP headers: Select **Add Header** to add **Name**/**Value** pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

8.2.6.3. Google Cloud Logging Destination

Cribl Stream supports sending log data to Google Cloud Logging, a real-time log storage and management service with search, analysis and alerting.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output to Google Cloud Logging

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Google Cloud Logging definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Log location type**: Select an option from the dropdown to determine where your log entries are categorized. Choose **Project**, **organization**, **Billing Account**, or **Folder**. For details, see Log Location Types.
    - **Log name expression**: Enter a JavaScript expression to compute the log name's value.
    - **Validate and correct log name**: Toggle on to ensure that log names are sanitized and truncated for compliance and reliability. Log name expression characters that do not meet Google requirements are replaced by an underscore `_`. Additionally, the log name is truncated to a maximum of 512 characters, which is the upper limit enforced by Google Cloud Logging. Without sanitization, Google Cloud Logging will reject requests with invalid log names, resulting in dropped events and potential data loss.
3. Under **Field Mappings**, configure the following:
    - **Payload format**: Select an option from the dropdown specify how data is structured within a log entry. Choose `Text` or `JSON`. For details, see Field Mappings

This Destination compresses all payloads, using gzip.

- **Log labels**: Click **Add label** to apply one or more labels to log entries. On each row of the resulting table, define each label with:
- **Label**: Arbitrary name for the label.
- **Value**: JavaScript expression to compute the label's value.
- **Resource type expression**: JavaScript expression to compute the field value for the managed resource type. Must evaluate to one of the monitored resource types listed here. If not specified, falls back to `global`.
- **Resource labels**: Click **Add label** to apply one or more labels to the managed resource. For details, see Resource Labels.
- **Severity expression**: JavaScript expression to compute the severity field's value. Must evaluate to one of the `LogSeverity` values listed here. Case-insensitive. If not specified, falls back to `DEFAULT`.
- **Insert ID expression**: JavaScript expression to compute the insert ID field's value.

4. Next, you can configure the following **Optional Settings**:
   - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. To add **Indexed Fields** to your LogEntry objects, provide JavaScript expressions to populate fields beyond the standard LogEntry structure, enabling richer analysis and correlation within Chronicle. For each field you want to index, provide a JavaScript expression. Ensure the expression's result matches the expected data type defined in the LogEntry documentation. Mismatched types prevent the field from appearing in Chronicle.
   - You can configure the following indexed fields using JavaScript expressions: **Trace expression**, **Span ID expression**, **Trace sampled expression**, **HTTP Request**, **Operation**, **Source Location**, and **Log Split**.

The LogEntry object's `timestamp` field is populated from the event's `_time` field.

6. Optionally, you can adjust the Authentication, Processing, and Advanced settings outlined in the sections below.
7. Select **Save**, then **Commit & Deploy**.

Log Location Types

Log Location Types allow you to categorize your log entries based on organizational structures. Choose one of the following options to associate log entries with a specific project, organization, billing account, or folder:

- `Project`: Displays a **Project ID expression** field. Here, enter a JavaScript expression to compute the value of the project ID to associate log entries with.
- `Organization`: Displays a **Organization ID expression** field. Enter a JS expression to compute the value of the organization ID to associate log entries with.
- `Billing Account`: Displays a **Billing account ID expression** field. Enter a JS expression to compute the value of the billing account ID to associate log entries with.
- `Folder`: Displays a **Folder expression** field. Enter a JS expression to compute the value of the folder ID to associate log entries with.

Field Mappings

Field Mappings specify how log entries should be structured. Choose one of the following options:

- `Text`: This default option displays a **Payload text expression** field. Here, enter JavaScript expression to compute the payload's value. Must evaluate to a JavaScript string value; if validation fails, this will default to a JSON string representation of the event.
- `JSON`: Displays a **Payload object expression** field. Enter a JS expression to compute the payload's value. Must evaluate to a JavaScript object value; if validation fails, this will default to the entire event.

Resource Labels

Each **Resource label** must correspond to a valid label for the resource type you've specified in the **Resource type expression**, or else Google Cloud Logging will drop it. On each row of the resulting table, define each label with:

- **Label**: Arbitrary name for the label.
- **Value**: JavaScript expression to compute the label's value.

Authentication

Use the **Authentication method** drop-down to select one of these options:

- **Auto**: This option uses the `GOOGLE_APPLICATION_CREDENTIALS` environment variable, and requires no configuration here.
- **Manual**: This default option displays a **Service account credentials** field for you to enter the contents of your service account credentials file (a set of JSON keys), as downloaded from Google Cloud. To insert the file itself, click the upload button at this field's upper right. As an alternative, you can use environment variables, as outlined here.

- **Secret**: This option exposes a drop-down in which you can select a stored secret that references the service account credentials described above. A **Create** link is available to store a new, reusable secret.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Connection timeout: Amount of time (in milliseconds) to wait for the connection to establish before retrying. Defaults to `10000` (10 sec.).

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30` sec.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. Defaults to `0` (unlimited).

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` sec.

Throttle request rate: Enter a maximum number of requests to send per second, as necessary to comply with Google Cloud's API limits. This field defaults to empty (no throttling), and accepts throttling rates as high as `2000` requests per second.

Environment: Optionally, specify a single Git branch on which to enable this configuration. If this field is empty, the config will be enabled everywhere.

Google Cloud Roles and Permissions

Your Google Cloud service account must have at least the following role:

- `Logs Writer`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

8.2.6.4. Google Cloud Pub/Sub Destination

Cribl Stream supports sending data to Google Cloud Pub/Sub, a managed real-time messaging service for sending and receiving messages between applications.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Prerequisites

When working with Google Cloud Pub/Sub, you need to know:

- The project where your credentials originated.
- The project that contains any topic you want to work with.

If the desired topic resides in the same project where your credentials originated, describe it using either its short name **or** its fully-qualified name.

If the desired topic resides in a different project than the one where your credentials originated, you **must** describe it using its fully-qualified name.

Here's an example of a fully-qualified topic name: `projects/my-project-id/topics/my-topic-id`.

The short name for the same topic would be: `my-topic-id`.

Configure Cribl Stream to Output to Pub/Sub

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Pub/Sub output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.

    - **Topic ID**: ID of the Pub/Sub topic to send events to. This static initial configuration of the topic is required; but you can optionally also override it dynamically on a per-event basis.

3. Next, you can configure the following **Optional Settings**:

- **Create topic**: Toggle on if you want Cribl Stream to create the topic on Pub/Sub if it does not exist.
- **Ordered delivery**: Toggle on if you want Cribl Stream to send events in the order that they arrived in the queue. For ordered delivery to work correctly, ensure that both the process receiving events and the subscription from which messages are read have ordered delivery enabled.
- **Region**: Region to publish messages to. Select `default` to allow Google to auto-select the nearest region. (If you've enabled **Ordered delivery**, the selected region must be allowed by message storage policy.)
- **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Persistent Queue, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Authentication method** drop-down to select one of these options:

- **Auto**: This option uses the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to provide the location of a credential JSON file. The environment variable can also be set directly in JSON format. When this option is set, Cribl Stream will first check whether the `GOOGLE_APPLICATION_CREDENTIALS` environment variable exists and properly set with valid credentials. If this environmental variable is missing or incorrect, Cribl Stream checks for credentials in two additional fallback locations, as described in the official Google Cloud documentation: How Application Default Credentials works. In addition to using the `GOOGLE_APPLICATION_CREDENTIALS` environment variable, you can use credentials attached to your Google Cloud instance.
- **Manual**: This default option displays a **Service account credentials** field for you to enter the contents of your service account credentials file (a set of JSON keys), as downloaded from Google Cloud.
  To insert the file itself, select the upload button at this field's upper right. As an alternative, you can use environment variables, as outlined here.
- **Secret**: This option exposes a drop-down in which you can select a stored secret that references the service account credentials described above. A **Create** link is available to store a new, reusable secret.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior.

Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Batch size: The maximum number of items the Google API should batch before it sends them to the topic. Defaults to `10` items.

Batch timeout (ms): The maximum interval (in milliseconds) that the Google API should wait to send a batch (if the configured **Batch size** limit has not been reached).. Defaults to `100` ms.

Queue size limit: Maximum number of queued batches before blocking. Defaults to `100`.

Batch size limit (KB): Maximum size for each sent batch. Defaults to `256` KB.

Flush period (sec): Enter the maximum time to allow between requests (in seconds). Low values could cause the payload size to be smaller than the configured **Batch size limit (KB)**. Defaults to `1` second.

Concurrent request limit: The maximum number of in-progress API requests before Cribl Stream applies backpressure. Defaults to `10`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Google Cloud Roles and Permissions

Your Google Cloud service account should have at least the following roles on topics:

- `roles/pubsub.publisher`
- `roles/pubsub.viewer` or `roles/viewer`

To enable Cribl Stream's **Create topic** option, your service account should have one of the following (or higher) roles:

- `roles/pubsub.editor`
- `roles/editor`

Either `editor` role confers multiple permissions, including those from the lower `viewer`, `subscriber`, and `publisher` roles. For additional details, see the Google Cloud Access Control topic.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Let's Change the Topic

The Pub/Sub Destination supports alternate topics specified at the event level in the `__topicOut` field. So (for example) if a Pub/Sub Destination is configured to send to main topic `topic1`, and Cribl Stream receives an event with `__topicOut: topic2`, then Cribl Stream will override the main topic and send this event to `topic2`.

However, a topic specified in the event's `__topicOut` field must already exist on Pub/Sub. If it does not, Cribl Stream cannot dynamically create the topic, and will drop the event. On the Destination's **Status** tab, the **Dropped** metric tracks the number of events dropped because a specified alternate topic did not exist.

(This example uses the short topic name, and that works as long as the topic resides in the project where your GCP credentials originated. If the topic resides in a different project, you must use the fully-qualified topic name, for example `__topicOut: projects/<my-project-id>/topics/topic2`.)

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.6.5. Google Cloud Storage Destination

Google Cloud Storage is a non-streaming Destination type.

Type: Non-Streaming | TLS Support: Yes | PQ Support: No

Configure Cloud Storage Permissions

Configure access permissions on the Cloud Storage side based on the authentication method you'll use on this Destination:

- **Auto authentication**: Supports both uniform and fine-grained access control.
- **Manual authentication**: Requires fine-grained access control on the buckets because it relies on ACLs to access resources.

The `storage.objects.create` permission is required.

If you enable **Verify if bucket exists** in the **Advanced Settings**, the `storage.objects.list` permission is also required.

Assign either of the following roles to grant the required permissions:

- `roles/storage.admin`
- `roles/storage.legacyBucketWriter`

To grant more limited access, combine these roles:

- `roles/storage.objectCreator` to provide the `storage.objects.create` permission
- `roles/storage.objectViewer` to provide the `storage.objects.list` permission

For details, see the Cloud Storage Overview of Access Control and Understanding Roles documentation.

Configure Cribl Stream to Output to Cloud Storage Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select**

Existing or **Add New**.

- To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.

2. In the **New Destination** modal, configure the following under **General Settings**:
   - **Output ID**: Enter a unique name to identify this Cloud Storage definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
   - **Description**: Optionally, enter a description.
   - **Bucket name**: Name of the destination bucket. This value can be a constant, or a JavaScript expression that can be evaluated only at init time. For example, referencing a Global Variable: `myBucket-${C.vars.myVar}`.
   - **Region**: Region where the bucket is located.
   - **Staging location**: Filesystem location in which to locally buffer files before compressing and moving to final destination. Cribl recommends that this location be stable and high-performance.

     In Cribl Stream, the **Staging location** field is not displayed or available on Cribl.Cloud-managed Worker Groups.

   - **Key prefix**: Root directory to prepend to path before uploading. Enter a constant, or a JS expression enclosed in single quotes, double quotes, or backticks.
   - **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (supported only on Linux, not Windows) exposes a Parquet Settings left tab, where you **must** configure certain options in order to export data in Parquet format.
3. Next, you can configure the following **Optional Settings**:
   - **Partitioning expression**: JavaScript expression that defines how files are partitioned and organized. Default is date-based. If blank, Cribl Stream will fall back to the event's `__partition` field value (if present); or otherwise to the root directory of the **Output Location** and **Staging Location**.
   - **Compression**: Data compression format used before moving to final destination. Defaults to `gzip` (recommended). This setting is not available when **Data format** is set to `Parquet`.
   - **File name prefix expression**: The output filename prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
   - **File name suffix expression**: The output filename suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.

To prevent files from being overwritten, Cribl appends a random sequence of 6 characters to the end of their names. This also applies to prefix and suffix expressions in file names.

For example, if you set the **File name prefix expression** to `CriblExec` and set the **File name suffix expression** to `.csv`, the file name might display as `CriblExec-adPRWM.csv` with `adPRWM` appended.

- **Backpressure behavior**: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Parquet, Processing, Retries and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Authentication method** drop-down to select one of these options:

- **Auto**: This option authenticates with the attached Google Cloud Platform (GCP) Service Account, relying on GCP IAM roles to access the appropriate GCP resources. This option is available only when Cribl Stream is on-prem, and the Worker Nodes are running in Google Compute Engine VMs on GCP. Supports both uniform and fine-grained access control.
- **Manual**: With this default option, authentication is via HMAC (Hash-based Message Authentication Code). To create a key and secret, see Google Cloud's Managing HMAC Keys for Service Accounts documentation. This option requires fine-grained access control on the GCS bucket because it relies on ACLs to access resources. Uniform access control is not supported with this method. This option exposes these two fields:
  - **Access key**: Enter the HMAC access key.
  - **Secret key**: Enter the HMAC secret.

The values for **Access key** and **Secret key** can be a constant, or a JavaScript expression (such as `${C.env.MY_VAR}`) enclosed in quotes or backticks, which allows configuration with environment variables.

- **Secret**: This option exposes a **Secret key pair** drop-down, in which you can select a stored secret that references the secret key pair described above. A **Create** link is available to store a new, reusable secret.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports `c*` wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. Toggle off (default) to expose the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Set the target memory size for page segments. Generally, set lower values to improve reading speed, or set higher values to improve compression. Value must be a positive integer smaller than the **Row group size**

value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Toggle on (default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Toggle on (default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300`. **Idle time limit (sec)**: Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Defaults to `30`. **Open file limit**: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to final output location. Defaults to `100`.

Cribl Stream will close files when **either** of the `File size limit (MB)` or the `Max file open time (sec)` conditions are met.

Add Output ID: Whether to append output's ID to staging location. Defaults to toggled on.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined Min free disk space amount.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).

- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Force close on shutdown: Toggle on to force all staged files to close during an orderly shutdown. This triggers immediate upload or finalization of any in-progress files, regardless of **Idle time**, **File open time**, or **File size** limits. Use this option to minimize data loss risk in ephemeral or auto-scaling environments. This could result in a higher number of small files, as files are closed before reaching their configured thresholds. Ensure your Worker Group or Fleet **Drain timeout** is set high enough to allow all staged data to be finalized before shutdown completes.

Enable dead-lettering: Toggle on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Endpoint: The Google Cloud Storage service endpoint. Typically, there is no reason to change the default https://storage.googleapis.com endpoint.

Object ACL: Select an Access Control List to assign to uploaded objects. Defaults to `private`.

Storage class: Select a storage class for uploaded objects.

Signature version: Signature version to use for signing requests. Defaults to `v4`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Common Issue

Nonspecific messages from Google Cloud of the form `Error: failed to close file` can indicate problems with the permissions listed above.

8.2.7. KAFKA

You can send data from Cribl Stream to the following Kafka services:

Destination	Description
Kafka	Send data to a Kafka topic
Confluent Cloud	Send data to Kafka topics on Confluent Cloud
Amazon MSK	Send data to a topic in Amazon MSK

8.2.7.1. Kafka Destination

Cribl Stream supports sending data to a Kafka topic.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Kafka uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must send events directly to receivers. You might need to adjust your firewall rules to allow this traffic.

Configure Cribl Stream to Output to Kafka

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Kafka definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Bootstrap servers**: List of Kafka bootstrap servers to connect to (for example, `localhost:9092`).
    - **Topic**: The topic on which to publish events. Can be overwritten using event's `__topicOut` field.
3. Next, you can configure the following **Optional Settings**:
    - **Acknowledgments**: Select the number of required acknowledgments. Defaults to `Leader`.
    - **Record data format**: Format to use to serialize events before writing to Kafka. Defaults to `JSON`. When set to `Protobuf`, the **Protobuf Format Settings** section (left tab) becomes available.
    - **Compression**: Codec to compress the data before sending to Kafka. Select `None`, `Gzip` `Snappy`, `LZ4`, or `ZSTD`.

Cribl strongly recommends enabling compression. Doing so improves Cribl Stream's performance, enabling faster data transfer using less bandwidth.

- **Backpressure behavior**: Select whether to block, drop, or queue incoming events when all receivers are exerting backpressure. Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Persistent Queue, TLS, Authentication, Processing, Retries, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.

Mode	Description
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Calculating the Time PQ Will Take to Engage

PQ will not engage until Cribl Stream has exhausted all attempts to send events to the Kafka receiver. This can take several minutes if requests continue to fail or time out.

To calculate the longest possible time this can take, multiply the values of **Request timeout** and **Retry limit**. For the default values (`60` seconds and `5`, respectively), this would be `60` seconds times `5` retries = 300 seconds, or 5 minutes.

TLS Settings (Client Side)

Enabled Default is toggled off. When toggled on:

- **Validate server certs**: Require client to reject any connection that is not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled off.

Server name (SNI): Leave this field blank. See Connecting to Kafka below.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**

Passphrase: Passphrase to use to decrypt private key.

Authentication

This section documents the SASL (Simple Authentication and Security Layer) authentication settings to use when connecting to brokers. Using TLS is highly recommended.

Enabled: Default is toggled off. When toggled on:

SASL mechanism: Use this drop-down to select the SASL authentication mechanism to use. The mechanism you select determines the controls displayed below.

PLAIN, SCRAM-256, or SCRAM-512

With any of these SASL mechanisms, select one of the following buttons:

Manual: Displays **Username** and **Password** fields to enter your Kafka credentials directly.

Secret: This option exposes a **Credentials secret** drop-down in which you can select a stored text secret that references your Kafka credentials. A **Create** link is available to store a new, reusable secret.

GSSAPI/Kerberos

Kerberos authentication is not supported for Cribl-managed Workers in Cribl.Cloud, but it is enabled for customer-managed (hybrid or on-prem) Worker Groups, whether the Leader is based in Cribl.Cloud or on-prem.

Selecting Kerberos as the authentication mechanism displays the following options:

Keytab location: Enter the location of the keytab file for the authentication principal.

Principal: Enter the authentication principal, for example: `kafka_user@example.com`.

Broker service class: Enter the Kerberos service class for Kafka brokers, for example: `kafka`.

You will also need to set up your environment and configure the Cribl Stream host for use with Kerberos. For details, see Kafka Authentication with Kerberos.

Schema Registry

This section governs Kafka Schema Registry Authentication for data encoded with Avro or JSON schemas, with schemas stored in the Confluent Schema Registry. Cribl Stream determines the schema type by inspecting the schema registered in the Confluent Schema Registry for the provided key and value schema IDs.

Enabled: Defaults to toggled off. When toggled on, displays the following controls:

Schema registry URL: URL for access to the Confluent Schema Registry. (For example, `http://localhost:8081`.)

Default key schema ID: Used when `__keySchemaIdOut` is not present to transform key values. Leave blank if key transformation is not required by default.

Default value schema ID: Used when `__valueSchemaIdOut` not present to transform `_raw`. Leave blank if value transformation is not required by default.

Connection timeout (ms): Specifies the maximum time allowed for establishing a successful connection to the schema registry. Defaults to 30000 ms (30 seconds). Valid range is 1000 ms - 60000 ms (or 1 sec to 60 sec).

Request timeout (ms): Maximum time to wait for a successful request from the schema registry. Defaults to 60000 ms (or 1 minute).

Retry Limit: Max number of times to retry requests to the schema registry before dropping the outgoing message. Defaults to 1; enter 0 to not retry at all.

Authentication enabled: Toggle on if your schema registry requires authentication.

- **Credentials secret**: Select or **Create** a stored text secret containing the username and password provided by your schema registry administrator.

TLS enabled: defaults to toggled off. When toggled on, displays the following TLS settings for the Schema Registry (in the same format as the TLS Settings (Client Side) above):

- **Validate server certs**: Require client to reject any connection that is not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled off.
- **Server name (SNI)**: Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.
- **Minimum TLS version**: Optionally, select the minimum TLS version to use when connecting.
- **Maximum TLS version**: Optionally, select the maximum TLS version to use when connecting.
- **Certificate**: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.
- **CA certificate path**: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.
- **Private key path (mutual auth)**: Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

- **Certificate path (mutual auth)**: Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
- **Passphrase**: Passphrase to use to decrypt private key.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For

example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Record size limit (KB, uncompressed): Maximum size (KB) of each record batch before compression. Setting should be < `message.max.bytes settings` in Kafka brokers. Defaults to `768`.

Events-per-batch limit: Maximum number of events in a batch before forcing a flush. Defaults to `1000`.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Connection timeout (ms): Maximum time to wait for a successful connection. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour).

Request timeout (ms): Maximum time to wait for a successful request. Defaults to `60000` ms (1 minute).

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Destination from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Destination to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Protobuf Format Settings

Definition set: From the drop-down, select `OpenTelemetry`. This makes the **Object type** setting available.

Object type: From the drop-down, select `Logs`, `Metrics`, or `Traces`.

Working with Protobufs

This Destination supports supports Binary Protobuf payload encoding. The Protobufs it sends can encode traces, metrics, or logs, the three types of telemetry data defined in the OpenTelemetry Project's Signals documentation:

- A **trace** tracks the progression of a single request.
    - Each trace is a tree of **spans**.
    - A span object represents the work being done by the individual services, or components, involved in a request as that request flows through a system.
- A **metric** provides aggreggated statistical information.
    - A metric contains individual measurements called **data points**.
- A **log**, in OpenTelemetry terms, is "any data that is not part of a distributed trace or a metric".

When configuring Pipelines (including pre-processing and post-processing Pipelines), you need to ensure that events sent to this Destination conform to the relevant Protobuf specification:

- For traces, opentelemetry/proto/trace/v1/trace.proto.
- For metrics, opentelemetry/proto/metrics/v1/metrics.proto.
- For logs, opentelemetry/proto/logs/v1/logs.proto.

This Destination will drop non-conforming events. If the Destination encounters a parsing error when trying to convert an event to OTLP, it discards the event and Cribl Stream logs the error.

Connecting to Kafka

Leave the TLS Settings > Server name (SNI) field blank

In Cribl Stream's Kafka-based Sources and Destinations (including this one), the Kafka library that Cribl Stream uses manages SNI (Server Name Indication) without any input from Cribl Stream. Therefore, you should leave the **TLS Settings** > **Server name (SNI)** field blank.

Setting this field in the Cribl Stream UI can cause traffic to be routed to the wrong brokers, because it interferes with the Kafka library's operation.

Connecting to a Kafka cluster involves using hostnames for two key components: bootstrap servers and brokers.

Brokers are servers that comprise the storage layer in a Kafka cluster. Bootstrap servers handle the initial connection to the Kafka cluster, and then return the list of brokers. A broker list can run into the hundreds. Every Kafka cluster has a `bootstrap.servers` property, defined as either a single `hostname:port` K-V pair,

or a list of them. If Cribl Stream tries to connect via one bootstrap server and that fails, Cribl Stream then tries another one on the list.

In the **General Settings** > **Bootstrap servers** list, you can enter either the hostnames of brokers that your Kafka server has been configured to use, or, the hostnames of one or more bootstrap servers. If Kafka returns a list of brokers that's longer than the list you entered, Cribl Stream keeps the full list internally. Cribl Stream neither saves the list nor makes it available in the UI. The connection process simply starts at the beginning whenever the Source or Destination is started.

Here's an overview of the connection process:

1. From the **General Settings** > **Bootstrap servers** list - where each broker is listed as a hostname and port - Cribl Stream takes a hostname and resolves it to an IP address.
2. Cribl Stream makes a connection to that IP address. Notwithstanding the fact that Cribl Stream resolved **one** particular hostname to that IP address, there may be **many** services running at that IP address - each with its own distinct hostname.
3. Cribl Stream establishes TLS security for the connection.

Although SNI is managed by the Kafka library rather than in the Cribl Stream UI, you might want to know how it fits into the connection process. The purpose of the SNI is to specify one hostname - that is, service - among many that might be running on a given IP address within a Kafka cluster. Excluding the other services is one way that TLS makes the connection more secure.

Working with Event Timestamps

By default, when an incoming event reaches this Destination, the underlying Kafka JS library adds a `timestamp` field set to the current time at that moment. The library gets the current time from the `Date.now()` JavaScript method. Events that this Destination sends to downstream Kafka receivers include this `timestamp` field.

Some production situations require the `timestamp` value to be the time an incoming event was originally created by an upstream sender, not the current time of its arrival at this Destination. For example, suppose the events were originally created over a wide time range, and arrive at the Destination hours or days later. In this case, the original creation time might be important to retain in events Cribl Stream sends to downstream Kafka receivers.

Here is how to satisfy such a requirement:

1. In a Pipeline that routes events to this Destination, use an Eval Function to add a field named `__kafkaTime` and write the incoming event's original creation time into that field. The value of `__kafkaTime` **must** be in Unix time (either seconds or milliseconds - both will work). For context, see this explanation of the related fields `_time` and `__origTime`.
2. This Destination will recognize the `__kafkaTime` field and write its value into the `timestamp` field. (This is similar to the way you can use the `__topicOut` field to overwrite a topic setting, as described above.)

If the `__kafkaTime` field is **not** present, the Destination will apply the default behavior (using `Date.now()`) described previously.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__topicOut`
- `__key`
- `__headers`
- `__keySchemaIdOut`
- `__valueSchemaIdOut`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

8.2.7.2. Confluent Cloud Destination

Cribl Stream supports sending data to Kafka topics on the Confluent Cloud managed Kafka platform.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Confluent Cloud uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must send events directly to receivers. You might need to adjust your firewall rules to allow this traffic.

Send Kafka Topic Data to Confluent Cloud

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination definition.
    - **Description**: Optionally, enter a description.
    - **Bootstrap servers**: List of Confluent Cloud bootstrap servers to connect to (for example, `myAccount.confluent.cloud:9092`).
    - **Topic**: The topic on which to publish events. Can be overwritten using event's `__topicOut` field.
3. Next, you can configure the following **Optional Settings**:
    - **Acknowledgments**: Select the number of required acknowledgments. Defaults to `Leader`.
    - **Record data format**: Format to use to serialize events before writing to Kafka. Defaults to `JSON`. When set to `Protobuf`, the **Protobuf Format Settings** section (left tab) becomes available.
    - **Compression**: Codec to use to compress the data before sending to Kafka. Select `None`, `Gzip`, `Snappy`, `LZ4`, or `ZSTD`. Defaults to `Gzip`.
    - **Backpressure behavior**: Select whether to block, drop, or queue incoming events when all receivers are exerting backpressure. Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary)

tag names.

4. Optionally, configure any Persistent Queue, Authentication, TLS, Schema Registry, Processing, Retries, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older

than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Calculating the Time PQ Will Take to Engage

PQ will not engage until Cribl Stream has exhausted all attempts to send events to the Kafka receiver. This can take several minutes if requests continue to fail or time out.

To calculate the longest possible time this can take, multiply the values of **Request timeout** and **Retry limit**. For the default values (`60` seconds and `5`, respectively), this would be `60` seconds times `5` retries = 300 seconds, or 5 minutes.

TLS Settings (Client Side)

Enabled When toggled on (default):

Autofill?: This setting is experimental.

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (such as the system's CA).

Server name (SNI): Leave this field blank. See Connecting to Kafka below.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Authentication

This section documents the SASL (Simple Authentication and Security Layer) authentication settings to use when connecting to bootstrap servers. Using TLS is highly recommended.

Enabled: Default is toggled off. When toggled on:

- **SASL mechanism**: Select the SASL (Simple Authentication and Security Layer) authentication mechanism to use. Defaults to `PLAIN`. `SCRAM-SHA-256`, `SCRAM-SHA-512`, and `GSSAPI/Kerberos` are also available. The mechanism you select determines the controls displayed below.

PLAIN, SCRAM-256, or SCRAM-512

With any of these SASL mechanisms, select one of the following buttons:

Manual: Displays **API key** and **Secret** fields to enter your Kafka credentials directly.

Secret: This option exposes a **Credentials secret** drop-down in which you can select a stored text secret that references your Kafka credentials. A **Create** link is available to store a new, reusable secret.

GSSAPI/Kerberos

Kerberos authentication is not supported for Cribl-managed Workers in Cribl.Cloud, but it is enabled for on-prem Worker Groups, whether the Leader is based in Cribl.Cloud or on-prem.

Selecting Kerberos as the authentication mechanism displays the following options:

Keytab location: Enter the location of the keytab file for the authentication principal.

Principal: Enter the authentication principal, for example: `kafka_user@example.com`.

Broker service class: Enter the Kerberos service class for Kafka brokers, for example: `kafka`.

You will also need to set up your environment and configure the Cribl Stream host for use with Kerberos. See Kafka Authentication with Kerberos for further detail.

Schema Registry

This section governs Kafka Schema Registry Authentication for data encoded with Avro or JSON schemas, with schemas stored in the Confluent Schema Registry. Cribl Stream determines the schema type by inspecting the schema registered in the Confluent Schema Registry for the provided key and value schema IDs.

Enabled: Default is toggled off. When toggled on, displays the following controls:

Schema registry URL: URL for access to the Confluent Schema Registry. (For example, `http://localhost:8081`.)

Default key schema ID: Used when `__keySchemaIdOut` is not present to transform key values. Leave blank if key transformation is not required by default.

Default value schema ID: Used when `__valueSchemaIdOut` not present to transform `_raw`. Leave blank if value transformation is not required by default.

Connection timeout (ms): Specifies the maximum time allowed for establishing a successful connection to the schema registry. Defaults to 30000 ms (30 seconds). Valid range is 1000 ms to 60000 ms (or 1 sec to 60 sec).

Request timeout (ms): Maximum time to wait for a successful request from the schema registry. Defaults to 60000 ms (or 1 minute).

Retry Limit: Max number of times to retry requests to the schema registry before dropping the outgoing message. Defaults to 1; enter 0 to not retry at all.

Authentication enabled: Toggle on if your schema registry requires authentication.

- **Credentials secret**: Select or **Create** a stored text secret containing the username and password provided by your schema registry administrator.

TLS enabled: defaults to toggled off. When toggled on, displays the following TLS settings for the Schema Registry (in the same format as the TLS Settings (Client Side) above):

- **Validate server certs**: Require client to reject any connection that is not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Default is toggled off.
- **Server name (SNI)**: Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

With a dedicated Confluent Cloud cluster hosted in Microsoft Azure, be sure to specify the **Server name (SNI)**. If this is omitted, Confluent Cloud might reset the connection to Cribl Stream.

- **Minimum TLS version**: Optionally, select the minimum TLS version to use when connecting.
- **Maximum TLS version**: Optionally, select the maximum TLS version to use when connecting.
- **Certificate**: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.
- **CA certificate path**: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.
- **Private key path (mutual auth)**: Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
- **Certificate path (mutual auth)**: Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
- **Passphrase**: Passphrase to use to decrypt private key.

Processing Settings

Post-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data before it is sent through this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Record size limit (KB, uncompressed): Maximum size (KB) of each record batch before compression. Setting should be < `message.max.bytes settings` in Kafka brokers. Defaults to `768`.

Events-per-batch limit: Maximum number of events in a batch before forcing a flush. Defaults to `1000`.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Connection timeout (ms): Maximum time to wait for a successful connection. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour).

Request timeout (ms): Maximum time to wait for a successful request. Defaults to `60000` ms (1 minute).

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Destination from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Destination to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Protobuf Format Settings

Definition set: From the drop-down, select `OpenTelemetry`. This makes the **Object type** setting available.

Object type: From the drop-down, select `Logs`, `Metrics`, or `Traces`.

Working with Protobufs

This Destination supports supports Binary Protobuf payload encoding. The Protobufs it sends can encode traces, metrics, or logs, the three types of telemetry data defined in the OpenTelemetry Project's Signals documentation:

- A **trace** tracks the progression of a single request.
    - Each trace is a tree of **spans**.

    - A span object represents the work being done by the individual services, or components, involved in a request as that request flows through a system.
- A **metric** provides aggreggated statistical information.
    - A metric contains individual measurements called **data points**.
- A **log**, in OpenTelemetry terms, is "any data that is not part of a distributed trace or a metric".

When configuring Pipelines (including pre-processing and post-processing Pipelines), you need to ensure that events sent to this Destination conform to the relevant Protobuf specification:

- For traces, opentelemetry/proto/trace/v1/trace.proto.
- For metrics, opentelemetry/proto/metrics/v1/metrics.proto.
- For logs, opentelemetry/proto/logs/v1/logs.proto.

This Destination will drop non-conforming events. If the Destination encounters a parsing error when trying to convert an event to OTLP, it discards the event and Cribl Stream logs the error.

Connecting to Kafka

> **Leave the TLS Settings > Server name (SNI) field blank**
>
> In Cribl Stream's Kafka-based Sources and Destinations (including this one), the Kafka library that Cribl Stream uses manages SNI (Server Name Indication) without any input from Cribl Stream. Therefore, you should leave the **TLS Settings** > **Server name (SNI)** field blank.
>
> Setting this field in the Cribl Stream UI can cause traffic to be routed to the wrong brokers, because it interferes with the Kafka library's operation. This is especially important for Confluent Cloud Dedicated clusters, which rely on SNI - as managed by the Kafka library - for routing.

Connecting to a Kafka cluster involves using hostnames for two key components: bootstrap servers and brokers.

Brokers are servers that comprise the storage layer in a Kafka cluster. Bootstrap servers handle the initial connection to the Kafka cluster, and then return the list of brokers. A broker list can run into the hundreds. Every Kafka cluster has a `bootstrap.servers` property, defined as either a single `hostname:port` K-V pair, or a list of them. If Cribl Stream tries to connect via one bootstrap server and that fails, Cribl Stream then tries another one on the list.

In the **General Settings** > **Bootstrap servers** list, you can enter either the hostnames of brokers that your Kafka server has been configured to use, or, the hostnames of one or more bootstrap servers. If Kafka returns a list of brokers that's longer than the list you entered, Cribl Stream keeps the full list internally. Cribl Stream neither saves the list nor makes it available in the UI. The connection process simply starts at the beginning whenever the Source or Destination is started.

Here's an overview of the connection process:

1. From the **General Settings** > **Bootstrap servers** list - where each broker is listed as a hostname and port - Cribl Stream takes a hostname and resolves it to an IP address.
2. Cribl Stream makes a connection to that IP address. Notwithstanding the fact that Cribl Stream resolved **one** particular hostname to that IP address, there may be **many** services running at that IP address - each with its own distinct hostname.
3. Cribl Stream establishes TLS security for the connection.

Although SNI is managed by the Kafka library rather than in the Cribl Stream UI, you might want to know how it fits into the connection process. The purpose of the SNI is to specify one hostname - that is, service - among many that might be running on a given IP address within a Kafka cluster. Excluding the other services is one way that TLS makes the connection more secure.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__topicOut`
- `__key`
- `__headers`
- `__keySchemaIdOut`
- `__valueSchemaIdOut`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.7.3. Amazon MSK Destination

Cribl Stream supports sending data to an Amazon Managed Streaming for Apache Kafka (MSK) topic.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Kafka uses a binary protocol over TCP. It does not support HTTP proxies, so Cribl Stream must send events directly to receivers. You might need to adjust your firewall rules to allow this traffic.

Configuring Cribl Stream to Output to Amazon MSK

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Amazon MSK Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Bootstrap servers**: List of Kafka bootstrap servers to connect to (for example, `localhost:9092`).
    - **Topic**: The topic on which to publish events. Can be overwritten using the event's `__topicOut` field.
    - **Region**: From the drop-down, select the name of the AWS Region where your Amazon MSK cluster is located.
3. Next, you can configure the following **Optional Settings**:
    - **Acknowledgments**: Select the number of required acknowledgments. Defaults to `Leader`.
    - **Record data format**: Format to use to serialize events before writing to Kafka. Defaults to `JSON`. When set to `Protobuf`, the **Protobuf Format Settings** section (left tab) becomes available.
    - **Compression**: Codec to compress the data before sending to Kafka. Select `None`, `Gzip`, `Snappy`, `LZ4`, or `ZSTD`. Defaults to `Gzip`.

Cribl strongly recommends enabling compression. Doing so improves Cribl Stream's performance, enabling faster data transfer using less bandwidth.

- **Backpressure behavior**: Select whether to block, drop, or queue incoming events when all receivers are exerting backpressure. Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Persistent Queue, TLS, Authentication, Processing, Retries, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.

Mode	Description
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Calculating the Time PQ Will Take to Engage

PQ will not engage until Cribl Stream has exhausted all attempts to send events to the Kafka receiver. This can take several minutes if requests continue to fail or time out.

To calculate the longest possible time this can take, multiply the values of **Request timeout** and **Retry limit**. For the default values (`60` seconds and `5`, respectively), this would be `60` seconds times `5` retries = 300 seconds, or 5 minutes.

TLS Settings (Client Side)

For Amazon MSK Sources and Destinations:

- IAM is the only type of authentication that Cribl Stream supports.
- Because IAM auth requires TLS, TLS is automatically enabled.

- **Validate server certs**: Require client to reject any connection that is not authorized by a CA in the **CA certificate path**, or by another trusted CA (such as the system's CA). Defaults to toggled on.

Server name (SNI): Leave this field blank. See Connecting to Kafka below.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**

Passphrase: Passphrase to use to decrypt private key.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Authentication

Use the **Authentication method** drop-down to select an AWS authentication method.

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, such as those running in a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.

- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair**: Use the drop-down to select an API key/secret key pair that you've configured in Cribl Stream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

Enable for MSK: Toggle on to use Assume Role credentials to access MSK.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role. This is required only when assuming a role that requires this ID in order to delegate third-party access. For details, see AWS' documentation.

Duration (seconds): Duration of the Assumed Role's session, in seconds. Minimum is 900 (15 minutes). Maximum is 43200 (12 hours). Defaults to 3600 (1 hour).

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2-20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Record size limit (KB, uncompressed): Maximum size (KB) of each record batch before compression. Setting should be < `message.max.bytes settings` in Kafka brokers. Defaults to `768`.

Events-per-batch limit: Maximum number of events in a batch before forcing a flush. Defaults to `1000`.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Connection timeout (ms): Maximum time to wait for a connection to complete successfully. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour).

Request timeout (ms): Maximum time to wait for Kafka to respond to a request. Defaults to `60000` ms (1 minute).

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `1000` (1 second).

Reauthentication threshold (ms): If the broker requires periodic reauthentication, this setting defines how long before the reauthentication timeout Cribl Stream initiates the reauthentication. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Destination from reauthenticating before the Kafka broker closes the connection.

A large value might cause the Destination to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Protobuf Format Settings

Definition set: From the drop-down, select `OpenTelemetry`. This makes the **Object type** setting available.

Object type: From the drop-down, select `Logs`, `Metrics`, or `Traces`.

Working with Protobufs

This Destination supports supports Binary Protobuf payload encoding. The Protobufs it sends can encode traces, metrics, or logs, the three types of telemetry data defined in the OpenTelemetry Project's Signals documentation:

- A **trace** tracks the progression of a single request.
    - Each trace is a tree of **spans**.
    - A span object represents the work being done by the individual services, or components, involved in a request as that request flows through a system.
- A **metric** provides aggreggated statistical information.
    - A metric contains individual measurements called **data points**.
- A **log**, in OpenTelemetry terms, is "any data that is not part of a distributed trace or a metric".

When configuring Pipelines (including pre-processing and post-processing Pipelines), you need to ensure that events sent to this Destination conform to the relevant Protobuf specification:

- For traces, opentelemetry/proto/trace/v1/trace.proto.
- For metrics, opentelemetry/proto/metrics/v1/metrics.proto.
- For logs, opentelemetry/proto/logs/v1/logs.proto.

This Destination will drop non-conforming events. If the Destination encounters a parsing error when trying to convert an event to OTLP, it discards the event and Cribl Stream logs the error.

Connecting to Kafka

Leave the TLS Settings > Server name (SNI) field blank

In Cribl Stream's Kafka-based Sources and Destinations (including this one), the Kafka library that Cribl Stream uses manages SNI (Server Name Indication) without any input from Cribl Stream. Therefore, you should leave the **TLS Settings** > **Server name (SNI)** field blank.

Setting this field in the Cribl Stream UI can cause traffic to be routed to the wrong brokers, because it interferes with the Kafka library's operation.

Connecting to a Kafka cluster involves using hostnames for two key components: bootstrap servers and brokers.

Brokers are servers that comprise the storage layer in a Kafka cluster. Bootstrap servers handle the initial connection to the Kafka cluster, and then return the list of brokers. A broker list can run into the hundreds. Every Kafka cluster has a `bootstrap.servers` property, defined as either a single `hostname:port` K-V pair, or a list of them. If Cribl Stream tries to connect via one bootstrap server and that fails, Cribl Stream then tries another one on the list.

In the **General Settings** > **Bootstrap servers** list, you can enter either the hostnames of brokers that your Kafka server has been configured to use, or, the hostnames of one or more bootstrap servers. If Kafka returns a list of brokers that's longer than the list you entered, Cribl Stream keeps the full list internally. Cribl Stream neither saves the list nor makes it available in the UI. The connection process simply starts at the beginning whenever the Source or Destination is started.

Here's an overview of the connection process:

1. From the **General Settings** > **Bootstrap servers** list - where each broker is listed as a hostname and port - Cribl Stream takes a hostname and resolves it to an IP address.
2. Cribl Stream makes a connection to that IP address. Notwithstanding the fact that Cribl Stream resolved **one** particular hostname to that IP address, there may be **many** services running at that IP address - each with its own distinct hostname.
3. Cribl Stream establishes TLS security for the connection.

Although SNI is managed by the Kafka library rather than in the Cribl Stream UI, you might want to know how it fits into the connection process. The purpose of the SNI is to specify one hostname - that is, service - among many that might be running on a given IP address within a Kafka cluster. Excluding the other services is one way that TLS makes the connection more secure.

Working with Event Timestamps

By default, when an incoming event reaches this Destination, the underlying Kafka JS library adds a `timestamp` field set to the current time at that moment. The library gets the current time from the `Date.now()` JavaScript method. Events that this Destination sends to downstream Kafka receivers include this `timestamp` field.

Some production situations require the `timestamp` value to be the time an incoming event was originally created by an upstream sender, not the current time of its arrival at this Destination. For example, suppose the events were originally created over a wide time range, and arrive at the Destination hours or days later. In this case, the original creation time might be important to retain in events Cribl Stream sends to downstream Kafka receivers.

Here is how to satisfy such a requirement:

1. In a Pipeline that routes events to this Destination, use an Eval Function to add a field named `__kafkaTime` and write the incoming event's original creation time into that field. The value of `__kafkaTime` **must** be in Unix time (either seconds or milliseconds - both will work). For context, see this explanation of the related fields `_time` and `__origTime`.
2. This Destination will recognize the `__kafkaTime` field and write its value into the `timestamp` field. (This is similar to the way you can use the `__topicOut` field to overwrite a topic setting, as described above.)

If the `__kafkaTime` field is **not** present, the Destination will apply the default behavior (using `Date.now()`) described previously.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__topicOut`
- `__key`
- `__headers`
- `__keySchemaIdOut`
- `__valueSchemaIdOut`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate

8.2.8. Metrics

You can send data from Cribl Stream to the following Metrics services:

Destination	Description
Graphite	Send data to a Graphite Destination
StatsD	Send data to a StatsD Destination
StatsD Extended	Send out data in expanded StatsD format

8.2.8.1. Graphite Destination

Cribl Stream supports sending data to a Graphite backend Destination.

Type: Streaming | TLS Support: No | PQ Support: Yes

Configure Cribl Stream to Output to a Graphite Backend

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Graphite definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Destination protocol**: Protocol to use when communicating with the Destination. Defaults to `UDP`.
    - **Host**: The hostname of the Destination.
    - **Port**: Destination port. Defaults to `8125`.
3. Next, you can configure the following **Optional Settings**:
    - **Throttling**: Displayed only when **General Settings** > **Destination protocol** is set to `TCP`. Rate (in bytes per second) at which at which to throttle while writing to an output. Also takes numerical values in multiples of bytes (KB, MB, GB, etc.). Default value of `0` indicates no throttling.
    - **Backpressure behavior**: Displayed only when **General Settings** > **Destination protocol** is set to `TCP`. Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Persistent Queue, Timeout, Processing, and Advanced settings outlined in the sections below.

5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Timeout Settings

This section is displayed only when **General Settings** > **Destination protocol** is set to `TCP`.

Connection timeout: Amount of time (in milliseconds) to wait for the connection to establish, before retrying. Defaults to `10000`.

Write timeout: Amount of time (milliseconds) to wait for a write to complete, before assuming connection is dead. Defaults to `60000`.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced

Record size limit (bytes): Used when Protocol is UDP. Specifies the maximum size of packets sent to the Destination. (Also known as the MTU - maximum transmission unit - for the network path to the destination system.) Defaults to `512`.

Flush period (sec): Used when Protocol is TCP. Specifies how often buffers should be flushed, sending records to the Destination. Defaults to `1`.

DNS resolution period (sec): Specify the interval (in seconds) to re-resolve hostnames. This reduces the frequency of DNS lookups, improving performance. Use this setting to balance the overhead of DNS lookup

calls with the expected frequency of changes in the DNS records. Defaults to `0` seconds, meaning DNS lookups occur for every outgoing batch.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.8.2. StatsD Destination

Cribl Stream supports sending data to a StatsD Destination.

Type: Streaming | TLS Support: No | PQ Support: Yes

Configure Cribl Stream to Output via StatsD

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this StatsD definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Destination protocol**: Protocol to use when communicating with the Destination. Defaults to `UDP`.
    - **Host**: The hostname of the Destination.
    - **Port**: Destination port. Defaults to `8125`.
3. Next, you can configure the following **Optional Settings**:
    - **Throttling**: Displayed only when **General Settings** > **Destination protocol** is set to `TCP`. Rate (in bytes per second) at which at which to throttle while writing to an output. Also takes numerical values in multiples of bytes (KB, MB, GB, etc.). Default value of `0` indicates no throttling.
    - **Backpressure behavior**: Displayed only when **General Settings** > **Destination protocol** is set to `TCP`. Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Persistent Queue, Timeout, Processing, and Advanced settings outlined in the sections below.

5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Timeout Settings

This section is displayed only when **General Settings** > **Destination protocol** is set to `TCP`.

Connection timeout: Amount of time (in milliseconds) to wait for the connection to establish, before retrying. Defaults to `10000`.

Write timeout: Amount of time (milliseconds) to wait for a write to complete, before assuming connection is dead. Defaults to `60000`.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Record size limit (bytes): Used when Protocol is UDP. Specifies the maximum size of packets sent to the Destination. (Also known as the MTU - maximum transmission unit - for the network path to the Destination system.) Defaults to `512`.

Flush period (sec): Used when Protocol is TCP. Specifies how often buffers should be flushed, sending records to the Destination. Defaults to `1`.

DNS resolution period (sec): Specify the interval (in seconds) to re-resolve hostnames. This reduces the frequency of DNS lookups, improving performance. Use this setting to balance the overhead of DNS lookup

calls with the expected frequency of changes in the DNS records. Defaults to `0` seconds, meaning DNS lookups occur for every outgoing batch.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.8.3. StatsD Extended Destination

Cribl Stream's StatsD Extended Destination supports sending out data in expanded StatsD format.

The output is an expanded StatsD metric protocol that supports dimensions, along with a sample rate for counter metrics. As with StatsD, downstream components listen for application metrics over UDP or TCP, can aggregate and summarize those metrics, and can relay them to virtually any graphing or monitoring backend.

For details about the syntax expected by one common downstream service, see Splunk's Expanded StatsD Metric Protocol documentation.

Type: Streaming | TLS Support: No | PQ Support: Yes

Configure Cribl Stream to Output via StatsD Extended

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this StatsD Extended definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Destination protocol**: Protocol to use when communicating with the Destination. Defaults to `UDP`.
    - **Host**: The hostname of the Destination.
    - **Port**: Destination port. Defaults to `8125`.
3. Next, you can configure the following **Optional Settings**:
    - **Throttling**: Displayed only when **General Settings** > **Destination protocol** is set to `TCP`. Rate (in bytes per second) at which at which to throttle while writing to an output. Also takes numerical values in multiples of bytes (KB, MB, GB, etc.). Default value of `0` indicates no throttling.
    - **Backpressure behavior**: Displayed only when **General Settings** > **Destination protocol** is set to `TCP`. Select whether to block, drop, or queue events when all receivers are exerting backpressure.

(Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.

- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Persistent Queue, Timeout, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Record size limit (bytes): Used when Protocol is UDP. Specifies the maximum size of packets sent to the Destination. (Also known as the MTU - maximum transmission unit - for the network path to the Destination system.) Defaults to `512`.

Flush period (sec): Used when Protocol is TCP. Specifies how often buffers should be flushed, sending records to the Destination. Defaults to `1`.

DNS resolution period (sec): Specify the interval (in seconds) to re-resolve hostnames. This reduces the frequency of DNS lookups, improving performance. Use this setting to balance the overhead of DNS lookup calls with the expected frequency of changes in the DNS records. Defaults to `0` seconds, meaning DNS lookups occur for every outgoing batch.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.9. New Relic Ingest

You can send data from Cribl Stream to the following New Relic Ingest services:

Destination	Description
New Relic Events	Send events to New Relic via the New Relic Event API
New Relic Logs & Metrics	Send events to New Relic Log API and Metric API

8.2.9.1. New Relic Events Destination

Cribl Stream supports sending events to New Relic via the New Relic Event API. Use this Destination to export ad hoc events that New Relic ingestion treats as custom events.

To export structured log and/or metric events, use Cribl Stream's New Relic Logs & Metrics Destination.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output Events to New Relic

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination.
    - **Description**: Optionally, enter a description.
    - **Region**: Select which New Relic region endpoint to use. Select `Custom` to manually enter your New Relic region's URL if it's not listed.
    - **Account ID**: Enter your New Relic account ID. (You can access this ID from New Relic's **account** drop-down, by selecting **Manage your plan**.)
    - **Event type**: Default `eventType` to apply when not specified in an event. You can use arbitrary values, as long as they do not conflict with New Relic reserved words. (Where an `eventType` is specified in an event, it will override this value.)
3. Under **Authentication**, select an **Authentication method** from the dropdown:

If an incoming event contains an internal field named `__newRelic_apiKey`, the New Relic Events Destination will use that field's value as the API key when sending the event to New Relic.

For events that do not contain a `__newRelic_apiKey` field, the Destination will use whatever API key you have configured in this section.

- **Manual**: This default option exposes an **API key** field. Directly enter your New Relic Ingest License API key, as you created or accessed it from New Relic's **account** drop-down. (For details, see the New Relic API Keys documentation.)
- **Secret**: This option exposes an **API key (text secret)** drop-down, in which you can select a stored secret that references a New Relic Ingest License API key. A **Create** link is available to store a new, reusable secret.

4. Next, you can configure the following **Optional Settings**:
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`. For the `Persistent Queue` option, see the section just below.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page.. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline to process data before sending the data out via this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx, 3xx` series	Drop the request
`408, 429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Defaults to toggled on, meaning compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `1024` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. Defaults to `0`, allowing unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1` second.

Extra HTTP headers: Click **Add Header** to insert extra headers as **Name/Value** pairs. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Verifying the New Relic Events Destination

Once you've configured event sources, create one or more Routes to send data to New Relic.

In New Relic, you can create visualizations incorporating the Cribl Stream-supplied data, then add them to new or existing dashboards as widgets.

Alternatively, in the New Relic backend, you can select **Query you data** (top nav) > **Events** (left tab), and then select the event type you exported from Cribl Stream.

To view more events, change the time frame at the upper right. To see raw events, click **Raw data** on the right.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.9.2. New Relic Logs & Metrics Destination

Cribl Stream supports sending events to the New Relic Log API and the New Relic Metric API.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

In Cribl Stream 3.1.2 and newer, this Destination now authenticates using New Relic's Ingest License API key. (New Relic will retire the Insights Insert API keys, which this Destination previously used for authentication.)

Also in v.3.1.2 and newer, Cribl Stream provides a separate New Relic Events Destination that you can use to send ad hoc (loosely structured) events to New Relic via the New Relic Event API.

Within New Relic's platform, you can monitor Cribl Stream's performance and data flow by installing New Relic's Cribl dashboard.

Configure Cribl Stream to Output to New Relic

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**: **Output ID**: Enter a unique name to identify this New Relic definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Region**: Select which New Relic region endpoint to use. Select `Custom` to manually enter your New Relic region's URL if it's not listed.
3. Under **Authentication**, select an **Authentication method** from the dropdown:

If an incoming event contains an internal field named `__newRelic_apiKey`, the New Relic Logs & Metrics Destination uses that field's value as the API key when sending the event to New Relic.

> For events that do not contain an `__newRelic_apiKey` field, the Destination uses whatever API key you have configured in the **Authentication method** settings.

- **Manual**: This default option exposes an **API key** field. Directly enter your New Relic Ingest License API key, as you created or accessed it from New Relic's **account** drop-down. (For details, see the New Relic API Keys documentation.)
- **Secret**: This option exposes an **API key (text secret)** drop-down, in which you can select a stored secret that references a New Relic Ingest License API key. A **Create** link is available to store a new, reusable secret.

4. Next, you can configure the following **Optional Settings**:
   - **Log type**: Name of the `logType` to send with events. For example, `observability` or `access_log`.

     This sets a default. Where a `sourcetype` is specified in an event, it will override this value.
   - **Log message field**: Name of the field to send as the log `message` value. If not specified, the event will be serialized and sent as JSON.
   - **Fields**: Additional fields to (optionally) add, as **Name-Value** pairs. Click **Add Field** to add more. For details, see [Fields]{#fields}
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`. For the `Persistent Queue` option, see the section just below.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Fields

These custom fields allow you to enrich your log data with specific details that are relevant to your use case. Click **Add Field** for more.

- **Name**: Enter the field name.
- **Value**:JavaScript expression to compute field's value, enclosed in single quotes, double quotes, or backticks. (Can evaluate to a constant.)

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior.

Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3

minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `1024` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1` second.

Extra HTTP headers: Click **Add Header** to insert extra headers as **Name/Value** pairs. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Verifying the New Relic Destination

Once you've configured log and/or metrics sources, create one or more Routes to send data to New Relic.

In New Relic, you can create visualizations incorporating the Cribl Stream-supplied data, then add them to new or existing dashboards as widgets.

Logs and metrics land in two different places in New Relic.

Log Queries

To access and query log data:

- Navigate to the New Relic home screen's **Logs** header option, and click the **(+)** button at right.
- Then to build your queries, use the **Find logs where** input field, and add desired columns to the table view below the graph,.

Metrics Queries

To access and query metrics data:

- From the New Relic home screen, *Click **Browse Data** > **Metrics** > **Can Search for metricNames**.
- Then, customize time range and dimensions to build the desired logic for your queries.
- Alternatively, you can use NRQL to build your own query searches.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

8.2.10. Prometheus

You can send data from Cribl Stream to the following Prometheus services:

Destination	Description
Prometheus	Send metric events to Prometheus remote write targets
Grafana Cloud	Send data to Loki for logs and Prometheus for metrics
Loki	Send log events to Loki

8.2.10.1. Prometheus Destination

Cribl Stream can send metric events to targets and third-party platforms that support Prometheus' remote write specification (overview here).

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

For an example of using this Destination to send host metrics to Grafana's Cribl Node Observability dashboard, see our System Metrics to Grafana topic.

Configure Cribl Stream to Output to Prometheus

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Prometheus output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Remote Write URL**: The endpoint to send events to, for example: `http://localhost:9200/write`
3. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Authentication, Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Auth token (text secret)**: This option exposes a **Token (text secret)** drop-down, in which you can select a stored text secret that references the bearer token described above. A **Create** link is available to store a new, reusable secret.
- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret)**: This option exposes a **Credentials secret** drop-down, in which you can select a stored text secret that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10

seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_host` (Cribl Stream Node that processed the event) and `cribl_wp` (Cribl Stream Worker Process that processed the event). Supports wildcards. Other options include:

- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_pipe` - Cribl Stream Pipeline that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx, 3xx` series	Drop the request
`401, 403, 408, 429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are **not** authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Metric renaming expression: A JavaScript expression that can be used to rename metrics. The default expression - `name.replace(/\\./g, \'_\')` - replaces all `.` characters in a metric's name with the Prometheus-supported `_` character. Use the `name` global variable to access the metric's name. You can access event fields' values via `__e.<fieldName>`.

Send metadata: Whether to generate and send metrics' metadata (`type` and `metricFamilyName`) along with the metrics. Toggling on displays this additional field:

- **Metadata flush period (sec)**: How frequently metrics metadata is sent out. Value must at least equal the base **Flush period (sec)**. (In other words, metadata cannot be flushed on a shorter interval.) Defaults to `60` seconds.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

If an event contains the internal field `__criblMetrics`, Cribl Stream will send it to the HTTP endpoint as a metric event. Otherwise, Cribl Stream will drop the event.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Unlike other HTTP-based Destinations, Prometheus does not display an **Advanced Settings** > **Compress** option. The Prometheus `remote_write` spec requires payloads to be protobuf-encoded and snappy-compressed, and data is always sent in this format. There is no separate user-configurable compression

setting for this Destination as Cribl Stream/Edge automatically encodes and compresses every payload before sending.

- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new one will be reattempted. This is to prevent sticking to a particular destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), a new connection will be established for the next request.
- When resolving the Destination's hostname, Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between Prometheus cluster nodes.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.10.2. Grafana Cloud Destination

Cribl Stream can send data to two of the services available in Grafana Cloud: Loki for logs and Prometheus for metrics. The Grafana Cloud Destination shapes events appropriately for Loki and Prometheus, and routes events to the correct endpoint for each service.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

For an alternative scenario that relies on our Prometheus Destination to send host metrics to Grafana's Cribl Node Observability dashboard, see our System Metrics to Grafana topic.

Preparing Prometheus and Loki to Receive Data from Cribl Stream

To define a Grafana Cloud Destination, you need a Grafana Cloud account.

While logged in to your Grafana account, navigate to the Grafana Cloud Portal, which should be located at `https://grafana.com/orgs/<your-organization-name>`, and complete the following steps.

Obtain an API key, setting its Role to `MetricsPublisher`. If you want Cribl Stream or an external KMS to manage the API key, configure a key pair that references the API key.

In the Prometheus tile, select **Send Metrics** to open the Prometheus configuration page. Write down:

- Your **Remote Write Endpoint** URL, for example: `https://prometheus-blocks-prod-us-central1.grafana.net/api/prom/push`.
- Your Prometheus **Username**.

In the Loki tile, select **Send Logs** to open the Loki configuration page. Write down:

- Your **Grafana Data Source settings** URL, for example: `https://logs-prod-us-central1.grafana.net`.
- Your Loki **User** ID.

Decide what type of authentication to use and prepare accordingly:

- If you choose Basic authentication, the username (**Username** in Prometheus, **User** in Loki) and password (simply your Grafana API key) will remain separate.
- If you choose token-based authentication, construct your tokens by concatenating username, colon (`:`), and password, for example `12345:cOQvDj6sJGFS3Bk2MguBW==`. Because the Prometheus and Loki usernames differ, you need to construct a separate token for each service.

Configure Cribl Stream to Output to Grafana Cloud

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Grafana Cloud output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Loki URL**: The endpoint to send log events to, for example: `https://logs-prod-us-central1.grafana.net`. This is the **Grafana Data Source settings** URL you wrote down earlier.
    - **Prometheus URL**: The endpoint to send metric events to, for example: `https://prometheus-blocks-prod-us-central1.grafana.net/api/prom/push`. This is the **Remote Write Endpoint** URL you wrote down earlier.
3. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Persistent Queue, Authentication, Processing, Retries, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues

- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Authentication

The **Authentication** tab provides separate **Loki** and **Prometheus** sections, enabling you to configure these inputs separately. The two sections provide identical options.

Select one of the following options for authentication:

- **Auth token**: Enter the bearer token that must be included in the authorization header. Use the token that you constructed earlier. In Grafana Cloud, the bearer token is generally built by concatenating the username and the API key, separated by a colon. For example: `<your-username>:<your-api-key>`.
- **Auth token (text secret)**: This option exposes a drop-down in which you can select a stored text secret that references the bearer token described above. A **Create** link is available to store a new, reusable

secret.

- **Basic**: This default option displays fields for you to enter HTTP Basic authentication credentials. **Username** is the Loki **User** or Prometheus **Username** that you wrote down earlier. **Password** is your API key in the Grafana Cloud domain.
- **Basic (credentials secret)**: This option exposes a **Credentials secret** drop-down, in which you can select a stored text secret that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

Processing Settings

Metric events can have dimensions, and log events have labels. Dimensions, labels, and their values are determined by several different settings in Cribl Stream. This section explains how that works, along with other kinds of settings.

Loki uses labels to define separate streams of logging data. This is a key concept. Cribl recommends that you familiarize yourself with the information and documentation Grafana provides about labels in Loki.

One canonical example is processing logs from servers in three environments: production, staging, and testing. You could create a label named `env` whose possible values are `prod`, `staging`, and `test`.

One basic principle is that if you set too many labels, you can end up with too many streams.

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output–both metric events, as dimensions; and, log events, as labels. Supports wildcards.

By default, includes `cribl_host` (Cribl Stream Node that processed the event) and `cribl_wp` (Cribl Stream Worker Process that processed the event). On the Loki side, this creates different streams, which prevents Loki from rejecting some events as being out of order when different Nodes or Worker Processes are emitting at different rates.

Other options include:

- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_pipe` - Cribl Stream Pipeline that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a

Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Send structured metadata: Adds key-value pairs from the `__structuredMetadata` field to logs as structured metadata. The field can be from a Source or added in a Pipeline. Non-string values are automatically converted to strings. To avoid unexpected results from this conversion, we recommend providing values as strings.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Loki and Prometheus might complain about entries being delivered out of order when **Request concurrency** is set > `1` and any of **Flush period (sec)**, **Body size limit (KB)**, or **Max events per request** are set to low values.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Metric renaming expression: A JavaScript expression that can be used to rename metrics. The default expression - `name.replace(/\\./g, \'_\')` - replaces all `.` characters in a metric's name with the Prometheus-supported `_` character. Use the `name` global variable to access the metric's name. You can access event fields' values via `__e.<fieldName>`.

Message format: Whether to send events as `Protobuf` (the default) or `JSON`.

Compress: When the **Message format** is `JSON`, this setting controls the data compression format used before sending the data to Grafana Cloud. Defaults to toggled on for GZIP-compression. (Applies only to Loki's JSON payloads. This toggle is hidden when the **Message format** is `Protobuf`, because both Prometheus' and Loki's Protobuf implementations are Snappy-compressed by default.)

Logs message field: The event field to send as log output, for example: `_raw`. All other event fields are discarded. If left blank, Cribl Stream sends a JSON representation of the whole event.

Logs labels: Name/value pairs where the value can be a static or dynamic expression that has access to all log event fields.

Failed request logging mode: Determines which data is logged when a request fails. Use the drop-down to select one of these options:

- `None` (default).
- `Payload`.

- `Payload + Headers`. Use the **Safe Headers** field below to specify the headers to log. If you leave that field empty, all headers are redacted, even with this setting.

Safe headers: List the headers you want to log, in plain text.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

If an event contains the internal field `__criblMetrics`, Cribl Stream will send it Prometheus as a metric event. If `__criblMetrics` is absent, Cribl Stream will treat the event as a log and send it to Loki.

The internal field `__structuredMetadata` is referenced when the **Send structured metadata** advanced setting is toggled on.

Labels:

The internal field `__labels` specifies labels to add to log events. If a label is set in both the `__labels` field and in **Logs labels** and/or **System fields**, Cribl Stream sends the value from `__labels` to Loki. Setting the `__labels` field in a Pipeline gives you a quick way to experiment with the logs being sent.

If there are no labels set (this would happen when **System fields**, **Logs labels**, and `__labels` are all empty), Cribl Stream adds a default `source` label, which prevents Loki from rejecting events. The `source` label the concatenation of `cribl`, underscore (`_`), source type, colon (`:`), source-name, where source name and type are values in the `__inputId` event field, for example: `cribl_metrics:in_prometheus_rw`. If `__inputId` is missing, `source` is set to `cribl`.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- The **Advanced Settings** > **Compress** toggle determines whether to compress the payload body before sending to Loki only. The toggle setting does not apply to Prometheus payloads, which are always compressed using Snappy.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new one will be reattempted. This is to prevent sticking to a particular destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), a new connection will be established for the next request.

- When resolving the Destination's hostname, Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between Grafana Cloud nodes.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.10.3. Loki Destination

Cribl Stream can send log events to Grafana's Loki log aggregation system.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configuring Cribl Stream to Output to Loki

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Loki output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Loki URL**: The endpoint to send events to, for example: `https://logs-prod-us-central1.grafana.net`.

        To proxy outbound HTTP/S requests, see System Proxy Configuration.
3. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Authentication, Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Auth token (text secret)**: This option exposes a drop-down in which you can select a stored text secret that references the bearer token described above. A **Create** link is available to store a new, reusable secret.
- **Basic**: This default option displays fields for you to enter HTTP Basic authentication credentials. **Username** is the Loki **User**. **Password** is your API key in the Grafana Cloud domain.
- **Basic (credentials secret)**: This option exposes a **Credentials secret** drop-down, in which you can select a stored text secret that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10

seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Loki uses labels to define separate streams of logging data. This is a key concept. Cribl recommends that you familiarize yourself with the information and documentation Grafana provides about labels in Loki.

One canonical example is processing logs from servers in three environments: production, staging, and testing. You could create a label named `env` whose possible values are `prod`, `staging`, and `test`.

One basic principle is that if you set too many labels, you can end up with too many streams.

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to log events as labels. Supports wildcards.

By default, includes `cribl_host` (Cribl Stream Node that processed the event) and `cribl_wp` (Cribl Stream Worker Process that processed the event). On the Loki side, this creates different streams, which prevents Loki from rejecting some events as being out of order when different Nodes or Worker Processes are emitting at different rates.

Other options include:

- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_pipe` - Cribl Stream Pipeline that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Validate server certs: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Send structured metadata: Adds key-value pairs from the `__structuredMetadata` field to logs as structured metadata. The field can be from a Source or added in a Pipeline. Non-string values are automatically converted

to strings. To avoid unexpected results from this conversion, we recommend providing values as strings.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `1`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `15`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Enable dynamic headers: Toggle on to enable dynamic HTTP headers where the Destination checks each event for the internal field `__headers`. Events with the same `__headers` are batched and sent together in a single HTTP request. Events with different `__headers` are sent in separate requests. Use this when you need per-event custom headers, such as for multi-tenant routing or dynamic authentication.

Message format: Whether to send events as `Protobuf` (the default) or `JSON`.

Compress: When the **Message format** is `JSON`, this setting controls the data compression format used before sending the data to Loki. Defaults to toggled on for GZIP-compression. (Applies only to Loki's JSON payloads. This toggle is hidden when the **Message format** is `Protobuf`, because both Prometheus' and Loki's Protobuf implementations are Snappy-compressed by default.)

Logs message field: The event field to send as log output, for example: `_raw`. All other event fields are discarded. If left blank, Cribl Stream sends a JSON or Protobuf representation of the whole event.

Logs labels: Name/value pairs where the value can be a static or dynamic expression that has access to all log event fields.

Failed request logging mode: Determines which data is logged when a request fails. Use the drop-down to select one of these options:

- `None` (default).
- `Payload`.
- `Payload + Headers`. Use the **Safe Headers** field below to specify the headers to log. If you leave that field empty, all headers are redacted, even with this setting.

Safe headers: List the headers you want to log, in plain text.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses internal fields to assist in forwarding data to a Destination.

- `__labels` specifies labels to add to log events. If a label is set in both the `__labels` field and in Logs labels and/or System fields, Cribl Stream sends the value from `__labels` to Loki. Setting the `__labels` field in a Pipeline gives you a quick way to experiment with the logs being sent.
- `__structuredMetadata` is referenced when the **Send structured metadata** advanced setting is toggled on.
- `__headers` allows you to define custom HTTP headers on a per-event basis when the **Enable dynamic headers** advanced setting is toggled on. Top-level string key-value pairs are supported. Non-string values are ignored.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.11. Splunk

You can send data from Cribl Stream to the following Splunk services:

Destination	Description
Splunk HEC	Stream data to a Splunk HEC receiver
Splunk Single Instance	Stream data to a Splunk instance
Splunk Load Balanced	Load-balance data streaming to multiple Splunk receivers

8.2.11.1. Splunk HEC Destination

The **Splunk HEC** Destination can stream data to a Splunk HEC (HTTP Event Collector) receiver through the event, raw, and S2S endpoints.

The data arrives to Splunk cooked and parsed, so it enters at the Splunk data pipeline's indexing segment.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Looking for a quick way to switch all of your S2S Destinations to Splunk HEC Destinations? Follow our how-to guide: Switch Cribl Stream Destinations from S2S to Splunk HEC.

The Splunk HEC Destination is recommended for Splunk Cloud. For details, see Splunk Cloud Platform and BYOL Integrations.

Configure Cribl Stream to Output to Splunk HEC Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Splunk HEC definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Load balancing**: Toggle on to specify multiple Splunk HEC endpoints and load weights. If toggled off and you notice that Cribl Stream is not sending data to all possible IP addresses, enable **Advanced Settings** > **Round-robin DNS**.
    - **Splunk HEC endpoint(s)**: URLs of Splunk HEC endpoints to send events to.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Displays an **HEC Auth token** field for you to enter your Splunk HEC token.

- **Secret**: This option exposes an **HEC Auth token (text secret)** drop-down, in which you can select a stored secret that references the HEC token described above. A **Create** link is available to store a new, reusable secret.

4. Next, you can configure the following **Optional Settings**:
   - **Exclude current host IPs**: This toggle appears when **Load balancing** is toggled on. It determines whether to exclude all IPs of the current host from the list of any resolved hostnames. Default is toggled off, which keeps the current host available for load balancing.
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Persistent Queue, TLS, Processing, Retries and Advanced settings outlined in the sections below.

If you're sending data to Splunk Cloud, Cribl recommends a maximum value of `1 MB` for **Body size limit (KB)** under Advanced settings. This upper limit is defined by the maximum content length for the default HTTP Event Collector (HEC) provided by Splunk Cloud.

6. Select **Save**, then **Commit & Deploy**.

Splunk HEC Endpoints

Use the **Splunk HEC Endpoints** table to specify a known set of receivers on which to load-balance data. The table appears only when **Load balancing** is toggled on.

To specify more receivers on new rows, select **Add Endpoint**. Each row provides the following fields:

- **HEC Endpoint**: Specify the URL to a Splunk HEC endpoint to send events to. See the Splunk documentation for details on getting your URL. For Splunk Cloud endpoints, `https` protocol is recommended. Supported endpoints include:
  - `/services/collector/event` (default)
  - `/services/collector/raw`
  - `/services/collector/s2s`
- **Load Weight**: Set the relative traffic-handling capability for each connection by assigning a weight (`> 0`). This column accepts arbitrary values, but for best results, assign weights in the same order of magnitude to all connections. Cribl Stream will attempt to distribute traffic to the connections according to their relative weights.

The final column provides an `X` button to delete any row from the table.

When you first enable load balancing, or if you edit the load weight once your data is load-balanced, give the logic time to settle. The changes might take a few seconds to register.

For details on configuring all these options, see About Load Balancing.

Indexer Acknowledgement

Do not toggle **Enable Indexer Acknowledgement** on for the Splunk token. With this toggle on, Splunk receivers expect the Channel GUID to be passed in, and requests will fail with errors of this form:

```
channel: output:splunk_cloud_hec
cid: w2
level: error
message: Request failed
reason: Received status code=400, method=POST, url=https://http-inputs-bazookatron.sp
response: {"text":"Data channel is missing","code":10}
time: 2023-02-14T15:26:03.413Z
```

Copy

TLS Settings (Client Side)

Enabled: Defaults to toggled off. When toggled on:

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a hostname, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

By default, this Destination rejects certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). You can control this behavior using the **Validate server certs** advanced setting.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key. Often the value of the `sslPassword` or similar parameter in the `outputs.conf` or `server.conf` file.

Single .pem File

If you have a **single** .pem file containing `cacert`, `key`, and `cert` sections, enter it in all of these fields above: **CA certificate path**, **Private key path (mutual auth)**, and **Certificate path (mutual auth)**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.

Mode	Description
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx, 3xx` series	Drop the request
`401, 403, 408, 429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Output multi-metrics: Toggle on to output multiple-measurement metric data points. (Supported in Splunk 8.0 and above, this format enables sending multiple metrics in a single event, improving the efficiency of your Splunk capacity.)

Validate server certs: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts. (This setting is available only when **General Settings** > **Load balancing** is toggled off.)

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`. Each request can potentially hit a different HEC receiver.

Body size limit (KB): Maximum size, in KB, of the request body. Defaults to `4096`. Lowering the size can potentially result in more parallel requests and also cause outbound requests to be made sooner.

If you're sending data to Splunk Cloud, Cribl recommends a maximum value of `1 MB` for **Body size limit (KB)**. This upper limit is defined by the maximum content length for the default HTTP Event Collector (HEC) provided by Splunk Cloud.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1`.

- Retries occur on the configured flush interval.
- Any HTTP response code in the `2xx` range is considered success.
- Any response code in the `5xx` range is considered a retryable error, which will not trigger Persistent Queue (PQ) usage.
- `4xx` responses are dropped by default and do not trigger PQ. To retry specific `4xx` codes (for example, `429`), add them explicitly under **Retries**. When configured as retryable, those responses will engage PQ (if PQ is configured as the Backpressure behavior).

Extra HTTP headers: Select **Add Header** to add **Name**/**Value** pairs to pass as additional HTTP headers. Values will be sent encrypted.

The next two fields appear only when you toggle **Load balancing** on in the **General Settings**.

DNS resolution period (seconds): Re-resolve any hostnames after each interval of this many seconds, and pick up destinations from A records. Defaults to `600` seconds.

Load balance stats period (seconds): The duration (in seconds) Cribl Stream tracks historical traffic data to influence load balancing decisions. Defaults to `300` seconds. If a hostname resolves to multiple IPs (via DNS A records), all IPs inherit the weight assigned to the hostname unless each IP is explicitly configured with its own weight. Cribl Stream prioritizes IP-specific settings over hostname-derived settings for load balancing.

Throttling: The rate (in bytes per second) to throttle while writing to the Destination. Enter `0` (the default) for no limit, or a value in multi-byte units such as KB, MB, or GB. When outbound traffic reaches this limit, Cribl Stream slows down sending to this Destination. Any additional data follows the existing **Backpressure behavior** and **Persistent Queue** settings.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Splunk App Settings provides two fields that take effect only in the Cribl App for Splunk. (Cribl Stream ignores their values.)

- **Next processing queue**: Specify the next Splunk processing queue to send the events to, after HEC processing. Defaults to `indexQueue`.
- **Default _TCP_ROUTING**: Specify the value of the `_TCP_ROUTING` field for events that do not have `_ctrl._TCP_ROUTING` set. Defaults to `nowhere`. This is useful only when you expect the HEC receiver to route this data on to another destination.

Event Format

Cribl Stream uses the internal field `__criblMetrics` to determine whether to process an event as a metric or a log. See Splunk's documentation on their event format for HEC events.

Metric event: When an event contains the `__criblMetrics` field, the Destination constructs a JSON object following Splunk's metric event schema.

- Any key-value pairs in `__criblMetrics.dims` are added as dimension fields to the final Splunk metric event.
- Metrics listed in `__criblMetrics.values` are formatted based on the **Output multi-metrics** advanced setting:
    - **Enabled**: A single event is created that bundles multiple metrics using the `metric_name:<name>` format for each value.
    - **Disabled**: A separate event is created for each metric, using the standard `metric_name` and `_value` fields.

Log event: If an event does not contain the `__criblMetrics` field, it's processed as a log event.

- If `_raw` is present, it becomes the value of the `event` property in the JSON payload.
- If `_raw` is absent, the entire event (minus internal fields) is serialized as a JSON object and used as the `event` property.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new connection will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If the server does not support keepalives - or if the server closes a pooled connection while idle - a new connection will be established for the next request.
- When resolving the Destination's hostname with load balancing disabled, Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between Splunk HEC servers.
- See Splunk's documentation on editing `fields.conf` to ensure the visibility of index-time fields sent to Splunk by Cribl Stream.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Cribl University offers an Advanced Troubleshooting > Destination Integrations: Splunk Cloud short course. To follow the direct course link, first log into your Cribl University account. (To create an account, select the **Sign up** link. You'll need to select through a short **Terms & Conditions** presentation, with chill music, before proceeding to courses - but Cribl's training is always free of charge.) Once logged

8.2.11.2. Splunk Single Instance Destination

This Splunk Destination can stream data to a **free** Splunk Cloud instance. From the perspective of the receiving Splunk Cloud instance, the data arrives cooked and parsed.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Looking for a quick way to switch all of your S2S Destinations to Splunk HEC Destinations? Follow our how-to guide: Switch Cribl Stream Destinations from S2S to Splunk HEC.

Splunk Cloud Platform

To establish secure connections to Splunk Cloud, you'll need to configure SSL settings using the private and public keys from the Splunk Cloud Universal Forwarder credentials package.

For details on which Destination to use for Splunk Cloud, see Splunk Cloud Platform and BYOL Integrations. The Splunk HEC Destination is recommended for most use cases.

Configure Cribl Stream to Output to Splunk Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Splunk Single Instance definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Address**: Hostname of the Splunk receiver.
    - **Port**: The port number on the host.
3. Next, you can configure the following **Optional Settings**:

- **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to

`Block`.

- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Persistent Queue, TLS, Processing, Timeout, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

TLS Settings (Client Side)

Enabled: Defaults to toggled off. When toggled on:

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Defaults to toggled on.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a hostname, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key. Often the value of the `sslPassword` or similar parameter in the `outputs.conf` or `server.conf` file.

Single .pem File

If you have a **single** .pem file containing `cacert`, `key`, and `cert` sections, enter it in all of these fields above: **CA certificate path**, **Private key path (mutual auth)**, and **Certificate path (mutual auth)**.

Timeout Settings

Connection timeout: Amount of time (in milliseconds) to wait for the connection to establish, before retrying. Defaults to `10000`.

Write timeout: Amount of time (in milliseconds) to wait for a write to complete, before assuming connection is dead. Defaults to `60000`.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Output multiple metrics: Toggle on to output multiple-measurement metric data points. (Supported in Splunk 8.0 and above, this format enables sending multiple metrics in a single event, improving the efficiency of your Splunk capacity.)

Minimize in-flight data loss: If toggled on (default), Cribl Stream will check whether the indexer is shutting down, and if so, will stop sending data. This helps minimize data loss during shutdown. (Note that Splunk logs will indicate that the Cribl app has set `UseAck` to `true`, even though Cribl does not enable full `UseAck` behavior.) If toggled off, exposes the following alternative option:

Failed health check limit: Displayed (and set to `1` by default) only if **Minimize in-flight data loss** is disabled. This option sends periodic requests to Splunk once per minute, to verify that the Splunk endpoint is still alive and can receive data. Its value governs how many failed requests Cribl Stream will allow before closing this connection.

A low threshold value improves connections' resilience, but by proliferating connections, this can complicate troubleshooting. Set to `0` to disable health checks entirely - here, if the connection to Splunk is forcibly closed, you risk some data loss.

Max S2S version: The highest version of the Splunk-to-Splunk protocol to expose during handshake. Defaults to `v4`; `v3` is also available.

Throttling: Throttle rate, in bytes per second. Defaults to `0`, meaning no throttling. Multiple-byte units such as KB, MB, GB, and so forth, are also allowed - for example, `42 MB`. When throttling is engaged, your **Backpressure behavior** selection determines whether Cribl Stream will handle excess data by blocking it, dropping it, or queueing it to disk.

- When used in conjunction with **Connection limit**, the throttling rate is applied per connection. For instance, if **Connection limit** is set to `2` and throttling is set to `3 MB`, each of the two connections can send data at a rate of up to 3 MB per second, resulting in a total potential throughput of 6 MB per second across both connections.

Nested field serialization: Specifies how to serialize nested fields into index-time fields. Defaults to `None`.

Log failed requests to disk: Toggle on to make the payload of the most recently failed request available for inspection. See Inspect Payload to Troubleshoot Closed Connections below.

Authentication method: Use the buttons to select one of these options:

- **Manual**: In the resulting **Auth token** field, enter the shared secret token to use when establishing a connection to a Splunk indexer.
- **Secret**: This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the auth token described above. A **Create** link is available to store a new, reusable secret.

Compression: This setting controls compression for Splunk-to-Splunk `v4` data. Select the **Compression** type from the drop-down list:

- `Disabled`: Compression is turned off. This is the default setting for both existing and new Destinations in Cribl Stream versions 4.11 and older. Connections will be closed if the server attempts to use compression.
- `Always`: Cribl Stream will always send compressed data, regardless of the receiver's compression configuration.

This setting is not available when **Max S2S version** is set to `v3`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Inspect Payload to Troubleshoot Closed Connections

When a downstream receiver closes connections from this Destination (or just stops responding), inspecting the payload of the most recently failed request can help you find the cause. For example:

- Suppose you send an event whose size is larger than the downstream receiver can handle.
- Suppose you send an event that has a `number` field, but the value exceeds the highest number that the downstream receiver can handle.

When **Log failed requests to disk** is enabled, you can inspect the last failed request payload. Here is how:

1. In the Destination UI, navigate to the **Logs** tab.
2. Find a log entry with a `connection error` message.
3. Expand the log entry.
4. If the message includes the phrase `See payload file for more info`, note the path in the `file` field on the next line.

Now you have the path to the directory where Cribl Stream is storing the payload from the last failed request.

Notes about Forwarding to Splunk

- Data sent to Splunk is not compressed.
- The only `ack` from indexers that Cribl Stream listens for and acts upon is the shutdown signal described in Minimize in-flight data loss above.
- If events have a Cribl Stream internal field called `__criblMetrics`, they'll be forwarded to Splunk as metric events.
- If events do **not** have a `_raw` field, they'll be serialized to JSON prior to sending to Splunk.
- See Splunk's documentation on editing `fields.conf` to ensure the visibility of index-time fields sent to Splunk by Cribl Stream.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Cribl University offers an Advanced Troubleshooting > Destination Integrations: Splunk Cloud short course. To follow the direct course link, first log into your Cribl University account. (To create an account, select the **Sign up** link. You'll need to select through a short **Terms & Conditions** presentation, with chill music, before proceeding to courses - but Cribl's training is always free of charge.) Once logged in, check out other useful Advanced Troubleshooting short courses and Troubleshooting Criblets.

8.2.11.3. Splunk Load Balanced Destination

The **Splunk Load Balanced** Destination can load-balance the data it streams to multiple Splunk receivers. Downstream Splunk instances receive the data cooked and parsed.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Looking for a quick way to switch all of your S2S Destinations to Splunk HEC Destinations? Follow our how-to guide: Switch Cribl Stream Destinations from S2S to Splunk HEC.

Splunk Cloud Platform

To establish secure connections to Splunk Cloud, you'll need to configure SSL settings using the private and public keys from the Splunk Cloud Universal Forwarder credentials package.

For details on which Destination to use for Splunk Cloud, see Splunk Cloud Platform and BYOL Integrations. The Splunk HEC Destination is recommended for most use cases.

Configure Cribl Stream to Load-Balance to Multiple Splunk Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Splunk LB Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Indexer discovery**: When enabled, Cribl Stream will automatically discover the indexers in an indexer clustering environment. </
      This hides both **Exclude current host IPs** and the **Destinations** section, and displays the following fields:

- **Site**: Clustering site whose indexers need to be discovered. In the case of a single site cluster, `default` is the default entry.
- **Cluster Manager URI**: Full URI of Splunk cluster manager, in the format: `scheme://host:port`. (Worker Nodes normally access the cluster manager on **port 8089** to get the list of currently online indexers.)
- **Refresh period**: Time interval (in seconds) between two consecutive fetches of the indexer list from the cluster manager. Defaults to `300` seconds (five minutes).
- **Validate cluster manager certificates**: If toggled on (default), Cribl Stream is configured to reject any cluster manager certificate during indexer discovery that is not authorized by your system's Certificate Authority (CA). Cribl strongly recommends the default setting unless you must allow untrusted (for example, self-signed) certificates.

3. Under **Authentication tokens**, select one of the **Authentication method** options for authenticating to the cluster manager for indexer discovery:
   - **Manual**: In the resulting **Auth token** field, enter the required token.
   - **Secret**: This option exposes an **Auth token (text secret)** drop-down, from which you can select a stored secret that references the auth token. A **Create** link is available to store a new, reusable secret.

     If you use multiple cluster managers, you can add multiple tokens to cycle through them if a connection to the primary cluster manager fails. Select **Add token** to configure another authentication token.

Each Worker Process performs its own indexer discovery according to the above settings.

4. The **Destinations** section appears only when **Indexer discovery** is toggled off (default). Here, you specify a known set of Splunk receivers on which to load-balance data. Then you assign a weight to Cribl Stream's connection to each receiver. The higher the weight assigned to a given connection, the more traffic Cribl Stream will attempt to allocate to that connection. Click **Add Destination** to specify more receivers on new rows. Each row provides the following fields:
   - **Address**: Hostname or IP address of the receiver. If you enter a hostname, Cribl Stream tries to resolve it to its IP address or addresses. If a hostname has multiple A records - that is, if multiple receivers are behind it - Cribl Stream will assign the weight you assigned to the host to all of its IP addresses.
   - **Port**: Port number to send data to.
   - **TLS**: Whether to inherit TLS configs from group setting, or disable TLS. Defaults to `inherit`.
   - **TLS servername**: Servername to use if establishing a TLS connection. If not specified, defaults to the connection host (if not an IP). Otherwise, uses the global TLS settings.

- **Load weight**: Set the relative traffic-handling capability for each connection by assigning a weight greater than `0`. For details, see Destinations below.

5. Next, you can configure the following **Optional Settings**:
   - **Exclude current host IPs**: Enable if you want to exclude all the current host's IP addresses from the list of resolved hostnames.
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`. When toggled to `Persistent Queue`, adds the Persistent Queue Settings section (left tab) to the modal.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
6. Optionally, you can adjust the Persistent Queue, TLS, Processing, Timeout, and Advanced settings outlined in the sections below.
7. Select **Save**, then **Commit & Deploy**.

Destinations

- For best results, the weights for all connections should have the same order of magnitude. For example, if you have three connections, weights of `10`, `50`, and `40` would work; so would weights of `1`, `5`, and `4` - but weights like `10`, `120`, and `9` mix different orders of magnitude and should be avoided.
- You can add connections and keep them unused except for special circumstances such as testing. Just assign these connections a weight of `0` and Cribl Stream will ignore them. When you want to use such a "parked" connection, assign it a weight greater than `0`.

The final column provides an `X` button to delete any row from the **Destinations** table.

For details on configuring all load balancing settings, see About Load Balancing.

Enabling Cluster Manager Authentication

To enable token authentication on the Splunk cluster manager, you can find complete instructions in Splunk's Enable or Disable Token Authentication documentation. This option requires Splunk 7.3.0 or higher, and requires the following capabilites: `list_indexer_cluster` and `list_indexerdiscovery`.

For details on creating the token, see Splunk's Create Authentication Tokens topic - especially its section on how to Configure Token Expiry and "Not Before" Settings.

Be sure to give the token an **Expiration** setting well in the future, whether you use **Relative Time** or **Absolute Time**. Otherwise, the token will inherit Splunk's default expiration time of `+30d` (30 days in the future), which will cause indexer discovery to fail.

If you have a failover site configured on Splunk's cluster manager, Cribl respects this configuration, and forwards the data to the failover site in case of site failure.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

TLS Settings (Client Side)

Enabled: Defaults to toggled off. When toggled on:

Validate server certs: Reject any server (indexer) certificate that is not authorized by a CA in the **CA certificate path** or by another trusted CA (such as the CA for your system). Defaults to toggled on.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a hostname, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key. Often the value of the `sslPassword` or similar parameter in the `outputs.conf` or `server.conf` file.

Single PEM File

If you have a **single** `.pem` file containing `cacert`, `key`, and `cert` sections, enter this file's path in all of these fields above: **CA certificate path**, **Private key path (mutual auth)**, and **Certificate path (mutual auth)**.

Timeout Settings

- **Connection timeout**: Amount of time (in milliseconds) to wait for the connection to establish, before retrying. Defaults to `10000` ms.
- **Write timeout**: Amount of time (in milliseconds) to wait for a write to complete, before assuming connection is dead. Defaults to `60000` ms.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Output multiple metrics: Toggle on to output multiple-measurement metric data points. (Supported in Splunk 8.0 and above, this format enables sending multiple metrics in a single event, improving the efficiency of your Splunk capacity.)

Minimize in-flight data loss: If toggled on (default), Cribl Stream will check whether the indexer is shutting down, and if so, will stop sending data. This helps minimize data loss during shutdown. (Note that Splunk logs will indicate that the Cribl app has set `UseAck` to `true`, even though Cribl does not enable full `UseAck` behavior.) If toggled off, exposes the following alternative option:

Failed health check limit: Displayed (and set to `1` by default) only if **Minimize in-flight data loss** is disabled. This option sends periodic requests to Splunk once per minute, to verify that the Splunk endpoint is still alive and can receive data. Its value governs how many failed requests Cribl Stream will allow before closing this connection.

A low threshold value improves connections' resilience, but by proliferating connections, this can complicate troubleshooting. Set to `0` to disable health checks entirely - here, if the connection to Splunk

> is forcibly closed, you risk some data loss.

Max S2S version: The highest version of the Splunk-to-Splunk protocol to expose during handshake. Defaults to `v4`; `v3` is also available.

DNS resolution period (seconds): Re-resolve any hostnames after each interval of this many seconds, and pick up destinations from A records. Defaults to `600` seconds.

Load balance stats period (seconds): How far back in time to keep traffic stats for load balancing purposes. Defaults to `300` seconds (five minutes).

Connection limit: Constrains the number of concurrent indexer connections, per Worker Process, to limit memory utilization. If set to a number > `0`, then on every DNS resolution period (or indexer discovery), Cribl Stream will randomly select this subset of discovered IPs to connect to. Cribl Stream will rotate IPs in future resolution periods - monitoring weight and historical data, to ensure fair load balancing of events among IPs.

Nested field serialization: Specifies whether and how to serialize nested fields into index-time fields. Select `None` (the default) or `JSON`.

Authentication method: Use the buttons to select one of these options:

- **Manual**: In the resulting **Auth token** field, enter the shared secret token to use when establishing a connection to a Splunk indexer.
- **Secret**: This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the auth token described above. A **Create** link is available to store a new, reusable secret.

Compression: This setting controls compression for Splunk-to-Splunk `v4` data. Select the **Compression** type from the drop-down list:

- `Disabled`: Compression is turned off. This is the default setting for both existing and new Destinations in Cribl Stream versions 4.11 and older. Connections will be closed if the server attempts to use compression.
- `Always`: Cribl Stream will always send compressed data, regardless of the receiver's compression configuration.

This setting is not available when **Max S2S version** is set to `v3`.

Log failed requests to disk: Toggle on to make the payload of the most recently failed request available for inspection. See Inspect Payload to Troubleshoot Closed Connections below.

Endpoint health fluctuation time allowance (ms): How long (in milliseconds) each receiver endpoint can report blocked, before this Destination as a whole reports unhealthy, blocking senders. (Grace period for

transient fluctuations.) Use `0` to disable the allowance; default is `100` ms; maximum allowed value is `60000` ms (1 minute).

Throttling: Throttle rate, in bytes per second. Multiple byte units such as KB, MB, GB, and so forth, are also allowed - for example, `42 MB`. A value of `0` (the default) indicates no throttling. When throttling is engaged, excess data will be dropped only if **Backpressure behavior** is set to **Drop events**. (Data will be blocked for all other **Backpressure behavior** settings.)

- The throttling rate is applied per Worker Process, per configured endpoint. For example, if you have two Worker Processes, each with two endpoints, and you set the throttling rate to `3 MB`, the total potential throughput rate is 12 MB per second: up to 3 MB per second for each of the four endpoints. The **Connection limit** setting is also applied per Worker Process. To continue the example, if you set **Connection limit** to `1`, the total potential throughput is 6 MB per second, even though each Worker Process has two configured endpoints.
- The throttling rate is applied after compression.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Inspect Payload to Troubleshoot Closed Connections

When a downstream receiver closes connections from this Destination (or just stops responding), inspecting the payload of the most recently failed request can help you find the cause. For example:

- Suppose you send an event whose size is larger than the downstream receiver can handle.
- Suppose you send an event that has a `number` field, but the value exceeds the highest number that the downstream receiver can handle.

When **Log failed requests to disk** is enabled, you can inspect the last failed request payload. Here is how:

1. In the Destination UI, navigate to the **Logs** tab.
2. Find a log entry with a `connection error` message.
3. Expand the log entry.
4. If the message includes the phrase `See payload file for more info`, note the path in the `file` field on the next line.

Now you have the path to the directory where Cribl Stream is storing the payload from the last failed request.

SSL Configuration for Splunk Cloud

To connect to Splunk Cloud, you will need to extract the private and public keys from the Splunk-provided Splunk Cloud Universal Forwarder credentials package. You will also need to reference the CA Certificate located in the same package.

You can reuse many of the settings in this Splunk Cloud package to set up Splunk Cloud Destinations. Use the following steps:

Step 1. Extract the `splunkclouduf.spl` package on the Cribl Stream instance that you will be connecting to Splunk Cloud. You will have a folder that looks something like this:

```
100_my-splunk-cloud_splunkcloud
        /default/
                outputs.conf
                limits.conf
                your-splunk-cloud_server.pem
                your-splunk-cloud_cacert.pem
```

Copy

Step 2. (optional) Test connectivity to Splunk Cloud, using the Root CA certificate:

```
echo | openssl s_client -CAfile 100_<your-splunk-cloud>_splunkcloud/default/my-splunk
```

Copy

To test the connection, you can use any of the URLs listed in the `[tcpout:splunkcloud]` stanza's `outputs.conf` section.

You can simplify Steps 3 and 4 below by dragging and dropping (or uploading) the `.pem` files into Cribl Stream's **New Certificates** modal. See Configure TLS for API and UI Access.

Step 3. Extract the private key from the Splunk Cloud certificate. At the prompt, you will need the `sslPassword` value from the `outputs.conf`. Using Elliptic Curve keys:

```
openssl ec -in 100_<your-splunk-cloud>_splunkcloud/default/<your-splunk-cloud>_server
```

Copy

If you are using RSA keys, instead use:

```
openssl rsa -in 100_<your-splunk-cloud>_splunkcloud/default/<your-splunk-cloud>_serve
```

Copy

Step 4. Extract the public key for the Server Certificate:

```
openssl x509 -in 100_<your-splunk-cloud>_splunkcloud/default/<your-splunk-cloud>_serv
```

Copy

Step 5. In the Splunk Load Balanced Destination's TLS Settings (Client Side) section, enter the following:

- CA Certificate Path: Path to `<your-splunk-cloud>_cacert.pem`.
- Private Key Path (mutual auth): Path to `private.pem` (Step 3 above).
- Certificate Path (mutual auth): Path to `server.pem` (Step 4 above).

In a distributed deployment, enter this Destination configuration on each Worker Group/Fleet that forwards to Splunk Cloud. Then commit and deploy your changes.

Step 6. In a distributed deployment, enable Worker UI access, and verify that the Certificate files have been distributed to individual workers. If they are not present, copy the Certificate files to the Workers, using exactly the same paths you used at the Group level.

Notes About Forwarding to Splunk

- Data sent to Splunk is **not** compressed.
- The only `ack` from indexers that Cribl Stream listens for and acts upon is the shutdown signal described in Minimize in-flight data loss above.
- If events have a Cribl Stream internal field called `__criblMetrics`, they'll be forwarded to Splunk as metric events.
- If events do not have a `_raw` field, they'll be serialized to JSON prior to sending to Splunk.
- You can copy and paste the Splunk Cloud servers from the `[tcpout:splunkcloud]` stanza into **General Settings** > **Destinations**. For example, from the example stanza below, you would copy only the bolded contents:

  ~~[tcpout:splunkcloud]~~
  ~~server =~~ **inputs1.your-splunk-cloud.splunkcloud.com:9997, inputs2.your-splunk-cloud.splunkcloud.com:9997, inputs3.your-splunk-cloud.splunkcloud.com:9997, inputs4.your-splunk-cloud.splunkcloud.com:9997, inputs5.your-splunk-cloud.splunkcloud.com:9997, inputs6.your-splunk-cloud.splunkcloud.com:9997, inputs7.your-splunk-cloud.splunkcloud.com:9997, inputs8.your-splunk-cloud.splunkcloud.com:9997, inputs9.your-splunk-cloud.splunkcloud.com:9997, inputs10.your-splunk-cloud.splunkcloud.com:9997, inputs11.your-splunk-cloud.splunkcloud.com:9997, inputs12.your-splunk-cloud.splunkcloud.com:9997, inputs13.your-splunk-cloud.splunkcloud.com:9997,**

inputs14.your-splunk-cloud.splunkcloud.com:9997, inputs15.your-splunk-cloud.splunkcloud.com:9997
~~compressed = false~~

- From `limits.conf`, copy the `[thruput]` value, then paste it into **Advanced Settings** > **Throttling**.
- If you enable both (1) indexer discovery and (2) TLS with certificate validation, you see might errors. This happens because the IPs discovered (by default) for the indexers will not match the fully qualified domain names (FQDNs) referenced by the indexer certificates. This causes the TLS handshake to fail.
    - As a workaround, use `server.conf` on each indexer, setting `register_forwarder_address = <your.idx.fqdn>`. Cribl Stream will now get that value, and the certificates will match.
- See Splunk's documentation on editing `fields.conf` to ensure the visibility of index-time fields sent to Splunk by Cribl Stream.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Cribl University's Advanced Troubleshooting short courses include Destination Integrations: Splunk LB

8.2.12. Internal

You can send data from Cribl Stream to the following internal receivers:

Destination	Description
Cribl HTTP	Send data between peer Nodes via HTTP
Cribl TCP	Send data between peer Nodes via raw TCP
Cribl Search	Send data to Cribl Search via HTTP
Default	Specify a default output from among your Destinations
DevNull	Drop events when testing Pipelines and Routes
Output Router	Routes data to Destinations based on defined rules

8.2.12.1. Cribl HTTP Destination

The Cribl HTTP Destination sends data to a Cribl HTTP Source. Relaying data between a Cribl HTTP Destination and Cribl HTTP Source pair prevents double-billing: You won't incur any additional charges or credit usage after initial ingest to the first Worker Group (depending on your Leader setup - see Transfer Data Between Workspaces or Environments for more details).

It's common to send data from Cribl Edge to a Cribl Stream Worker Group using this kind of pairing.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

For guidance on when to choose this Source versus the Cribl TCP Destination, see Interoperation Protocols.

Within Cribl Stream, the Cribl HTTP Destination is available only in Distributed deployments. In single-instance mode, or for testing, you can use the Webhook Destination instead. However, this substitution will not facilitate sending all internal fields, as described below.

How It Works

You can use the Cribl HTTP Destination to transfer data between Worker Nodes and Workers. If the Cribl HTTP Destination sends data to its Cribl HTTP Source counterpart on another Worker, you're billed for ingress only once - when Cribl first receives the data.

Using a Cribl HTTP Destination/Source pair, you're not charged for subsequent data sent to:

- Workers that share a Leader.
- Workers that don't share a Leader, but are configured in Cribl.Cloud Workspaces within the same Organization.
- Workers that do not share a Leader, but that do have the exact same on-prem license installed.

This use case is common in hybrid Cribl.Cloud deployments, where a customer-managed (hybrid) Worker Node sends data to a Worker in Cribl.Cloud for additional processing and routing to Destinations. However, the Cribl HTTP Destination/Source pair can similarly reduce your metered data ingress in other scenarios, such as on-prem Edge to on-prem Stream.

As one usage example, assume that you want to send data from one Node deployed on-prem, to another that is deployed in Cribl.Cloud. You could do the following:

- Create an on-prem File System Collector (or whatever Collector or Source is suitable) for the data you want to send to Cribl.Cloud.

- Create an on-prem Cribl HTTP Destination.
- Create a Cribl HTTP Source, on the target Stream Worker Group or Edge Fleet in Cribl.Cloud.
- For an on-prem Node configure a File System Collector to send data to the Cribl HTTP Destination, and from there to the Cribl HTTP Source in Cribl.Cloud.
- On Cribl-managed Nodes in Cribl.Cloud, make sure that TLS is either disabled on both the Cribl HTTP Destination and the Cribl HTTP Source it's sending data to, or enabled on both. Otherwise, no data will flow. On Cribl.Cloud instances, the Cribl HTTP Source ships with TLS enabled by default.

Configuration Requirements

The key points about configuring this architecture are:

- This Destination's **Cribl endpoint** field must point to the **Address** and **Port** you've configured on its peer Cribl HTTP Source(s).
- Cribl 3.5.4 was a breakpoint in Cribl HTTP Leader/Worker communications. Nodes running the Cribl HTTP Destination on Cribl 3.5.4 and newer can receive data only from Nodes running v.3.5.4 and newer. Nodes running the Cribl HTTP Destination on Cribl 3.5.3 and older can receive data only from Nodes running v.3.5.3 and older.

Configure a Cribl HTTP Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Load balancing**: When toggled off (default), if you notice that Cribl Stream is not sending data to all possible IP addresses, enable **Advanced Settings** > **Round-robin DNS**. When toggled on, see Load Balancing Settings below.
    - **Cribl endpoint**: URL of a Cribl Worker to send events to, for example, `http://localhost:10200`.

The **Cribl endpoint** field appears only when you toggle **Load balancing** off. Its value must point to the **Address** and **Port** you've configured on the peer Cribl HTTP Source to which you're sending.

3. Next, you can configure the following **Optional Settings**:
    - **Compression**: Codec to use to compress the data before sending. Defaults to `Gzip`.
    - **Throttling**: Throttle rate per Worker Process, in bytes per second. Defaults to `0` in Cribl Stream, meaning no throttling, and to `256 KB` in Cribl Edge. Multiple-byte units such as KB, MB, GB, and so forth, are also allowed - for example, `42 MB`. When throttling is engaged, your **Backpressure behavior** selection determines whether Cribl Stream will handle excess data by blocking it, dropping it, or queueing it to disk. You can use this setting to limit the risk of overwhelming your deployment when sending high volumes of data from Cribl Edge to Cribl Stream.
    - **Backpressure behavior**: Specifies whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`. See Persistent Queue Settings below.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, configure any TLS settings, Persistent Queue, Retries, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Load Balancing Settings

Enabling the **Load balancing** toggle displays the following controls.

Exclude Current Host IPs: Displays under **Optional Setting** when **Load balancing** is toggled on. This toggle determines whether to exclude all IPs of the current host from the list of any resolved hostnames. Defaults to No, which keeps the current host available for load balancing.

Cribl Worker Endpoints: In this table, you specify a set of Cribl Workers on which to load-balance data. To specify more Workers on new rows, click **Add Endpoint**. Each row provides the following fields.

Cribl Endpoint: Enter the URL of a Worker to send events to.

Must point to the **Address** and **Port** configured on a peer Cribl HTTP Source to which you're sending.

Load weight: Set the relative traffic-handling capability for each connection by assigning a weight (> `0`). This column accepts arbitrary values, but for best results, assign weights in the same order of magnitude to all connections. Cribl Stream will attempt to distribute traffic to the connections according to their relative weights.

The final column provides an `X` button to delete any row from the table.

For details on configuring all these options, see About Load Balancing.

TLS Settings (Client Side)

Use TLS: Defaults to toggled off. When toggled on:

Autofill?: This setting is experimental.

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Defaults to toggled on. Overrides the **Validate server certs** setting in **Advanced Settings**.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)

- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

> If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.

- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an

exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Auth Token

This section is only available for on-prem environments that have a Connected Environment. It is intended for use when migrating data from an on-prem environment to a Cribl.Cloud environment.

If you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from on-prem Worker Nodes to their counterpart Cribl.Cloud Worker Nodes. See Migrate Data from On-Prem to Cribl.Cloud for implementation details.

Advanced Settings

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Defaults to toggled on. This setting is also configurable in **TLS Settings (Client Side)**. If you enable Client-Side TLS, the validation rules defined there will override this setting.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts. Only displayed when the General Settings tab's **Load balancing** option is disabled.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Max events per request: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header

names.

Exclude fields: Fields to exclude from the event. This Destination excludes `__kube_*`, `__metadata`, and `__winEvent` by default and forwards all other Internal Fields. For more information about `__winEvent`, see the Windows Event Logs Source topic.

If you are running Cribl Stream 4.0.x (or earlier) and are using the Kubernetes Metrics Source with this Destination, consider excluding the following fields to reduce event size:

`!__inputId,!__outputId,!__criblMetrics,__*` .

The contents of `__raw` are often redundant with the `_raw` field's contents. Where they are identical, consider excluding one of the two.

Auth Token TTL minutes: The number of minutes before the internally generated authentication token expires, valid values between 1 and 60.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

The following options are added if you enable the General Settings tab's **Load balancing** option:

DNS resolution period (seconds): Re-resolve any hostnames after each interval of this many seconds, and pick up destinations from A records. Defaults to `600` seconds.

Load balance stats period (seconds): Lookback traffic history period. Defaults to `300` seconds. (Note that If multiple receivers are behind a hostname - i.e., multiple A records - all resolved IPs will inherit the weight of the host, unless each IP is specified separately. In Cribl Stream load balancing, IP settings take priority over those from hostnames.)

Internal Fields Loopback to Sources

The Cribl HTTP and Cribl TCP Destinations differ from all other Destinations in the way they handle internal fields: They normally send data back to their respective Cribl Sources - where Cribl internal fields, metrics, and sender-generated fields can all be useful.

These Destinations forward all internal fields by default, except for any that you exclude in **Advanced Settings** > **Exclude fields**.

As examples, if the following fields are present on an event forwarded by a Cribl HTTP or Cribl TCP Destination, they'll be accessible in the ingesting Cribl HTTP/TCP Source: `__criblMetrics`, `__srcIpPort`, `__inputId`, and `__outputId`.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

8.2.12.2. Cribl TCP Destination

The Cribl TCP Destination sends data to a Cribl TCP Source. Relaying data between a Cribl TCP Destination and Cribl TCP Source pair prevents double-billing: You won't incur any additional charges or credit usage after initial ingest to the first Worker Group (depending on your Leader setup - see Transfer Data Between Workspaces or Environments for more details).

It's common to send data from Cribl Edge to a Cribl Stream Worker Group using this kind of pairing.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

For guidance on when to choose this Source versus the Cribl HTTP Destination, see Interoperation Protocols.

Within Cribl Stream, the Cribl TCP Destination is available only in Distributed deployments. In single-instance mode, or for testing, you can use the TCP JSON Destination instead. However, this substitution will not facilitate sending all internal fields, as described below.

How It Works

You can use the Cribl TCP Destination to transfer data between Worker Nodes and Workers. If the Cribl TCP Destination sends data to its Cribl TCP Source counterpart, you're billed for ingress only once - when Cribl first receives the data.

Using a Cribl HTTP Destination/Source pair, you're not charged for subsequent data sent to:

- Workers that share a Leader.
- Workers that don't share a Leader, but are configured in Cribl.Cloud Workspaces within the same Organization.
- Workers that do not share a Leader, but that do have the exact same on-prem license installed.

This use case is common in hybrid Cribl.Cloud deployments, where a customer-managed (hybrid) Worker Node sends data to a Worker in Cribl.Cloud for additional processing and routing to Destinations. However, the Cribl TCP Destination/Source pair can similarly reduce your metered data ingress in other scenarios, such as on-prem Cribl Edge to on-prem Cribl Stream.

As one usage example, assume that you want to send data from one Node deployed on-prem, to another that is deployed in Cribl.Cloud. You could do the following:

- Create an on-prem File System Collector (or whatever Collector or Source is suitable) for the data you want to send to Cribl.Cloud.

- Create an on-prem Cribl TCP Destination.
- Create a Cribl TCP Source, on the target Cribl Stream Worker Group or Cribl Edge Fleet in Cribl.Cloud.
- For an on-prem Node configure a File System Collector to send data to the Cribl TCP Destination, and from there to the Cribl TCP Source in Cribl.Cloud.
- On Cribl-managed Nodes in Cribl.Cloud, make sure that TLS is either disabled on both the Cribl TCP Destination and the Cribl TCP Source it's sending data to, or enabled on both. Otherwise, no data will flow. On Cribl.Cloud instances, the Cribl TCP Source ships with TLS enabled by default.

Configuration Requirements

The key points about configuring this architecture are:

- When you configure the Cribl TCP Destination, its **Address** and **Port** values must point to the **Address** and **Port** you've configured on the Cribl TCP Source.
- Cribl 3.5.4 was a breakpoint in Cribl TCP Leader/Worker communications. Nodes running the Cribl TCP Source on Cribl 3.5.4 and newer can send data only to Nodes running v.3.5.4 and newer. Nodes running the Cribl TCP Source on Cribl 3.5.3 and older can send data only to Nodes running v.3.5.3 and older.

Configure a Cribl TCP Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Load balancing**: When toggled on, see Load Balancing Settings below.

      The following two fields appear **only** if **Load balancing** is toggled off (default), and must match the **Address** and **Port** you've configured on the peer Cribl TCP Source to which you're sending.
    - **Address**: Hostname of the receiver.

- **Port**: Port number to connect to on the host, for example, `10300`.

3. Next, you can configure the following **Optional Settings**:
   - **Compression**: Codec to use to compress the data before sending. Defaults to `Gzip`.
   - **Throttling**: Throttle rate, in bytes per second. Defaults to `0`, meaning no throttling. Multiple-byte units such as KB, MB, and so forth, are also allowed - for example, `42 MB`. When throttling is engaged, your **Backpressure behavior** selection determines whether Cribl Stream will handle excess data by blocking it, dropping it, or queueing it to disk. When used in conjunction with **Connection limit**, the throttling rate is applied per connection. For instance, if **Connection limit** is set to `2` and throttling is set to `3 MB`, each of the two connections can send data at a rate of up to 3 MB per second, resulting in a total potential throughput of 6 MB per second across both connections.
   - **Backpressure behavior**: Specifies whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`. See Persistent Queue Settings below.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, configure any TLS settings, Persistent Queue, Timeout, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Load Balancing Settings

Enabling the **Load balancing** toggle displays the following controls:

Exclude Current Host IPs

Displays under **Optional Setting** when **Load balancing** is toggled on. This toggle determines whether to exclude all IPs of the current host from the list of any resolved hostnames. Default is toggled off, which keeps the current host available for load balancing.

Destinations

The **Destinations** table is where you specify a set of receivers on which to load-balance data. Click **Add Destination** to specify more receivers on new rows. Each row provides the following fields:

Address: Hostname of a receiver. Optionally, you can paste in a comma-separated list, in `<host>:<port>` format.

Port: Port number to send data to on the host.

Each **Address**/**Port** combination must match the **Address** and **Port** configured on a peer TCP Source to which you're sending.

TLS: Whether to inherit TLS configs from group setting, or disable TLS. Defaults to `inherit`.

TLS servername: Servername to use if establishing a TLS connection. If not specified, defaults to connection host (if not an IP); otherwise, uses the global TLS settings.

Load weight: Set the relative traffic-handling capability for each connection by assigning a weight (> `0`). This column accepts arbitrary values, but for best results, assign weights in the same order of magnitude to all connections. Cribl Stream will attempt to distribute traffic to the connections according to their relative weights.

The final column provides an `X` button to delete any row from the table.

For details on configuring all these options, see About Load Balancing.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10

seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

TLS Settings (Client Side)

Use TLS: Defaults to toggled off. When toggled on:

Autofill?: This setting is experimental.

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Default is toggled off.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate name: The name of the predefined certificate.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Timeout Settings

Connection timeout: Amount of time (in milliseconds) to wait for the connection to establish before retrying. Defaults to `10000`.

Write timeout: Amount of time (in milliseconds) to wait for a write to complete before assuming connection is dead. Defaults to `60000`.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Auth Token

This section is only available for on-prem environments that have a Connected Environment. It is intended for use when migrating data from an on-prem environment to a Cribl.Cloud environment.

If you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from on-prem Worker Nodes to their counterpart Cribl.Cloud Worker Nodes. See Migrate Data from On-Prem to Cribl.Cloud for implementation details.

Advanced Settings

Auth Token TTL minutes: The number of minutes before the internally generated authentication token expires, valid values between 1 and 60.

Exclude fields: Fields to exclude from the event. This Destination excludes `__kube_*`, `__metadata`, and `__winEvent` by default and forwards all other Internal Fields. For more information about `__winEvent`, see the Windows Event Logs Source topic.

If you are running Cribl Stream 4.0.x (or earlier) and are using the Kubernetes Metrics Source with this Destination, consider excluding the following fields to reduce event size:

`!__inputId,!__outputId,!__criblMetrics,__* .`

The contents of `__raw` are often redundant with the `_raw` field's contents. Where they are identical, consider excluding one of the two.

Log failed requests to disk: Toggling on makes the payload of the most recently failed request available for inspection. See Inspect Payload to Troubleshoot Closed Connections below.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

The following options are added if you enable the General Settings tab's **Load balancing** option:

DNS resolution period (seconds): Re-resolve any hostnames after each interval of this many seconds, and pick up destinations from A records. Defaults to `600` seconds.

Load balance stats period (seconds): Lookback traffic history period. Defaults to `300` seconds. (Note that If multiple receivers are behind a hostname - i.e., multiple A records - all resolved IPs will inherit the weight of the host, unless each IP is specified separately. In Cribl Stream load balancing, IP settings take priority over those from hostnames.)

Max connections: Constrains the number of concurrent receiver connections, per Worker Process, to limit memory utilization. If set to a number > `0`, then on every DNS resolution period, Cribl Stream will randomly select this subset of discovered IPs to connect to. Cribl Stream will rotate IPs in future resolution periods - monitoring weight and historical data, to ensure fair load balancing of events among IPs.

Inspect Payload to Troubleshoot Closed Connections

When a downstream receiver closes connections from this Destination (or just stops responding), inspecting the payload of the most recently failed request can help you find the cause. For example:

- Suppose you send an event whose size is larger than the downstream receiver can handle.
- Suppose you send an event that has a `number` field, but the value exceeds the highest number that the downstream receiver can handle.

When **Log failed requests to disk** is enabled, you can inspect the last failed request payload. Here is how:

1. In the Destination UI, navigate to the **Logs** tab.
2. Find a log entry with a `connection error` message.
3. Expand the log entry.

4. If the message includes the phrase `See payload file for more info`, note the path in the `file` field on the next line.

Now you have the path to the directory where Cribl Stream is storing the payload from the last failed request.

Internal Fields Loopback to Sources

The Cribl TCP and Cribl HTTP Destinations differ from all other Destinations in the way they handle internal fields: They normally send data back to their respective Cribl Sources - where Cribl internal fields, metrics, and sender-generated fields can all be useful.

These Destinations forward all internal fields by default, except for any that you exclude in **Advanced Settings** > **Exclude fields**.

As examples, if the following fields are present on an event forwarded by a Cribl HTTP or Cribl TCP Destination,

8.2.12.3. Cribl Search Destination

The Cribl Search Destination enables Cribl Stream to send events directly to Cribl Search over a dedicated, internal HTTP path.

Use the Cribl Search Destination when you want to send data from:

- Worker Groups to Cribl Search running in the same Cribl.Cloud organization.
- Worker Groups to a connected Cribl.Cloud environment that hosts Cribl Search.
- Worker Groups to Cribl Search as an ingestion endpoint, instead of configuring generic Cribl HTTP Destinations by hand.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

For Cribl Stream Worker Groups that are managed in Cribl.Cloud and sending to Cribl Search in the same Organization, TLS for Cribl-to-Cribl traffic is managed automatically. You do not configure TLS settings directly on this Destination.

How It Works

With the Cribl Search Destination:

- The Destination serializes events using the same HTTP payload format as Cribl HTTP.
- Events are delivered over HTTP to a Cribl HTTP Source in the Cribl.Cloud environment that hosts Cribl Search. That Source is preconfigured to forward all received events into Cribl Search's ingestion pipeline.
    - That Cribl HTTP Source typically runs in a different Worker Group than the one where you configured the Cribl Search Destination, and it serves as the HTTP entry point for Cribl Search ingestion.
- Cribl Search indexes the events according to its own dataset and schema configuration.

Data Transfer Behavior

You can use the Cribl Search Destination to transfer data between Worker Groups and Cribl Search without incurring additional ingest charges, as long as it sends data to a Cribl HTTP Source that acts as the entry point for Cribl Search ingestion, and your deployment meets the eligibility rules in Transfer Data Between Workspaces or Environments.

If the Cribl Search Destination sends data to its Cribl HTTP Source counterpart for Search on another Worker Node, you're billed for ingress only once when Cribl first receives the data. Using a Cribl Search Destination and

Cribl HTTP Source pair, you're not charged for subsequent data sent to:

- Worker Nodes that share a Leader.
- Worker Nodes that don't share a Leader but are configured in Cribl.Cloud Workspaces within the same Organization.
- Worker Nodes that do not share a Leader but that have the exact same on-prem license installed.

Prerequisites

Before configuring the Cribl Search Destination:

- You must have Cribl Search enabled in a Cribl.Cloud organization.
- For cross-environment or cross-Workspace paths (for example, on-prem Stream to Cloud Search), both environments must be on paid plans and eligible for free data transfer via Cribl TCP/HTTP.
    - For those paths, configure Cribl-to-Cribl authentication using Auth Tokens on both the Cribl HTTP Source in Cribl.Cloud that receives events for Cribl Search, and on the on-prem Cribl Search Destination as described in Transfer Data Between Workspaces or Environments.

For Worker Groups running in Cribl.Cloud and sending to Cribl Search in the same Organization, the Cribl Search Destination uses the existing Cribl-to-Cribl authentication path automatically. No additional tokens are required.

Configuration Requirements

The Cribl Search Destination's **Cribl endpoint** must point to the **Address** and **Port** configured on the corresponding Cribl HTTP Source that serves as the HTTP entry point for Cribl Search in the target Worker Group.

Configure a Cribl Search Destination

1. On the top bar, select **Products**, and then select Cribl Stream. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** (Stream) or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination modal**, configure the following under **General Settings**:

- **Output ID**: Enter a unique name to identify this Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original Output ID.
- **Description**: Optionally, enter a description.
- **Cribl endpoint**: URL of the Cribl HTTP Source that receives events for Cribl Search. In Cribl.Cloud this is auto-populated as `https://local_search.<workspace>.<organizationId>.cribl.cloud:10200`. In self-managed or connected environments, use the host and port from your Cribl HTTP Source.

3. Next, you can configure the following **Optional Settings**:
   - **Throttling**: Throttle rate per Worker Process, in bytes per second. Defaults to `0`, meaning no throttling. Multiple-byte units such as KB, MB, GB, and so forth, are also allowed (for example, 42 MB). When throttling is engaged, your backpressure behavior selection determines whether Cribl Stream will handle excess data by blocking it, dropping it, or queueing it to disk.
   - **Backpressure behavior**: Specifies whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to **Block**. See Persistent Queue Settings below.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the Destinations page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, configure any Processing Settings, Auth Tokens, and Advanced Settings settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

> If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new

requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Auth Tokens

This section is only available for on-prem environments that have a Connected Environment. It is intended for use when migrating data from an on-prem environment to a Cribl.Cloud environment.

If you previously had an on-prem deployment of Cribl and you are moving forward with a Cloud-first or Cloud only solution, you can use connected environments to onboard your data from on-prem Worker Nodes to their counterpart Cribl.Cloud Worker Nodes. See Migrate Data from On-Prem to Cribl.Cloud for implementation details.

Advanced Settings

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Internal Fields Loopback to Sources

Because the Cribl Search Destination is implemented as a Cribl HTTP Destination, it shares the same internal-fields behavior as the Cribl HTTP and Cribl TCP Destinations. These Destinations normally send data back to their respective Cribl Sources, where Cribl internal fields, metrics, and sender-generated fields can be useful. They forward all internal fields by default, except for any that you exclude in **Advanced Settings** > **Exclude fields**. For example, if `__criblMetrics`, `__srcIpPort`, `__inputId`, or `__outputId` are present on an event forwarded by the Cribl Search Destination, they'll be accessible in the ingesting Cribl HTTP Source.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

8.2.12.4. Default Destination

The **Default** Destination simply enables you to specify a default output from among your already-configured Destinations.

Type: Internal | TLS Support: N/A | PQ Support: N/A

Configure Cribl Stream's Default Destination

In Cribl Stream configure Default.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). A **Default** Destination is preconfigured and ready to use in the right **Destinations** column. Hover over the tile and click its **Configure** button to proceed.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. From the resulting **Destinations** page, click anywhere on the `default` row to proceed.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: This is prefilled with the default value `default`, which cannot be changed via the UI.
    - **Default Output ID** drop-down to select one of your configured Destinations.
3. Select **Save**, then **Commit & Deploy**.

Preventing Circular References

If you've configured an Output Router Destination with a branch that points to this Default Destination (`default:default`), you cannot select that Output Router here. This restriction prevents a circular dependency.

8.2.12.5. DevNull Destination

The DevNull Destination simply drops events. Cribl provides this as a basic output to test Pipelines and Routes.

Type: Internal | TLS Support: N/A | PQ Support: N/A

Configure Cribl Stream to Forward to DevNull

DevNull requires no configuration: A DevNull Destination is preconfigured and active as soon as you install Cribl Stream. To verify this, follow the direction below.

On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:

- To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). A **DevNull** Destination is preconfigured and ready to use in the right **Destinations** column.
- To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. From the resulting **Destinations** page, click anywhere on the `devnull` row to proceed.

8.2.12.6. Output Router Destination

The Output Router Destination enables dynamic, rule-based routing of events to one or more downstream Destinations. It acts as a meta-destination, evaluating each event against an ordered set of filter rules (JavaScript expressions) to determine the appropriate Destinations (outputs).

This is useful where routing decisions depend on event content, such as sending security logs to a SIEM and application logs to cloud storage, all from a single Pipeline.

How It Fits Into the Data Flow

- **Source ingestion**: Events are ingested from Sources.
- **Pre-processing Pipelines**: Pre-processing Pipelines attached to Sources run first. Fields created or modified here are available for routing and Output Router rule evaluation.
- **Route Pipelines**: If configured, regular Pipelines attached to Routes process events next. Fields set here are also available to Output Router rules.
- **Output Router**: The Output Router evaluates its rules (**Filter expressions**) against each event using fields from pre-processing and Route Pipelines. Matching rules determine which Destinations receive the event. If multiple rules match and final is `false`, the event is cloned and sent to all matching Destinations.

  Events that do not match a rule are sent to the Default Destination.
- **Destination(s)**: Events are delivered to the selected Destinations.
- **Post-processing Pipelines**: Post-processing Pipelines attached to Destinations run last, after routing. Fields created or modified here are not available to Output Router rules.

Type: Internal | TLS Support: N/A | PQ Support: N/A

Configure an Output Router Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
   - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.

- To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.

2. In the **New Destination** modal, configure the following under **General Settings**:
   - **Router name**: Enter a unique name to identify this Router definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
   - **Description**: Optionally, enter a description.
   - **Rules**: A list of event routing rules. Rules are evaluated in order, top-down. For details, see Rules.
3. You can configure the following **Optional Settings**:
   - **System fields**: A list of fields to automatically add to events that use this Destination. For details, see System Fields.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. On the **Advanced Settings** tab you can configure an **Environment** if you're using GitOps. Optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.
5. Select **Save**, then **Commit & Deploy**.

Rules

Rules route events by examining event content. These rules are processed sequentially from top to bottom.

Filter expression: A JavaScript expression to select events to route to Destinations. Filter expressions evaluate event fields after pre-processing (input conditioning at the Source) and after any Route Pipeline processing, but before any post-processing (output conditioning at the Destination). Fields created or modified in pre-processing or Route Pipelines are available to Output Router filter expressions. Fields created or modified during post-processing are not available, as post-processing occurs after routing decisions are made.

Use the expression `true` to match any remaining events you want sent, like in this example.

Output: The Destination to send matching events to. This can be any configured Destination, except for another Output Router or a Default Destination that points back to an Output Router.

Description: Optionally, enter a description of this rule's purpose.

Final: By default (toggled on), event routing stops after this rule matches. Toggle off if you want the event to be evaluated against other rules lower in the stack, allowing cloning to multiple Destinations.

System Fields

By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Example

Scenario:

- Send all events where `host` starts with `66` to Destination `San Francisco`.
- From the rest of the events: Send all events with `method` field `POST` or `GET` to both `Seattle` and `Los Angeles` (i.e., clone them).
- Send the remaining events to `New York City`.

Router Name: **router66**

Filter Expression	Output	Final
`host.startsWith('66')`	**San Francisco**	Yes
`method=='POST' || method=='GET`	**Seattle**	No
`method=='POST' || method=='GET'`	**Los Angeles**	Yes
`true`	**New York**	Yes

8.2.13. CrowdStrike Falcon LogScale Destination

The **CrowdStrike Falcon LogScale** Destination can stream data to a LogScale HEC (HTTP Event Collector) in JSON or Raw format.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

(In Cribl Stream 3.5.x, this Destination was labeled **Humio HEC**. Some links from this page might still lead to "Humio"-branded resources that CrowdStrike has not renamed.)

Recommendations

To load-balance to customer-managed LogScale receivers, Cribl recommends placing a load balancer in front of cluster nodes, following LogScale Manual Cluster Deployment recommendations. If you are using LogScale Cloud, it provides its own load balancing.

We recommend sending events with the `sourceType` field set to a LogScale parser. This tells LogScale which parser to use to extract fields (for example, `"sourceType":"json"`).

If LogScale cannot match the `sourceType` value to a parser, it will use the `kv` parser, and you will get an error that LogScale could not resolve the specified parser. Alternatively, you can assign a parser to the ingest token that you use to authenticate this Destination.

The `fields` element does not support Nested JSON. Any nested elements will be dropped.

Configure Cribl Stream to Output to CrowdStrike Falcon LogScale Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:

- **Output ID**: Enter a unique name to identify this CrowdStrike Falcon LogScale definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
- **Description**: Optionally, enter a description.
- **LogScale endpoint**: URL of a CrowdStrike Falcon LogScale endpoint to send events to (for example, `https://cloud.us.humio.com:443/api/v1/ingest/hec`).

  JSON-formatted events normally go to `/api/v1/ingest/hec` or `/services/collector`.

  Raw (simple line-delimited) events normally go to `/api/v1/ingest/hec/raw` or `/services/collector/raw`.

- **Request format**: Select how you want events formatted, either `JSON` or `Raw`. Make sure your selection here matches the **LogScale endpoint** you specify above.

3. Under **Authentication**, select an **Authentication method** from the dropdown:
   - **Manual**: Displays a **LogScale Auth token** field for you to enter your CrowdStrike Falcon LogScale HEC API token.
   - **Secret**: Displays a **LogScale token (text secret)** drop-down, in which you can select a stored secret that references the API token described above. A **Create** link is available to store a new, reusable secret.
4. Next, you can configure the following **Optional Settings**:
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Optionally, you can adjust the Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues

- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.

- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an

exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Defaults to toggled on to reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA).

Round-robin DNS: Defaults to toggled on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Defaults to toggled on to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`. Each request can potentially hit a different HEC receiver.

Body size limit (KB): Maximum size, in KB, of the request body. Defaults to `4096`. Lowering the size can potentially result in more parallel requests and also cause outbound requests to be made sooner.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1`.

- Retries happen on this flush interval.
- Any HTTP response code in the `2xx` range is considered success.
- Any response code in the `5xx` range is considered a retryable error, which will not trigger Persistent Queue (PQ) usage.
- Any other response code will trigger PQ (if PQ is configured as the Backpressure behavior).

Extra HTTP headers: Click **Add Header** to add **Name/Value** pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers that you want to declare as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new connection will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If the server does not support keepalives - or if the server closes a pooled connection while idle - a new connection will be established for the next request.
- Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between CrowdStrike Falcon LogScale servers.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.14. ClickHouse Destination

The ClickHouse Destination enables seamless integration with ClickHouse. This Destination routes observability data, including logs, metrics, and traces, to ClickHouse for enhanced monitoring and analysis.

Data is formatted and transmitted via HTTP requests, offering both synchronous and asynchronous insert options for flexibility. Authentication is secured with Basic or SSL User Certificate methods, while robust retry settings ensure reliable data delivery. Flexible data mapping, including automatic and manual field-to-column mapping, guarantees structured ingestion. To optimize performance, the integration supports batching and asynchronous inserts, leveraging ClickHouse's capabilities for efficient data delivery.

To get started, connecting to ClickHouse is simple. Grab your ClickHouse instance's HTTP interface URL, database name, and table name, then add these details to your Destination settings. Need more detailed instructions or configuration options? See below!

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Configure a ClickHouse Destination

In Cribl Stream, set up a ClickHouse Destination.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing**.
    - To configure via the Routes, select **Data** > **Destinations** (Stream) or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the Destination modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**. A **Description** is optional.
    - **URL**: The URL of your ClickHouse instance's HTTP interface. This should include the protocol (http or https) and the port number if it's not the default. Example: `http://localhost:8123`.
    - **ClickHouse database**: Enter the name of the ClickHouse database where the target table resides. Ensure that the database exists and the authenticated user has the necessary permissions to write to it.

- **ClickHouse table**: Enter the name of the target table. The name can contain letters (`A-Z`, `a-z`), numbers (`0-9`), and underscores `_`, and must start with either a letter or underscore. The table must exist and have a schema compatible with the data being sent.
- **Format**: Choose the format for events sent to ClickHouse. Choose between:
  - **JSONCompactEachRowWithNames** (default): Optimized for efficiency by minimizing data size, making it ideal for transmitting data to ClickHouse over networks where bandwidth is a concern.
  - **JSONEachRow**: Standard JSON format with extra spacing, useful for debugging or human inspection. If the time spent reformatting data into the compact format exceeds the cost of transmitting more data, `JSONEachRow` may be preferable. For example, in scenarios where the ClickHouse instance runs on the same machine as Cribl, the network cost is negligible, and the time spent reformatting doesn't provide significant benefits.
- **Authentication type**: ClickHouse supports several authentication methods for its HTTP interface.
  - **None**: Don't use authentication.
  - **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
  - **Basic (credentials secret)**: This option exposes a **Credentials secret** drop-down to select a stored text secret that contains your username and password credentials referenced by a secret. A **Create** link is available to store a new, reusable secret.
  - **SSL User Certificate**: Use to securely connect to ClickHouse. Provide the username for certificate authentication. This method ensures a higher level of security by using SSL certificates to verify the identity of the client.
- **Mapping**: Determines how fields from Cribl events are mapped to ClickHouse columns. Options include `Automatic` and `Custom`.
  - **Automatic**: Uses default mappings.
    - **Exclude fields**: A list of fields to exclude from the data sent to ClickHouse. This can help reduce the size of the data and avoid sending unnecessary information.
  - **Custom**: Manually specify column mappings. Select **Retrieve table columns** to connect to a publicly accessible ClickHouse instance, authenticate using the provided credentials, and fetch the table schema. The retrieved column names and types will be automatically populated in the **Column Mapping** table, allowing you to focus on mapping the appropriate data values.
    - The **Column Mapping** table allows you to define how event data fields are mapped to ClickHouse table columns. This is essential for ensuring that data is correctly ingested into ClickHouse with the appropriate structure and types. See Data Types in ClickHouse for details.

        - **Column name**: The name of the column in the ClickHouse table. This should match the column name defined in the ClickHouse schema.
        - **Column type**: The data type of the column in the ClickHouse table (such as, `String`, `Int32`, `DateTime`). This ensures that the data is stored in the correct format.
        - **Column value**: A JavaScript expression that defines how to extract or transform the event data to fit into the specified column. This allows for flexible data manipulation and transformation before ingestion.
    - **Async**: The **Async inserts** toggle enables asynchronous inserts, allowing ClickHouse to buffer data in memory before writing it to disk. This can improve performance but may risk data loss if the server crashes before the data is written.
        - **Wait for async inserts** waits for the ClickHouse server to confirm that the data has been fully written to disk before proceeding. This ensures a higher level of data integrity by reducing the risk of data loss if the server encounters an issue before full insertion. This setting balances throughput and data assurance, making it suitable for high-volume logging where occasional data loss is acceptable.
    - **Optional Settings**:
        - **Backpressure behavior**: Whether to `Block`, `Drop`, or `Persistent Queue` events when all receivers are exerting backpressure.
        - **Tags**: Optionally, add tags that you can use to filter and group Destinations in Cribl Stream's **Manage Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
3. If you're using TLS, select **TLS Settings (Client Side)** on the left tabs and toggle **Use TLS** on. See the TLS Settings section for details. Default is toggled off.
4. The **Processing Settings**, **Retries**, and **Advanced Settings** provide granular control over how data is handled and transmitted.
    - Processing Settings allow you to define pre-processing and post-processing Pipelines, ensuring that events conform to the relevant Protobuf specifications for traces, metrics, and logs. This includes adding metadata, applying transformations, and conditioning the data before it is sent to the destination.
    - Retries allow you to specify how the system should handle different HTTP response status codes and timeouts, ensuring reliable data delivery.
    - Advanced Settings offer additional configuration options that enhance the flexibility and performance of data transmission.
5. Select **Save**, then **Commit & Deploy**.
6. Use the **Test** tab in the Destination's configuration modal to validate the configuration and connectivity of your Destination.

TLS Settings (Client Side)

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output - both metric events, as dimensions; and log events, as labels. Supports wildcards.

By default, includes `cribl_pipe` (Cribl Stream Pipeline that processed the event).

Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After`

header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. You can set this limit to as high as 500 MB (`512000` KB). Be aware that high values can cause high memory usage per Worker Node, especially if a dynamically constructed URL (see Internal Fields) causes this Destination to send events to more than one URL. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass to all events as additional HTTP headers. Values will be sent encrypted. You can also add headers dynamically on a per-event basis in the `__headers` field. See Internal Fields below.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Log last schema mismatch: Toggle on to enable logging of the most recent event that fails to match the ClickHouse table schema. When enabled, an entry with the message `Failed to serialize event. See payload file for more info.` is written to the system log. This log entry points you to the temporary output file `last-failed-format-event.json`, which contains the data for the most recent failed event.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Troubleshoot

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.15. Cloudflare R2 Destination

The Cloudflare R2 Destination sends data to Cloudflare R2 object storage using an S3-compatible API.

Type: Non-Streaming | TLS Support: Configurable | PQ Support: No

Configure a Cloudflare R2 Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **R2 endpoint**: Cloudflare R2 service URL. For example: `https://<ACCOUNT_ID>.r2.cloudflarestorage.com`.
    - **R2 bucket name**: Name of the R2 bucket. This value can be a constant, or a JavaScript expression that will be evaluated only at init time. For example, referencing a Global Variable: `myBucket-${C.vars.myVar}`. Ensure that the bucket already exists, otherwise R2 will generate "bucket does not exist" errors.

      Event-level variables are not available for JavaScript expressions. This is because the bucket name is evaluated only at Destination initialization. If you want to use event-level variables in file paths specify them in the **Partitioning Expression** field, because this is evaluated for each file.

    - **Staging location**: Filesystem location in which to locally buffer files before compressing and sending. Cribl recommends that this location be stable and high-performance.

      The **Staging location** field is not displayed or available on Cribl.Cloud-managed Worker

> Nodes.

- **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (supported only on Linux, not Windows) exposes a **Parquet Settings** left tab, where you **must** configure certain options in order to export data in Parquet format.

3. Next, you can configure the following **Optional Settings**:
   - **Key prefix**: A constant or JavaScript expression that is prepended to each object key (file name) when writing data to the bucket. The key prefix is evaluated per file and can use global variables or environment variables, but not event-level fields. This enables logical organization of objects within the bucket, simulating directory structures using forward slashes `/`.
   - **Partitioning expression**: JavaScript expression that defines how files are partitioned and organized. Default is date-based. If blank, Cribl Stream will fall back to the event's `__partition` field value (if present); or otherwise to the root directory of the **Output Location** and **Staging Location**.

     > Cribl Stream's internal `__partition` field can be populated in multiple ways. The precedence order is: explicit **Partitioning expression** value -> `${host}/${sourcetype}` (default) **Partitioning expression** value -> user-defined `event.__partition`, set with an Eval Function (takes effect only where this **Partitioning expression** field is blank).

   - **Compression**: Data compression format used before sending. Defaults to `gzip` (recommended). This setting is not available when **Data format** is set to `Parquet`.
   - **File name prefix expression**: The output file name prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
   - **File name suffix expression**: The output file name suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.
   - **Backpressure behavior**: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Processing, Parquet, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Authentication Method** drop-down to select either **Auto** or **Secret**:

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. This option exposes a **Secret key pair** drop-down, in which you can select a stored secret that references the set of user-associated IAM credentials described above. A **Create** link is available to store a new, reusable secret.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.

- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Set the target memory size for page segments. Generally, set lower values to improve reading speed, or set higher values to improve compression. Value must be a positive integer smaller than the **Row group size** value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Leave toggled on (the default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Leave toggled on (the default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to final output location. Defaults to `100`.

Cribl Stream will close files when **either** of the `File size limit (MB)` or the `File open time limit (sec)` conditions is met.

Concurrent file parts limit: Maximum number of parts to upload in parallel per file. A value of `1` tells the Destination to send the whole file at once. When set to `2` or above, IAM permissions must include those required for multipart uploads. Defaults to `4`; highest allowed value is `10`.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined **Min free disk space** amount.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

Remove empty staging dirs: Toggle this on to delete empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove empty staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).

Storage class: Select a storage class for uploaded objects. Defaults to `Standard`.

Server-side encryption: Server side encryption type for uploaded objects. Defaults to `none`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Verify if bucket exists: Toggle off if you can access files within the bucket, but not the bucket itself. This resolves errors of the form: `error initializing...Bucket does not exist`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

IAM Permissions

The following permissions are always needed to write to an Amazon S3-compatible object store:

- `s3:ListBucket`
- `s3:GetBucketLocation`
- `s3:PutObject`

If your Destination needs to do multipart uploads to S3, two more permissions are needed:

- `kms:GenerateDataKey`
- `kms:Decrypt`

See the AWS documentation.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

"Bucket does not exist - self signed certificate"

Example Error Text:

```
message: Bucket does not exist - self signed certificate
stack: Error: self signed certificate
    at TLSSocket.onConnectSecure (node:_tls_wrap_1535:34)
    at TLSSocket.emit (node:events:513:28)
    at TLSSocket.emit (node:events:489:12)
    at TLSSocket._fini... Show more
tmpPath: /opt/cribl/s3_bucket/cribl/2023/03/28/CriblOut-XTNGh5.0.json.tmp
```

Copy

Can occur when:

- the Destination uses a self-signed cert that has not yet been trusted by Cribl Stream.
- the user does not have appropriate permissions to view the bucket.
- the authentication token generated by your computer has a timestamp that is out of sync with the server's time, resulting in an authentication failure.

Recommendation

Disable **Advanced Settings** > **Reject unauthorized certificates** or; get the sender's cert and add it to the `NODE_EXTRA_CA_CERTS` path for validation; or verify that the user has appropriate permissions.

8.2.16. Cortex XSIAM Destination

The **Cortex XSIAM** Destination allows you to seamlessly stream event data to Palo Alto Networks' Cortex XSIAM (Extended Security Intelligence and Automation Management) for advanced threat detection and response.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

XSIAM Data Requirements

Cribl Stream provides a generic **Palo Alto XSIAM** Pack designed to extract and map mandatory fields. Cribl Stream automatically validates and converts these fields into HTTP headers, ensuring compliance with XSIAM requirements. For details on using the Pack, see how to map data to the XSIAM Destination.

Optimized Data Delivery:

- **Automatic Format Parsing**: Detects event formats, including CEF, LEEF, Syslog, JSON, and Raw.
- **Batched Transmission**: To optimize data ingestion and reduce overhead, events are grouped into batches. Set the **Flush period** to control latency. Allowed values are `1` second or greater.
- **Aggregation**: Uses batched keys derived from the event format and system fields.
- **Payload Size**: Enforces a 5 MB limit per individual event and a 9.5 MB limit per batch. Warnings are logged if data exceeds these limits.
- **Rate Limiting**: There's a default request rate limit of `400` requests per second. Customizable with the **Throttle request rate limit** setting to align with your XSIAM capacity.
- **JSON formatting**: Events are serialized into JSON format for XSIAM ingestion.
    - The `Content-Type: application/json` header is included in all requests.
    - The `data` field is represented as a native JSON object, ensuring direct parsing of its contents. This native JSON structure allows XSIAM to efficiently query and analyze the data within the field, for example:

```
{"data": {"resource": "/done", "path": "/done", "isBase64Encoded": false}, "coll
```

Copy

Prerequisite

In XSIAM, create a data collector instance and obtain an authorization token and endpoint URL. See the appropriate Cortex XSIAM documentation below for instructions.

- Cortex XSIAM 2.x
- Cortex XSIAM Premium
- Cortex XSIAM Enterprise
- Cortex XSIAM NG SIEM

Configure an XSIAM Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **XSIAM endpoint**: XSIAM endpoint URL to send events to, following this format `https://api-{tenant external URL}/logs/v1/event`. Defaults to `http://localhost:8088/logs/v1/event`.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Bearer Token**: Displays an **Auth token** field for you to enter your XSIAM authorization token.
    - **Secret**: Displays an **Auth token (text secret)** drop-down to select a stored secret. A **Create** link is available to store a new, reusable secret.
4. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. If you assigned the **Backpressure behavior** to `Persistent Queue` you can adjust the Persistent Queue settings.
6. Optionally, you can adjust the Retries, and Advanced settings outlined in the sections below.

7. Select **Save**.
8. Map data to the XSIAM Destination.
9. **Commit & Deploy** your changes.

Map data to the XSIAM Destination

In Cribl Stream, add the **Palo Alto XSIAM** Pack to your Worker Group. Navigate to **Processing** > **Packs** from the Worker Group submenu, and add the Pack. Once added, you'll be able to assign the Pack to your Route or QuickConnect that maps data from Sources to the XSIAM Destination.

- The Pack includes several Pipelines designed for specific data types.

  Configure the Pack **Route** to use the appropriate **Pipeline** for your data type.

- If you're sending different data types for your XSIAM Destination, create a separate Pack with the appropriate Pipeline for each data type.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10

seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by

manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1`.

- Retries happen on this flush interval.
- Any HTTP response code in the `2xx` range is considered success.
- Any response code in the `5xx` range is considered a retryable error, which will not trigger Persistent Queue (PQ) usage.
- Any other response code will trigger PQ (if PQ is configured as the **Backpressure behavior**).

Throttle request rate limit: Maximum number of HTTP requests sent per second. Defaults to `400`, maximum `2000`. Requests exceeding this limit are stored in the persistent queue until capacity becomes available.

Extra HTTP headers: Click **Add Header** to add **Name/Value** pairs to pass as additional HTTP headers. Values will be sent encrypted.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new connection will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If the server does not support keepalives - or if the server closes a pooled connection while idle - a new connection will be established for the next request.

Troubleshooting

This table outlines common API call errors, showing response codes and log messages for troubleshooting.

Scenario	Response Code	Response Body
Unauthorized access	`401`	N/A (Typically no body for a `401` response)
No Source-Identifier header or an invalid UUID provided in the Source-Identifier header	`400`	`missing or unknown UUID was signaled in Source-Identifier header`
The "Other" (non-XSIAM) integration UUID (`af01292940d7426594d3d3e55ae17ee0`) was signaled in the Source-Identifier but the	`400`	`must signal both vendor and product for non-xsiam integrations`

Scenario	Response Code	Response Body
Vendor and/or Product headers are missing or empty.		
Missing Format header	`400`	`missing format header`
Missing Integration-Identifier header	`400`	`missing Integration-Identifier header`

Configuration Modal

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly

8.2.17. CrowdStrike Falcon Next-Gen SIEM Destination

The **CrowdStrike Falcon Next-Gen SIEM** Destination sends data to CrowdStrike Falcon Next-Gen SIEM.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Prerequisites

Connector

Within CrowdStrike Falcon Next-Gen SIEM's **Data onboarding** section, create a `Cribl Data Connector`. Select `Cribl` as the connector type during setup to leverage future functionalities and enhancements unavailable through generic HTTP Event Collector (HEC) connectors. This also allows you to filter ingested `Data Sources` by `Cribl Data Connector` for streamlined management of Cribl Stream data streams.

SIEM Parser

Each Next-Gen SIEM connector requires an assigned parser. Map each connector to the appropriate parser for its vendor/product combination. If you have multiple connectors for the same vendor/partner, you can reuse the same parser. If no suitable parser exists, create a new one or clone and modify an existing parser with a similar data format.

By default, CrowdStrike Falcon Next-Gen SIEM parsers expect data in the vendor's original format. Ensure you only send the `_raw` field. For standard data ingestion, use the `passthru` parser. This will preserve the original event format sent to Next-Gen SIEM. However, some products, like Zscaler, usually require specific output formats.

See how to test your SIEM parser.

Syslog

When using a Cribl Stream Syslog Source, explicitly enter `_raw` in the **Fields to keep** list under the Syslog Source's **Optional Settings**. This prevents Cribl Stream from automatically parsing the syslog event into multiple fields, which would unnecessarily increase the event size and ensures that the original syslog event is routed from the Source to the CrowdStrike Falcon Next-Gen SIEM Destination.

Configuring a CrowdStrike Falcon Next-Gen SIEM Destination

1. From the top nav, click **Manage**, then select a **Worker Group** to configure. Next, you have two options:

- To configure via QuickConnect, click **Routing** then **QuickConnect** (Cribl Stream) or **Collect** (Cribl Edge). Next, click **Add Destination** and from the resulting drawer's tiles, select **CrowdStrike Falcon Next-Gen SIEM**. Next, click either **Add Destination** or (if displayed) **Select Existing**.
- To configure via the Routes, click **Data** then **Destinations** (Cribl Stream) or **More** then **Destinations** (Cribl Edge). Select **CrowdStrike Falcon Next-Gen SIEM** from the list of tiles or the **Destinations** list. Next, click **Add Destination** to open a **New Destination** modal.

2. In the Destination modal, configure the following under **General Settings**:
   - **Output ID**: Enter a unique name to identify this Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
   - **Next-Gen SIEM endpoint**: The URL for sending data to CrowdStrike's Falcon Next-Gen SIEM. This URL is generated within the CrowdStrike platform and directs your data to the appropriate SIEM instance for processing.
   - **Request format**: Select `JSON` as the request format. This field defaults to `JSON`, which is the format CrowdStrike Falcon Next-Gen SIEM expects for proper data delivery.

The `Raw` option sends only the event's `_raw` value. This option is intended for advanced use cases where specific data manipulation or custom formatting is required. Use with caution, as it may not be compatible with standard Next-Gen SIEM ingest requirements.

3. Under **Authentication** select an **Authentication method** from the drop-down:
   - **Manual**: Displays a **Next-Gen SIEM authentication token** field for you to enter your authentication token.
   - **Secret**: Displays a **Next-Gen SIEM authentication token (text secret)** drop-down, in which you can select a stored secret that references the API token described above. A **Create** link is available to store a new, reusable secret.
4. Next, you can configure the following **Optional Settings** that you'll find across many Cribl Destinations:
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
5. Under **Processing** > **Post-Processing** keep the `cribl_pipe` field to assist with mapping and troubleshooting within NG SIEM. The payload sent to NG SIEM must contain only the `_raw` field. Do not include `_time`, `host`, or any other fields. NG SIEM ingest licensing is based on the size of the entire payload, and default NG SIEM parsers are designed to process only the `_raw` field.
   - To retain only the `_raw` field and discard all others, configure an Eval Function within a post-processing Pipeline as follows:

- In the **Keep fields** input, enter `_raw`.
- In the **Remove fields** input, enter `*`.

6. Optionally, configure any Persistent Queue, Retries, and Advanced settings outlined in the below sections.
7. Select **Save**, then **Commit & Deploy**.
8. Use the **Test** tab in the Destination's configuration modal to validate the configuration and connectivity of your Destination.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).

- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Defaults to toggled on to reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA).

Round-robin DNS: Defaults to toggled on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Defaults to toggled on to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`. Each request can potentially hit a different HEC receiver.

Body size limit (KB): Maximum size, in KB, of the request body. Defaults to `4096`. Lowering the size can potentially result in more parallel requests and also cause outbound requests to be made sooner.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1`.

- Retries happen on this flush interval.
- Any HTTP response code in the `2xx` range is considered success.
- Any response code in the `5xx` range is considered a retryable error, which will not trigger Persistent Queue (PQ) usage.
- Any other response code will trigger PQ (if PQ is configured as the Backpressure behavior).

Extra HTTP headers: Click **Add Header** to add **Name/Value** pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers that you want to declare as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new connection will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If the server does not support keepalives - or if the server closes a pooled connection while idle - a new connection will be established for the next request.
- Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between CrowdStrike Falcon LogScale servers.

Testing Next-Gen SIEM Parser

For data sources like syslog or flat-file logs, you can test your NG SIEM parser using sample data from Cribl Stream. This process is less applicable to API-driven data, as NG SIEM often performs pre-processing on API calls before parsing.

1. **Extract sample `_raw` data**: In Cribl Stream, while editing the relevant Pipeline, open the sample data output and copy the content of the `_raw` field.

2. **Add test event to parser**: In the NG SIEM parser editor, select **Add test** and paste the copied `_raw` data.
3. **Validate parsing**: Select **Test parser** to confirm that the event fields are parsed correctly.

If you are testing a default NG SIEM parser, you must first create a copy of the parser. Then, edit the copy and add the sample test event in the **Test** section.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.18. Databricks Destination

The Databricks Destination sends data to Databricks Unity Catalog volumes.

Type: Non-Streaming | TLS Support: N/A | PQ Support: N/A

Configure a Databricks Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Filesystem definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (supported only on Linux, not Windows) exposes a **Parquet Settings** left tab, where you **must** configure certain options in order to export data in Parquet format.
3. Next, you can configure the following **Optional Settings**:
    - **Upload path**: Specify an optional directory prefix for files uploaded to the Databricks events volume. It accepts a JavaScript expression (enclosed in quotes or backticks) that evaluates at initialization, allowing dynamic path generation using global variables. For example: `myEventsVolumePath-${C.vars.myVar}`.
      If unset, files are uploaded to the root of the events volume. Use this setting to organize output data by customer, date, or other runtime variables for efficient partitioning and querying in Databricks.
    - **Staging location**: The local filesystem directory where files are temporarily buffered before compression and upload to Databricks. This should be a performant and reliable storage path to avoid bottlenecks or data loss during output operations. Defaults to `$CRIBL_HOME/state/outputs/staging`.

- **Partitioning expression**: JavaScript expression that defines how files are partitioned into subdirectories within the Databricks **Upload path**. Evaluated per event, typically for time-based or custom partitioning. Defaults to `C.Time.strftime(_time ? _time : Date.now()/1000, '%Y/%m/%d')`. This default partitions files by event time (or current time if `_time` is absent), creating a directory structure like `2025/10/29/`.
  If blank, Cribl Stream will fall back to the event's `__partition` field value (if present); or otherwise to the root directory of the **Upload path**.
- **Compression**: Data compression format used before moving to the final destination. Defaults to `gzip` (recommended). This setting is not available when **Data format** is set to `Parquet`.
- **File name prefix expression**: The output file name prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
- **File name suffix expression**: The output file name suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.

  To prevent files from being overwritten, Cribl Stream appends a random sequence of six characters to the end of their names. This also applies to prefix and suffix expressions in file names.

  For example, if you set the **File name prefix expression** to `CriblExec` and set the **File name suffix expression** to `.csv`, the file name might display as `CriblExec-adPRWM.csv` with `adPRWM` appended.
- **Backpressure Behavior**: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. The **Unity Catalog OAuth** settings allow Cribl Stream to authenticate to your Databricks Workspace as an application (known as a service principal) without using a personal access token.
   - **Workspace ID**: Databricks Workspace identifier used for OAuth authentication. See how to find your Workspace ID for details.
   - **Client ID**: OAuth client ID for Unity Catalog authentication. See how to create a service principal for details.
   - **Client Secret**: Select an existing text secret from the drop-down or add a new text secret by selecting **Create**. See how to create and manage secrets for details.

- **OAuth scope**: The level of permission requested. The default `all-apis` scope requests permission to access all Databricks REST APIs that the service principal has been granted privileges. See authorize service principal access to Databricks with OAuth for details.

5. The **Databricks Settings** define the specific location within the Unity Catalog three-level namespace (`catalog.schema.object`) where Cribl Stream will write data.
   - **Catalog**: The top-level container for your data in Unity Catalog. The default value `main` is the catalog that Databricks automatically creates within a new Unity Catalog metastore.
   - **Schema**: The second level of the namespace, also known as a database. A schema contains tables, views, and volumes. Defaults to `external` to indicate that the data is coming from an external source.
   - **Events volume name**: Name of the events volume in Databricks where files are stored. Defaults to `events`.
6. Optionally, you can adjust the Processing, Parquet, and Advanced settings outlined in the sections below.
7. Select **Save**, then **Commit & Deploy**.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Set the target memory size for page segments. Generally, set lower values to improve reading speed, or set higher values to improve compression. Value must be a positive integer smaller than the **Row group size** value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Toggle on (default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Toggle on (default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to final output location. Defaults to `100`.

Cribl Stream will close files when **either** of the `File size limit (MB)` or the `File open time limit (sec)` conditions are met.

Compression level: Specifies the level of compression to apply before moving files to the final destination. Higher compression levels reduce file size but increase CPU usage. Choose from `Best Speed`, `Normal`, or `Best Compression`. The default level, `Best Speed`, will prioritize faster compression over smaller file sizes.

Writing high watermark (KB): Sets the high watermark for writing files, in kilobytes. This determines the buffer size for file writes, impacting performance and memory usage. Enter a numeric value representing the buffer size in kilobytes. The default size, `64` will use a 64 KB buffer for file writes.

Header line: If set, this line will be written to the beginning of each output file, followed by a newline character. This can be useful for adding a header row to CSV files.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined Min free disk space amount.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Force close on shutdown: Toggle on to force all staged files to close during an orderly shutdown. This triggers immediate upload or finalization of any in-progress files, regardless of **Idle time**, **File open time**, or **File size** limits. Use this option to minimize data loss risk in ephemeral or auto-scaling environments. This could result in a higher number of small files, as files are closed before reaching their configured thresholds. Ensure your Worker Group or Fleet **Drain timeout** is set high enough to allow all staged data to be finalized before shutdown completes.

Enable dead-lettering: Toggle on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.19. Datadog Destination

Cribl Stream can send log, metric, and trace events to Datadog. Datadog supports metrics of type `gauge`, `counter`, `rate`, and `distribution` via its REST API.

Cribl Stream sends data via two primary Datadog endpoints, which must be accessible from your Cribl environment:

- Logs: `http-intake.logs.{domain}`
- Metrics: `api.{domain}`
- APM Traces: `trace.agent.{domain}`

The `{domain}` placeholder in the endpoints represents the specific Datadog region you're using. By default, this is set to `datadoghq.com` for the US region. You can configure an alternative region with the **Datadog site** under **Optional Settings** below.

Supported Domains:

- US: `datadoghq.com` (default)
- US3: `us3.datadoghq.com`
- US5: `us5.datadoghq.com`
- Europe: `datadoghq.eu`
- US1-FED: `ddog-gov.com`
- AP1: `ap1.datadoghq.com`
- Custom: `custom.datadoghq.com`

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output to Datadog

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.

- To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.

2. In the **New Destination** modal, configure the following under **General Settings**:
   - **Output ID**: Enter a unique name to identify this Destination definition.
   - **Description**: Optionally, enter a description.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
   - **Manual**: Displays a field for you to enter an **API key** that is available in your Datadog profile.
   - **Secret**: This option exposes an **API key (text secret)** drop-down, in which you can select a stored secret that references the API access token described above. A **Create** link is available to store a new, reusable secret.
   - **API key**: Enter your Datadog organization's API key.
4. Next, you can configure the following **Optional Settings**:
   - **Datadog site**: Select the Datadog region you are sending to. Defaults to `US`; the other options are `US3`, `US5`, `Europe`, `US1-FED`, `AP1`, and `Custom`. Select `Custom` to manually enter your Datadog region's URL if it's not listed.
   - **Send logs as**: Specify the content type to use when sending logs. Defaults to `application/json`, where each log message is represented by a JSON object. The alternative `text/plain` option sends one message per line, with newline `\n` delimiters.
   - **Message field**: Name of the event field that contains the message to send. If not specified, Cribl Stream sends a JSON representation of the whole event (regardless of whether **Send logs as** is set to JSON or plain text).
   - **Source**: Name of the source to send with logs. If you're sending logs as JSON objects (that is, you've selected **Send logs as**: `application/json`), the event's `source` field (if set) will override this value.
   - **Host**: Name of the host to send with logs. If you're sending logs as JSON objects, the event's `host` field (if set) will override this value.
   - **Service**: Name of the service to send with logs. If you're sending logs as JSON objects, the event's `__service` field (if set) will override this value.
   - **Datadog tags**: List of tags to send with logs, such as `env:prod`, `env_staging:east`. These tags enhance search and filtering in Datadog. For log events, these tags are used in the default batching behavior controlled by the **Batch by tags** advanced setting. High tag cardinality can affect performance.

When events also include the `ddtags` internal field, its value must be a single string containing a comma-separated list of key:value pairs (for example, `env:dev,app:myapp`), not an array. Cribl

Stream merges these UI tags with the event `ddtags` value; event tags do not override tags configured here.

- **Severity**: Default value for message severity. If you're sending logs as JSON objects, the event's `__severity` field (if set) will override this value.

Datadog uses the above five fields (`source`, `host`, `__service`, `tags`, and `__severity`) to enhance searches.

- **Allow API key from events**: If toggled on, any API key in the `__agent_api_key` internal field will override the **API key** field's value. This option is useful if events originate from multiple Datadog Agent Sources, each configured with a different API key. (For further details, see Managing API Keys.)
- **Backpressure behavior**: Specify whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

> If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a

Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default; recommended) to compress the payload body before sending.

Send counter metrics as 'count': Unless toggled on, Datadog might transform `counter` metrics to `gauge`. Learn more about Datadog metrics types.

Batch by tags: When toggled on (default), log events are batched using both the API key and **Datadog tags** to optimize logical grouping. However, high cardinality in Datadog tags (many unique tag values) can result in excessive concurrent requests and potential `408` errors from Datadog. Toggling this setting off changes the batching behavior to group log events solely by API key, reducing batch count and improving performance.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (s): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

If an event contains the internal field `__criblMetrics`, Cribl Stream will send it to Datadog as a metric event. Events that do not contain `__criblMetrics` are sent as logs or traces, based on their structure. No internal fields are supported for metric events or trace events.

You can use these fields to **override** outbound event values for log events:

- `ddtags`: Optional tags to send with log events. The value must be a single string containing a comma-separated list of key:value pairs, not an array. For example: `env:dev,app:myapp`. If you send `ddtags` as an array (such as `["env:dev","app:myapp"]`), Datadog rejects the payload. When both the Datadog Destination Datadog tags setting and the event `ddtags` field are set, Cribl Stream merges them into a single tag list before sending them to Datadog.
- `__service`
- `__severity`

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

For More Information

You might find these Datadog references helpful:

- Submit Metrics
- Send Logs
- Metrics Types

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

8.2.20. Dynatrace HTTP Destination

The Dynatrace HTTP Destination sends logs to Dynatrace's SaaS or ActiveGate endpoints.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Configure a Dynatrace HTTP Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing**.
    - To configure via the Routes, select **Data** > **Destinations** (Stream) or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the Destination modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**. A **Description** is optional.
    - **Endpoint**: Select the type of Dynatrace endpoint where you want to send your data.
        - **Cloud**: Sends data to Dynatrace's SaaS endpoint. Provide your Dynatrace **Environment ID**. This ID uniquely identifies your Dynatrace environment and is required for routing data correctly. You can find your environment ID in your Dynatrace account settings.
        - **ActiveGate**: Choose this option if you are routing data through an ActiveGate. Provide your Dynatrace **Environment ID** and **ActiveGate domain** in the additional fields that appear. Your ActiveGate domain is displayed in your Dynatrace settings in the **Details** section of an ActiveGate under the **Hostname** field. The format should be a fully qualified domain name (FQDN) or IP address. For example, `https://{activeGate-domain}:9999/e/{environment-id}/api/v2/logs/ingest`.
        - **Manual**: Select this option if you want to manually specify the endpoint URL. This is useful for custom setups or non-standard configurations. You will need to enter the full **URL** in the field that appears. Can be overwritten by an event's `__url` field.
3. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.

- **Tags**: Optionally, add tags that you can use to filter and group Destinations in Cribl Stream's **Manage Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Select **Authentication** from the sidebar and select an **Authentication type**.
   - `Auth token`: Provide a Dynatrace API access token to authenticate in the **Token** field. This is the most common method for securing data transmission to Dynatrace. You can generate this token in your Dynatrace account settings under the API tokens section. Ensure that the token has the necessary permissions for the data you are sending.
   - `Token (text secret)`: Create or select a Cribl stored text secret that contains your Dynatrace API access token.
5. Optionally, configure any Persistent Queue, Processing, Retries, and Advanced settings outlined in the below sections.
   - **Persistent Queue** ensures data durability and reliability by buffering and preserving incoming events when the receiver is down or exhibiting backpressure.
   - **Processing Settings** allow you to define pre-processing and post-processing Pipelines. This includes adding metadata, applying transformations, and conditioning the data before it's sent.
   - **Retries** allow you to specify how the system should handle different HTTP response status codes and timeouts, ensuring reliable data delivery.
   - **Advanced Settings** offer additional configuration options that enhance the flexibility and performance of data transmission.

To track data sent via Cribl, you can add a custom attribute in the **Extra HTTP headers** section under Advanced Settings. For example, select `Add Header` and enter `cribl` as the name and `yes` as the value.

6. Select **Save**, then **Commit & Deploy**.
7. Use the **Test** tab in the Destination's configuration modal to validate the configuration and connectivity of your Destination.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)

- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.

- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx, 3xx` series	Drop the request
`408, 429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations

send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will

sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle this off if you want Cribl Stream to close the connection immediately after sending a request.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. You can set this limit to as high as 5 MB (`5000` KB). Be aware that high values can cause high memory usage per Worker Node, especially if a dynamically constructed URL (see Internal Fields) causes this Destination to send events to more than one URL. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass to all events as additional HTTP headers. Values will be sent encrypted. You can also add headers dynamically on a per-event basis in the `__headers` field. See Internal Fields below.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header

names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__criblMetrics`
- `__url`
- `__headers`

If an event contains the internal field `__url`, that field's value will override the **General Settings** > **Endpoint** value. This way, you can determine the endpoint URL dynamically.

For example, you could create a Pipeline containing an Eval Function that adds the `__url` field, and connect that Pipeline to your Dynatrace HTTP Destination. Configure the Eval Function to set `__url`'s value to a URL that varies depending on a global variable, or on some property of the event, or on some other dynamically-generated value that meets your needs.

If an event contains the internal field `__headers`, that field's value will be a JSON object containing Name-value pairs, each of which defines a header. By creating Pipelines that set the value of `__headers` according to conditions that you specify, you can add headers dynamically.

For example, you could create a Pipeline containing Eval Functions that add the `__headers` field, and connect that Pipeline to your Dynatrace HTTP Destination. Configure the Eval Functions to set `__headers` values to Name-value pairs that vary depending on some properties of the event, or on dynamically-generated values that meet your needs.

Here's an overview of how to add headers:

- To add "dynamic" (a.k.a. "custom") headers to some events but not others, use the `__headers` field.
- To define headers to add to **all** events, use **Advanced Settings** > **Extra HTTP Headers**.
- An event can include headers added by both methods. So if you use `__headers` to add `{ "api-key": "foo" }` and **Extra HTTP Headers** to add `{ "goat": "Kid A" }`, you'll get both headers.
- Headers added via the `__headers` field take precedence. So if you use `__headers` to add `{ "api-key": "foo" }` and **Extra HTTP Headers** to add `{ "api-key": "bar" }`, you'll get only one header, and that will be `{ "api-key": "foo" }`.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, it will throw away the connection and attempt a new one. This is to prevent sticking to a particular destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), Cribl Stream will establish a new connection for the next request.
- When resolving the Destination's hostname, Cribl Stream will pick the first IP in the list for use in the next connection. Enable **Round-robin DNS** to better balance the distribution of events among destination cluster nodes.

Troubleshoot

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.21. Dynatrace OTLP Destination

The Dynatrace OTLP Destination sends telemetry data, including traces, logs, and metrics, to Dynatrace using the OpenTelemetry Protocol (OTLP). It can send data to various Dynatrace endpoints, including SaaS, Environment ActiveGate, and Containerized Environment ActiveGate.

To get started, connecting to Dynatrace is simple. Select your endpoint and provide your access token. Need more detailed instructions or configuration options? See below!

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Configure a Dynatrace OTLP Destination

In Cribl Stream, set up a Dynatrace OTLP Destination.

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing**.
    - To configure via the Routes, select **Data** > **Destinations** (Stream) or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the Destination modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**. A **Description** is optional.
    - **OTLP version**: Already set to `1.3.1` and cannot be changed.
    - **Protocol**: Already set to `HTTP` and cannot be changed. Uses Binary Protobuf encoding (does not support JSON Protobuf).
    - **Endpoint type**: Select the type of Dynatrace endpoint where you want to send your data.
        - **SaaS**: Sends data to Dynatrace's SaaS endpoint.
        - **ActiveGate**: Choose this option if you are routing data through an ActiveGate.
    - **Endpoint**: The endpoint URL for the Dynatrace environment. You'll need to provide your Dynatrace environment ID. This ID uniquely identifies your Dynatrace environment and is required for routing data correctly. You can find your environment ID in your Dynatrace account settings.

        - **SaaS** defaults to `https://{your-environment-id}.live.dynatrace.com/api/v2/otlp`. Replace the `{your-environment-id}` placeholder with your environment ID.
        - **ActiveGate** defaults to `https://{your-activegate-domain}:9999/e/{your-environment-id}/api/v2/otlp`. Replace the `{your-activegate-domain}` placeholder with your ActiveGate domain, and `{your-environment-id}` with your environment ID. Your ActiveGate domain is displayed in your Dynatrace settings in the **Details** section of an ActiveGate under the **Hostname** field. The format should be a fully qualified domain name (FQDN) or IP address.
        - You can specify a custom endpoint using either a valid URL or an IP address (IPv4 or IPv6). Enclose IPv6 addresses in square brackets (for example, `[2001:db8:85a3:0000:0000:8a2e:0370:7334]`).
        - To proxy outbound HTTP/S requests, see System Proxy Configuration.
    - **Auth token (text secret)**: Create or select a Cribl stored text secret that contains your Dynatrace API access token, ensuring secure authentication for data transmission. You can generate this token in your Dynatrace account settings under the API tokens section. Ensure that the token has the necessary permissions for the data you are sending.
    - **Optional Settings**:
        - You can customize the path instead of using the default cloud endpoints. This is useful when integrating with a hosted Lightstep satellite.
        - **Backpressure behavior**: Whether to `Block`, `Drop`, or `Persistent Queue` events when all receivers are exerting backpressure.
        - **Tags**: Optionally, add tags that you can use to filter and group Destinations in Cribl Stream's **Manage Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
3. The **Processing Settings**, **Retries**, and **Advanced Settings** provide granular control over how data is handled and transmitted.
    - Processing Settings allow you to define pre-processing and post-processing Pipelines, ensuring that events conform to the relevant Protobuf specifications for traces, metrics, and logs. This includes adding metadata, applying transformations, and conditioning the data.
    - Retries allow you to specify how the system should handle different HTTP response status codes and timeouts, ensuring reliable data delivery.
    - Advanced Settings offer additional configuration options that enhance the flexibility and performance of data transmission.
4. Select **Save**, then **Commit & Deploy**.
5. Use the **Test** tab in the Destination's configuration modal to validate the configuration and connectivity of your Destination.

Custom HTTP Path

To send data to a custom location, override the default endpoint paths. When the **Protocol** is set to `HTTP`, this Destination sends data to the following endpoints by default:

- Traces: `<endpoint>/v1/traces`
- Metrics: `<endpoint>/v1/metrics`
- Logs: `<endpoint>/v1/logs`

`<endpoint>` refers to the base URL you specified in the **Endpoint** setting above.

To override a default path, provide only the desired path in the corresponding field: **Traces endpoint override**, **Metrics endpoint override**, or **Logs endpoint override**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.

Mode	Description
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output - both metric events, as dimensions; and log events, as labels. Supports wildcards.

By default, includes `cribl_pipe` (Cribl Stream Pipeline that processed the event).

Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx, 3xx` series	Drop the request
`401, 403, 408, 429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle off if you want Cribl Stream to close the connection immediately after sending a request.

Compression: Compression type to apply to messages sent to the OpenTelemetry endpoint. `Gzip` (the default) and `None` are available for both protocols.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30` sec.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `2048` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events. See Dynatrace's best practices.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` sec.

Extra HTTP headers: Select **Add Header** to define additional HTTP headers to pass to all events. Each row is a **Name**-**Value** pair. Values will be sent encrypted.

Safe headers: Add headers here to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: Optionally, specify a single Git branch on which to enable this configuration. If this field is empty, the config will be enabled everywhere.

Protocol and Transport Support

This Destination leverages the OpenTelemetry Protocol (OTLP) v1.3.1 to send observability data to Dynatrace. OTLP defines Protocol buffer (Protobuf) schemas for its payloads (requests and responses). With the HTTP transport, this Destination supports Binary Protobuf payload encoding, but currently does not support JSON Protobuf.

When configuring Pipelines (including pre-processing and post-processing Pipelines), you need to ensure that events sent to this Destination conform to the relevant Protobuf specification.

- For traces, opentelemetry/proto/trace/v1/trace.proto
- For metrics, opentelemetry/proto/metrics/v1/metrics.proto
- For logs, opentelemetry/proto/logs/v1/logs.proto

This Destination will drop non-conforming events. If the Destination encounters a parsing error when trying to convert an event to OTLP, it discards the event and Cribl Stream logs the error.

Troubleshoot

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.22. Exabeam Security Operations Platform Destination

Cribl Stream supports sending data to the Exabeam security operations platform or (as some people think of it) SIEM.

The Exabeam Destination supersedes and improves upon the "old way" to get data from Cribl Stream to Exabeam using the Cribl Webhook Destination.

Type: Non-Streaming | TLS Support: Yes | PQ Support: No

For examples of using Cribl Stream to optimize inbound events for Exabeam, see:

- Splunk to Exabeam, which demonstrates combining several Cribl Functions to transform events before forwarding to Exabeam.
- Several Exabeam Packs on the Cribl Packs Dispensary, which provide processing Pipelines that you can import and adapt to your needs.

Understand How Exabeam and Cribl Stream Work Together

Exabeam parsers expect to receive events in original, unmodified form. To make sure this happens, you'll sometimes need to use a combination of Event Breakers and Pipelines. In general, knowing this can help you avoid problems or troubleshoot any that do crop up.

When you send data to Exabeam, Cribl Packs are your friends, because Cribl and Exabeam have partnered to create a set of Exabeam-specific Packs, each of which ensures that events from one common data source arrives at Exabeam in precisely the expected form. Cribl Packs also give you the option of dropping events on a per-data-source basis, so that Cribl Stream never sends irrelevant events to Exabeam. To view the Exabeam-specific Cribl Packs, navigate to **Processing** > **Packs**. Then select **Add Pack** > **Add from Dispensary**, and enter `Exabeam` in the search field.

Cribl strongly recommends that you use **only** the Exabeam-specific, Cribl-authored Packs with this Destination. Other Packs can perform transformations - especially reduction - that prevent the Exabeam parsers from triggering.

When the original data source is Microsoft Active Directory (AD) or LDAP, you'll need to set up an Exabeam Site Collector to perform the initial data capture.

Because Exabeam runs in the Google Cloud Storage (GCS) platform, some setting names include references to GCS.

Finally, Cribl recommends placing Cribl Stream Worker Nodes as close as possible to the sources of the events you want to send to Exabeam. This reduces the length of the path that data must traverse before Cribl Stream optimizes it with compression, reduction, dropping events, and/or redacting sensitive information.

Configure Cloud Storage Permissions

For Cribl Stream to send data to Exabeam buckets, the following access permissions must be set on the Cloud Storage side:

- Fine-grained access control must be enabled on the buckets.
- The Google service account or user must have the Storage Admin or Owner role.

For details, see the Cloud Storage Overview of Access Control and Understanding Roles documentation.

Create the Cribl Cloud Collector in Exabeam

Log in to the Exabeam Security Operations Platform, and on the **Home** page, select **Cloud Collectors**. In the resulting page, select the **Cribl** tile to see a list of Cribl Collectors (the first time you do this, the list will be empty).

Select **New Collector**, and in the resulting **Collector for Cribl** drawer:

1. Enter a name for your new Cribl Cloud Collector.
2. Optionally, in **Advanced Settings**, configure the **Metadata** value **TIMEZONE**
3. If your desired site already exists, select it from the **SITE** drop-down.
    - If your desired site does not exist yet, skip the drop-down and select the **manage your sites** link underneath. This will take you to a separate UI where you will enter a value for the **Metadata** value **SITENAME**, and Exabeam will generate the associated metadata value **SITE ID**. When you have done this, Exabeam will take you back to the previous UI, and you must then select your newly-created site from the **SITE** drop-down.
4. Select **Install**.

A `Hooray!` confirmation modal will now replace the **Collector for Cribl** drawer. In this modal:

1. Select the copy icon to the right of the encoded **Connection String**. Save this string in a secure and handy location - it is required when configuring the Cribl Stream Exabeam Destination.
2. Select **Go to Overview** to exit the modal.

Although optional, configuring metadata is useful because it allows a single instance of the Exabeam SIEM to differentiate between data from multiple sources. This is relevant for large enterprises whose IT infrastructure comprises many sources whose IP addresses overlap.

Configure Cribl Stream to Output to Cloud Storage Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Exabeam Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - Select **Autofill with Exabeam Connection String** to automatically enter the values for the following **Exabeam Settings**. Or enter them directly.
    - **Google Cloud Storage bucket**: Name of the destination bucket - in Exabeam's Cribl Cloud Collector this is called **GCS Bucket Name**. This value can be a constant, or a JavaScript expression that can be evaluated only at init time. For example, you can reference a Global Variable like this: `myBucket-${C.vars.myVar}`.
    - **Google Cloud Storage bucket region**: In Exabeam's Cribl Cloud Collector this is called **GCS Bucket Region**. It is the GCS region where the bucket is located.
    - **Collector instance ID**: In Exabeam's Cribl Cloud Collector this is called **Instance ID**.
    - **Access key**: In Exabeam's Cribl Cloud Collector this is called **Access Key**.
    - **Secret**: In Exabeam's Cribl Cloud Collector this is called **Secret Key**.
    - **Site name**: In Exabeam's Cribl Cloud Collector this is called **SITENAME**. This determines where your data is categorized. Set a static site name or dynamically assign it using a JavaScript expression. For example, to dynamically set the site name based on the `cribl_pipeline` field you'd use the expression `${C.vars.cribl_pipeline}`. This assigns the site name based on the Pipeline value.
    - **Site ID**: In Exabeam's Cribl Cloud Collector this is called **SITE ID**.
    - **Timezone offset**: In Exabeam's Cribl Cloud Collector this is called **TIMEZONE**.
3. Optionally, you can adjust the Processing and Advanced settings outlined in the sections below.
4. Select **Save**, then **Commit & Deploy**.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports `c*` wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Backpressure behavior: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300` seconds (5 minutes).

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to final output location. Defaults to `100`.

Cribl Stream will also close any files larger than 10 MB.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined Min free disk space amount.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up

cleanup but increase memory usage.

Enable dead-lettering: Toggle this on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates (such as self-signed certificates, for example) that cannot be verified against a valid Certificate Authority. Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Tags: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Troubleshooting

Verify that you are sending the original, unmodified events. Study the applicable Exabeam parsers to make sure you understand exactly what they're doing. If you need to modify events, make sure that when you do, the relevant parsers still trigger.

Nonspecific messages from Google Cloud of the form `Error: failed to close file` can indicate problems with the permissions listed above.

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.23. Microsoft Fabric Real-Time Intelligence

Cribl Stream supports sending data to Eventstreams in Microsoft Fabric Real-Time Intelligence.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Before you create a Microsoft Fabric Real-Time Intelligence Destination in Cribl Stream, create a **Cribl** data source in Microsoft Fabric. Microsoft Fabric will then provision the connection values (like the bootstrap server) that you must use when you configure a Microsoft Fabric Real-Time Intelligence Destination in Cribl Stream.

Configuring Cribl Stream to Output to Microsoft Fabric Real-Time Intelligence

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: A unique name to identify this Microsoft Fabric Real-Time Intelligence Destination. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Bootstrap server**: The value of the **Bootstrap server** setting for the **Cribl** data source in Microsoft Fabric.
    - **Topic name**: The value of the **Topic name** setting for the **Cribl** data source in Microsoft Fabric. Can be overwritten using the `__topicOut` field.
3. Next, you can configure the following **Optional Settings**:
    - **Acknowledgments**: Control the number of required acknowledgments. Defaults to `Leader`. Setting this control to `Leader` or `All` may significantly slow throughput. If the impact is too great for your use case, set this control to `None`.
    - **Record data format**: Format to use to serialize events before writing. Defaults to `JSON`.

        - `JSON` serializes the entire event from your Pipeline, including all metadata and enriched fields, into a single JSON object.
        - `Field _raw` sends only the contents of the `_raw` field as the payload. This option provides maximum throughput and minimal overhead. If using raw data, send the data in CSV or JSON format.
    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the TLS, Authentication, Processing, Retries and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

TLS Settings (Client Side)

Enabled: Defaults to toggled on.

Validate server certs: Defaults to toggled on.

Authentication

Authentication parameters to use when connecting to Microsoft Fabric. Using TLS is highly recommended.

SASL mechanism: SASL (Simple Authentication and Security Layer) authentication mechanism to use. Defaults to `PLAIN`, which exposes Basic Authentication options from the **SASL mechanism** settings in the **Cribl** data source in Microsoft Fabric. Select `OAUTHBEARER` to enable OAuth Authentication via a different set of options.

Basic Authentication

Selecting the `PLAIN` SASL mechanism provides the options listed in this section.

SASL JASS username: The value is `$ConnectionString` (immutable).

SASL JASS password: The stored secret that references the **SASL JASS password-primary** or **SASL JASS password-secondary** in the **Cribl** data source in Microsoft Fabric. The secret can reside in Cribl Stream's internal secrets manager or (if enabled) in an external KMS. A **Create** link is available if you need a new secret.

OAuth Authentication

Selecting the `OAUTHBEARER` SASL mechanism provides the options listed in this section.

Microsoft Entra ID authentication endpoint: Specifies the Microsoft Entra ID endpoint from which to acquire authentication tokens. Defaults to `https://login.microsoftonline.com`. You can instead select `https://login.microsoftonline.us` or `https://login.partner.microsoftonline.cn`.

Client ID: Enter the `client_id` to pass in the OAuth request parameter.

Tenant identifier: Enter your Microsoft Entra ID subscription's directory ID (tenant ID).

Scope: Enter the scope to pass in the OAuth request parameter. This will be of the form: `https://<Bootstrap-Server-Hostname>/.default`. For example, for a bootstrap server `myfabricserver.example.com:9093`, **Scope** would be `https://myfabricserver.example.com/.default`.

Authentication method: Use the buttons to select one of these options:

- **Secret**: Exposes a **Client secret (text secret)** drop-down, in which you can select a stored secret that references the `client_secret`. A **Create** link is available to define a new secret.
- **Certificate**: Exposes a **Certificate** drop-down, in which you can select a stored certificate. A **Create** link is available to define a new cert.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_group` - Cribl Stream Worker Group that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_mode` - Distributed mode of the instance that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

These settings provide flexibility in handling retries for failed messages, allowing you to balance between quick retries and avoiding excessive load on the system.

The default configuration starts with a 300 milliseconds retry interval, doubles the interval after each retry, and caps the maximum retry interval at 30 seconds. The system will attempt to retry the request up to five times before considering it a failure.

Retry limit: Maximum number of times to retry a failed request before the message fails. Defaults to `5`. Enter `0` to not retry at all. For example, if set to `5`, the system will attempt to retry the request up to five times before considering it a failure.

Initial retry interval (ms): Initial value used to calculate the retry interval, in milliseconds. This value determines the starting point for the retry delay. The default (and minimum) is `300` ms (three seconds). The maximum allowed value is `600000` ms (10 minutes). For example, if set to `1000` ms, the first retry will occur after one second.

Backoff multiplier: Set the backoff multiplier (`2`–`20`) to control the retry frequency for failed messages. The multiplier is used in an exponential backoff formula to increase the delay between retries. For faster retries, use a lower multiplier. For slower retries with more delay between attempts, use a higher multiplier. For example, with an initial retry interval of `1000` ms and a multiplier of `2`, the retry intervals will be 1,000 ms, 2,000 ms, 4,000 ms, and so on. See the Kafka documentation for details.

Backoff limit (ms): The maximum wait time for a retry, in milliseconds. This setting caps the exponential backoff delay to prevent excessively long wait times. The default (and minimum) value is `30000` ms (30 seconds), and the maximum is `180000` ms (180 seconds). For example, if the calculated retry interval exceeds 180,000 ms, the retry will occur after 180,000 ms instead.

Advanced Settings

Record size limit (KB, uncompressed): Maximum size (KB) of each record batch before compression. Setting should be < `message.max.bytes` settings in bootstrap server. Defaults to `768`.

Events-per-batch limit: Maximum number of events in a batch before forcing a flush. Defaults to `1000`.

Flush period (sec): Maximum time between requests. Low settings could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Connection timeout (ms): Maximum time to wait for a successful connection. Defaults to `10000` ms (10 seconds). Valid range is `1000` to `3600000` ms (1 second to 1 hour).

Request timeout (ms): Maximum time to wait for Kafka to respond to a successful request. Defaults to `60000` ms (1 minute).

Authentication timeout (ms): Maximum time to wait for Kafka to respond to an authentication request. Defaults to `10000` (10 seconds).

Reauthentication threshold (ms): Time window during which Cribl Stream can reautheticate if needed, measured backward from the moment when credentials are set to expire. Defaults to `10000` (10 seconds).

A small value for this setting, combined with high network latency, might prevent the Destination from reauthenticating before the bootstrap server closes the connection.

A large value might cause the Destination to send reauthentication messages too soon, wasting bandwidth.

The Kafka setting `connections.max.reauth.ms` controls the reuthentication threshold on the Kafka side.

Working with Event Timestamps

By default, when an incoming event reaches this Destination, the underlying Kafka JS library adds a `timestamp` field set to the current time at that moment. The library gets the current time from the `Date.now()` JavaScript method. Events that this Destination sends to downstream Kafka receivers include this `timestamp` field.

Some production situations require the `timestamp` value to be the time an incoming event was originally created by an upstream sender, not the current time of its arrival at this Destination. For example, suppose the events were originally created over a wide time range, and arrive at the Destination hours or days later. In this case, the original creation time might be important to retain in events Cribl Stream sends to downstream Kafka receivers.

Here is how to satisfy such a requirement:

1. In a Pipeline that routes events to this Destination, use an Eval Function to add a field named `__kafkaTime` and write the incoming event's original creation time into that field. The value of `__kafkaTime` **must** be in Unix time (either seconds or milliseconds - both will work). For context, see this explanation of the related fields `_time` and `__origTime`.
2. This Destination will recognize the `__kafkaTime` field and write its value into the `timestamp` field. This is similar to the way you can use the `__topicOut` field to overwrite a topic setting, as described above.

If the `__kafkaTime` field is **not** present, the Destination will apply the default behavior (using `Date.now()`) described previously.

Fabric Workspace Permissions

To add a **Cribl** data source in Microsoft Fabric and retrieve the values necessary to configure the Microsoft Fabric Real-Time Intelligence Destination in Cribl Stream, you must have **Contributor** (or higher) Fabric workspace permissions.

To use `OAUTHBEARER` authentication, you need **Member** (or higher) Microsoft Fabric workspace permissions so that you can add a service principal app to the workspace with **Contributor** (or higher) permissions.

See Permission model in the official Microsoft documentation for more information.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__topicOut`
- `__key`
- `__headers`
- `__keySchemaIdOut`
- `__valueSchemaIdOut`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.24. Filesystem/NFS Destination

Filesystem is a non-streaming Destination type that Cribl Stream can use to output files to a local file system or a network-attached file system (NFS).

Type: Non-Streaming | TLS Support: N/A | PQ Support: N/A

In Cribl.Cloud, the Filesystem Destination is only available on customer-managed hybrid Worker Nodes.

Configure Cribl Stream to Output to Filesystem Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Filesystem definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Output location**: Final destination for the output files.
    - **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (supported only on Linux, not Windows) exposes a **Parquet Settings** left tab, where you **must** configure certain options in order to export data in Parquet format.
3. Next, you can configure the following **Optional Settings**:
    - **Staging location**: Local filesystem location in which to buffer files before compressing and moving them to the final destination. Cribl recommends that this location be stable and high-performance.

        The **Staging location** field is not displayed or available on Cribl.Cloud-managed Worker Nodes.

- **Partitioning expression**: JavaScript expression that defines how files are partitioned and organized. Default is date-based. If blank, Cribl Stream will fall back to the event's `__partition` field value (if present); or otherwise to the root directory of the **Output Location** and **Staging Location**.
- **Compression**: Data compression format used before moving to final destination. Defaults to `gzip` (recommended). This setting is not available when **Data format** is set to `Parquet`.
- **File name prefix expression**: The output filename prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
- **File name suffix expression**: The output filename suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.

  To prevent files from being overwritten, Cribl appends a random sequence of 6 characters to the end of their names. This also applies to prefix and suffix expressions in file names.

  For example, if you set the **File name prefix expression** to `CriblExec` and set the **File name suffix expression** to `.csv`, the file name might display as `CriblExec-adPRWM.csv` with `adPRWM` appended.
- **Backpressure Behavior**: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Processing, Parquet, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.

- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Set the target memory size for page segments. Generally, set lower values to improve reading speed, or set higher values to improve compression. Value must be a positive integer smaller than the **Row group size** value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Toggle on (default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Toggle on (default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to final output location. Defaults to `100`.

Cribl Stream will close files when **either** of the `File size limit (MB)` or the `File open time limit (sec)` conditions are met.

Compression level: Specifies the level of compression to apply before moving files to the final destination. Higher compression levels reduce file size but increase CPU usage. Choose from `Best Speed`, `Normal`, or `Best Compression`. The default level, `Best Speed`, will prioritize faster compression over smaller file sizes.

Writing high watermark (KB): Sets the high watermark for writing files, in kilobytes. This determines the buffer size for file writes, impacting performance and memory usage. Enter a numeric value representing the buffer size in kilobytes. The default size, `64` will use a 64 KB buffer for file writes.

Header line: If set, this line will be written to the beginning of each output file, followed by a newline character. This can be useful for adding a header row to CSV files.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined Min free disk space amount.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

For a Destination originally configured in a Cribl Stream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this toggle's state. This is so that upon Cribl

> Stream upgrade and restart, any files pending in the original staging directory will not be lost. To enable this option for such Destinations, Cribl's recommended migration path is:
>
> - Clone the Destination.
> - Where Routes reference the original Destination, redirect them to instead reference the new, cloned Destination.
>
> This way, the original Destination will process pending files (after an idle timeout), and the new, cloned Destination will process newly arriving events with **Add output ID** enabled.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Force close on shutdown: Toggle on to force all staged files to close during an orderly shutdown. This triggers immediate upload or finalization of any in-progress files, regardless of **Idle time**, **File open time**, or **File size** limits. Use this option to minimize data loss risk in ephemeral or auto-scaling environments. This could result in a higher number of small files, as files are closed before reaching their configured thresholds. Ensure your Worker Group or Fleet **Drain timeout** is set high enough to allow all staged data to be finalized before shutdown completes.

Enable dead-lettering: Toggle on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

To export events from an intermediate stage **within a Pipeline** to a file, see the Tee Function.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.25. Honeycomb Destination

Cribl Stream supports sending events to a Honeycomb dataset.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output to Honeycomb

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Honeycomb definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Dataset name**: Name of the dataset to send events to. (For example, `iLoveObservabilityDataset`.)
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Displays a field for you to enter the **API key** for the team to which the dataset belongs.
    - **Secret**: This option exposes an **API key (text secret)** drop-down, in which you can select a stored secret that references the API key described above. A **Create** link is available to store a new, reusable secret.
    - **API key**: Enter your Honeycomb organization's API key.
4. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request

Status Code	Action
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new one will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), a new connection will be established for the next request.
- When resolving the Destination's hostname, Cribl Stream will pick the first IP in the list for use in the next connection. Enable Round-robin DNS to better balance distribution of events between destination cluster nodes.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.26. InfluxDB Destination

Cribl Stream supports sending data to InfluxDB (versions 1.x and 2.0.x) and InfluxDB Cloud.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Configure Cribl Stream to Output to InfluxDB

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this InfluxDB definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Write API URL**: The URL of an InfluxDB cluster to send events to:
        - v1 API: `http://localhost:8086/write`
        - v2 API: `http://localhost:8086/api/v2/write`
    - **Use v2 API**: You can enable the InfluxDB v2 API with InfluxDB version 1.8 or later. Toggle off (default) to fall back to the v1 API and display a **Database** field. If you toggle **Use v2 API** on, Cribl Stream communicates using InfluxDB's v2 API, and instead displays these two fields:
        - **Bucket**: Enter the bucket to write to (required).
        - **Organization**: The Organization ID corresponding to the specified **Bucket** (required in this configuration, although InfluxDB v.1.8 will ignore it).
    - **Database**: Name of the database on which to write data points. Displayed only for configurations that use the v1 API (required).
3. Next, you can configure the following **Optional Settings**:
    - **Timestamp precision**: Sets the precision for the supplied Unix time values. Defaults to `Milliseconds`.

- **Dynamic value fields**: When toggled on (default), Cribl Stream will pull the value field from the metric name. For example, `db.query.user` will use `db.query` as the measurement and `user` as the value field.
- **Value field name**: Name of the field in which to store the metric when sending to InfluxDB. This will be used as a fallback if dynamic name generation is enabled but fails. Defaults to `value`.
- **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Authentication

Use the **Authentication type** drop-down to select one of these options:

None: This default setting does not use authentication.

Auth token: Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.

Auth token (text secret): This option exposes a **Token (text secret)** drop-down, in which you can select a stored text secret that references the bearer token described above. A **Create** link is available to store a new, reusable secret.

Basic: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.

Basic (credentials secret): This option exposes a **Credentials secret** drop-down, in which you can select a stored text secret that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are **not** authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.27. MinIO Destination

MinIO is a non-streaming Destination type, to which Cribl Stream can output objects.

Type: Non-Streaming | TLS Support: Configurable | PQ Support: No

Configure Cribl Stream to Output to MinIO Destinations

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this MinIO definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **MinIO endpoint**: MinIO service URL (for example, http://minioHost:9000).
    - **MinIO bucket name**:Name of the destination MinIO bucket. This value can be a constant, or a JavaScript expression that will be evaluated only at init time. For example, referencing a Global Variable: `myBucket-${C.vars.myVar}`. Ensure that the bucket already exists, otherwise MinIO will generate "bucket does not exist" errors.

      Event-level variables are not available for JavaScript expressions. This is because the bucket name is evaluated only at Destination initialization. If you want to use event-level variables in file paths, Cribl recommends specifying them in the **Partitioning Expression** field (described below), because this is evaluated for each file.

    - **Staging location**: Filesystem location in which to locally buffer files before compressing and moving to final destination. Cribl recommends that this location be stable and high-performance.

      The **Staging location** field is not displayed or available on Cribl.Cloud-managed Worker Nodes.

- **Data format**: The output data format defaults to `JSON`. `Raw` and `Parquet` are also available. Selecting `Parquet` (supported only on Linux, not Windows) exposes a **Parquet Settings** left tab, where you **must** configure certain options in order to export data in Parquet format.

3. Next, you can configure the following **Optional Settings**:
   - **Key prefix**: Root directory to prepend to path before uploading. Enter a constant, or a JS expression enclosed in single quotes, double quotes, or backticks. Prefix to apply to files/objects before uploading to the specified bucket. MinIO will display key prefixes as folders.
   - **Partitioning expression**: JavaScript expression that defines how files are partitioned and organized. Default is date-based. If blank, Cribl Stream will fall back to the event's `__partition` field value (if present); or otherwise to the root directory of the **Output Location** and **Staging Location**.

     Cribl Stream's internal `__partition` field can be populated in multiple ways. The precedence order is: explicit **Partitioning expression** value -> `${host}/${sourcetype}` (default) **Partitioning expression** value -> user-defined `event.__partition`, set with an Eval Function (takes effect only where this **Partitioning expression** field is blank).

   - **Compression**: Data compression format used before moving to final destination. Defaults to `gzip` (recommended). This setting is not available when **Data format** is set to `Parquet`.
   - **File name prefix expression**: The output filename prefix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `CriblOut`.
   - **File name suffix expression**: The output filename suffix. Must be a JavaScript expression (which can evaluate to a constant), enclosed in quotes or backticks. Defaults to `` `.${C.env["CRIBL_WORKER_ID"]}.${__format}${__compression === "gzip" ? ".gz" : ""}` ``, where `__format` can be `json` or `raw`, and `__compression` can be `none` or `gzip`.
   - **Backpressure behavior**: Select whether to block or drop events when all receivers are exerting backpressure. (Causes might include an accumulation of too many files needing to be closed.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the Authentication, Processing, Parquet, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

How MinIO Composes File Names

The full path to a file consists of:

`<bucket_name>/<keyprefix><partition_expression | __partition><file_name_prefix><filename>.<extension>`

As an example, assume that the **MinIO bucket name** is `bucket1`, the **Key prefix** is `aws`, the **Partitioning expression** is `` `${host}/${sourcetype}` ``, the source is undefined, the **File name prefix** is the default `CriblOut`, and the **Data format** is `json`. Here, the full path as displayed in MinIO would have this form: `/bucket1/aws/192.168.1.241/undefined/CriblOut-<randomstring>0.json`

Although MinIO will display the **Key prefix** and **Partitioning expression** values as folders, both are actually just part of the overall key name, along with the file name.

Authentication

Use the **Authentication Method** drop-down to select one of these options:

Auto: This default option uses the AWS SDK for JavaScript to automatically obtain credentials in the following order of attempts:

1. **IAM Roles for Amazon EC2**: Loaded from AWS Identity and Access Management (IAM) roles attached to an EC2 instance.
2. **Shared Credentials File**: Loaded from the shared credentials file (`~/.aws/credentials`).
3. **Environment Variables**: Loaded from environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
4. **JSON File on Disk**: Loaded from a JSON file on disk.
5. **Other Credential-Provider Classes**: Other credential-provider classes provided by the AWS SDK for JavaScript.

The `Auto` method works both when running on AWS and in other environments where the necessary credentials are available through one of the above methods.

SSO Providers

When using the auto authentication method, you can leverage SSO providers like SAML and Okta to issue temporary credentials. These credentials should be set in the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. The AWS SDK will then use these environment variables to authenticate.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, for example, those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key**: Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key**: Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

The values for **Access key** and **Secret key** can be a constant, or a JavaScript expression (such as `${C.env.MY_VAR}`) enclosed in quotes or backticks, which allows configuration with environment variables.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. This option exposes a **Secret key pair** drop-down, in which you can select a stored secret that references the set of user-associated IAM credentials described above. A **Create** link is available to store a new, reusable secret.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Parquet Settings

To write out Parquet files, note that:

- On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.
- Cribl Edge Workers support Parquet only when running on Linux, not on Windows.
- See Parquet Schemas for pointers on how to avoid problems such as data mismatches.

Automatic schema: Toggle on to automatically generate a Parquet schema based on the events of each Parquet file that Cribl Stream writes. When toggled off (the default), exposes the following additional field:

- **Parquet schema**: Select a schema from the drop-down.

If you need to modify a schema or add a new one, follow the instructions in our Parquet Schemas topic. These steps will propagate the freshest schema back to this drop-down.

Parquet version: Determines which data types are supported, and how they are represented. Defaults to `2.6`; `2.4` and `1.0` are also available.

Data page version: Serialization format for data pages. Defaults to `V2`. If your toolchain includes a Parquet reader that does not support `V2`, use `V1`.

Group row limit: The number of rows that every group will contain. The final group can contain a smaller number of rows. Defaults to `10000`.

Page size: Set the target memory size for page segments. Generally, set lower values to improve reading speed, or set higher values to improve compression. Value must be a positive integer smaller than the **Row group size** value, with appropriate units. Defaults to `1 MB`.

Log invalid rows: Toggle on to output up to 20 unique rows that were skipped due to data format mismatch. Log level must be set to `debug` for output to be visible.

Write statistics: Leave toggled on (the default) if you have Parquet tools configured to view statistics - these profile an entire file in terms of minimum/maximum values within data, numbers of nulls, etc.

Write page indexes: Leave toggled on (the default) if your Parquet reader uses statistics from Page Indexes to enable page skipping. One Page Index contains statistics for one data page.

Write page checksum: Toggle on if you have configured Parquet tools to verify data integrity using the checksums of Parquet pages.

Metadata (optional): The metadata of files the Destination writes will include the properties you add here as key-value pairs. For example, one way to tag events as belonging to the OCSF category for security findings would be to set **Key** to `OCSF Event Class` and **Value** to `2001`.

Advanced Settings

File size limit (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

File open time limit (sec): Maximum amount of time to write to a file. Files open for longer than this limit will be closed and moved to final output location. Defaults to `300`.

Idle time limit (sec): Maximum amount of time to keep inactive files open. Files open for longer than this limit will be closed and moved to final output location. Defaults to `30`.

Open file limit: Maximum number of files to keep open concurrently. When exceeded, the oldest open files will be closed and moved to final output location. Defaults to `100`.

Cribl Stream will close files when **either** of the `File size limit (MB)` or the `File open time limit (sec)` conditions is met.

Concurrent file parts limit: Maximum number of parts to upload in parallel per file. A value of `1` tells the Destination to send the whole file at once. When set to `2` or above, IAM permissions must include those required for multipart uploads. Defaults to `4`; highest allowed value is `10`.

Disk space protection: Specifies whether to `Block` (default) or `Drop` incoming events when the disk space falls below the globally defined **Min free disk space** amount.

Add Output ID: When toggled on (default), adds the **Output ID** field's value to the staging location's file path. This ensures that each Destination's logs will write to its own bucket.

For a Destination originally configured in a Cribl Stream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this toggle's state. This is to avoid losing any files pending in the original staging directory, upon Cribl Stream upgrade and restart. To enable this option for such Destinations, Cribl's recommended migration path is:

- Clone the Destination.
- Redirect the Routes referencing the original Destination to instead reference the new, cloned Destination.

This way, the original Destination will process pending files (after an idle timeout), and the new, cloned Destination will process newly arriving events with **Add output ID** enabled.

Remove empty staging directories: When toggled on (the default), Cribl Stream deletes empty staging directories after moving files. This prevents the proliferation of orphaned empty directories. When enabled, exposes this additional option:

- **Staging cleanup period**: How often (in seconds) to delete empty directories when **Remove staging dirs** is enabled. Defaults to `300` seconds (every 5 minutes). Minimum configurable interval is `10` seconds; maximum is `86400` seconds (every 24 hours).
- **Directory batch size**: Specifies how many directories are scanned per batch during automatic cleanup of empty staging directories. Set between `10` and `10000`, defaults to `1000`. Higher values speed up cleanup but increase memory usage.

Enable dead-lettering: Toggle this on to set a maximum number of retries, and to move files to a designated directory when write failures exceed that limit. This prevents data flow blockage and excessive error logging due to undeliverable files. When enabled, exposes two additional fields:

- **Dead-letter location**: Specify the storage location for undeliverable files. Defaults to `$CRIBL_HOME/state/outputs/dead-letter`.
- **Maximum retry limit**: Configure the retry limit for failed file deliveries. This setting defines how many times the system will attempt to move a file to its intended location before it is deemed undeliverable and placed in the dead-letter directory. Defaults to `20`.

Force close on shutdown: Toggle on to force all staged files to close during an orderly shutdown. This triggers immediate upload or finalization of any in-progress files, regardless of **Idle time**, **File open time**, or **File size** limits. Use this option to minimize data loss risk in ephemeral or auto-scaling environments. This could result in a higher number of small files, as files are closed before reaching their configured thresholds. Ensure your Worker Group or Fleet **Drain timeout** is set high enough to allow all staged data to be finalized before shutdown completes.

Region: Region where the MinIO service/cluster is located. Leave blank when using a containerized MinIO.

Object ACL: ACL (Access Control List) to assign to uploaded objects. Defaults to `Private`.

Storage class: Select a storage class for uploaded objects. Defaults to `Standard`.

Server-side encryption: Server side encryption type for uploaded objects. Defaults to `none`.

Signature version: Signature version to use for signing MinIO requests. Defaults to `v4`.

Reuse connections: Whether to reuse connections between requests. Toggling on (default) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (for example, self-signed certificates). Defaults to toggled on.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

IAM Permissions

The following permissions are always needed to write to an Amazon S3-compatible object store:

- `s3:ListBucket`
- `s3:GetBucketLocation`
- `s3:PutObject`

If your Destination needs to do multipart uploads to S3, two more permissions are needed:

- `kms:GenerateDataKey`
- `kms:Decrypt`

See the AWS documentation.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Common Issues

"Parse Error: Expected HTTP/"

The Worker is trying to use HTTP, but the server is expecting HTTPS.

Parse Error: Expected HTTP

"Bucket does not exist - self signed certificate"

Example Error Text:

```
message: Bucket does not exist - self signed certificate
stack: Error: self signed certificate
    at TLSSocket.onConnectSecure (node:_tls_wrap_1535:34)
    at TLSSocket.emit (node:events:513:28)
    at TLSSocket.emit (node:events:489:12)
    at TLSSocket._fini... Show more
tmpPath: /opt/cribl/s3_bucket/cribl/2023/03/28/CriblOut-XTNGh5.0.json.tmp
```

Copy

Can occur when:

- the Destination uses a self-signed cert that has not yet been trusted by Cribl Stream.
- the user does not have appropriate permissions to view the bucket.
- the authentication token generated by your computer has a timestamp that is out of sync with the server's time, resulting in an authentication failure.

Recommendation

Disable **Advanced Settings** > **Reject unauthorized certificates** or; get the sender's cert and add it to the `NODE_EXTRA_CA_CERTS` path for validation; or verify that the user has appropriate permissions.

8.2.28. NetFlow Destination

The NetFlow Destination exclusively forwards NetFlow v5 and v9 UDP traffic to NetFlow collectors. The NetFlow Destination relies on the `__netflowRaw` field that is generated from a Cribl Stream NetFlow Source. The raw NetFlow packet data is sent directly to the configured NetFlow collector(s). Events without `__netflowRaw` are discarded.

Type: Non-Streaming | TLS Support: No | PQ Support: Yes

Requirements

- **Enable pass-through**: The NetFlow Source must have **Enable pass-through** toggled on to generate events containing `__netflowRaw`.
- **Routing**: Ensure only events with `__netflowRaw` are routed to the NetFlow Destination.

Raw Forwarding

For both NetFlow v5 and v9, Cribl Stream:

- Can forward NetFlow packets to other NetFlow collectors. However, it cannot modify the contents of the incoming packet. In other words, Cribl Stream forwards the NetFlow/IPFIX export payload as it was received, without modification to the flow records themselves. The original transport headers are not preserved, so the packet is not an exact byte-for-byte copy of the original network packet.
- Only routes NetFlow packets from upstream Exporters and cannot generate its own NetFlow packets.
- Cannot send non-NetFlow input data to NetFlow collectors.

Beyond Raw Forwarding

While the NetFlow Destination relies solely on `__netflowRaw`, other fields such as `srcAddr` and `packets` generated by the NetFlow Source are used for:

- **Non-NetFlow Destinations**: Providing structured, human-readable data for systems like Splunk or Amazon S3.
- **Analytics**: Supporting filtering, enrichment, and aggregation within Cribl Stream.
- **Internal processing**: Enabling Pipeline logic and routing decisions.
- **Validation**: Assisting in debugging and verifying parsed NetFlow data.

Configure a NetFlow Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this NetFlow definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **NetFlow Destinations**: Add the downstream NetFlow collectors to which Cribl Stream should send data.
        - **Address:** Hostname or IP address of the NetFlow collector.
        - **Port:** Port number to connect to on the NetFlow collector. Defaults to `2055`, which is the standard port for NetFlow traffic.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Processing and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

Advanced Settings

DNS resolution period (sec): Re-resolve any hostnames after each interval of this many seconds, and pick up destinations from records. Defaults to `0` seconds. A value of `0` means every datagram sent will incur a DNS lookup. A non-zero value improves performance but can reduce the overall reliability if the DNS records for the downstream NetFlow collectors change frequently.

Enable Source IP spoofing: This feature is available only for on-prem or hybrid Worker Groups (not available in Cribl.Cloud or on Cribl Edge Nodes). Toggle on to use the event Source IP and port for outgoing UDP packets.

This field preserves the event's original IP address and port from the internal `__srcIpPort` field, rather than the Worker Process's IP. This is useful when sending data to systems that rely on the Source IP to identify the original sender. For the prerequisites, see `udp-sender` Helper Installation and Setup.

Maximum transmission unit (MTU): Displayed when **Enable Source IP spoofing** is enabled. Sets the maximum size of NetFlow records. Defaults to `1500`. To avoid packet fragmentation, keep this value less than or equal to the MTU (maximum transmission unit) defined for `udp-sender`, but make sure it is large enough for data to fit in one packet. When the record length exceeds the MTU, the packet will be dropped.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

`udp-sender` Helper Installation and Setup

Enabling IP spoofing requires creating raw network sockets, which demands elevated system privileges. To maintain the Cribl Stream security model, this operation is isolated into a dedicated helper program. IP spoofing is not available on Cribl Edge.

To enable IP spoofing, complete the following installation and privilege configuration steps on every Worker Node:

1. Download the `udp-sender` helper binary. The source code and releases are available on the public GitHub repository: https://github.com/criblio/udp-sender/.
2. Install the executable binary at the required path: `/usr/bin/udp-sender`
3. Set permissions: A system administrator with root access must grant the binary the necessary Linux capability (`CAP_NET_RAW`). This allows the helper to construct and send raw network packets. Use the following command to set the capability:
   ```
   sudo setcap 'cap_net_raw+eip' /usr/bin/udp-sender
   ```
4. If you are using package installers, the installation script will automatically create a custom `udp-senders` group and set the necessary capabilities on the binary using the `setcap` command. You will need to manually ensure the `cribl` user is added to this group.

If the `udp-sender` helper fails to start (for example, due to missing permissions), the Destination will enter into an error state. Check the Worker Process logs for an error message indicating a failure to create a raw socket, which confirms the `CAP_NET_RAW` capability has not been correctly applied to the `/usr/bin/udp-sender` file.

For additional technical details on the binary's function and installation, consult the README in the public GitHub repository.

Internal Fields

The NetFlow Destination forwards the `__netflowRaw` field that is generated from a Cribl Stream NetFlow Source to downstream NetFlow collectors.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.29. OpenTelemetry (OTel) Destination

Cribl Stream supports sending events to OpenTelemetry Protocol (OTLP)-compliant targets. (Cribl Stream can receive OTel events through the OTel Source.) Besides native OTel Trace, Metric, and Log events, you can send Cribl Stream's Gauge metric events through this OTel Destination.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Protocol and Transport Support

This Destination supports either of the transports that OTLP specifies: gRPC or HTTP. OTLP defines Protocol buffer (Protobuf) schemas for its payloads (requests and responses). With the HTTP transport, this Destination supports Binary Protobuf payload encoding, but does not support JSON Protobuf.

When configuring Pipelines (including pre-processing and post-processing Pipelines), you need to ensure that events sent to this Destination conform to the relevant Protobuf specification. Links below are for OTLP 1.3.1:

- For traces, opentelemetry/proto/trace/v1/trace.proto
- For metrics, opentelemetry/proto/metrics/v1/metrics.proto
- For logs, opentelemetry/proto/logs/v1/logs.proto

The OTel Destination will drop non-conforming events. If the Destination encounters a parsing error when trying to convert an event to OTLP, it discards the event and Cribl Stream logs the error.

If sending data to a downstream receiver that accepts OTLP 1.3.1 log events, you must select **Advanced Settings** > **OTLP version** > `1.3.1`. Otherwise, the Destination will send log events formatted for OTLP `0.10.0`. For context, see Breaking Changes Between OTLP 0.19.0 and 1.3.1.

Configure Cribl Stream to Output to OTel

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.

2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this OTel output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **OTLP version**: The drop-down offers `0.10.0` (default) and `1.3.1`.
    - **Protocol**: Use the drop-down to choose the protocol to use when sending data: `gRPC` (the default), or `HTTP`. Additional settings display in the Advanced for both gRPC and HTTP.

      When set to `HTTP`, the UI adds the Retries section (left tab) and displays three additional settings below:
    - **Traces endpoint override**: By default, the Destination will send traces to `<endpoint>/v1/traces`, where `<endpoint>` is what you specified for the **Endpoint** setting above. To send traces to a different endpoint, enter that endpoint here.
    - **Metrics endpoint override**: By default, the Destination will send metrics to `<endpoint>/v1/metrics`, where `<endpoint>` is what you specified for the **Endpoint** setting above. To send metrics to a different endpoint, enter that endpoint here.
    - **Logs endpoint override**: By default, the Destination will send logs to `<endpoint>/v1/logs`, where `<endpoint>` is what you specified for the **Endpoint** setting above. To send logs to a different endpoint, enter that endpoint here.
    - **Endpoint**: Where to send events, in any of a variety of formats (FQDN, PQDN, IP address and port, etc). Supports both IPv4 and IPv6 - IPv6 addresses must be enclosed in square brackets. The same endpoint is used for both Traces and Metrics. If no port is specified, we normally default to the standard port for OTel Collectors, 4137 - however, if TLS is enabled or the endpoint is an HTTPS-based URL, we default instead to port 443.

      To proxy outbound HTTP/S requests, see System Proxy Configuration.
    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
3. Optionally, you can adjust the Authentication, TLS, Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
4. Select **Save**, then **Commit & Deploy**.

Authentication

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Enter the bearer token that must be included in the authorization header. Since OpenTelemetry runs over gRPC, authorization headers are sent as Metadata entries which are essentially key-value pairs. For example: `Bearer <your-configured-token>`.
- **Auth token (text secret)**: This option exposes a drop-down in which you can select a stored text secret that references the bearer token described above. A **Create** link is available to store a new, reusable secret.
- **Basic**: This default option displays fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret)**: This option exposes a **Credentials secret** drop-down, in which you can select a stored text secret that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

TLS Settings (Client Side)

TLS is available only when **General Settings** > **Protocol** is set to `gRPC`.

Enabled Default is toggled off. When toggled on:

Validate server certs: Toggle on (default) to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior.

Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output - both metric events, as dimensions; and log events, as labels. Supports wildcards.

By default, includes `cribl_pipe` (Cribl Stream Pipeline that processed the event).

Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

This tab is displayed when **General Settings** > **Optional Settings** > **Protocol** is set to `HTTP`.

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request

Status Code	Action
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

This tab's options depend on whether **General Settings** > **Protocol** is set to the `gRPC` or `HTTP` transport. The common settings directly are displayed for both transports.

Compression: Compression type to apply to messages sent to the OpenTelemetry endpoint. `Gzip` (the default) and `None` are available for both protocols; `Deflate` is available for gRPC only.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30` sec.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` sec.

Environment: Optionally, specify a single Git branch on which to enable this configuration. If this field is empty, the config will be enabled everywhere.

Additional Settings for gRPC

When **General Settings** > **Protocol** is set to `gRPC`, the **Advanced Settings** tab adds the following options.

Connection timeout: Amount of time (milliseconds) to wait for the connection to establish before retrying. Defaults to `10000` (10 sec.).

Keep alive time (seconds): How often the sender should ping the peer to keep the connection alive. Defaults to `30`.

Metadata: Extra information to send with each gRPC request. Click **Add Metadata** to add each item as a **Key-Value** pair. The **Key** field is arbitrary. The **Value** field is a JavaScript expression that is evaluated just once, when this Destination is initialized. If you pass credentials as metadata, Cribl recommends using C.Secret().

Additional Settings for HTTP

When **General Settings** > **Protocol** is set to `HTTP`, the **Advanced Settings** tab adds the following options.

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle this off if you want Cribl Stream to close the connection immediately after sending a request.

Extra HTTP headers: Click **Add Header** to define additional HTTP headers to pass to all events. Each row is a **Name-Value** pair. Values will be sent encrypted.

Safe headers: Add headers here to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly

8.2.30. SentinelOne AI SIEM Destination

The **SentinelOne AI SIEM** Destination sends data to SentinelOne's Singularity AI SIEM platform using the HTTP Event Collector (HEC) protocol.

Type: Streaming | TLS Support: Yes | PQ Support: No

Configure a SentinelOne AI SIEM Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Destination. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Region**: Select the geographical region for your SentinelOne AI SIEM ingestion endpoint. This must match your tenant's deployment to ensure correct data routing and compliance.
        - **US** - `United States: https://ingest.us1.sentinelone.net`
        - **CA** - `Canada: https://ingest.ca1.sentinelone.net`
        - **EMEA** - `Europe, Middle East, and Africa: https://ingest.eu1.sentinelone.net`
        - **AP** - `India: https://ingest.ap1.sentinelone.net`
        - **APS** - `India (South): https://ingest.aps1.sentinelone.net`
        - **AU** - `Australia: https://ingest.apse2.sentinelone.net`
        - **Custom Tenant**: Enter your **Base AI SIEM endpoint URL** in this format:
            - `https://<Your-S1-Tenant>.sentinelone.net`.
            - The URL must begin with `http://` or `https://`, can include a port number, must not have a trailing slash, and match this pattern: `^https?://[a-zA-Z0-9.-]+(:[0-`

9]+)?$.

- **AI SIEM endpoint path**: Select the HTTP API route used to send events to SentinelOne AI SIEM. This choice determines the expected data format and how custom fields are handled.
  - `/services/collector/event` (Default): Select this path to send structured JSON payloads with standard HEC top-level fields. The **Event Fields** tab is available to configure metadata fields. Standard HEC fields are evaluated and set as top-level fields in the outgoing JSON payload. Custom metadata fields are included under the `fields` object.
  - `/services/collector/raw`: Select this path to send raw, unstructured log lines (plain text). The **Raw Fields** tab is available to configure metadata fields. The fields are sent as query parameters in the request URL.

3. Under **Authentication**, select an **Authentication method** from the dropdown:
   - **Manual**: Displays an **AI SIEM API Key** field for you to enter the API key string used to authenticate requests to the SentinelOne AI SIEM endpoint. This key is included in the `Authorization` header as a Bearer token for each HTTP request sent.
   - **Secret**: This option exposes an **AI SIEM API Key (text secret)** drop-down, in which you can select a stored secret that references the API key. A **Create** link is available to store a new, reusable secret.

This Destination does not support Mutual TLS (mTLS).

4. Next, you can configure the following **Optional Settings**:
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Depending on the **AI SIEM endpoint path** you selected, you can configure either **Event Fields** or **Raw Fields**. Fields have a default expression that you can adjust for dynamic data extraction or static assignment of values.

Event Fields are JavaScript expressions and **Raw Fields** are string expressions except for **serverHost** which is a JavaScript expression. In JavaScript expressions, you must enclose text constants in single quotes, like `'security'`. Enter string expressions directly without quotes, spaces are allowed.

Field	Event Default	Raw Default
serverHost	`__e.host || C.os.hostname()`	`C.os.hostname()`

Field	Event Default	Raw Default
logFile	`__e.source || (__e.__criblMetrics ? 'metrics' : 'cribl')`	`cribl`
parser	`__e.sourcetype || 'dottedJson'`	`hecRawParser`
dataSource.category	`'security'`	`security`
dataSource.name	`__e.__dataSourceName || 'cribl'`	`cribl`
dataSource.vendor	`__e.__dataSourceVendor || 'cribl'`	`cribl`
event.type		

6. Optionally, you can adjust the Processing, Retries and Advanced settings outlined in the sections below.
7. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10

seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by

manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure

and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`. Each request can potentially hit a different HEC receiver.

Body size limit (KB): Maximum size, in KB, of the request body. Defaults to `5120`. Lowering the size can potentially result in more parallel requests and also cause outbound requests to be made sooner.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `5`.

- Retries happen on this flush interval.
- Any HTTP response code in the `2xx` range is considered success.
- Any response code in the `5xx` range is considered a retryable error, which will not trigger persistent queue (PQ) usage.
- Any other response code will trigger PQ (if PQ is configured as the Backpressure behavior).

Extra HTTP headers: Select **Add Header** to add **Name**/**Value** pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new connection will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If the server does not support keepalives - or if the server closes a pooled connection while idle - a new connection will be established for the next request.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly

8.2.31. SentinelOne DataSet Destination

In a future release we're deprecating the SentinelOne DataSet Destination in favor of the SentinelOne AI SIEM Destination. The SentinelOne AI SIEM Destination uses SentinelOne's latest integration endpoints, offering enhanced flexibility and functionality.

Cribl Stream can send log events to the SentinelOne/Scalyr DataSet platform via the DataSet API. This Destination sends batches of events, as JSON, to that API's addEvents endpoint.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output to DataSet

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this output definition.
    - **Description**: Optionally, enter a description.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Displays a field for you to enter an **API key** that is available in your DataSet profile.
    - **Secret**: This option exposes an **API key (text secret)** drop-down to select a stored secret that references an API key. A **Create** link is available to store a new, reusable secret.
    - **API key**: Enter your DataSet API key that has `Log Write Access`.
4. Next, you can configure the following **Optional Settings**:
    - **DataSet site**: Select the US (default), `Europe`, or `Custom` region. If you select `Custom`, enter your custom endpoint URL.
    - **Message field**: Name of the event field that contains the message to send. If not specified, Cribl Stream sends all non-internal fields of events passing through the Destination. For details, see

Message Field.

- **Exclude fields**: Fields to exclude from the event if the **Message field** either is unspecified or refers to an object. Ignored if the **Message field** is a string, number, or boolean. If empty, Cribl Stream sends all non-internal fields. Default exclude fields are `sev`, `_time`, `ts`, and `thread`. We automatically send these fields as metadata of the event, in DataSet's required format. This is to avoid charges for field bytes - metadata bytes do not count toward ingestion.
- **Server/host field**: Name of the event field that contains the server or host that generated the event. Cribl Stream groups events by the value of this field, and gives them a unique session token to conform to the DataSet API. Each group is sent out as a separate batch; therefore, Cribl recommends specifying a field with a low cardinality, to avoid queuing up many different batches at the Destination. If not specified, or not a string, the implicit default value is `cribl_<outputId>`. For more details, see Rate Limiting and Session Management.
- **Timestamp field**: Name of the event field that contains the event timestamp. Cribl Stream sends this value as part of each event's metadata, not as an attribute field on the event.

  Timestamps are automatically converted to a nanosecond-precision string. If an event does not contain the field specified **Timestamp field**, or if the value cannot be converted into a nanosecond-precision string, Cribl Stream assigns a timestamp using the first valid output returned from `ts`, `_time`, or `Date.now()`, in that order.

- **Severity**: Use the drop-down to assign a default value to the `severity` field (which is sent as event metadata, not as an attribute field). Cribl Stream falls back to this value when an event contains no valid `sev` or `__severity` field. For details, see Severity.
- **Backpressure behavior**: Specify whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the Persistent Queue, Processing, Retries and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Message Field

If **Message field** is specified, we follow this logic:

- If an event does not contain the specified field, send the whole event (except internal fields).
- If an event has the specified field, and the field's value is a non-object, send the event in the format: `{ message: <value from event> }`.

- If an event has the specified field, and the field's value is an object, send the event in the format: `{ <all fields from the object> }`.

Severity

DataSet's severity model ranges from `0` least-severe (finest) to `6` most-severe (fatal).

- Where an event's `sev` field contains an integer in this range, Cribl Stream passes it through as the severity.
- Where the `sev` field contains a string matching DataSet's enum (`finest`, `finer`, `fine`, `info`, `warning`, `error`, `fatal`), Cribl Stream converts it to the corresponding integer.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.

Mode	Description
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Whether to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. When enabled, any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. When disabled, all `Retry-After` headers are ignored.

Settings for failed HTTP requests: When you want to automatically retry requests that receive particular HTTP response status codes, use these settings to list those response codes.

For any HTTP response status codes that are not explicitly configured for retries, Cribl Stream applies the following rules:

Status Code	Action
Any in the `1xx`, `3xx`, or `4xx` series	Drop the request
Any in the `5xx` series	Retry the request

Upon receiving a response code that's on the list, Cribl Stream first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

By default, `429 (Too Many Requests)` is the only response code configured for automatic retries. For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: When you want to automatically retry requests that have timed out, toggle this control on to display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).

- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Defaults to toggled on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Max events per request: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Click **Add header** to define **Name**/**Value** pairs to pass as additional HTTP headers.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers here to declare them as safe to log in plaintext. (Sensitive headers like `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Rate Limiting and Session Management

DataSet rate limits incoming data per session, not based on API key or IP address. The rate limit is 10 MB/sec per session, but the recommended limit is 2.5 MB/sec. To exceed this limit, split events into multiple sessions.

In Cribl Stream, the session contains:

- A unique session ID
- An optional `serverHost` describing the data source, defaulting to `cribl_${this.outId}`
- Optional properties like `logfile` and `parser` (not currently configurable)

Events are grouped into batches based on `serverHost`, with each batch having its own session ID and rate limit.

Configuration Tips

- **Default Configuration**: If **Server/host field** is left empty, all events will have `serverHost: cribl_${this.outId}`, sharing the same session ID and rate limit.
- **Increasing Sessions**: Set **Server/host field** to an event field with higher cardinality to create multiple sessions and increase the combined rate limit.
- **Practical Solutions**:
  - **Original Source Server/Host**: Set **Server/host field** to a field describing the original source server/host name.
  - **Worker Process ID**: In a Pipeline, set a field like `worker_id` to `C.env.CRIBL_WORKER_LABEL` and configure **Server/host field** to `worker_id`.
  - **Random Distribution**: Use an **Eval** function to set a field to `Math.floor(Math.random() * 8)` and configure **Server/host field** to this field to divide events into multiple sessions.

Internal Fields

The `__severity` field is included in the severity assignment order, after the `sev` field. The order is `sev`, `__severity`, then the configured default **Severity**.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

8.2.32. ServiceNow Cloud Observability

The ServiceNow Cloud Observability Destination enables seamless integration with ServiceNow's Cloud Observability platform. This Destination routes observability data, including logs, metrics, and traces, to ServiceNow for enhanced monitoring and analysis. To ensure your data conforms to the expected format, refer to the Protocol section below.

To get started, connecting to ServiceNow Cloud Observability is simple. Grab your ServiceNow Cloud Observability access token, pick your preferred protocol and endpoint, then add the token to your Destination settings. Need more detailed instructions or configuration options? See below!

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Configuring a ServiceNow Cloud Observability Destination

In Cribl Stream, set up a ServiceNow Cloud Observability Destination.

1. From the top nav, click **Manage**, then select a **Worker Group** to configure. Next, you have two options:
    - To configure via QuickConnect, click **Routing** then **QuickConnect** (Cribl Stream) or **Collect** (Cribl Edge). Next, click **Add Destination** and from the resulting drawer's tiles, select **ServiceNow Cloud Observability**. Next, click either **Add Destination** or (if displayed) **Select Existing**.
    - To configure via the Routes, click **Data** then **Destinations** (Cribl Stream) or **More** then **Destinations** (Cribl Edge). Select **ServiceNow Cloud Observability** from the list of tiles or the **Destinations** list. Next, click **Add Destination** to open a **New Destination** modal.
2. In the Destination modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **OTLP version**: Already set to `1.3.1` and cannot be changed.
    - **Protocol**: Use the drop-down to choose the protocol to use when sending data:
        - gRPC (default): Uses Protocol Buffers (Protobuf) for serialization.
        - HTTP: Uses Binary Protobuf encoding (does not support JSON Protobuf).
    - **Endpoint**: The endpoint to send data to ServiceNow. Default options are `ingest.lightstep.com:443` (United States) and `ingest.eu.lightstep.com:443` (Europe).
        - You can specify a custom endpoint using either a valid URL or an IP address (IPv4 or IPv6). Enclose IPv6 addresses in square brackets (for example, `[2001:db8:85a3:0000:0000:8a2e:0370:7334]`).

        - To proxy outbound HTTP/S requests, see System Proxy Configuration.
    - **Auth token (text secret)**: Create or select a Cribl stored text secret that contains your ServiceNow Cloud Observability access token, ensuring secure authentication for data transmission.
    - **Optional Settings**:
        - When using `HTTP`, you can customize the path instead of using the default cloud endpoints. This is useful when integrating with a hosted Lightstep satellite.
        - **Backpressure behavior**: Whether to `Block`, `Drop`, or `Persistent Queue` events when all receivers are exerting backpressure.
        - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
3. Depending on the protocol you've selected, you have the following options:
    - **gRPC** provides TLS settings.
    - **HTTP** provides retry options, in addition to custom paths.
4. The **Processing Settings** and **Advanced Settings** provide granular control over how data is handled and transmitted.
    - Processing Settings allow you to define pre-processing and post-processing Pipelines, ensuring that events conform to the relevant Protobuf specifications for traces, metrics, and logs. This includes adding metadata, applying transformations, and conditioning the data.
    - Advanced Settings offer additional configuration options that enhance the flexibility and performance of data transmission.
        - For `gRPC`, this includes connection timeout, keep-alive time, and metadata headers.
        - For `HTTP`, this includes options like custom path overrides for traces, metrics, and logs, as well as settings for validating server certificates and using round-robin DNS. HTTP is simpler to set up and more widely compatible with existing tools and platforms.
5. Click **Save**, then **Commit & Deploy**.
6. Use the **Test** tab in the Destination's configuration modal to validate the configuration and connectivity of your Destination.

Custom HTTP Path

To send data to a custom location, override the default endpoint paths. When the **Protocol** is set to `HTTP`, this Destination sends data to the following endpoints by default:

- Traces: `<endpoint>/traces/otlp/v0.9`
- Metrics: `<endpoint>/metrics/otlp/v0.9`
- Logs: `<endpoint>/v1/logs`

`<endpoint>` refers to the base URL you specified in the **Endpoint** setting above.

To override a default path, provide only the desired path in the corresponding field: **Traces endpoint override**, **Metrics endpoint override**, or **Logs endpoint override**.

TLS Settings (Client Side)

TLS is available only when **General Settings** > **Protocol** is set to `gRPC`.

Use TLS Default is toggled off. When toggled on:

Validate server certs: Toggle on (default) to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output - both metric events, as dimensions; and log events, as labels. Supports wildcards.

By default, includes `cribl_pipe` (Cribl Stream Pipeline that processed the event).

Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.

- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

This tab is displayed when **General Settings** > **Protocol** is set to `HTTP`.

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

This tab's options depend on whether **General Settings** > **Protocol** is set to the `gRPC` or `HTTP` transport.

Compression: Compression type to apply to messages sent to the OpenTelemetry endpoint. `Gzip` (the default) and `None` are available for both protocols; `Deflate` is available for gRPC only.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30` sec.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to `5`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `2048` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` sec.

Auth token name: Name of the header or metadata key that will carry your ServiceNow Cloud Observability access token during data transmission. The default is `lightstep-access-token` and is customizable to match your specific requirements.

Environment: Optionally, specify a single Git branch on which to enable this configuration. If this field is empty, the config will be enabled everywhere.

Additional Settings for gRPC

When **General Settings** > **Protocol** is set to `gRPC`, the **Advanced Settings** tab adds the following options.

Connection timeout: Amount of time (milliseconds) to wait for the connection to establish before retrying. Defaults to `10000` (10 sec.).

Keep alive time (seconds): How often the sender should ping the peer to keep the connection alive. Defaults to `30`.

Metadata: Extra information to send with each gRPC request. Click **Add Metadata** to add each item as a **Key-Value** pair. The **Key** field is arbitrary. The **Value** field is a JavaScript expression that is evaluated just once, when this Destination is initialized. If you pass credentials as metadata, Cribl recommends using C.Secret().

Additional Settings for HTTP

When **General Settings** > **Protocol** is set to `HTTP`, the **Advanced Settings** tab adds the following options.

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle this off if you want Cribl Stream to close the connection immediately after sending a request.

Extra HTTP headers: Click **Add Header** to define additional HTTP headers to pass to all events. Each row is a **Name**-**Value** pair. Values will be sent encrypted. You can also add headers dynamicall,y on a per-event basis, in the `__headers` field.

Safe headers: Add headers here to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Protocol and Transport Support

This Destination leverages the OpenTelemetry Protocol (OTLP) v1.3.1 to send observability data to ServiceNow. It supports both gRPC and HTTP protocols, ensuring flexibility in data transmission. OTLP defines Protocol buffer (Protobuf) schemas for its payloads (requests and responses). With the HTTP transport, this Destination supports Binary Protobuf payload encoding, but currently does not support JSON Protobuf.

When configuring Pipelines (including pre-processing and post-processing Pipelines), you need to ensure that events sent to this Destination conform to the relevant Protobuf specification.

- For traces, opentelemetry/proto/trace/v1/trace.proto
- For metrics, opentelemetry/proto/metrics/v1/metrics.proto
- For logs, opentelemetry/proto/logs/v1/logs.proto

This Destination will drop non-conforming events. If the Destination encounters a parsing error when trying to convert an event to OTLP, it discards the event and Cribl Stream logs the error.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.33. SignalFx Destination

Cribl Stream supports sending events to SignalFx, now rebranded as Splunk Observability Cloud.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output to SignalFx

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this SignalFX definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Realm**: Observability Cloud realm name (for example, `us0`). Required.
3. In the **Authentication** settings, select one of the following options from the menu:
    - **Manual**: Displays an **Auth token** field for you to enter your Observability Cloud API access token. See the Authentication tokens topic in the API Reference.
    - **Secret**: This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the API access token described above. A **Create** link is available to store a new, reusable secret.
4. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the Persistent Queue, Processing, Retries and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request

Status Code	Action
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA - for example, the system's CA.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1` second.

Extra HTTP headers: Click **Add Header** to insert extra headers as **Name/Value** pairs. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Notes About SignalFx

SignalFx has been rebranded as Splunk Observability Cloud, but the API endpoints still have URLs at `signalfx.com`.

For details on integrating with SignalFx, see the Developer Guide. Especially relevant is the Send Traces, Metrics and Events section of the Observability Cloud API Reference.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

8.2.34. SNMP Trap Destination

Cribl Stream supports forwarding of SNMP Traps out.

Type: Streaming | TLS Support: No | PQ Support: No

If you've modified SNMP trap events within the Pipeline, use the SNMP Trap Serialize Function to serialize compliant events into SNMP traps before sending them to an SNMP Trap Destination.

Configure Cribl Stream to Forward to SNMP Traps

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this SNMP Trap definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **SNMP trap destinations**: For each destination configured, enter the destination host **Address** and destination **Port**. **Port**: Defaults to `162`. To specify additional SNMP trap destinations to forward traps to on new rows, select **Add Destination**.
3. Next, you can configure the following **Optional Settings**:
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Processing and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

Advanced Settings

DNS resolution period (sec): Specify the interval (in seconds) to re-resolve hostnames. This reduces the frequency of DNS lookups, improving performance. Use this setting to balance the overhead of DNS lookup calls with the expected frequency of changes in the DNS records. Defaults to `0` seconds, meaning DNS lookups occur for every outgoing trap.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Considerations for Working with SNMP Traps Data

- It's possible to work with SNMP metadata (i.e., we'll decode the packet). Options include dropping, routing, etc. However, packets **cannot** be modified and sent to another SNMP Destination.
- SNMP packets can be forwarded to non-SNMP Destinations (for example, Splunk, Syslog, S3, etc.).
- SNMP packets can be forwarded to other SNMP Destinations. However, the contents of the incoming packet cannot be modified - i.e., we'll forward the packets verbatim as they came in.
- Non-SNMP input data **cannot** be sent to SNMP Destinations.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.35. Sumo Logic Destination

Cribl Stream can send logs and metrics to Sumo Logic over HTTP. Sumo Logic offers a Hosted Collector that supports HTTP Sources to receive data over an HTTP POST request.

To send traces to Sumo Logic you can use Cribl's OpenTelemetry Destination pointing to Sumo Logic's OTLP/HTTP Source.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

How Sumo Logic Handles Data

- When an event contains the internal field `__criblMetrics`, it's sent to Sumo Logic as a metric event. Otherwise, it's sent as a log event.
- Data sent to Sumo Logic needs to be UTF-8 encoded and they recommend a data payload have a size, before compression, of 100 KB to 1 MB.
- Sumo Logic may impose throttling and caps on your log ingestion to prevent your account from using On-Demand Capacity, see Sumo Logic's manage ingestion documentation for details.

Prerequisites

In Sumo Logic, create or retrieve an HTTP Logs and Metrics Source's unique endpoint URL. See Sumo Logic's HTTP Logs and Metrics Source documentation for details. You will need the `Manage Collectors` role capability in Sumo Logic.

After creating the Source in Sumo Logic, the URL associated with the Source is displayed. Copy the endpoint URL so you can enter it when you configure the Cribl Stream Sumo Logic Destination.

Configure a Sumo Logic Destination

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.

2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Sumo Logic Destination definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **API URL**: Enter the endpoint URL of the Sumo Logic HTTP Source. This is provided by Sumo Logic after creating the Source, see prerequisites for details. For example, `https://endpoint6.collection.us2.sumologic.com/receiver/v1/http/<long-hash>`.
3. Next, you can configure the following **Optional Settings**:
    - **Custom source name** and **Custom source category** are unique settings to Sumo Logic. These allow you to override the Sumo Logic HTTP Source's **Source Name** and **Source Category** values with HTTP headers. Alternatively, you can define the `__sourceName` and `__sourceCategory` fields on events to assign a custom value at the event level.

    The remaining configurations are Cribl settings that you'll find across many Cribl Destinations.

    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure.
    - **Data Format**: This drop-down defaults to `JSON`. Change this to `Raw` if you prefer to preserve outbound events' raw format instead of JSONifying them.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, configure any Persistent Queue, Processing, Retries, and Advanced settings outlined in the below sections.

> Requests to Sumo Logic are created in separate batches for each unique combination of **Source Name** and **Source Category** within the event payload. If you have a large number of unique pairs, this could lead to many individual batches being held in memory at any given time. You can increase the **Buffer memory limit** to give more memory allowance for buffering outgoing requests, allowing batches to grow larger before being flushed. The default flushing mechanism would discard smaller batches before they reach an expected size due to reaching the **Buffer memory limit**.

5. Click **Save**, then **Commit & Deploy**.
6. Verify that data is searchable in Sumo Logic. See the Verify Data Flow section below.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination

has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high

as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).

- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `1024` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

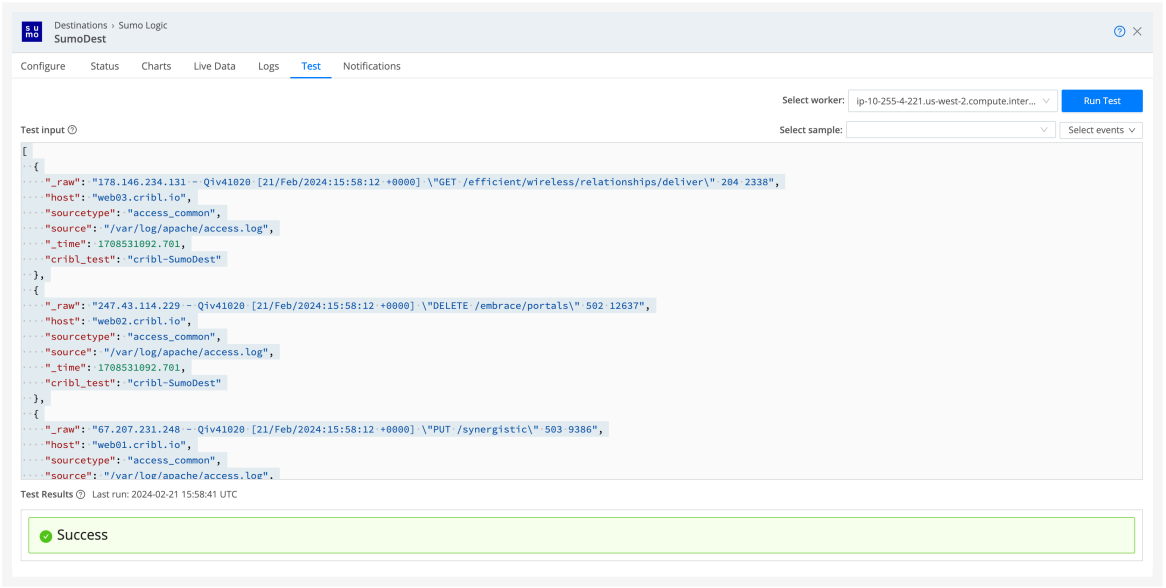
Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Verify Data Flow

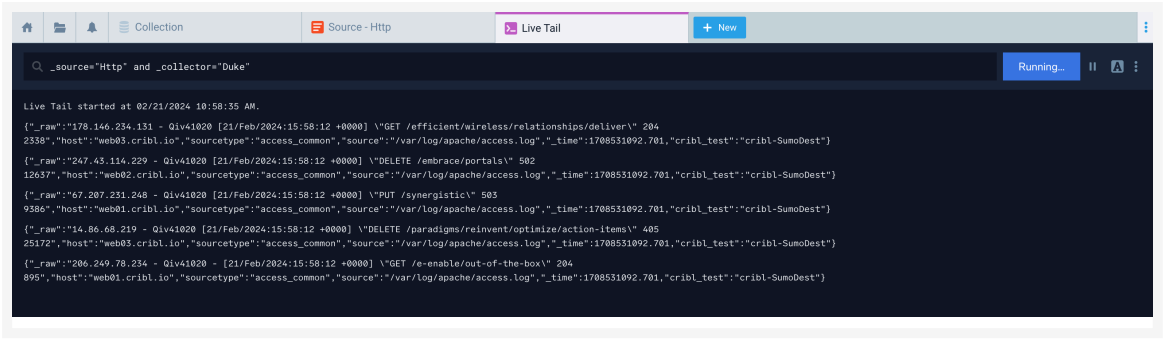
To verify data flow, we'll use the Destination's **Test** feature while running a **Live Tail** session in Sumo Logic.

1. In Sumo Logic, run a new Live Tail session against the name of the HTTP Logs and Metrics Source or the **Custom source name** you defined when you configured the Sumo Logic Destination. You'll need to use the `_source` metadata field. For example, if the name of the Source is `HTTP` your search would be `_source="HTTP"`. Ensure you've clicked the **Run** button to start the Live Tail session.
2. In Cribl Stream, open the Destination configuration modal and select the **Test** tab. You can leave the default data or select from the available samples. Click **Run Test**.



Example Destination Test

3. Look back to your Sumo Logic Live Tail session and you should see the sample data from the Cribl test displayed in your Live Tail results.



Example Live Tail Results

Troubleshooting

If you receive an error when verifying data flow, take note of the response code and reference Sumo Logic's documentation for common issues to investigate.

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new one will be reattempted. This is to prevent sticking to a particular destination when there is a constant flow of events.
- If the server does not support keepalives (or if the server closes a pooled connection while idle), a new

8.2.36. Syslog Destination

Cribl Stream supports sending out data over syslog via TCP or UDP.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

This Syslog Destination supports RFC 3164 and RFC 5424. **Before** you configure this Destination, review Syslog Format Options and Structure Syslog Output below. This will help you ensure that your configuration will structure RFC-compliant outbound events that downstream services can read. Additionally, the Syslog Destination does not compress outbound data. Any compression settings you configure under Persistent Queue affect only how queued data is stored on disk.

Configure Cribl Stream to Output Data in Syslog Format

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Syslog definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Protocol**: The network protocol to use for sending out syslog messages. Defaults to `TCP`; `UDP` is also available.
    - **Load balancing**: Cribl Stream displays this option only when the **Protocol** is set to `TCP`. Toggle on to specify multiple destinations. See Load Balancing Settings below.

    Turning **Load balancing** off displays the following two additional fields:
    - **Address**: Address/hostname of the receiver.
    - **Port**: Port number to connect to on the host.

    If **Load balancing** is disabled and you notice that Cribl Stream is not sending data to all possible IP addresses, enable **Advanced Settings > Round-robin DNS**.

- **Record size limit**: Displayed when **Protocol** is set to `UDP`. Sets the maximum size of syslog messages. Defaults to `1500`, and must be ≤ `2048`. To avoid packet fragmentation, keep this value less than or equal to the MTU (maximum transmission unit) for the network path to the downstream receiver. When the message length exceeds the size limit, it is truncated and followed with an ellipsis ("…").

3. Next, you can configure the following **Optional Settings**:
   - **Facility**: Default value for message facility. If set, will be overwritten by the value of `__facility`. Defaults to `user`.
   - **Severity**: Default value for message severity. If set, will be overwritten by the value of `__severity`. Defaults to `notice`.
   - **App name**: Default value for application name. If set, will be overwritten by the value of `__appname`. Defaults to `Cribl`.
   - **Message format**: The syslog message format supported by the receiver. Defaults to `RFC3164`.
   - **Timestamp format**: The timestamp format to use when serializing an event's time field. Options include `Syslog` (default) and `ISO8601`. For details, see Timestamp format below.
   - **Throttling**: Throttle rate, in bytes per second. Defaults to `0`, meaning no throttling. Multiple-byte units such as KB, MB, or GB are also allowed - for example, `42 MB`. When throttling is engaged, your **Backpressure behavior** selection determines whether Cribl Stream will handle excess data by blocking it, dropping it, or queueing it to disk. For details, see Throttling below.
   - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
   - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.
4. Optionally, you can adjust the Persistent Queue, TLS, Processing, Timeout, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Load Balancing Settings

Load balancing is available only when the **Protocol** is set to `TCP`.

Toggling **Load balancing** on replaces the static **General Settings** > **Address** and **Port** fields with the following controls:

Exclude current host IPs: Appears when **Load balancing** is toggled on. It determines whether to exclude all IPs of the current host from the list of any resolved hostnames. Default is toggled off, which keeps the current host

available for load balancing.

Destinations

Use the **Destinations** table to specify a known set of receivers on which to load-balance data. To specify more receivers on new rows, click **Add Destination**. Each row provides the following fields:

Address: Hostname of the receiver. Optionally, you can paste in a comma-separated list, in `<host>:<port>` format.

Port: Port number to send data to on this host.

TLS: Whether to inherit TLS configs from group setting, or disable TLS. Defaults to `inherit`.

TLS servername: Servername to use if establishing a TLS connection. If not specified, defaults to connection host (if not an IP). Otherwise, uses the global TLS settings.

Load weight: Set the relative traffic-handling capability for each connection by assigning a weight (> `0`). This column accepts arbitrary values, but for best results, assign weights in the same order of magnitude to all connections. Cribl Stream will attempt to distribute traffic to the connections according to their relative weights.

The final column provides an `X` button to delete any row from the table.

For details on configuring all these options, see About Load Balancing.

Timestamp Format

For `Syslog`, the timestamp format is `Mmm dd hh:mm:ss`, using the timezone of the syslog device. For example: `Jan 18 16:56:36`.

For `ISO8601`, the basic timestamp format is `YYYY-MM-DD`. For example: `2023-01-18`. For extended formats, add hours, minutes, seconds, decimal fractions (optional), and timezone. For example: `2023-01-18T16:56:36.996-08:00`.

Throttling

When **Throttling** is used in conjunction with **Connection limit**, the throttling rate is applied per connection. For instance, if **Connection limit** is set to `2` and throttling is set to `3 MB`, each of the two connections can send data at a rate of up to 3 MB per second, resulting in a total potential throughput of 6 MB per second across both connections.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior.

Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

TLS Settings (Client Side)

Enabled: Defaults to toggled off. When toggled on:

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA - for example, the system's CA. Defaults to toggled on.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Timeout Settings

These timeout settings apply only to the TCP protocol.

Connection timeout: Amount of time (in milliseconds) to wait for the connection to establish, before retrying. Defaults to `10000`.

Write timeout: Amount of time (milliseconds) to wait for a write to complete, before assuming connection is dead. Defaults to `60000`.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

The first three options are always displayed:

Octet count framing: When enabled, Cribl Stream prefixes messages with their byte count, regardless of whether the messages are constructed from `message` or `__syslogout`. When disabled, Cribl Stream omits prefixes, and instead appends a `\n` to messages. RFCs 5425 and 6587 describe how octet counting works in syslog.

Enable Source IP spoofing: Cribl Stream displays this option only when **Protocol** is set to `UDP`. This feature is available only for on-prem or hybrid Worker Groups (not available in Cribl.Cloud or on Cribl Edge Nodes). Toggle on to use the event Source IP and port for outgoing UDP packets.

This field preserves the event's original IP address and port from the internal `__srcIpPort` field, rather than the Worker Process's IP. This is useful when sending data to systems that rely on the Source IP to identify the original sender. For the prerequisites, see `udp-sender` Helper Installation and Setup.

Log failed requests to disk: Toggle on to make the payload of the most recently failed request available for inspection. See Inspect Payload to Troubleshoot Closed Connections below.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

The following options are added if you enable the General Settings tab's **Load balancing** option:

DNS resolution period (seconds): Re-resolve any hostnames after each interval of this many seconds, and pick up destinations from A records.

- For TCP: This helps maintain up-to-date IP addresses for your TCP connections. Defaults to `600` seconds.
- For UDP: This reduces the frequency of DNS lookups for UDP destinations, improving performance. Defaults to `0` seconds, meaning DNS lookups occur for every outgoing message.

Load balance stats period (seconds): Lookback traffic history period. Defaults to `300` seconds. If multiple receivers - that is, multiple A records - are behind a hostname, all resolved IPs will inherit the weight of the host unless each IP is specified separately. In Cribl Stream load balancing, IP settings take priority over those from hostnames.

Connection limit: Constrains the number of concurrent receiver connections per Worker Process, to limit memory utilization. If you set this to a number greater than `0`, Cribl Stream will randomly select that many discovered IPs to connect to on every DNS resolution period. Cribl Stream will rotate IPs in future resolution periods - monitoring weight and historical data, to ensure fair load balancing of events among IPs.

`udp-sender` Helper Installation and Setup

Enabling IP spoofing requires creating raw network sockets, which demands elevated system privileges. To maintain the Cribl Stream security model, this operation is isolated into a dedicated helper program. IP spoofing is not available on Cribl Edge.

To enable IP spoofing, complete the following installation and privilege configuration steps on every Worker Node:

1. Download the `udp-sender` helper binary. The source code and releases are available on the public GitHub repository: https://github.com/criblio/udp-sender/.
2. Install the executable binary at the required path: `/usr/bin/udp-sender`
3. Set permissions: A system administrator with root access must grant the binary the necessary Linux capability (`CAP_NET_RAW`). This allows the helper to construct and send raw network packets. Use the following command to set the capability:
   ```
   sudo setcap 'cap_net_raw+eip' /usr/bin/udp-sender
   ```
4. If you are using package installers, the installation script will automatically create a custom `udp-senders` group and set the necessary capabilities on the binary using the `setcap` command. You will need to manually ensure the `cribl` user is added to this group.

If the `udp-sender` helper fails to start (for example, due to missing permissions), the Destination will enter into an error state. Check the Worker Process logs for an error message indicating a failure to create a raw socket, which confirms the `CAP_NET_RAW` capability has not been correctly applied to the `/usr/bin/udp-sender` file.

For additional technical details on the binary's function and installation, consult the README in the public GitHub repository.

Inspect Payload to Troubleshoot Closed Connections

When a downstream receiver closes connections from this Destination (or just stops responding), inspecting the payload of the most recently failed request can help you find the cause. For example:

- Suppose you send an event whose size is larger than the downstream receiver can handle.
- Suppose you send an event that has a `number` field, but the value exceeds the highest number that the downstream receiver can handle.

When **Log failed requests to disk** is enabled, you can inspect the last failed request payload. Here is how:

1. In the Destination UI, navigate to the **Logs** tab.
2. Find a log entry with a `connection error` message.
3. Expand the log entry.
4. If the message includes the phrase `See payload file for more info`, note the path in the `file` field on the next line.

Now you have the path to the directory where Cribl Stream is storing the payload from the last failed request.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to downstream services. Fields for this Destination:

- `__priority`
- `__facility`
- `__severity`
- `__procid`
- `__appname`
- `__msgid`
- `__syslogout`

Syslog Format Options

Unlike other Cribl Stream Destinations, the Syslog Destination applies an additional post-processing step after the Pipeline(s) transform events. This additional step structures the data for compliance with the syslog transport protocol (RFC 3164 and/or RFC 5424) before it is transmitted to downstream services.

- With RFC 3164-compliant messages, you'll get `priority`, `timestamp`, `host`, and `message`, such that `message` includes `appname` and `procid`. You will not get `msgid` or `STRUCTURED-DATA`.
- With RFC 5424-compliant messages - that is, structured data - Cribl Stream will send the contents of `event.structuredData`, whether or not that content is valid according to the RFC. Cribl Stream will make no attempt to frame, enclose, or validate the data. If present, structured data **should** be in the form `[ SD-ID *PARAM-NAME=PARAM-VALUE]` but Cribl Stream will not enforce this. Cribl Stream will send whatever is in that field, or, if the field is missing, send a dash (`-`).

In **General Settings**, you can format the timestamps, format the message delivering the event, and set defaults for **Facility**, **Severity**, and **App name** (in RFC 5424, these are Facility, Severity, and APP-NAME, respectively).

Here is an example of an RFC 3164-compliant syslog event:

- `<13>Jul 11 10:34:35 testbox testing[42]`

Here is an example of an RFC 5424-compliant syslog event:

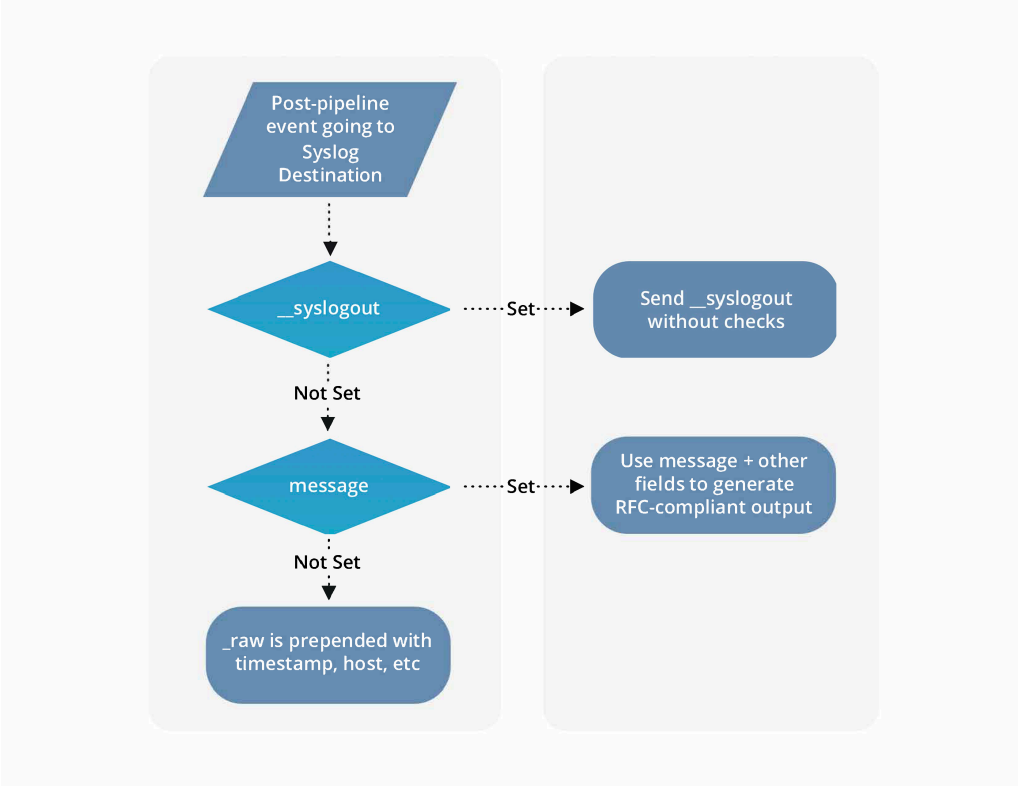
- `<214>1 2022-07-11T18:58:45.000Z testbox testapp 42 - [exampleSDID@32473 iut="3" foo="bar" this="that"] Here is a message`

Structure Syslog Output

Two approaches are available for specifying how the Syslog Destination will structure its output. One relies on the `message` field, and the other relies on the `__syslogout` field. The Syslog Destination determines what to do based on whether or not those fields are set, as the flowchart below makes clear.

- **Use `message` (Simplest Approach)**: Cribl recommends using this option, which requires you to configure `message` and to leave `__syslogout` unset. When `message` is defined, Cribl Stream discards `_raw` and composes a new payload using the other syslog-related fields. Cribl Stream automatically processes the values of the `message`, `_time`, `host`, and other fields, and creates an ISO-compliant timestamp and a properly formatted event. In the flowchart, the rounded rectangle that reads "Use message + other fields ..." represents this approach.
- Export `__syslogout` (Alternative Approach): When you define `__syslogout`, Cribl Stream sends this field and its value as the entire syslog message, discards `_raw`, and ignores all the other fields. In the flowchart, the rounded rectangle that reads "Send __syslogout ..." represents this approach.

As the flowchart shows, if **neither** `message` nor `__syslogout` are set, Cribl Stream uses the existing `_raw` field as the `message` field (with or without a header), and prepends all the other syslog-related fields to the event. While this could be considered a third approach, Cribl does not recommend it, and we will not describe it further in this topic.



Syslog output flow

Before you configure this Destination, review the subsection below that explains the approach that you prefer.

Use `message` (Simplest Approach)

Cribl recommends this approach for environments where events enter Cribl Stream from the Syslog Source, because that Source supplies all required fields.

This approach is the simplest and least error-prone because Cribl Stream creates the payload for you. The following fields are required:

- `facility`
- `severity`
- `_time`
- `host`
- `message`

Cribl Stream will create a new syslog-adherent payload using these required fields. See Field Logic and Behavior below. To specify which RFC your events should adhere to, use **General Settings** > **Optional Settings** > **Message format**.

If these optional fields are included in the event, Cribl Stream will use them appropriately when constructing the payload:

- `appname`
- `host`
- `procid`
- `msgid` (RFC 5424 only)
- `structuredData` (RFC 5424 only)

Field Logic and Behavior

Let's examine the fields in a syslog-formatted event. Keep in mind that if Severity, Facility, or APP-NAME are not set on the event, Cribl Stream defaults to the corresponding value(s) configured in the Syslog Destination (**Severity**, **Facility**, and **App name**, respectively).

- `__priority`: If this field is present Cribl Stream will use it and ignore `facility` and `severity`.
- The version (used for RFC 5424 only, and not a field that you can set) follows priority (with no spaces in between).
- `__facility` or `facility`: Cribl Stream uses this field to calculate priority. The RFCs define Facility codes.
- `__severity` or `severity`: Cribl Stream also uses this field to calculate priority. The RFCs define Severity levels.

Use either (a) `__facility` and `__severity` or (b) `facility` and `severity`. For these fields, avoid mixing double-underscore naming with no-underscore naming, because if you mix them, Cribl Stream will ignore one of them and you will get unintended results.

If `__priority` is not set, Cribl Stream calculates the Priority from the values of Severity and Facility. Cribl Stream uses the formula: Priority = ((8 * Facility) + Severity).

- For example, if the `facility` is `13` (Security) and the `severity` is `2` (Critical), the `priority` will be `(13 * 8) + 2 = 106`.

Now let's look at the rest of the fields. The list below shows field names in priority order: if both `__appname` and `appname` are present, Cribl Stream uses `__appname`.

- `_time`: The value of `_time` in Cribl events is in epoch format, but the syslog RFCs dictate that each event's timestamp must be in human-readable format (see TIMESTAMP in RFC 3164 or RFC 5424). When defining a Syslog Destination, you configure this with **General Settings** > **Timestamp format**. The `ISO8601` option (ISO 8601 format) defines the method for specifying time zone and year, while the older `Syslog` format lacks this information. For this reason, Cribl recommends `ISO8601`.

- `host`: the value for the required host field in a syslog event (see HOSTNAME in RFC 3164 or RFC 5424), following the timestamp.
- `__appname` or `appname`: The required application name (APP-NAME in RFC 5424). This is typically the name of the daemon or process that is logging any given event.
- `__procid` or `procid`: This optional field (PROCID in RFC 5424) is the Process ID. Use a numeric value, optionally surrounded with brackets - for example, `[4321]`. Cribl Stream will automatically adjust the spaces and syntax to ensure RFC-compliant formatting.
- `__msgid` or `msgid`: This optional field (MSGID in RFC 5424 only) is the Message ID.
- `structuredData`: This optional field (STRUCTURED-DATA in RFC 5424 only) contains a namespace and zero or more key-value pairs. Cribl Stream does not validate whether it adheres to the RFC.

Export `__syslogout` (Alternative Approach)

The most common use of this approach is to send to a non-syslog downstream receiver such as a raw TCP listener. While there is no native "raw TCP" Destination in Cribl Stream, this approach - where you configure a Syslog Destination with the TCP protocol and `__syslogout` - can be an effective method for delivering raw data over TCP to a downstream receiver that does not enforce syslog RFCs.

When you set `__syslogout`, that field becomes the entire syslog message sent. Neither `_raw` nor any other metadata are sent downstream.

If you enable **Octet count framing**, Cribl Stream will prepend the number of bytes of the constructed `__syslogout` field to the message before sending it.

Cribl Stream does not check whether the value of `__syslogout` is RFC-compliant. This means that to send proper, RFC-compliant syslog messages, you must manually construct the `__syslogout` payload, starting with `_time`, for all the fields that Cribl Stream would automatically handle with the first approach above.

If the value of `__syslogout` fails to adhere to the syslog RFCs, a downstream syslog server may misinterpret the event; drop non-compliant events entirely; or, try to "fix" the format by supplying missing fields.

8.2.37. TCP JSON Destination

Cribl Stream supports sending data over TCP in JSON format.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Configure Cribl Stream to Output Data in TCP JSON Format

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this TCP JSON definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Load balancing**: Toggle on to specify multiple destinations. For details, see Load Balancing Settings below.
      The following two fields appear **only** when toggled off.
    - **Address**: Hostname of the receiver.
    - **Port**: Port number to connect to on the host.
3. In the **Authentication** settings, select one of the following options from the menu:
    - **Manual**: In the resulting **Auth token** field, you can optionally enter an auth token to use in the connection header.
    - **Secret**: This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the `authToken` header field value described above. A **Create** link is available to store a new, reusable secret.
4. Next, you can configure the following **Optional Settings**:
    - **Exclude current host IPs**: Appears when **Load balancing** is toggled on. It determines whether to exclude all IPs of the current host from the list of any resolved hostnames. Default is toggled off, which keeps the current host available for load balancing.

- **Compression**: Codec to use to compress the data before sending. Defaults to `Gzip`.
- **Throttling**: Throttle rate, in bytes per second. Defaults to `0`, meaning no throttling. Multiple-byte units such as KB, MB, GB, and so forth, are also allowed - for example, `42 MB`. When throttling is engaged, your **Backpressure behavior** selection determines whether Cribl Stream will handle excess data by blocking it, dropping it, or queueing it to disk. For details, see Throttling below.
- **Backpressure behavior**: Specifies whether to block, drop, or queue events when all receivers are exerting backpressure. Defaults to `Block`. For details, see Persistent Queue Settings below.
- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, you can adjust the Persistent Queue, TLS, Processing, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Load Balancing Settings

Enabling the **Load balancing** toggle replaces the static **General Settings** > **Address** and **Port** fields with the following controls:

Destinations

Use the **Destinations** table to specify a known set of receivers on which to load-balance data. To specify more receivers on new rows, click **Add Destination**. Each row provides the following fields:

Address: Hostname of the receiver. Optionally, you can paste in a comma-separated list, in `<host>:<port>` format.

Port: Port number to send data to on this host.

TLS: Whether to inherit TLS configs from group setting, or disable TLS. Defaults to `inherit`.

TLS servername: Servername to use if establishing a TLS connection. If not specified, defaults to connection host (if not an IP). Otherwise, uses the global TLS settings.

Load weight: Set the relative traffic-handling capability for each connection by assigning a weight (> `0`). This column accepts arbitrary values, but for best results, assign weights in the same order of magnitude to all connections. Cribl Stream will attempt to distribute traffic to the connections according to their relative weights.

The final column provides an `X` button to delete any row from the table.

For details on configuring all these options, see About Load Balancing.

Throttling

When used **Throttling** is used in conjunction with **Connection limit**, the throttling rate is applied per connection. For instance, if **Connection limit** is set to `2` and throttling is set to `3 MB`, each of the two connections can send data at a rate of up to 3 MB per second, resulting in a total potential throughput of 6 MB per second across both connections.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

TLS Settings (Client Side)

Use TLS: Defaults to toggled off. When toggled on:

Autofill?: This setting is experimental.

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). Defaults to toggled on.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Timeout Settings

Connection timeout: Amount of time (in milliseconds) to wait for the connection to establish before retrying. Defaults to `10000`.

Write timeout: Amount of time (in milliseconds) to wait for a write to complete before assuming connection is dead. Defaults to `60000`.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Advanced Settings

Log failed requests to disk: Toggle on to make the payload of the most recently failed request available for inspection. See Inspect Payload to Troubleshoot Closed Connections below.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

In General Settings, toggle **Load balancing** on to add the following settings:

DNS resolution period (seconds): Re-resolve any hostnames after each interval of this many seconds, and pick up destinations from A records. Defaults to `600` seconds.

Load balance stats period (seconds): Lookback traffic history period. Defaults to `300` seconds. (Note that If multiple receivers are behind a hostname - i.e., multiple A records - all resolved IPs will inherit the weight of the host, unless each IP is specified separately. In Cribl Stream load balancing, IP settings take priority over those from hostnames.)

Connection limit: Constrains the number of concurrent receiver connections, per Worker Process, to limit memory utilization. If set to a number > `0`, then on every DNS resolution period, Cribl Stream will randomly select this subset of discovered IPs to connect to. Cribl Stream will rotate IPs in future resolution periods - monitoring weight and historical data, to ensure fair load balancing of events among IPs.

Inspect Payload to Troubleshoot Closed Connections

When a downstream receiver closes connections from this Destination (or just stops responding), inspecting the payload of the most recently failed request can help you find the cause. For example:

- Suppose you send an event whose size is larger than the downstream receiver can handle.
- Suppose you send an event that has a `number` field, but the value exceeds the highest number that the downstream receiver can handle.

When **Log failed requests to disk** is enabled, you can inspect the last failed request payload. Here is how:

1. In the Destination UI, navigate to the **Logs** tab.
2. Find a log entry with a `connection error` message.
3. Expand the log entry.
4. If the message includes the phrase `See payload file for more info`, note the path in the `file` field on the next line.

Now you have the path to the directory where Cribl Stream is storing the payload from the last failed request.

Format

TCP JSON events are sent in newline-delimited JSON format, consisting of:

1. A header line. Can be empty, for example: `{}`. If **Auth Token** is enabled, the token will be included here as a field called `authToken`. In addition, if events contain common fields, they will be included here under `fields`.
2. A JSON event/record per line.

See an example in our TCP JSON Source topic.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

You can also view the Monitoring page that provides a comprehensive overview of data volume and rate, helping you identify delivery issues. Analyze the graphs showing events and bytes in/out over time.

8.2.38. Wavefront Destination

Cribl Stream supports sending events to Wavefront analytics.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output to Wavefront

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Wavefront definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Domain name**: WaveFront domain name, for example, `longboard`. Required.
3. In the **Authentication** settings, select one of the following options from the menu:
    - **Manual**: Displays an **API key** field for you to enter your Wavefront API auth token. See Wavefront's Generating an API Token topic.
    - **Secret**: This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the API auth token described above. A **Create** link is available to store a new, reusable secret.
4. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

5. Optionally, configure any Persistent Queue, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request

Status Code	Action
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Toggle on to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA).

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1` second.

Extra HTTP headers: Click **Add Header** to insert extra headers as **Name/Value** pairs. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Proxying Requests

If you need to proxy HTTP/S requests, see System Proxy Configuration.

Notes About Wavefront

For details on integrating with Wavefront, see these Wavefront resources:

- Direct Data Ingestion, and adjacent topics on Wavefront Proxies.
- Wavefront Data Format.

Troubleshooting

The Destination's configuration modal has helpful tabs for troubleshooting:

Live Data: Try capturing live data to see real-time events as they flow through the Destination. On the **Live Data** tab, click **Start Capture** to begin viewing real-time data.

Logs: Review and search the logs that provide detailed information about the delivery process, including any errors or warnings that may have occurred.

Test: Ensures that the Destination is correctly set up and reachable. Verify that sample events are sent correctly by clicking **Run Test**.

8.2.39. Webhook Destination

Cribl Stream can send log and metric events to webhooks and other generic HTTP endpoints.

Type: Streaming | TLS Support: Configurable | PQ Support: Yes

Configure Cribl Stream to Output via HTTP

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this HTTP output definition. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Load balancing**: Toggle on to specify multiple Webhook URLs and load weights. If toggled off and you notice that Cribl Stream is not sending data to all possible IP addresses, enable **Advanced Settings** > **Round-robin DNS**.
    - **Webhook URL**: Endpoint URL to send events to. For details, see Webhook URLs below.
3. Next, you can configure the following **Optional Settings**:
    - **Exclude current host IPs**: Appears when **Load balancing** is toggled on. It determines whether to exclude all IPs of the current host from the list of any resolved hostnames. Default is toggled off, which keeps the current host available for load balancing.
    - **Method**: The HTTP verb to use when sending events. Defaults to `POST`. Change this to `PUT` or `PATCH` where required by the endpoint.
    - **Format**: The format in which to send out events. The choices are: `NDJSON`, `JSON Array`, `Custom`, and `Advanced`. For details, see Format below.
    - **Backpressure behavior**: Whether to block, drop, or queue events when all receivers are exerting backpressure.

- **Tags**: Optionally, add tags that you can use to filter and group Destinations on the **Destinations** page. These tags aren't added to processed events. Use a tab or hard return between (arbitrary) tag names.

4. Optionally, you can adjust the TLS, Authentication, Persistent Queue Settings, Processing, and Advanced settings outlined in the sections below.
5. Select **Save**, then **Commit & Deploy**.

Webhook URLs

Use the **Webhook URLs** table to specify a known set of receivers on which to load-balance data. To specify more receivers on new rows, click **Add URL**. Each row provides the following fields:

Webhook LB URL: Specify the Webhook URL to send events to - for example, `http://webhook:8000/`

Load weight: Set the relative traffic-handling capability for each connection by assigning a weight (`> 0`). This column accepts arbitrary values, but for best results, assign weights in the same order of magnitude to all connections. Cribl Stream will attempt to distribute traffic to the connections according to their relative weights.

The final column provides an `X` button to delete any row from the table.

For details on configuring all these options, see About Load Balancing.

When you first enable load balancing, or if you edit the load weight once your data is load-balanced, give the logic time to settle. The changes might take a few seconds to register.

Format

When sending events to a Webhook Destination, you can choose the format in which they are delivered.

- `NDJSON`: Newline-delimited JSON (the default).
- `JSON Array`: Arrays in JSON-parseable format.
- `Custom`: Events and event batches (payloads) whose format you define using JavaScript **expressions**. See Custom Format below.
- `Advanced`: Events and event batches (payloads) whose format you define using JavaScript **code**. Enables the Webhook Destination to integrate with APIs not otherwise supported in Cribl Stream. See Advanced Format below.

TLS Settings (Client Side)

Enabled Default is toggled off. When toggled on:

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key.

Authentication

Select one of the following options for authentication:

- **None**: Don't use authentication.
- **Auth token**: Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Auth token (text secret)**: This option exposes a **Token (text secret)** drop-down, in which you can select a stored text secret that references the bearer token described above. A **Create** link is available to store a new, reusable secret.
- **Basic**: Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret)**: This option exposes a **Credentials secret** drop-down, in which you can select a stored text secret that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.
- **OAuth**: Configure authentication via the OAuth protocol. Exposes multiple extra options - see OAuth Authentication below.

OAuth Authentication

Selecting **OAuth** exposes the following additional fields:

- **Login URL**: Endpoint for the OAuth API call, which is expected to be a POST call.
- **OAuth secret**: Secret parameter value to pass in the API call's request body.

- **OAuth secret parameter name**: Secret parameter name to pass in API call's request body.
- **Token attribute name**: Name of the auth token attribute in the OAuth response. Can be top-level (for example, `token`); or nested, using a period (for example, `data.token`).
- **Authorize expression**: JavaScript expression used to compute the Authorization header to pass in requests. Uses `${token}` to reference the token obtained from the login POST request, for example: `` `Bearer ${token}` ``.
- **Refresh interval (secs.)**: How often to refresh the OAuth token. Default `3600` seconds (1 hour); minimum `1` second; maximum `300000` seconds (about 83 hours).
- **OAuth parameters**: Optionally, select **Add parameter** for each additional parameter you want to send in the OAuth login request. Cribl Stream will combine the secret with these parameters, and will send the URL-encoded result in a POST request to the endpoint specified in the **Login URL**. We'll automatically add the `Content-Type` header `application/x-www-form-urlencoded` when sending this request.
  In each row of the resulting table, enter the parameter's **Name**. The corresponding **Value** is a JavaScript expression to compute the parameter's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.
- **Authentication headers**: Optionally, select **Add Header** for each custom auth header you want to define and send in the OAuth login request. Stream will automatically add the `Content-Type` header `application/x-www-form-urlencoded` when sending this request.
  In each row of the resulting table, enter the custom OAuth header's **Name**. The corresponding **Value** is a JavaScript expression to compute the header's value. This can also evaluate to a constant. Values not formatted as expressions - for example, `id` instead of `` `${id}` `` - will be evaluated as strings.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

> If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request
`408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a

Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (for example, the system's CA). Defaults to toggled on.

Round-robin DNS: Toggle on to enable round-robin DNS lookup across multiple IP addresses, IPv4 and IPv6. When a DNS server resolves a Fully Qualified Domain Name (FQDN) to multiple IP addresses, Cribl Stream will sequentially use each address in the order they are returned by the DNS server for subsequent connection attempts.

Compress: Toggle on (default) to compress the payload body before sending.

Keep alive: By default, Cribl Stream sends `Keep-Alive` headers to the remote server and preserves the connection from the client side up to a maximum of 120 seconds. Toggle this off if you want Cribl Stream to close the connection immediately after sending a request.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`.

Body size limit (KB): Maximum size of the request body before compression. Defaults to `4096` KB. You can set this limit to as high as 500 MB (`512000` KB). Be aware that high values can cause high memory usage per Worker Node, especially if a dynamically constructed URL (see Internal Fields) causes this Destination to send events to more than one URL. The actual request body size might exceed the specified value because the Destination adds bytes when it writes to the downstream receiver. Cribl recommends that you experiment with the **Body size limit** value until downstream receivers reliably accept all events.

Buffer memory limit (KB): Total amount of memory used to buffer outgoing requests waiting to be sent. If left blank, defaults to 5 times the max body size (if set). If 0, no limit is enforced. This provides granular control over the memory allocated for buffering outgoing batched requests. Increasing the limit allows batches to grow larger before being flushed, improving efficiency for data with high cardinality (a large number of unique batches). Finding the optimal balance between efficient data transfer and memory usage involves adjusting both the **Buffer memory limit** and **Body size limit** settings.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1`.

Extra HTTP headers: Name-value pairs to pass to all events as additional HTTP headers. Values will be sent encrypted. You can also add headers dynamically on a per-event basis in the `__headers` field. See Internal Fields below.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Custom Format

Choosing `Custom` from the **General Settings** > **Format** drop-down exposes the additional fields described in this section. **Source expression** and most of the rest of the controls define the format of individual events - that

is, what data you want to make available from the event. **Batch expression** defines the wrapper object for batched events - that is, how to package events within the request payload. Both expressions **must** be enclosed in backticks.

- **Source expression**: JavaScript expression to evaluate on every event; Cribl Stream will send the result of that evaluation instead of the original event. Sample expression: `` `${fieldA}, ${fieldB}` `` (with literal backticks). Defaults to `__httpOut` - that is, the value of the `__httpOut` field. Use the button at right to open a validation modal.
- **Drop when null**: If toggled on, Cribl Stream will drop events when the **Source expression** evaluates to `null`.
- **Event delimiter**: Delimiter string to insert between events. Defaults to the newline character (`\n`). Cannot be a space (this will be converted to `\n`).
- **Content type**: Content type to use for requests. Defaults to `application/x-ndjson`. Any content types set in **Advanced Settings** > **Extra HTTP headers** will override this entry.
- **Batch expression**: Expression specifying how to format the payload for each batch. Defines a wrapper object in which to include the formatted events, such as a JSON document. This enables requests to APIs that require such objects. To reference the events to send, use the `${events}` variable. An example expression to send the batch inside a JSON object would be: `{"items" : [${events}] }`.

Custom Format Example

Suppose that the API you're sending events to requires request payloads in the following form:

```
{
   "Events":[
      {
         "one":"1",
         "two":"2"
      },
      {
         "one":"1.1",
         "two":"2.2"
      }
   ]
}
```

Copy

And here's how the events coming into your Webhook Destination look - note that the field names need to change, and `field3` and `field4` need to be dropped:

```
{ "field1": 1, "field2": 2, "field3": 3, "field4": 4 }
{ "field1": 1.1, "field2": 2.2, "field3": 3.3, "field4": 4.4 }
```

Copy

To send these events in the required form, you'd do the following:

- Set the **Source expression** to `` `{ "one": "${field1}", "two": "${field2}" }` `` - note that this renames and sends only the desired fields (`one` and `two`).
- Enter the comma (`,`) as your **Event delimiter**.
- Set the **Batch expression** to `` `{ "Events": [${events}] }` ``.

Advanced Format

Choosing `Advanced` from the **General Settings** > **Format** drop-down exposes the JavaScript code blocks described in this section. You can define a format using either one of the code blocks, or both.

- **Format inbound event**: JavaScript code block that receives incoming events (as JavaScript objects) and converts them into strings. Multiple strings are concatenated to form a batch, the batches constituting the payloads of the HTTP requests that the Destination sends to the downstream API.
- **Format outbound payload**: JavaScript code block that receives batches before they are sent out as HTTP request payloads. This gives you the opportunity to modify the payload format from the way it arrives - as a string of concatenated events - to whatever format the downstream API requires.

Code Block Restrictions

Generally speaking, anything forbidden in JavaScript strict mode is forbidden in the code blocks. Specifically, the following are **not allowed**:

- `console`, `eval`, `uneval`, `Function (constructor)`, `Promises`, `setTimeout`, `setInterval`, `global`, `globalThis`, `window`, and `set`.

Code blocks **can** include `for` loops, `while` loops, and JavaScript methods such as `map`, `reduce`, `forEach`, `some`, and `every`. For further details, see Supported JavaScript Options.

Only skilled JavaScript developers should use the Advanced format. This is to avoid unintended results - such as creating infinite loops, or otherwise failing to return - that could needlessly add to your throughput burden.

Advanced Format Syntax

Here are the syntax conventions for manipulating events and payloads within the code blocks.

Format inbound event Syntax

- `__e`: References the event object.
- `__e['fieldName']`: References an individual field in the event object.
- `__e.asJSON(): string`: Method to safely convert the object to JSON by removing internal content that is not relevant for a downstream system. **Always** use this method instead of directly converting the object to JSON string using `JSON.stringify(__e)`. This avoids circular references that can cause `JSON.stringify(__e)` to fail.

Format outbound event Syntax

In addition to the syntax conventions defined above, the **Format outbound event** code block has its own special conventions:

- `__e['__payload']`: Accesses the `payload` attribute of the JavaScript object that the code block receives. The `__payload` attribute contains the actual batch (payload) to be formatted.
- `__e['__payloadOut']`: After you modify the batch (payload) as desired, assign it to this variable. If the `__payloadOut` attribute is not present, the code block sends the payload as-is. See the JSON Array Example below.

Advanced Format Process

The **Format inbound event** and **Format outbound payload** code blocks are populated by placeholder JavaScript code. This code expresses the basic flow you must understand to define an Advanced format. Let's examine the code - then you'll be able to see how defining any given Advanced format is just a matter of manipulating events and payloads within the basic flow.

The function definitions `formatEvent (__e : CriblEvent)` and `formatPayload (__e : CriblEvent)` are shown for reference. Do not include them in your code blocks. You only need to enter the internal JavaScript code.

The Format inbound event Code Block

The event to format appears in the first line, referenced as `__e: CriblEvent`. First, you serialize the event as a string; the simplest way to do that is with the method `e.asJSON()`. Then, you assign it back to the object in a variable named `__eventOut`.

```
formatEvent (__e : CriblEvent) {
  // TODO: Serialize the event to a string here.
  let formattedEvent = __e.asJSON();

  // Assign formatted event to __eventOut.
  __e['__eventOut'] = formattedEvent;
}
```

Copy

At this point, the `formatEvent()` function ensures that each incoming event will be serialized as a string. (In the more elaborate examples below, the format of that string will be manipulated.) Each new event, in string form, is appended to the last, building a longer string that will become a **batch**. How many events can be in a batch is governed by **Advanced Settings** > **Events-per-request limit**.

Once a batch is complete, the next code block can operate on it.

The Format outbound payload Code Block

Now the `formatPayload ()` function receives an event whose `payload` attribute is the batch produced by the previous code block. You assign that attribute to the `formattedPayload` variable. If you want to manipulate the batch, do that here. Once you have transformed the batch, you assign it to the `__payloadOut` variable, and the Webhook Destination sends the batch out as the payload of an HTTP request.

Transforming batches is not required. To send batches out exactly how they leave the **Format inbound event** code block, just leave the **Format outbound payload** code block blank.

```
formatPayload: (__e : CriblEvent) {
  // Access payload to format here
  let formattedPayload = __e['payload']

  // TODO: Format payload here

  // Assign formatted payload to payloadOut
  __e['payloadOut'] = formattedPayload
}
```

Copy

JSON Array Example

Suppose you want to communicate with an API that accepts data in JSON array format.

Use the **Format inbound event** code block to format the events:

```
__e['__eventOut'] = `${__e.asJSON()},` // Serialize to JSON and add trailing comma
```

Copy

Use the **Format outbound event** code block to format the payload:

```
// Remove trailing comma and wrap payload in array
__e['__payloadOut'] = `[${payload.substring(0,payload.length-1)}]`
```

Copy

Elasticsearch Bulk API Example

The Elasticsearch Bulk API requires each event to be formatted with two lines of output. While this cannot be done with the Webhook Destination's other format options, it is doable with Advanced Format. (The Elasticsearch API Source, Elasticsearch Destination and Elastic Cloud Destination **do** work with the Elasticsearch Bulk API.)

Here's how Bulk API events must be formatted, structurally:

```
{ <action>:  { <metadata> } }
{ <serializedEvent> }
```

Copy

Here's an example of an event formatted for Bulk API:

```
{ "index" : { "_index" : "test", "_id" : "1" } }
{ "field1" : "value1", "field2" "value2", "field3" : "value3" }
```

Copy

Use the **Format inbound event** code block to format the events:

```
let time=__e['_time'] || Date.now();
__e['_time'] = undefined;
__e['@timestamp'] = C.Time.strftime(time, '%Y-%m-%dT%H:%M:%S.%LZ');
if(typeof __e['host'] === 'string') {
  const host = __e['__host'] || {};
  host.name = __e['host'];
  __e['host'] = host;
}
__e['__eventOut'] = `{"create":{"_index":"${__e['_index'] || 'cribl'}", "pipeline":"$
```

Copy

Leave the **Format outbound event** code block blank.

Splunk HEC Example

Use the **Format inbound event** code block to format the events:

```
  const eventIn = __e;
  const time = eventIn._time;
  const event = eventIn._raw;
  const {host, source, sourcetype, index} = eventIn;
  eventIn._time = undefined;
  eventIn._raw = undefined;
  eventIn.host = undefined;
  eventIn.source = undefined;
  eventIn.sourcetype = undefined;
  eventIn.index = undefined;
  const fields = {};
  const obj2send = {time, index, host, source, sourcetype, event, fields};
  for (const key of ['index', 'host', 'source', 'sourcetype']) {
    // only convert if the key/val is present and is not a string
    const oldValue = obj2send[key];
    if (oldValue != null && typeof oldValue !== 'string') {
      obj2send[key] = JSON.stringify(oldValue);
    }
  }
  for (let key of Object.keys(eventIn)) {
    if(key.startsWith('__'))
      continue;
    let val = eventIn[key];
    if(val === undefined)
      continue;
    if(typeof val === 'object')
      val = JSON.stringify(val);
    obj2send.fields[key] = val;
  }
__e['__eventOut'] = `${JSON.stringify(obj2send)}\n`;
```

Copy

Leave the **Format outbound event** code block blank.

Internal Fields

Cribl Stream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__criblMetrics`

- `__url`
- `__headers`

If an event contains the internal field `__criblMetrics`, Cribl Stream will send it to the HTTP endpoint as a metric event. Otherwise, Cribl Stream will send it as a log event.

When **Load balancing** is disabled, if an event that contains the internal field `__url`, that field's value will override the **General Settings** > **Webhook URL** value. This way, you can determine the endpoint URL dynamically.

For example, you could create a Pipeline containing an Eval Function that adds the `__url` field, and connect that Pipeline to your Webhook Destination. Configure the Eval Function to set `__url`'s value to a URL that varies depending on a global variable, or on some property of the event, or on some other dynamically-generated value that meets your needs.

If an event contains the internal field `__headers`, that field's value will be a JSON object containing Name-value pairs, each of which defines a header. By creating Pipelines that set the value of `__headers` according to conditions that you specify, you can add headers dynamically.

For example, you could create a Pipeline containing Eval Functions that add the `__headers` field, and connect that Pipeline to your Webhook Destination. Configure the Eval Functions to set `__headers` values to Name-value pairs that vary depending on some properties of the event, or on dynamically-generated values that meet your needs.

Here's an overview of how to add headers:

- To add "dynamic" (a.k.a. "custom") headers to some events but not others, use the `__headers` field.
- To define headers to add to **all** events, use **Advanced Settings** > **Extra HTTP Headers**.
- An event can include headers added by both methods. So if you use `__headers` to add `{ "api-key": "foo" }` and **Extra HTTP Headers** to add `{ "goat": "Kid A" }`, you'll get both headers.
- Headers added via the `__headers` field take precedence. So if you use `__headers` to add `{ "api-`

8.2.40. Wiz Defend

The Wiz Defend Destination can stream data from Cribl Stream to the Wiz Defend cloud security platform over an HTTP Event Collector (HEC)-compatible endpoint.

Type: Streaming | TLS Support: Yes | PQ Support: Yes

Configure Cribl Stream to Output to Wiz Defend

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group. Next, you have two options:
    - To configure via QuickConnect, navigate to **Routing** > **QuickConnect** (Stream) or **Collect** (Edge). Select **Add Destination** and select the Destination you want from the list, choosing either **Select Existing** or **Add New**.
    - To configure via the Routes, select **Data** > **Destinations** or **More** > **Destinations** (Edge). Select the Destination you want. Next, select **Add Destination**.
2. In the **New Destination** modal, configure the following under **General Settings**:
    - **Output ID**: Enter a unique name to identify this Wiz Defend Destination. If you clone this Destination, Cribl Stream will add `-CLONE` to the original **Output ID**.
    - **Description**: Optionally, enter a description.
    - **Wiz Data Center**: Your Wiz deployment data center (for example, `us1`, `us8`, `eu1`). You can overwrite this field and enter your own value.
    - **Wiz Environment**: Your Wiz deployment environment. Options are `Production` or `Government`.
    - **Wiz Connector ID**: The unique identifier for the specific Cribl connector defined in your Wiz Settings. This is used to cross-validate the bearer token and ensure traffic is originating from the authorized integration.
    - **Wiz Defend Source type**: Select a Source type from the dropdown. Options are: `AWS Cloud Trail`, `Azure Activity Logs`, and `GCP Audit Logs`. Cribl sends the selected value in the `WizDefend-Source-Type` HTTP header so Wiz can classify incoming data by Source or workload type.
3. Under **Authentication**, select an **Authentication method** from the dropdown:
    - **Manual**: Displays an **Authentication token** field for you to enter your Wiz Defend token.
    - **Secret**: This option exposes an **Authentication token (text secret)** drop-down, in which you can select a stored secret that references the Wiz Defend token described above. A **Create** link is

available to store a new, reusable secret.

4. Next, you can configure the following **Optional Settings**:
    - **Backpressure behavior**: Select whether to block, drop, or queue events when all receivers are exerting backpressure. (Causes might include a broken or denied connection, or a rate limiter.) Defaults to `Block`.
    - **Tags**: Optionally, add tags that you can use to filter and group Destinations on the
5. Optionally, you can adjust the Persistent Queue, TLS, Processing, Retries, and Advanced settings outlined in the sections below.
6. Select **Save**, then **Commit & Deploy**.

TLS Settings (Client Side)

Enabled: Defaults to toggled off. When toggled on:

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a hostname, not an IP address.

Minimum TLS version: Optionally, select the minimum TLS version to use when connecting.

Maximum TLS version: Optionally, select the maximum TLS version to use when connecting.

Certificate: Select an existing certificate name from the drop-down or add a new certificate by selecting **Create**. See Add a TLS Certificate to a Worker Group for details.

CA certificate path: Path on client containing CA certificates (in PEM format) to use to verify the server's cert. Path can reference `$ENV_VARS`.

By default, this Destination rejects certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (for example, the system's CA). You can control this behavior using the **Validate server certs** advanced setting.

Private key path (mutual auth): Path on client containing the private key (in PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Certificate path (mutual auth): Path on client containing certificates in (PEM format) to use. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.

Passphrase: Passphrase to use to decrypt private key. Often the value of the `sslPassword` or similar parameter in the `outputs.conf` or `server.conf` file.

Single .pem File

If you have a **single** .pem file containing `cacert`, `key`, and `cert` sections, enter it in all of these fields above: **CA certificate path**, **Private key path (mutual auth)**, and **Certificate path (mutual auth)**.

Persistent Queue Settings

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.

This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5` GB. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further optimize Worker startup connections and CPU load in the Worker Processes settings.

Clear Persistent Queue: For Cloud Enterprise only, click this button if you want to delete the files that are currently queued for delivery to this Destination. If you click this button, a confirmation modal appears. Clearing the queue frees up disk space by permanently deleting the queued data, without delivering it to downstream receivers. This button only appears after you define the **Output ID**.

Use the **Clear Persistent Queue** button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

Processing Settings

Post-Processing

Pipeline: Pipeline or Pack to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the Cribl Stream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` - Cribl Stream Node that processed the event.
- `cribl_input` - Cribl Stream Source that processed the event.
- `cribl_output` - Cribl Stream Destination that processed the event.
- `cribl_route` - Cribl Stream Route (or QuickConnect) that processed the event.
- `cribl_wp` - Cribl Stream Worker Process that processed the event.

Retries

Honor Retry-After header: Toggle on to honor a `Retry-After` header, provided that the header specifies a delay no longer than 180 seconds. Cribl Stream/Edge limits the delay to 180 seconds even if the `Retry-After` header specifies a longer delay. Any `Retry-After` header received takes precedence over all other options configured in the **Retries** section. Toggle off to ignore all `Retry-After` headers.

Default Retry Behavior: By default, Cribl Stream/Edge applies predefined retry strategies for certain failed requests to the Destination using an exponential backoff algorithm. You can override these settings by manually configuring the retry behavior for a particular error code.

Destinations created before the 4.16.0 release do not support the default `4xx` series retry behavior. If your Destination was created prior to this release and you'd like it to support the behavior, reconfigure the Destination.

To retry requests, Cribl Stream/Edge applies the following rules by default:

Status Code	Action
Any in the `1xx`, `3xx` series	Drop the request

Status Code	Action
`401`, `403`, `408`, `429` errors	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached
Any other `4xx` series	Drop the request
Any in the `5xx` series	Retry in memory or persistent queue if enabled, and initiate backpressure after limit is reached

By default, any `429 (Too Many Requests)` response that is not listed in the retry settings is treated as a non-retryable `4xx` error and dropped. If you want `429` responses retried, ensure `429` is included in the **Settings for failed HTTP requests**.

Settings for failed HTTP requests: You can also manually configure Cribl Stream/Edge to automatically retry requests that receive particular HTTP response status codes.

Upon receiving a response code that's on the list, Cribl Stream/Edge first waits for a set time interval called the **Pre-backoff interval** and then begins retrying the request. Time between retries increases based on an exponential backoff algorithm whose base is the **Backoff multiplier**, until the backoff multiplier reaches the **Backoff limit (ms)**. At that point, Cribl Stream/Edge continues retrying the request without increasing the time between retries any further.

If the sender (which manages the connection to the Destination) is at capacity, it will not accept any incoming events. These incoming events originate internally from a previous stage of the data flow when Destinations send outbound requests to their respective external services, and they include retry requests and new requests. Any events that were already in transit when the sender reached capacity will continue to be processed downstream.

Sender capacity is freed up when an outgoing request succeeds or encounters a non-retryable error. When the sender has available capacity again, it will resume accepting incoming events. This capacity management is influenced by the number of active connections and configured limits, such as concurrency and buffer sizes. If a Pipeline sends events faster than the Destination can process, the buffers may fill up, leading to backpressure and `Sender at capacity` warnings. This backpressure prevents the sender from accepting additional requests until capacity is restored.

For each response code you want to add to the list, select **Add Setting** and configure the following settings:

- **HTTP status code**: A response code that indicates a failed request, for example `429 (Too Many Requests)` or `503 (Service Unavailable)`.
- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.

- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Retry timed-out HTTP requests: Toggle on to automatically retry requests that have timed out and display the following settings for configuring retry behavior:

- **Pre-backoff interval (ms)**: The amount of time to wait before beginning retries, in milliseconds. Defaults to `1000` (one second).
- **Backoff multiplier**: The base for the exponential backoff algorithm. A value of `2` (the default) means that Cribl Stream/Edge will retry after 2 seconds, then 4 seconds, then 8 seconds, and so on.
- **Backoff limit (ms)**: The maximum backoff interval Cribl Stream/Edge should apply for its final retry, in milliseconds. Default (and minimum) is `10,000` (10 seconds); maximum is `180,000` (180 seconds, or 3 minutes).

Advanced Settings

Validate server certs: Reject certificates that are not authorized by a trusted CA (for example, the system's CA). Defaults to toggled on.

Compress: Toggle on (default) to compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests per Worker Process. When Cribl Stream hits this limit, it begins throttling traffic to the downstream service. Defaults to `5`. Minimum: `1`. Maximum: `32`. Each request can potentially hit a different HEC receiver.

Body size limit (KB): Maximum size, in KB, of the request body. Defaults to `4096`. Lowering the size can potentially result in more parallel requests and also cause outbound requests to be made sooner.

Events-per-request limit: Maximum number of events to include in the request body. The `0` default allows unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Body size limit**. Defaults to `1`.

- Retries occur on the configured flush interval.
- Any HTTP response code in the `2xx` range is considered success.
- Any response code in the `5xx` range is considered a retryable error, which will not trigger Persistent Queue (PQ) usage.

- `4xx` responses are dropped by default and do not trigger PQ. To retry specific `4xx` codes (for example, `429`), add them explicitly under **Retries**. When configured as retryable, those responses will engage PQ (if PQ is configured as the Backpressure behavior).

Extra HTTP headers: Select **Add Header** to add **Name**/**Value** pairs to pass as additional HTTP headers. Values will be sent encrypted.

Failed request logging mode: Use this drop-down to determine which data should be logged when a request fails. Select among `None` (the default), `Payload`, or `Payload + Headers`. With this last option, Cribl Stream will redact all headers, except non-sensitive headers that you declare below in **Safe headers**.

Safe headers: Add headers to declare them as safe to log in plaintext. (Sensitive headers such as `authorization` will always be redacted, even if listed here.) Use a tab or hard return to separate header names.

Environment: If you're using GitOps, optionally use this field to specify a single Git branch on which to enable this configuration. If empty, the config will be enabled everywhere.

Notes on HTTP-Based Outputs

- To proxy outbound HTTP/S requests, see System Proxy Configuration.
- Cribl Stream will attempt to use keepalives to reuse a connection for multiple requests. After two minutes of the first use, the connection will be thrown away, and a new connection will be reattempted. This is to prevent sticking to a particular Destination when there is a constant flow of events.
- If the server does not support keepalives, or if the server closes a pooled connection while idle, a new

8.3. Manage Backpressure

Backpressure refers to mechanisms that manage the flow of data when production outpaces consumption. Sometimes an upstream Source can cause a backpressure event when Cribl Stream receives a sudden influx of data that it can't immediately process. And sometimes a downstream Destination can cause a backpressure event when the receiver doesn't have the capacity to receive incoming data from Cribl Stream. Backpressure events carry the risk of data loss, system crashes, or other undesirable effects.

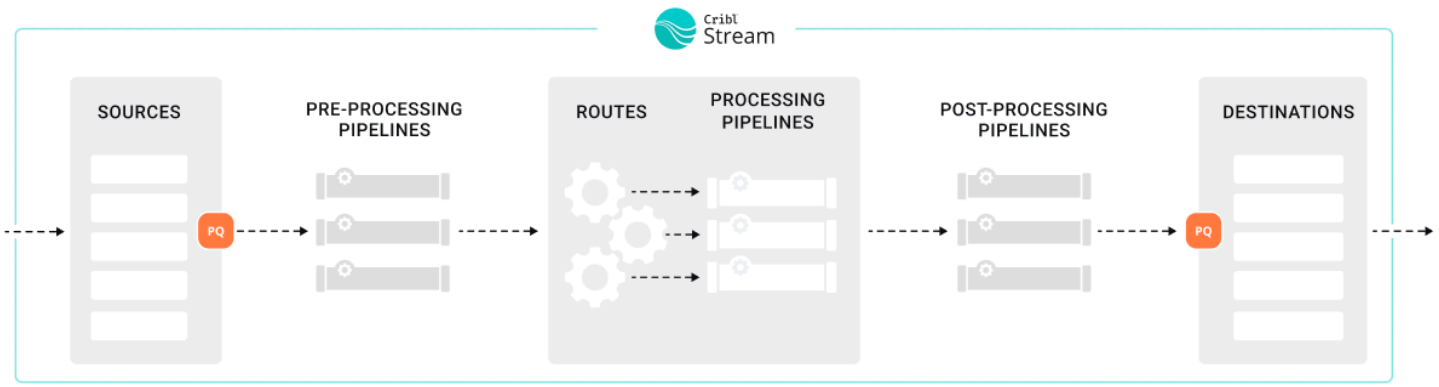
Fortunately, Cribl Stream has built-in methods for responding to backpressure events. It has feedback mechanisms and triggers for detecting backpressure events. Cribl Stream also includes various settings for responding to backpressure events when they occur, such as enabling persistent queueing to store data during backpressure events.

Read the pages in this section to learn more about how Cribl Stream handles backpressure and backpressure configuration options:

- About Persistent Queues
- Optimize Source Persistent Queues
- Optimize Destination Persistent Queues
- About Destination Backpressure Triggers

8.3.1. About Persistent Queues

Cribl Stream's persistent queues (PQ) supplement in-memory queues to minimize data loss during a backpressure event. When a backpressure event occurs, Cribl Stream can engage a backpressure queue to temporarily store data by writing it to disk during the outage. When the outage resolves, Cribl Stream then forwards data from the queue.



You can enable persistent queues at both ends of the data flow

Cribl Stream's persistent queue (PQ) feature helps minimize data loss in these situations:

- A downstream Destination receiver is unreachable.
- A downstream Destination receiver is slow to respond. (This feature is only available in Cribl Stream 4.9.0 and newer.)
- When a configured Source sends a higher volume of data than Cribl Stream can process.

Persistent queues are best in environments that need more durability and reliability. Persistent queues might be a good fit if:

- Your environment regularly experiences outages or backpressure events that last longer than short-term memory queues can sustain, which is typically only a few seconds.
- Your business or organization has a low level of tolerance for data loss.
- You need data from upstream senders that do not provide their own buffering for backpressure queues, such as UDP senders.

You can use persistent queues on:

- Push Sources
- Streaming Destinations

See Supported Sources and Supported Destinations for more information.

The effectiveness of persistent queues to minimize data loss depends on your deployment type, configuration settings, and how much disk space you have allocated for persistent queues. For more information about how to optimize and tune your persistent queue settings, see:

- Optimize Source Persistent Queues
- Optimize Destination Persistent Queues

The persistent queue feature is only available for certain Cribl plans or license types. See the Pricing page for details.

How Persistent Queues Work

The persistent queue feature works differently on Sources than it does on Destinations. For information about how it works, see:

- How Source Persistent Queues Work
- How Destination Persistent Queues Work

When to Use Persistent Queues

When deciding whether to use persistent queueing or not, these factors should guide your decisions:

Cloud or Self-Managed (On-Prem) Deployments

If your environment is a Cribl-managed Cribl.Cloud deployment, Cribl manages the performance and tuning of your cloud deployment. For that reason, there is not a clear downside or tradeoff for enabling persistent queue. Instead, ensure you understand how persistent queues work for both Sources and Destinations and then make a decision about whether to enable it or not. For more information, see:

- Source persistent queue: For Cribl.Cloud Only: Whether to Enable Source Persistent Queue
- Destination persistent queue: For Cribl.Cloud Only: Whether to Use a Hybrid Group

For self-managed (on-prem) deployments, evaluate the additional factors described on this page when making a decision about implementing persistent queues.

Support for Backpressure Queues on Senders and Receivers

For both cloud and on-prem deployments, take some time to learn about the upstream and downstream senders and receivers for your data. Do these senders and receivers support backpressure queues? For example, upstream senders that are UDP-based do not support backpressure queues, but TCP-based Destinations tend to have backpressure queue support.

If the upstream sender or downstream receiver supports backpressure queueing, determine whether to use the native backpressure queue features in those senders and receivers instead of the Cribl Stream persistent queues. In general, you should only use backpressure queues on either Cribl Stream *or* the sender/receiver. Using them both together on both ends of transmission is redundant and can result in data latency issues. However, latency is a natural consequence of queueing, so you need to balance your needs for data resiliency against your needs for timeliness.

What is the difference between a persistent queue and a backpressure queue?

Persistent queues are specifically on-disk storage. Backpressure queues can be either in-memory or on-disk buffers. Most systems have some type of backpressure queueing, which is not quite the same as the persistent queue feature provided by Cribl. When you integrate a sender or receiver to Cribl Stream, ensure you research and understand the options provided by those senders and receivers. Carefully consider how these two systems might interact together.

Likelihood of Outages and Backpressure

For self-managed (on-prem) deployments, consider the overall likelihood of outages or other backpressure events that could result in data loss:

- Does your system regularly experience outages or backpressure?
- What is the duration of these outages or events?
- Do the outages or events often last longer than short-term memory queues can sustain?

If so, then persistent queueing might be the right choice for your system.

Risk Tolerance for Data Loss

For self-managed (on-prem) deployments, determine whether your organization has a high or low risk tolerance for data loss. If they have a high need for data integrity, persistent queueing might be appropriate.

Infrastructure Costs Needed for Persistent Queue

For self-managed (on-prem) deployments, you may need to balance your business requirements against the cost of maintaining the resources and infrastructure necessary to engage persistent queues to prevent data loss. Persistent queues incur costs related to processing, computing, and storage because you have to write the data to disk.

Persistent queues can be resource-intensive for Worker Processes. When enabled, each Worker Process maintains a queue for both Source and Destination persistent queues. Depending on your configuration and the frequency of outages, Worker Processes may allocate significant processing power and compute resources to managing these queues.

Persistent queues also introduce overhead when they engage. Writing to or reading from the disk is slower and demands CPU resources. For instance, using Source persistent queues in Smart Mode can increase CPU overhead if the queue frequently engages and disengages during operation.

Fortunately, performance tuning your persistent queue system could help reduce costs. For more information, see:

- Optimize Source Persistent Queues
- Optimize Destination Persistent Queues

Persistent Queue Limitations and Constraints

Persistent queues:

- Operate at the Worker Process level, with each Worker Process sharing the same configuration settings for a given Source or Destination. These settings are inherited from the respective Source or Destination. The size and engagement model of the queue may vary depending on the specific Source or Destination.
- Have a finite size. If the system cannot deliver data, it might eventually exhaust available disk space.
- Provide limited data protection in cases of application failure. For example, a crash may cause the loss of in-memory data.
- Offer limited data protection in cases of hardware failure, such as disk failure, data corruption, or machine/host loss.

If you turn off persistent queue for a Source or Destination (or shut down the whole Source or Destination) before its queues drain, you will orphan any events still queued on disk.

The following constraints apply to certain Destinations:

- With load-balanced Destinations, persistent queue engages only when *all of the Destination's receivers* are slow or blocking data flow. A single live receiver will prevent persistent queue from engaging on the corresponding Destination. Persistent queue will not engage if at least one outbound TCP socket connection is active, even if the receiver is sending a backpressure signal.
- For Kafka-based Destinations (such as Kafka, Confluent Cloud, Amazon MSK, and Azure Event Hubs), persistent queues engage when Cribl Stream can't reach the brokers.
- The system controls HTTP/S-based Destinations to prevent retries of failed events. However, some events in transit to the Destination may still emerge as duplicates.

Supported Sources

Cribl Stream supports persistent queueing for Push Sources. These sources ingest streaming data, which makes it possible to enable persistent queue.

Persistent queueing is not available for Pull and Collector Sources, which ingest data intermittently and/or from static files.

To determine whether Cribl Stream supports persistent queue for a specific Source, consult the documentation for that Source.

Supported Destinations

Cribl Stream supports persistent queue for:

- HTTP-based Destinations
- TCP Load-Balanced Destinations
- Destinations that write to a single receiver (without load balancing enabled)

To determine whether Cribl Stream supports persistent queue for a specific Destination, consult the documentation for that Destination.

8.3.2. Optimize Source Persistent Queues

Cribl Stream's Source persistent queues can supplement in-memory queues to prevent data loss during a backpressure event, such as when a configured Source sends a higher volume of data than Cribl Stream can process. See Supported Sources for more information about which Sources support persistent queueing.

Read this guide to learn how to optimize your Source persistent queue. This guide is helpful when you are in the initial planning and configuration stage and also when you have an active deployment that you want to performance tune. This guide explains:

- How Source persistent queues respond to a backpressure event.
- The overall goals and key metrics involved in optimizing Source persistent queues.
- An in-depth discussion of the Source persistent queue configuration options and their potential impact on your system.
- The available notifications, monitoring dashboards, and troubleshooting tools for Source persistent queues.

For a general overview of persistent queues, see About Persistent Queues.

> On Cribl-managed Cribl.Cloud Workers with an Enterprise plan, you can enable persistent queue for supported Sources. However, be aware that mode options and storage limits have specific restrictions in Cribl.Cloud. See For Cribl.Cloud Only: `Always On` is the Only Available Mode for more information.

How Source Persistent Queues Work

Backpressure occurs when there is a temporary imbalance between the rates of inbound and outbound data that is either coming in or going out of Cribl Stream.

Different events trigger persistent queues on Sources:

- Error conditions.
- Degraded performance–such as slowdowns or short backups–in downstream Cribl resources.
- External receivers might trigger backpressure.

When Source persistent queue engages, it writes and stores data on a local file system in your environment until the backpressure event resolves.

How Sources Handle Backpressure

Source persistent queue can serve as a stopgap for upstream senders that don't provide their own buffering, or whose small buffers cannot queue for long. This applies to several syslog senders and HTTP-based Sources.

Sources that have persistent queue support have the option to supplement in-memory buffering with persistent queue. When you enable persistent queue, you can choose between these Source persistent queue trigger conditions:

Condition	Description
Always On	Uses persistent queue as a disk buffer for all events.
Smart	Only engages persistent queue upon backpressure from downstream receivers or Cribl internal processes.

In `Always On` mode, the persistent queue constantly writes all data to a backup disk buffer regardless of whether there is a backpressure event or normal conditions. After the backpressure event ends, the persistent queue flushes every 10 seconds when it drains in `Always On` mode.

In `Smart` mode, persistent queue engages under any of these conditions:

- A connected Destination is fully down and Destination-side persistent queue is not enabled.
- A connected Destination is slow or blocking.
- A downstream Cribl Pipeline or Route is slow.

When persistent queue engages on a Source in `Smart` mode:

- Events queue *before* any pre-processing Pipeline takes effect.
- Metrics (data in) will show `0` bytes/events while the system buffers events to the disk queue.
- Cribl Stream Pipelines and Routes process the data with a delay as the persistent queue drains.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Enabling persistent queue on a Source is redundant if the upstream sender already safeguards events in its own disk buffer. Enabling persistent queue on both the sender and Cribl Stream could cause data latency. Check the settings for the sender to ensure you know how whether that specific sender already safeguards against data loss.

If you turn off Source persistent queue (or shut down the whole Source) before its queues drain, you will orphan any events still queued on disk.

Configure Source Persistent Queue

You configure Source persistent queue behavior when you configure each individual Source. Consult the documentation for that Source for more information about initial configuration.

This section provides a configuration quick start for experienced users. However, new Cribl Stream users should consider reading the full guidance in this topic to get a deeper understanding of the decisions and ideal system architecture design for persistent queue.

To configure persistent queue on a Source:

1. In the **Persistent Queue Settings** tab, toggle **Enable persistent queue** off. See Whether to Enable Persistent Queues for Backpressure Events for guidance about making the decision to enable persistent queue.
2. For On-Prem only: Select an appropriate **Mode** from the available options. See When and How to Store Data to Disk During a Backpressure Event for information about which mode to select.

   On Cribl-managed Cribl.Cloud Workers with an Enterprise plan, you can enable persistent queues for supported Sources. By default, Source persistent queue is not enabled, so you must explicitly decide to turn it on.

   When you enable Source persistent queue in Cribl.Cloud, the only available mode is `Always On`, which writes data to disk at all times. `Smart` mode is not available. See For Cribl.Cloud Only: `Always On` is the Only Available Mode for more information.
3. For On-Prem only: Set the **Buffer size limit** to determine the maximum number of events to hold in memory before reporting backpressure to the sender. See When to Report Backpressure to the Sender for more information.
4. For On-Prem only: Set the **Commit frequency** to determine how many events to send downstream before committing that Cribl Stream has read them. See How Many Events to Commit During a Backpressure Event for more information.
5. For On-Prem only: Set the **File size limit** to specify the maximum amount of data stored in each queue file before the file closes and starts a new file. Set the **Queue size limit** to determine the maximum amount of disk space that Cribl allows the queue to consume on each Worker Process. See How Much Data to Store in Memory for more information about these settings.
6. Set the **Queue file path** to determine the location for the persistent queue files. See Where to Store Data and How Much Disk Space to Allocate for more information.

7. Optional: Set the **Compression** type to apply compression to the data before the queue file closes. See Whether to Compress Data for more information.

Overview of Source Persistent Queue Optimization

A well-optimized data throughput system can efficiently and reliably processes and transfer data from Sources to Destinations. Optimized systems maximize performance while minimizing latency, errors, and resource consumption. An optimized system has these characteristics:

- **High efficiency:** Data flows seamlessly at the maximum achievable rate without bottlenecks or excessive resource usage. The system gracefully handles backpressure events, ensuring that data intelligently throttles or queues without overwhelming downstream components.
- **Reliability:** The system consistently handles variations in data volume or speed without losing or corrupting data, even during spikes or failures.
- **Scalability:** The system adapts smoothly to increasing data loads or additional data senders or receivers without degradation in performance.
- **Low latency:** The system processes and delivers data in near real-time, meeting the required time constraints for your business use cases.
- **Cost-efficient resource allocation:** The system uses CPU, memory, disk, and network resources effectively, avoiding over-provisioning or waste.
- **Security and compliance:** The system transmits data securely, meeting all relevant privacy, encryption, and regulatory requirements.

Ultimately, you and your business stakeholders should have confidence in the system's performance, reliability, and ability to adapt to future needs. Persistent queue is an important tool to achieve this confidence.

Persistent Queue Optimization Metrics

The most effective way to performance tune is to understand which key metrics give insights into your persistent queues system's overall health. Effective system administrators study their deployments to establish a performance baseline that they can use for comparison if a performance issue arises. Most system administrators understand their system well and can easily identify peak data periods.

Pay attention to these metrics:

- **CPU utilization:** When your system gets sudden increases in data loads, you might notice that your Worker Processes operate at full capacity and use all their CPU. Generally, that's a sign of inefficiency or the need for more resources. If needed, you can teleport into a single Worker and view all the processes and their individual CPU utilization. To see your CPU utilization, you can use the system monitoring tools to analyze CPU utilization. If needed, you can also teleport into a single Worker and view all the processes and their individual CPU utilization.

- **Data latency:** Some persistent queues can add latency because of the amount of overhead time it takes to write data to disk. As a starting point, assume that each event incurs a few milliseconds of added latency. Be aware that data latency can vary by file system, such as whether you are using a local file system or a Network File System (NFS). NFS file systems tend to introduce higher and more variable latency. To measure the latency impact of your persistent queue system, compare the throughput times of your system without persistent queues enabled and then introduce persistent queue and measure the difference.
- **Memory:** When your system experiences sudden increases in data, your Workers need memory to store that data in temporary buffers. Consider what should happen in your system when memory is fully utilized.
- **Data storage:** When you back up data to a disk, that requires available disk space to use. Allocating memory and disk space can potentially be expensive. Be aware of the rate at which you consume disk memory and the general costs you estimate you will incur to write data to disk. Have systems in place for knowing when disk space is running out and adding more memory as needed.
- **Data throughput:** You need to understand your system's throughput under normal conditions so that you can identify potential bottlenecks in the system. Use the system monitoring tools to develop this baseline.

Key Decisions When Configuring Source Persistent Queues

You configure Source persistent queue behavior when you configure each individual Source. Consult the documentation for that Source for more information about initial configuration.

This section explains the decision-making process you should consider when implementing persistent queue on a Source. It explains the benefits and tradeoffs of different configuration settings for most use cases.

Whether to Enable Persistent Queues for Backpressure Events

The first decision you need to make is whether to use persistent queue to respond to backpressure. See When to Use Persistent Queues for a detailed explanation of which kinds of environments benefits from persistent queues.

In general, you should not use persistent queue is you are already using a backpressure mechanism on a upstream sender. Implementing backpressure mechanisms such as persistent queue on both Cribl Stream and the Source is redundant and could cause unnecessary data latency or impact data throughput.

The alternatives to using persistent queue are to block data or to drop events when a backpressure event occurs. Consider blocking events if you need Cribl to return block signals back to an upstream sender. If the sender also supports backpressure response mechanisms, the sender may stop sending data.

Consult the documentation for the sender to ensure you know how that specific sender handles a backpressure signal. In general, TCP-based senders support backpressure and stop sending data once Cribl Stream stops sending TCP acknowledgments back to it.

When and How to Store Data to Disk During a Backpressure Event

Once the backpressure queue behavior engages, you need to determine how it should write data to disk. Source persistent queues have two available options (modes) for storing data during a backpressure event:

Mode	Description
Always On	Uses persistent queue as a disk buffer for all events.
Smart	Only engages persistent queue upon backpressure from downstream receivers or Cribl internal processes.

The default is **Always On** mode. This mode is helpful if your business has a low tolerance for data loss. However, using **Always On** this mode could have impacts on the persistent queue optimization metrics, such as disk space, throughput, data latency, and CPU utilization.

Considerations for using **Always On** mode:

- This mode can increase memory demands compared to Sources without persistent queue enabled. If memory is a constraint, you can experiment with tuning the **Buffer size limit** setting. However, smaller buffers will reduce the efficiency of disk writes, adding more latency.
- **Always On** mode has higher disk requirements than **Smart** mode. Cribl generally recommends that you allocate each Worker Node enough disk space for at least 1 day's worth of queued throughput. For example: If one Source is sending 1 TB/day to two evenly distributed Worker Processes, you should provision at least 500 GB queue storage for each Worker Process. Expand this basic calculation for additional Sources. From there, expand or contract it based on the typical length of outages you experience.
- Cribl also recommends that you deploy the highest-performance storage available. This prevents IOPS (input/output operations per second) and throughput bottlenecks for both read and write operations.

Considerations for using **Smart** mode:

- `Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.
- If you're using **Smart** mode and observe that it causes Source persistent queue to engage frequently, on option is to reduce the **File size limit** setting. Source persistent queue excessively triggers backpressure only when it reaches the last queued file *and* that file is large. Keeping this threshold low prevents that condition.

Consider experimenting with these modes to determine what is appropriate for your needs. Establish a baseline for system performance before you enable **Always On** and then observe the impacts when you turn

that mode on.

For Cribl.Cloud Only: `Always On` is the Only Available Mode

On Cribl-managed Cribl.Cloud Workers with an Enterprise plan, you can enable persistent queues for supported Sources. By default, Source persistent queue is not enabled, so you must explicitly decide to turn it on.

When you enable Source persistent queue in Cribl.Cloud, the only available mode is `Always On`, which writes data to disk at all times. `Smart` mode is not available.

When you enable Source persistent queue in Cribl.Cloud, Cribl automatically allocates up to 1 GB of disk space per Source, per Worker Process. You cannot change this limit. This 1 GB limit applies to outbound uncompressed data, and the queue does not perform any compression. If the queue fills up, Cribl Stream blocks data flow, including outbound data.

When to Report Backpressure to the Sender

You can use the **Buffer size limit** setting to determine the maximum number of events to hold in memory before reporting backpressure to the sender. Once the number of events reaches this limit, Cribl Stream begins writing the queue to disk. This setting defaults to `1000`.

This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher number of connections. For higher numbers of connections, consider increasing the limit.

How Many Events to Commit During a Backpressure Event

Use the **Commit frequency** setting to determine how many events to send downstream before committing that Cribl Stream has read them. The defaults is `42`.

How Much Data to Store in Memory

When you enable persistent queues, you can configure the **File size limit** setting to specify the maximum amount of data stored in each queue file before the file closes and starts a new file. You can enter a value in units such as KB, MB, and more. If left unspecified, Cribl Stream/Edge defaults to 1 MB.

Additionally, you can use the **Queue size limit** setting to determine the maximum amount of disk space that Cribl allows the queue to consume on each Worker Process. Once the queue reaches this limit, the Source will stop queueing data and engage the configured backpressure mode. You can enter a value in units such as KB, MB, GB, or TB.

If the **Queue size limit** is not specified, it defaults to 5 GB. You can adjust this value to better suit your needs. For Sources with undefined **Queue size limit** values, queues exceeding 5 GB will trigger the configured queue-full behavior until you resize the limit.

To set a higher maximum value, navigate to **Worker Group/Fleet Settings > General Settings > Limits > Storage > Worker Process PQ size limit**. The corresponding key in the `limits.yml` file is `maxPQSize`. (See Config Files: limits.yml for more information.) The default for this setting is 1 TB. Before increasing this limit, consult Cribl Support for guidance.

> The information in this section applies only to on-prem or hybrid Worker Groups. For Cribl-managed Cribl.Cloud Organizations using persistent queue, Cribl automatically allocates up to 1 GB of disk space per Source, per Worker Process. You cannot change this limit.
>
> This 1 GB limit applies to outbound uncompressed data, and the queue does not perform any compression. If the queue fills up, Cribl Stream/Edge blocks data flow, including outbound data.

Whether to Compress Data

When the queue file reaches its maximum file size, you can use the optional **Compression** setting to apply compression to the data before the queue file closes. Currently, the only available compression codec is Gzip.

If you enable **Compression** and also enter a **Queue size limit**, be aware of how these options interact:

- Cribl Stream/Edge currently applies the **Queue size limit** against the **uncompressed** data volume entering the queue.
- With this combination, Cribl recommends that you set the **Queue size limit higher** than the volume's total available disk space, not including compression. This will maximize queue saturation and minimize data loss.

Where to Store Data and How Much Disk Space to Allocate

Use the **Queue file path** setting to determine the location for the persistent queue files. The default is `$CRIBL_HOME/state/queues`. When writing data, Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this path.

For queuing to operate properly, you must provide sufficient disk space. In Distributed mode, this minimum disk space configuration is set on each Worker Group/Fleet. You can access this setting at **Group/Fleet settings** > **General Settings** > **Limits** > **Min free disk space**.

If available disk space falls below this minimum threshold, Cribl Stream/Edge will stop maintaining persistent queues, and data loss will begin. The default minimum is 5 GB.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this value, Cribl Stream/Edge will append `/<worker-id>/<output-id>`.

Source Persistent Queue Notifications

Some Cribl licenses allow you to configure Notifications for Source persistent queues for on-prem deployments. In general, Cribl recommends setting Notifications for these events when using persistent queue to ensure you are aware of important system events. These Notifications always appear in Cribl Stream's user interface and internal logs. You can also send them to external systems.

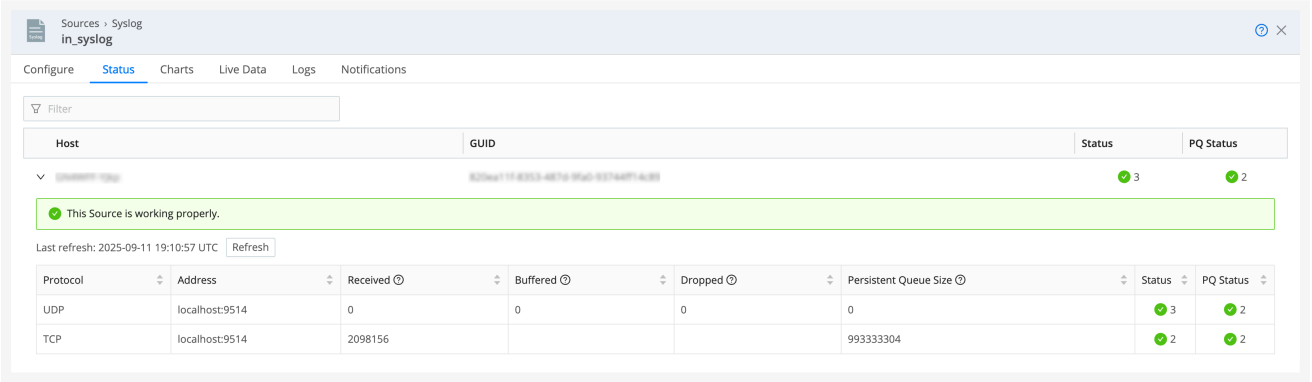
You can set Notifications for these Source persistent queue scenarios:

- Persistent queue files exceed a configurable percentage of allocated storage.
- A Source reaches the queue-full state, meaning it is running out of available disk space.
- Persistent Queue Usage (saturation).

See About Notifications for information about adding Notifications for Sources.

Monitor Source Persistent Queues

Cribl Stream provides metrics and health indicators to monitor your Source persistent queue through the **Status** tab on your Source. The **Status** tab aggregates metrics and health from all Worker Processes, allowing you to assess the health of your persistent queues and the status of the Source in real time.



Example of the Source Status tab

To view the Source persistent queue metrics:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Data**, select **Sources**.
2. From the sidebar, select the type of Source you want to view.
3. Select the Source from the list of currently configured Sources.
4. In the resulting modal, select the **Status** tab.

Inside this view, you can expand individual Worker Nodes for more details.

For additional analysis, you can also use Monitoring dashboards:

1. Navigate to **Monitoring** on the product sidebar.
2. Navigate to **System** > **Queues (Sources)** and view relevant data.

Reference: Source Persistent Queue Fields

In the **Persistent Queue Settings** tab, you can optionally specify persistent queue storage, using the following controls. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Source Persistent Queues (sPQ)

On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the **Enable persistent queue** toggle. If enabled, PQ is automatically configured in `Always On` mode, with a maximum queue size of 1 GB disk space allocated per PQ-enabled Source, per Worker Process.

The 1 GB limit is on uncompressed inbound data, and the queue does not perform any compression. This limit is not configurable. For configurable queue size, compression, mode, and other options below, use a hybrid Group.

Enable persistent queue: Default is toggled off. When toggled on:

Mode: Select a condition for engaging persistent queues.

- `Always On`: This default option will always write events to the persistent queue, before forwarding them to the Cribl Stream data processing engine.
- `Smart`: This option will engage PQ only when the Source detects backpressure from the Cribl Stream data processing engine.

`Smart` mode only engages when necessary, such as when a downstream Destination becomes blocked *and* the **Buffer size limit** reaches its limit. When persistent queue is set to `Smart` mode, Cribl attempts to flush the queue when every new event arrives. The only time events stay in the buffer is when a downstream Destination becomes blocked.

Buffer size limit: The maximum number of events to hold in memory before reporting backpressure to the sender and writing the queue to disk. Defaults to `1000`. This buffer is for all connections, not just per Worker Process. For that reason, this can dramatically expand memory usage. Connections share this limit, which may result in slightly lower throughput for higher numbers of connections. For higher numbers of connections, consider increasing the limit.

Commit frequency: The number of events to send downstream before committing that Stream has read them. Defaults to `42`.

File size limit: The maximum data volume to store in each queue file before closing it and (optionally) applying the configured **Compression**. Enter a numeral with units of KB, MB, and so forth. If not specified, Cribl Stream applies the default `1 MB`.

Queue size limit: The maximum amount of disk space that the queue is allowed to consume on each Worker Process. Once this limit is reached, this Source will stop queueing data and block incoming data. Required, and defaults to `5` GB. Accepts positive numbers with units of `KB`, `MB`, `GB`, and so forth. Can be set as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** in Group or Fleet settings.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this field's specified path, Cribl Stream will append `/<worker-id>/inputs/<input-id>`.

Compression: Optional codec to compress the persisted data after a file closes. Defaults to `None`; `Gzip` is also available.

In Cribl Stream 4.1 and newer, the Source persistent queue default **Mode** is `Always on`, to best ensure events' delivery. For details on optimizing this selection, see Optimize Source Persistent Queues (sPQ).

You can optimize Workers' startup connections and CPU load at **Group/Fleet settings** > Worker Processes.

8.3.3. Optimize Destination Persistent Queues

Cribl Stream's Destination persistent queues supplement in-memory queues to prevent data loss during a backpressure event, such as when a downstream Destination receiver is unreachable or slow to respond. See Supported Destinations for more information about which Destinations support persistent queueing.

Read this guide to learn how to optimize your Destination persistent queue. This guide is helpful when you are in the initial planning and configuration stage and also when you have an active deployment that you want to performance tune. This guide explains:

- How Destination persistent queues respond to a backpressure event.
- The overall goals and key metrics involved in optimizing Destination persistent queues.
- An in-depth discussion of the Destination persistent queue configuration options and their potential impact on your system.
- The available notifications, monitoring dashboards, and troubleshooting tools for persistent queues.

See also:

- About Persistent Queues
- About Destination Backpressure Triggers

> On Cribl-managed Cribl.Cloud Workers with an Enterprise plan, you can enable persistent queue for supported Destinations. However, be aware that mode options and storage limits have specific restrictions in Cribl.Cloud. See For Cribl.Cloud Only: Whether to Use a Hybrid Group for more information.

How Destination Persistent Queues Work

Backpressure occurs when there is a temporary imbalance between the rates of inbound and outbound data that is either coming in or going out of Cribl Stream.

Different events trigger persistent queues on Destinations. For Destinations, the triggers depend on what kind of Destination it is, such as whether it is filesystem-based, TCP, UDP, and so on. See Destination Backpressure Triggers for more detailed information.

When persistent queue (PQ) engages, it writes and stores data on a local file system in your environment until the backpressure event resolves.

Destinations handle data rate imbalances using the following order of operations:

1. **In-memory buffers store data under normal conditions.** Under normal conditions, Destinations store data in in-memory buffers before sending the data to a receiver. These in-memory buffers on Destinations help absorb temporary imbalances between inbound and outbound data rates.
2. **If inbound data rates begin to spike, data shifts to storage in all the available buffers for that Destination.** Cribl Stream then attempts to output events at a more appropriate rate for the downstream receiver.

The threshold for triggering a backpressure event varies by Destination type. See Destination Backpressure Triggers for more information.

3. **A backpressure event triggers when buffers are full.**
4. **Backpressure behavior engages.** During a backpressure event, Cribl Stream responds in one of the following ways, based on the Destination **Backpressure behavior** options you configured on the Destination:

Setting	Description
Block	All data is blocked until the backpressure event has resolved. If set to Block, the output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender. If the sender also supports backpressure backpressure response mechanisms, the sender may stop sending data. Consult the documentation for the sender to ensure you know how that specific sender handles a backpressure signal. In general, TCP-based senders support backpressure and stop sending data once Cribl Stream stops sending TCP acknowledgments back to it.
Drop Events	The Destination will discard all new events until the backpressure event has resolved and the receiver is ready.
Persistent Queue	If the Destination supports it, the Destination will engage the persistent queue behavior.

5. **If enabled, persistent queue engages.** If the persistent queue behavior engages, Cribl Stream responds in one of the following ways, based on its configured backpressure mode:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream immediately queues and stores all data on a disk for all events, even when there is no backpressure.

You can configure additional settings to determine:

- How much data to store in memory.
- What happens to data when the queue reaches its capacity limit.

6. **When the backpressure event ends, data starts flowing out of the persistent queue and into the Destination.** Cribl Stream begins to drain persistent queues when the receiver can accept data. In the case of load-balanced receivers, this means when at least one receiver can accept data. By default, after data flow is re-established, the persistent queue flushes every 10 seconds. It uses FIFO (first in, first out) event forwarding, ensuring Cribl Stream sends earlier queued events first when receivers recover. However, in Cribl Stream 4.1 and newer, you can instead prioritize new events by disabling **Strict ordering** on a Destination.

If you turn off Destination persistent queue (or shut down the whole Destination) before its queues drain, you will orphan any events still queued on disk. See Steps to Safely Disable and Clear Persistent Queues for more information.

Configure Destination Persistent Queue

You configure Destination persistent queue behavior when you configure each individual Destination. Consult the documentation for that Destination for more information about initial configuration.

This section provides a configuration quick start for experienced users. However, new Cribl Stream users should consider reading the full guidance in this topic to get a deeper understanding of the decisions and ideal system architecture design for persistent queue.

To configure persistent queue on a Destination:

1. In the **General Settings** tab, set the **Backpressure Behavior** option to **Persistent Queue**. See Whether to Enable Persistent Queues for Backpressure Events for guidance about the decision to enable persistent queue.
2. In the **Persistent Queue Settings** tab, select an appropriate **Mode** from the available options. See When and How to Store Data to Disk During a Backpressure Event for information about which mode to select.
3. Set the **File size limit** to specify the maximum amount of data stored in each queue file before the file closes and starts a new file. Set the **Queue size limit** to determine the maximum amount of disk space that Cribl allows the queue to consume on each Worker Process. See How Much Data to Store in Memory for more information about these settings.
4. Set the **Queue file path** to determine the location for the persistent queue files. See Where to Store Data and How Much Disk Space to Allocate for more information.

5. Optional: Set the **Compression** type to apply compression to the data before the queue file closes. See Whether to Compress Data for more information.
6. Set the **Queue-full behavior** to determine what happens when the persistent queue reaches its **Queue size limit** and runs out of space. See What Happens to Data when the Queue Reaches Its Configured Limits for more information.
7. The **Backpressure duration limit** setting displays when the **Mode** is set to `Backpressure`. It controls the duration a Destination must experience backpressure before data is queued to disk. The default waiting time is `30` seconds. Lowering this limit strengthens data loss prevention, while increasing it helps prevent unnecessary queue activation in environments with frequent, short network interruptions.
8. Optional: If needed, turn off **Strict Ordering** to more finely tune the **Drain rate limit (EPS)**. See How to Drain Persistent Queues When Backpressure Events End for more information.

Overview of Destination Persistent Queue Optimization

A well-optimized data throughput system can efficiently and reliably processes and transfer data from Sources to Destinations. Optimized systems maximize performance while minimizing latency, errors, and resource consumption. An optimized system has these characteristics:

- **High efficiency:** Data flows seamlessly at the maximum achievable rate without bottlenecks or excessive resource usage. The system gracefully handles backpressure events, ensuring that data intelligently throttles or queues without overwhelming downstream components.
- **Reliability:** The system consistently handles variations in data volume or speed without losing or corrupting data, even during spikes or failures.
- **Scalability:** The system adapts smoothly to increasing data loads or additional data senders or receivers without degradation in performance.
- **Low latency:** The system processes and delivers data in near real-time, meeting the required time constraints for your business use cases.
- **Cost-efficient resource allocation:** The system uses CPU, memory, disk, and network resources effectively, avoiding over-provisioning or waste.
- **Security and compliance:** The system transmits data securely, meeting all relevant privacy, encryption, and regulatory requirements.

Ultimately, you and your business stakeholders should have confidence in the system's performance, reliability, and ability to adapt to future needs. Persistent queue is an important tool to achieve this confidence.

Persistent Queue Optimization Metrics

The most effective way to performance tune is to understand which key metrics give insights into your persistent queues system's overall health. Effective system administrators study their deployments to establish

a performance baseline that they can use for comparison if a performance issue arises. Most system administrators understand their system well and can easily identify peak data periods.

Pay attention to these metrics:

- **CPU utilization:** When your system gets sudden increases in data loads, you might notice that your Worker Processes operate at full capacity and use all their CPU. Generally, that's a sign of inefficiency or the need for more resources. If needed, you can teleport into a single Worker and view all the processes and their individual CPU utilization. To see your CPU utilization, you can use the system monitoring tools to analyze CPU utilization. If needed, you can also teleport into a single Worker and view all the processes and their individual CPU utilization.
- **Data latency:** Some persistent queues can add latency because of the amount of overhead time it takes to write data to disk. As a starting point, assume that each event incurs a few milliseconds of added latency. Be aware that data latency can vary by file system, such as whether you are using a local file system or a Network File System (NFS). NFS file systems tend to introduce higher and more variable latency. To measure the latency impact of your persistent queue system, compare the throughput times of your system without persistent queues enabled and then introduce persistent queue and measure the difference.
- **Memory:** When your system experiences sudden increases in data, your Workers need memory to store that data in temporary buffers. Consider what should happen in your system when memory is fully utilized.
- **Data storage:** When you back up data to a disk, that requires available disk space to use. Allocating memory and disk space can potentially be expensive. Be aware of the rate at which you consume disk memory and the general costs you estimate you will incur to write data to disk. Have systems in place for knowing when disk space is running out and adding more memory as needed.
- **Data throughput:** You need to understand your system's throughput under normal conditions so that you can identify potential bottlenecks in the system. Use the system monitoring tools to develop this baseline.

Key Decisions When Configuring Destination Persistent Queues

You configure Destination persistent queue behavior when you configure each individual Destination. Consult the documentation for that Destination for more information about initial configuration.

This section explains the decision-making process you should consider when implementing persistent queue on a Destination. It explains the benefits and tradeoffs of different configuration settings for most use cases.

Whether to Enable Persistent Queues for Backpressure Events

The first decision you need to make is whether to use persistent queue to respond to backpressure. See When to Use Persistent Queues for a detailed explanation of which kinds of environments benefits from persistent

queues.

In general, you should not use persistent queue is you are already using a backpressure mechanism on a downstream receiver. Implementing backpressure mechanisms such as persistent queue on both Cribl Stream and the Destination is redundant and could cause unnecessary data latency or impact data throughput.

The alternatives to using persistent queue are to block data or to drop events when a backpressure event occurs. Consider blocking events if you need Cribl to return block signals back to an upstream sender. If the sender also supports backpressure backpressure response mechanisms, the sender may stop sending data.

Consult the documentation for the sender to ensure you know how that specific sender handles a backpressure signal. In general, TCP-based senders support backpressure and stop sending data once Cribl Stream stops sending TCP acknowledgments back to it.

For Cribl.Cloud Only: Whether to Use a Hybrid Group

On Cribl-managed Cribl.Cloud Workers (with an Enterprise plan), persistent queue does not come with many of the configuration options described in the rest of this page. For Cribl.Cloud, enabling persistent queue only exposes the destructive **Clear Persistent Queue** button. Use this button with caution to avoid data loss. See Steps to Safely Disable and Clear Persistent Queues for more information.

When you enable persistent queues in Cribl.Cloud, Cribl automatically allocates up to 1 GB of disk space per Destination, per Worker Process. This 1 GB limit applies to outbound uncompressed data, and the queue does not perform any compression. If the queue fills up, Cribl Stream blocks data flow.

You cannot change this limit. If the queue fills up, Cribl Stream/Edge will block outbound data. Use a hybrid Group if you need to configure:

- Queue size
- Compression
- Backpressure modes
- Other persistent queue settings

When and How to Store Data to Disk During a Backpressure Event

Once the backpressure queue behavior engages, you need to determine how it should write data to disk. You have three available options (modes) for storing data during a backpressure event:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.

Mode	Description
Always On	Cribl Stream immediately queues and stores all data on a disk for all events, even when there is no backpressure.

Consider using the **Always On** mode if your business has a low tolerance for data loss. However, do so with some caution. Enabling this mode could have impacts on the persistent queue optimization metrics, such as disk space, throughput, data latency, and CPU utilization.

Consider experimenting with this mode to see whether it's right for your business needs. Establish a baseline for system performance before you enable **Always On** and then observe the impacts when you turn that mode on. Ensure you have enough disk storage space to handle your data storage needs.

How Much Data to Store in Memory

When you enable persistent queues, you can configure the **File size limit** setting to specify the maximum amount of data stored in each queue file before the file closes and starts a new file. You can enter a value in units such as KB, MB, and more. If left unspecified, Cribl Stream/Edge defaults to 1 MB.

Additionally, you can use the **Queue size limit** setting to determine the maximum amount of disk space that Cribl allows the queue to consume on each Worker Process. Once the queue reaches this limit, the Source will stop queueing data and engage the configured backpressure mode. You can enter a value in units such as KB, MB, GB, or TB.

If the **Queue size limit** is not specified, it defaults to 5 GB. You can adjust this value to better suit your needs. For Sources with undefined **Queue size limit** values, queues exceeding 5 GB will trigger the configured queue-full behavior until you resize the limit.

To set a higher maximum value, navigate to **Worker Group/Fleet Settings** > **General Settings** > **Limits** > **Storage** > **Worker Process PQ size limit**. The corresponding key in the `limits.yml` file is `maxPQSize`. (See Config Files: limits.yml for more information.) The default for this setting is 1 TB. Before increasing this limit, consult Cribl Support for guidance.

> The information in this section applies only to on-prem or hybrid Worker Groups. For Cribl-managed Cribl.Cloud Organizations using persistent queue, Cribl automatically allocates up to 1 GB of disk space per Source, per Worker Process. You cannot change this limit.
>
> This 1 GB limit applies to outbound uncompressed data, and the queue does not perform any compression. If the queue fills up, Cribl Stream/Edge blocks data flow, including outbound data.

Whether to Compress Data

When the queue file reaches its maximum file size, you can use the optional **Compression** setting to apply compression to the data before the queue file closes. Currently, the only available compression codec is Gzip.

If you enable **Compression** and also enter a **Queue size limit**, be aware of how these options interact:

- Cribl Stream/Edge currently applies the **Queue size limit** against the **uncompressed** data volume entering the queue.
- With this combination, Cribl recommends that you set the **Queue size limit higher** than the volume's total available disk space, not including compression. This will maximize queue saturation and minimize data loss.

Where to Store Data and How Much Disk Space to Allocate

Use the **Queue file path** setting to determine the location for the persistent queue files. The default is `$CRIBL_HOME/state/queues`. When writing data, Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this path.

For queuing to operate properly, you must provide sufficient disk space. In Distributed mode, this minimum disk space configuration is set on each Worker Group/Fleet. You can access this setting at **Group/Fleet settings** > **General Settings** > **Limits** > **Min free disk space**.

If available disk space falls below this minimum threshold, Cribl Stream/Edge will stop maintaining persistent queues, and data loss will begin. The default minimum is 5 GB.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. To this value, Cribl Stream/Edge will append `/<worker-id>/<output-id>`.

What Happens to Data when the Queue Reaches Its Configured Limits

You can use the **Queue-Full Behavior** setting to determine what happens when the persistent queue reaches its **Queue size limit** and runs out of space. There are two options:

Setting	Description
Block	Blocks all data until the backpressure event has resolved. If set to Block, the output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
Drop new data	The Destination will discard all new events until the backpressure event has resolved and the receiver is ready.

How to Drain Persistent Queues When Backpressure Events End

When a backpressure event ends, the persistent queue begins to drain to the downstream Destination. You can configure the order of event delivery and the rate at which Cribl Stream forwards events.

Use the **Strict Ordering** setting to configure the order that Cribl Stream sends data to the Destination. Toggle **Strict Ordering** on (default) to use FIFO (first in, first out) processing. This means Cribl prioritizes earlier queued events before sending newly arrived data. Toggle **Strict Ordering** off to change this behavior, prioritizing new data over the events already in the queue.

If you toggle **Strict Ordering** off, you must configure the **Drain rate limit (EPS)**. This setting allows you to throttle the rate at which events drain from the persistent queue to the Destination. The default value of `0` disables throttling. Throttling the drain rate helps Cribl manage the incoming stream of live data more effectively, preventing delays caused by queue processing. It also improves the throughput of active connections by reserving additional resources for them.

The Drain rate limit (EPS) is set in Events Per Second (EPS). To determine your system's EPS value:

1. Open the Monitoring page. From Cribl Stream's sidebar, select **Monitoring** to access dashboards displaying data for all Worker Groups and all nodes.
2. Filter by a specific node. Use the **Groups** menu at the top to restrict monitoring data to a single Worker Node.
3. Adjust the time granularity. Use the time menu change the display granularity from the default `15 min` to `1 day` for better accuracy.
4. Locate the EPS metrics. On the Monitoring page, find the **Events In and Out** metrics group. Copy the value listed under THROUGHPUT IN (AVG). For example, a typical value might be `31,000 EPS`.

Cribl recommends setting the **Drain Rate Limit** to 10% of the Worker Process EPS to prioritize live data processing. To calculate an appropriate drain rate:

1. Determine EPS per Worker Process. Divide the Worker Process EPS by the number of Worker Processes (WPs). For example: `31,000 EPS / 14 WPs = 2,200 EPS per WP`.
2. Calculate 10% of EPS per Worker Process. Multiply the result by `0.10` to find the recommended drain rate. For example: `10% of 2,200 EPS = 220 EPS`.
3. Configure the **Drain Rate Limit**. In the persistent queue settings for the Destination, enter this value in the **Drain rate limit (EPS)** field. For example: `220`.

After applying this setting, monitor the queue's drain rate. If the queue is draining too slowly, carefully increase the **Drain rate limit (EPS)** while ensuring it doesn't interfere with live data processing. Avoid setting the limit too high, as this can impact the performance of live streams. If needed, consider optimizing Worker startup connections and managing CPU load to further enhance performance at **Group Settings** > Worker Processes.

Cribl's guarantee for data delivery is that all data is delivered at least once, but not exactly once. Sometimes Cribl Stream sends duplicate data events when the persistent queue drains.

Steps to Safely Disable and Clear Persistent Queues

If you want to disable persistent queuing, always wait for the queue to fully drain. The **Clear Persistent Queue** button is available as a destructive fallback. This action frees up disk space, but it also deletes the queued data *without* sending it on to receivers.

After the queue has drained, then you can select a different backpressure behavior. Save the configuration after changing it.

Destination Persistent Queue Notifications

Some Cribl licenses allow you to configure Notifications for Destination persistent queues for on-prem deployments. In general, Cribl recommends setting Notifications for these events when using persistent queue to ensure you are aware of important system events. These Notifications always appear in Cribl Stream's user interface and internal logs. You can also send them to external systems.

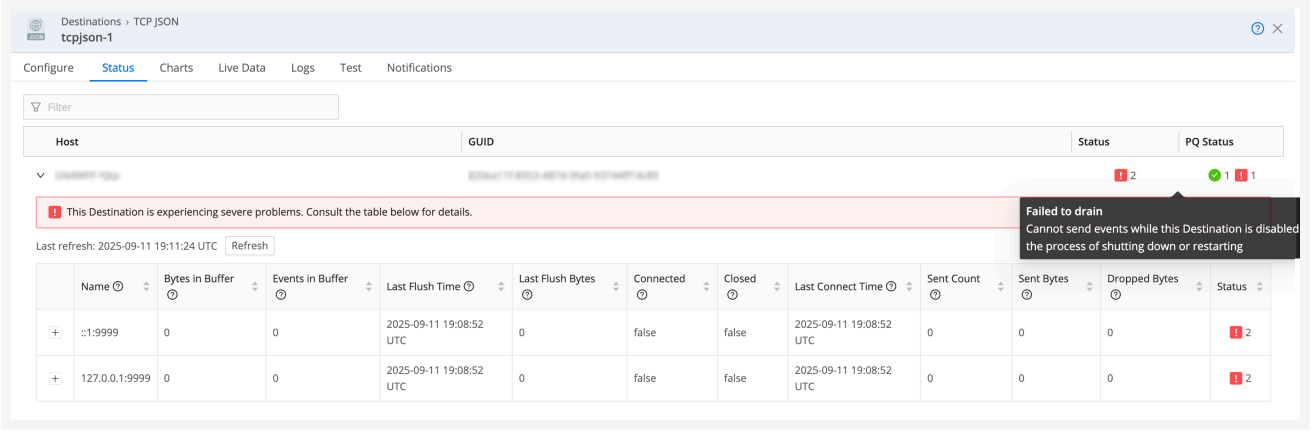
You can set notifications for these scenarios:

- Persistent queue files exceed a configurable percentage of allocated storage.
- A Destination reaches the queue-full state, meaning it is running out of available disk space.
- A Destination backpressure engages, causing blocked or dropped events.
- Persistent Queue Usage (saturation).

See About Notifications for information about adding Notifications for Destinations.

Monitor Destination Persistent Queues

Cribl Stream provides metrics and health indicators to monitor your Destination persistent queue through the **Status** tab on your Destination. Cribl Stream provides metrics and health indicators to monitor your Source persistent queue through the **Status** tab on your Source. The **Status** tab aggregates metrics and health from all Worker Processes, allowing you to assess the health of your persistent queues and the status of the Destination in real time.



Example of the Destination Status tab

To view the Destination persistent queue metrics:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Data**, select **Destinations**.
2. From the sidebar, select the type of Destination you want to view.
3. Select the Destination from the list of currently configured Destination.
4. In the resulting modal, select the **Status** tab.

Inside this view, you can expand individual Worker Nodes for more details.

For additional analysis, you can also use Monitoring dashboards:

1. Navigate to **Monitoring** on the product sidebar.
2. Navigate to **System** > **Queues (Sources)** and view relevant data.

Troubleshooting Tools

Cribl internal logs for Cribl-managed Workers might include entries labeled `Revived PQ Worker Process`. These indicate where a Cribl-managed Worker was newly assigned to drain a persistent queue left over from a shut-down Worker Node. You can use these log events to troubleshoot Cloud PQ issues.

Reference: Destination Persistent Queue Fields

The **Persistent Queue Settings** tab displays when the **Backpressure behavior** option in **General settings** is set to **Persistent Queue**. Persistent queue buffers and preserves incoming events when a downstream Destination has an outage or experiences backpressure.

Before enabling persistent queue, learn more about persistent queue behavior and how to optimize it with your system:

- About Persistent Queues
- Optimize Destination Persistent Queues (dPQ)
- Destination Backpressure Triggers

> On Cribl-managed Cloud Workers (with an Enterprise plan), this tab exposes only the destructive **Clear Persistent Queue** button (described at the end of this section). A maximum queue size of 1 GB disk space is automatically allocated per PQ-enabled Destination, per Worker Process. The 1 GB limit is on outbound uncompressed data, and no compression is applied to the queue.
>
> This limit is not configurable. If the queue fills up, Cribl Stream/Edge will block outbound data. To configure the queue size, compression, queue-full fallback behavior, and other options below, use a

> hybrid Group.

Mode: Use this menu to select when Cribl Stream/Edge engages the persistent queue in response to backpressure events from this Destination. The options are:

Mode	Description
Error	Queues and stores data on a disk when the Destination is unavailable or in an error state.
Backpressure	Queues and stores data to a disk when it detects backpressure from the Destination until the backpressure event resolves.
Always On	Cribl Stream or Edge immediately queues and stores all data on a disk for all events, even when there is no backpressure.

> If a Worker/Edge Node starts with an invalid **Mode** setting, it automatically switches to **Error** mode. This might happen if the Worker/Edge Node is running a version that does not support other modes (older than 4.9.0), or if it encounters a nonexistent value in YAML configuration files.

File size limit: The maximum data volume to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Queue size limit: The maximum amount of disk space that the queue can consume on each Worker Process. When the queue reaches this limit, the Destination stops queueing data and applies the **Queue-full** behavior. Defaults to `5 GB`. This field accepts positive numbers with units of `KB`, `MB`, `GB`, and so on. You can set it as high as `1 TB`, unless you've configured a different **Worker Process PQ size limit** on the Worker Group/Fleet Settings page.

Queue file path: The location for the persistent queue files. Defaults to `$CRIBL_HOME/state/queues`. Cribl Stream/Edge will append `/<worker-id>/<output-id>` to this value.

Compression: Set the codec to use when compressing the persisted data after closing a file. Defaults to `None`. `Gzip` is also available.

Queue-full behavior: Whether to block or drop events when the queue begins to exert backpressure. A queue begins to exert backpressure when the disk is low or at full capacity. This setting has two options:

- **Block**: The output will refuse to accept new data until the receiver is ready. The system will return block signals back to the sender.
- **Drop new data**: Discard all new events until the backpressure event has resolved and the receiver is ready.

Backpressure duration Limit: When **Mode** is set to `Backpressure`, this setting controls how long to wait during network slowdowns before activating queues. A shorter duration enhances critical data loss prevention, while a longer duration helps avoid unnecessary queue transitions in environments with frequent, brief network fluctuations. The default value is `30` seconds.

Strict ordering: Toggle on (default) to enable FIFO (first in, first out) event forwarding, ensuring Cribl Stream/Edge sends earlier queued events first when receivers recover. The persistent queue flushes every 10 seconds in this mode. Toggle off to prioritize new events over queued events, configure a custom drain rate for the queue, and display this option:

- **Drain rate limit (EPS)**: Optionally, set a throttling rate (in events per second) on writing from the queue to receivers. (The default `0` value disables throttling.) Throttling the queue drain rate can boost the throughput of new and active connections by reserving more resources for them. You can further

8.3.4. About Destination Backpressure Triggers

This page describes the conditions that can trigger backpressure in Cribl Stream Destinations. Backpressure exists when a destination receives more data than it can currently send. It typically results from network connectivity problems or when a Destination is receiving data faster than it can forward that data to the downstream service.

When a Destination signals backpressure, it notifies the Sources generating data that its buffers are full and that the incoming data flow should pause until there is room for more data.

You can configure your desired behavior through a Destination's **Backpressure Behavior** drop-down. Where other options are not displayed, Cribl Stream's default behavior is **Block**. For details about all the above behaviors and options, see Persistent Queues.

HTTP-Based Destinations

HTTP-based Destinations trigger backpressure when they encounter errors while connecting or when they receive certain HTTP response codes. If the HTTP call returns a `500` series (`500-599`) HTTP code, the Destination adds the events in the buffer back to its internal buffer and tries to send them again. The retry attempt can also trigger backpressure, depending on the frequency at which this happens.

If the Destinaton receives a `400` series response code, it will drop (not retry) the events, because these response codes indicate an error in payload or formatting (for example). See this Mozilla page on the `400 Bad Request` code for more information.

Finally, HTTP-based Destinations trigger backpressure when their processing loads exceed the specified limit. HTTP-based Destinations handle requests by creating a connection, sending the request payload, and receiving and parsing a response. Because this process can take time, HTTP Destinations allow more than one request to be in progress at a time. When the number of requests in flight exceeds the value set in **Advanced Settings** > **Request concurrency**, the Destination will trigger backpressure.

The following conditions will trigger backpressure:

- Connection failure.
- Connection or request timeouts.
- Data overload, such as when the Source sends more data than the Destination will accept.
- Reaching the limit on concurrent (in-flight) requests.
- HTTP `500` series responses.

You can set the number of in-flight or concurrent requests in **Advanced Settings** > **Request concurrency**.

See Available Destinations for information about which Destinations are HTTP-based.

Filesystem-Based Destinations

Filesystem-based Destinations that send data to cloud-based services trigger backpressure when Cribl Stream can't move a file or upload it to the cloud.

Filesystem-based Destinations queue events in files on disk in a staging directory. When a batch of events is ready for transmission, Cribl Stream closes the file, optionally compresses it, and transmits the file to the downstream service.

For the Filesystem Destination, the batch simply moves from the staging directory to the output directory.

For other filesystem-based Destinations, Cribl Stream uploads the file to a cloud service, such as AWS S3, Azure Blob Storage, or other. For these Destinations, a failure to upload the file will trigger backpressure.

These scenarios include:

- Permissions problem writing to the staging directory, such as when the user running the Cribl Stream process does not have permission to write to the staging directory.
- The filesystem for the staging or output directory is full (impacts Filesystem Destination only).
- The credentials provided for the cloud service are invalid.
- The Destination doesn't have the permissions required to upload to the cloud service.
- Cloud service permissions issue, such as when an upload fails because the cloud user (identified in credentials) does not have write permissions.
- Network connectivity - The Destination can't communicate with the cloud service due to network connectivity issues.

See Available Destinations for information about which Destinations are filesystem-based.

TCP-Based Destinations

TCP Sources establish a persistent connection with a Destination and send data. These Sources maintain connection to the Destination and will re-establish that connection if they detect a close or write timeout.

If the connection fails, Cribl Stream uses a backoff algorithm to avoid spiking the CPU. This backoff logic makes the first retry at 2 seconds, then gradually increases the time between retries up to a maximum of 60 seconds.

The following conditions can trigger backpressure from a TCP-based Destination:

- Connection failure.
- Authentication failure.

- Write timeout, where a Source sends data but doesn't receive an `ack` at the TCP layer. In this case, Cribl Stream closes the connection to the Destination and tries to re-establish it.
- The Destination receives data from the Source faster than it can send data over the network to the downstream service, causing internal buffers to fill up. This condition can occur due to a slow network connection or slow processing on the downstream service.When space in the internal buffers becomes available, backpressure is relieved, allowing the Source to send more data.

See Available Destinations for information about which Destinations are TCP-based. Some Destinations use either TCP or UDP to send messages, but they trigger backpressure and use the backoff algorithm described above only when used with the TCP protocol.

Kafka

Kafka is a binary protocol that runs over TCP. The protocol defines three roles - producers, consumers, and brokers.

Producers send data to a Kafka topic (think of it as a message queue). Kafka consumers read data from Kafka topics. Brokers receive data from topic producers and send data to topic consumers, meaning they serve as go-betweens for producers and consumers.

Cribl Stream's Kafka-based Destinations are producers. They send data to Kafka brokers. Since the protocol is TCP-based, the conditions that trigger backpressure are similar to those of Cribl Stream's TCP Destinations.

The specifics for connection and request retries are defined by the Kafka JS library in conjunction with the configuration parameters defined for each Destination. In all cases, the Destinations have internal buffers that are sent to the Broker when they're considered full (on a size and time basis).

The following conditions can trigger backpressure for Kafka-based Sources:

- Failure to connect to the broker, lack of network connectivity, or the broker is down.
- Invalid credentials - Destination connects successfully, but the connection is rejected due to a failure to authenticate.
- Invalid permissions - The credentials assigned do not have proper permissions to send events to the topic.
- The Destination is attempting to send data faster than the broker can receive it. Backpressure will be relieved when internal buffers have room for more events.

Kafka Retries and PQ

In some situations, Kafka Destinations may take a long time to engage the PQ (1 to 6 minutes by default).

This delay can result from the default settings used with the Kafka Destination. The relevant settings are all in the **Advanced Settings** UI:

- **Connection timeout (ms)**: Defaults to `10000`,
- **Request timeout (ms)**: Defaults to `60000`.
- **Retry limit**: Defaults to `5`.

These default settings mean that Cribl Stream will retry five times after a failed connection, with a 10-second timeout between each attempt. It will also retry five files after a failed request, with a timeout of 60 seconds between each attempt.

For faster PQ engagement, update these values. You can use smaller values for **Connection timeout** and **Request timeout** to decrease the interval between retries. You can also use smaller values for **Retry limit** or even set it to `0` to eliminate retries altogether.

UDP

UDP is a connectionless protocol, so there is no concept of backpressure at the network layer. As a result, UDP Destinations will never trigger backpressure.

See Available Destinations for information about which Destinations support UDP. Except for SNMP Trap, these Destinations also support TCP, but the UDP versions do not trigger backpressure.

SQS

The SQS (Simple Queue Service) Destination uses the AWS SQS library to send events to the Destination's configured queue. Like Kafka, this Destination acts as a producer that publishes messages to the provisioned SQS queue.

This Destination has five buffers with 10 events per buffer for a total of 50 events that can be queued before backpressure is enabled. The reason that backpressure might be generated for this Destination include:

- Invalid credentials.
- Invalid permissions.
- Queue does not exist and **Create Queue** is disabled.
- Lack of network connectivity.

Amazon Kinesis

Kinesis runs over HTTPS and behaves like Pub/Sub Destinations.

This Destination serializes and publishes events in a newline-delimited JSON format. In the context of the Kinesis Destination, it behaves as a producer where a Kinesis source is considered a consumer. Since the protocol is HTTP-based, the conditions that trigger backpressure are similar to those of other Cribl Stream HTTP Destinations.

The Kinesis library defines the specifics for connection and request retries in conjunction with the configuration parameters defined for each Destination. In all cases, the Destinations have internal buffers that are sent to the Broker when they're considered full (on a size and time basis).

The following conditions can trigger backpressure for Kinesis-based Destinations:

- Failure to connect to the Kinesis service, network connectivity, or the service is down.
- Invalid credentials - Connects successfully but connection is rejected due to a failure to authenticate.
- Invalid permissions - The credentials assigned do not have proper permissions to send events to the Kinesis stream.
- The Destination is attempting to send data faster than the service can receive it. Backpressure will be relieved when internal buffers have room for more events.

Google Cloud Pub/Sub

Google Cloud Pub/Sub is similar to Kafka and Kinesis in that there are producers that send data and consumers that receive data. The Cribl Stream Google Cloud Pub/Sub Destination is a producer that sends data to the Google Cloud Pub/Sub service. Google Cloud Pub/Sub leverages gRCP, however, which uses protocol buffers as both its interface definition language (IDL) and as its underlying message interchange format.

The conditions that trigger backpressure for a Google Cloud Pub/Sub Destination are:

- Failure to connect to the Google Pub/Sub service, network connectivity, or the service is down.
- Invalid credentials - Connects successfully but connection is rejected due to a failure to authenticate.
- Invalid permissions - The credentials assigned do not have proper permissions to send events to the topic.
- The Destination is attempting to send data faster than the service can receive it. Backpressure will be relieved when internal buffers have room for more events.

OpenTelemetry

OpenTelemetry supports gRPC or HTTP to send messages. For the gRPC protocol, the same rules described for other TCP Destinations generally apply to OpenTelemetry. The key difference is that this Destination uses the OpenTelemetry library, which handles write timeouts and reconnection logic.

For the HTTP protocol, the same rules apply to our HTTP Destinations for retries based on connectivity or `500` series response codes.

Load-Balanced Destinations

HTTP-Based Load-Balanced Destinations

Some HTTP-based Destinations support two modes - single host and multiple load-balanced hosts.

With a load-balanced version of an HTTP Destination, Cribl Stream spreads events across all configured Destinations, using a weighted algorithm. Under normal circumstances, this algorithm spreads the load across the configured Destinations according to the weight assigned to the Destination.

When one of the load-balanced Destinations is out of service (due to a write timeout or `500` series response code - buffers full), Cribl Stream removes that Destination from the list of hosts and moves any in-flight buffers to the next available host. Cribl Stream will try to resend events to the downed host in the background, and it will resume sending data to that host once the connection is restored.

See Available Destinations for information about which HTTP-based Destinations support load balancing.

TCP Load-Balanced Destinations

Some TCP-based Destinations support two modes - single host and multiple load-balanced hosts.

With a load-balanced TCP Destination, Cribl Stream spreads events across all configured Destinations using a weighted algorithm. Under normal circumstances, this algorithm spreads the load across the configured Destinations according to the weight assigned to each Destination.

When one of the load-balanced Destinations is out of service (due to a write timeout or connection failure), Cribl Stream removes that Destination from the list of hosts and moves any in-flight buffers to the next available host. Cribl Stream will try to reconnect to the downed host and will resume sending data to that host once the connection is restored.

TCP load-balanced Destinations can be configured with one or multiple receivers. If one or more receivers go down, Cribl Stream will continue sending data to any healthy receivers.

See Available Destinations for information about which TCP Destinations support load balancing.

Output Router

The Output Router is a wrapper destination that groups together 1..*N* Destinations. It's used in conjunction with event filters to route data to a selected Destination. The Output Router Destination emits backpressure only when one of its downstream Destinations is blocking.

8.4. About Load Balancing

Cribl Stream will attempt to load-balance outbound data as fairly as possibly across all endpoints. For example, if FQDNs/hostnames are used as the Destination addresses, and each resolves to 5 (unique) IPs, then each Worker Process will have its # of outbound connections = {# of IPs x # of FQDNs} for purposes of the Destination.

Data is sent by all Worker Processes to a randomly selected endpoint, and the amount sent to each connection depends on these parameters:

1. Respective destination **weight**: If a **Load Weight** is set to `0`, Cribl Stream will not try to connect to that IP (nor to any IPs resolved for that FQDN). When a connection is blocked, Cribl Stream will apply a 10% penalty to the load weight, and will select the connection with the lower adjusted load.
2. Respective destination **historical data**: By default, historical data is tracked for 300s. Cribl Stream uses this data to influence the traffic sent to each destination, to ensure that differences decay over time, and that total ratios converge towards configured weights.

Sources share a single Worker Process for load balancing, so you don't need to dedicate a separate Worker Process for load balancing for each Source. Load balancing Worker Processes have a high throughput rate because they only handle Event Breakers.

On Destinations that support load balancing, Cribl recommends enabling the feature, even if you only have one hostname - because this hostname can expand to multiple IPs. If you must disable load balancing, then to ensure robust connections, enable the alternative **Advanced Settings** > **Round-robin DNS** option. This will allow Cribl Stream to follow the DNS host's IP rotation, preventing connection pinning.

If a request fails, Cribl Stream will resend the data to a different endpoint. Cribl Stream will block only if **all** endpoints are experiencing problems.

Example: Equal Distribution

Suppose we have two connections, A and B, each with weight of `1` (i.e., they are configured to receive equal amounts of data). Suppose further that the load-balance stats period is set at the default `300s` and - to make things easy - for each period, there are 200 events of equal size (bytes) that need to be balanced.

Interval	Time Range	Events to balance
1	*time=0s —> time=300s*	**200**

Both A and B start this interval with 0 historical stats each.

Let's assume that, due to various circumstances, 200 events are "balanced" as follows: `A = 120 events` and `B = 80 events` - a difference of **40 events** and a ratio of **1.5:1**.

Interval	Time Range	Events to balance
2	*time=300s —> time=600s*	**200**

At the beginning of interval 2, the load-balancing algorithm will look back to the previous interval stats and carry **half** of the receiving stats forward. In other words, connection A will start the interval with **60** and connection B with **40**. To determine how many events A and B will receive during this next interval, Cribl Stream will use their weights and their stats as follows:

Total number of events: `events to balance + stats carried forward = 200 + 60 + 40 = 300`.

Number of events per each destination (weighted): `300/2 = 150` (they're equal, due to equal weight).

Number of events to send to each destination `A: 150 - 60 = 90` and `B: 150 - 40 = 110`.

Totals at the end of interval 2: `A=120+90=210`, `B=80+110=190`, a difference of **20 events** and a ratio of **1.1:1**.

Over the subsequent intervals, the difference becomes exponentially less pronounced, and eventually insignificant. Thus, the load gets balanced fairly.

Example: Unequal Distribution

Suppose you have three connections, A, B, and C, weighted to receive unequal amounts of data:

- A: 1
- B: 2
- C: 7

Assume that the load-balance stats period is set at the default `300s` and for each period, 200 events of equal size (bytes) need to be balanced:

Interval	Time Range	Events to balance
1	*time=0s —> time=300s*	**200**

All three connections (A, B, and C) start this interval with 0 historical stats each.

Let's assume that 200 events are "balanced" as follows: `A = 30 events`, `B = 50 events`, and `C = 120 events` - a difference of **90 events** and a ratio of **3:5:12**.

Interval	Time Range	Events to balance
2	*time=600s —> time=600s*	**200**

At the beginning of interval 2, the load-balancing algorithm will look back to the previous interval stats and carry **half** of the receiving stats forward. This means that connection A will start the interval with **15**, connection B with **25**, and connection C with **60**. To determine how many events each connection will receive during this next interval, Cribl Stream will use their weights and their stats as follows:

Total number of events: `events to balance + stats carried forward = 200 + 15 + 25 + 60 = 300`.

Number of events per each destination (weighted): `300 / (1 + 2 + 7) = 300 / 10 = 30`.

Number of events to send to each destination `A: 30 - 15 = 15`; `B: 60 - 25 = 35`; `C: 210 - 60 = 150`.

Totals at the end of interval 2: `A: 30 + 15 = 45`; `B: 50 + 35 = 85`; `C: 120 + 150 = 270`.

The difference between the highest connection (C) and the lowest (A) is **225 events**. The ratio at the end of interval 2 is approximately **1:1.89:6**.

Over the subsequent intervals, the difference becomes exponentially less pronounced, and eventually insignificant. Thus, the load gets balanced fairly.

8.5. Configure Host Metadata

Cribl Stream can add a `__metadata` property to every event produced from every enabled Source. You can view the metadata collected for a Worker Node by selecting the Worker Node on the **Workers** page.

The metadata surfaced by a Worker Node can be used to:

- Enrich events (with an internal `__metadata` field).
- Display metadata to users as a part of instance exploration.

You can customize the type of metadata collected at **Group Settings** > **General Settings** > **Limits** > **Metadata**. Add and select metadata sources by selecting **Add Source** and choosing an option from the list.

In Cribl Stream, all the **Event metadata sources** are disabled by default.

The sources that you can select here include:

Source	Description
`os`	Reports details for the host OS and host machine, like OS version, kernel version, CPU and memory resources, hostname, network addresses, and so on.
`cribl`	Reports the Cribl Stream version, Distributed mode, Worker Nodes (for managed instances), and configuration version.
`aws`	Reports details on an EC2 instance, including the instance type, hostname, network addresses, tags, and IAM roles. For security reasons, we report only IAM role **names**.
`env`	Reports environment variables.
`kube`	Reports details on a Kubernetes environment, including the node, Pod, and container. For details, see __metadata.kube Property.

When these metadata sources are enabled (and can get data), Cribl Stream will add the corresponding property to events, with a nested property for each enabled source.

Some metadata sources work only in configured environments. For example, the `aws` source is available only when running on an AWS EC2 instance. And `kube` applies only to on-prem deployments.

If your security tools report denied outbound traffic to IP addresses like `169.254.169.168` or `169.254.169.254`, you can suppress these by removing `aws` from the metadata sources described

above. If you have a proxy setup, Cribl recommends adding these IP addresses to your `no_proxy` environment variable.

`__metadata.kube` Property

For the `__metadata.kube` property (displayed as `kube` in the list of event metadata sources) to report details on a Kubernetes environment, Cribl Stream needs to know where it is running. Set the `KUBE_K8S_POD` environment variable to the name of the Pod in which Cribl Stream is running. This gives the `__metadata.kube` property the information it needs to report on the `node` and `pod` properties.

If `/proc/self/cgroup` is working, the `container` property information will be available, too.

8.6. Cribl Edge To Cribl Stream

Cribl Edge automatically discovers logs, metrics, application data, and so on - in real time - from your configured endpoints, and delivers them to Cribl Stream or any supported destination. Meanwhile, Cribl Stream can help collect, reduce, enrich, transform, and route data from Cribl Edge to any destination. And using a Cribl HTTP Source, you can collect and route data from Edge Nodes to Stream Worker Nodes without incurring additional cost.

For more information about sending data between organizations and Workspaces without paying for data twice, see Transfer Data Between Workspaces or Environments.

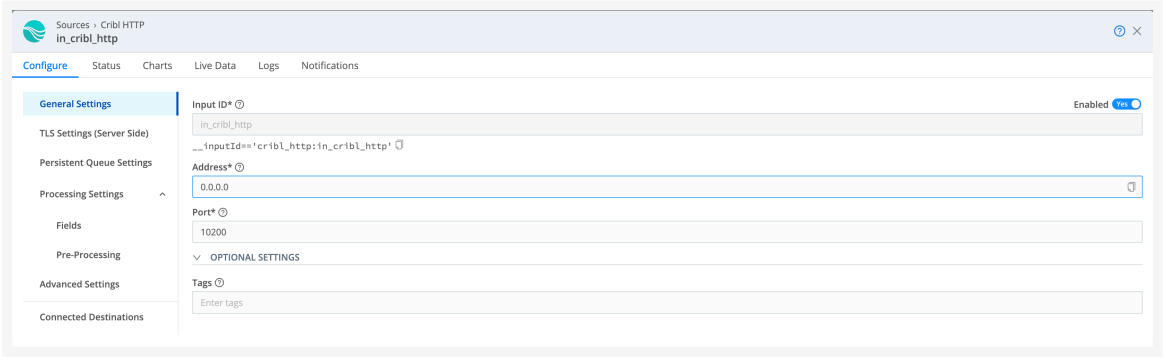
This guide outlines how to route data from an Edge Node (or an entire Fleet) to an existing Stream Worker Group for additional processing. We will walk you through the following:

- Configure Cribl HTTP Source on Cribl Stream to receive data from the Edge Node.
- Configure the Exec Source on Cribl Edge to collect data on the Edge Node.
- Configure the Cribl HTTP Destination on Cribl Edge to send data to Cribl Stream.
- Configure a Route to Send the Data.

And finally, we will confirm the data flow.

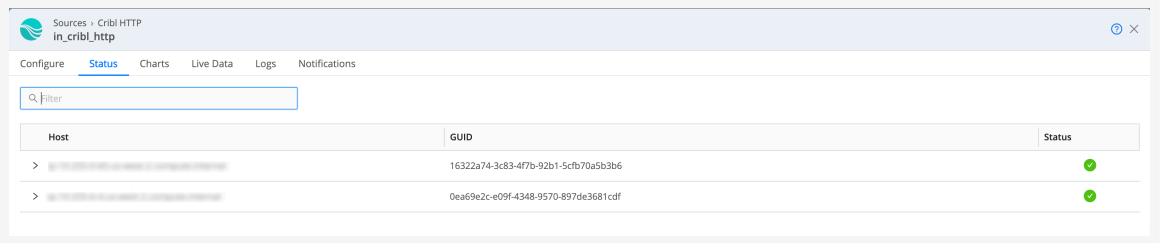
Configure the Cribl HTTP Source on Cribl Stream

In Cribl Stream, start by configuring and enabling a Cribl HTTP Source. The key requirement here is to set the **Port** to listen on. For this example, we'll use the default `10200`. (Optionally, you can also configure TLS, enable persistent queue, and/or a pre-processing Pipeline.)



Configuring a Cribl HTTP Source

When done, **Commit** and **Deploy** your changes. Before moving on to the next step, confirm that your Source is healthy.



Status of the Cribl HTTP Source

On Cribl-managed Workers in Cribl.Cloud, make sure that TLS is either disabled on both the Cribl HTTP Source and the Cribl HTTP Destination it's receiving data from, or enabled on both. Otherwise, no data will flow. In the Source, TLS is enabled by default.

Configure the Exec Source on Cribl Edge

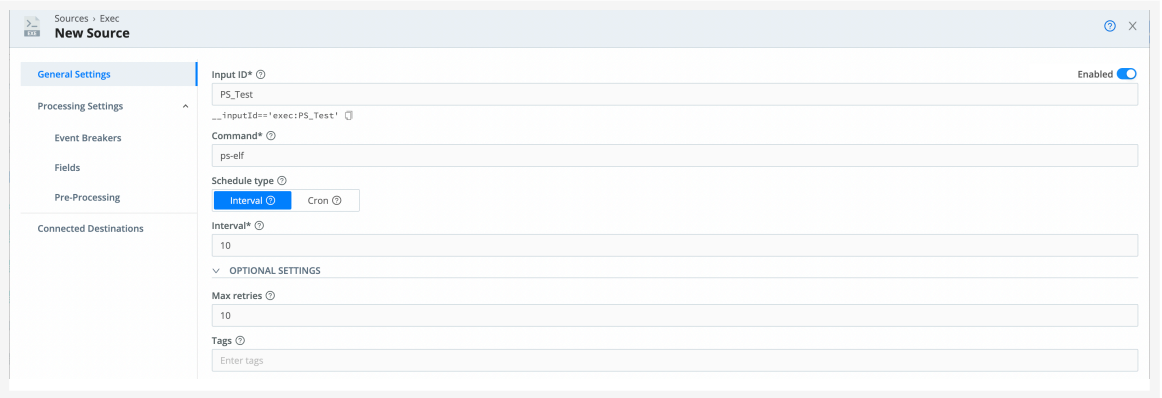
Next, we'll configure the Exec Source on your Edge Node. This Source will break the incoming streams of data into discrete events, and send them to Cribl Stream.

In this step, you can swap out the Exec Source by instead configuring a System Metrics or File Monitor Source. Or, configure multiple Sources to connect to the same Destination.

The Exec Source enables you to periodically execute a command and collect its `stdout` output. In the Exec Source's configuration modal, specify:

- Which command to execute.
- The number of times to attempt running the command.
- The interval between attempts.

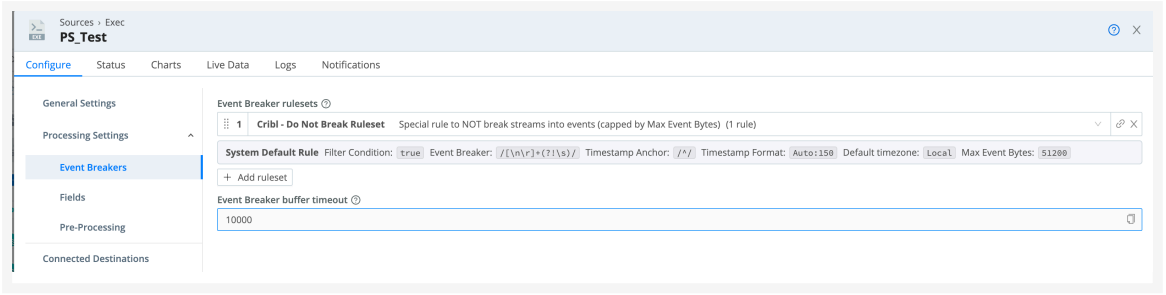
In our example, we are running the `ps` command to list and retrieve running processes every `10` seconds.



Configuring an Exec Source

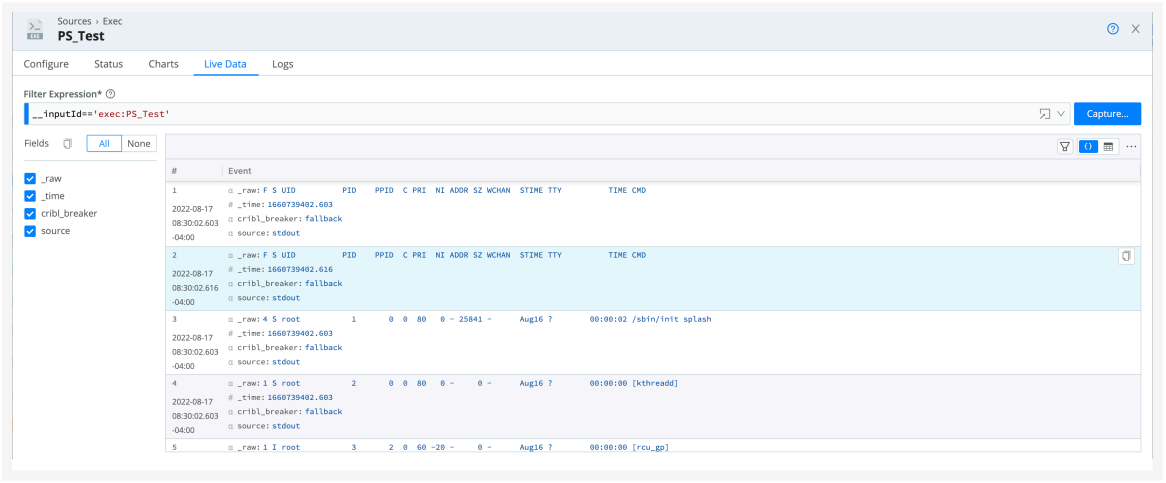
If we don't configure an Event Breaker, then with each capture we run on the dataset, each process will be ingested as its own event, without the header information. So to structure the data, we'll add an Event Breaker.

On the Exec Source configuration modal's left tab, select **Event Breaker**. In the **Event Breaker rulesets** drop-down, select `Cribl - Do Not Break Ruleset`.



Apply an Event Breaker

Next, preview your data on the modal's **Live Data** tab.



Preview Live Data

Configure the Cribl HTTP Destination on Cribl Edge

To get the data flowing, we'll configure the Cribl HTTP Destination on your Edge Node. A few things to note when configuring this Destination:

- The Cribl endpoint field in this Destination must point to the **Address** and **Port** you entered on the Cribl HTTP Source you configured in the first step.
- Optionally, define your **Compression**, **Throttling**, and **Backpressure behavior** requirements.
- On Cribl-managed Worker/Edge Nodes in Cribl.Cloud, make sure that TLS is either disabled on both the Cribl TCP Destination and the Cribl TCP Source it's sending data to, or enabled on both. Otherwise, no data will flow. In the Destination, TLS is disabled by default.

Once you've configured your Destination, test it to verify that your Edge Node can communicate with the Stream Worker Group.

Configure a Route to Send the Data

Finally, configure a Route to send your data to Cribl Stream. For this example, we can use the `passthru` Pipeline.

Confirm the Data Flow

To confirm that your data is flowing, navigate back to Cribl Stream's Cribl HTTP Source. Run a **Live Data** capture on the Source.

You can also check the Monitoring page > **Data** submenu, to isolate the throughput on your Source.

8.7. Transfer Data Between Workspaces Or Environments

Cribl Stream provides you with the ability to transfer data between Cribl Worker Nodes that report to the **same** Leader or different Leaders, via the Cribl TCP and Cribl HTTP Source and Destination pairs.

You can enable data sharing across Worker Nodes that do not share a Leader as long as both Organizations have **active, paid Cribl.Cloud plans** or **active, paid licenses**.

We designed this workflow to support scenarios where you can't or don't want to create a direct connection between Leader Nodes, such as:

Use case	Description
High/Low Security Environments	Isolated and protected network environments that handle sensitive data (for example, air-gapped networks managed by Cribl Edge) sending data to an external Cribl Stream processing tier.
Cribl Stream to Cribl Stream	Send data between Cribl Stream environments or Workspaces that exist within the overall Cribl system that you own and pay for.
Decentralized Management	Cribl Edge environments managed by separate Leaders forwarding data to a central Cribl Stream instance.

Key Aspects of Data Transfer Between Environments

Understand how you're billed for data transfers between environments and separate, paid Organizations.

Connectivity

The sending and receiving Organizations do not need to share a Leader. Use a paired **Cribl HTTP** or **Cribl TCP** Source/Destination.

Trust and Billing

- The **single ingest** benefit applies only when Workspaces are in the same **Cribl.Cloud Organization**, or when on-prem Leaders have the exact same paid license key installed.
- You can transfer data between separate paid **Cribl.Cloud Organizations**, but the receiving Organization will incur ingest charges.

Example: Customer to MSSP across separate Cloud Organizations

An enterprise customer (paid Cribl.Cloud Organization A) can send data via **Cribl HTTP** or **Cribl TCP** to their MSSP (paid Cribl.Cloud Organization B). This setup is supported. Because the

> Organizations are different, the **single ingest** benefit does not apply across them; the MSSP Organization will be billed for the received data.

Requirements and Caveats

- Both Organizations must have active, paid Cribl.Cloud plans or paid licenses.
- Workers on 4.11.1 and older can't share data with Workers on 4.12.0 and newer.
- For Cribl-managed Workers using **Cribl HTTP** in Cribl.Cloud, ensure **TLS settings match** between the Cribl HTTP Source and Destination (the Source enables TLS by default).

Limitations

You **can't** transfer data across Organizations in these scenarios:

- Workers on 4.11.1 and older communicating with Workers on 4.12.0 and newer.
- Free tier Workers communicating between themselves within the same Workspace.

How We Secure the Data Transfer

A shared-key mechanism validates and establishes trust between Leaders. The shared key will apply when the Leader of the sending environment has the exact same license key installed as the receiving environment, or when any Cribl.Cloud environment has Workspaces set up in the same Organization.

Set up Sources and Destinations to Transfer Data

Prerequisites / Before You Begin

Ensure the following criteria are met before configuring:

- Both of the environments for which you want to transfer data have an active billing contract or paid license.
- You have Leader networking configured to allow cross-environment communication between Worker Nodes.
- You have a Source configured to collect data (events, logs, metrics, files, etc). This Source will send the data to the Cribl TCP or Cribl HTTP Destination on the receiving Organization.
- On Cribl-managed Worker Nodes in Cribl.Cloud, make sure that TLS is either disabled on both the Cribl HTTP Source and the Cribl HTTP Destination it's receiving data from, or enabled on both. Otherwise, no data will flow. In the Source, TLS is enabled by default.

Choose Your Source and Destination Pair

You can send data between Organizations using one of these Source and Destination pairs:

- Cribl TCP Destination and Cribl TCP Source.
- Cribl HTTP Destination and Cribl HTTP Source.

For more information about these two options and how to decide which one is right for your Organization, read Interoperation Protocols.

Configure Cross-Workspace Data Collection

This guide will cover the Cribl TCP setup, but the general workflow is the same for the Cribl HTTP Source and Destination.

This workflow assumes you already have data ingested in Cribl Stream, and you want to move the data to a Worker Node in another Organization:

1. Configure the Cribl TCP Source in the Organization where you want to **receive** data.
2. Configure the Cribl TCP Destination in the second Organization. This Destination will **send** data to the receiving Organization.
    - In the **Cribl Endpoint** field, enter the Leader hostname in the **Address** field, in one of these formats:
        - Cribl.Cloud: `https://<your-workspace-name>.cribl.cloud`
        - On-prem: `http://localhost:<port>`
3. If your setup meets the criteria (see Key Aspects of Data Transfer for a refresher), you will see data flowing into the Cribl TCP Source from Workers that are not connected to the same Leader. Neat!

Troubleshooting

If you experience any issues when transferring data between Workspaces or Organizations, make sure you can access both environments. Cribl Support will need to engage all involved Organizations in any support case or call.

8.8. Integrating With Other Services

The following topics provide detailed guides and better practice for integrating with different external services.

Source	Description
AWS Cross-Account Data Collection	Collecting and writing data across multiple AWS accounts
Amazon S3 Better Practices	Read, write, and replay with S3
Amazon Security Lake Integration	Integrate Cribl Stream's Amazon Security Lake Destination with its Namesake Service
Collecting Logs from Amazon ECS Containers	Collect logs from Amazon Elastic Container Service (ECS)
Preparing the Azure Workspace	Preparing the Azure Workspace for Cribl Integrations
Azure Event Hubs Integrations	Ingest data from an Azure Event Hub
Microsoft Sentinel Integration	Set up Cribl to send data to the Microsoft Sentinel SIEM
Splunk Cloud Platform and BYOL Integrations	Integrating with the Splunk Cloud Platform by using the Splunk HTTP Event Collector
Switch Cribl Stream Destinations from S2S to Splunk HEC	How to switch your data collection from Splunk to Splunk (S2S) to HTTP Event Collector (HEC)
Splunk to Elasticsearch	Routing data from Splunk to Elasticsearch services
Splunk to Exabeam	Exporting parsed data to Exabeam
Splunk Stream to Cribl Stream	Sending data from Splunk Stream packages to Cribl Stream
Decryption of Data in Splunk	Decrypting data with Splunk
Stream-to-Splunk Encryption	Encrypting and decrypting data moving from Cribl Stream to Splunk
Syslog Best Practices	Best practices for processing syslog events
Palo Alto Syslog to Cribl.Cloud	Palo Alto Syslog Forwarding to Cribl.Cloud
Configure Upstream Logging Agents	Connect common logging agents to Cribl Stream
BigPanda/Webhook Integration	Configure Cribl Stream to send Webhook notifications to the BigPanda IT Ops platform

Source	Description
Gigamon to Cribl Stream	Sending data from Gigamon to Cribl Stream
System Metrics to Grafana	Sending data from System Metrics to Grafana
Kafka Authentication with Kerberos	This topic describes a procedure for configuring Kafka Sources and Destinations for Kerberos authentication.
Moogsoft/Webhook Integration	Configure Cribl Stream to send Webhook notifications to Moogsoft
Nightfall Integration	Integrating Cribl Stream with Nightfall to detect and redact sensitive information
Managing QRadar Licenses	Reduce QRadar costs with fine-grained control over events sent
Slack/Webhook Integration	This page describes how to integrate Cribl Stream with Slack via the Cribl Stream Webhook Destination.
Zscaler NSS Virtual Machine	Reduce the size of ZScaler logs with Cribl Stream
Tanium to Cribl Stream	Sending data from Tanium to Cribl Stream

8.8.1. Amazon

The following topics provide detailed guides and better practice for integrating with Amazon services.

Source	Description
AWS Cross-Account Data Collection	Collecting and writing data across multiple AWS accounts
Amazon S3 Better Practices	Read, write, and replay with S3
Amazon Security Lake Integration	Integrate Cribl Stream's Amazon Security Lake Destination with its Namesake Service
Collecting Logs from Amazon ECS Containers	Collect logs from Amazon Elastic Container Service (ECS)

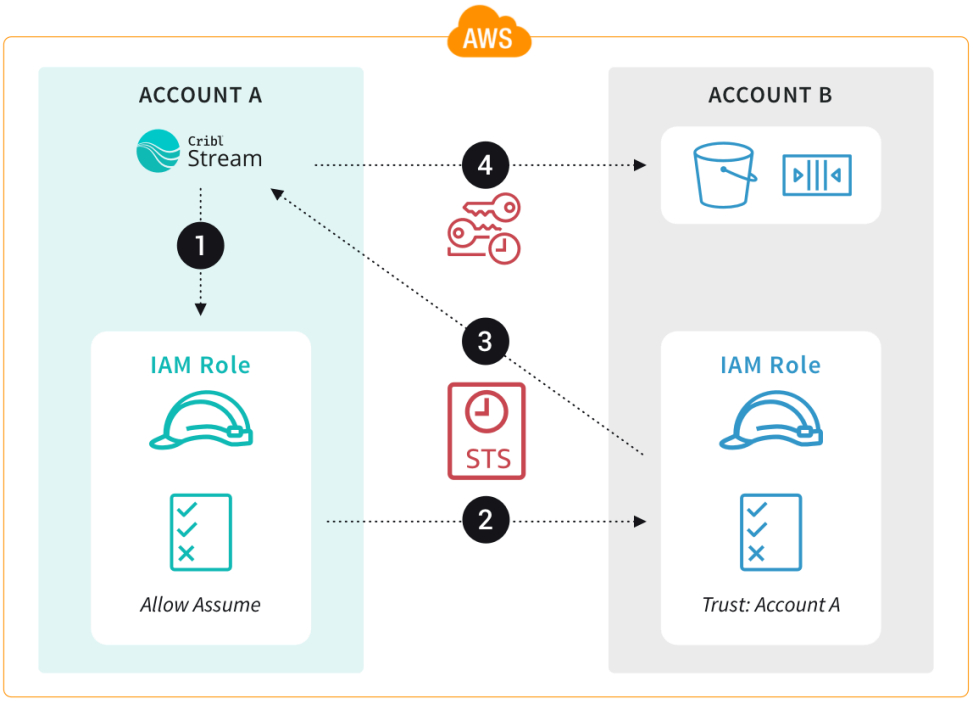
8.8.1.1. AWS Cross-Account Data Collection

Cribl uses multiple AWS accounts for different purposes, including our public Cribl.Cloud service (deployment details here). We built our account strategy from the ground up using the AWS Control Tower framework, as summarized in our Logging in a Multi-Account AWS Environment blog post.

However, some organizations use a legacy account structure that doesn't consolidate logs to a single account. This creates a challenge in collecting data or logs from peer accounts, as permission boundaries now need to be crossed. Whether you need to collect data from, or write data to, another account, the `AssumeRole` permission allows for cross-account access without the need to generate static IAM keys.

AssumeRole Example

A usage example is a central logging account that has access to other organization accounts to consume logs, but not to perform other actions. Let's say we have two AWS accounts, A and B. We want account A to be able to access resources in account B. We can build a policy inside account B that allows permissions to access the target resources. We can then specify, in account B, that we trust account A to be allowed to use this role. The diagram below illustrates how the `AssumeRole` action permits the Trusted Account (A) access to resources in the Trusting Account (B).



Using STS and temporary credentials to access an S3 bucket and SQS queue across accounts

Here's how Cribl Stream would work with `AssumeRole` permissions inside AWS:

1. The Cribl Stream Worker has an EC2 instance role attached.
2. The IAM role in Account A permits the EC2 instance to assume the role in Account B (and Account B trusts Account A).
3. Temporary IAM credentials are returned to the EC2 instance.
4. Cribl Stream uses the temporary IAM credentials to access the resources in Account B.

Since this guide's original publication, we've updated `logstream` to `cribl-stream` in example ARNs below, to match our renamed product.

Configure IAM AssumeRole Permissions

First, in account A, we build a policy that allows only the ability to assume the role inside account B. This policy restricts users from being able to access any resources that they don't need to see. We can also revoke this trust relationship at any time, without having to worry about an account still having keys and (therefore) access to the data.

In our example, we want to access VPC Flow logs inside an S3 bucket in account B (ID 222222222222) from account A (ID 111111111111). We'll start building the two policies in account B, and then move to account A.

Account B Configuration

In account B, we build a policy to enable access to the S3 bucket (`vpc-flow-logs-for-cribl`) with the least privileges required for Cribl Stream. This policy is the one that changes depending on what you need to accomplish - e.g., reading from an S3 bucket, writing to Amazon Kinesis Data Streams, etc.

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Action": "s3:GetObject",
     "Resource": "arn:aws:s3:::vpc-flow-logs-for-cribl/*"
   },
   {
     "Effect": "Allow",
     "Action": "s3:ListBucket",
     "Resource": "arn:aws:s3:::vpc-flow-logs-for-cribl"
   }
 ]
}
```

Copy

Next, we need to attach a Trust Relationship to Account B's IAM role to permit the `AssumeRole` action from account A:

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Principal": {
       "AWS": "arn:aws:iam::111111111111:role/account-a-cribl-stream-assumerole-role'
     },
     "Action": "sts:AssumeRole",
     "Condition": {
       "StringEquals": {
         "sts:ExternalId": "cribl-s3cre3t"
       }
     }
   }
 ]
}
```

Copy

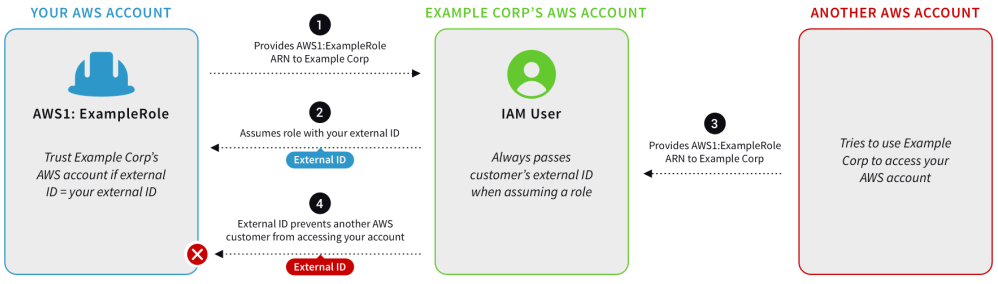
If you are creating a Trust Relationship for a Cribl.Cloud Worker managed by Cribl, you would instead paste the AWS Worker's role ARN in the format shown below. This option applies only to Cribl-managed Workers, not to customer-managed (hybrid or on-prem) Workers:

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Principal": {
       "AWS": "arn:aws:iam::111111111111:role/main-default"
     },
     "Action": "sts:AssumeRole",
     "Condition": {
       "StringEquals": {
         "sts:ExternalId": "cribl-s3cre3t"
       }
     }
   }
 ]
}
```

Copy

Why an External ID Condition in the Trust Policy?

It is important to configure an AWS External ID, especially if you have third parties accessing your AWS accounts. The External ID protects from the confused deputy problem, where a third party obtains access through an intermediary. The External ID is not a password or secret, but it should still be protected from accidental sharing.



Mitigating the confused deputy vulnerability

Account A Configuration

In account A, we next configure a new IAM role with the following policy.

If you are creating a Trust Relationship for a Cribl-managed Cloud Worker, skip this section. Cribl has implicitly preconfigured Account A for you.

For our example, we only want the role to be able to use the `AssumeRole` action, but you can add additional statements to meet your needs:

```
{
 "Version": "2012-10-17",
 "Statement": {
   "Effect": "Allow",
   "Action": "sts:AssumeRole",
   "Resource": "arn:aws:iam::222222222222:role/account-b-cribl-stream-role-to-assume'
 }
}
```

Copy

Since we need our EC2 instance to be able to assume this role, we will configure the Trust Relationship as follows:

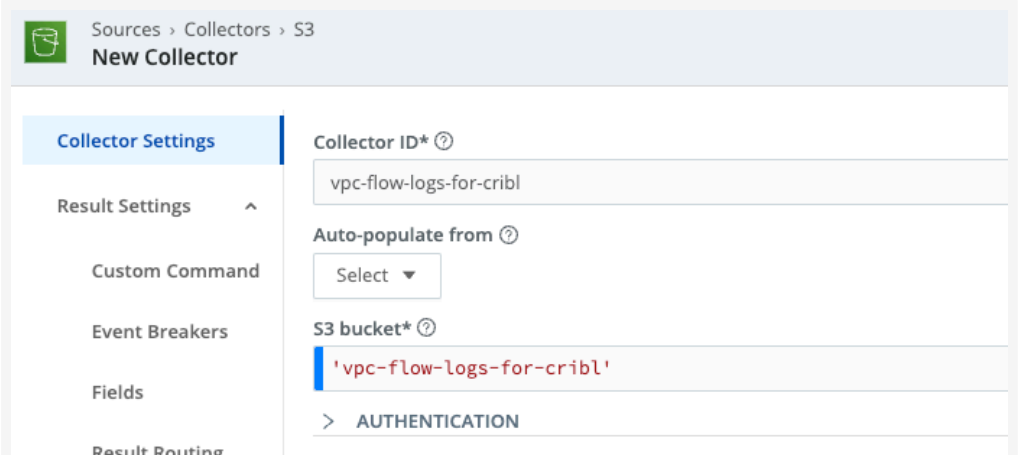
```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Effect": "Allow",
     "Principal": {
       "Service": "ec2.amazonaws.com"
     },
     "Action": "sts:AssumeRole"
   }
 ]
}
```

Copy

Configure Cribl Stream

Now that our `AssumeRole` policies have been built, we can configure Cribl Stream to assume the correct role to access the resources we need. Configure your Source, Collector, or Destination with the appropriate

`AssumeRole` ARN and External ID. While this screenshot shows an S3 collector specifically, all AWS sources and destinations support Assume Role functionality.



8.8.1.2. Integrate Cribl Stream With AWS Security Hub

You can integrate Cribl Stream with AWS Security Hub to enable your security teams to ingest, standardize, and route Security Hub findings in near real time. This integration streamlines security operations by providing a unified, flexible, and extensible data pipeline for AWS security events, leveraging the Open Cybersecurity Schema Framework (OCSF) v1.6 for consistent data structure and interoperability.

Key features include:

- Near real-time ingestion of AWS Security Hub findings via AWS EventBridge and Cribl Stream Webhook.
- Standardization of findings to OCSF v1.6, including support for AWS-specific extensions.
- Flexible routing to multiple destinations, including Cribl Lake, SIEMs, and data warehouses.
- Support for transforming and enriching third-party security findings.
- Secure, scalable architecture with recommended authentication and IAM permissions.

Architecture and Data Flow

This integration uses an event-driven architecture to deliver Security Hub findings to Cribl Stream for processing and routing:

1. AWS Security Hub generates a finding.
2. AWS EventBridge captures the finding event.
3. An EventBridge Rule filters and forwards events to a target.
4. The target is typically an AWS SNS Topic, which pushes the data to a Cribl Stream HTTP Endpoint (Webhook).
5. Cribl Stream receives the data, processes and standardizes it (for example, to OCSF v1.6), and routes it to one or more Destinations.

Configure Event Streaming from AWS Security Hub to Cribl Stream

The workflow to configure event streaming from AWS Security Hub to Cribl Stream consists of four parts:

1. Create an Amazon SNS topic.
2. Configure a Cribl Stream HTTP Endpoint as an SNS subscription.
3. Create an EventBridge rule in the AWS Management Console.
4. Enable the Webhook in Cribl Stream.

Create an Amazon SNS Topic

1. In the AWS Management Console, go to **Simple Notification Service (SNS)**.
2. Create a new topic (for example, `Cribl-SecurityHub-Findings`).

Configure a Cribl Stream HTTP Endpoint as an SNS Subscription

1. In Cribl Stream, navigate to Sources and enable an HTTP Endpoint (Webhook). Copy the unique URL.
2. In AWS SNS, create a subscription for the topic created above. The Protocol is `HTTPS`. For the URL, paste the Cribl Stream HTTP Endpoint URL.

Create an EventBridge Rule

1. In the AWS Management Console, go to **Amazon EventBridge**.
2. Select **Create rule** and provide a descriptive name (for example, `SecurityHub-to-Cribl`).
3. Define the event pattern as follows:
    - **Event source**: `AWS services`
    - **AWS service**: `Security Hub`
    - **Event type**: `Security Hub Findings - Custom Action` or `Security Hub Findings - Imported`
      Example format:

```
{
 "source": ["aws.securityhub"],
 "detail-type": ["Security Hub Findings - Imported"]
}
```

Copy

    - Set the target to the SNS Topic created in Step 1.

Enable the Webhook in Cribl Stream

1. In Cribl Stream, go to **Data > Sources**.
2. Select **Add new** and select **HTTP Endpoint**.
    - **Input ID**: Provide a unique ID (for example, `securityhub_webhook`).
    - **Authentication**: Configure Shared Secret authentication for security.

For advanced customization, consider using an intermediate Lambda or API Gateway.

- **Output**: Connect the source to a specific pipeline (for example, `securityhub_pipeline`) for OCSF conversion and routing.

Process and Standardize Data in Cribl Stream

Next, you can pre-process the data, standardize the JSON payload, and optionally enrich data with external or internal events.

1. Handle the SNS envelope/wrapper if present.
2. Convert the Security Hub JSON payload to OCSF v1.6 format, adding AWS context/extensions as needed.
3. Optionally add external or internal data (for example, CMDB lookups, CloudTrail events).

Route Findings and Manage Destinations

After processing, you can route Security Hub findings to multiple destinations:

Destination Service	Cribl Stream Destination Type	Purpose
Cribl Lake	Amazon S3 (managed by Cribl)	Long-term, cost-effective storage for historical analysis and compliance.
SIEM Platform	Splunk HEC, Kafka, ElasticSearch	Real-time security analysis, alerting, and incident response.
Data Warehouse	Amazon S3, Snowflake, Google BigQuery	Business intelligence and operational reporting on security trends.

This flexible routing ensures Security Hub findings are immediately available for analysis and archival across your security ecosystem.

Transform Third-Party Findings

Cribl Stream can also convert third-party security findings into OCSF v1.6, including AWS-specific extensions. This enables a unified approach to security data management, regardless of the original source format.

Original Source	Original Format	Cribl Stream Action	Output Format
Third-Party Tool 1	Proprietary	Convert and Add AWS Extensions	OCSF v1.6 with AWS Extensions

Original Source	Original Format	Cribl Stream Action	Output Format
Third-Party Tool 2	Custom JSON	Convert and Add AWS Extensions	OCSF v1.6 with AWS Extensions

Required Identity Access Management (IAM) Permissions

To enable robust integration, ensure the IAM role associated with your Cribl deployment has the following permissions (at minimum):

Endpoint	Permission(s)
`ec2_instances`, and so forth	`ec2:DescribeInstances`, `ec2:DescribeVolumes`, `ec2:DescribeSecurityGroups`, and so forth
`lambda_functions`	`lambda:ListFunctions`
`iam_*`	`iam:ListPolicies`, `iam:ListRoles`, `iam:ListUsers`, `iam:ListGroups`, `iam:ListMFADevices`
`cloudformation_*`	`cloudformation:ListExports`, `cloudformation:ListStacks`, `cloudformation:ListStackSets`
`dynamodb_backups`	`dynamodb:ListBackups`
`rds_*`	`rds:DescribeDBInstances`, `rds:DescribeDBClusterEndpoints`, and so forth
`cloudtrail_events`	`cloudtrail:LookupEvents`
`vpc_*`	`ec2:DescribeNetworkInterfaces`, `ec2:DescribeVpcs`, `ec2:DescribeSubnets`
`efs_file_systems`	`elasticfilesystem:DescribeFileSystems`

For a full list and details, see the Cribl documentation on AWS API permissions.

Security Best Practices

We recommend following these best practices to keep your AWS Security Hub integration safe:

- Always use Shared Secret authentication for HTTP Endpoints.
- Limit IAM permissions to the minimum required for your use case.

- Use AWS CloudFormation templates provided by Cribl for automated, secure setup of IAM roles and trust relationships.

Getting Started

If you are new to AWS Security Hub or Cribl Stream, refer to the following resources:

- Enable Security Hub in AWS
- Configure Cribl Stream

8.8.1.3. Amazon S3 Better Practices

The “better practices” in this guide all apply to Cribl Stream’s Amazon S3-based Sources and Destinations. Many also apply to other object stores, like Azure Blob Storage, MinIO, and Google Cloud Storage.

View the AWS S3 with Cribl Best Practices video presentation from Cribl Community Office Hours.

Foundations: Cribl Stream and Object Storage

In this first section, we’ll lay out four basic factual underpinnings:

- How Cribl writes to object storage.
- Write to a Cribl Amazon S3 Destination.
- Read from Cribl’s Amazon SQS Source.
- Replay from Amazon S3 buckets, via Cribl’s S3 Collector.

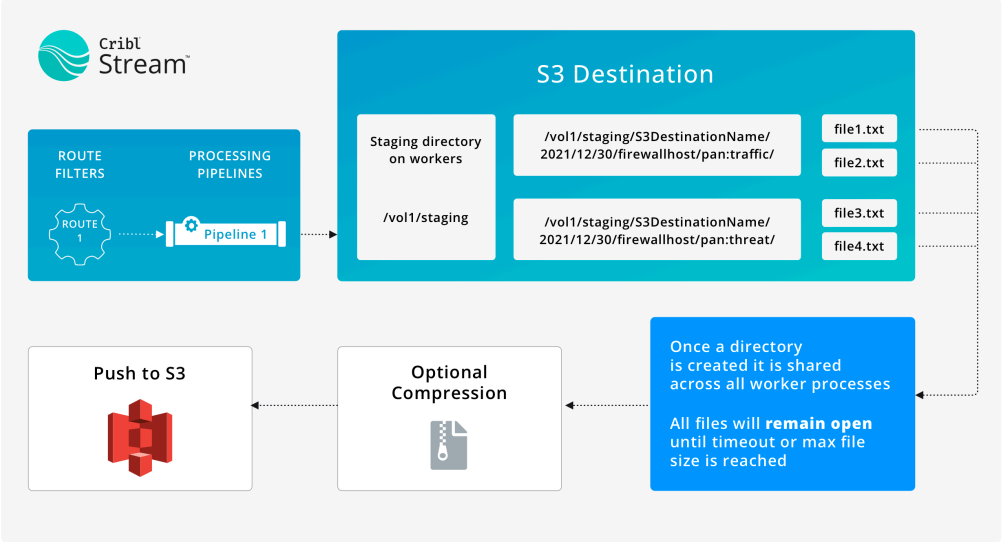
Then, this guide will go on to detail 10 “better practices” (and a bonus one!) as we cover four broad topics:

- Write to object storage, generically.
- Cardinality and partitioning expressions.
- Amazon S3-specific optimization.
- Better partitioning expressions.

How Cribl Writes to Object Storage

Cribl does not write data **directly** to its object storage Destinations, which include Amazon S3; any solution with an S3 API; and other object storage Destinations, like Azure Blob Storage, MinIO, and Google Cloud Storage.

Data persists locally to a “staging location” on the Workers. The Worker Processes append new data to the staging location based on partitioning expressions and configurations such as file size and max timeout settings. Each Worker Process, governed by its own specific settings, creates and maintains its own set of files.



Write to object storage - example

Write to a Cribl Amazon S3 Destination

When writing to our Amazon S3 Destination, or any object storage, consider two main settings: **File name prefix expression** and **Partitioning expression**.

Cribl allows a maximum of 2000 open files per Worker Process per S3 Destination. On a system with 14 Worker Processes, that translates to a maximum of 28,000 open files per S3 Destination. On a system with 30 CPUs, that translates to 60,000 open files.

When a specific Worker Process hits the maximum open files, it closes all open files, moves them to the final output location, and creates new ones. If you have multiple object storage Destinations configured in Cribl, ensure that the total maximum open files across all these Destinations is less than your system's configured maximum open files. To figure out this maximum, multiply the number of file-based Destinations by max open file limit for each such Destination and by number of Worker Processes (Destinations x max open file limit x Worker Processes).

The default maximum open file settings on many Linux systems are too low for Cribl Stream when writing to an object storage Destination. You'll need to increase your system max open files and/or process max open files settings, as explained below.

Additionally, to limit the number of files in each directory, **Partitioning expression** should specify dates down to the hour or minute.

Read from Cribl's Amazon SQS Source

When reading data using Cribl Stream's Amazon SQS Source, a few settings can greatly speed up data retrieval. These are:

- **General Settings** > **Queue**, a.k.a. the file name filter.
- **Advanced Settings** > **Max messages**.

- **Advanced Settings** > **Num receivers**.

Even more importantly, data retrieval speed is also governed by Destination settings and Pipeline efficiency.

Replay from Amazon S3 Buckets

When replaying from S3 via Cribl's S3 Collector, optimize your performance by adjusting the following:

- **Optional Settings** > **Path**, a.k.a. the filter.
- Pipeline efficiency.
- Destination settings.
- Destination health.

The **Path** should specify the time down to the hour, or in some cases, down to the minute.

Also, consider the API limits for your object storage bucket. AWS sets a limit of 3,500 writes/second and 5,500 reads/second per S3 prefix. An S3 prefix is a path on Amazon S3, including bucket name and any subdirectories up to but not including the object name. The S3 prefix always ends in (and includes) the slash before the object name. In its simplest form, the S3 prefix is just `BucketName/`.

AWS API limits apply to all attempted access to these prefixes, cumulatively. You probably won't hit the write limits while putting the "better practices" in this guide into effect. However, populating too many files in any one directory might trigger throttling when reading and performing replays.

Some Cribl Destinations have **File prefix expression** and/or **Partitioning expression** settings. Do not confuse these with the AWS prefix.

Write to Object Storage, Generically

When you deploy a Cribl Amazon S3 Compatible Stores Destination, start with a simple global **Partitioning expression** and the default **File name prefix expression**.

This will work as long as cardinality remains below 2,000 during the max timeout period. Also, apply the procedures to increase the system max open file settings, as described later in this guide.

Better Practice 1: Mind the Partitioning Expression

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d/%H')}/${index ? index
```

Copy

If cardinality exceeds 2,000, you can either:

- Remove the less-important fields from the expression; or
- Create different partitioning expressions for different event types, categories, etc.

On the **Advanced Settings** tab:

- Match the **Max open files** to cardinality, without exceeding 2,000.
- Toggle **Remove staging dirs** on.
- Set the **Storage class** to `Intelligent Tiering`.

Still in **Advanced Settings**, keep the default settings for:

- **Max file size (MB)**: `32`.
- **Max file open time (sec)**: `300`.
- **Max file idle time (sec)**: `30`.

As you gain deeper understanding of the data, consider the following options:

- Create a set of different S3 Destinations, each with its own **Partitioning expression**. They can all point to the same S3 bucket. Be aware that doing so will increase the number of API requests because events will be spread over more files.
- If the numbers of open files becomes too large, discuss alternative architectures (e.g., creating additional Worker Groups) with your Cribl Account team or Support.

Better Practice 2: Optimize System Settings for Max Open Files

Consider the workflow for writing to Cribl S3 Destinations:

- Each Cribl Worker Process creates its own set of files in a staging directory based on **Partitioning expression** and **File name expression**.
- The maximum partitioning expression cardinality allowed is 2,000. This translates to a maximum of 2,000 open files per Worker Process per object storage Destination.
- Each Destination writes files to its staging directory until the number of files exceeds the configured maximum, or the Destination reaches any of the following limits: **File size limit (MB)**, **Idle time limit (sec)**, or **File open time limit (sec)**.

When writing to an object store, more Worker Nodes with fewer CPUs is better than fewer Worker Nodes with more CPUs (e.g., 5 Workers with 12 vCPUs is better than 2 systems with 32 vCPUs).

You should apply this principle to realize two benefits:

- To maximize the number of open files.

- To have more S3 Destinations with more precisely-focused partitioning expressions.

Linux defaults for maximum open files tend to be too low for writing to Cribl S3 Destinations, but the 65,536 limit recommended by Cribl is a good starting point. When setting the maximum open files substantially higher than 65,536, you should track system health.

You can further increase the maximum number of open files allowed by your system based on the number of S3 Destinations you are planning to use and the number of Worker Processes on each Node. The table below provides some examples:

# of S3 Destinations in Cribl	Max cardinality per Dest.	Open file limit in Cribl Dest.	# Worker Process on each Worker Node	Min Linux per-process open file limit	Min Linux system plus Cribl user open file and 1k reserved for the system
1	2,000	2,000	10	3,000	21,000
1	2,000	2,000	14	3,000	29,000
1	2,000	2,000	22	3,000	45,000
2	2,000	2,000	10	5,000	41,000
2	2,000	2,000	14	5,000	57,000
2	2,000	2,000	22	5,000	89,000
5	2,000	2,000	10	11,000	101,000
5	2,000	2,000	14	11,000	141,000
5	2,000	2,000	22	11,000	221,000

Better Practice 3: Update Max Open File Settings on Hosts

Here are basic instructions that cover many Linux distributions. (Check the documentation for your specific Linux distribution and version for any variations required.)

Edit the following settings. You will be required to reboot.

```
# Update `fs.file-max` to a very large value run (as root):
sysctl -w fs.file-max=<open-files-limit>

# or edit sysctl.conf (as root):

vi /etc/sysctl.conf
fs.file-max = <open-files-limit>
```

Copy

```
# vi /etc/security/limits.conf (as Cribl user)
* soft nproc 65536
* hard nproc 65536
* soft nofile 65536
* hard nofile 65536
```

Copy

After you reboot, verify the settings:

```
# sysctl -p
# cat /proc/sys/fs/file-max
# sysctl fs.file-max
```

Copy

Cardinality and Partitioning Expressions

As we've seen, partitioning expressions and file expressions interact to determine cardinality. In this section, we'll examine this interaction more closely and learn better practices based on what we find.

Where Cardinality Begins

Partitioning expressions specify the directory structure that Cribl adds when writing to an object storage Destination.

For example, the following partitioning expression extracts the `year`, `month`, `day`, `hour`, `minute`, `index`, `host`, `sourcetype`, `server_ip`, `status_code`, and `Method`, and populates them as the directory structure.

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d/%H')
/${index ? index : 'no_index'}
/${host ? host : 'no_host'}
/${sourcetype ? sourcetype : 'no_sourcetype'}
/${server_ip}
/${status_code}
/${Method}/`
```

Copy

A file expression prefix comes into play next. The default file expression is just a random file name. However, an expression like this would create files that also include the hour and minute in each file name. This reflects the time the file was created, not the time of each event.

```
${C.Time.strftime(_time ? _time : Date.now() / 1000, '%H%M')}
```

Together, the partitioning expression and the file expression must not produce a cardinality greater than 2,000.

Calculate Cardinality

Cardinality is the quantity of combinations of unique values, within a time period, for each parameter in the expressions.

For example, a partitioning expression could specify the following:

- Within an hour.
- 5 unique sourcetype values.
- 10 unique host values.
- 2 unique source values for each host.

For this example, we could calculate the maximum cardinality as `(5*10*2)=100`.

It's important to also consider the timeframe this is calculated within. That cardinality might be 100 over a 1-minute period, but might be 500 over a 5-minute period.

If we further add a file name prefix expression, this increases the cardinality. For example, let's add the method and top-level URI as file name prefix parameters. If there are 3 possible methods, and 10 URIs, the cardinality just went up from 100 to `100*3*10=3,000`.

Cardinality has the greatest impact on how you should configure an object storage Destination in Cribl Stream. For every combination of partitioning expressions, file prefix expressions, and Worker Processes, there can be

an open file on the Worker Node's system.

A partitioning expression and file prefix expression should produce fewer than 2,000 potential combinations. When a Worker Process exceeds the configured **Open file limit**, this generates errors; all open files for that Worker Process are simultaneously closed; and, new files will be created.

The Worker Process might experience some extra load while all files are processed. If this occurs infrequently, that's generally okay. But if it happens several times per hour, you should consider how to improve the combination of partitioning and file expressions.

As a starting point, consider these two strategies:

1. Write to one S3 bucket, incorporating parameters common to each event.

For example, here's a partitioning expression that specifies year, month, day, hour, minute, index, sourcetype, source, and host:

```
${C.Time.strftime(_time ? _time:Date.now()/1000,'%Y/%m/%d/%H/%M')/${index}/${host}/${
```

Copy

2. Dedicate different S3 Destinations to different event types. During replays, this makes searching specific data faster. However, this benefit comes at the expense of needing a larger number of open files. Should you pursue this option, test large numbers of open files on a test system in your environment. Such testing is especially crucial in virtualized environments with shared disk, CPU, and memory resources.

Better Practice 4: Select Fields for Partitioning and File Expressions

The main factors to consider when selecting fields for a partitioning or file name prefix expression are:

- Configure Time-based filtering: All partitioning expressions should include `year`, `month`, `day`, `hour`, and optionally, `minute`. This facilitates filtering during replays. It also limits the files in any one directory, so that replays are not throttled when returning large volumes of files.
- Identify Universal fields: Fields that users will typically search on during replays across all datasets. Examples include `sourcetype`, `host`, `source`, `category`, and `index`.
- Optionally, configure specific fields for different datasets.
  Examples include:
  - Web logs: server IP, method, status code.
  - Firewall logs: source zone, destination zone, accept/deny status.
  - Flow logs: protocol, accept/deny status.

  Be careful when pursuing this option as the number of open files can grow exponentially with too many multiple partition and file name prefix expressions.

Analyze Partitioning Expressions

Let's work through some examples of partitioning expressions, with hypothetical possible values for each parameter, to see what cardinalities they produce.

Example 1

This one produces reasonable results.

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d/%H')}
/${sourcetype}/${host}/${AttackType}`
```

Copy

Possible values:

- `sourcetype`: 5.
- `host`: 100.
- `AttackType`: 5.

Potential cardinality: `5 x 100 x 5 = 2,500`.

Example 2

Here, too, cardinality is kept commendably low.

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d/%H')}/${state}/${inte
```

Copy

Possible values:

- `state`: 2.
- `interface`: 50.
- `dest_ip`: 20.

Potential cardinality: `2 x 100 x 10 = 2,000`.

Example 3

This is a bad partitioning expression which produces out-of-control cardinality. Back to the drawing board!

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d/%H')}/${state}/${inte
```

Copy

Possible values:

- `state`: 2.
- `interface`: 100.
- `src_ip`: 65,535.

Potential cardinality: `2 x 100 x 65,535 = 13,107,000`.

For more examples like these, see Better Partitioning Expressions.

See How Cardinality Changes Over Time

In the examples so far, we've assumed typical expected values for parameters. Another useful exercise is to estimate the maximum possible values of each parameter, and multiply each value to obtain maximum theoretical cardinality.

You should also investigate how cardinality changes depending on when and how long you measure it. To do this, query/search your analytic tool to determine cardinality for different durations over several days.

Run the searches over different duration spans (e.g., 1 minute, 3 minutes, 5 minutes) to determine the best timeout period. For example, your cardinality might be 2,000 over a 2-minute span, but 10,000 over a 5-minute span. In this case, it is most certainly beneficial to set the maximum file timeout to 2 minutes.

Below are examples of Splunk searches over different span periods, with maximum open file time set to 3 minutes. Change the fields to the ones you'd prefer to incorporate into a partition and/or file name prefix expression, and try running them in your environment.

Here's an example using a "traditional" search method:

```
earliest=-24h index=*
|bin _time span=3m
| stats dc(source) as dc by host, sourcetype _time
| stats sum(dc) by _time
```

Copy

Here's an example using `tstats`:

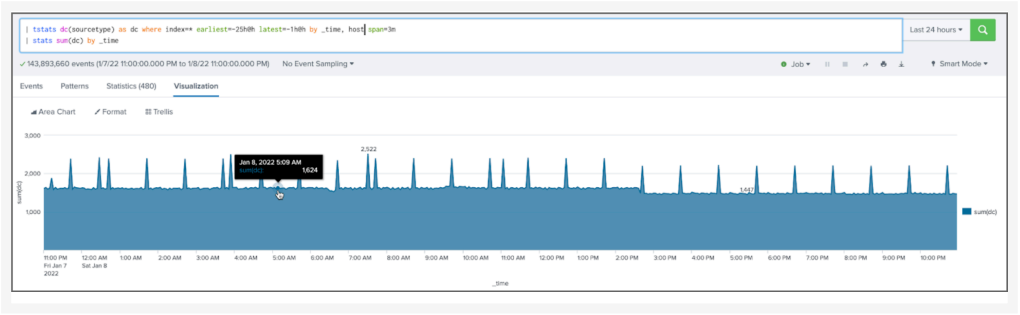
```
| tstats dc(source) as dc where index=* earliest=-25h@h latest=-1h@h by _time, host,
| stats sum(dc) by _time
```

Copy

The `tstats` search produces the results shown in the diagram below.

- Cardinality mostly stays below 1,625.
- There are some hourly peaks around 2,500. These spikes, which exceed the max allowed cardinality of 2,000, are infrequent.

All of this is generally acceptable.



cardinality Check Results

Better Practice 5: Size a Separate Fast(er) Disk for Staging Location

The system's ability to handle enough open files is a key factor that drives performance. This requires:

- Setting **Open file limit** high enough.
- Ensuring that the staging directory on the Workers can handle simultaneous writing tens of thousands of open files. The metric to use here is Input/Output Operations per Second (IOPS). A good starting point is 2,000 IOPS.

Cribl recommends that you allocate a separate disk for the staging directory on Workers, especially when you have configured multiple object storage Destinations. Follow these general, conservative guidelines:

- Toggle **Advanced Settings** > **Remove staging dirs** on. This means that any empty directory that has not been written to for an hour, is deleted.
- Disk size of 500 GB should be adequate, since the Workers need only enough disk capacity for a few minutes' worth of data. 500 GB errs on the side of caution, allowing us to store as much as 4 hours of data to disk. See the formula and examples below.
- Because Cribl writes to the staging directory only when the object storage Destinations are healthy, disk size does not usually become an issue. If the Worker encounters or triggers a backpressure situation, it stops writing to the staging directory.

- The staging directories are not intended for high availability, but rather to support creation of large-enough files to write to object storage.

Replay from a dedicated Worker Group. Do not compete for resources against production, real-time, streaming resources to perform replays!

Determine Staging Directory Disk Size

You can calculate an optimal size for your staging directory disk using the following formula:

- `Data Volume` (GB/day) you expect to ingest, times `1.25` metadata factor; divided by
- `1440` (minutes in a day), times
- `Duration` (minutes) you want to store data on disk; divided by
- `Number of Workers` writing to a staging directory.

The result will be disk size per Worker.

The figure below illustrates the formula.

$$\frac{Data\ Volume\ (GB/day) \times 1.25\ metadata\ factor}{1440\ mins/day} \times mins\ to\ store\ data\ to\ disk \div number\ of\ Workers$$

Here are some examples:

Data Volume	Number of Workers	Duration (Minutes)	Disk Size per Worker
1 TB/day	3	120	35 GB
10 TB/day	5	240	417 GB
50 TB/day	45	240	232 GB

Amazon S3 Optimization

Once you configure your Amazon S3 permissions, verify access, and (if applicable) discover your Cribl.Cloud Organization's AWS ARN, you can start applying some more better practices.

Amazon S3 Permissions

If your Workers are hosted outside AWS, you will need an access key and secret key to access the S3 bucket. If the Workers are EC2 instances or in AWS EKS, it's most ideal to assign an EC2 service role to the instances. The role will need access to the S3 bucket. For policies for reading and writing to/from Amazon S3, see AWS Cross-Account Data Collection.

Here's a policy to grant a role, or user access, with read and write permissions the S3 bucket. Just change the bucket name in this policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::bucketname/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": "arn:aws:s3:::bucketname"
    }
  ]
}
```

Copy

If `assumeRole` is required, here's a sample policy to assume the role that has access to the S3 bucket:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:role/account-a-cribl-stream-assumerole-role
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "cribl-s3cre3t"
        }
      }
    }
  ]
}
```

Copy

Test your policy with AWS's IAM policy simulator to ensure there are no boundary policies that might conflict with this policy.

Verify S3 Access

To troubleshoot access to your S3 bucket, install the AWS CLI on one of the Workers. Log in using your AWS user and Region. Assume the role if necessary. Then, attempt to download a file from the bucket or write a new file to the bucket.

Example of assuming a role:

```
aws sts assume-role --role-arn "arn:aws:iam::123456789012:role/example-role" --role-s
```

Copy

Example of validating the role you'll use to run subsequent commands:

```
aws sts get-caller-identity
```

Copy

Example of listing files in a bucket:

```
aws s3 ls s3://s3-bucket-name
```

Copy

Example of sending stdin to a specified bucket:

```
aws s3 cp - <target> [--options]
```

Copy

Example of writing a local file to the bucket:

```
aws s3 cp localfilename.txt s3://bucket-name
```

Copy

Example of reading a file from S3:

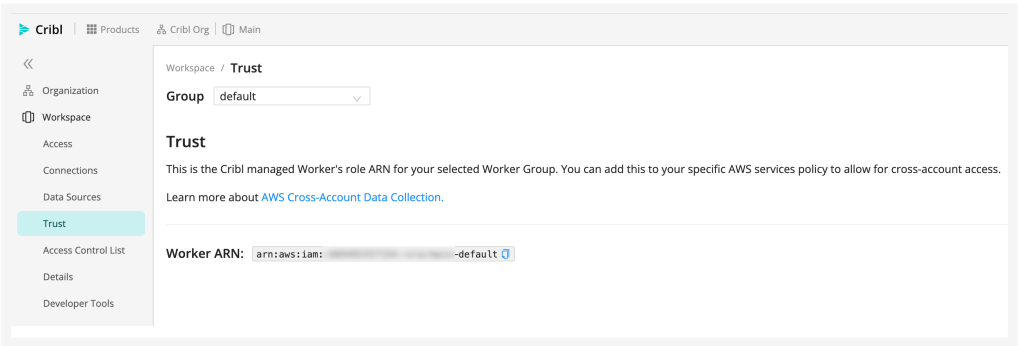
```
aws s3 cp s3://bucket-name localfilename.txt
```

Copy

Access Amazon S3 from Cribl.Cloud

Every Cribl.Cloud Organization exposes its own unique AWS ARN.

From the main Cribl.Cloud portal, navigate to the **Workspace** you are working in, and select **Trust** from the sidebar to expose the Cribl.Cloud ARN.



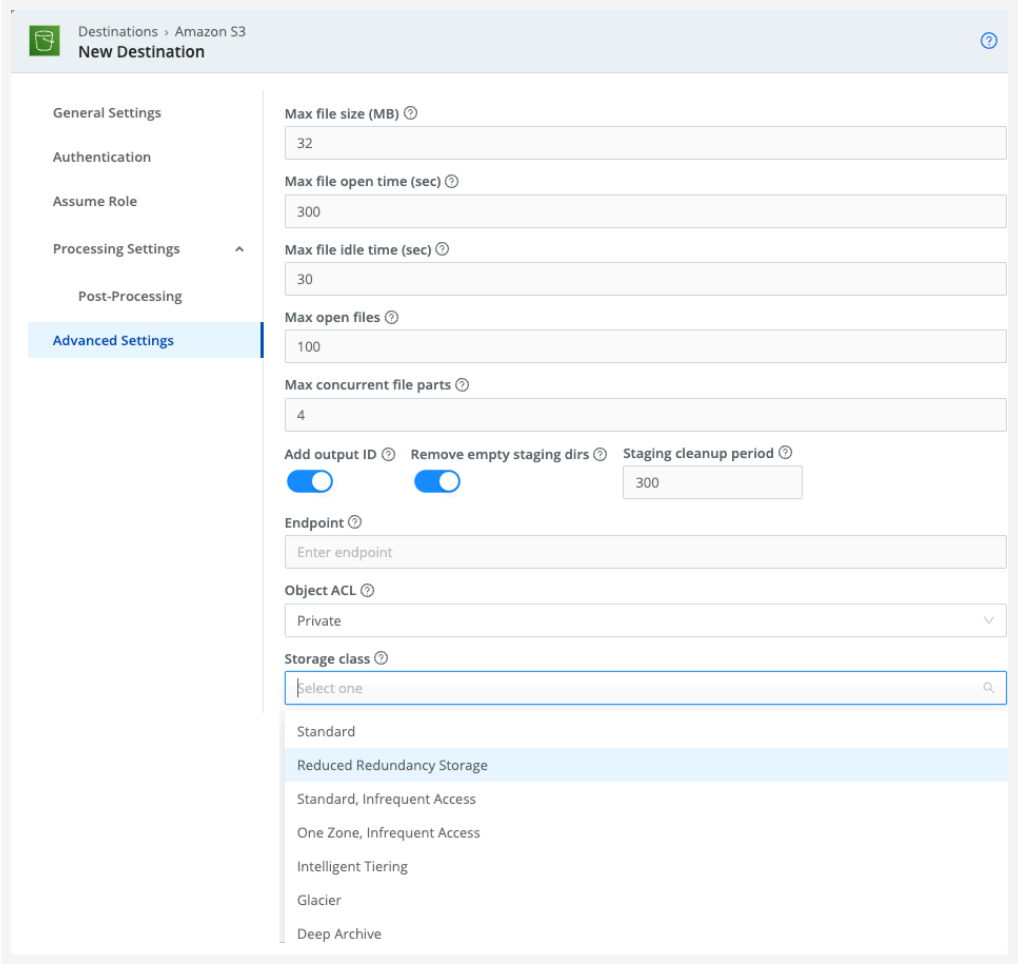
Cribl.Cloud ARN

Configure your AWS account with access to the S3 bucket, to allow Cribl.Cloud to assume that role, as illustrated above in Amazon S3 Permissions.

Better Practice 6: Tuning Amazon S3 Destination Settings

Under the Amazon S3 Destination's **General Settings** and **Advanced Settings** tabs are several settings for tuning your Amazon S3 Destination. Below are some recommendations for tuning those settings, in order of importance:

1. **Staging Location**: Change this to a directory on the Workers with increased IOPS, to accommodate thousands or tens of thousands of simultaneous open files. (Cannot be configured on Cribl.Cloud-managed Workers.)
2. **Remove staging dirs**: This setting must be enabled, so as not to leave millions of empty directories on the system over time.
3. **Compress**: Always enable compression, to reduce the size of files uploaded and downloaded from S3. This can reduce your S3 bill by 8-15x. Note that compression does add to the processing time - but transfer time will be substantially faster.
4. **Max open files**: Adjust this to be slightly larger than the data's cardinality, but not greater than 2,000. Also, ensure that the system can support the number of open files for all Worker Processes, and across all configured S3 Destinations. This value is set per process. 32CPUs*100 - 3200 max open files. Increase this value to avoid "Too many open files" errors. Default open file limit on Linux is 1024/process.
5. **Max file size (MB)**: If you have a separate S3 Destination for high-volume data sources, such as VPC Flow Logs or firewall logs, consider raising this value in order to create fewer files. Start with 64 MB. Otherwise, the default 32 MB setting is generally acceptable.
6. **Max file open time (sec)**: First, determine your cardinality over 1 minute, 3 minutes, 5 minutes, and 10 minutes. Adjust this setting to match the lowest-cardinality period. E.g., if cardinality is 10,000 over 5 minutes, but 2,000 over 90 seconds, change this setting to 90 seconds. If cardinality is 2,000 over 5 minutes, then the default 5 minutes (300 seconds) is acceptable.
7. **Max idle time (sec)**: The default 30 seconds is generally acceptable. In cases with high cardinality, lowering this setting can result in fewer open files when processing bursty data.
8. **Data format**: If using a Passthru Pipeline, change this setting to `Raw`. Otherwise, leave it at the JSON default to preserve all metadata added to the events.
9. **Storage class**: For Amazon S3 Destinations, consider the "infrequent access tier" or "intelligent tier" to reduce overall cost if the data will be rarely searched. Examples of rarely-searched data include threat-hunting or compliance-reporting use cases. "Standard, Infrequent Access" and "Intelligent Tiering" options should be used on low-access use cases such as threat hunting or compliance audit.
10. **AWS Region**: Leave this field blank for non-AWS Destinations, unless instructed otherwise. For AWS, set this to the nearest geographic location to the Workers.

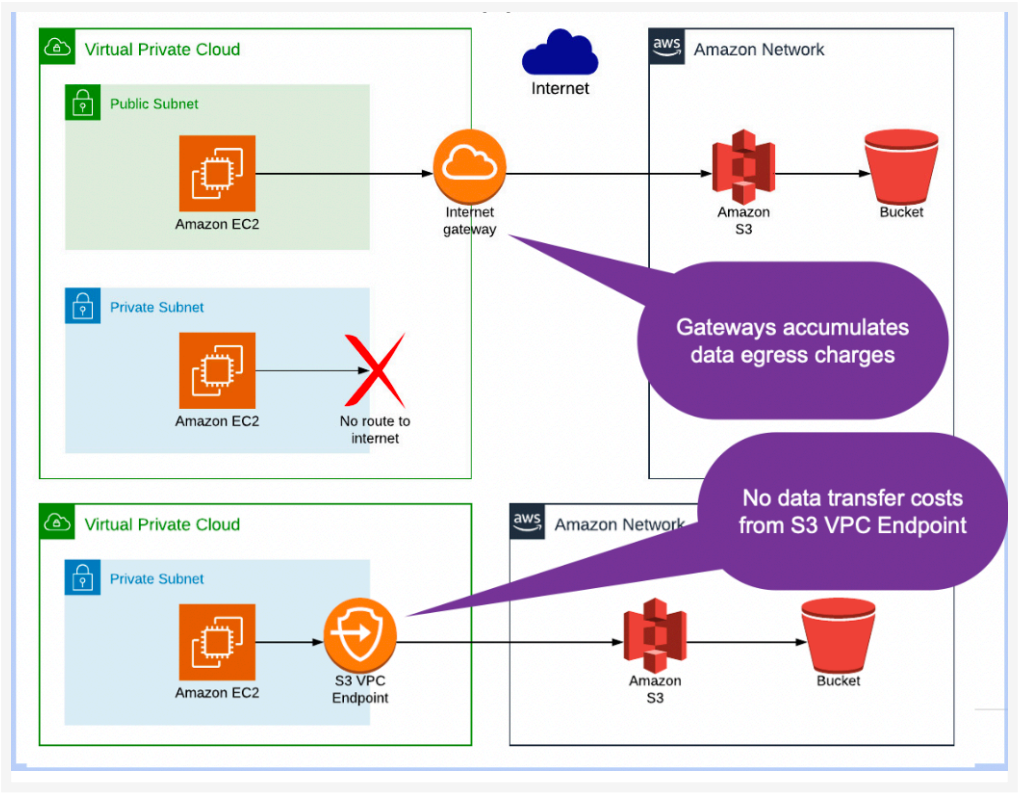


Configuring S3 Destination Settings

Better Practice 7: Lower AWS Costs

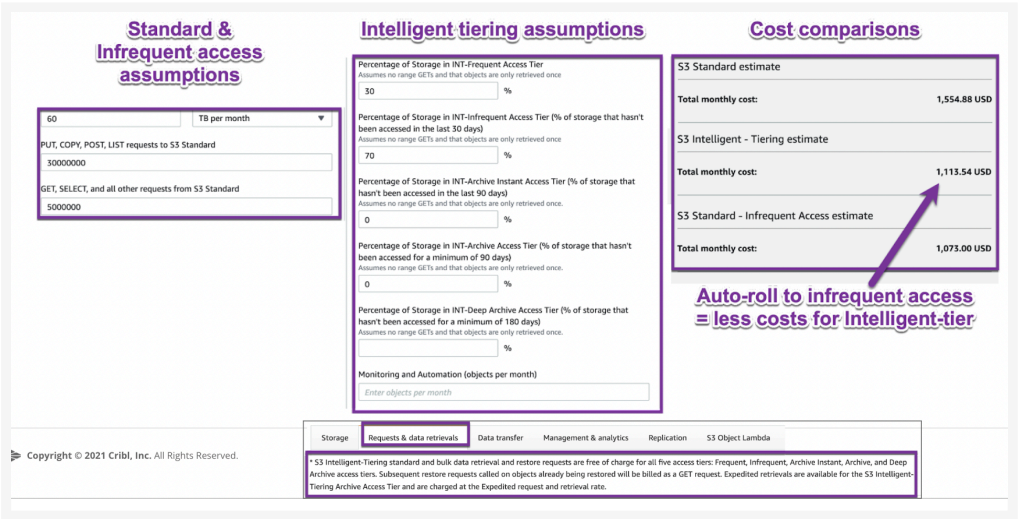
Lowering AWS costs can be achieved by a few means. First and foremost, ensure that compression is enabled on your Cribl S3 Destination and set to `gzip`. This will reduce both egress costs and S3 storage costs.

To eliminate egress costs from your Workers to S3, map a VPC endpoint on the S3 bucket to the VPC where your Workers reside.



Mapping a VPC endpoint to an S3 bucket

Writing to S3's Intelligent storage class, or infrequent access tier, can reduce long-term costs. Intelligent storage automatically routes data to cheaper S3 tiers when it is not searched or touched for certain periods of time. Consult the AWS Cost Estimator to calculate the differences. Old data can also be rolled off to a much cheaper option such as Glacier Instant Retrieval or Glacier Deep Archive. Below are sample cost comparisons:



AWS Cost Comparison

Cribl Stream Collectors and Replay features don't support S3 Glacier or S3 Deep Glacier due to long retrieval times for those storage classes.

> Collectors and Replay features do support S3 Glacier Instant Retrieval when you're using the S3 Intelligent-Tiering storage class.

Better Practice 8: Monitor Amazon S3 Buckets

AWS may limit S3 access if cumulative API writes are greater than 3,500 writes/second, or cumulative API reads are greater than 5,500 reads/second for a prefix (a.k.a., directory within S3). While it would be impossible to track every single AWS prefix's API requests, Cloudtrail offers "rate exceeded" events if you are ever over the API request limits.

Here's a Splunk search to report on all Cloudtrail events for S3. Filter or look for rate exceeded within the `errorMessage` field:

```
index=aws sourcetype=aws:cloudtrail errorCode=* "resources{}.ARN"=* | rename "resources{}.ARN" as ARN | stats count by eventName errorCode errorMessage ARN
```

Cloudtrail Logs Search

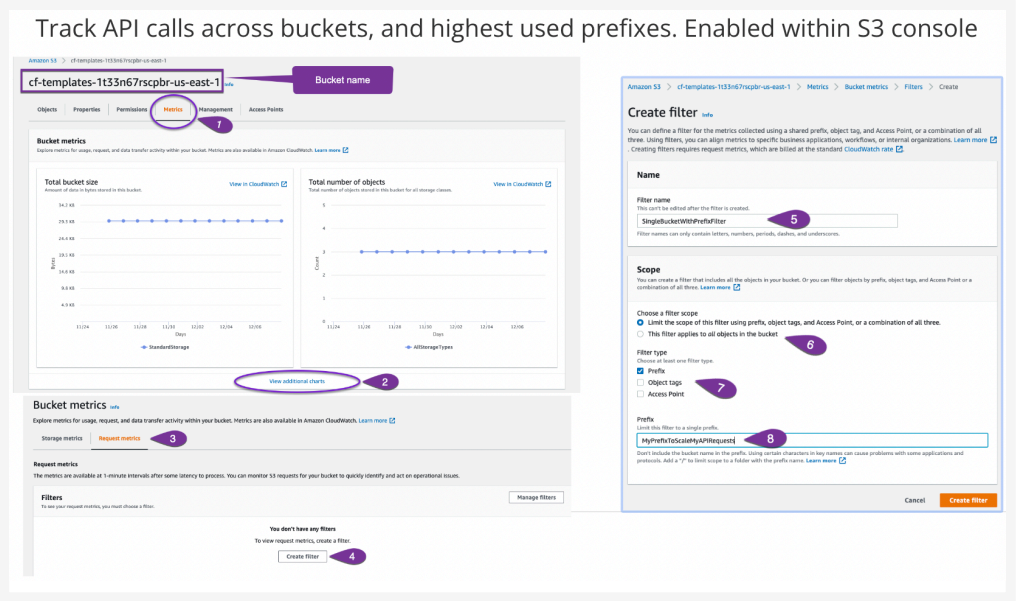
If there are `rate exceeded` messages from writing to Amazon S3, then a new strategy for partition and file name prefix expressions might be warranted. You will want to create fewer files, so consider one or more of the following options:

- Increasing timeouts.
- Eliminating file name expression prefixes.
- Updating partitioning expressions.

If there are rate exceeded messages on reading from Amazon S3, then you should likely apply better filtering. This might also warrant a new strategy for how files are written to S3.

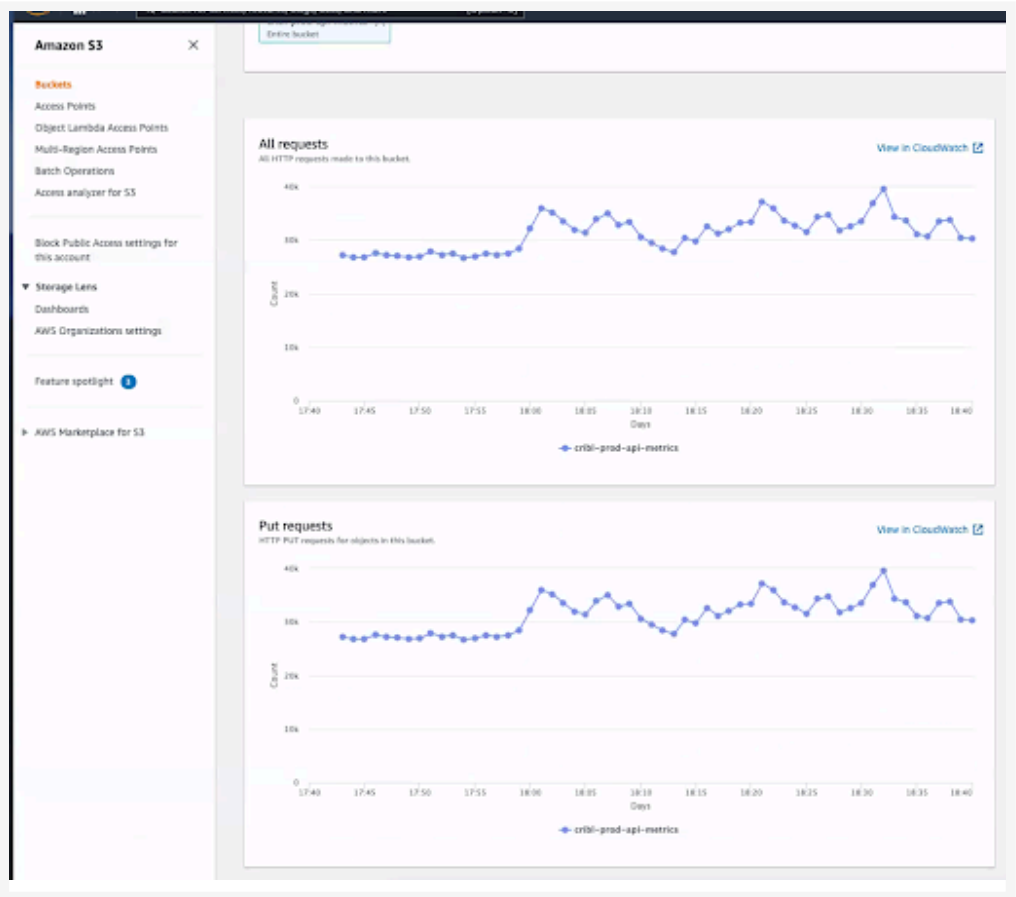
For trending API calls, you'll need to focus on a specific prefix. This might be useful for specific AWS prefixes, as there will be hundreds of thousands or millions. API calls can be tracked via AWS Cloudwatch, but first need to be enabled for tracking within the S3 bucket. See AWS' Creating a Metrics Configuration that Filters by Prefix, Object Tag, or Access Point topic.

The screenshot below illustrates the locations to enable tracking API calls for a specific prefix:



Enable Monitoring in S3

Here's a Cloudwatch chart trending all requests, and write requests, for an S3 prefix. This is granular to 1 minute. Because the API limits are per second, some extrapolations are necessary to estimate per-second values.



Cloudwatch Chart

Consider both the write and read behavior when designing your partitioning expressions and file name prefix expressions.

Better Practice 9: Monitor Worker Node Performance, and Enable Notifications

Aside from the standard CPU load, disk space, memory usage metrics, consider tracking some additional metrics. If the Worker Nodes are deployed as EC2 instances, track the Cloudwatch metrics for the EC2 instances and associated disks. Cribl Support will ask for these metrics to track Amazon S3 issues:

- `BurstBalance`
- `VolumeReadBytes`
- `VolumeWriteBytes`
- `VolumeReadOps`
- `VolumeWriteOps`
- `VolumeQueueLength`

On the Linux host, the following command allows you to track the number of open files. This would be another useful metric to track over time:

```
lsof | grep <common_upperlevel_staging_dir> | wc -l
```

You can also configure Notifications in Cribl Stream when the S3 Destination is unhealthy or triggering backpressure. Once the S3 Destination is enabled, and a Notification target is set, you need to:

- Edit Cribl Stream's S3 Destination to configure Notifications.
- Add Notifications for different conditions.

Better Practice 10: Dedicate a Separate Worker Group for Replays

Execute Replays from a dedicated Worker Group. Streaming real-time data needs dedicated Worker Groups. You do not want CPU or memory conflicts on production systems that are streaming mission-critical, real-time machine data. Kubernetes or AWS spot instances are good options for on-demand Worker Processes that are needed only for infrequent replays.

Bonus Better Practice: Tune Amazon S3 Source Settings

For Cribl Stream's SQS-based Amazon S3 Source (as opposed to our S3 Collector), a few settings can speed up processing of new files added to an S3 bucket:

1. Under **General Settings**, apply a **Filename filter** if you want to pull only a subset of files from the S3 filter. The value is a regular expression (e.g., *suricata*)

2. Under **Advanced Settings**, set **Max messages** as high as 10 to fetch more SQS messages from the queue at once.
3. Under **Advanced Settings**, set the **Num receivers** to vertically scale the number of simultaneous processes pulling SQS messages and downloading S3 files. For S3 buckets with centralized Cloudtrail logs, or VPC flows across multiple accounts, raise the setting to 5 or 10.

Improve Partitioning Expressions

Creating partitioning expressions is an art. To get a sense of the possibilities and risks, let's consider both positive and negative examples.

Partitioning Expression 1: The Good

```
${C.Time.strftime(_time ? _time : Date.now()/ 1000, '%Y/%m/%d/%H/%M')}
```

Copy

Key Questions	Answers
Potential cardinality every *minute*?	1
How many open files if there are 30 Worker Processes?	30 open files
How many files in the S3 prefix, if the timeout is the default 60 seconds (assuming no file size limit)?	30 * 1 = 30 files
What if there are 10 Nodes in the Worker Group? How many files in each S3 prefix?	30 * 10 = 300 files
What happens when a replay fetches data for this hour?	Under API limit of 5,500 reads/sec

Result

- A good partitioning expression.

Partitioning Expression 2: The Ambiguous

```
${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d/%H')}
/${index ? index : 'no_index'}
/${host ? host : 'no_host'}
/${sourcetype ? sourcetype : 'no_sourcetype'}
```

Copy

Key Questions	Answers
Potential cardinality every 3 minutes?	3,000
How many active subdirectories in the staging dir?	3,000
What is the excess cardinality?	3,000 - 2,000 = 1,000
What are maximum open files if there are 14 Worker Processes on a Worker ?	14 * 2,000 = 28,000 open files
What happens when maximum open files of 2,000 is reached for any one Worker Process? What happens if it happens too frequently?	All files closed & written to S3 at once = 2,000 writes. Potential for backpressure if Cribl Stream takes long enough to write the files.
What's the maximum number of files in any S3 prefix?	14 * 10 Worker Process * (60min/3min) = 2800 files max in any S3 prefix

Additional assumptions:

- Cardinality of 3,000 every 3 mins.
- File write timeout of 3 minutes.
- File size limit never reached.
- 10 Worker Process in group.

Result

- Proceed with caution.

Solution

- Track the occurrences of `max open files reached` messages
- If the `max open files reached` message occurs too frequently, consider reducing the cardinality. You might achieve this by lowering the timeout period to 2 minutes or 1 minute.

Partitioning Expression 3: The Bad

```
${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d/%H')}
```

Copy

Key Questions	Answers
Potential cardinality every hour?	1
How many open files if there are 30 Worker Processes?	30 open files.
How many files in the S3 prefix, if the timeout is the default 60 seconds (assuming no file size limit)?	(30 * 60) = 1800 files
What if there are 10 Nodes in the Worker Group? How many files in each S3 prefix?	(1,800 * 10) = 18,000 files
What happens when a replay fetches data for this hour?	Could exceeed the API limit of 5,500 reads/sec.

Result

- A bad partitioning expression.

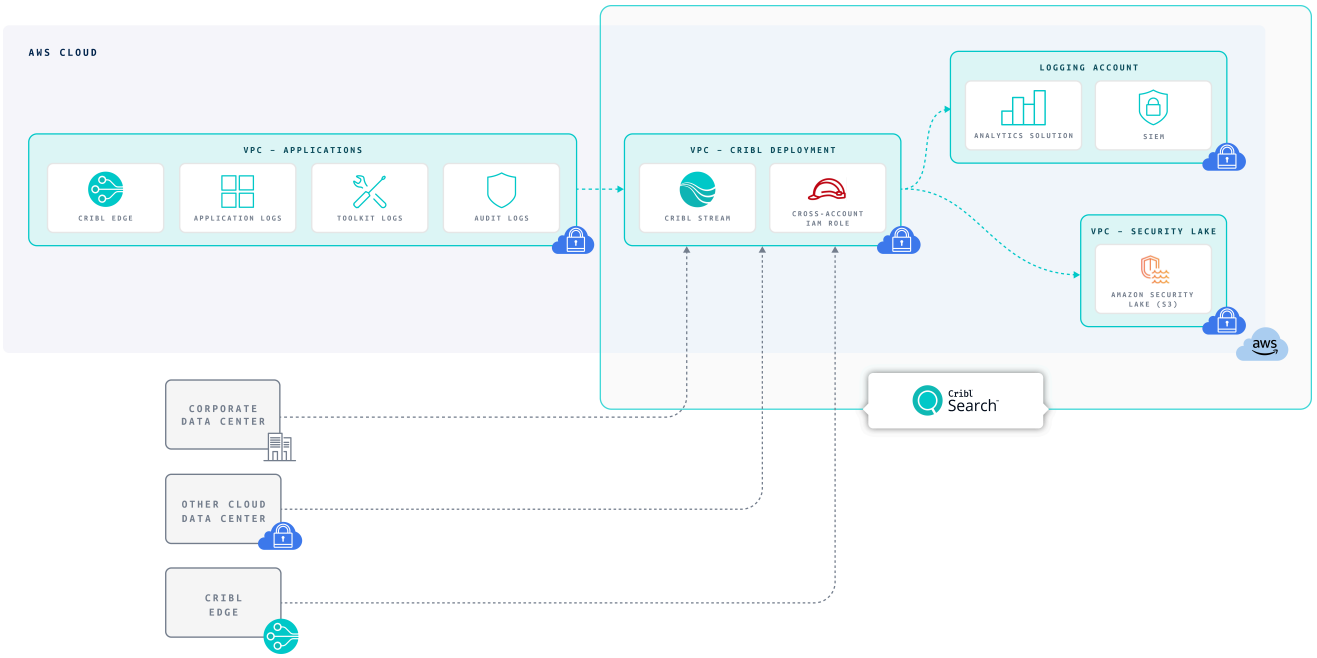
Solution

- Change the timeout to 1800 seconds to allow files to grow in size.

Partitioning Expression 4: The Terrible

8.8.1.4. Amazon Security Lake Integration

Cribl Stream supports sending data to Amazon Security Lake as Parquet files that adhere to the Open Cybersecurity Schema Framework (OCSF). The diagram below illustrates the all-AWS variant of this architecture.



Integrating Cribl Stream with Amazon Security Lake

This guide walks you through an example where Cribl Stream, running in Cribl.Cloud, sends Palo Alto Networks logs to Amazon Security Lake. The workflow starts with a Cribl Stream Syslog Source, which ingests the Palo Alto data and routes it to a Cribl Stream Amazon Security Lake Destination, which then writes Parquet files to Amazon Security Lake. These Parquet files organize the data according to the OCSF schema for the appropriate OCSF Event Class, in this case Network Activity [4001].

You can work with other data sources besides Palo Alto Networks, and other OCSF Event Classes besides Network Activity [4001]. See Going Beyond This Example below.

If you want to use Amazon Security Lake with a Cribl Stream instance deployed on-prem or in your private cloud, the setup procedures will differ from those documented here. Please contact Cribl via the `#packs` channel of Cribl Community Slack.

Set Up Amazon Security Lake

Complete the Amazon Security Lake Getting Started instructions, paying particular attention to the following actions:

1. Enable Amazon Security Lake for your AWS account.
2. Define the **collection objective** and **target objective**.

3. Select the Regions where you want to create S3 buckets to store the data in OCSF format.

Note your Amazon Security Lake S3 bucket name for use later in the setup process.

Set Up Cribl Stream

Complete the procedures in this section before doing anything else in Cribl Stream.

First, find and note the Amazon Resource Name (ARN) that enables AWS to locate your Cribl Stream instance.

1. Log in to Cribl.Cloud.
2. In the sidebar, select **Workspace**, and then select **Trust**.
3. Select the copy icon next to the **Worker ARN**, and paste the ARN into a local text editor.
4. Copy the 12-digit number in the middle of the ARN. This is the AWS Account ID you'll need later in the setup process.

For example:

- If your ARN is `arn:aws:iam::222233334444:role/main-default`, then …
- Your AWS Account ID will be `222233334444`.

Next, install the Cribl Pack that Cribl Stream will use to send data to Amazon Security Lake in the required OCSF format. (As the reference shows, the Palo Alto Networks data chosen for this example requires the **OCSF Post Processor Pack**.)

1. On the top bar, select **Products**, and then select **Cribl Stream**.
2. Select the name of the Worker Group whose Worker Nodes you want to send data to Amazon Security Lake. (For this example, we'll use the `default` Worker Group.)
3. Navigate to **Processing** > **Packs** (Stream) or **More** > **Packs** (Edge) and select **Add Pack**. From the drop-down, select **Import from Git**.
4. For the **URL**, enter:

```
https://github.com/asc-me-cribl/cribl_ocsf_postprocessing
```

Copy

5. For the **New Pack ID**, enter:

```
Cribl-OCSF_Post_Processor
```

Copy

6. Select **Import** to close the modal. Cribl Stream will take a little time to finish importing the Pack.

At the bottom of the list of Packs, you should now see your newly imported Pack, listed with the display name of **OCSF Post Processor**.

Finally, you will need the Parquet schema that supports the OCSF Event Class you're working with. In this example, that's OCSF Event Class 4001, so as shown in the reference below, you'll need the `OCSF 4001 Network Activity` Parquet schema.

- First, download the Parquet schema from this GitHub repo.
- Then, upload the schema to the Cribl Stream Parquet schema library as described in Parquet Schemas.

Once this is done, the Parquet schema you need should be available when you configure the **Parquet Settings** for your Cribl Stream Amazon Security Lake Destination.

Please post any questions you have about these procedures to the `#packs` channel of Cribl Community Slack.

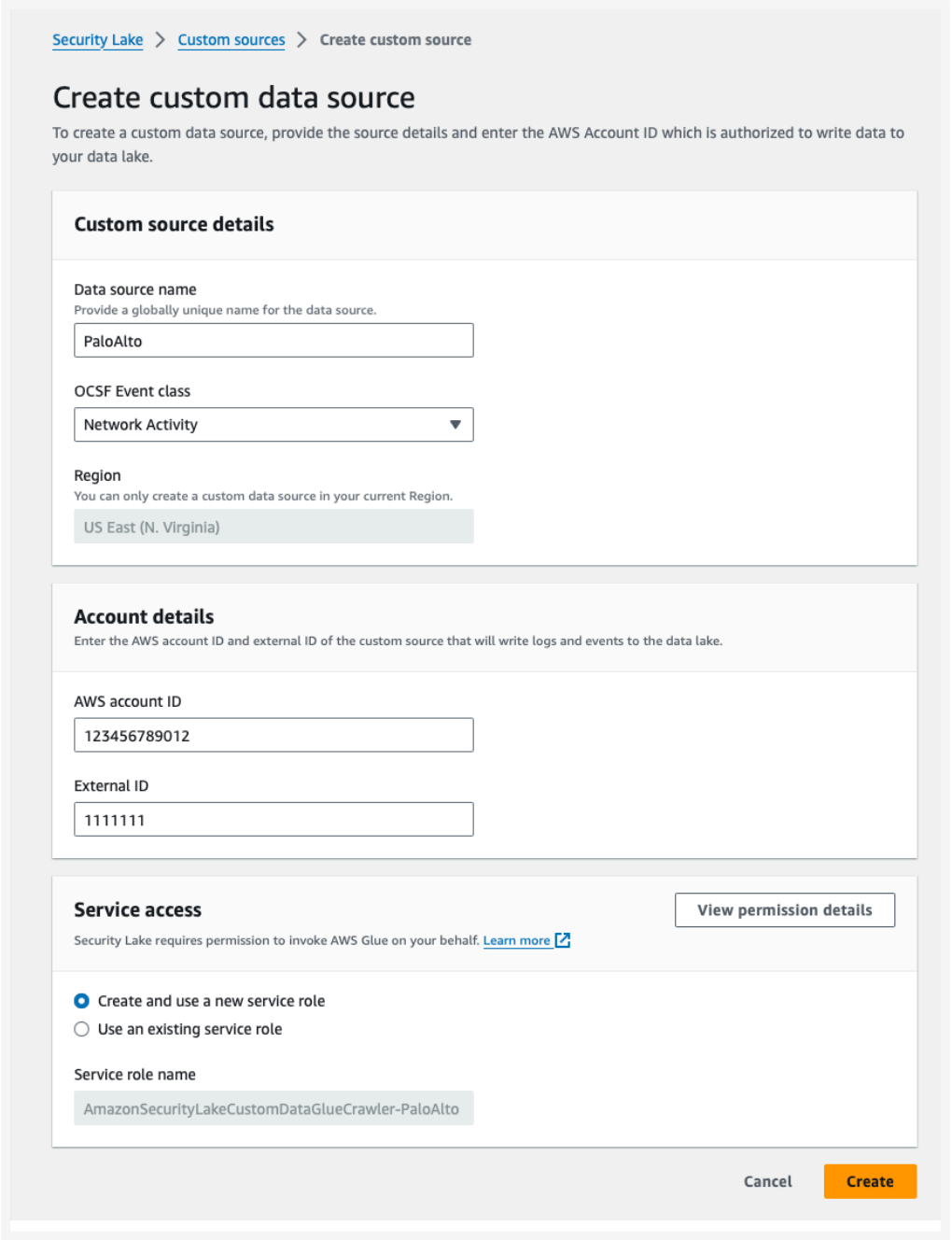
Create an Amazon Security Lake Custom Source

Because Cribl is not a native Amazon service, you'll configure what Amazon calls a **Security Lake custom source**, and the Cribl Stream Amazon Security Lake Destination will write to that.

Before you begin:

1. Have your Cribl.Cloud AWS account ID, which you identified earlier, handy.
2. Ensure that you have permissions in AWS Lake Formation to create AWS Glue databases and tables. For more information, see Granting and revoking permissions using the named resource method in the AWS docs.

Then complete the instructions here in the AWS docs. These docs will take you through the **Create custom data source** template show below.



Creating an Amazon Security Lake custom source

Once the custom source has been created, still in AWS, navigate to **Security Lake** > **Custom sources** and you should see your new custom source in the list.

Now locate the **Provider role ARN**. This is the ARN that Cribl Stream will use to declare its role to AWS.

1. Select the copy icon next to the **Provider role ARN** to add it to your clipboard. Keep the ARN handy for use later in the setup process.
2. Navigate to the **Identity and Access Management (IAM)** screen.
3. From the left menu, select **Access management** > **Roles**.
4. In the **Roles** search box, paste the part of the ARN that begins `AmazonSecurityLake` and ends with the Region.

5. When the role name appears below, select the link.
6. In the **Trust relationships** tab, view the `ExternalId` element in the JSON object that appears there.

This is the External ID for the trust relationship, which you created when went through the template. You will need this ID later in the setup process.

Create a Cribl Amazon Security Lake Destination

Back in Cribl Stream, create a new Amazon Security Lake Destination in the Routing UI as described in the Destination topic. In the **New Destination** modal, configure the settings specified below. For all settings **not** mentioned in the following notes, you can keep the defaults.

General Settings

S3 Bucket name: The S3 bucket name you noted when setting Up Amazon Security Lake.

Region: Select the Region where the S3 bucket is located.

Optional Settings

File name prefix expression: Keep the default (`CriblOut`) if it satisfies your requirements; otherwise, edit to suit your needs.

Authentication

Authentication method must be set to `Auto` (the default).

Assume Role

Cribl strongly recommends using the **Auto** authentication method in conjunction with the **Assume Role** settings as described below. Both Cribl and Amazon discourage the use of other approaches.

When using **Assume Role** to access resources in a different Region than Cribl Stream, you can target the AWS Security Token Service (STS) endpoint specific to that Region by using the `CRIBL_AWS_STS_REGION` environment variable on your Worker Node. To specify a custom STS endpoint, set the `CRIBL_AWS_STS_ENDPOINT` environment variable. If both are set, `CRIBL_AWS_STS_ENDPOINT` takes precedence. Setting an invalid Region or endpoint results in a fallback to the global STS endpoint.

AssumeRole ARN: This is the **Provider role ARN** you copied after creating your custom source in Amazon Security Lake.

External ID: Enter the ID that you specified when creating your Amazon Security Lake custom source.

Processing Settings

Post-Processing

Pipeline: From the drop-down, select the Pack you installed earlier:

```
PACK Cribl-OCSF_Post_Processor (OCSF Post Processor)
```

Copy

System fields: Remove any fields specified here.

Parquet Settings

Parquet schema: From the drop-down select `OCSF 4001 Network Activity`, since OCSF Event Class 4001 describes the events coming from Palo Alto Networks in this example.

At this point, you can select **Save**; the defaults for **Advanced Settings** should work fine for this example. Then, **Commit** and **Deploy**.

You must create one Cribl Stream Amazon Security Lake Destination for **each unique pairing** of a data source with an OCSF Event Class.

- In this example, we're ingesting Palo Alto Networks Threat and Traffic data, which requires the OCSF Network Activity [4001] Event Class. That's one pairing.
- If you also wanted to ingest CrowdStrike Network Activity data, that would be a new data source paired with the same OCSF Event Class - such as, **that would be a second unique pairing**. You would need to create a **separate** Cribl Stream Amazon Security Lake Destination for that pairing.

The reference below shows all the possible pairings of data sources and OCSF Event Classes.

Connect a Cribl Source to the S3 Destination

We'll now configure a Cribl Stream Syslog Source to ingest Palo Alto Networks system logs, using the QuickConnect UI. (The reference below shows what Cribl Stream Source to use for each supported data source.)

Navigate to QuickConnect as described here. After selecting the Syslog Source tile, select **Select Existing**, then select the preconfigured `in_syslog` Source. When prompted to `Switch in_syslog to send to QuickConnect instead of Routes?`, select `Yes`.

Select **+** and drag the connection line to the S3 Destination you created above.

The connection between the Syslog Source and the S3 Destination should now be enabled.

Commit and **Deploy** the changes.

Send Data to Amazon Security Lake

Return to your AWS Console. You should see Parquet files landing in the S3 bucket. These files should contain the Palo Alto Networks syslog data you sent through Cribl Stream.

Troubleshooting and Testing

Good things to double-check:

- Are you sending the events to your Amazon Security Lake in Parquet format?
- Are your permissions set properly for your IAM role to write to the S3 bucket?

If the answer to either question is "No," your data will not make it into Amazon Security Lake.

To send sample events from the GitHub repo, using netcat:

1. Download the `oYLbFU.json` sample file.
2. Run the following command, replacing `<cribl_org_id>` with your Cribl.Cloud Organization ID:

```
cat oYLbFU.json | nc default.main.<cribl_org_id>.cribl.cloud 10070
```

Copy

With a Cribl.Cloud Enterprise plan, generalize the above URL's `default.main` substring to `<groupName>.main` when sending to other Worker Groups.

Beyond This Example

If you want to send data from sources other than Palo Alto Networks, you can adapt the above instructions to use the appropriate Source and Pack (and/or modifications to the OCSF Post Processor Pack) in Cribl Stream. This holds true as long as the OCSF Event Classes you want to work with are among those supported by Cribl Stream. For each unique pairing of a data source with an OCSF Event Class, you'll need to create a separate Amazon Security Lake Destination.

In the next section is a list of the supported data sources, what OCSF Event Classes they handle, and what Packs and Parquet schemas you need to work with them.

Reference: Supported Data Sources

Data Source	OCSF Event Classes Handled	Cribl Source	Cribl Pack Display Name	Parquet Schema(s)

Azure Audit Logs	3002, 3004	REST Collector against Microsoft Graph API	Azure Audit Logs to OCSF	`OCSF 3002 Authentication`
				
				`OCSF 3004 Entity Management`

Azure NSG Flow Logs	4001	Azure Blob Storage Source, run as scheduled jobs, not as a Pull Source	Azure NSG Flow Logs	`OCSF 4001 Network Activity`
Cisco ASA	4001	Syslog	Cisco ASA	`OCSF 4001 Network Activity`
Cisco FTD	4001	Syslog	Cisco FTD	`OCSF 4001 Network Activity`
CrowdStrike Account Change	3001	CrowdStrike FDR	Crowdstrike FDR Pack	`OCSF 3001 Account Change`
CrowdStrike Authentication	3002	CrowdStrike FDR	Crowdstrike FDR Pack	`OCSF 3002 Authentication`
CrowdStrike Network Activity	4001	CrowdStrike FDR	Crowdstrike FDR Pack	`OCSF 4001 Network Activity`
GCP Audit Logs Account Activity	3001	Google Cloud Pub/Sub	Google Cloud Audit Logs	`OCSF 3001 Account Change`
Palo Alto Networks (PAN) Threat and Traffic	4001	Syslog	OCSF Post Processor Pack	`OCSF 4001 Network Activity`

SentinelOne	1001, 2001, 3002, 4001	Amazon S3	SentinelOne Cloud Funnel	`OCSF 1001 File System Activity`
				
				`OCSF 2001 Security Finding`
				
				`OCSF 3002`

8.8.1.5. Collecting Logs From Amazon ECS Containers

When using the Amazon Elastic Container Service (ECS), Cribl Stream can eliminate the additional expense of CloudWatch Logs by selecting an alternative Docker logging driver.

By default, the `awslogs` driver will forward all container logs to CloudWatch Logs. This requires an additional Amazon service, such as Amazon Data Firehose or AWS Lambda, to export logs to the desired destination.

But by using Docker's `splunk` logging driver, you can forward the ECS container logs directly to a Cribl Stream Worker Group, bypassing CloudWatch Logs.

For example, when using Cribl.Cloud, you can forward logs to your Organization's Splunk HEC endpoint. This endpoint information can be found in your Cloud Organization's Data Onboarding configuration.

Example Task Definition

Below is a sample JSON task definition with the required `logConfiguration` section configured to forward Splunk HEC payloads to Cribl Stream:

```
{
    "containerDefinitions": [
        {
            "name": "my-example-container",
            "essential": true,
            "image": "public.ecr.aws/my/containerimage:latest",
            ...
            "logConfiguration": {
                "logDriver": "splunk",
                "options": {
                    "splunk-url": "https://main-default-<org>.cribl.cloud:8088",
                    "splunk-token": "176FCEBF-4CF5-4EDF-91BC-703796522D20",
                    ...
                }
            }
        }
    ],
    ...
}
```

Copy

Additional Details

For additional configuration options on the `splunk` logging driver, see Docker's documentation.

For additional details on the `LogConfiguration` section, see the Amazon ECS documentation.

8.8.2. Azure

The following topics provide detailed guides and better practice for integrating with Azure services.

Source	Description
Preparing the Azure Workspace	Preparing the Azure Workspace for Cribl Integrations
Azure Event Hubs Integrations	Ingest data from an Azure Event Hub
Microsoft Sentinel Integration	Set up Cribl to send data to the Microsoft Sentinel SIEM

8.8.2.1. Prepare The Azure Workspace For Cribl Integrations

This guide walks you through configuring Azure to receive data from Cribl Stream. Currently, Cribl Stream supports sending data to **custom tables** and some **native tables** in Azure Log Analytics workspaces. For more information, see Azure's Supported Tables topic.

Before You Begin

To send data to custom tables, you need to configure a Data Collection Rule (DCR) with Stream definitions. We provide example DCR templates in our Cribl-Microsoft GitHub repository to simplify this process, and recommend using the Syslog template as the starting point. Note that the Syslog template is provided under the `DataCollectionRules` path and is intended for the **DCR kind:direct** approach.

Choose Your Data Collection Method

Decide which method best suits your deployment:

- **Method 1: DCR kind:direct (Recommended)**: This is the simplest approach. The `logIngestion` URL is embedded directly in the DCR, so a separate Data Collection Endpoint (DCE) is not needed. This is ideal if you don't require Private Link connectivity or complex network routing.
- **Method 2: Data Collection Endpoint (DCE)**: Use this method if you need **Private Link connectivity** for network isolation, specific data routing, or a single ingestion endpoint for multiple DCRs.

For most deployments, DCR `kind:direct` is the recommended approach due to its simplicity. Only use a DCE if you have specific networking requirements such as Private Link connectivity, network isolation, or centralized endpoint management across multiple regions.

Deployment Approaches

This guide covers **manual configuration** using the Azure portal. For automation and templates, refer to the Cribl-Microsoft GitHub repository. The GitHub repository contains various resources and approaches that can simplify and automate the deployment process, which is particularly useful for production environments or repeated deployments.

The following sections walk through manual Azure portal configuration for those who prefer or need to understand the step-by-step process.

Prepare the Azure Workspace

First, you must prepare the Azure Workspace to receive data from Cribl Stream by registering Cribl Stream as an authorized application with the Microsoft Identity Platform.

Use Copy buttons, where available, to copy complete values for later use without truncation.

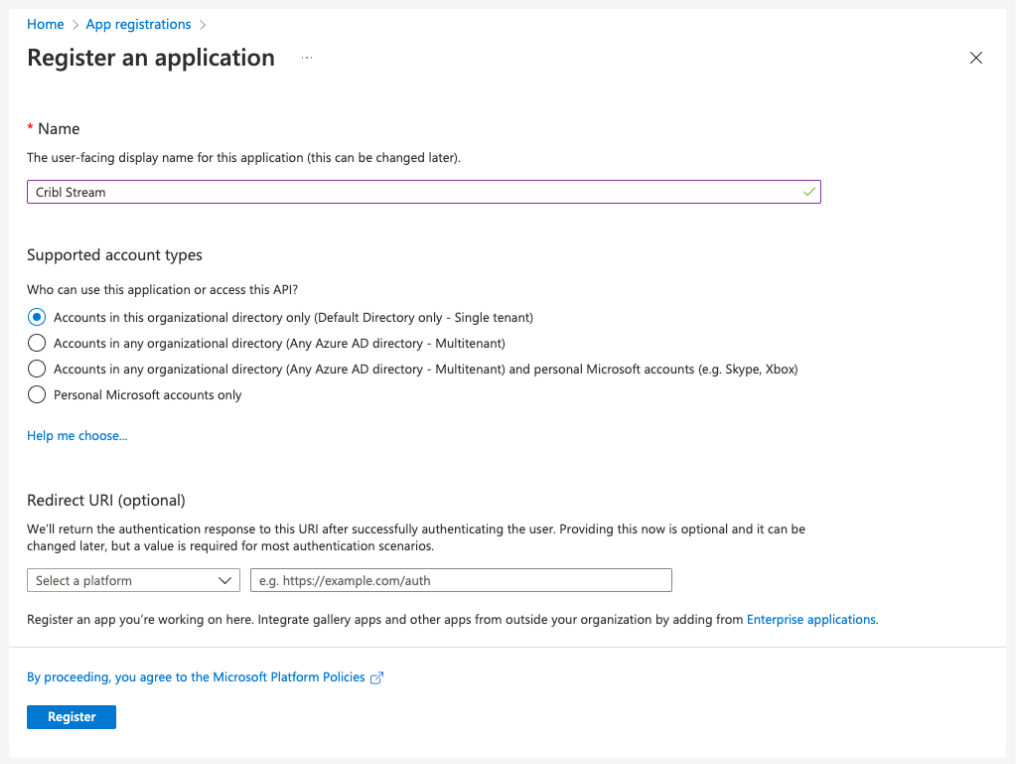
Common Setup Steps

Regardless of which configuration path you choose, you'll need to complete these initial steps.

Create Credentials for a New Azure Application

Register Cribl Stream as an authorized application with the Microsoft Identity Platform.

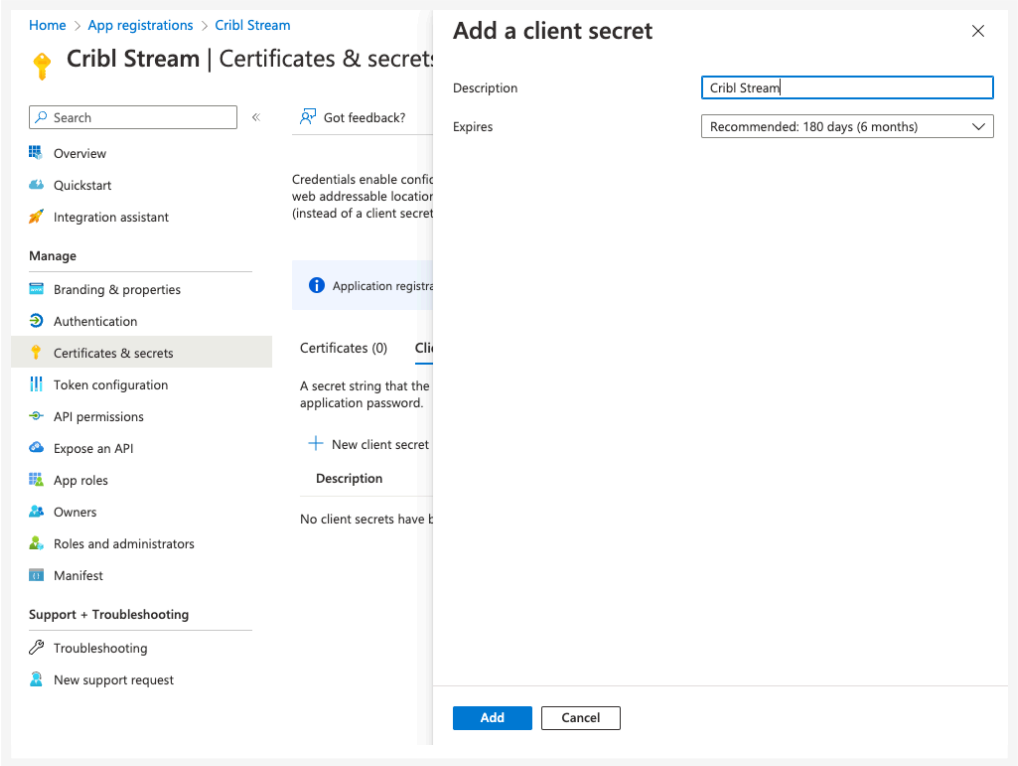
1. Log into the **Azure** portal as an `Administrator`.
2. Navigate to the portal's **App registration** section.
3. Click **New registration** to register an application.
4. Register a new application with the name `Cribl Stream`.



Registering an Application

5. Click **Register**.
6. Store the **Application (client) ID** and **Directory (tenant) ID** for later steps.

7. In your newly registered application, select the **Certificates & secrets** tab.
8. Create a **New client secret**.
9. Store the secret **Value** for later steps.



Creating a client secret

Find the Log Analytics Workspace Resource ID

1. Navigate to the Azure portal's **Log Analytics workspaces** service.
2. Select the workspace that will receive data.
3. From the **Overview** page, select **JSON view**, and store the value of the **Resource ID** (gray box at the top) for later use.

Configure Data Collection (Choose Your Path)

Path A: Using DCR `kind:direct` (Recommended)

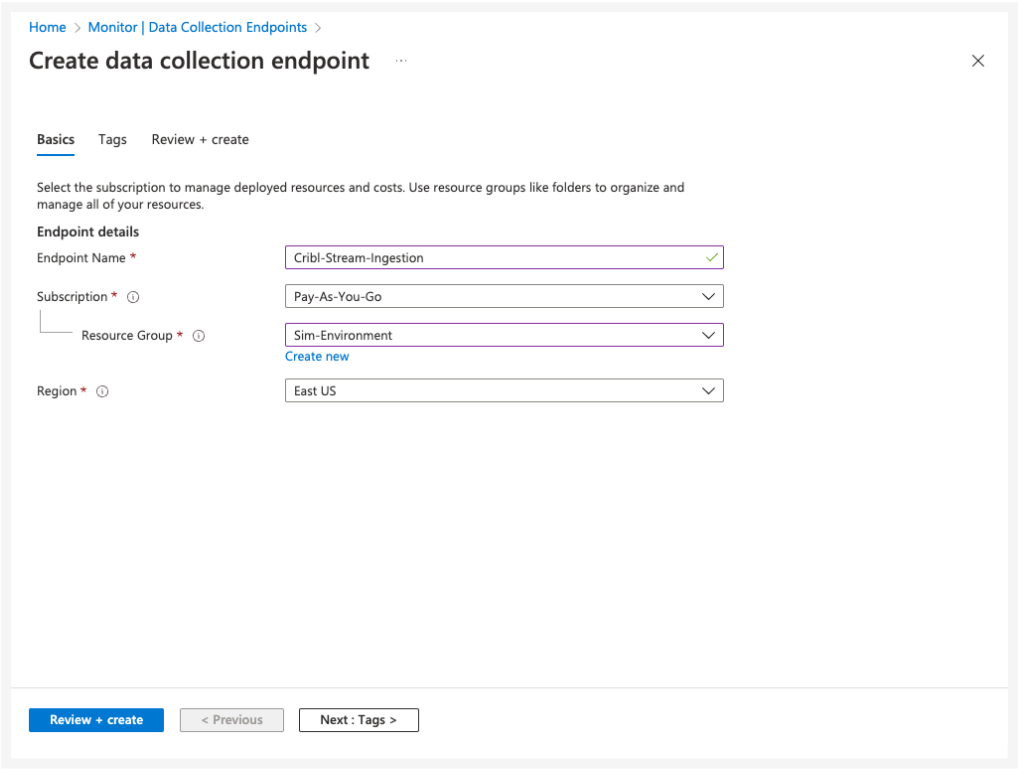
If you chose the DCR `kind:direct` method, you can skip directly to Create and Configure the Data Creation Rule. The `logIngestion` URL will be embedded automatically.

Path B: Using a DCE

If you need a DCE, create it first.

1. Navigate to the Azure portal's Monitor service.

2. Under **Settings**, select **Data Collection Endpoints**.
3. Select **Create** and name the endpoint `Cribl-Stream-Ingestion`.
4. Select the appropriate Subscription and Resource Group used by your organization.
5. Select **Review + create**, review your changes, then select **Create**.
6. In the list of **Data Collection Endpoints**, open the newly created endpoint. (You might need to refresh the page to see it.)
7. Navigate to the **Overview** page, and copy and store the **Logs Ingestion** URL.
8. Select **JSON view**, and store the value of the **Resource ID** (gray box at the top) for later use.



Creating a DCE

After creating the DCE, proceed to **Create and Configure the Data Collection Rule**.

Create and Configure the Data Collection Rule

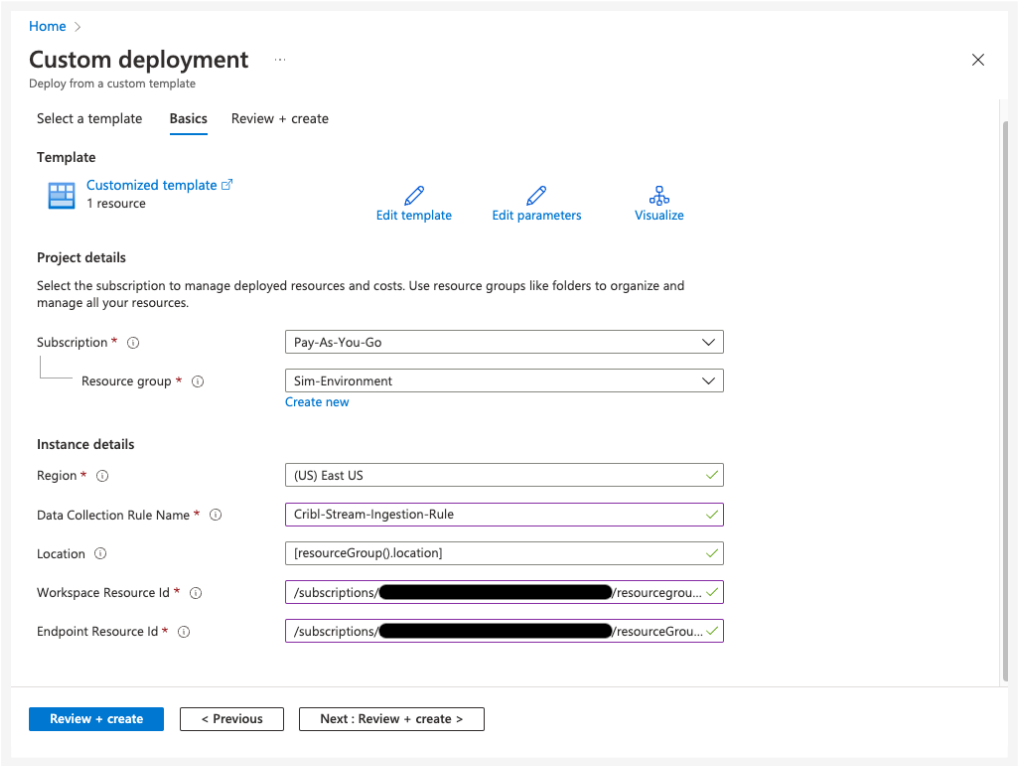
Both configuration paths require a DCR, but they use different templates:

- **DCR `kind:direct`**: Creates a DCR with `kind:direct` that includes an embedded `logIngestion URL`.
- **DCE method**: Creates a standard DCR that references the DCE.

Both templates are available in the Cribl-Microsoft GitHub repository. Browse to the appropriate template for your use case.

Deploy the DCR Template

1. Navigate to **Deploy a custom template**.
2. Select **Build your own template in the editor**, then **Load file** to upload the appropriate DCR Template from the Cribl-Microsoft repository.
3. Click **Save**.
4. Select the appropriate Subscription and Resource Group used by your organization.
5. Name the rule `Cribl-Stream-Ingestion-Rule`.
6. Enter the required parameters based on your configuration path:
    - **For DCR kind:direct configuration:** Enter only the Log Analytics **Workspace Resource ID**.
    - **For DCE configuration:** Enter the Log Analytics **Workspace Resource ID**, and the **Endpoint Resource ID**.
7. To create the Data Collection Rule, select **Review + create**, followed by **Create**.

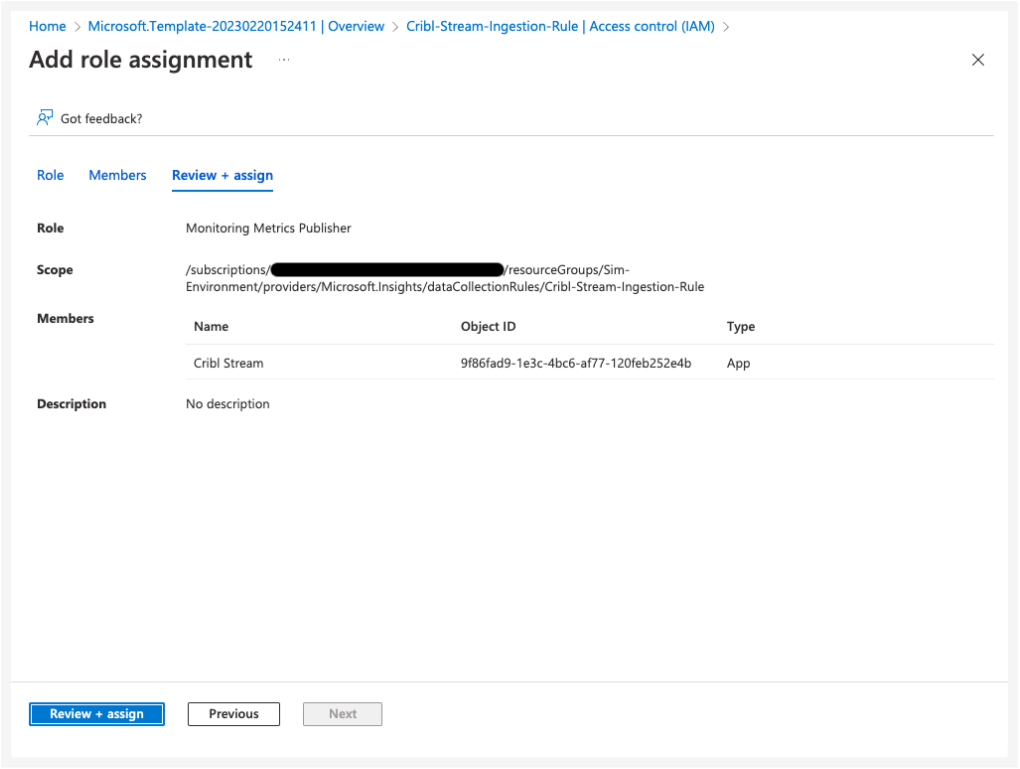


Deploy the DCR Template

Configure Permissions: Add Role Assignment

1. Once the template has been deployed, select **Go to resource**.
2. From the Overview page, select the **JSON view**, and store the **immutableId** from the JSON body (without quotes) for later use.
3. Click the Data Collection Rule's **Access control (IAM)**.
4. Click **Add role assignment**.

5. Select the **Monitoring Metrics Publisher** role, then select **Next**.
6. Click **+ Select members**.
7. Search for and select `Cribl Stream` (the app you created earlier), then select **Select** to confirm the selection.
8. Click **Review + assign** to review changes.
9. Click **Review + assign** again to implement the permissions update.



Adding a Role Assignment

Required Fields for Cribl Stream Configuration

After completing this setup, configure your Cribl Stream Azure Destination using the values in the following fields:

- `Application (client) ID`
- `Directory (tenant) ID`
- `Client secret value`
- `Log Analytics Workspace Resource ID`
- `Data Collection Rule immutableId`

Additional fields for DCE configurations:

- `Data Collection Endpoint Resource ID`

- `Logs Ingestion URL`

Next Steps

1. Configure your Cribl Stream Azure Destination using the collected credentials.
2. Test the Pipeline.
3. Monitor data flow in your Log Analytics workspace.

Additional Resources

- Cribl-Microsoft GitHub Repository - DCR templates and integration artifacts
- Azure Monitor data collection
- Data Collection Rules overview

8.8.2.2. Azure Event Hubs Integrations

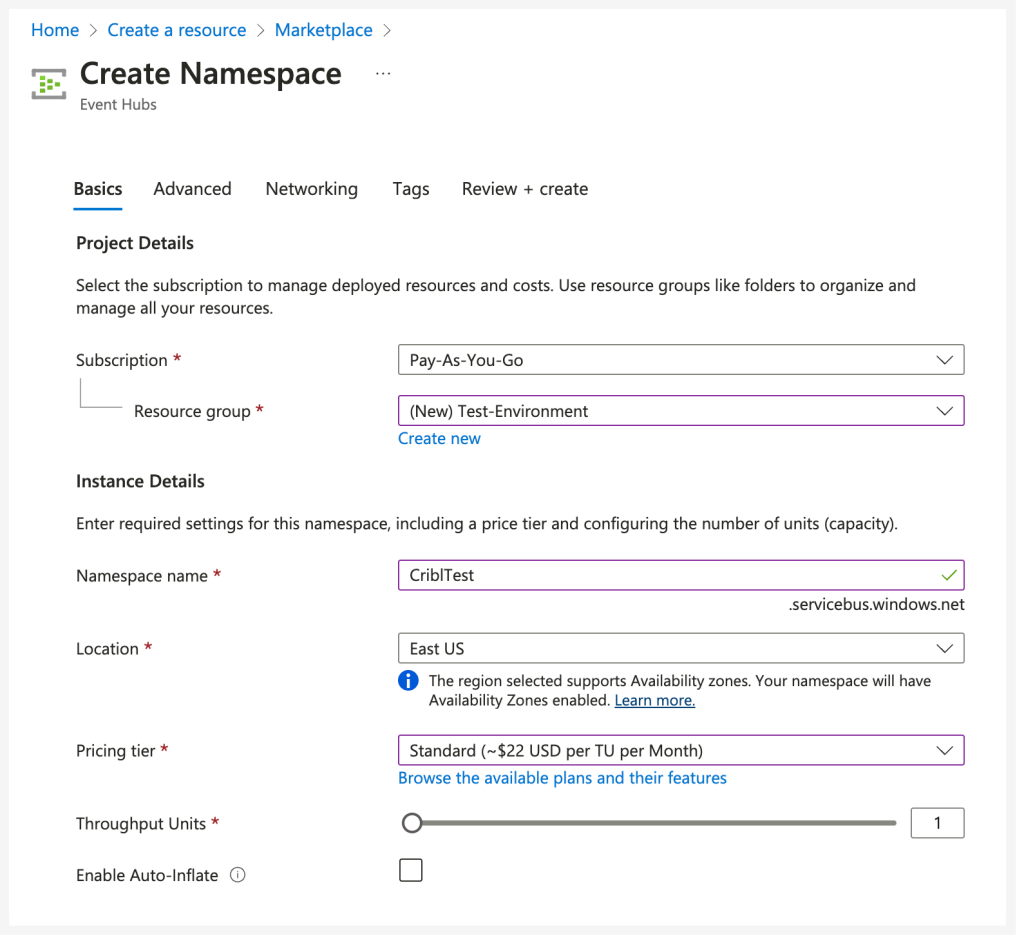
To ingest data from an Azure Event Hub, you can create an Event Hub and a Cribl Stream Azure Event Hubs Source configured to consume data from the Event Hub. This page documents that integration.

Your Azure Event Hubs account must be at the Standard (or a higher) pricing tier, because the Basic pricing tier does not support the PLAIN authentication method Cribl Stream uses for Event Hubs.

Prepare Azure Event Hubs to Send Data to Cribl Stream

First, create an Azure Event Hub as described in the Azure documentation.

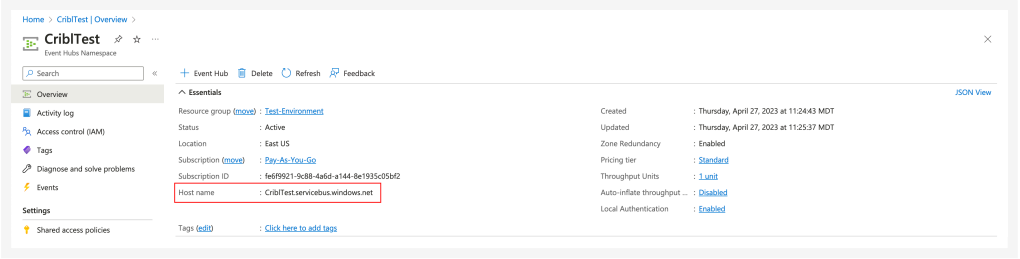
In this tutorial, we use `CriblTest` as the **Namespace name**. You can use any **Namespace name** you like, as long as it is unique across Azure.



Creating an Event Hubs Namespace

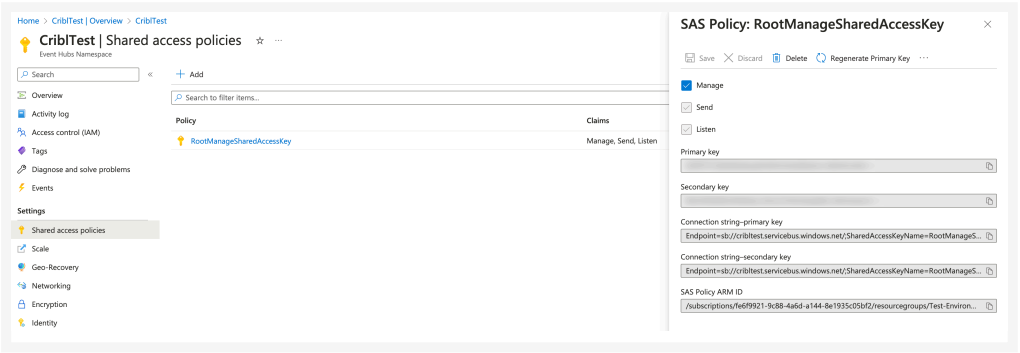
Collect the information you will need when configuring Cribl Stream:

1. In the **Deployment** page, click **Go to Resource**.
2. Copy the **Host Name** and store it.



Finding the Host name and Shared access policies

3. Click **Shared Access Policies** to open the page where you can select policies for your Event Hubs Namespace, and then click `RootManageSharedAccessKey` to show details for that policy.



Viewing Shared access policies

4. Copy and securely store the **Connection String-primary key**.
5. Click on Event Hubs on the left-hand column and copy the Event Hubs name you want to ingest.

Restricting Public Network Access for Cribl.Cloud Deployments

If you are using a Cribl.Cloud deployment, you have the option of restricting public network access to the Event Hubs environment.

To restrict access, you will need to obtain the egress IP in your Cribl.Cloud portal, then use it to configure public network access in the Event Hubs environment.

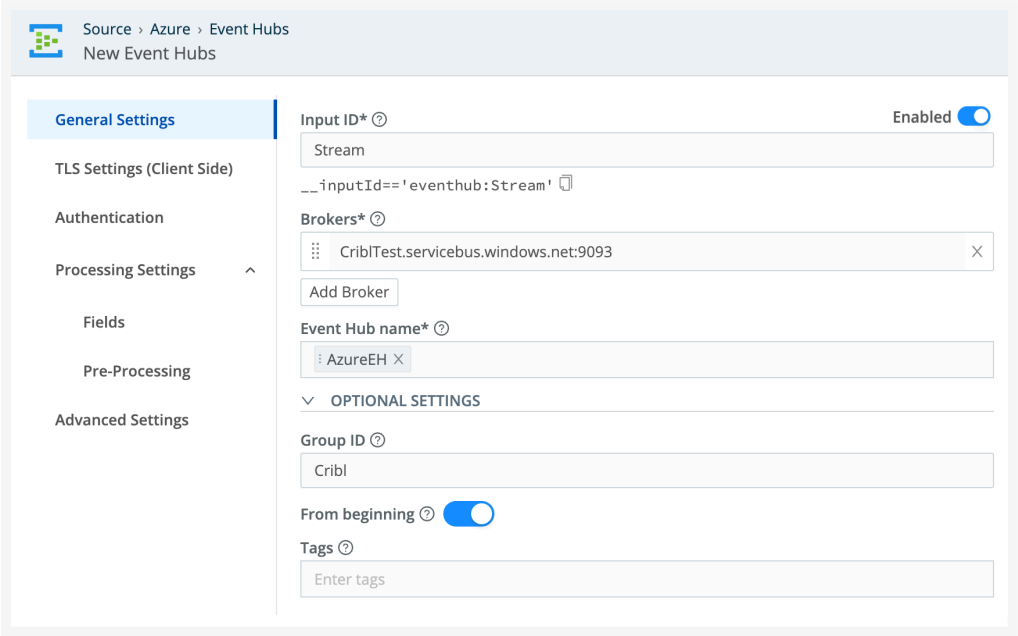
1. To obtain the egress IP, click **Access Details** and copy the **Egress IP** of your Cribl.Cloud Organization.
2. In the Azure Event Hubs environment, select **Networking** > **Public network access** > **Selected networks**.
3. On that same page, paste the Cribl.Cloud egress IP into the **Address range** field in the Event Hubs environment.

Configure the Azure Event Hubs Source in Cribl Stream

1. From a Cribl Stream instance's or Group's **Manage** submenu, select **Data** > **Sources**, then select **Azure Event Hubs** from the **Manage Sources** page's tiles or left nav. Click **New Source** to open the **Azure Event Hubs** > **New Source** modal.

2. In the **General Settings** tab, enter the following values:

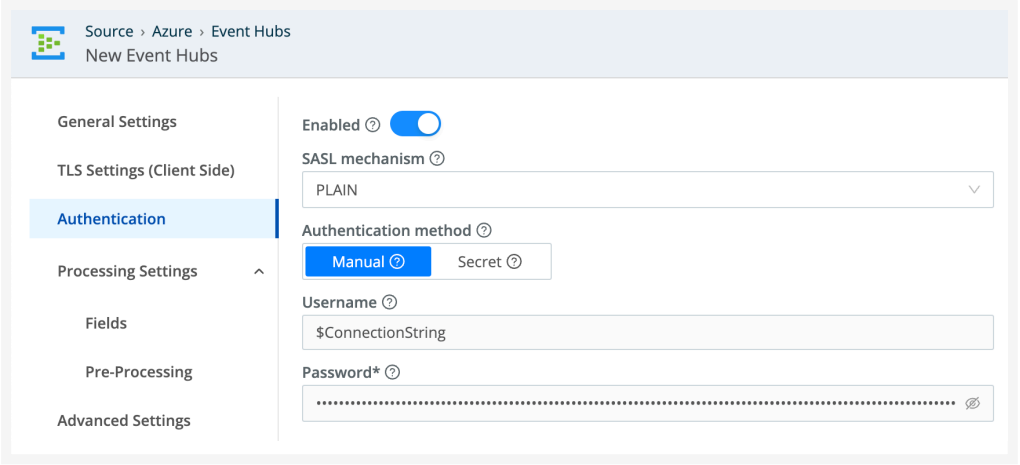
   - **Input Id**: A unique ID of your choice. For example: `Stream`.
   - **Brokers**: End the Host Name you wrote down earlier and append port `9093`. For this example: `CriblTest.servicebus.windows.net:9093`.
   - **Event Hub name**: The name of your Azure Event Hub. For this example: `AzureEH`. The **Event Hub Name** is equivalent to a Kafka topic. It should not be the same as the Azure Event Hubs namespace value you created above.
   - **Group ID**: This value can be `Cribl`, but the **Group ID** cannot be shared with other technologies. Sharing this value across technologies may cause unexpected behavior, such as failing to collect logs reliably.



The General Settings tab

3. In the **Authentication** tab, enter or select the following values:

   - **SASL mechanism**: Options include `PLAIN` and `OAUTHBEARER`.
   - **Username**: `$ConnectionString` (the default generated by Azure).
   - **Authentication Method**: Select `Manual` to use the Connection String Key generated by Azure Event Hubs.
   - **Password**: Enter the **Connection String-primary key** that you copied earlier.



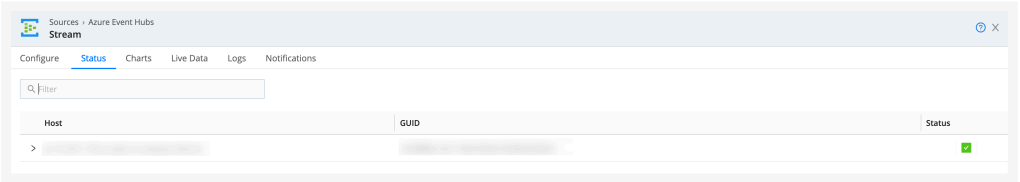
The Authentication tab

Verify that Data is Flowing from Your Azure Event Hub to Cribl Stream

Before you can verify that data is reaching Cribl Stream, you must ensure that it is flowing out of your Azure Event Hub in the first place.

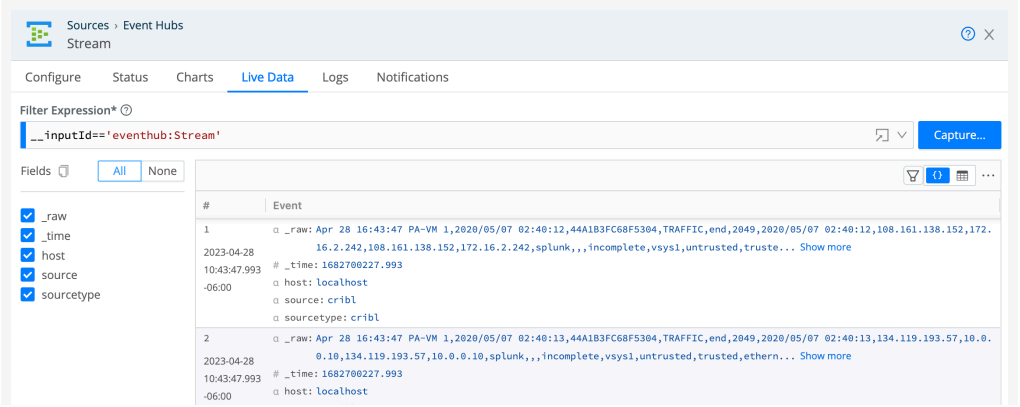
One option is to configure a Datagen and a Route to send data to the Event Hub destination. We'll assume you have done that, or gotten data flowing from your Azure Event Hub in some other way.

1. In Cribl Stream, open the **Sources** > **Azure Event Hubs** > **Cribl Stream** page. This should show your Source, with a message confirming that it is working properly.



A working Source

2. Open the **Live Data** tab. You should see the data that is flowing from your Azure Event Hub to Cribl Stream.


Sources › Event Hubs
Stream
Configure
Status
Charts
Live Data
Logs
Notifications
Filter Expression*
__inputId=='eventhub:Stream'
Capture...
Fields
All
None
_raw
_time
host
source
sourcetype
#
Event
1
2023-04-28 10:43:47.993 -06:00
_raw: Apr 28 16:43:47 PA-VM 1,2020/05/07 02:40:12,44A1B3FC68F5304,TRAFFIC,end,2049,2020/05/07 02:40:12,108.161.138.152,172.16.2.242,108.161.138.152,172.16.2.242,splunk,,,incomplete,vsys1,untrusted,truste... Show more
_time: 1682700227.993
host: localhost
source: cribl
sourcetype: cribl
2
2023-04-28 10:43:47.993 -06:00
_raw: Apr 28 16:43:47 PA-VM 1,2020/05/07 02:40:13,44A1B3FC68F5304,TRAFFIC,end,2049,2020/05/07 02:40:13,134.119.193.57,10.0.0.10,134.119.193.57,10.0.0.10,splunk,,,incomplete,vsys1,untrusted,trusted,ethern... Show more
_time: 1682700227.993
host: localhost

8.8.2.3. Microsoft Sentinel SIEM Integration

You can send events through Cribl Stream Microsoft Sentinel Destinations to the Microsoft Sentinel SIEM in Azure.

This use case demonstrates how to send data from Cribl Stream to Microsoft Sentinel in various Azure environments, including Azure Public Cloud, Azure US Government Cloud, and Microsoft Azure operated by 21Vianet. While we'll cover key points about the Cribl Stream Microsoft Sentinel Destination here, you should refer to the Sentinel Destination topic for complete UI details.

Before You Begin

Ensure that you have prepared your Azure Workspace. You must complete this step before you can send events through the Microsoft Sentinel Destination to the Microsoft Sentinel SIEM.

If you do not yet have Microsoft Sentinel up and running, please join us on Cribl's Community Slack at https://cribl-community.slack.com/ and let us know; very likely, a wise goat will offer time-saving suggestions.

Event Types

Of the many event types (also called "tables") present in the Microsoft ecosystem, this topic provides a starting point for integrating Cribl with Sentinel. For the complete list of supported tables and their schemas, refer to the Microsoft documentation for supported tables.

To plan your Cribl/Sentinel integration, start by deciding which event types (a.k.a. tables) you want to send to Sentinel. Common event types you might work with include:

- `CommonSecurityLog`
- `SecurityEvent`
- `Syslog`
- `WindowsEvent`

Choosing a Data Collection Strategy

When sending data to Sentinel, you have a decision to make that balances ease of troubleshooting with the number of Cribl resources you'll need:

- **One Cribl Destination for a single Data Collection Rule (DCR)**: This approach minimizes the number of Destinations you need in Cribl. You can send all data to a single Sentinel Destination, which is configured to use a single DCR that handles all of your event types. While this reduces Cribl resource overhead, it can make troubleshooting more complex as all data flows through a single point.

- **A unique Cribl Destination for each DCR**: This approach simplifies troubleshooting and provides greater control over your data. By creating a unique Sentinel Destination for each DCR, you can easily isolate and debug issues related to a specific data type. This is the recommended approach for most deployments, as it makes managing your data flows easier.

Creating Data Collection Rules

In Cribl Stream 4.1.4 and later, DCRs (data collection rules) support direct log ingestion URLs. This update means that DCEs (data collection endpoints) are no longer needed unless your Log Analytics workspace has restricted public access, such as with private endpoints.

The Sentinel SIEM can have multiple ingestion endpoints, called Data Collection Endpoints (DCEs). In turn, each DCE can have multiple Data Collection Rules (DCRs). A DCR can dictate the structure of the data flowing into one table, or multiple tables.

For ready-to-use DCR templates and automation options covering all supported tables, visit the Cribl-Microsoft GitHub repository. Cribl provides DCR templates for various tables. You can use these templates to quickly set up your DCRs.

Ensure that the field names in your data match the schema defined in the Data Collection Rule (DCR) for Microsoft Sentinel. The DCR specifies the expected structure, including field names and types. Mismatched field names can result in dropped fields.

To determine which Sentinel table should ingest a specific data source, review the Microsoft Sentinel Content Hub catalog. For instance, the Palo Alto Sentinel Solution sends all Palo Alto Firewall data to the `CommonSecurityLog` table. In contrast, the Crowdstrike Solution uses multiple tables, including custom ones, for data ingestion.

Besides the Azure documentation, you can study the Cribl template to understand how DCRs are structured. For example, each table's columns are defined in a sub-element of the `resources` > `properties` > `streamDeclarations` element. The `dataFlows` element - at the same level as `streamDeclarations` - specifies the table that a given type of data should populate, via the `outputStream` sub-element. It also specifies the query that will translate incoming fields into values in the table's columns, via the `transformKql` sub-element.

Read more about DCRs in the Microsoft Sentinel documentation.

Custom DCR Template Errors

If you get an error such as `Table for output stream 'Microsoft-CommonSecurityLog' is not available for destination 'logAnalyticsWorkspace'`), it is probably because you don't

> have Sentinel assigned to your Log Analytics workspace. Please add Microsoft Sentinel to your Log Analytics Workspace to correct this error.

Preparing Your Azure Environment

To configure your Cribl Stream Microsoft Sentinel Destination, you must provide your log ingestion URL. This resides either in a Data Collection Endpoint, or in a Data collection Rule of `kind:Direct` and a Data Collection Rule (DCR) immutable `ID`. You configure this in **General Settings** > **Endpoint configuration**. Cribl recommends the `ID` option, as it automatically generates the correct URL format, and is more reliable than manually constructing URLs.

The ID format follows this pattern:

- **Data Collection Rule ID**: The immutable ID of your DCR.
- **Stream Name**: The specific stream within the DCR you're targeting.

In most respects, Cribl Stream's Microsoft Sentinel and Webhook Destinations are alike. This section notes the few differences, while highlighting key points about configuring Microsoft Sentinel Destinations.

In Microsoft Sentinel Destination, **General Settings** > **Authentication** offers fewer options than the Webhook Destination. This is because OAuth is Cribl Stream's only supported option for Microsoft Sentinel, for now. Some fields that present in other Destinations always have the same values for Microsoft Sentinel, and Cribl Stream fills those out for you without exposing them in the UI.

The Microsoft Sentinel Destination does not support client-side TLS.

The value for **Advanced Settings** > **Body size limit (KB)** value defaults to `1000`. The intent here is to keep request bodies from exceeding Microsoft Sentinel's limit of 1 MB, meaning 1024 KB including headers. If necessary, you can lower the **Body size limit (KB)** even further.

8.8.3. Splunk

The following topics provide detailed guides and better practice for integrating with Splunk services.

Source	Description
Splunk Cloud Platform and BYOL Integrations	Integrating with the Splunk Cloud Platform by using the Splunk HTTP Event Collector
Switch Cribl Stream Destinations from S2S to Splunk HEC	How to switch your data collection from Splunk to Splunk (S2S) to HTTP Event Collector (HEC)
Splunk to Elasticsearch	Routing data from Splunk to Elasticsearch services
Splunk to Exabeam	Exporting parsed data to Exabeam
Splunk Stream to Cribl Stream	Sending data from Splunk Stream packages to Cribl Stream
Decryption of Data in Splunk	Decrypting data with Splunk
Stream-to-Splunk Encryption	Encrypting and decrypting data moving from Cribl Stream to Splunk

8.8.3.1. Splunk Cloud Platform And BYOL Integrations

The Splunk HTTP Event Collector (HEC) is the preferred method for integrating with the Splunk Cloud Platform. It's easy to set up, offers superior compression, and efficiently load balances data across multiple indexers in a distributed Splunk environment. While Splunk-to-Splunk (S2S) can be used for specific scenarios, such as legacy integrations or granular data distribution, HEC generally provides a more straightforward and efficient integration process.

Cribl Stream provides multiple integrations for sending data to the Splunk Cloud Platform. The following table outlines the supported Cribl Stream Destinations and Splunk protocols for different Splunk Cloud Platform deployment scenarios:

Cribl Stream Destination	Splunk Protocol	Splunk Deployment
Splunk HEC Destination	Splunk HEC	- Distributed Splunk Cloud Platform - Bring Your Own License (BYOL) deployment (either in a non-Splunk cloud or on-prem)
Splunk Load Balanced Destination	S2S	- Distributed Splunk Cloud Platform - BYOL deployment
Splunk Single Instance Destination	S2S	- Single-instance Splunk Cloud Platform (trial or smaller deployments)

For BYOL deployments, leverage the `.pem` and `outputs.conf` files already in use on your Splunk Universal Forwarders to maintain consistency and simplify the security setup. The Splunk documentation has detailed instructions on securing your Splunk indexers to ensure the overall security of your deployment.

8.8.3.2. Switch Cribl Stream Destinations From S2S To Splunk HEC

This guide is for users that want to change their Splunk data flow from using Splunk to Splunk (S2S) to HTTP Event Collector (HEC). The guide will provide step-by-step instructions for making this change.

Prerequisites

You must have the following prerequisites before following this guide:

- A Cribl Stream instance
- A Splunk instance

Configure the queue Setting

If your Splunk Technology Add-ons (TAs) include many index-time operations like SEDCMD or TRANSFORMS, be aware of potential data quality impacts. By default, Splunk HEC sends data through a queue where these operations (often referred to as extractions) may be applied, potentially altering your original data.

To bypass these index-time operations (extractions) and ensure data integrity, configure the `queue` setting in the `inputs.conf` file in Splunk to use the `rulesetQueue` on a per-token stanza level. This approach allows for newer Ingest Actions to be applied through the Ruleset queue.

Modifying the `queue` setting in the **global** stanza of `inputs.conf` will impact all HEC tokens and may have unintended consequences for other HEC-based integrations.

1. Locate and open the `inputs.conf` file on your Splunk instance. This file is typically located in the `$SPLUNK_HOME/etc/system/local` directory.
2. Add or modify the following within the file:

```
[http://cribl]
disabled = 0
index = main
token = ${SPLUNK_HEC_TOKEN}
queue = rulesetQueue
```

Copy

3. Replace `${SPLUNK_HEC_TOKEN}` with your actual HEC token.

4. Save the `inputs.conf` file and restart the Splunk instance for the changes to take effect.

If you are using Splunk Cloud, you must open a ticket with Splunk to request a change to the `inputs.conf` file. Make sure you request that they change the `queue` setting only for the Cribl integration, not on a global level.

Create the Splunk HTTP Event Collector Token

Once you complete this section, you will have a new HEC token in Splunk that can receive data from Cribl Stream.

1. Log in to Splunk.
2. Navigate to **Settings** then **Add Data**.
3. Select **Monitor** from the **Add Data** menu.

Select a Source

1. Choose **HTTP Event Collector**.
2. Enter a name for your token (`cribl-HEC` for example).
3. Click **Next**.

Validate Input Settings

1. For **Source type**, select **Automatic** (unless you are setting this token for a specific Sourcetype).
2. For **Index**, select your desired Allowed Indexes.

If you want to send Splunk logs such as data from `_internal`, set the Allowed Indexes to `N/A`. You may want different tokens for different types of data.

3. Click **Next**.

Review HEC Settings

1. Ensure that **Allowed indexes** it set to `N/A`, unless you want to restrict users to only send data to specific indexes using this token.
2. Click **Submit**.

You should now have a new, valid HTTP Event Collector (HEC) token that you can use with Cribl Stream. This token allows Cribl Stream to receive events for any index that is available in Splunk.

Create a Splunk HEC Destination in Cribl Stream

Using the token you just created in Splunk, create a Splunk HEC Destination in Cribl Stream.

1. Log in to Cribl Stream.
2. Click **Data** then **Destinations**.
3. From the resulting page's tiles or the **Destinations** left nav, select **Splunk**, then > **HEC**
4. Click **Add Destination** to open a **New Destination** modal and fill in the required fields:

Field	Configuration
Output ID	Enter a descriptive name for your Destination.
Load balancing	Decide whether to enable Load Balancing for this Destination.
Splunk HEC Endpoints	Enter the endpoint for your Splunk instance, in the following format: `https://http-inputs-<CLOUD_ORG>.splunkcloud.com/services/collector/event`
Authentication method	Set to **Manual**.
HEC Auth token	Enter the Splunk HEC API access token from the Create the Splunk HTTP Event Collector Token section.

5. Depending on your data volume and event size, you may also want to tune Advanced Settings (**Body size limit**, **Request concurrency**, **Flush period (sec)**). See Splunk HEC Advanced Settings for specific guidance.
6. Click **Save**.
7. In a Distributed deployment, you'll also need to **Commit & Deploy** to push the configurations to your Workers.

Test the Destination Connection

1. Open the Splunk HEC Destination you just created and click the **Test** tab.
2. Add the index field to **Test input** and click **Run test**.
3. Check the bottom of the **Test** tab. If the banner shows a green Success indicator, your endpoint and token are valid.

Switch from S2S to Splunk HEC

Now, you can switch from Splunk S2S to HEC Destination.

1. In Cribl Stream, navigate to **Data** then **Destinations**.

2. Choose either **Splunk Single Instance** or **Splunk Load Balanced** depending on which applies to you.
3. Open the Splunk S2S Destination and check for any **QuickConnects** Sources.
4. Click directly on a Source in this column to it.
5. Switch all **QuickConnects** Sources from the Splunk S2S Destination to the Splunk HEC Destination.

Check Routes and Packs

Once you've switched all QuickConnects references to Splunk HEC:

1. Navigate to **Processing**, **Packs** and **Routing**, **Data Routes** and look for references to the Splunk S2S Destinations:
    - On the **Data Routes** page.
    - On the **Routes** page for the Pack.
2. Change the Splunk S2S Destinations you find to the Splunk HEC Destination.
3. Once you're finished, **Commit & Deploy** the changes. In standalone instances, the changes take place as soon as you save them.

We recommended that you make these changes incrementally and validate each change inside of your Splunk instance for any data irregularities before you move on.

Now, you've successfully switched all Splunk S2S Destinations in Cribl Stream to the Splunk HEC Destination.

8.8.3.3. Splunk To Elasticsearch

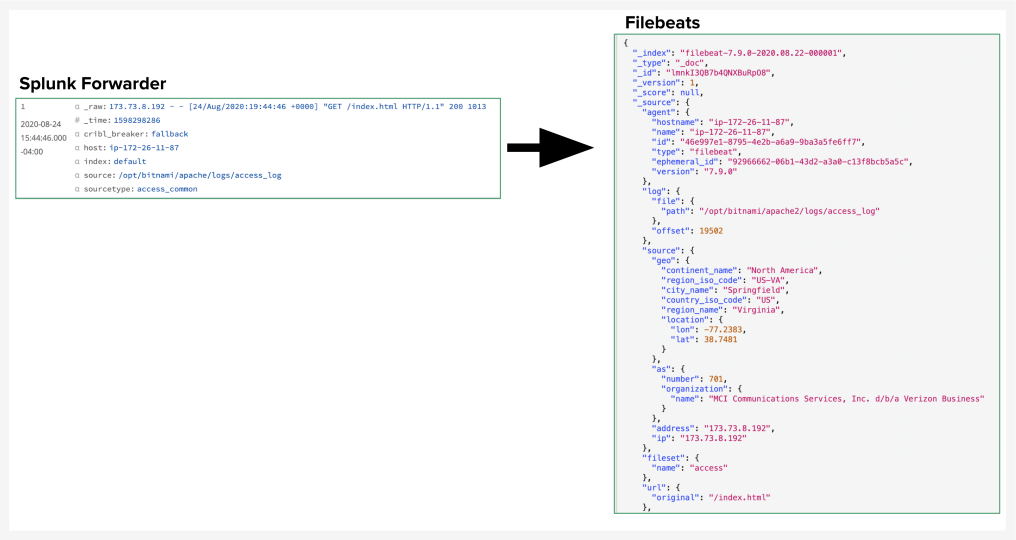
To route data from existing Splunk infrastructure to Elasticsearch services, you might face a daunting task: re-architecting your entire forwarding tier. This could require retooling lots of servers - up to hundreds, or thousands - to uninstall their Splunk forwarders, and swap in Elastic-compatible agents.

Cribl Stream can reduce this effort to just a few hours: Configure one Splunk `outputs.conf` stanza to output to Cribl Stream, and propagate that across all your Splunk servers. Done!

Next, you can easily configure Cribl Stream to listen for Splunk data on one port, and to route that data to all the Elasticsearch destinations you want to feed.

Transforming Data from Splunk to Elastic Format

Also, in Cribl Stream's core, you can easily design a Pipeline that modifies the original Splunk event into Elastic's Common Schema - making it look exactly like an event generated by an Elastic agent. These transformations help you make the most of Elastic's offerings, like Filebeats, etc.



Transforming to Elastic Common Schema

Some of the Cribl Stream Functions useful in transforming Splunk-generated events into Elastic's format are:

- Regex Extract: Extract a portion of the raw event, and place it into a specified field.
- Lookup: key off the host IP to add fields like `hostname`, `name`, `id`, and `type`.
- Eval: Turn key-value pairs into nested JSON objects.
- GeoIP: Correlate source IP to a geographic database.

We'll show all four in our example Pipeline below, although you might need only a subset.

Goat Rid of Some Guesswork

Cribl Stream will offer you further time savings as you configure the internal data transformation. Cribl Stream's Pipeline Data Preview features enable you to test transformations' results as you build your Pipeline, before you commit or run it.

This eliminates blind guesswork in Splunk configuration files to specify source -> index transformations, check the results, and then start all over again. In particular, Cribl Stream's Regex Extract Function provides a regex101-like UI, to facilitate precisely designing and debugging your regex expressions.

Let's goat started on the example.

Configure Splunk Forwarder

First, in a Splunk App, configure a Splunk forwarder (UF or HF) to specify your Cribl Workers as destinations. Use outputs.conf stanzas of this form:

.../outputs.conf

```
[tcpout]
disabled = false
defaultGroup = cribl, <optional_clone_target_group>, ...

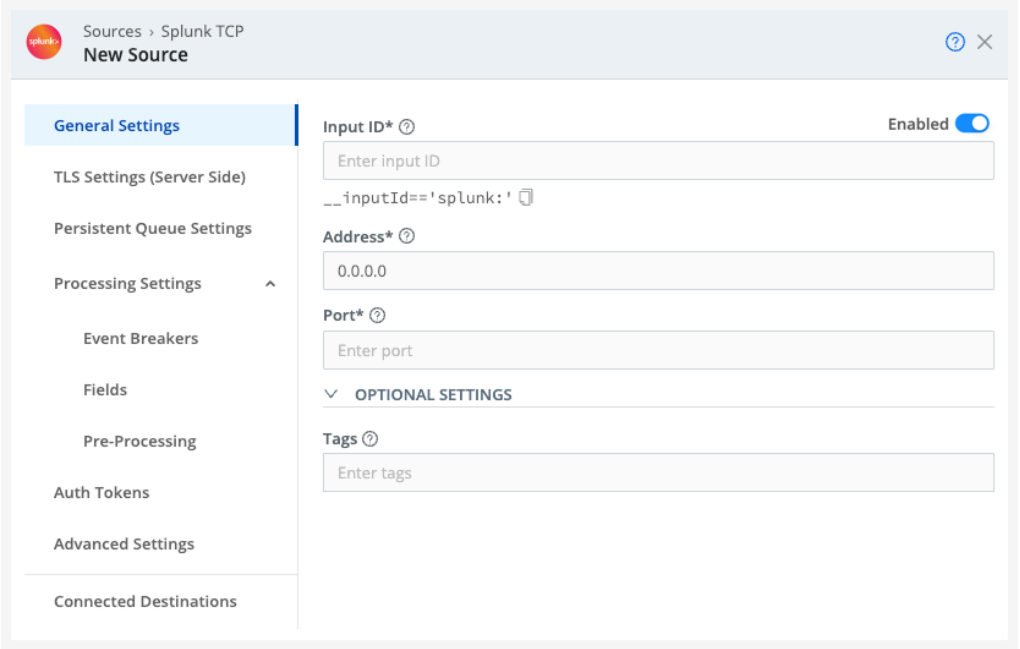
[tcpout:cribl]
server = [<cribl_ip>|<cribl_host>]:<port>, [<cribl_ip>|<cribl_host>]:<port>, ...
sendCookedData=true
```

Copy

Push the app using the deployment server.

Configure Splunk Source in Cribl Stream

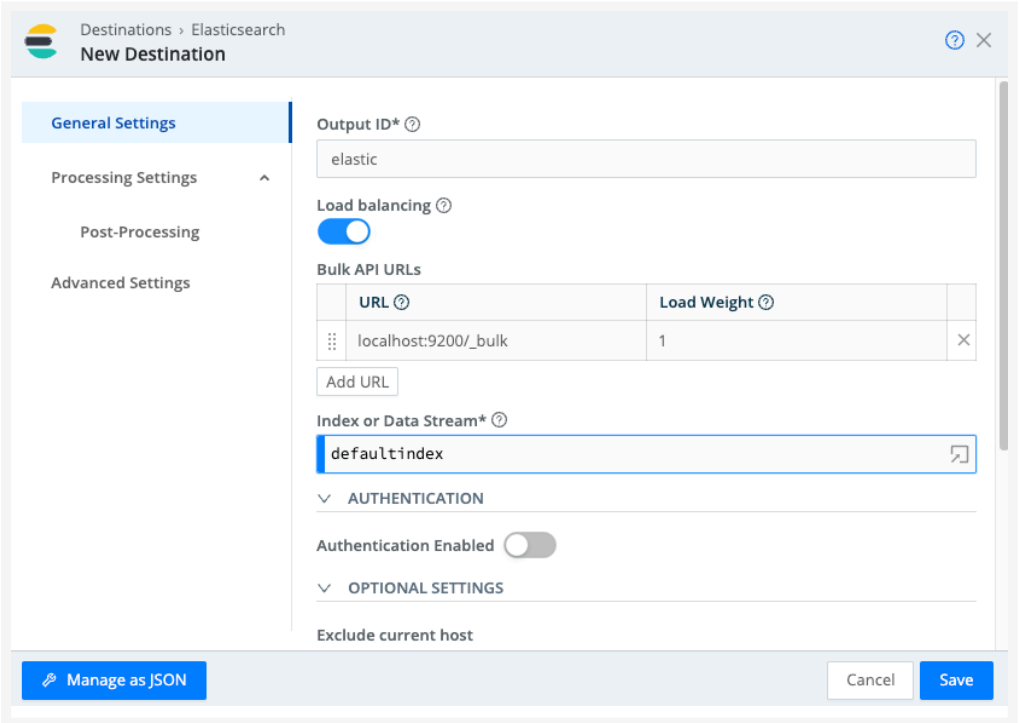
Next, in Cribl Stream, configure a Splunk Source. The key requirement here is to set the **Port** to listen on. (Optionally, you can also configure TLS, Event Breakers, metadata fields, and/or a pre-processing Pipeline.)



Splunk Source configuration

Configure Elasticsearch Destination

To configure Cribl Stream's output, set up an Elasticsearch Destination by specifying the **Bulk API URL** and **Index**.



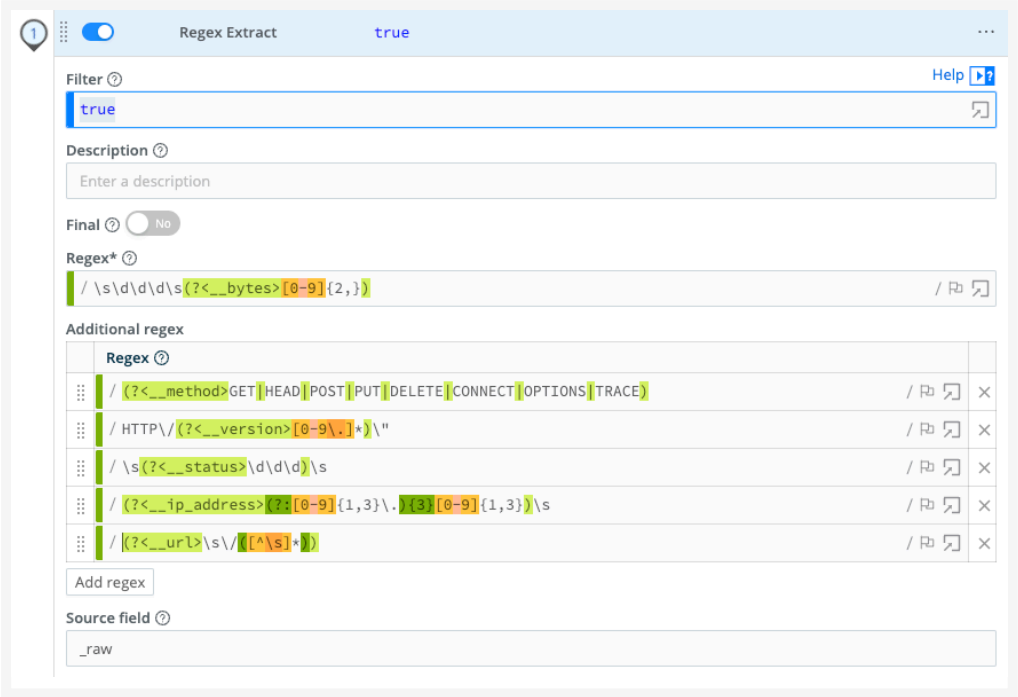
Elasticsearch Destination configuration

Configure Pipeline

Next, this section shows several Functions that you can assemble into a Pipeline to transform incoming Splunk events to match the Elastic Common Schema.

Regex Extract Function

First, use a Regex Extract Function to break the Splunk events into fields. Try the sample configuration shown below:



Regex Extract Function

Here are the six rows of regex in this example:

Regex; Additional Regex

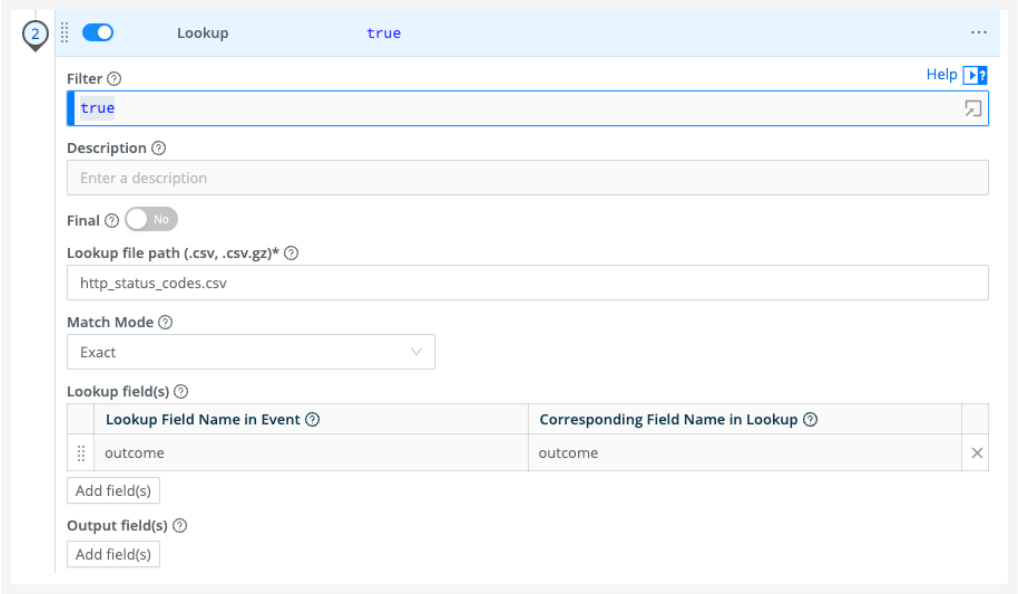
```
/\s\d\d\d\s(?<__bytes>[0-9]{2,})/
/(?<__method>GET|HEAD|POST|PUT|DELETE|CONNECT|OPTIONS|TRACE)/
/HTTP\/(?<__version>[0-9\.]*)\"/
/\s(?<__status>\d\d\d)\s/
/(?<__ip_address>(?:[0-9]{1,3}\.){3}[0-9]{1,3})\s/
/(?<__url>\s\/([^\s]*))/
```

Copy

As you refine your expression, capture a sample of incoming Splunk data to test your regex's results in Cribl Stream's right Preview pane.

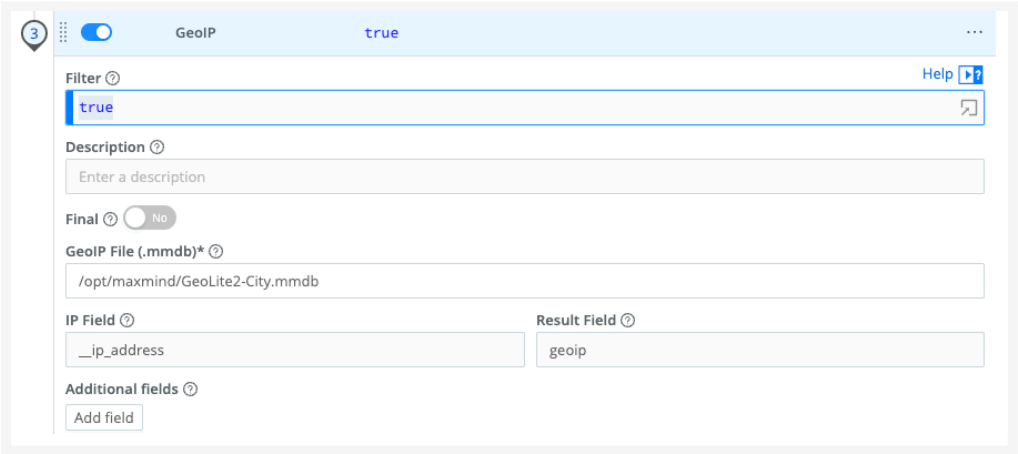
Lookup Function

In this example, we next add a Lookup Function, to translate HTTP error codes to readable text. Note this Function's optional **Reload Period** field, in which you can define a reload interval for a lookup file whose contents refresh frequently.



GeoIP Function

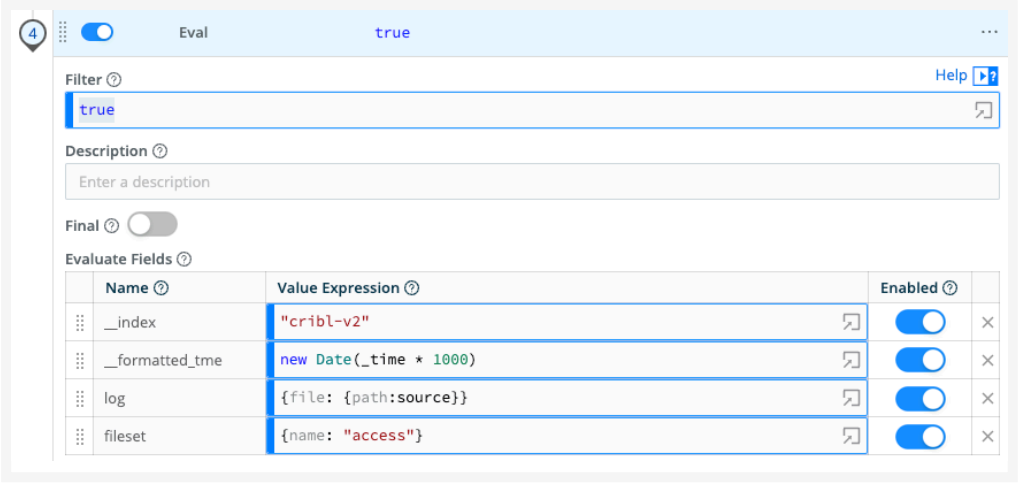
To enrich the Splunk data, we next use a GeoIP Function. This a specialized lookup against a database of IP addresses by geographic location. This Function's output can provide Elasticsearch with `location` fields like `lat` and `long`.



GeoIP specialized lookup

Eval Function

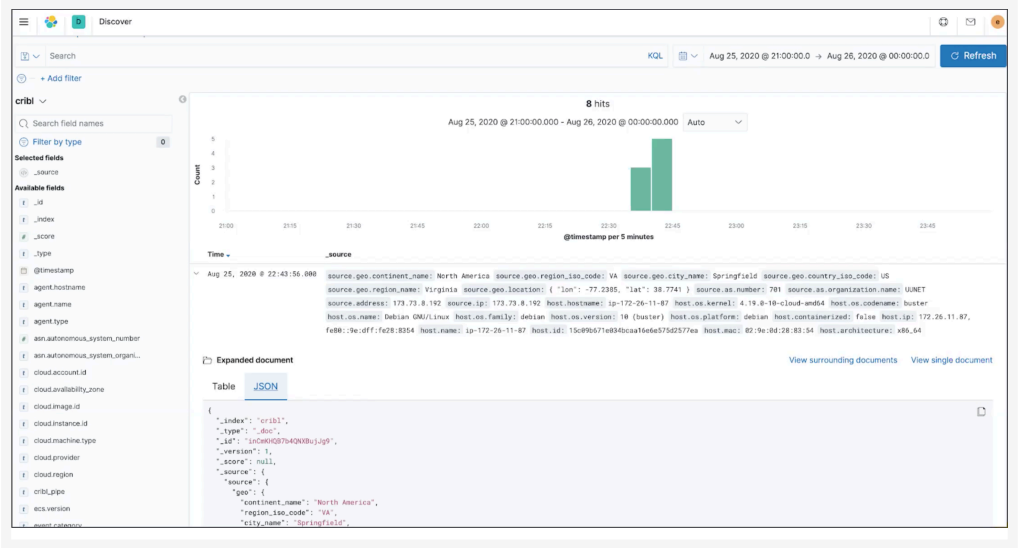
Finally, to further enrich the outbound events, the Pipeline uses an Eval Function. This adds multiple key-value pairs that define and populate fields conforming to the Elastic Common Schema.



Eval Function

Results

After attaching your Pipeline to a Route, here's an an exported event, all happy in Elasticsearch with nested JSON.



Event as exported to Elasticsearch

8.8.3.4. Splunk To Exabeam

In many organizations, the IT department uses a tool like Splunk for operational logging, while the Security team relies on Exabeam to prevent insider threats. These tools use separate agents to access the same data, leading to some data-sharing conundrums:

- Installing the Exabeam agent in parallel with Splunk would duplicate the data.
- Some servers, like domain controllers, allow only a single agent. In this case, you can't feed two platforms with the same data.
- Querying Splunk for the data would introduce extra latency and overhead costs.
- Forwarding data directly from Splunk Universal Forwarders (UFs) is a nonstarter. Classic logs from Splunk UFs embed newlines and special characters, which break Exabeam's parser.

Cribl Stream can help you unblock these issues: Ingest data directly into Cribl Stream from Splunk UFs running on the domain controller, and transform the events in Stream before routing them to Exabeam.

While this example is written around a Splunk-to-Exabeam scenario, you can use the same general techniques to connect and transform data between several other upstream and downstream services.

Transform Data from Splunk to Exabeam's Format

In Cribl Stream's core, you can easily design a Pipeline that modifies the original Splunk event to fit the format that Exabeam expects. Some Cribl Stream Functions useful for this transformation are:

- Serialize: Remove all of the newlines and spaces, and then transform the data into `JSON` format.
- Mask: Remove the special characters, and replace them with space.
- Eval: Create a new field called `Message`, and remove everything else using the **Remove Fields** option.

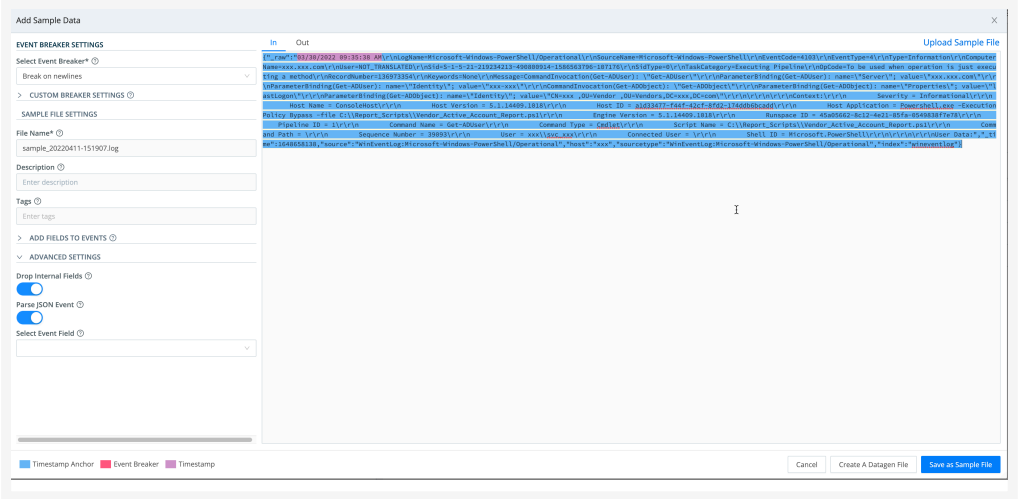
In this guide, we'll show you how to:

1. Capture sample logs.
2. Apply the Functions (outlined above) to transform the sample log into Exabeam's expected format.
3. Validate your fields with Exabeam Parsers.
4. Stream your event to Exabeam to test the entire sequence.

Capture Sample Logs

Start with sample logs of the event data you plan to work with. In our example, we'll copy and paste the log sample straight into Cribl Stream.

Once you've pasted the log sample, enter a unique **File Name** on the modal's left side, then select **Save as Sample File** at right.



Saving captured data

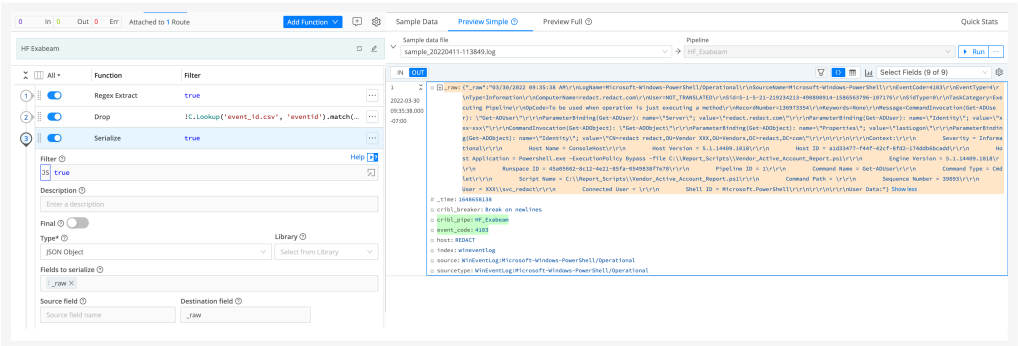
In your production environment, you can filter incoming events in real time (e.g., using a filter expression like `Windows Security logging`) to identify your in-scope Exabeam data.

Configure Pipeline

Next, this section shows relevant Functions that you can assemble into a processing Pipeline to transform the sample log into Exabeam's expected format.

Serialize Function

First, use a Serialize Function to change the event's format into `JSON`. Then, remove the extra lines and spaces, because Exabeam treats each newline as a separate event.

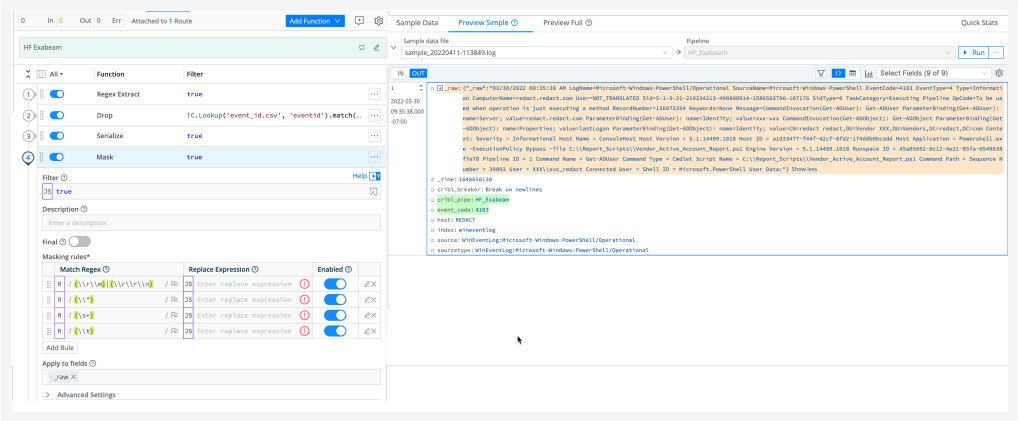


Reformatting the event

Mask Function

Next, use a Mask Function to remove extra `\n` and `\r` characters. (Otherwise, Exabeam's regex filter would reject the characters and drop whole fields that need to be matched and extracted.)

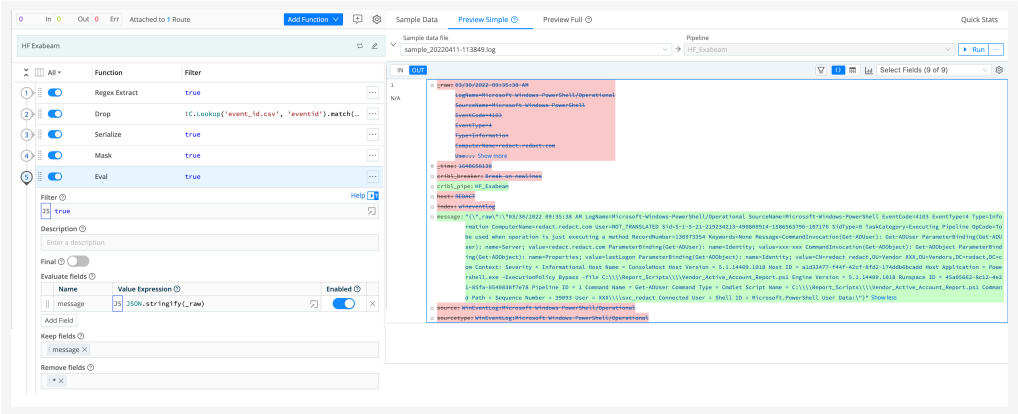
In this example, we are using a Mask Function to remove these special characters and replace them with spaces.



Extraneous newlines/returns to remove

Eval Function

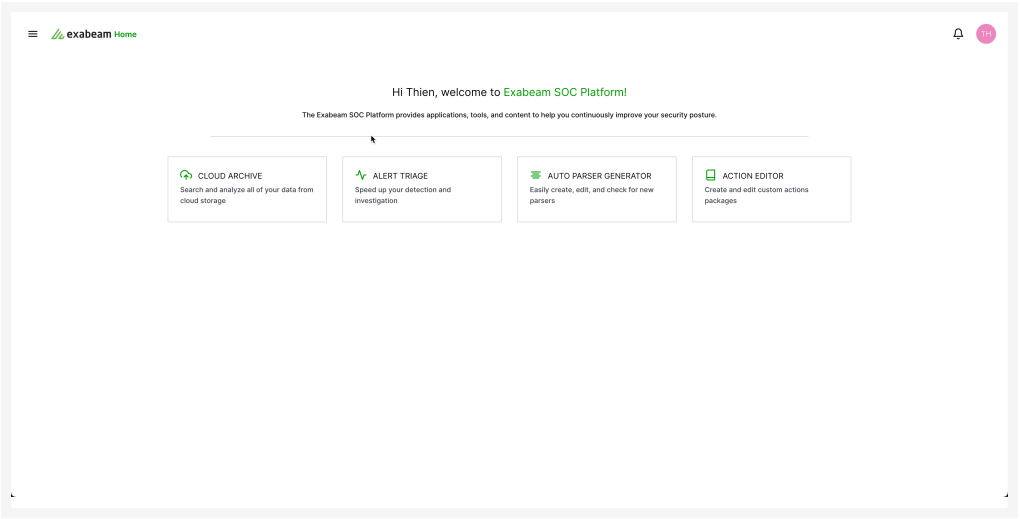
As the last step in Cribl Stream, use an Eval Function to enrich the outbound events, by adding key-value pair that defines and populates a field conforming to the Exabeam schema. Name the new field `Message`. (The `Message` field will populate `Raw Message` in the Exabeam Data Lake.) Remove everything else using **Remove Fields**.



Eval Function to add an Exabeam-specific field

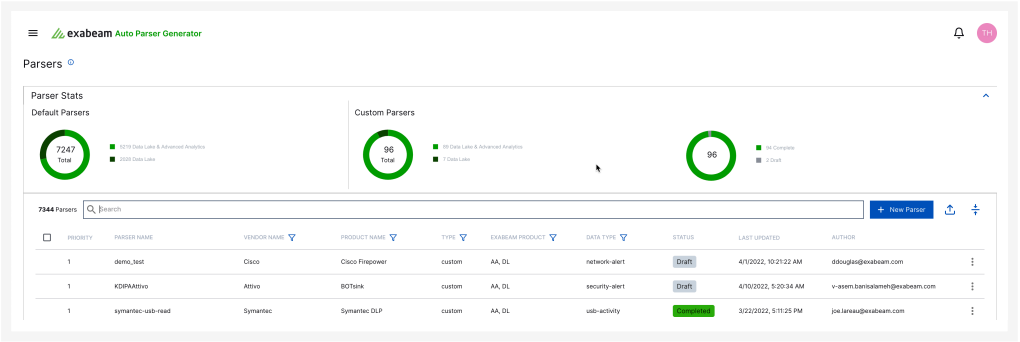
Goat the Fields?

Use Exabeam's Auto Parser Generator (APG) tool to validate your fields. If you don't have the APG tool, contact Exabeam support to request access via your ECP account. This tool validates matching parsers based on your event.



APG tool access in Exabeam

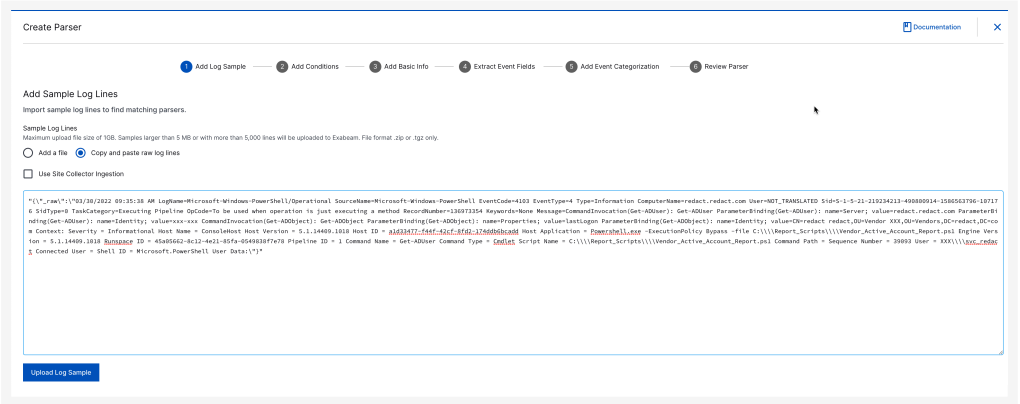
In the APG tool, select **New Parser**.



New Parser option

On the **Create Parser** page, select **Copy and paste raw log lines**.

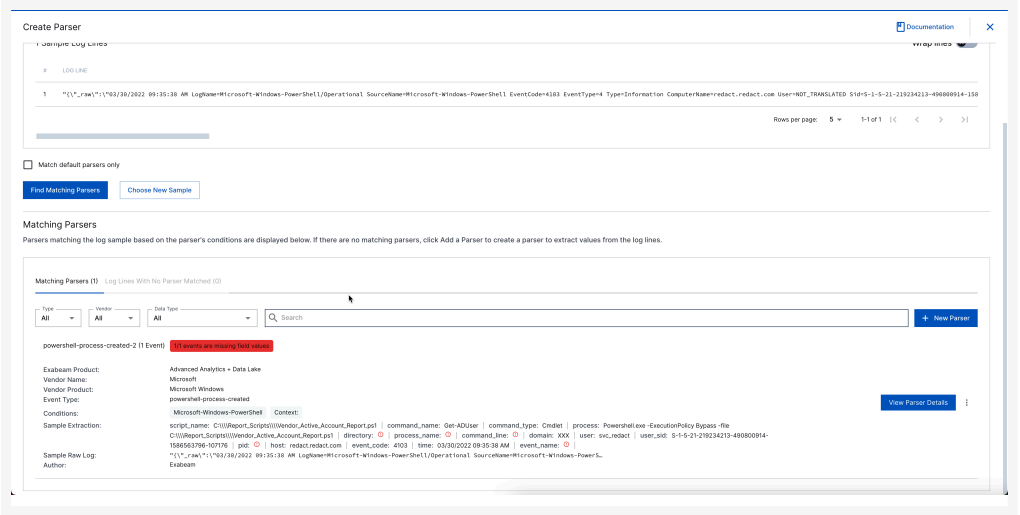
In the text box, paste the **Message** field value from the your sample file and select **Upload Log Sample**.



Paste Message Field

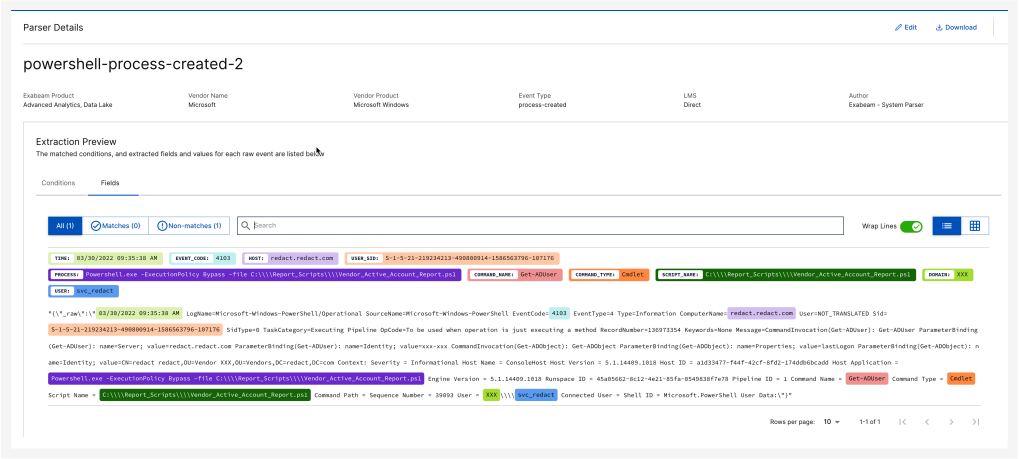
Copy the *Message** field value to your clipboard for a later step in Stream it to Exabeam.

Select **View Parsing Details** to view parsers matching this event type.



New Parser option

Validate that all the fields you need are populated: `src_ip`, `dest_host`, `user`, etc.



Paste Message Field

Do not change Field Names in Exabeam (e.g., `src_ip` to `source_IP`). Exabeam has its own Field Name format that matches its Advanced Analytics template.

At the bottom of the **Extraction Preview** page, select **Configuration Files** to inspect the parsers. If required, you can download the parser to change the regex configurations.

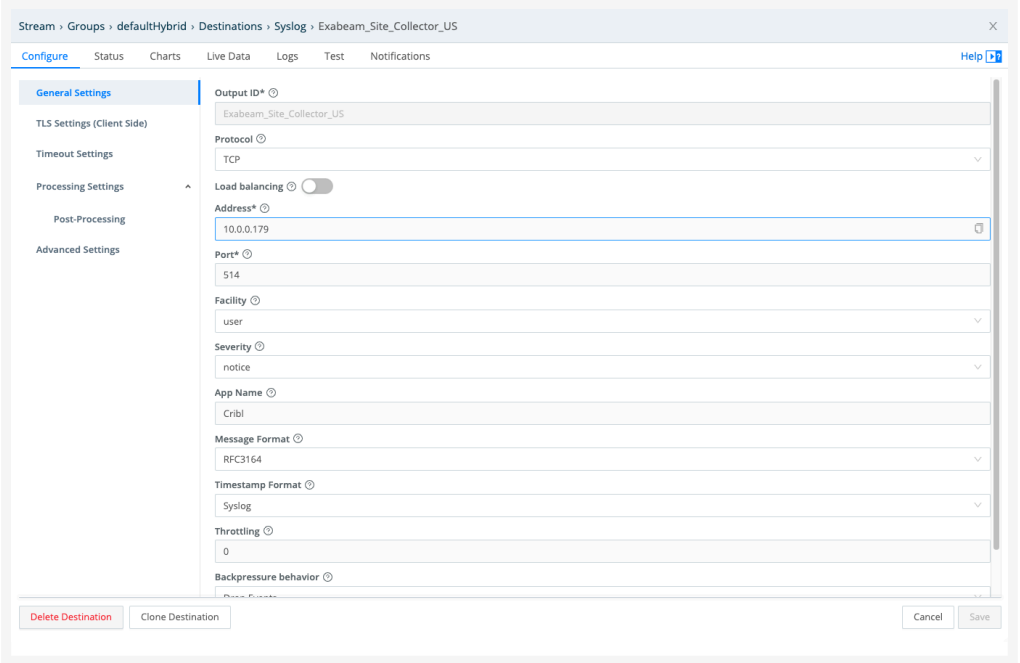
Parser Details

Stream It to Exabeam

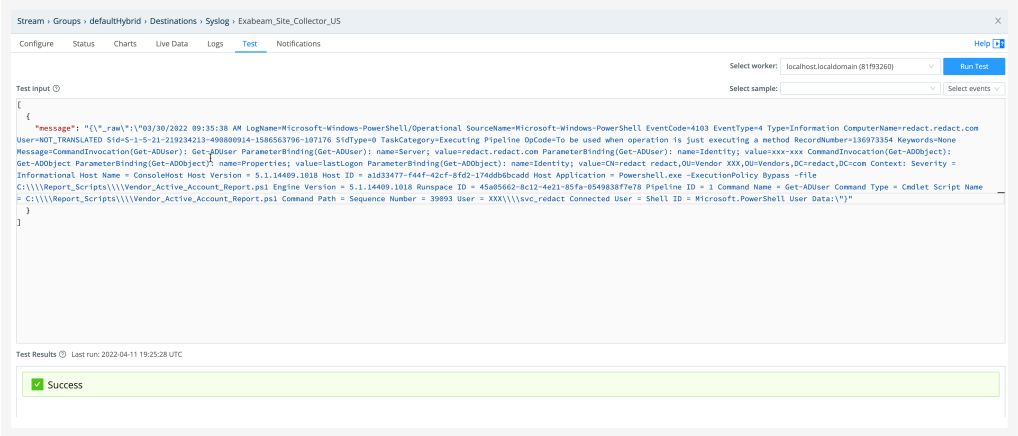
Our final test is to send a single event to Exabeam, and validate the results in Exabeam's Data Lake or Advanced Analytics.

Before testing, configure your Exabeam implementation's `Asset Name` and `Username` to a dummy account. Exabeam's Advanced Analytics has hundreds of models out of the box, and you do not want to ruin these models while testing out your data.

1. From Cribl Stream Home, select a Worker Group, and then select **Data** > **Destinations** > **Syslog**.
2. Select **Add Destination** to configure and save a new Syslog Destination to export data to Exabeam. Configure the **Address**, **Port**, **Message format**, **Timestamp format**, and other options to match your Exabeam implementation.

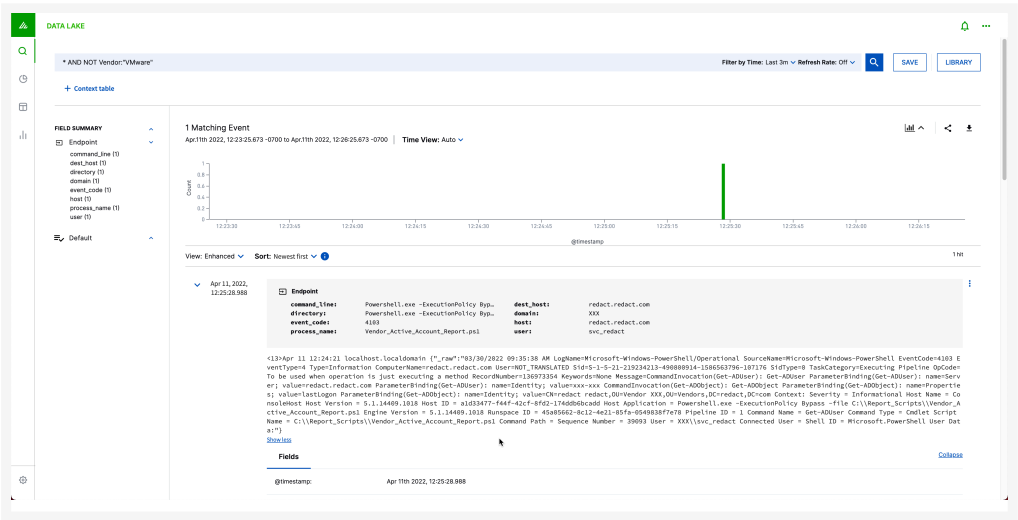


3. Reopen the Syslog Destination. On its **Test** tab, copy and paste the **Message** field's value you copied onto your clipboard above.



Syslog Test tab

4. Within a few seconds, you should see your event display in the Exabeam Data Lake. Search for your forwarder IP/Host (Example syntax: `Forwarder:"IP/host`).



Validating event in Exabeam

5. You can validate that you have the right parser by matching the `exa_parser_name` with the parser in the Auto Parser Generator.

8.8.3.5. Splunk Stream To Cribl Stream

Splunk Stream is a set of three Splunk packages that, combined, enable you to capture and work with streams of network event data. This adds up to a collector for streaming data. You can deploy Splunk Stream in either of two forms:

- As part of a Splunk Universal Forwarder (we'll call this the forwarder-based config), sending data to a Cribl Stream Splunk TCP Source; or
- As an independent stream forwarder (we'll call this the ISF config), running on a compatible Linux machine, and sending data to a Cribl Stream Splunk HEC Source.

In either case, the collector process (not to be confused with a Cribl Stream Collector) will need to call home to a Splunk Enterprise process with the Splunk Stream app installed. You'll manage the collector's settings in Splunk Enterprise.

For the broader Splunk configuration story, see Splunk's documentation. Here, we'll explain one small part of the Splunk configuration process: how to configure Splunk Stream to send the data it captures to Cribl Stream. See the section below - either Forwarder-based or ISF - that corresponds to your use case.

Setting Up the Forwarder-Based Config

Install the Stream TA on the Universal Forwarder targets, either manually or via the deployment server. You'll need to add a new `inputs.conf` stanza pointing to your Splunk Stream App management instance. Adapt the example stanza below, replacing the placeholder with the hostname or IP address of your management host.

New stanza in config file

```
[streamfwd://streamfwd]
splunk_stream_app_location = https://<your_management_host>:8000/en-us/custom/splunk_
disabled = 0
```

Copy

The `outputs.conf` for the Universal Forwarder with Splunk Stream App is the same as for a Universal Forwarder with a non-streaming collector. See these examples.

To verify that your setup is working, run a Live Capture in your Cribl Stream Splunk TCP Source with appropriate filters. Once the Universal Forwarder is sending data to your Cribl Stream Workers, you're ready to begin working with sample captures, Routes, and Pipelines.

If no data seems to be coming through, check the logs located at `/opt/splunkforwarder/var/log/splunk/splunkd.log` on each machine where your Forwarder is

running.

Setting Up the ISF Config

Splunk Stream as an independent stream forwarder (ISF) can run only on Ubuntu- or RHEL-based x64 Linux machines that have bzip2 installed.

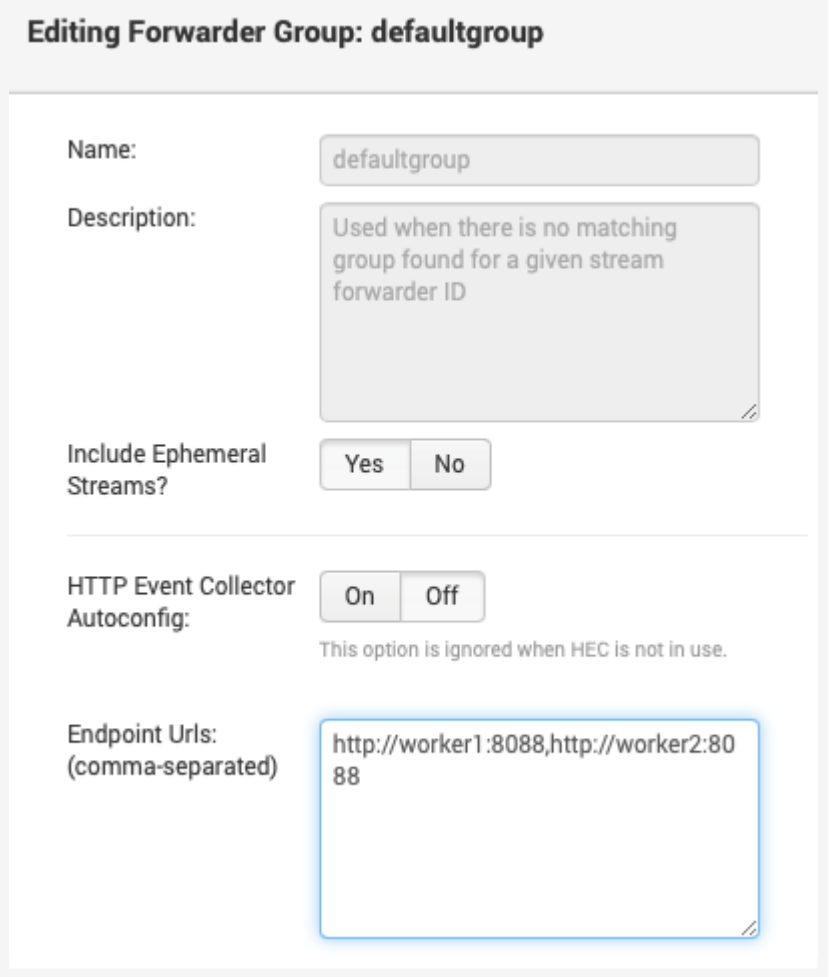
To install Splunk Stream, begin in the Splunk UI:

- In the Stream App, navigate to **Configuration** > **Distributed Forwarder Management**.
- Click the **Install Stream Forwarders** button.
- In the resulting modal, under the text `To get data from other machines, run this command on your data source machine`, copy the `curl` command. E.g.:
  `curl -sSL http://DellT20:8000/en-us/custom/splunk_app_stream/install_streamfwd | sudo bash`

On each Linux machine where you want to install Splunk Stream, run the `curl` command that you copied. Installation will fail if the machine lacks `bzip2`.

Once installation is complete, return to the Splunk UI.

1. To update the HEC endpoint URL, navigate to **Actions** > **Edit Forwarder Group** for the desired group.
2. Toggle **HTTP Event Collector Autoconfig** off.
3. In the **Endpoint URLs** field, list your Cribl Stream Worker URLs, as shown here:



Editing forwarder group to specify Cribl Worker URLs

4. Click **OK**.

Finally, on each Linux machine where you're running Splunk Stream, update the ISF settings with the proper HEC token. (While it is technically possible to run without a token, Cribl strongly recommends against this practice.)

1. In Cribl Stream, copy the HEC token from your Cribl Stream Splunk HEC Source definition (the value of the `__hecToken` internal field).
2. On the desired Linux machines, open `/opt/streamfwd/local/streamfwd.conf` in a text editor.
3. Add the following stanza, substituting your HEC token for the placeholder:

```
[streamfwd]
httpEventCollectorToken = <HEC_token_from_your_Cribl_Stream_Source>
```

Copy

4. Run the following command to restart the ISF:

```
systemctl restart streamfwd
```

Copy

To verify that your setup is working, run a Live Capture in your Cribl Stream Splunk HEC Source with appropriate filters.

You can also check ISF status in Splunk. Navigate to **Apps** > **Splunk Stream** > **Admin Dashboards** > **Stream Forwarder Status**:

8.8.3.6. Decryption Of Data In Splunk

Currently, Cribl Stream supports decryption only when Splunk is the end system. In Splunk, decryption is available to users of any role with permissions to run the `cribldecrypt` command that ships with Cribl App for Splunk. Further restrictions can be applied with Splunk **capabilities**. This page provides details.

For clarity, Cribl introduced the `cribldecrypt` as an alias to the original `decrypt` command. Use this alias to avoid conflicts with Splunk's internal commands. (We show it in the examples below.) Both are, in fact, aliases to the actual command: `/path/2/cribl --splunk-decrypt`. You can use both aliases.

Usage

The `cribldecrypt` command is used to display Cribl-encrypted fields in cleartext. It is an alias to the `decrypt` command. This command decrypts fields only for the encryption keys that the user can access.

The example below retrieves data from a Splunk index with Cribl-encrypted data, and pipes it to the `cribldecrypt` command:

```
index=index_with_encrypted_fields | cribldecrypt
```

Copy

Decrypting in Splunk

Decryption in Splunk is implemented via a custom command called `cribldecrypt`. To use the command, users must belong to a Splunk role that has permissions to execute it. Capabilities, which are aligned to Cribl Key Classes, can be associated with a particular role to further control the scope of `cribldecrypt`.

Decrypt Command Is Search Head ONLY

To ensure that keys don't get distributed to all search peers - including peers that your search head can search, but you don't have full control over - `cribldecrypt` is scoped to run locally on the installed search head.

Restricting Access with Splunk Capabilities

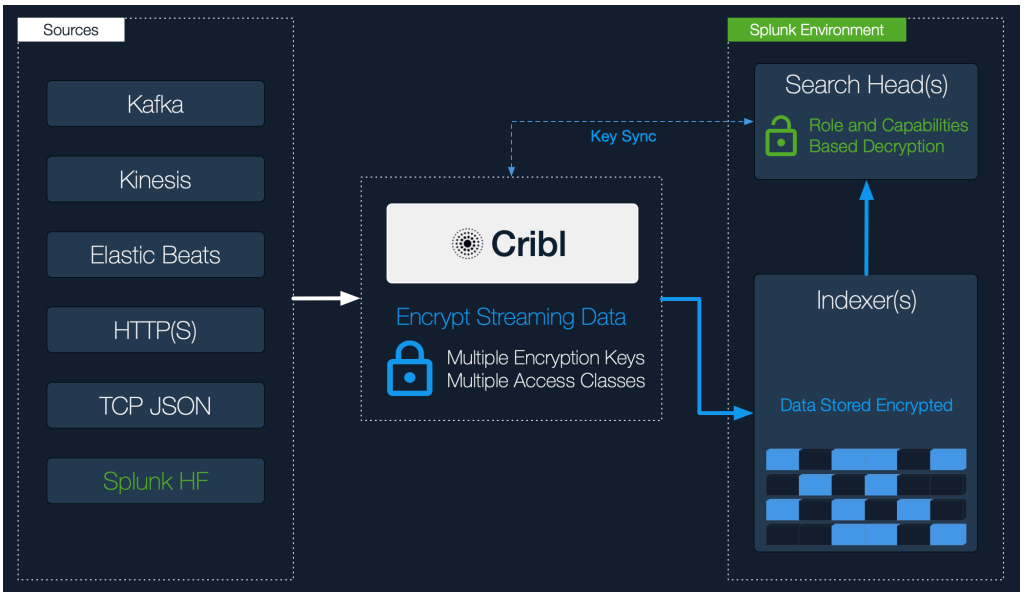
In Splunk, capability names should follow the format `cribl_keyclass_N`, where `N` is the Cribl Key Class. For example, a role with capability `cribl_keyclass_1` has access to all key IDs associated with key class `1`.

Capability Name	Corresponding Cribl Key Class

`cribl_keyclass_1`
`cribl_keyclass_2`
…
`cribl_keyclass_N`
	1
	2
	…
	N

Configuring Splunk Search Head to Decrypt Data

You set up decryption in Splunk according to this schematic:



1. Download the Cribl Stream App for Splunk from Cribl's Download Cribl Stream page: In the **Select Release Version** dropdown, select the Cribl App for Splunk option.

2. To install the Cribl Stream App for Splunk on your search head, untar the package into your `$SPLUNK_HOME/etc/apps` directory.
   The app will run in search head mode by default. If the app has previously been installed and later modified, you can convert it to search head mode with the command: `$CRIBL_HOME/bin/cribld mode-searchhead`. (When installed as a Splunk app, `$CRIBL_HOME` is `$SPLUNK_HOME/etc/apps/cribl`.)

3. Assign permissions to the `criblddecrypt` command, per your requirements.

4. Assign capabilities to your roles, per your requirements. If you'd like to create more capabilities, ensure that they follow the naming convention defined above.

5. In the `$SPLUNK_HOME/etc/apps/cribl/local/cribl/auth/` directory, sync `cribl.secret` | `keys.json`. (To successfully decrypt data, the `criblddecrypt` command will need access to the same keys that were used to encrypt, **in the Cribl instance where encryption happened**.)
   - In a single-instance deployment, the `cribl.secret` and `keys.json` files reside in: `$CRIBL_HOME/local/cribl/auth/`.

- In a distributed deployment, these files reside on the Leader Node in: `$CRIBL_HOME/groups/<group-name>/local/cribl/auth/`.
- When using Cribl Stream's UI, you can download these files by clicking the **Get Key Bundle **button.

Sync/copy these files over to their counterparts on the search head (decryption side). In a non-Splunk integration, you would copy these assets to wherever decryption will take place.

Modifying Keys

When you update keys by editing the `keys.json` file, you must add them back to the directories above (respectively, on a single instance or on a distributed deployment's Leader Node).

8.8.3.7. Stream-To-Splunk Encryption

Encrypt at Ingest-Time and Decrypt in Splunk

With Cribl Stream, you can encrypt your sensitive data in real time before it's forwarded to and stored at a destination. Using the out-of-the-box Mask function, you can define patterns to encrypt with specific key IDs or key classes. To decrypt in Splunk, you will need to install Cribl App for Splunk on your search head. (The app will default to `mode-searchhead`.)

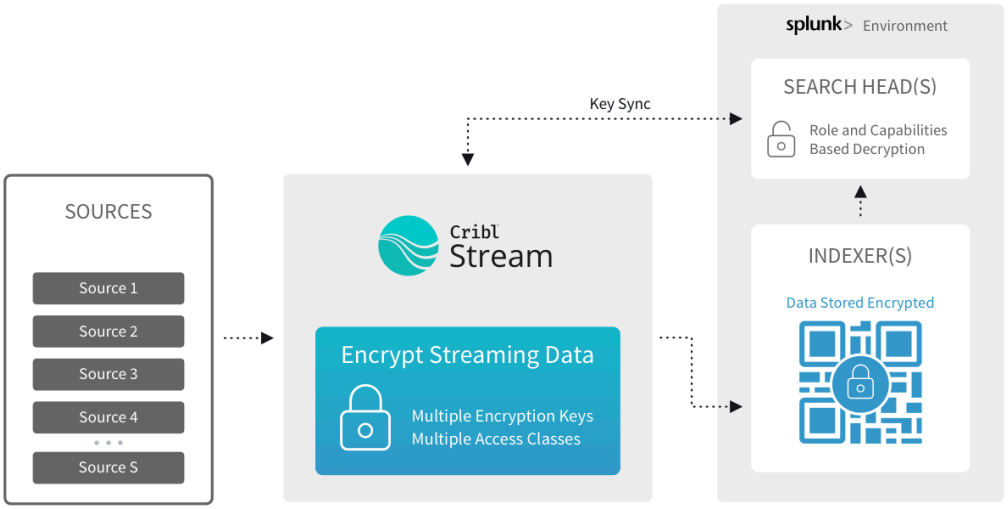
Keys and Key Classes

Symmetric encryption keys can be configured through the CLI or the UI. They're used to encrypt the patterns, and users are free to define as many keys as required.

Key classes are collections of keys that can be used to implement multiple levels of access control. Users (or groups of users) that have access to data with encrypted patterns can be associated with key classes. You can use these classes to provide more-granular access rights, such as read versus decryption permissions on a dataset.

Encrypt in Cribl Stream and Decrypt in Splunk

1. Define one or more keys and key classes on Cribl Stream. (See UI- and CLI-based instructions.)
2. Sync `auth` with the decryption side (Splunk Search Head). (The Splunk-side directory is `$SPLUNK_HOME/etc/apps/cribl/local/cribl/auth/`.)
3. Apply the Mask function to patterns of interest, using C.Crypto.encrypt().
4. Decrypt on the Splunk search head, using Role-Based Access Control on the `decrypt` command.

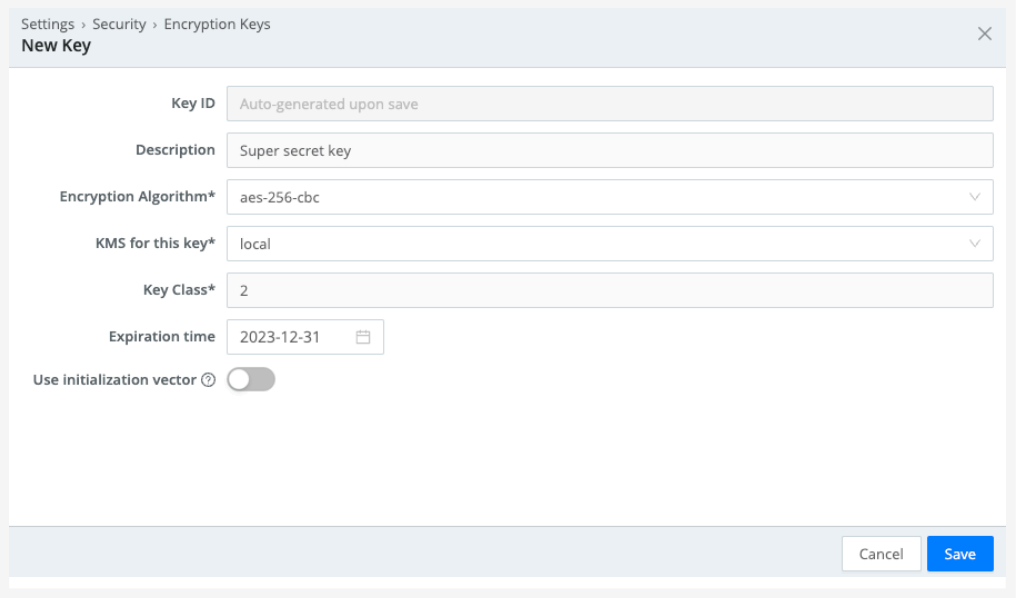


Examples

Encryption Side

You can generate keys via the UI or the CLI.

To generate keys via the UI, access **Group Settings** > **Security** > **Encryption Keys**:



Adding a new encryption key

To generate one or more keys via the CLI, pattern your commands after these examples.

In a single-instance deployment:

```
$CRIBL_HOME/bin/cribl keys add -c 1 -i ... $CRIBL_HOME/bin/cribl keys add -c <N> -i
```

In a distributed deployment, to generate keys on a Worker Group named `uk`:

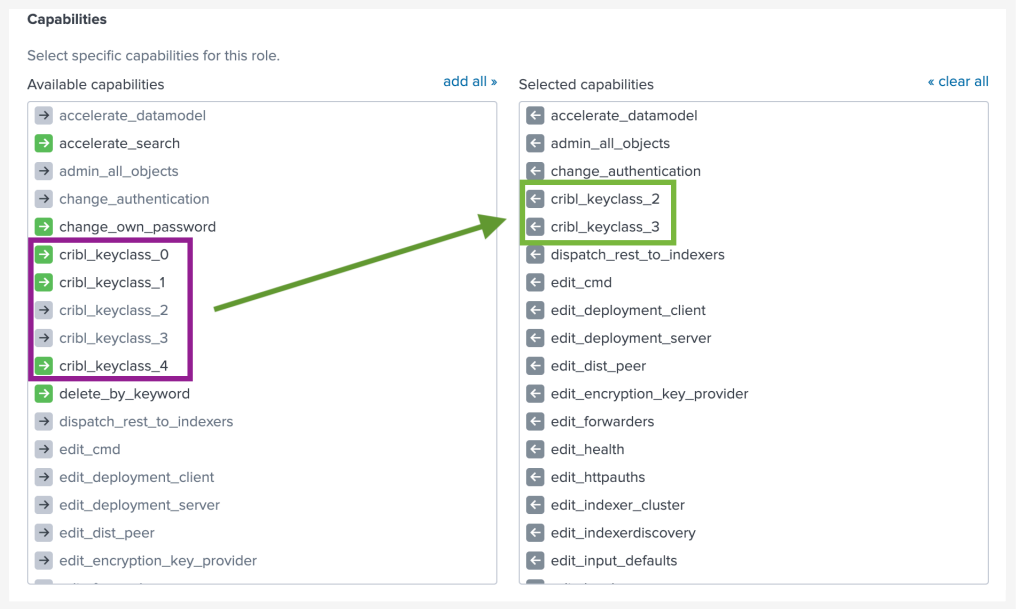
```
$CRIBL_HOME/bin/cribl keys add -c 1 -i -g uk ... $CRIBL_HOME/bin/cribl keys add -c <N>
-i -g uk
```

Add `-e <epoch>` to the above commands if you'd like to set expiration for your keys.

For all command/syntax options, see Adding Keys.

Decryption Side

- Download the Cribl Stream App for Splunk from Cribl's Download Cribl Stream page: In the **Select Release Version** dropdown, select the Cribl App for Splunk option.
- To install the Cribl Stream App for Splunk on your search head, untar the package into your `$SPLUNK_HOME/etc/apps` directory. The app will default to `mode-searchhead`.
- Assign permissions to the `decrypt` command, per your requirements.
- Assign capabilities to your Roles, per your requirements. Capability names should follow the format `cribl_keyclass_N`, where `N` is the Cribl Key Class. For example, a role with capability `cribl_keyclass_1` has access to all key IDs associated with key class `1`. You can use more capabilities, as long as they follow this naming convention.



Selecting capabiities

- In the `$SPLUNK_HOME/etc/apps/cribl/local/cribl/auth/` directory, sync `cribl.secret` | `keys.json`. (To successfully decrypt data, the `decrypt` command will need access to the same keys that were used to encrypt, **in the Cribl instance where encryption happened**.)
    - In a single-instance deployment, the `cribl.secret` and `keys.json` files reside in: `$CRIBL_HOME/local/cribl/auth/`.
    - In a distributed deployment, these files reside on the Leader Node in: `$CRIBL_HOME/groups/<group-name>/local/cribl/auth/`.
    - When using Cribl Stream's UI, you can download these files by clicking **Get Key Bundle**.

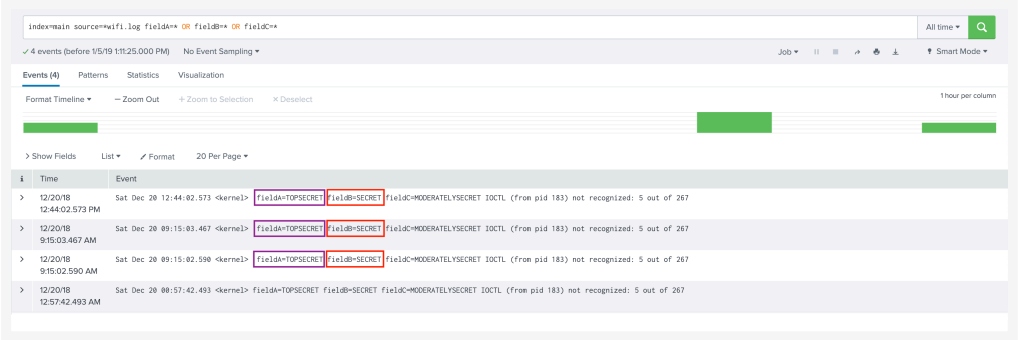
  Sync/copy these files over to their counterparts on the search head (decryption side). In a non-Splunk integration, you would copy these assets to wherever decryption will take place.

Modifying Keys

> When you update keys by editing the `keys.json` file, you must add them back to the directories above (respectively, on a single instance or on a distributed deployment's Leader Node).

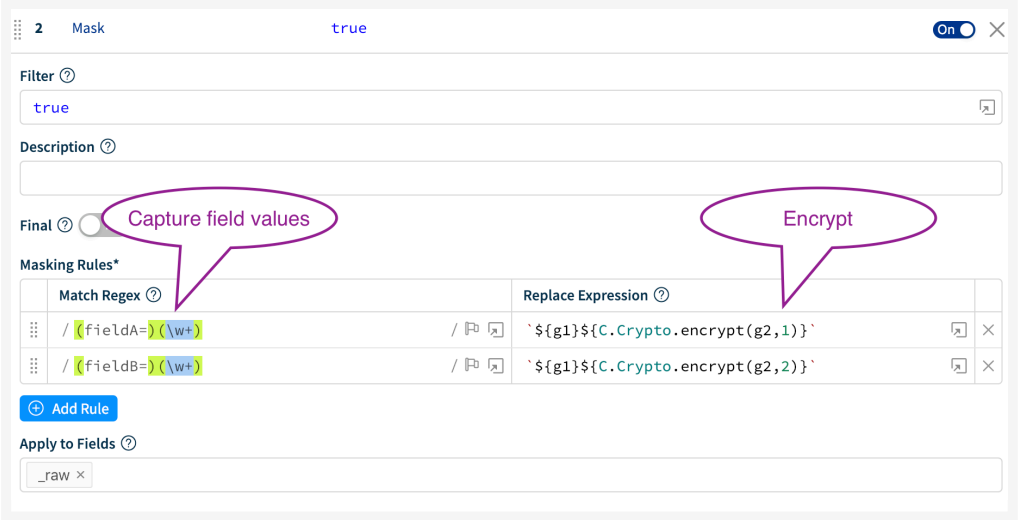
Usage

Before Encryption: Sample un-encrypted events. Notice the values of `fieldA` and `fieldB`.



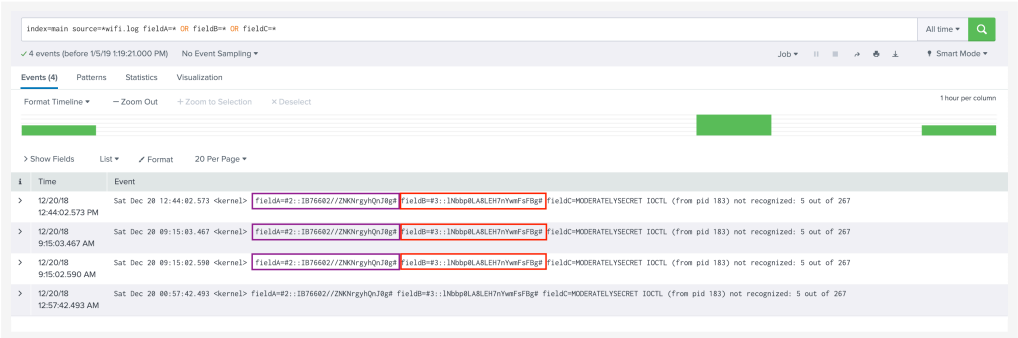
Events before encryption

Next, encrypt `fieldA` values with key class `1`, and `fieldB` with key class `2`.



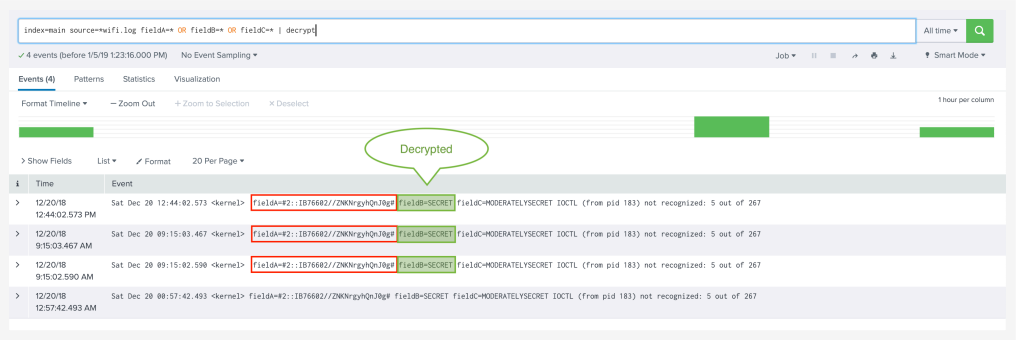
Encrypting two fields with separate key classes

After Encryption: again, notice the values of `fieldA` and `fieldB`.



Both fields encrypted

Here, we've decrypted `fieldB` but not `fieldA`. This is because the logged-in user has been assigned the capability `cribl_keyclass_2`, but not `cribl_keyclass_1`.



One field decrypted

8.8.4. Syslog

The following topics provide detailed guides and better practice for integrating with Syslog services.

Source	Description
Syslog Best Practices	Best practices for processing syslog events
Palo Alto Syslog to Cribl.Cloud	Palo Alto Syslog Forwarding to Cribl.Cloud

8.8.4.1. Syslog Best Practices

Cribl Stream can process a syslog stream directly. Moving to Cribl Stream from existing syslog-ng or rsyslog servers fully replaces those solutions with one that is fully supported and easily managed.

Processing syslog in Cribl Stream allows you to readily address these common challenges of ingesting data from syslog senders:

- **Architecture**: Cribl Stream routes the syslog stream directly, immediately, and securely to the destinations of your choice, reducing latency and management overhead.
- **Volume**: Removing redundant data and unnecessary fields in Cribl Stream typically reduces volume 20-30% overall. It also optimizes the data for downstream services like Splunk or Elasticsearch.
- **Timestamp handling**: Cribl Stream intelligently processes events sent from different time zones. It can embed a new, consistent timestamp, and can auto-correct timestamps that are off by an exact number of hours.
- **Severity/Facility accuracy**: Each syslog event begins with a bracketed integer that represents Facility and Severity, as defined in the Syslog Protocol. Cribl Stream translates this code (e.g., `<165>`, `<0>`) into the correct Facility and Severity values.
- **Metadata**: Cribl Stream can automatically set metadata fields, including `sourcetype` and `index`.

This tutorial outlines best practices for replacing your syslog server with Cribl Stream. To go even a bit deeper, check out this Cribl Office Hours video.

The Goal: Optimizing Syslog Events

By default, a Cribl Stream Syslog Source produces eight fields: `_time`, `appname`, `facility` (numeric), `facilityName` (text), `host`, `message`, `severity` (numeric), and `severityName` (text).

Default, parsed syslog event

While this default parsing makes the output much more readable, we haven't saved any volume - and we now have redundant pairs of fields (numeric versus text) that represent facility and severity.

Our next logical step is to streamline syslog events to something more like this:

```
3
2021-11-18
16:33:06.000
-08:00
  _raw: dropbear[25092]: Child connection from 222.186.30.76:19708
  _time: 1637281986
  cribl_pipe: CriblSyslogPreProcessing
  facility: 10
  facilityName: authpriv
  host: 66.241.71.15
  index: syslog
  message: dropbear[25092]: Child connection from 222.186.30.76:19708
  severity: 6
  severityName: info
  source: syslog:in_syslog:udp
  sourcetype: syslog
```

Default, parsed syslog event

This accomplishes all of the following:

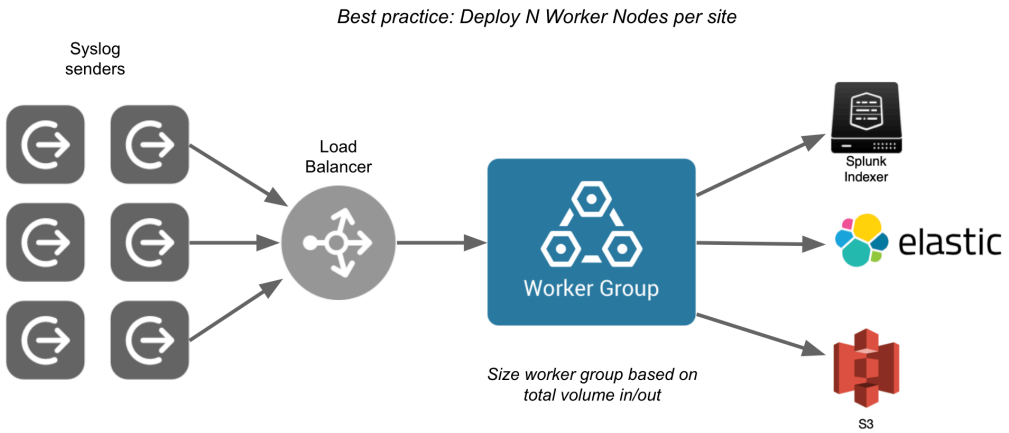
- Extracts the essentials.
- Removes the redundancies.
- Adds a new field to identify the Cribl Stream Pipeline (which we're about to build).
- Adds metadata that the Destination needs.
- Shrinks the outbound `_raw` payload to just its `message` component.

Once we optimize syslog in this way, we can achieve still further efficiencies by dropping or downsampling frequent events, and by balancing high-volume syslog inputs across Cribl Stream worker processes.

Architecture: On-Prem Cribl Stream Deployments

Syslog data, especially when sent via UDP, is best collected as close to the source as possible. Ideally, you should capture syslog data **at its origin**. (This is true of syslog in general, not just syslog processed in Cribl Stream.)

Also, because syslog senders have no built-in load balancing, Cribl strongly recommends using a load balancer to distribute the load across multiple Worker Nodes.



Example architecture for a single site or location

The load balancer shown above is dedicated only to communication between the syslog senders and the Stream Worker Nodes. Any load balancing between the Worker Group and its Stream Leader would be handled by a separate load balancer.

Load Balancers

The load balancer in front of the Leader UI can be an Application Load Balancer or a Network Load Balancer.

The load balancer between the Cribl Workers and the Cribl Leader listening on port 4200 must be a Network Load Balancer or other load balancer that supports health checks.

For load balancer configuration examples showing how to distribute syslog traffic across multiple Cribl Stream Worker Nodes, see Third-Party Load Balancer Example Configurations for Cribl Stream Syslog.

High-Volume Syslog Senders

For general guidance on handling high-volume senders in Cribl Stream, see Scaling for High-Volume Syslog Senders for details on capacity planning, TCP load balancing, and other performance considerations.

For individual high-volume syslog senders sending via TCP above 300 GB/day, it is recommended to enable the TCP Load Balancing function (available in Cribl Stream single-instance and Distributed deployments). This feature prevents "TCP pinning" by distributing the syslog traffic across all Worker Processes, improving overall performance. When enabled, a dedicated load balancer process distributes incoming data via IPC sockets. For details, see TCP Load Balancing.

UDP Versus TCP

For any given syslog device, you might need to choose between using UDP or TCP. Which is best depends on the characteristics of the sender. Here are some basic guidelines:

- For both the sender and Cribl Stream, UDP imposes lower overhead than TCP. The stateless nature of UDP allows each log event to be directed to a different worker thread than the last. This ensures maximum utilization of the Worker Nodes. See Sizing and Scaling for more details.
- For lower-volume senders, use TCP if the sender supports it.
- For all other use cases, use UDP.

Optimize Syslog with Cribl.Cloud

When deploying syslog solutions with Cribl.Cloud, consider these recommendations for optimal performance and security.

Prioritize Hybrid Stream Workers for On-Premise Syslog

For syslog senders located within your on-premise environment, deploying Cribl Hybrid Stream Workers is strongly recommended. Place two or more Workers in the same network proximity as your syslog devices, managed by your Cribl.Cloud Leader. This strategy offers significant advantages:

- **Enhanced UDP delivery**: We recommend local processing due to UDP's "best-effort" nature and lack of encryption. Hybrid Workers improve delivery reliability and prevent sensitive data from traversing the internet unencrypted.
- **Robust Security**: By keeping unencrypted TCP traffic within your local network (especially from senders lacking TCP/TLS support), you maintain data security. Cribl Stream can then encrypt data in transit between hybrid Workers and Cribl.Cloud.
- **Simpler Port Management**: Hybrid Workers provide flexible port configuration, enabling dedicated ports per device type for streamlined management and monitoring.
- **Improved Monitoring**: Dedicated ports enable the Cribl Monitoring dashboards to report volumes for each major type of syslog sender. While this benefit technically applies to Cribl.Cloud deployments as well, the number of available ports is limited. See the next section for additional details on Cribl Cloud port limitations.

If your environment does not allow for the deployment of Hybrid Cribl Stream Workers, the details below will help you optimize delivery to Stream Workers hosted in Cribl Cloud.

Port Considerations in Cribl.Cloud

Cribl.Cloud provides a limited selection of pre-configured ports for common use cases, primarily for receiving Syslog data.

In addition to the pre-configured Syslog ports, Cribl.Cloud offers a range of configurable TCP ports from `20000` to `20010`. These ports can be assigned to **any push-based Source** (including but not limited to Syslog) to receive **any supported data type**.

Pre-configured Syslog Ports

Cribl.Cloud provides the following fixed ports specifically for Syslog traffic:

Port	Protocol	Usage	Notes
6514	TCP + TLS	Encrypted Syslog	Standard for Syslog over TCP + TLS.
9514	TCP or UDP	Unencrypted Syslog	

Port	Protocol	Usage	Notes
514	TCP or UDP	Unencrypted Syslog	Redirected internally to `101514`. This port is available for devices that cannot be configured to use a non-default Syslog port.

For unencrypted **UDP** delivery of Syslog traffic, only ports `514` and `9514` are available.

Port Flexibility with Hybrid Stream Workers

For maximum port flexibility, **Hybrid Stream Workers** support any port, offering a significant advantage over the standard, more restricted port options available directly in Cribl-managed Workers.

Important Security Considerations

Avoid sending unencrypted traffic over the Internet. When using Cribl.Cloud, using syslog with TLS is strongly recommended. Unencrypted options accommodate senders lacking TLS support. For such senders, consider deploying Hybrid Stream Workers on the same LAN to keep unencrypted traffic on private networks.

Pipeline Planning

In Cribl Stream, we speak of **pre-processing Pipelines**, **processing Pipelines** (or just plain **Pipelines**), and **post-processing Pipelines**. Cribl recommends combining the first two in an arrangement that we've found to be optimal for syslog.

Pre-Processing Syslog Sources

Pre-processing Pipelines are useful for standardizing or normalizing incoming data from multiple Syslog Sources. Pre-processing can ensure consistency, streamline ingest-time processing for syslog events, and reduce the need to replicate the functionality and logic in each processing Pipeline that is associated with a Syslog Source.

However, pre-processing may not be necessary or desirable in every case. For example, if you have unique processing requirements for different subsets of data, it may be more effective to configure separate processing Pipelines according to your needs.

The Cribl Pack for Syslog Input provides a pre-processing Pipeline that modifies the data that Cribl Stream delivers. Before you deploy the pre-processing Pipeline, review the Pipeline's comments as described in Importing the Pre-Processing Pipeline to understand the available functions and options and enable or disable them as desired.

If your syslog senders produce events with embedded newline or carriage-return characters (multi-line or otherwise malformed syslog), treat them as malformed syslog rather than standard single-line

> events. In those cases, consider ingesting the data via a Raw TCP or Raw UDP Source with a custom Event Breaker, as described in Alternative Sources for Malformed Syslog .

Pipeline for Selected Syslog Data Subsets

Configure dedicated Pipelines (and Routes!) for each distinct subset of data that arrives via syslog senders. These subsets might include DHCP logs from the router, traffic logs from the firewall, operational logs from load balancers, and virtualization logs from on-prem virtualization hypervisors.

Certain kinds of subset-specific processing have become Cribl Stream best practices. These include:

- For ESXi or other hypervisor logs, use the Drop Function to drop `debug`-level logs.
- For firewall logs, enrich with DNS names, GeoIP fields, and lookups from a threat list.
- For load balancer logs, use the Suppress Function to reduce volume.

Unless you're new to Cribl Stream, you've already created your own Pipelines, so we're not going to review that here. (If you are new to Cribl Stream, consider running through the free Cribl Stream Overview sandbox course ASAP.)

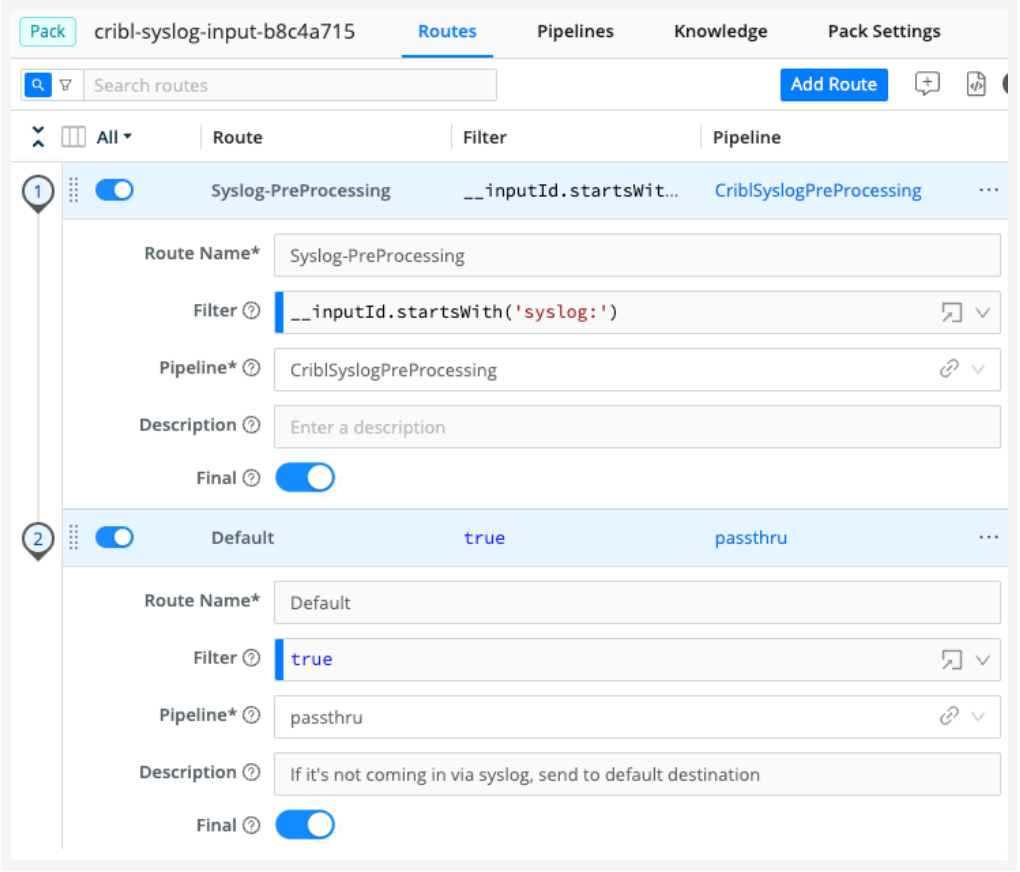
Importing the Pre-Processing Pipeline (Optional)

Even before setting up a Cribl Stream Syslog Source, you have the option to install the Cribl Pack for Syslog Input (a.k.a. `cribl-syslog-input` Pack), which provides the pre-processing Pipeline for mixed-use Syslog Cribl Sources. This is particularly helpful when you have various types of syslog devices sending data to a single port, such as port 514. When you configure dedicated ports for each type of device, it's generally best to use a Pipeline or Pack that is specifically tailored to that type of data.

If this is your first time installing from the Cribl Dispensary, see the full directions with screenshots. Otherwise:

1. Open the Cribl Pack Dispenary's Cribl Pack for Syslog Input page.
2. Under **Releases** at right, click the link for the latest release.
3. In the resulting **Assets** section, right-click the `.crbl` file to download it locally.
4. In Cribl Stream, select a Worker Group, then select **Processing** > **Packs**.
5. Click the **Add New** button, and select **Import from file**.
6. Select the Pack you downloaded, and follow the prompts from there. In most cases, when installing a Dispensary Pack, you should not enter an override value for **New Pack ID**.
7. Review the Pack's `README` file, available in the Pack's **Settings** link and also online.

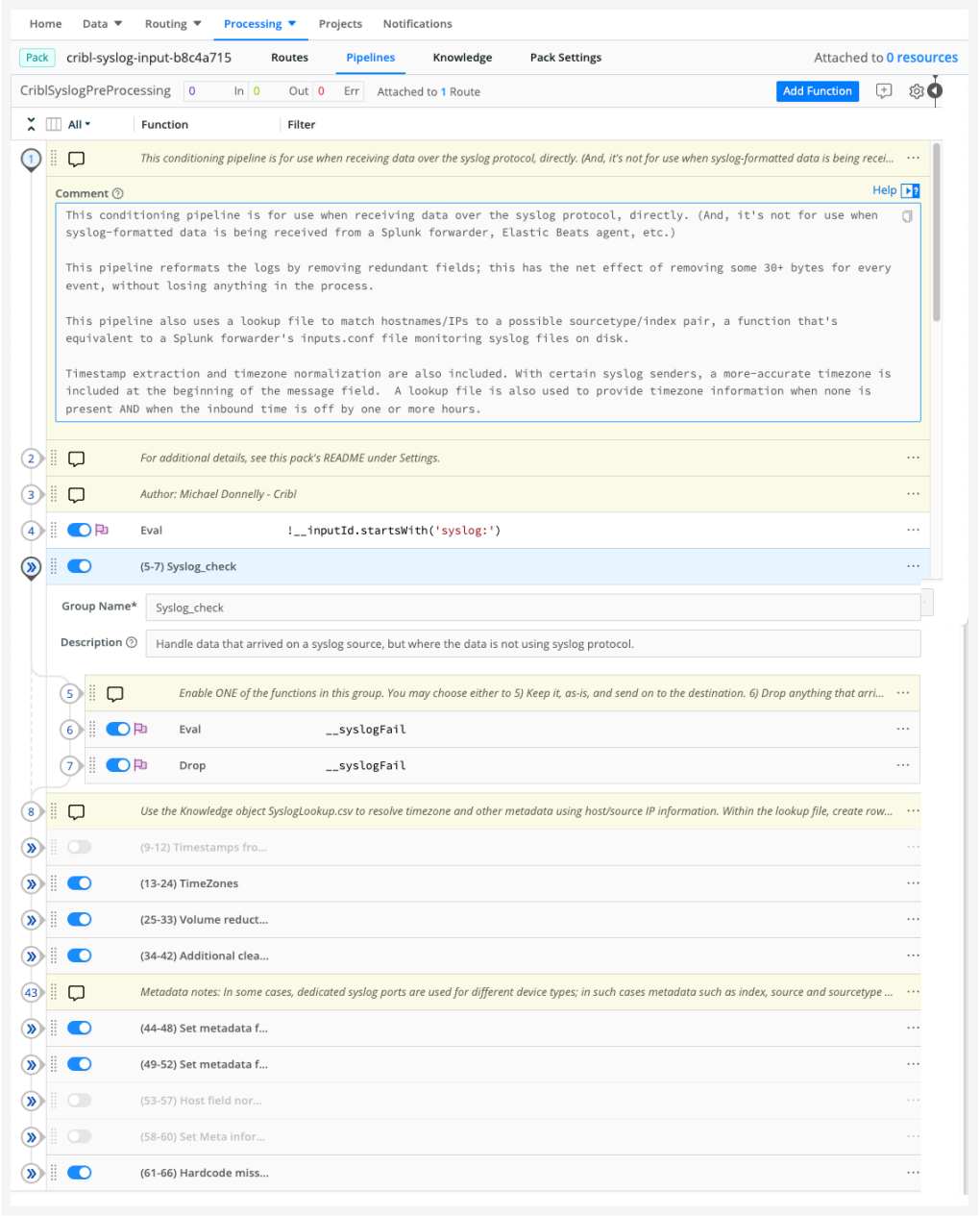
Let's examine what this Syslog Input Pack provides, starting with the Routes page.



Routes page in the Cribl Pack for Syslog Input

The first Route matches based on `inputId`. Anything arriving via a Cribl Stream Syslog Source will match this filter, as long as you've configured the Source to use the Pack, as shown below.

On first installation of the Pack, the Pipeline defaults to commonly used configurations. Cribl **strongly** recommends reviewing the Pipeline's internal documentation to understand which features are enabled. This documentation consists of Comments for each section, and a Description for each Function to explain what's happening in that step.

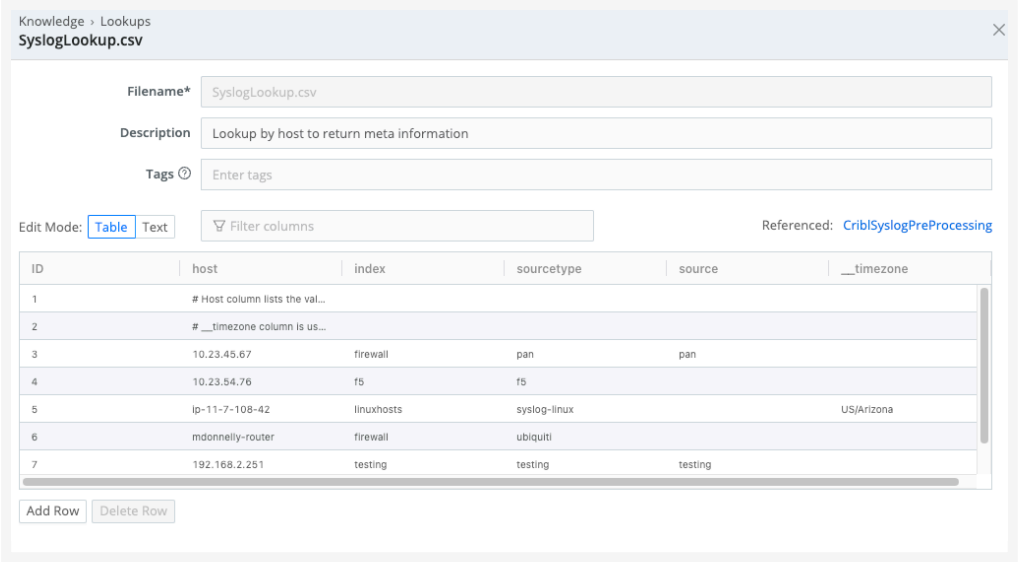


The Pipeline's internal documentation

Examining this documentation should make it clear what changes (if any) are needed to suit your deployment environment. Go ahead and make those changes now.

The Lookup File

The Pack ships with a lookup file called `SyslogLookup.csv`, whose contents you should replace as necessary. To access the file, navigate to **Processing** > **Packs**, select the `cribl-syslog-input` Pack, and click **Knowledge**.



Stock `SyslogLookup.csv` file, to be filled with customer data

Creating Syslog Sources

Now that you've imported the pre-processing Pipeline, the next step is to ensure that you have the Syslog Sources you need.

Configure the `in_syslog` Source

Cribl Stream ships with a Syslog Source named `in_syslog`, which is preconfigured to listen for both UDP and TCP traffic on Port 9514. You can clone or directly modify this Source to further configure it, and then enable it.

Use the `in_syslog` Source for syslog senders that are hard-coded to send to port 514, as described above. In the **QuickConnect** UI: Click **Add Source**. From the resulting drawer's tiles, select [**Push** >] **Syslog**. Click **Select Existing**, then `in_syslog`.

Or, in the **Data Routes** UI: From the top nav of a Cribl Stream instance or Group, select **Data** > **Sources**. From the resulting page's tiles or the **Sources** left nav, select [**Push** >] **Syslog**. In the **Manage Sources** page, click `in_syslog`.

Configure the fields and options as follows:

General Settings

- **Enabled**: Toggle on.
- **UDP Port** and **TCP Port**: Assuming you're following the guidelines in this tutorial, you'll have a load balancer relaying incoming traffic from port 514 to port 9514. Therefore, leave the `in_syslog` Source configured to listen on its default port, 9514, for both TCP and UDP.

TLS Settings (TCP Only)

- Enabling TLS is strongly recommended if the Source will be receiving data across the Internet.

Processing Settings

- **Metadata**: There is no need to add fields here, because for generic senders coming in on 9514, the Pack will set the meta-information via the `SyslogLookup.csv` file. What you need to do is edit the lookup file to add content appropriate to your deployment.
- **Pre-processing**: From the **Pipeline** drop-down, select `PACK: cribl-syslog-input (Syslog Preprocessing)`.

Create a Source for Each Device Class

As explained above, you should now create a Syslog Source for each vendor/class of syslog sender. Create each Syslog Source as follows:

In the QuickConnect UI: Click **Add Source**. From the resulting drawer's tiles, select [**Push** >] **Syslog**. Click **Add New**.

If you use QuickConnect, remember that each Source/Destination pair will be parallel and independent.

Or, in the **Data Routes** UI: From the top nav of a Cribl Stream instance or Group, select **Data** > **Sources**. From the resulting page's tiles or the **Sources** left nav, select [**Push** >] **Syslog**. Click **New Source** to open a **New Source** modal.

Configure the fields and options as follows:

General Settings

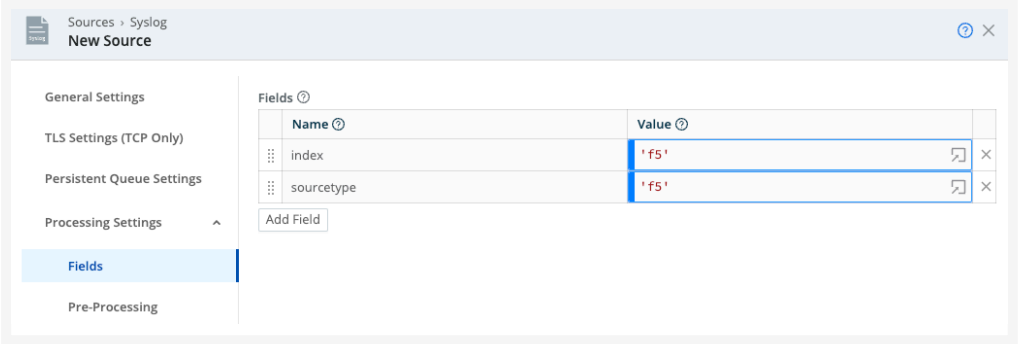
- **Input ID**: This field specifies the name for the Source, which will appear in the `__inputId` field as well as on Monitoring pages that show Source information. Common examples include `in_syslog_cisco_switch` for Cisco switches, `in_syslog_f5` for F5 load balancers, and so on.
- **UDP Port** and **TCP Port**: Enter the dedicated port you have chosen for the device class, and use UDP or TCP according to the recommendations above.

TLS Settings (TCP Only)

- Enabling TLS is strongly recommended if the Source will be receiving data across the Internet.

Processing Settings

- **Metadata**: Select **Fields (Metadata)** and add the fields appropriate for the intended Destination(s), such as `sourcetype`, `index`, `__timezone`, and any other meta-information you want to tie to the sender's hostname or IP address.



Metadata fields tied to a dedicated-port Syslog source

- **Pre-processing**: From the **Pipeline** drop-down, select `PACK: cribl-syslog-input (Syslog Preprocessing)`.

Adding Processing Pipelines

Congratulations! You've gotten a nice grounding in syslog processing, and you've seen methods with which you can:

- Properly architect the deployment environment, where syslog sender data travels through a load balancer, which then distributes it across a Cribl Stream Worker Group.
- Import a Pack from the Cribl Pack Dispensary.
- Create new Cribl Stream Syslog Sources which use the **Cribl Pack for Syslog Input**.
- Hard-code meta-information for specific ports, or use the lookup file to map meta-information for specific hosts.

Your logical next step: using Pipelines and Routes, transform syslog data in a way that makes sense for your particular syslog sender(s).

We'll look at two use cases:

- QuickConnect a Syslog Sender to a Source
- Route Multiple Syslog Senders to a Source

The point is to see how Cribl Stream offers different approaches for different scenarios. For configuring one dedicated Source to receive a given, single dataset, QuickConnect is ideal. But to accommodate a mix of data arriving on the same Source and port, and needing to be divided into subsets, and processed differently by different Pipelines - that is where we need Routes.

Use Case A: QuickConnect a Syslog Sender to a Source

> This use case includes receiving input data on the `1517` port, which is not available on Cribl.Cloud. The scenario presented here will only work in an on-prem deployment.

Of the many possible examples, we'll focus on reducing the volume of VMware ESXi syslog data by dropping events of `debug` severity.

Data reduction will be significant, because `debug` severity events can make up 80-90% of syslog data sent from ESXi servers. Not only that: Reducing the data volume reduces the CPU load and storage used to process the data, and searches will respond faster. In a world where we're searching for needles in a haystack, dropping 80% of the hay makes everything better.

To set this up:

1. In your Worker Group, navigate to **Pipelines**, click **Add Pipeline**, then click **Add Pipeline**.
2. Name the new Pipeline `DropNoisySyslog` or something similar.
3. Click **Comment** and enter a good description of what you're doing. (This is an important best practice!)
4. Add a Drop Function with filter `severityName=='debug'`. (You could also use sampling, or dynamic sampling if you wanted to be fancy about this.)
5. Click **Save** and we're done.

Next, we need to direct VMware ESXi data to our Pipeline. We'll do this using QuickConnect, which is the fastest way to configure a new Source, tie it to a Destination, and assign pre-processing and processing Pipelines to that Source. (We could also do this with Routes; that simply requires a few more configuration steps.)

In the **QuickConnect** UI:

1. Click **Add Source**.
2. From the resulting drawer's tiles, select [**Push** >] **Syslog**.
3. Click **Add New**.
4. Configure the fields and options as follows:
    - **Name**: `in_syslog_ESXi`.
    - **TCP**: `1517`. This is the TCP port on which ESXi servers send their logs.
    - **Pre-Processing** > **Pack**: Select `cribl-syslog-input`.
    - **Fields (metadata)**: Add a `sourcetype` field with value `esxilogs`, and an `index` field with value `vmware-esxi`.

Adding metadata appropriate for ESXi

5. Click **Save**.
6. On the main **QuickConnect** page, drag a connector from `in_syslog_ESXi` to each Destination you want to receive these events. When prompted, select **Pipeline**, select your `DropNoisySyslog`, and click **Save**.
7. Commit and deploy, and you're all set!

Use Case B: Route Multiple Syslog Senders to a Source

Let's imagine you have incoming data from a router. This data is tagged with `sourcetype=myrouter`, and it includes a mix of DHCP actions, login/authentication attempts, and firewall traffic logs.

Our goal is to send each of the three data subsets of data - DHCP, login, and firewall - to its own Pipeline.

We know that the **Cribl Pack for Syslog Input** has a Lookup Function that - for data arriving on the default port - should return meta-information such as `sourcetype` or `index`. We can combine this metadata with some matching of strings within the data, in Route filters that direct the right data to the right Pipeline.

Although in reality, the `myrouter` firewall does not exist, the example that follows shows the art of the possible. We'll spell out the procedures as if the scenario were real. Perhaps one day you'll use it as a template for an actual deployment.

Create a Pipeline for Each Data Subset

- Create three Pipelines, naming each one for a data subset: `myrouter-dhcp`, `myrouter-auth`, and `myrouter-fw-traffic`.
- Leave the Pipelines empty for now; we'll add Functions later on.

Create a Route for Each Data Subset

1. Navigate to **Routing** > **Data Routes**.
2. Click **Add Route**.
3. Configure a Route as specified in the table below:

Name	Filter	Pipeline
`myrouter-dhcp`	`sourcetype=='myrouter' && raw.match('dhcpd3:')`	`myrouter-dhcp`
`myrouter-auth`	`sourcetype=='myrouter' && raw.match('login')`	`myrouter-auth`
`myrouter-fw-traffic`	`sourcetype=='myrouter' && raw.match('TRAFFIC')`	`myrouter-fw-traffic`
`myrouter-other`	`sourcetype=='myrouter'`	`DropNoisySyslog`

4. Repeat the preceding steps until you have configured four new Routes - one for each of the three data subsets, plus a final Route to drop events that don't match our Filters (i.e., noise).

For all Routes, toggle **Final** on and set the output to whatever Destination you think is appropriate for the given data subset.

5. Click the `myrouter-dhcp` Route's ••• (Options) menu, then select **Group Actions** > **Create Group**.
6. Name new Group `myrouter`.
7. Drag the Routes you created above into the new Group. (Be sure that `myrouter-other` is listed last.)
8. Drag the new Group to whichever vertical position makes the most sense for your deployment.

Once you've completed all the above steps, then after collapsing your new Routes, you should see something like this:

8.8.4.2. Syslog TLS To Cribl.Cloud (Palo Alto Example)

This guide explains how to securely forward syslog data to your Cribl.Cloud instance using Transport Layer Security (TLS). Properly configured TLS encryption ensures that sensitive log data is protected during transmission across networks.

Benefits of TLS-Encrypted Syslog

- **Data Security**: Encrypts sensitive log data during transmission.
- **Integrity Verification**: Ensures data hasn't been tampered with during transit.
- **Compliance**: Helps meet security requirements for regulated industries.
- **Authentication**: Verifies the identity of your Cribl.Cloud instance.

General Configuration Steps

Follow these steps to set up your connection.

Determine Your Cribl.Cloud Ingest Address

Before configuring your syslog sender, you'll need to identify your Cribl.Cloud ingest address:

1. Log into your Cribl.Cloud portal.
2. From the top nav, select **Workspace**.
3. In the sidebar, select **Access**.
4. Note the **Public Ingress address** (for example, `default.main.<organizationId>.cribl.cloud`).

Obtain Cribl.Cloud CA Certificates

Cribl.Cloud uses Google Trust Services (GTS) certificates to secure connections. You have the following two options.

Use Trusted Public Certificate Authorities (Recommended)

Most systems already trust Google Trust Services certificates, so no additional certificate configuration may be necessary if your system's trust store is up to date.

Manually Import Google Trust Services Certificates

Google Trust Services publishes the necessary certificates in a user-friendly repository. Follow these steps to download the required certificates:

1. Visit the Google Trust Services Root Repository.
2. Download the following certificates:
    - `GTS Root R1` (PEM format): This is the root certificate for Google Trust Services.
        - **Fingerprint**: `d9:47:43:2a:bd:e7:b7:fa:90:fc:2e:6b:59:10:1b:12:80:e0:e1:c7:e4:e4:0f:a3:c6`
    - `WR1` (PEM format): This is the intermediate certificate issued by Google Trust Services.
        - **Fingerprint**:: `b1:0b:6f:00:e6:09:50:9e:87:00:f6:d3:46:87:a2:bf:ce:38:ea:05:a8:fd:f1:cd:c4`

You only need to download these certificates once.

Next, import these certificates into your syslog sender's trust store.

Configure Your Syslog Sender

The following are general steps for configuring a syslog sender (specific steps vary by vendor):

1. Configure a new syslog server profile/destination.
2. Set the protocol to `TLS/SSL` (not TCP).
3. Use port `6514` for TLS-encrypted syslog.
4. Set the destination address to your Cribl.Cloud ingest address.
5. Configure the appropriate log format (typically BSD syslog).
6. Enable certificate validation using your system's trust store or imported certificates.

Vendor-Specific Configuration: Palo Alto Example

Panorama management appliances give you insights into what is happening behind your Palo Alto Networks firewalls, including details about applications, URLs, threats, data files, and patterns. By forwarding logging data directly from Palo Alto firewalls or Panorama management appliances to your Cribl.Cloud instance, you can shape, process, search, and route the data to the analytics platform of your choice.

This section outlines how to configure your Palo Alto Networks firewalls or Panorama management appliances to connect securely to your Cribl.Cloud instance using Google Trust Services (GTS) as the Certificate Authority (CA).

Import Trusted Roots into PAN

Download the GTS certificate certificate as outlined above. Next, import the certificates into PAN. In the PAN OS:

1. Go to **Device** > **Certificate Management** > **Certificates**.
2. Select the **Import** button at the bottom.
3. The **Import Certificate** configuration modal appears. Import both downloaded certificates (`GTS Root R1` and `WR1`).
4. Once you have imported both certificates, go to the **Device Certificates** table and select the blue link for each certificate you just imported.
5. Select the **Trusted Root CA** check box for each certificate.

Configure the Syslog Server Profile

Now that you've imported the trusted certificates, proceed with creating a new Syslog Server Profile:

1. After importing the certs, navigate to **Device** > **Server Profiles** > **Syslog**.
2. Create a new Syslog Server Profile.

If you're using Syslog over TLS, make sure to set the profile to use SSL instead of TCP.

Even if a PAN rule does not explicitly reference the TLS certificate that you added from Cribl, TLS will detect the certificate information and use it correctly.

3. Configure the Syslog server field with your Cribl.Cloud Organization's ingest address. To find your ingest address, navigate to your Cribl.Cloud portal and select the gear on the top bar.
4. Select **Network Settings** and choose the Worker Group you're sending data to. For example, sending to the default Group your ingest address would look like: `default.main.<organizationId>.cribl.cloud.`

Configure Log Settings to Use the Syslog Profile

Next, configure your log settings to use the Syslog Profile.

1. Go to **Device > Log Settings**.
2. Ensure the desired log types are configured to use the newly created Syslog Server Profile.

Configure Firewall Rule Log Forwarding

To selectively forward specific log types, create a Log Forwarding Profile and define the rules for which traffic should be forwarded.

1. Use the **Log Forwarding Profile** to create a ruleset and configure the appropriate log types to be forwarded.
2. Go to **Objects** > **Log Forwarding** and configure each relevant section with the appropriate settings.

Forward Admin Activity Logs

For additional context, you can forward your admin activity logs:

1. Go to **Device** > **Setup** > **Management**, and select the gear in the **Logging and Reporting** settings.
2. Enable the **Debug and Operational Commands** and **UI Actions**, and choose the appropriate Syslog server.

Customize OCSP Validation Settings for the `syslog-ng` Process

If the certificate validation fails, follow these steps to customize the validation settings for the `syslog-ng` process. This requires shell access to your device.

These commands are for an individual PAN OS device and might differ for Panorama. Refer to the official PAN-OS documentation for specific instructions on your version.

```
set syslogng ssl-conn-validation all-conns skip
set syslogng ssl-conn-validation explicit OCSP skip CRL skip EKU skip
```

Copy

Validation

To validate whether you have successfully set up the TLS-encrypted Syslog, check the logs on your PAN OS.

Troubleshooting

If you run into issues not anticipated above, you can use some of the following commands to further diagnose them:

```
# restart the syslog-ng process
debug syslog-ng restart

# see syslog-ng process logs
tail follow yes mp-log syslog-ng.log

# get stats for processes
debug log-receiver statistics
debug syslog-ng stats

# tcpdump activity on the syslog port and view the results
# note: control-c to exit the tcpdump, it will not show anything on the cli when runr
```

8.8.5. Configure Upstream Logging Agents

This page explains how to quickly connect a wide selection of common logging agents and other log sources to Cribl Stream. The examples below are equally valid for Cribl.Cloud and on-prem Cribl Stream instances.

Whichever source you choose, start by doing a live capture on the corresponding Source in Cribl Stream. Verify that data is coming in. Then save your capture to a sample file, which will help you build your Pipelines.

Fluent Bit and Fluentd

Both Fluent Bit (written in C) and fluentd (written in Ruby) are open-source log collectors, processors, and aggregators.

Fluent Bit to HEC

Of Fluent Bit's many output options, several work with Cribl Stream. Here's how to connect a Cribl Stream Splunk HEC Source to Fluent Bit with the Splunk HEC formatting option.

You'll need to copy and paste the following:

- From the Cribl.Cloud sidebar, select **Workspace** > **Data Sources** and copy the `in_splunk_hec` ingest address.
- From a Cribl Stream instance or Worker Group, select **Data** > **Sources** > **Splunk HEC** > `<source_name>` > **Auth Tokens**, and copy the HEC token.

Decide how to adjust the `match` parameter for your use case. For example, an asterisk (wildcard) value will send **all** events to Cribl Stream.

Edit the Fluent Bit settings accordingly. They're usually in `/etc/td-agent-bit/td-agent-bit.conf`, in a form similar to this:

```
[OUTPUT]
    name         splunk
    match        *
    host         in.main-default-<organization>.cribl.cloud
    port         8088
    splunk_token <HEC_token>
    tls          on
    # optional
```

Copy

Save the changes and restart the `td-agent-bit` service.

Fluent Bit to Elastic Sources

You can also connect Cribl Stream to Fluent Bit, using an Elastic Source.

```
[OUTPUT]
    name          es
    match         *
    host          10.0.21.134
    port          9200
    index         my_index
    type          my_type
```

Copy

Fluentd to HEC

If you don't already have it, install the `splunk_hec` fluentd mod:

```
sudo gem install fluent-plugin-splunk-hec
```

You'll need to copy and paste the following:

- From the Cribl.Cloud sidebar, select **Workspace** > **Data Sources** and copy the `in_splunk_hec` ingest address.
- From a Cribl Stream instance or Worker Group, select **Data** > **Sources** > **Splunk HEC** > `<source_name>` > **Auth Tokens**, and copy the HEC token.

Decide how to adjust the `match` parameter for your use case. Use an asterisk (wildcard) if you want to send **all** events to Cribl Stream.

Edit the `<match` section of the fluentd settings accordingly. They could be in `/etc/fluent/fluent.conf` or another location, as described in the fluentd docs.

```
<match **>
  @type splunk_hec
  @log_level info
  hec_host in.main-default-<organization>.cribl.cloud
  hec_port 8088
  hec_token <HEC_token>
  index <index_name>
  source_key <file_path>
  <format>
    @type json
  </format>
</match>
```

Copy

Save the changes and restart fluentd.

Splunk

Cribl Stream can directly ingest the native Splunk2Splunk (S2S) protocol. As a result, both Splunk universal and heavy forwarders can send log data to Cribl Stream.

Splunk Forwarder (Universal or Heavy) to Splunk TCP

See how to configure a Splunk Forwarder in the Splunk TCP Source document.

Elastic

In the Elastic ecosystem, Elastic Filebeat is an agent that functions as a lightweight shipper for forwarding and centralizing log data.

Elastic Filebeat

Cribl Stream can ingest the native Elasticsearch streaming protocol directly.

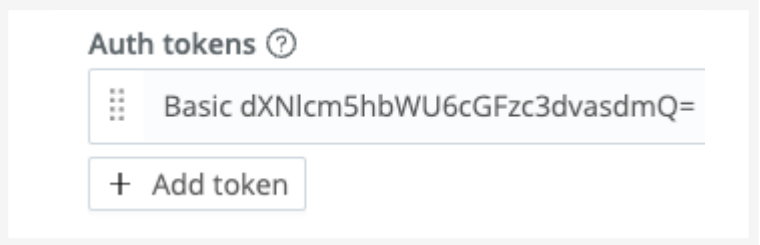
Cribl recommends setting an authorization token so that your receiver will accept only your traffic. This setting is in the **Auth Tokens** field of Cribl Stream's **Sources** > **Elasticsearch API** > **General Settings** tab.

If you want to enable Basic authentication:

- Concatenate your username and password with a colon in between, like this: `username:password`.
- Encode the `username:password` string using base64.

- Prepend the string `Basic ` (including the trailing space) to the encoded string.
- Enter the resulting string in the **Auth tokens** field as described above.

You'll see something like this:



Basic auth in the Cribl Stream Elasticsearch API Source

```
output.elasticsearch:
  hosts: ["in.main-default-<organization>.cribl.cloud:9200/search"]
  protocol: "https"
  ssl.verification_mode: "full"
  username: <some_username>
  password: <some_password>
```

Copy

syslog

To receive syslog data in Cribl Stream, create a native Syslog Source. See Cribl's best practices for working with syslog data.

> Cribl recommends not using UDP over the public internet. UDP examples are included here for test purposes only.
>
> UDP, by definition, does not guarantee delivery. While TCP mitigates this problem, using either UDP or TCP without TLS over the public internet exposes unencrypted data.

syslog-ng (TLS over TCP)

Cribl recommends using TLS over TCP to send data from syslog-ng to Cribl Stream.

Here's an example of `syslog-ng.conf` edited such that all event data will be encrypted on the way to Cribl.Cloud. Adapt the snippet to your use case and edit `syslog-ng.conf` accordingly.

```
destination d_syslog {
    syslog("in.main-default-<organization>.cribl.cloud"
        transport("tls")
        port(6514)
        tls(
            peer-verify(required-trusted)
            ca-dir("/etc/syslog-ng/ca.d")
        )
    );
};

log {
    source(s_network);
    destination(d_syslog);
};
```

Copy

Save the changes and restart the syslog-ng service.

syslog-ng (UDP or TCP)

Here's an example of `syslog-ng.conf` - for test purposes only, because it does not use TLS.

Adapt the snippet to your use case, and edit `syslog-ng.conf` accordingly. You can change `tcp` to `udp` to switch protocols.

```
destination d_syslog {
    syslog("in.main-default-<organization>.cribl.cloud"
        transport("tcp")
        port(9514)
    );
};

log {
    source(s_network);
    destination(d_syslog);
};
```

Copy

Save the changes and restart the syslog-ng service.

syslog-ng with Load-Balanced Outputs

You can forward logs from syslog-ng to multiple Cribl Stream Workers through a redundant, load-balanced output by using the syslog-ng `elasticsearch-http` output module. With this arrangement, syslog-ng receives syslog messages over TCP and/or UDP and outputs events over the Elasticsearch HTTP Bulk API to Cribl Stream Workers.

> Forwarding to Cribl Stream from the syslog-ng `elasticsearch-http` output module requires syslog-ng version 3.19 or higher.
>
> To check your syslog-ng version, run the following command:

```
syslog-ng -V
```

Copy

Adapt the snippet to your use case, and edit `syslog-ng.conf` accordingly. To learn more about syslog-ng destination flags, see the syslog-ng Administration Guide.

```
@version: 3.33
@include "scl.conf"

# Inputs
source s_network {
    # flags(no-parse) disables syslog-ng parsing of syslog messages (raw passthru)
    network(transport("tcp") port("9514") flags(no-parse));
    network(transport("udp") port("9514") flags(no-parse));
};

destination d_elasticsearch_tls {
    # https://www.syslog-ng.com/technical-documents/doc/syslog-ng-open-source-editior
    elasticsearch-http(url("https://worker1:9200/_bulk" "https://worker2:9200/_bulk"
        batch-lines(100)
        batch-bytes(512Kb)
        batch-timeout(2500)
        persist-name("d_elasticsearch-http-load-balance")
        type("")
        index("syslog")
        # Auth token
        #headers("Authorization: <token from Stream>")
        # or Basic Auth
        #user("username")
        #password("password")
    );
};

log {
    source(s_network);
    destination(d_elasticsearch_tls);
};
```

Copy

Save the changes and restart the syslog-ng service.

rsyslog (TLS over TCP)

You'll need to install the `rsyslog-gnutls` package if it's not already on your system. E.g.:

```
apt install rsyslog-gnutls
```

If the `rsyslog-gnutls package` is not already on your system, run the following command (or equivalent) to install it:

```
apt install rsyslog-gnutls
```

Cribl recommends using TLS over TCP to send data from rsyslog to Cribl Stream.

Here's an example of `rsyslog.conf`, edited such that all event data will be encrypted on the way to Cribl.Cloud. Adapt the snippet to your use case, and edit `rsyslog.conf` accordingly:

```
# path to the crt file.
# You will need a valid ca cert file. Most linux distros come with this
$DefaultNetstreamDriverCAFile /etc/ssl/certs/ca-certificates.crt

*.*     action(
         type="omfwd"
         target="in.main-default-<organization>.cribl.cloud"
         port="6514"
         protocol="tcp"
         action.resumeRetryCount="100"
         queue.type="linkedList"
         queue.size="10000"
         StreamDriver="gtls"
         StreamDriverMode="1" # run driver in TLS-only mode
)
```

Copy

Save the changes and restart the rsyslog service.

rsyslog (UDP or TCP)

Here's an example of `rsyslog.conf` - for test purposes only, because it does not use TLS.

Adapt the snippet to your use case, and edit `rsyslog.conf` accordingly. You can change `tcp` to `udp` to switch protocols.

```
*.*     action(
          type="omfwd"
          target="in.main-default-<organization>.cribl.cloud"
          port="9514"
          protocol="tcp"
          action.resumeRetryCount="100"
          queue.type="linkedList"
          queue.size="10000"
)
```

Copy

Save the changes and restart the rsyslog service.

Appliances that Send syslog

Many appliances emit syslog data. At a minimum, configure your appliance(s) to send syslog over TCP with TLS enabled, if possible.

Even better, stand up a Cribl Stream instance close to the appliances in your network topology. Have the appliances send syslog to the nearby Cribl Stream, which can optionally transform, filter, and/or enhance the data, and then send it to a more centralized Cribl Stream cluster in Cribl.Cloud or an one-prem location.

The Cribl Stream worker that's closest to the appliance will use TCP JSON, with TLS enabled, to send syslog to the second Cribl Stream instance:

Sending syslog from an appliance to Cribl Cloud

The benefits of placing the first Cribl Stream instance close to the log producer include:

- Since there are fewer network hops, UDP becomes a more viable option, if you want it.
- If you're using TCP, that can be more performant here than it would be across the internet.
- The first Cribl Stream instance can do data reduction and/or compression. These operations reduce the volume of the data hitting the wire on the way to the next hop.
- This can help keep data egress costs under control.

Vector

Vector is Datadog's tool for building observability pipelines.

Vector to HEC

Vector supports Splunk's HTTP Event Collector (HEC). Here's how to connect a Cribl Stream Splunk HEC Source to Vector.

For Cribl.Cloud especially, Cribl recommends setting an authorization token so that your receiver will accept only your traffic. This setting is in Cribl Stream's **Sources** > **Splunk HEC** > **Auth Tokens** tab.

The snippet below references a random syslog generator called `dummy_logs` for testing purposes. Adapt the snippet to your use case and edit `vector.toml` (or `vector.yaml` or `vector.json`) accordingly.

```
[sinks.my_sink_id]
type           = "splunk_hec"
inputs         = [ "dummy_logs" ]
endpoint       = "https://in.main-default-<organization>.cribl.cloud:8088"
compression    = "gzip"
token          = "<optional-but-recommended>"
encoding.codec = "json"
```

Copy

Vector to Elasticsearch

Vector supports Elasticsearch API. Here's how to connect a Cribl Stream Elasticsearch API Source to Vector.

The snippet below references a random syslog generator called `dummy_logs` for testing purposes. The snippet also specifies Basic authentication. Follow the procedure for enabling Basic authentication described in the Elastic Filebeat example above.

Adapt the snippet to your use case and edit `vector.toml` (or `vector.yaml` or `vector.json`) accordingly.

```
[sinks.elastic_test]
type          = "elasticsearch"
inputs        = [ "dummy_logs" ]
endpoint      = "https://in.main-default-<organization>.cribl.cloud:9200/search"
compression   = "gzip"
index         = "my_es_index"
auth.user     = "<some_username>"
auth.password = "<some_password>"
auth.strategy = "basic"
```

Copy

Pull-based Sources

All the above data sources **push** data into Cribl Stream. Here are brief notes on configuring some popular Cribl Stream-native Pull Sources:

Amazon Kinesis

You can create an Amazon Kinesis Source in Cribl Stream. This is straightforward for on-prem Cribl Stream instances.

8.8.6. BigPanda/Webhook Integration

You can configure Cribl Stream to send Webhook notifications to the BigPanda IT Ops platform. These notifications arrive in BigPanda as Alerts, which BigPanda correlates into Incidents.

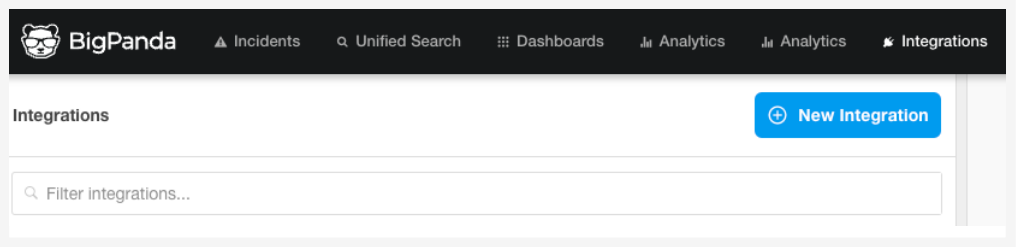
Before you begin, you should have an Admin account on a BigPanda Cloud instance.

To supplement this guide, reference the Cribl Integration document from BigPanda.

Prepare BigPanda to Receive Data from Cribl Stream

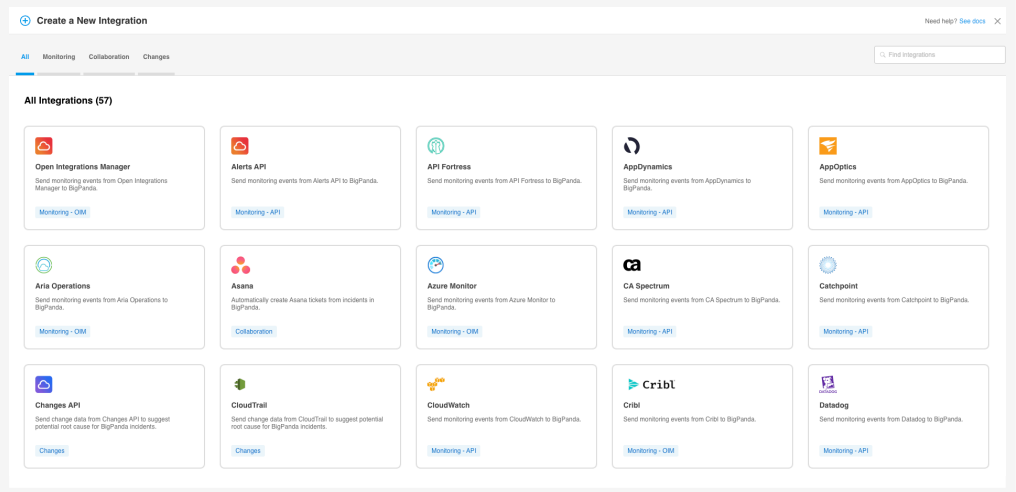
The BigPanda **App Key** and **Access Token** are separate and independent. The **Access Token** is a 32-character string that is part of the value that BigPanda generates for the `Authorization` HTTP header. (It functions like an auth token or bearer token.)

1. Log into your BigPanda Cloud instance as an Admin.
2. In the **Integrations** tab, select **New Integration**.



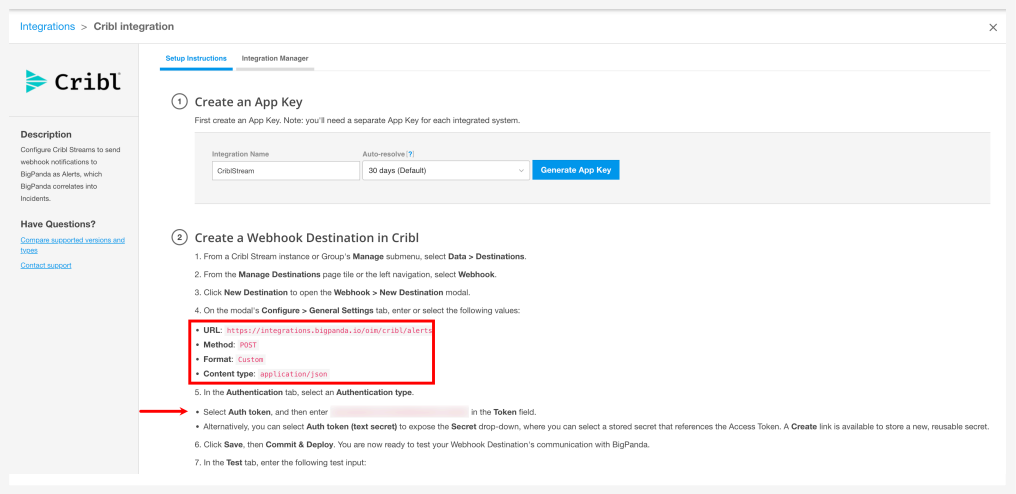
The **Integrations** tab

3. In the **Create a New Integration** modal, select the Cribl tile.



The **Create a New Integration** modal

- This opens the **Cribl Integration** page.



The **Cribl Integration** page

4. In the **Create an App Key** section, generate an App Key named `Cribl Stream`. You'll need the App Key when configuring Cribl Stream in the next section. Cribl Stream will insert the App Key into every event it sends to BigPanda.
5. BigPanda will generate a page containing setup instructions for the Cribl Stream Webhook Destination. Store the following information:
    - **URL**: `https://integrations.bigpanda.io/oim/cribl/alerts`.
    - **Method**: `POST`.
    - **Format**: `Custom`.
    - **Content Type**: `application/json`.
    - **Authentication token**: A 32-character `<auth-token>`.

The BigPanda page will also contain test code. You'll need the preceding information and the test code when configuring Cribl Stream in the next section.

Configure the Webhook Destination in Cribl Stream

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Destinations**. Then select **Webhook**. Next, select **Add Destination**.
2. On the modal's **Configure** > **General Settings** tab, enter or select the following values:
    - **URL**: Enter the Alerts API endpoint URL, for example: `https://integrations.bigpanda.io/oim/cribl/alerts`.
    - **Method**: `POST`.

- **Format**: `Custom`.
- **Content type**: `application/json`.

3. In the **Authentication** tab, select an **Authentication type**.
   - You can select **Auth token** and provide the 32-character **Authentication token** you got from BigPanda after creating the App Key in the **Token** field.
   - Alternatively, select **Auth token (text secret)** to expose the **Secret** drop-down, in which you can select a stored secret that references the Access Token. A **Create** link is available to store a new, reusable secret.
4. Select **Save**, then **Commit & Deploy**. You are now ready to test your Webhook Destination's communication with Big Panda.
5. In the **Test** tab, enter the test code from the BigPanda setup instructions. It will look like this:

```
[
        {
                "app_key": "<your_app_key>",
                "status": "critical",
                "host": "production-database-1",
                "timestamp": 1402302570,
                "check": "CPU overloaded",
                "description": "CPU is above upper limit (70%)",
                "cluster": "production-databases",
                "my_unique_attribute": "myUniqueValue987654321"
        }
]
```

Copy

6. Select **Run Test**.

This should send an alert to BigPanda.

BigPanda Alerts API Requirements

HTTP payloads sent to the BigPanda Alerts API must satisfy rules that are beyond the scope of this topic. For details, see the BigPanda documentation about Parameters and Troubleshooting an Integration.

However, at at minimum, three fields are required:

1. `app_key`.
2. `status`.

3. `host` OR `service` OR `application` OR `device`.

Thus, the test input shown above works even if you omit all but the first three fields.

There are other possibilities for the third field, but they require understanding how BigPanda determines the `primary_property` of an Alert, plus some additional BigPanda configuration. See the BigPanda links above for details.

BigPanda Alert Deduplication

BigPanda processes an alert's Primary and Secondary properties on ingestion. These properties default to `host` and `check`.

When an alert has the same Primary and Secondary properties as those received in previous alerts, the Incident Console shows only the timestamp from the initial event.

For more information on this functionality, see the BigPanda pages Deduplication and Incident Identifier.

Verify that BigPanda is Receiving Notifications and Events

In the BigPanda **Incidents** tab, you should see an Incident whose Source is `Cribl Stream`. The details of the test input you sent from the Webhook Destination should appear in an Alert within that Incident. If so: It works!

8.8.7. Gigamon To Cribl Stream

Gigamon and Cribl have integrated the Gigamon GigaVUE Cloud Suite with Cribl Stream to optimize telemetry intelligence delivery across hybrid cloud infrastructures. This integration enables seamless data formatting and routing, enhancing the value of existing tools by providing comprehensive observability and security.

Gigamon's Deep Observability Pipeline, with GigaVUE Cloud Suite, delivers deep visibility into hybrid cloud traffic, improving security and performance by addressing East-West application traffic blind spots. Cribl's data management solution enhances threat detection and incident response through seamless access, enrichment, and routing of telemetry data from multiple sources.

In this integration, data is sent from the Gigamomn Application Metadata Exporter to a Cribl Stream HTTP Raw Source.

Prerequisites

- Your Gigamon GigaVUE Cloud Suite has the AMI module enabled.
- You've enabled the appropriate Application Metadata Intelligence attributes.
- You know your Cribl Stream Worker Group ingress address, shown on the **Overview** page under **Group Information**.

Configuring Cribl Stream

1. In your Cribl Stream account, select the Worker Group that will receive the data.
2. Select **Processing**, then **Packs**. Add the **Gigamon** Pack from the Dispensary to your Worker Group.
3. Select **Routing**, then **QuickConnect**. Select **Add Source** and add a new **Raw HTTP** Source.
4. Configure the new Raw HTTP Source as follows:
    - **General Settings**:
        - **Input ID**: Enter a unique name to identify this Raw HTTP Source definition. Such as `Raw HTTP`.
        - **Address**: Enter the address to bind on. Use `0.0.0.0` to bind on all addresses.
        - **Port**: Enter `20001`. This port number is used in the Cribl Stream Worker Group ingress address you'll provide to Gigamon.
    - **Authentication**:
        - Select **Add token** and **Generate** an **Auth token**. You'll provide this token to Gigamon when configuring your **Monitoring Session**.

5. Click **Save**.
6. Use QuickConnect to connect your new Raw HTTP Source to Cribl Destinations. Be sure to use the Gigamon Pack for the connection configuration.
7. **Commit & Deploy** your changes.

Configuring Gigamon to Export Data to Cribl Stream:

1. In your Gigamon instance, navigate to the **Traffic** section from the FM instance.
2. Select the location of your deployment, like `AWS`.
3. Choose and edit your **Monitoring Instance**.
4. Select the **Details** for your AMX instance.
5. Create a new Cloud Tool Export as follows:
    - **Alias**: `Cribl`
    - **Cloud Tool**: `Other`
    - **Endpoint**: Provide your Cribl Stream Worker Group ingress address with port `20001`. For example, `https://workergroup.main.orgname.cribl.cloud:20001`.
    - **Headers**:
        - **Authorization**: Provide the **Auth token** you generated when configuring the Cribl Stream **Raw HTTP Source**.
        - **Content-Type**: `application/json`
    - **Enable Export**: Check the box to enable.
    - **Format**: `JSON`
6. Click **Save** and **Deploy** the changes.

Verifying Data Flow

There are many ways to monitor the status of your Source. Navigate to the Source and use the available tabs.

- Use the **Live Data** tab to capture metadata received by Cribl Stream.
- The **Charts** tab displays throughput details.
- The **Status** tab provides details about the Workers in the group and their status.

You can also monitor Source throughput on the Monitoring page.

8.8.8. System Metrics To Grafana

Cribl Stream can collect metrics from the host on which it is running, and can populate some standard metrics dashboards right out of the box. For metrics details, see Linux System Metrics Details. While, Grafana is a multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources.

Cribl has configured a Prometheus dashboard to display Cribl Stream System Metrics, and shared the dashboard in the Grafana library, which is public. This is a relatively simple dashboard suitable for showing aggregate metrics.

Another useful dashboard is the one that's commonly used with Prometheus and their Node Exporter agent, found here. This dashboard can handle the highly detailed metrics.

In this guide, we will show you how to route system metrics to a Grafana dashboard for both basic and highly detailed metrics visualizations.

We will walk you through the following:

- Configure System Metrics Source on Cribl Stream to receive metrics from the host.
- Configure the Prometheus Destination to send data to Grafana Cloud.
- Access Grafana Cloud to see the metrics and visualizations.

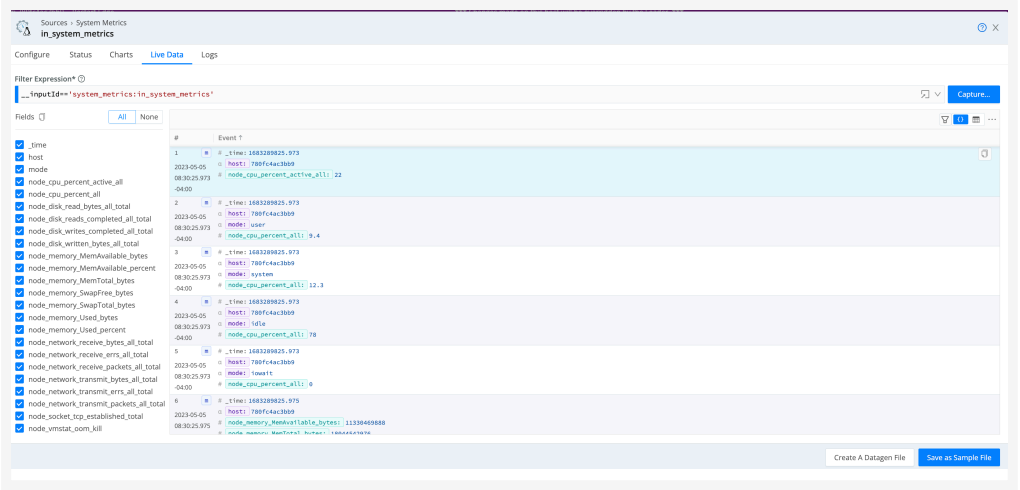
Configure the System Metrics Source on Cribl Stream

In Cribl Stream, start by configuring and enabling a System Metrics Source. The key requirement here is to to decide on the level of details for the metrics you want to collect. In the **Host Metrics** and **Container Metrics** tabs:

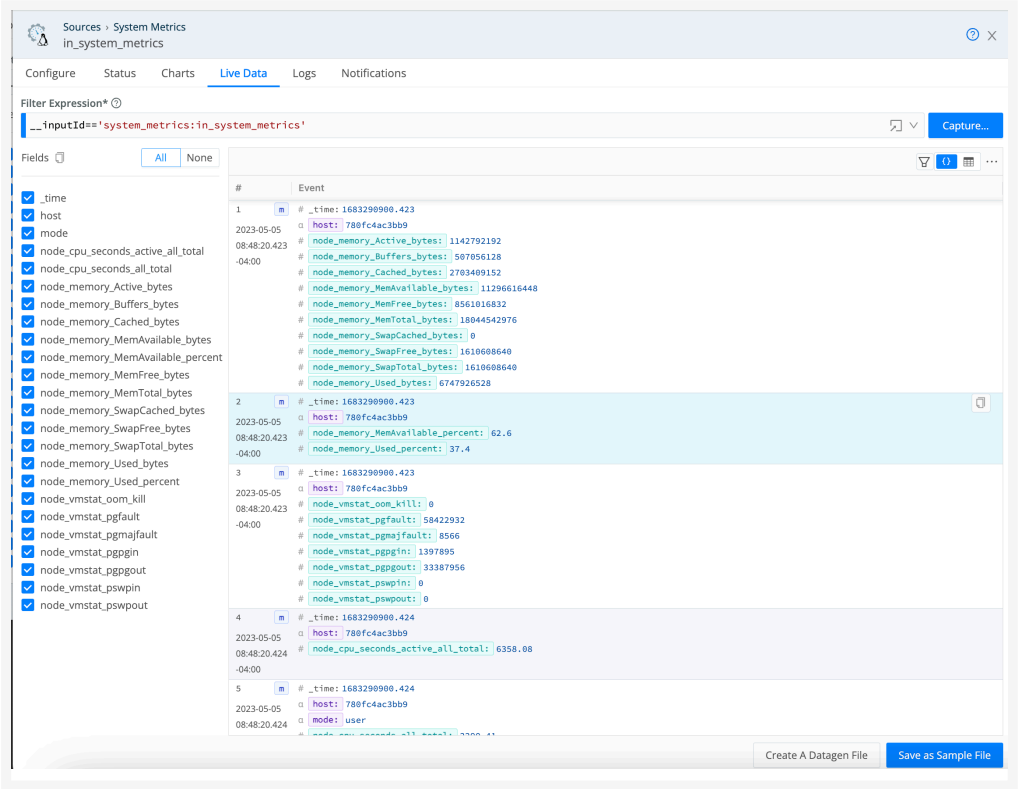
- For the relatively simple dashboard showing aggregate metrics, choose the **Basic** mode for most of your metrics.
- For the highly detailed metrics visualizations dashboard, choose the **All** mode for all of your metrics.

In the **Pre-Processing** tab, select the `prometheus_metrics` **Pipeline**.

When done, **Commit** and **Deploy** your changes. Before moving on to the next step, check the **Live Data** tab to make sure the metrics are generated.



Basis System Metrics

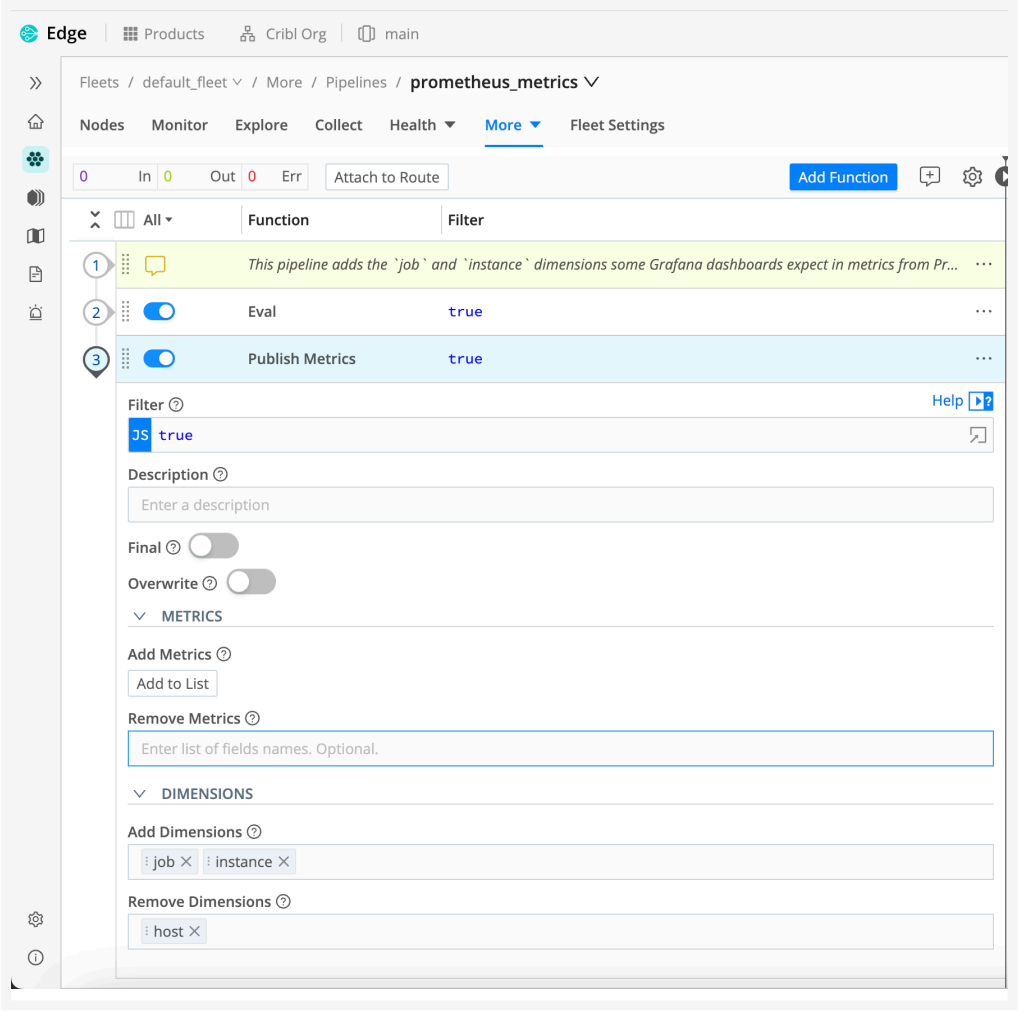


Detailed System Metrics

(Optional) Adjust the `prometheus_metrics` Pipeline

Optionally, you can adjust the `prometheus_metrics` Pipeline to include the `host` dimension or add/remove more dimensions.

Select a **Fleet** to configure. From the Fleet's submenu, select **More** > **Pipelines**.

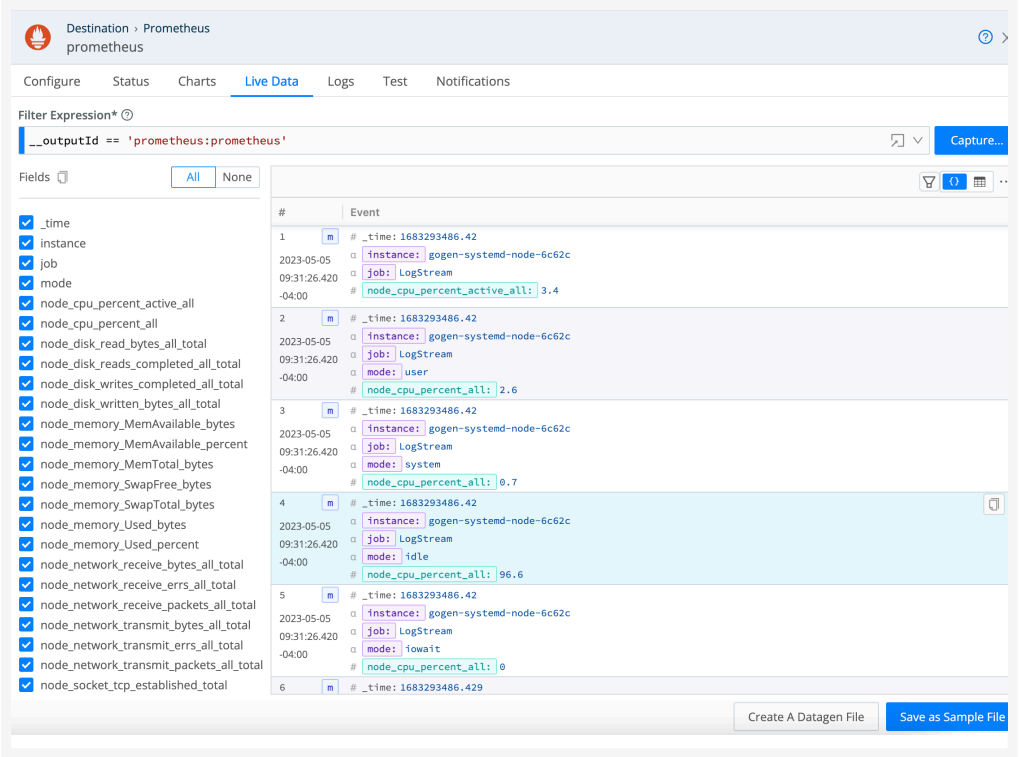


Adjust the Pipeline

Configure the Prometheus Destination

Next, we'll configure the Prometheus Destination to send data to Grafana Cloud.

Next, preview your data on the modal's **Live Data** tab.

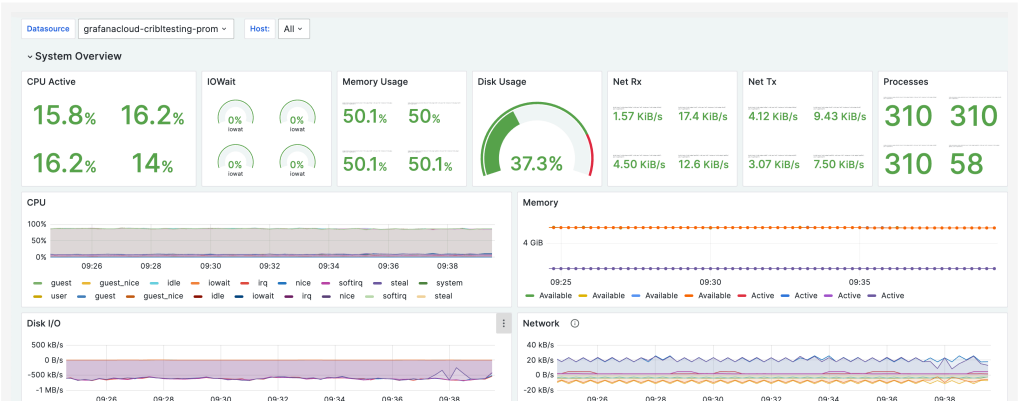


Preview Live Data

Visualize the Data in Grafana

Navigate to your Grafana Cloud instance, and select **Explore** on the left panel to review the node metrics. Selecting metrics and running the query will produce a chart and table of metric events that can be added to new or existing dashboards for analysis. If you are missing some expected metrics, you might need to adjust the variables in your Grafana Cloud instance. For details, see Variables.

Use the Grafana **Filter** option to browse the dashboards for your Prometheus data. Below are the Cribl dashboards with the basic and detailed metrics shown.



8.8.9. Kafka Authentication With Kerberos

This page describes how to configure a Kafka Source and Destination to use Kerberos authentication. It also documents the verification steps to test your setup.

Availability:

- Kerberos is available only on Linux, so you'll have to use a Linux host or Linux container to work through this procedure.
- Kerberos is not supported for Cribl-managed Cribl.Cloud Workers, but it is enabled for customer-managed (hybrid or on-prem) Worker Groups, whether the Leader is based in Cribl.Cloud or on-prem.

For additional information on Kafka configuration, see also our Kafka Source or Kafka Destination reference topics.

Set up the Cribl Stream Host

Once you have set up an instance of Kafka with Kerberos, perform the following steps on the same host where Cribl Stream is running:

1. Install `krb5-user` (or the equivalent for non-Ubuntu systems) on your Worker Node.
2. Ensure that you have a valid `/etc/krb5.conf` file. Consult your Kerberos admin if you don't have one. Kerberos authentication with Kafka specifically requires `rdns = false` in this config, but no other special settings. Cribl Stream supports the file and directory cache types.
3. Ensure that the keytab path in your configuration points to a valid keytab on each Worker Node.

Configure the Kafka Destination

Create a new Kafka Destination with the following settings:

General Settings

- **Brokers**: `broker.kerberos-demo.local:9092`
- **Topic**: `Cribl`

Authentication Settings

- **Enabled**: Toggle on.
- **SASL mechanism**: `GSSAPI/Kerberos`

- **Keytab location**: `/<path-to-client-keytab>/kafka-client.key`. This path to the valid keytab file must be available on all Worker Nodes.
- **Principal**: `kafka_producer@TEST.CONFLUENT.IO`
- **Broker service class**: `kafka`

Configure the Kafka Source

Create a new Kafka Source with the following settings:

General Settings

- **Brokers**: `broker.kerberos-demo.local:9092`
- **Topic**: `Cribl`
- **Group ID**: `Cribl`

Authentication Settings

- **Enabled**: Toggle on.
- **SASL mechanism**: `GSSAPI/Kerberos`
- **Keytab location**: `/<path-to-client-keytab>/kafka-client.key`
- **Principal**: `kafka_producer@TEST.CONFLUENT.IO`
- **Broker service class**: `kafka`

Verify the Setup

1. Open the **Live Data** tab and start a capture with **Capture Time (sec)** and **Capture Up to N Events** set to high values (for example, `1000`).
2. In a separate browser tab, navigate to the Destination modal's **Test** tab and select **Run Test** a few times.
3. Return to the Source's **Live Data** tab and verify that the Source is capturing events.

8.8.10. Moogsoft/Webhook Integration

Written for Cribl by Douglas Bothwell, formerly of Moogsoft.

You can configure Cribl Stream to send Webhook notifications to Moogsoft, using custom integrations. A custom integration is a user-defined Moogsoft endpoint that ingests JSON payloads, and converts them to Moogsoft events or metrics.

Note these options:

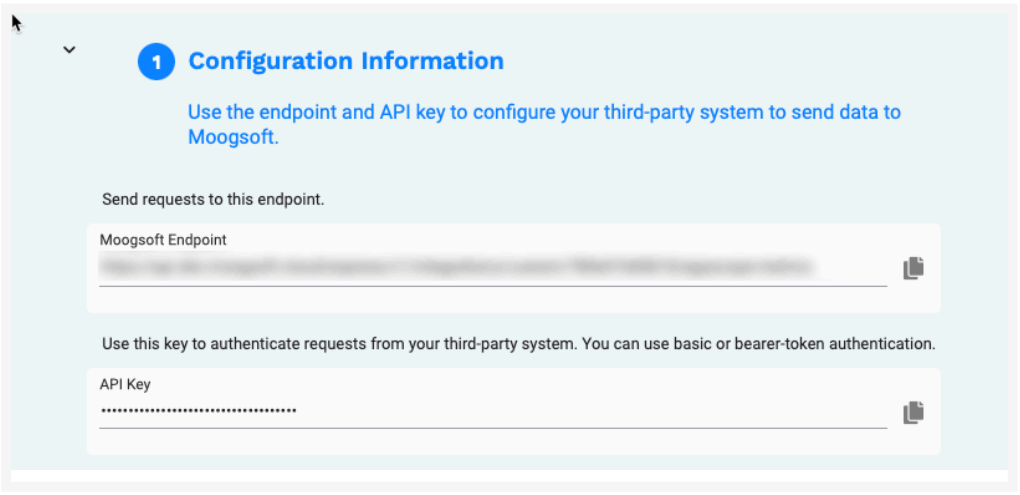
- A Moogsoft custom integration can ingest either events or metrics. Each custom integration has its own separate API key, and its own Cribl-to-Moogsoft mappings.
- There is no limit to the number of custom integrations you can create.
- The examples below illustrate simple event and metric mappings. You can define your own mappings for each endpoint, based on the data types and payloads you want to send. For details, see Moogsoft's Create Your Own Integration documentation.

Create the Moogsoft Custom Integration

Log into your Moogsoft SaaS instance as an Owner or Administrator, and then:

1. Choose **Data Config** > **Ingestion Services** > **Create Your Own Integration**.
2. Select **Add New Integration**.
3. Specify an integration endpoint and description.
4. Specify the data type to send to the endpoint: **Events** or **Metrics**.

The integration setup screen appears, with the URL and the API key for the new endpoint.



The Moogsoft integration setup screen

Configure the Webhook Destination

Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Data** > **Destinations**. Select **Webhook**. Next, select **Add Destination** to open the **Webhook** > **New Destination** modal, and configure the following.

General Settings

URL: The URL for the new custom integration (copy this from the Moogsoft UI).

Method: `POST`.

Format: `Custom`.

Content type: `application/json`.

Advanced Settings

Extra HTTP Headers: Select **Add Header** to define this extra header:

Name	Value
`apiKey`	The API key for the new custom integration (copy this from the Moogsoft UI).

Select **Save**, then **Commit** and **Deploy**. Now you are ready to map your events and/or metrics.

Mapping Cribl Data to Moogsoft

Your mappings will differ, depending on the data you want to send. However, the simple examples below, in which we map a Cribl Stream payload to a Moogsoft custom integration, illustrate principles that apply to all custom mappings:

- Moogsoft has a defined event schema and metric schema. Each schema includes a set of required fields. Your custom integration must include mappings for all required fields.
- You can define custom tags for Cribl Stream fields that do not have Moogsoft equivalents.
- You can also specify default values in case Cribl Stream sends an object with a missing field.
- The Moogsoft event schema includes a `severity` field. You can map Cribl Stream fields and values to Moogsoft severities, or define a default severity if a payload does not include this information.

Define and validate your mappings based on the type of data you plan to send to Moogsoft:

- Event Mapping
- Metric Mapping

Event Mapping

To illustrate how to map Cribl Stream payloads to Moogsoft events, we'll walk through an example with a payload of Cribl Stream syslog messages. You'll follow the same overall workflow for any events payload.

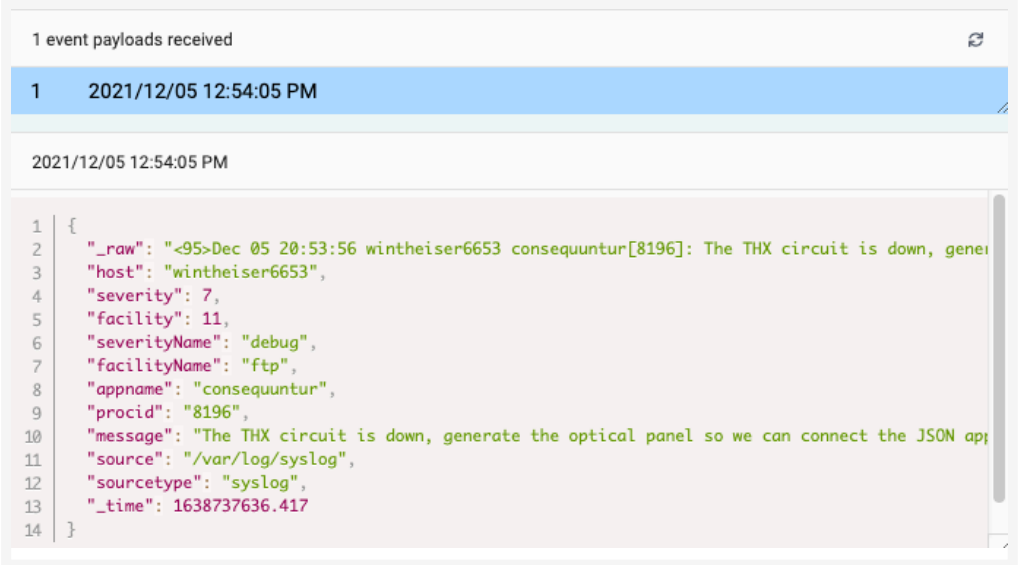
Send a Sample Payload to Moogsoft

In Cribl Stream, navigate to your Webhook Destination's configuration modal, select the **Test** tab, then:

1. In the **Test input** field, define one or more JSON payloads for the Cribl data you want to send. To map syslog events, set the **Select sample** drop-down to **syslog.log**.
2. Select **Test**.

Map the Event Fields

In Moogsoft, view the custom integration's config screen. Under **Map Your Data**, you should now see the payload you just sent.



The event payload appears in Moogsoft

Select the payload, and then define your Cribl-to-Moogsoft mappings. Your Cribl Stream data will largely determine the mappings you want. See Events Object in the Moogsoft API docs.

Here are some reasonable mappings for the syslog payload in this example:

Cribl Source Field(s)	Moogsoft Target Field
`message`	`description`
`appname`	`service`
`facilityName`	`check`
`severity`, `severityName`	`severity`
`procid`	`tag.process-id`

Map the Severities

The Moogsoft event schema has a severity field. You can specify integers or strings, from `0`, meaning Clear, to `5`, meaning Critical. Here are some reasonable mappings.

Syslog Severities	Moogsoft Severities
Emergency (`0`), Alert (`1`), Critical (`2`)	Critical (`5`)
Error (`3`)	Major (`4`)
Warning (`4`)	Warning (`2`)
Informational (`6`), Debug (`7`)	Unknown (`1`)
Notice (`5`)	Clear (`0`)

Verify your Mappings

Once you save and apply your mappings in Moogsoft, do the following:

1. In Cribl Stream, return to the **Test** tab for the Webhook Destination. Select **Test** again to send another payload.
2. In Moogsoft, go to the **Alerts** screen. You should now see a new alert based on the payload you just sent.

Metric Mapping

To illustrate how to map Cribl Stream payloads to Moogsoft events, we'll walk through an example with a payload of Cribl Stream syslog messages. You'll follow the same overall workflow for any metrics payload.

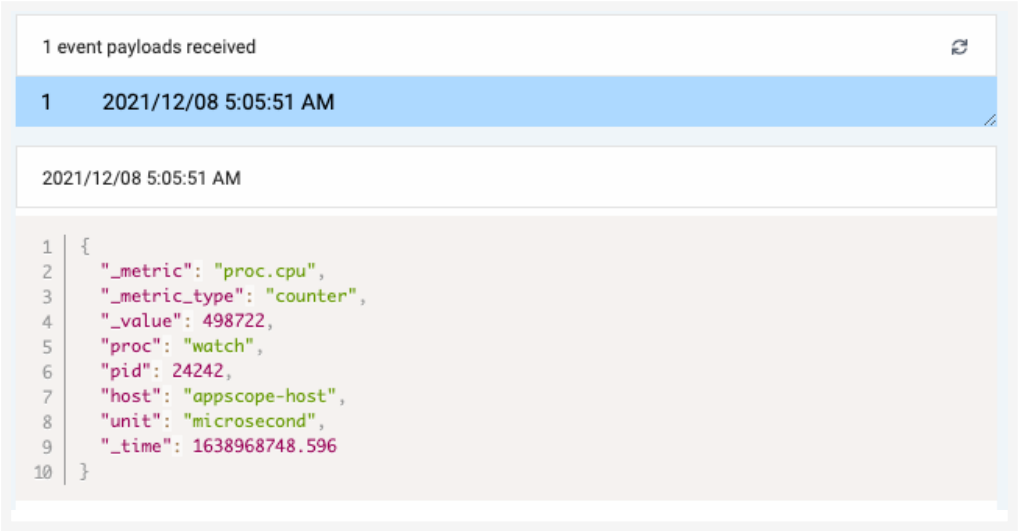
Send a Sample Payload to Moogsoft

In Cribl Stream, navigate to your Webhook Destination's configuration modal, select the **Test** tab, then:

1. In the **Test input** field, define one or more JSON payloads for the Cribl data you want to send. To map syslog events, set the **Select sample** drop-down to **appscope-metrics.log**.
2. Select **Test**.

Map the Metrics Fields

In Moogsoft, view the custom integration's config screen. Under **Map Your Data**, you should now see the payload you just sent.



The metrics payload appears in Moogsoft

Select the payload and then define the Cribl-to-Moogsoft mappings you want. Your Cribl data will largely determine these mappings. See Metric Datum Object in the Moogsoft API docs.

Here are some reasonable mappings for the AppScope payload you just sent:

Cribl Source Fields	Moogsoft Target Field
`_metric`	`metric`
`_value`	`data`
`host`	`source`
`unit`	`tag.unit`
`pid`	`tag.pid`

Verify Your Mappings

After you save and apply your mappings in Moogsoft, test them like this:

1. In Cribl Stream, return to the Webhook Destination's **Test** tab. Select **Test** again to send another payload.

8.8.11. Nightfall Integration

Thanks to Nightfall for contributing this page and the corresponding Nightfall Pack, which you can install in Cribl Stream and customize as described below.

The Nightfall Pack relies on Nightfall's Data Loss Prevention (DLP) engine, which uses machine learning to detect sensitive information in events streamed through Cribl. The Pack enables you to monitor your Pipelines in real time for PII, PHI, PCI, and other sensitive data. Nightfall discovers and automatically redacts the sensitive content before it can reach any Destination. Aside from redaction, you can use the Pack to automatically label problematic event data.

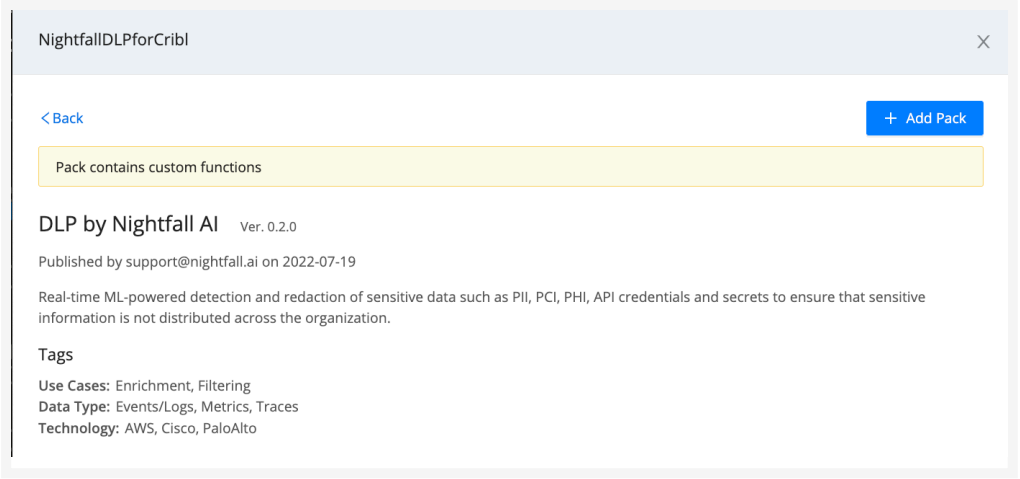
The risk of sensitive data spreading across data pipelines grows as organizations work with data from more sources. Nightfall can help ensure that sensitive data doesn't accumulate in the first place, facilitating compliance with data-security regulations. This is important for Cloud-first organizations that must scale observability while complying with regional or industry-specific compliance regimes.

Configure the Detection Engine

1. Sign up for a free Nightfall Developer Platform account, if you have not already done so. This entitles you to scan up to 3GB of event data per month for free.
2. Log in to the Nightfall Dashboard.
3. Create an API Key.
4. Create the Detection Rules and/or Policy that you want Nightfall to scan with.
   Detection Rules enable you to specify which Detectors will scan each event. A Detection Rule simply redacts sensitive information from events.
   A **Policy** governs alerts, which Nightfall can send via email, Slack, or webhook. (Assuming that you want to send data to a Cribl Stream Destination and/or a SIEM, use a webhook.) A Policy can include up to 20 previously defined Detection Rules. It can also turn redaction off, if desired.

Install the Nightfall Pack

1. Navigate to **Products** > **Stream** > **Worker Groups**. Select a Worker Group, then go to **Processing** > **Packs**.
2. Select **Add Pack**, then select **Add from Dispensary** to open the Packs Dispensary drawer.
3. Navigate to the **DLP by Nighfall AI** Pack's tile.
4. Select the Nightfall Pack's tile to display its details page.

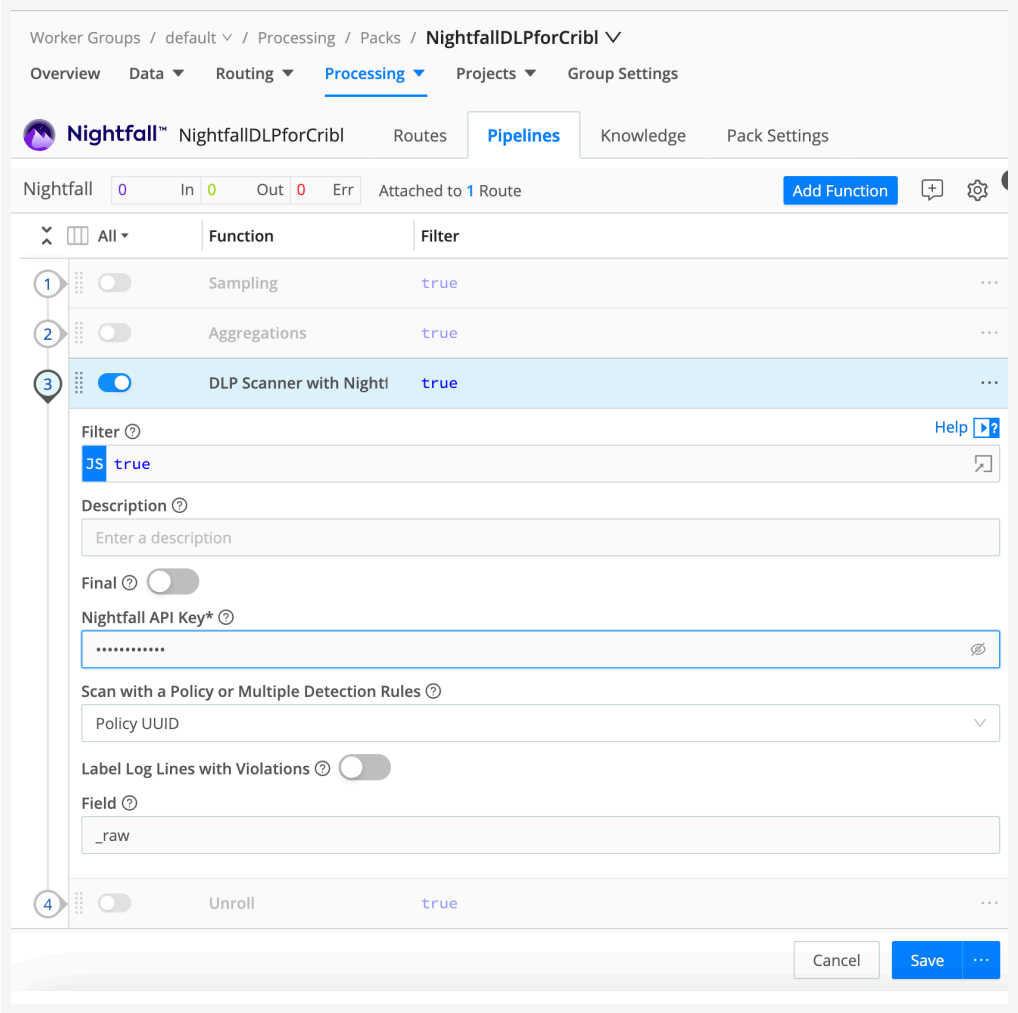


The Nightfall Pack details page

5.Select **Add Pack**. Shortly, you'll see a banner confirming that the Pack is installed.

Configure the Nightfall Pack

1. From the **Packs** page, select the new Pack. This opens the Pack's **Routes** tab.
2. In the **Pipeline** column, select `Nightfall`.
3. From the resulting Nightfall Pipeline configuration, expand the **DLP Scanner with Nightfall AI** Function.



Selecting the Nightfall Pipeline

4. In the **Nightfall API Key** field, enter the API key you created in the Nightfall Dashboard.
5. Set the **Scan with a Policy or Multiple Detection Rules** drop-down to match your detection engine configuration.
   Selecting **Policy UUID** exposes a corresponding field to enter the single UUID.
   Selecting **Detection Rule UUID** exposes a **Add Rule**. Click this to enter as many Detection Rule UUIDs as you want to specify.
6. (Optionally:) Label sensitive events. If you toggle **Label Log Lines with nightfall_violations** on, each event where Nightfall has detected sensitive information will have an added `nightfall_violations: true` field.

8.8.12. Managing QRadar Licenses

If you're using a SIEM (security information and event management) system like QRadar that's subject to an events-per-second (EPS) licensing model, managing license costs can be an issue. Cribl Stream enables you to proactively manage your license by slowing EPS growth.

Cribl Stream does so by offering fine-grained control over what events to send to the QRadar Event Processor, without breaking the expected LEEF (Log Event Extended Format) data format. You can report on event codes, and make decisions about what to drop, based on the data in the events. Some of what we discuss here also applies to SIEMs generally, not just QRadar.

Confronting QRadar Constraints

QRadar imposes constraints on data through several of its components:

- QRadar Event Collectors can send data only to QRadar. Elastic and other systems of retention or analysis are not an option.
- QRadar Event Collectors can send data only in LEEF format. The pipeline cannot provide additional streams of data in different formats such as JSON or syslog.
- Incoming data must adhere to a predefined schema, based on IBM LEEF format. You can't reformat events, nor reduce the size of an event, without breaking the correlation rules within the QRadar Event Processor.
- When incoming data exceeds the EPS license limits, QRadar will drop or stop ingesting data. Meanwhile, since the license is based on EPS event velocity, license costs increase linearly: the higher the EPS count, the higher the cost.

If your event velocity approaches the license limit, your options are unappealing:

- You can try to exclude some data, but the correlation rules and constraints on data formatting restrict your ability to be selective. You can drop only broad categories of data, such as an entire Windows Event ID, leaving you at a risk of missing something important.
- You can hope that you never actually exceed the EPS limit, but if you guess wrong, QRadar dropping or not ingesting data can be disruptive and compromise your enterprise security posture.

Cribl Stream enables you to surgically minimize these risks, and get the most out of your existing licenses.

Dropping Events Selectively

A common tactic to avoid exceeding an EPS license limit is to start dropping an entire class of events. This is a crude approach that carries its own risks. With Cribl Stream, you can analyze the data within the event, and decide what to keep based on your priorities.

Analyzing a Windows Event

Here's an example of how to analyze and selectively drop Windows Security Event ID 4674. This event documents operations attempted on a privileged object. While it's commonly discarded when admins are feeling pressed to drop events, 4674 is actually a critical ID that should be kept some, or most, of the time.

Event ID 4674 includes the logon types shown in the table below.

Logon Type	Logon Title	Description
2	`Interactive`	Activity on the system's keyboard and screen.
3	`Network`	Connection to a shared folder on this computer from elsewhere on the network.
4	`Batch`	Processes might be executing on behalf of a user without their direct intervention.
5	`Service`	Service startup by the Service Control Manager.
7	`Unlock`	An unattended workstation with password-protected screen saver.
8	`NetworkCleartext`	Logon with credentials sent in cleartext. Most often indicates a logon to IIS with "basic authentication".
9	`RemoteInteractive`	Logon with RunAs, or mapping a network drive with alternate credentials.
10	`CachedInteractive`	Logon with cached domain credentials, such as when logging on to a laptop when away from the network.

Each logon type is assigned a different security value.

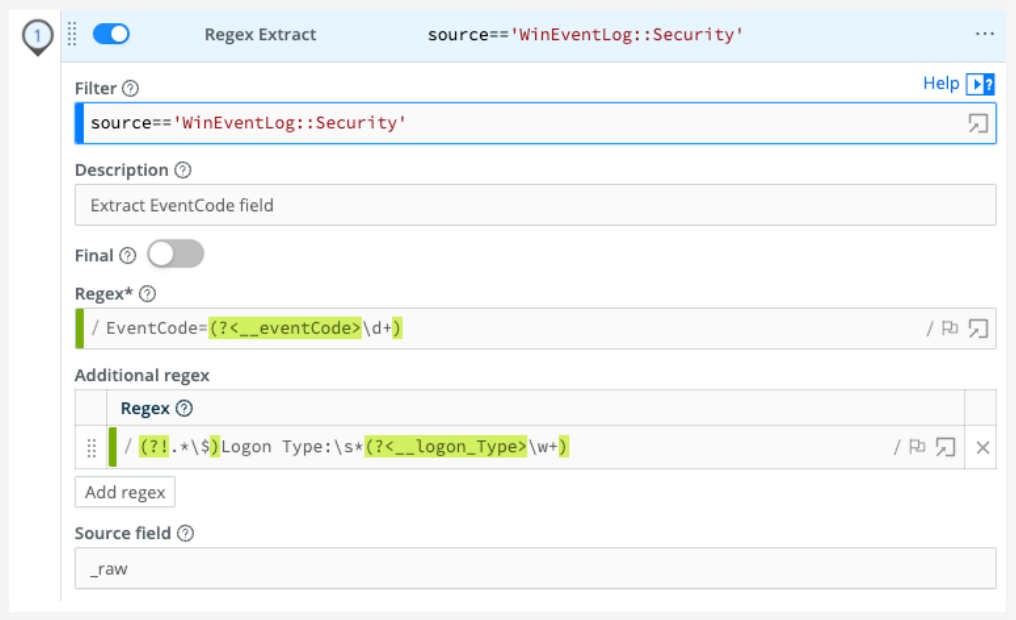
Let's suppose that your QRadar deployment is in an enterprise where inbound file sharing is disabled. In that case, you don't need logon type 3 events. So let's see how to use Cribl Stream to keep all ID 4674 events except logon type 3 events, which we'll' drop.

Dropping the Excess Baggage

Use sample data to validate the reduction methods described below. Don't apply these methods to your production SIEM until you are confident they won't interfere with the usability of your system.

1. Save your Windows Events to Cribl Stream using Live Capture.
2. Create a Pipeline and name it `Windows_Event_Reduction`.

3. Add the Regex Extract Function to your Pipeline to create fields for `EventCode` and `Logon Type`. These fields will be the foundation for a Filter that drops only Windows events of ID 4624, logon type 3.

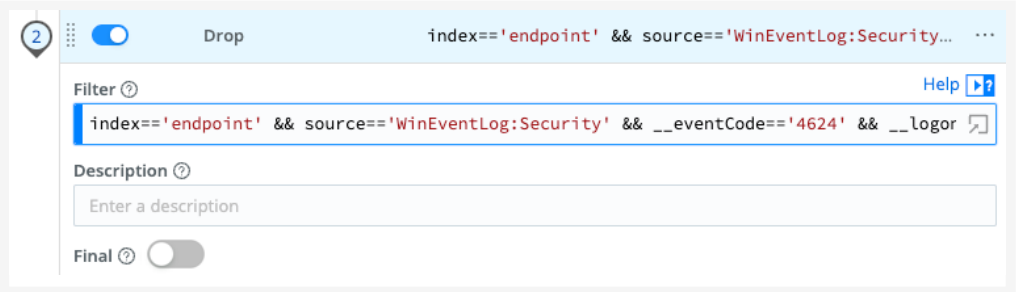


Regex Function on Pipeline

4. Add the Drop Function to your Pipeline using the Filter below.

```
index=='endpoint' && source=='WinEventLog:Security' && __eventCode=='4624' && __logor
```

Copy



Drop Function

5. Apply the Pipeline to your Windows Sources. Cribl Stream will then start dropping the unneeded events.

Other Reduction Examples

Apply the options in this section only after careful analysis of your own environment and priorities. Even if you choose to exclude these events from QRadar, you might want to send them to an object store for

> later auditing and/or replay.

WinEvents

- Event ID 4624: Look for Subtypes (Logon Type) that can be dropped.
- Event ID's 4634 and 4674: An account was logged off. Most security use cases don't require Logoff events.
- Event ID 4673: A privileged service was called. This event rarely contains actionable intel. It's a large-volume data source that can be dropped, in exchange for other, more-valuable event IDs.
- Event ID 4674: An operation was attempted on a privileged object. Similar to Event ID 4673, this event is largely considered "noise," providing little value. Consider dropping.
- Event ID 4689: A process was exited. This source, when paired with Event ID 4688, can tell you "how long a process was running." Typically, this is unimportant information, and can be omitted from your event collection.

Palo Alto

The following filters can be helpful to reduce the noise:

- All external-to-internal traffic AND `action=deny`: `Ingress traffic && action=deny`
- `PAN Traffic logs && application=ping|ibm-bigfix|outlook-web-online|google-base|traceroute|ms-teams|ic mp|outlook-web-online|ms-office365-base|crowdstrike`

Cisco ASA

- All external-to-internal traffic AND `action=deny`: `Ingress traffic && action=deny`
- Teardown UDP Connection && Event ID 302016
- Teardown TCP connection && Event ID 302014
- Teardown ICMP connection && Event ID 302021
- Teardown Translation && Event ID 305010|305012

8.8.13. Slack/Webhook Integration

You can configure Cribl Stream to send simple Notifications to Slack via a Webhook URL. These Notifications arrive in Slack as messages in a channel that you choose. For more information, see our Slack Notifications topic.

However, if you have specific event data such as alert-based or conformance data that you want to route to Slack for prompt attention, you can use Cribl Stream's Webhook Destination to route or filter this data.

You can also enhance your messages with more advanced formatting. Consult the Slack documentation for detailed information.

Cribl Stream does not currently support the use of special characters in Slack notifications sent via a Webhook Destination.

Creating a Slack/Webhook integration is a three-step process:

1. Create a Slack demo app.
2. Configure the Webhook Destination.
3. Verify that Slack is receiving messages.

Create a Slack Demo App

1. Create a Slack app by following these Slack instructions on sending messages with incoming webhooks.
2. Test the cURL command supplied when setting up the Slack app. For example:

```
curl -X POST "https://hooks.slack.com/services/T00000000/B0000000000/XXXXXXXXXX
-H "Content-Type: application/json" \
-d '{
  "text": "Hello, World!"
}'
```

Copy

Configure a Webhook Destination

Configure a Webhook Destination in Cribl Stream as described in Webhook. However, you must override some of the default settings:

- In **General Settings**, use the **URL** provided for your Slack demo app.
- In **General Settings** > **Optional Settings**, set **Format** to `Custom`.
- In **General Settings** > **Optional Settings**, set **Source expression** to `` `{"text":"${_raw}"}` `` (with literal backticks).
- In **Advanced Settings**:
  - Toggle **Validate server certs** off.
  - Toggle **Compress** off.
  - Set **Events-per-request limit** to `1`.

Verify That Slack Is Receiving Messages

To verify that Slack is receiving message, send a test message, using the **Test** UI. For example:

```
[
  {
    "_raw": "Hello, World!"
  }
]
```

Copy

If the integration is working correctly, this test message will arrive in the Slack channel you're using for your demo app.

8.8.14. Zscaler NSS Virtual Machine

Zscaler Nanolog Streaming Service (NSS) uses a VM to stream traffic logs in real time from the Zscaler Nanolog to a SIEM. Cribl Stream can take the place of the SIEM in this arrangement. Then you can use Cribl Stream to greatly reduce the size of ZScaler logs.

For Zscaler Cloud NSS, use the Zscaler Cloud NSS Source, which uses HTTP/HTTPS.

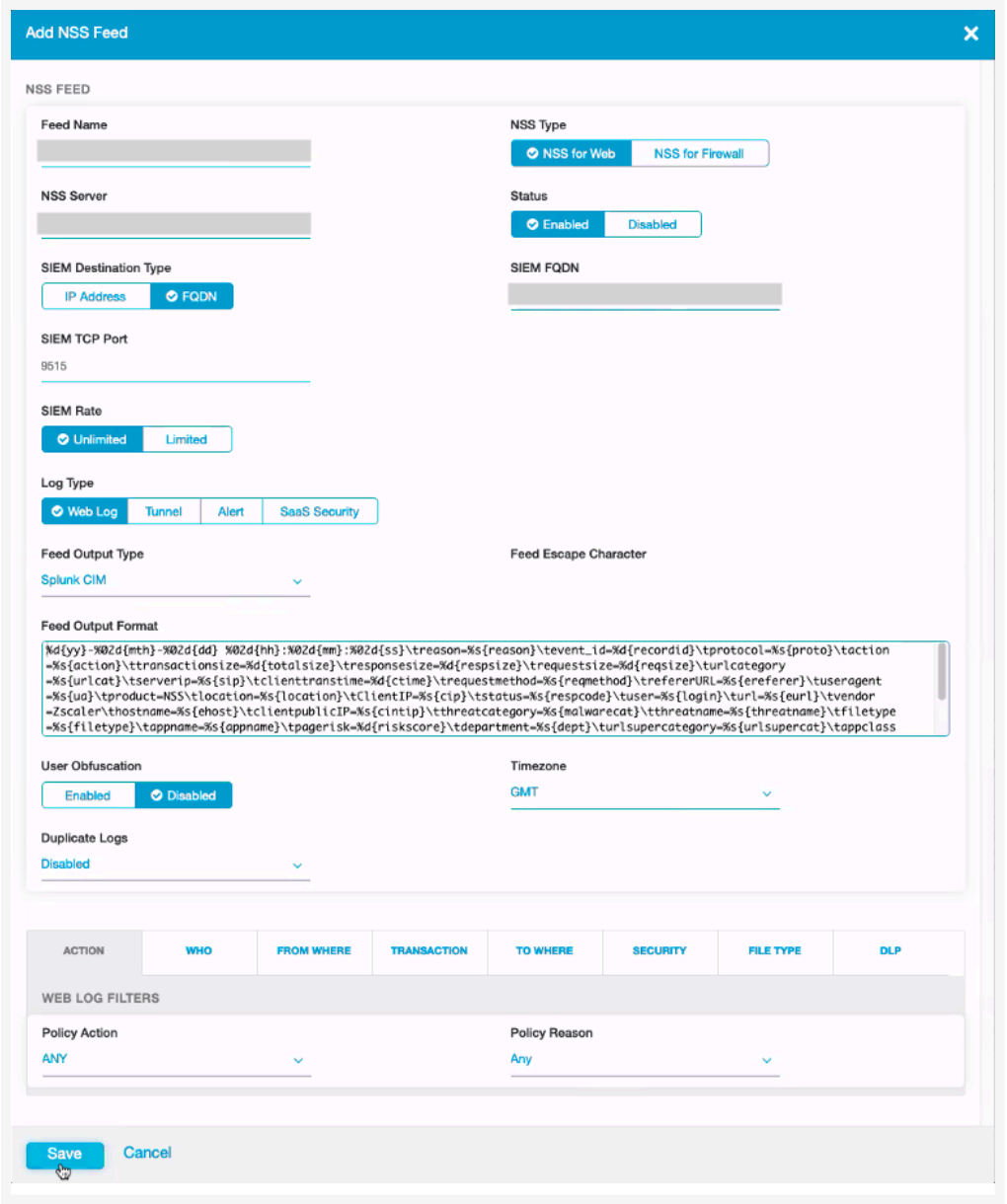
Configuring NSS to Send Data to Cribl Stream

Since Nanolog forwards data to a single IP address or FQDN, Cribl recommends that you use a load balancer to distribute data among Cribl Stream Workers.

Nanolog delivers data using a raw TCP connection.

In Zscaler:

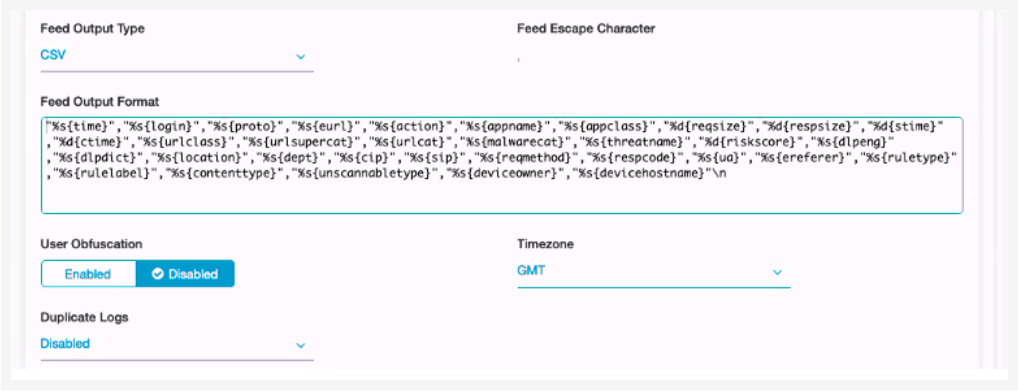
- Go to **Administration** > **Nanolog Streaming Service**.
- In the **NSS Feeds** tab, click **Add NSS Feed** to open the following configuration window:



Adding a Zscaler NSS feed

- Enter a **Feed Name** that identifies this feed as one that sends data to Cribl Stream.
- Enter the IP address or FQDN for either your Cribl Stream instance, or the load balancer you're using with your Cribl Stream instances.
- Select a **Feed Output Type**. `Splunk CIM`, a tab-delimited key/value format, is a typical choice.

Alternatively, you choose a different option, such as CSV:



Choosing a CSV output format

Example Pipeline

Cribl Stream can reduce Zscaler log size by (1) reformatting and reshaping the data, and (2) suppressing, sampling, and dropping appropriate fields.

The following code block shows how to correctly parse tab-delimited key-value pairs.

```
let temp = {};

// Substr drops the timestamp from _raw, otherwise the split does not work correctly
__e['_raw'].substr(20).split('\t').forEach((element) => {
    // Split K=V on the first equal sign
    let eq = element.indexOf('=')
    let name = element.substr(0, eq);
    let value = element.substr(eq + 1);

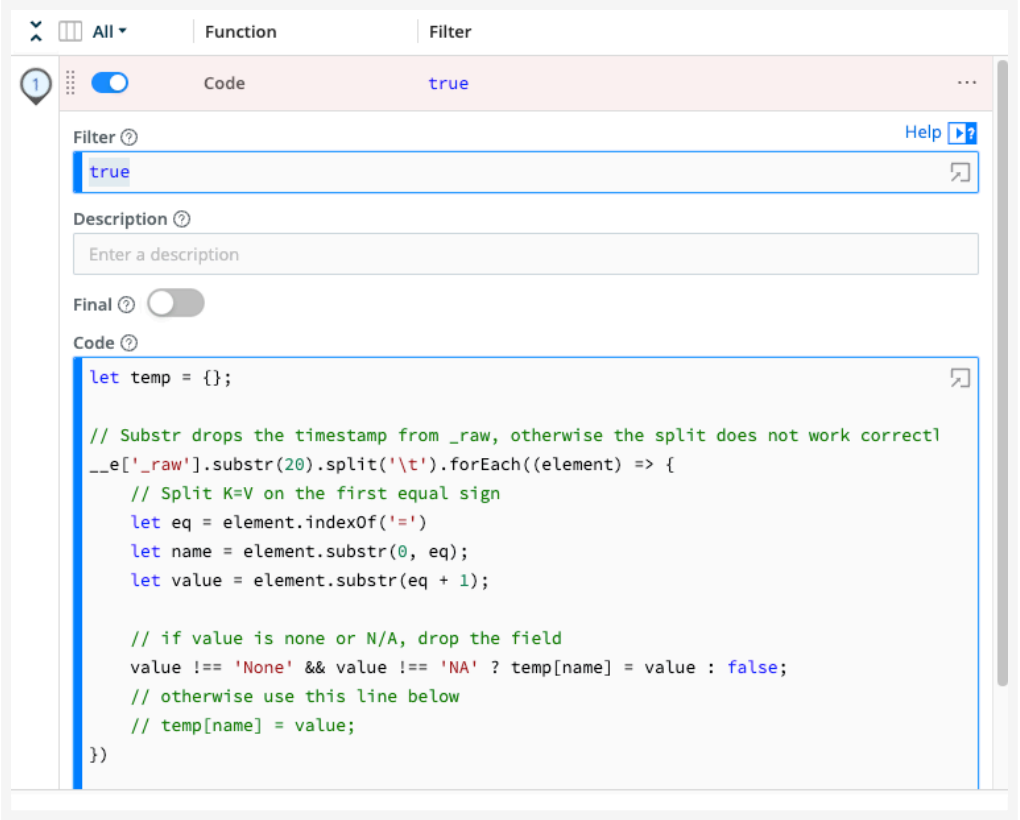
    // if value is none or N/A, drop the field
    value !== 'None' && value !== 'NA' ? temp[name] = value : false;
    // otherwise use this line below
    // temp[name] = value;
})

__e['_raw'] = temp;
```

Copy

Here's an example Pipeline that uses the parsing code above. (You can directly import this Pipeline in JSON form.)

A Code Function parses the data:

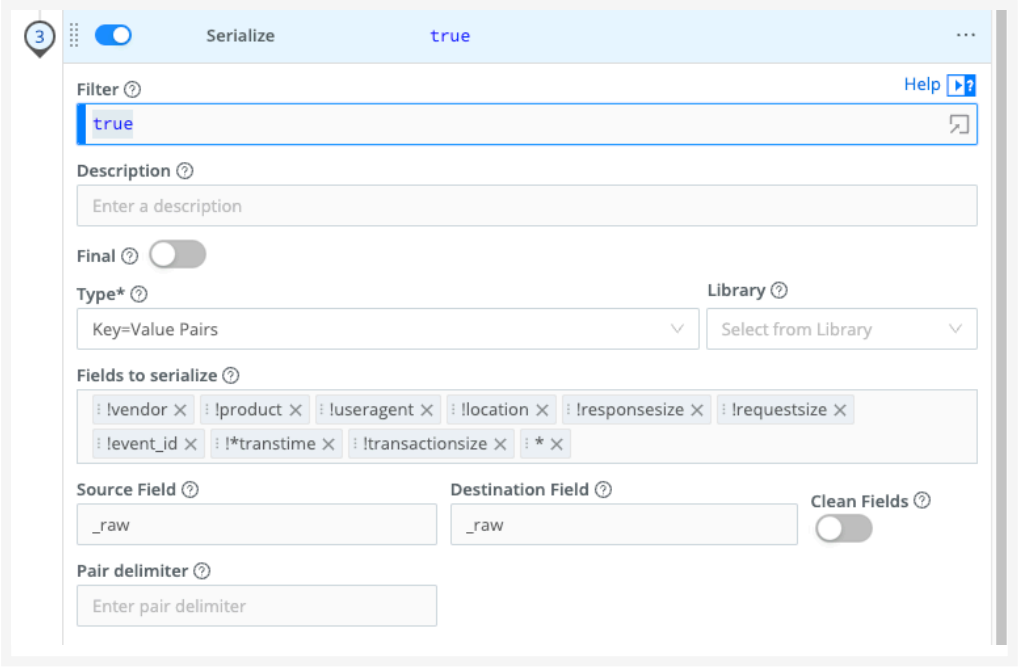


Parsing with a Code Function

An Eval Function reshapes the data:

Reshaping with Eval

And finally, a Serialize Function drops unwanted fields:



Dropping fields with Serialize

Example Pipeline JSON

To import the example Pipeline directly, copy and save the JSON below, then follow these instructions.

Zscaler Example Pipeline

```
{
  "id": "zscaler",
  "conf": {
    "output": "default",
    "groups": {},
    "asyncFuncTimeout": 1000,
    "functions": [
      {
        "id": "code",
        "filter": "true",
        "disabled": false,
        "conf": {
          "maxNumOfIterations": 5000,
          "code": "let temp = [];\n\n// Substr drops the timestamp from _raw, otherwi
```

8.8.15. Tanium To Cribl Stream

Tanium is a security platform that provides rapid searches across multiple endpoints. Tanium integrates with a finite number of tools using its Tanium Connect module, but this can still leave gaps in conforming Tanium output to the specific data formats required by unsupported tools and destinations.

Cribl Stream can help collect, reduce, enrich, transform, and route data from Tanium to any destination. This includes SIEM (Security Information and Event Management) tools, logging tools, or other analytics platforms. In this guide, we'll explain how to configure Tanium Connect to send Tanium-captured data to Cribl Stream. For further details, see Tanium's Configuring SIEM Destinations topic.

Configure Tanium Connect

A Tanium Connection is essentially a scheduled search/collection of data linked to a destination. Tanium translates queries into **Questions**. It formats the Questions into the Tanium Search Language for a search, providing near real-time results.

The first step in configuring Tanium Connect is to set up a Question. A Question can be a Saved Search, Question Log, Client Status, or an Event.

To configure a new connection, go to the **Tanium Module** page and click **Tanium Connect**. On the **Connect Overview** page, scroll to the **Connections** section, and click **Create Connection**.

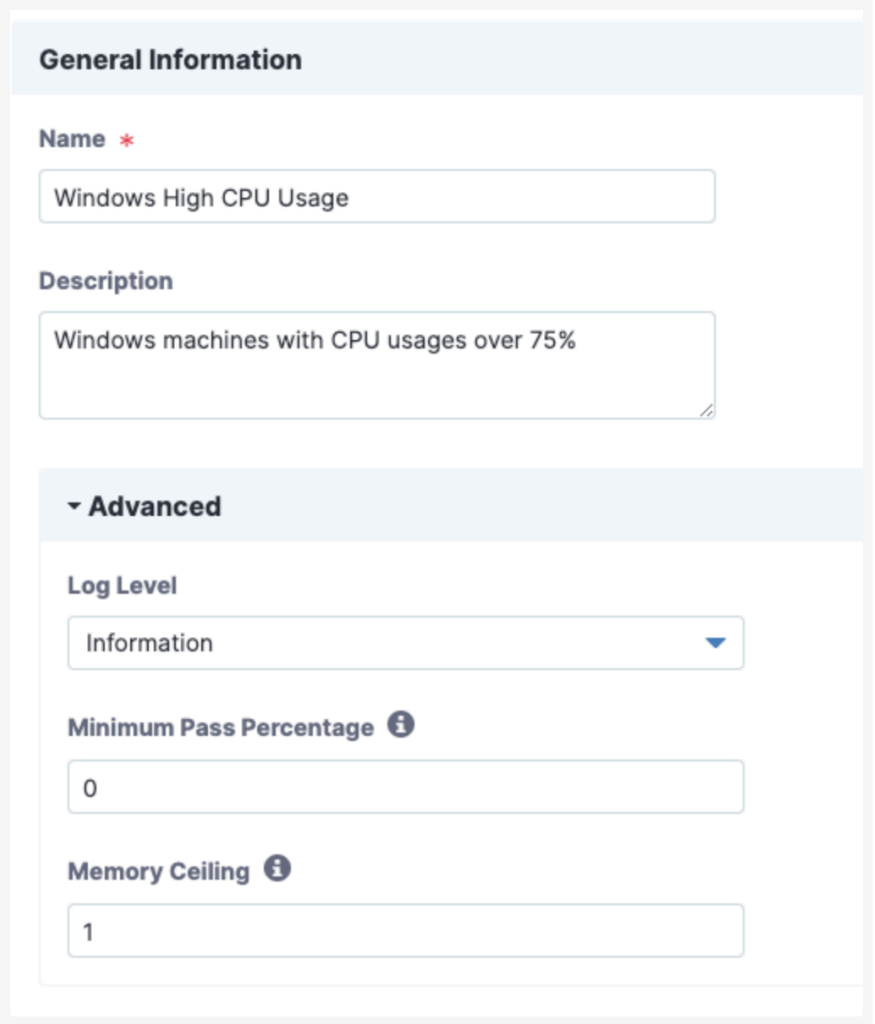
Specify General Connection Information

Name: A unique name for your connection.

Description: An optional description for this connection.

Advanced Settings: Optionally, configure the following fields:

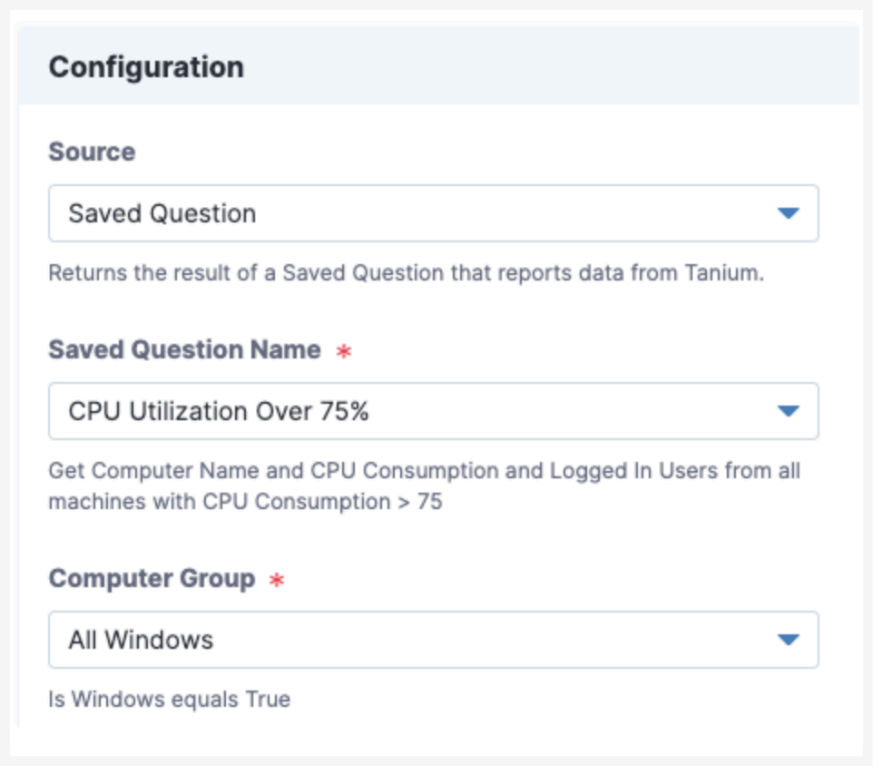
- **Log Level**: Defaults to **Information**. Change the **Log Level** to **Trace** or **Debug** if plan to debug the connection. Alternatively, set the **Log Level** to **Warning**, **Error**, or **Fatal** to reduce the amount of logging.
- **Minimum Pass Percentage**: Minimum percentage of the expected rows to process for the connection to succeed.
- **Memory Ceiling (GB)**: Maximum memory for the node process to run the connection.



General Connection Information tab

Configure the Connection Source

This section enables you to specify the type of data you are sending to your destination. The data is usually information from Tanium, such as a Saved Question, Question Log, Client Status, or Event. The settings vary depending on the Source.



Connection Source Configuration

When sending logs to Cribl Stream, you must create a network egress rule in Tanium for port 10060. See Tanium's Configure a new network egress rule topic.

Configure Your Destination

In this section, we'll configure Cribl Stream as the destination, using the following fields.

Destination: Select the destination type. For our example, we'll configure it as a `Socket Receiver`.

New: Configure a new destination.

Name: Specify a unique name for your new destination. (This field is displayed only when configuring a **new** destination.)

Existing: Update the settings on an existing destination. (Note that this will affect all the connections that use this destination.)

Destination Name: Drop-down list where you can specify a preconfigured destination. Displayed only when updating an **existing** destination.

Copy Settings: Copy settings from a preconfigured destination.

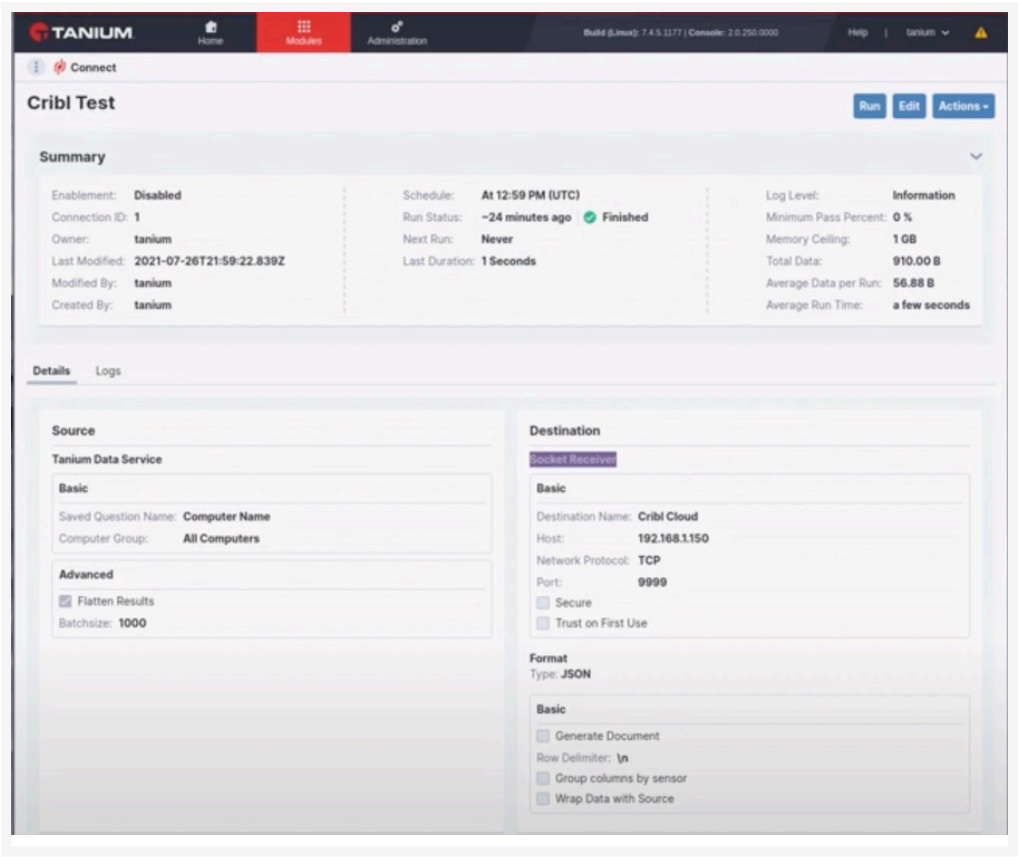
Host: Specify the destination'server host.

Network Protocol: Specify how to connect to the server (e.g., `TCP`).

Port: Enter the port number to listen on.

Secure: Select this option to use TLS encryption.

Trust on First Use: Select this option to accept the certificate presented from the server, and to trust **only** that certificate for future connection runs.



Sample Cribl Stream Configured Destination

Format the Data

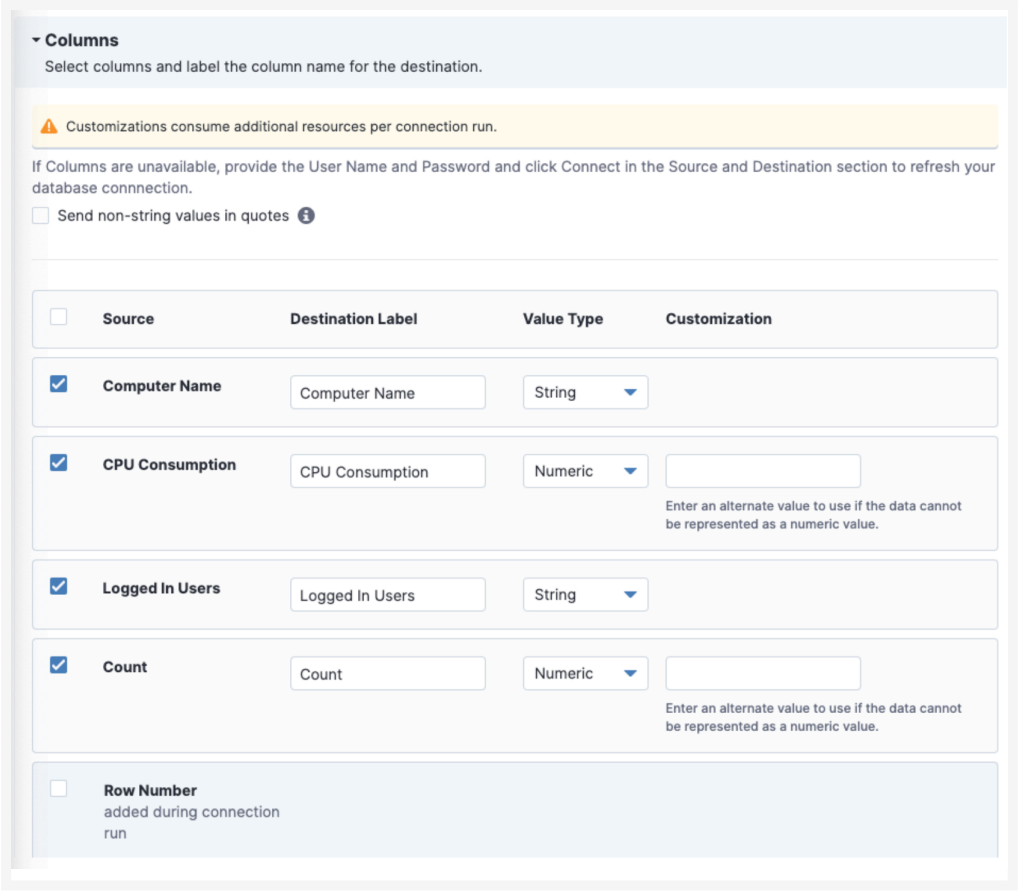
When you select a destination, the expected data format is displayed by default. For example, if you select Splunk, the `Syslog RFC 5424` automatically pre-populates the **Format Type** field. However, you can customize the format as needed. (For details, see Tanium's Format Types topic.)

Format Type: When sending to Cribl Stream, select the `JSON` data format for best results.

In the **Columns** section, configure the columns that you want to pass on to your Destination. (For details, see Tanium's Column Customizations topic.)

- **Source**: Check the box next to each **Source** to include the columns in your Destination.
- **Destination Labels**: You can optionally assign a new column heading. Defaults to the **Source** name.
- **Value Types**: You can change the data type to `String`, `Numeric`, or `Data/Time` value.
    - If you select a `Numeric` value, you must specify a default value. It can be any integer.

- If you select a `Data/Time` value, specify the format to apply for the column. R(For details, see Tanium's Time Stamp Variables topic.)



Configure your columns

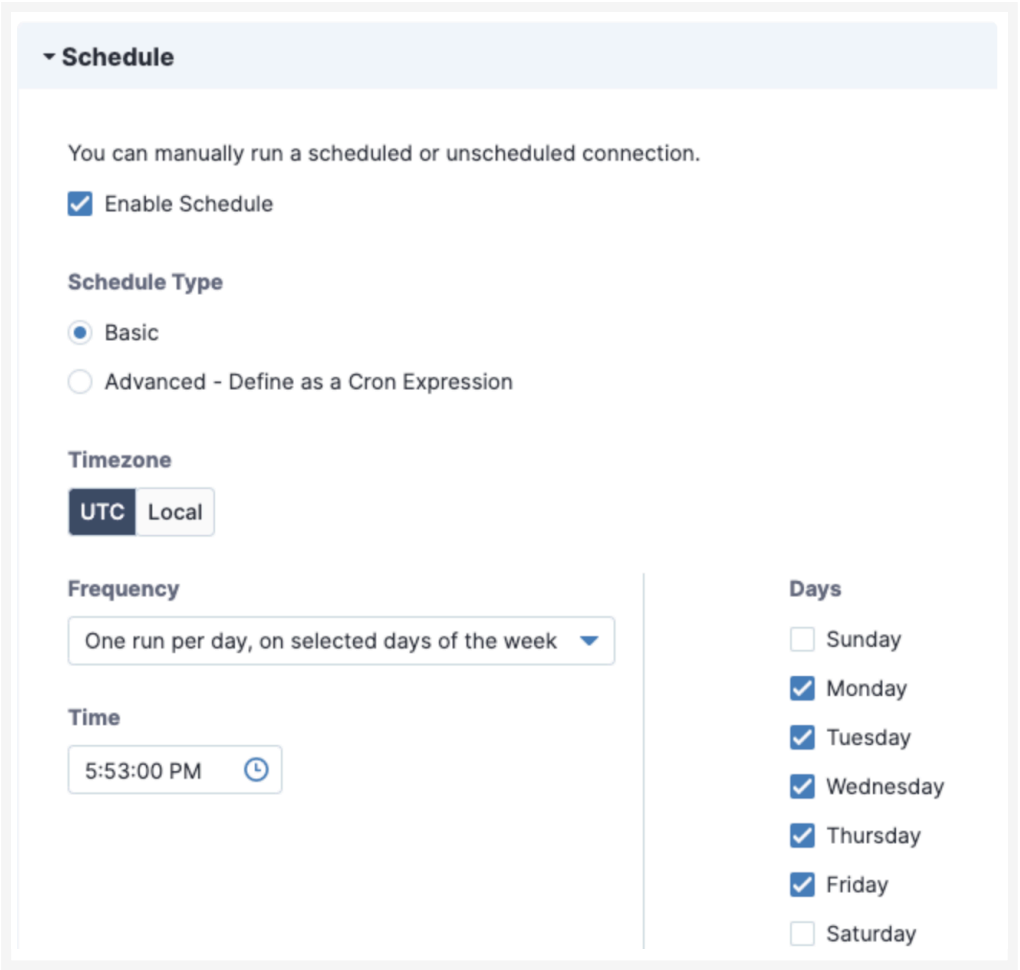
Schedule the Connection

Connections can run at a highly configurable time interval - anywhere from multiple times per hour, to daily, weekly, or monthly intervals. The **Schedule** section allows you to enable and configure the scheduler.

Enable Schedule: If you do not enable the scheduler here, the connection will run only when you explicitly run it.

Schedule Type: Select **Basic** to build a schedule with the provided controls.

Advanced - Define as a Cron Expression: Select this field to view or edit the cron expression directly.



Configure your columns

9. Packs

Packs enable Cribl Stream administrators and developers to pack up and share complex configurations and workflows across multiple Worker Groups, or across organizations.

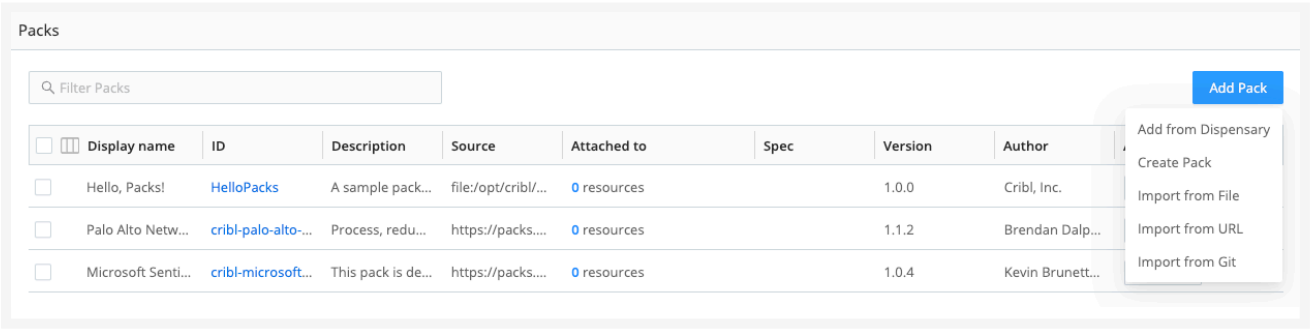
This page is a general reference for Packs. For more information about specific use cases for Packs, see:

- Onboard Faster With Packs for information about using Packs from the Cribl Dispensary.
- Pack-Based Configuration Management for information about using Packs to templatize your entire end-to-end Cribl deployment into a single, portable, reusable unit.
- Packs Publication Standards for community guidelines about contributing a Pack to the Cribl Dispensary.

Get Packs

You can add Packs to your deployment by importing them from:

- Packs Dispensary
- File
- URL
- Git repo



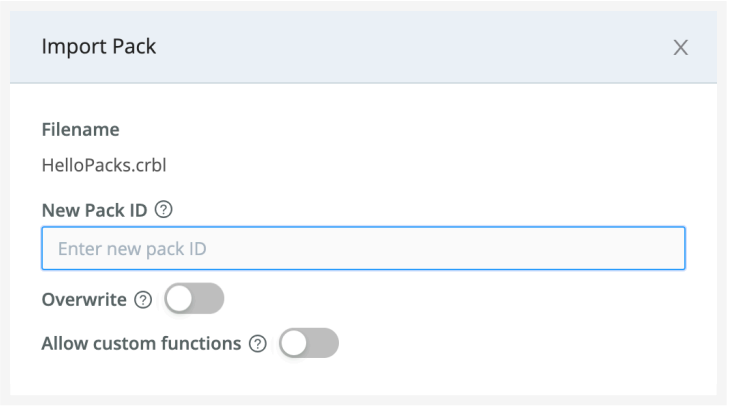
Importing a Pack

Pack Naming on Import

Each Pack that you install within a given Worker Group (or Single-instance deployment) must have a unique ID. The ID is based on the internal configuration of the Pack - not its container file name, nor on its **Display name**.

If you import a Pack whose internal ID matches an installed Pack - whether an update, or just a duplicate - you can choose one of two options:

- Assign a unique **New Pack ID**. This will import the Pack as a new, separate Pack with the new ID.
- Toggle **Overwrite** on to overwrite the installed Pack, reusing the same ID.



Renaming a Pack on import

Be careful about enabling the **Overwrite** option. Doing so will cause the imported Pack to completely overwrite your existing Pack configuration.

Each Pack within a Worker Group must have a unique **Pack ID**, so you cannot share an ID between two (or more) installed Packs.

Add From Dispensary

See Onboard Faster With Packs for more information about using Packs from the Cribl Dispensary.

Import from a File

To import a Pack (`.crbl` file) from your local filesystem:

1. Navigate to the **Packs** page.
2. Select **Add Pack** at the upper right and select **Import from File**.
3. Select the file to import.
4. Optionally, give the Pack an explicit, unique **New Pack ID**. (For details about this option, see Upgrade an Existing Pack.)
5. Select **Import** to confirm the import.
6. If the Pack you are importing binds variables to any fields, you will see a **Configure Variables** button in the notification about successful import. Select it to go to the **Variables** page, where you can configure the values for the variables for your deployment.

Import from a URL

To import a Pack from a known, public or internal, URL:

1. Navigate to the **Packs** page.
2. Select **Add Pack** at the upper right and select **Import from URL**.
3. Enter a valid URL for the Pack's source. (This field's input is validated for URL format, but not for accuracy, before you submit the modal.)
4. Optionally, give the Pack an explicit, unique **New Pack ID**. (See Upgrade an Existing Pack.)
5. Select **Import** to confirm the import.
6. If the Pack you are importing binds variables to any fields, you will see a **Configure Variables** button in the notification about successful import. Select it to go to the **Variables** page, where you can configure the values for the variables for your deployment.

To import a Pack from a public URL, the Leader Node (or single instance) requires internet access. A Distributed deployment Leader can then deploy the Pack to Workers even if the Workers lack internet access.

Import from Git Repos

When you import a Pack from a private Git repository using a personal access token, you can also push changes back to the same branch or repository you imported from. If you don't provide an access token, Cribl Stream cannot write back to the repository. This means you won't be able to push changes to a repository without authenticating with a personal access token.

Some additional considerations:

- If you don't specify a branch during import, Cribl Stream assumes you want to use the repository's default branch.
- You cannot push changes to a Git tag. Importing from a tag is allowed, but writeback is only supported when importing from a branch.
- If you need to change the token used for Git operations, you'll need to re-import the Pack using the new token.

To import a Pack from a known public or private Git repo:

1. Navigate to the **Packs** page.
2. Select **Add Pack** at the upper right and select **Import from Git**.
3. Enter the source repo's valid URL.
   This field's input is validated for URL format, but not for completeness or accuracy, before you submit the modal. When targeting a private repo, use the format: `https://<username>:`

`<token/password>@<repo-address>`. Public repos need only `https://<repo-address>`, as shown in the following example.

4. Optionally, give the Pack an explicit, unique **New Pack ID**. (See Upgrade an Existing Pack.)
5. Optionally, enter a **Branch or tag** to filter the import source using the repo's metadata. You can specify a branch (such as `master`) or a tag (such as a release number: `0.5.1`, and so forth).
6. Select **Import** to confirm the import.
7. If the Pack you are importing binds variables to any fields, you will see a **Configure Variables** button in the notification about successful import. Select it to go to the **Variables** page, where you can configure the values for the variables for your deployment.

To import a Pack from a public repo, the Leader Node (or single instance) requires internet access. A Leader in a Distributed deployment can then deploy the Pack to Workers even if the Workers lack internet access.

Prepare Pack for Use

After you have imported a Pack, take the following steps to ensure it is ready for use on your Worker Nodes:

- **Verify secrets:** If the Pack was exported in `merge` mode, any encrypted fields will have been removed for security. You must re-enter your secrets in the imported Pack to ensure it works as expected.
- **Configure variables:** If you did not configure values for a Pack variable during import, you can do it later by navigating to **Knowledge** > **Variables** in the imported Pack.
- **Commit and Deploy:** If your Pack includes Collectors and jobs, you must Commit and Deploy it after import. This step is critical because it ensures your Worker Nodes have the correct Pack context to successfully run jobs.

Use Packs

Version Compatibility

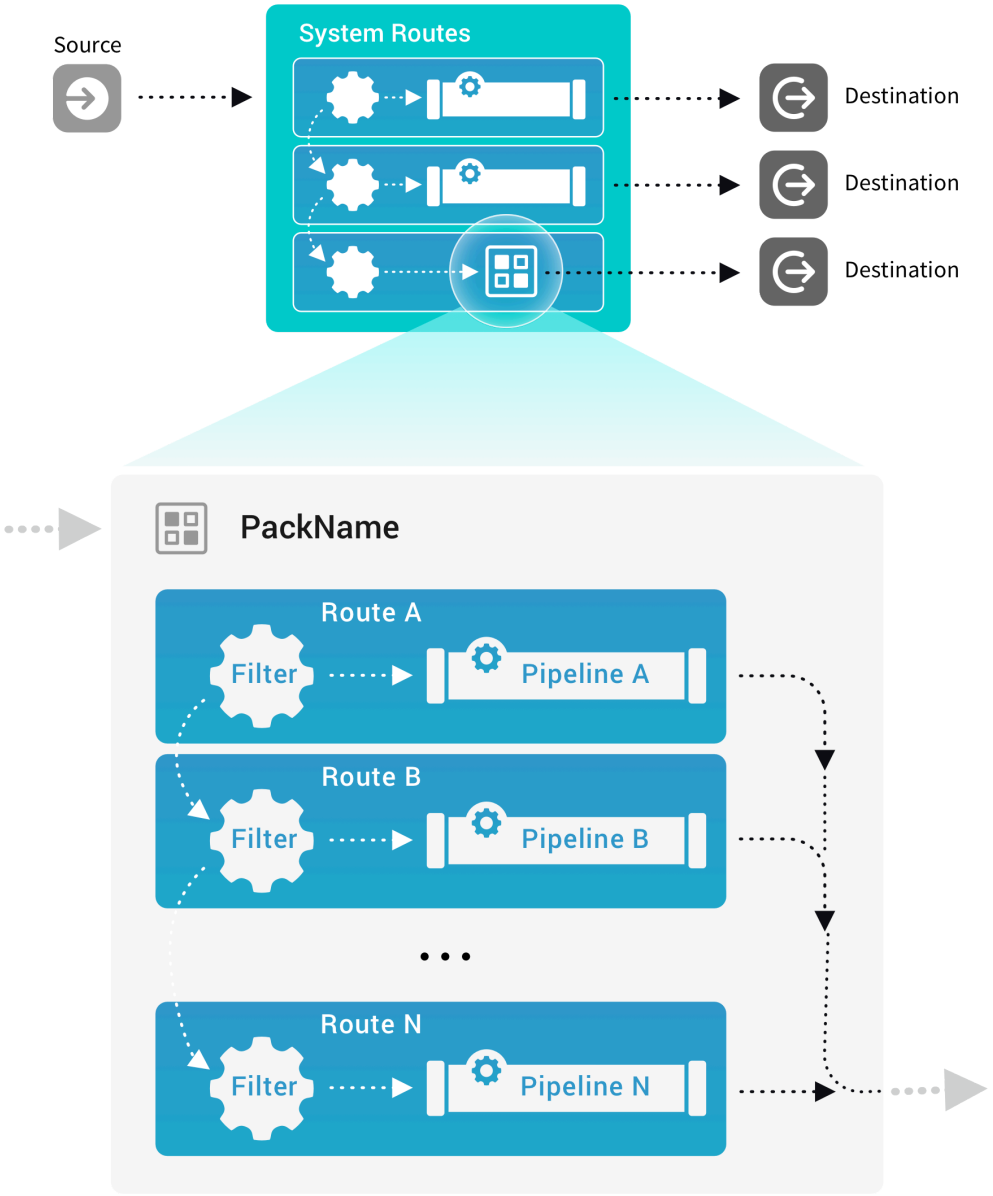
Packs created or modified in Cribl Stream 4.0.x cannot be used in any Cribl Stream version older than 4.0. (If you try, you'll see a `should NOT have additional properties` error.) To avoid this problem, Cribl recommends that you upgrade to Cribl Stream 4.0.x or newer.

For compatibility questions about any individual Packs, please contact us in the Cribl Community Slack `#packs` channel.

For Packs that do not contain Sources or Destinations, you can use them in all places where you can reference a Pipeline:

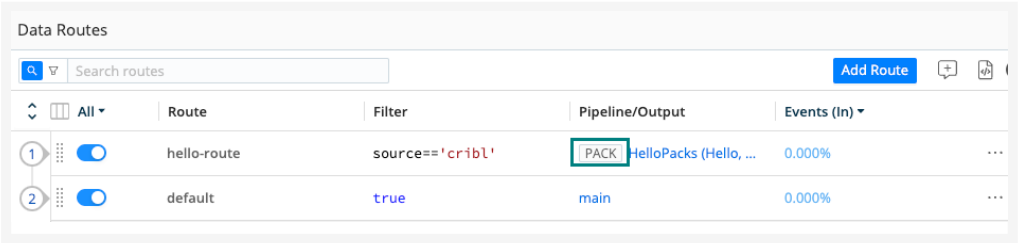
- In Sources, where you attach pre-processing Pipelines.
- In Destinations, where you attach post-processing Pipelines.
- In Routes, in the Routing table's **Pipeline** and **Destination** columns.

This expanded view shows how a Pack can replace a Pipeline in a Route:



A Pack can play the roles of an enhanced Pipeline

Packs are distinguished in the display with a **PACK** badge, as you can see here in the Routing table:



PACK badge in Routing table's Pipeline column

The **PACK** badge is also displayed when you click into a resource - shown here on one of the Routes from the above table:

PACK badge on a Pack connected to a Route

Cribl Stream's **Monitoring** page includes a **Packs** link where you can monitor the throughput of a Pack.

Access Packs

You access Packs differently, depending on your deployment type.

Single-Instance Deployment

In a Single-instance deployment, Packs are global. On the top bar, select **Processing** > **Packs**. On the filesystem, Packs (including those that you add) are stored at `$CRIBL_HOME/default/`.

Distributed Deployment

In a Distributed deployment, Packs are associated with (and installed within) Worker Groups. In the sidebar, select **Worker Groups**, and then select the Worker Group you want to manage. Next, from that Worker Group' submenu, select **Processing** > **Packs** (Stream) or **More** > **Packs** (Edge).

Each Group's Packs are stored at `$CRIBL_HOME/groups/<groupName>/default/`.

By design, you can readily share Packs **across** Worker Groups by copying them between Groups.

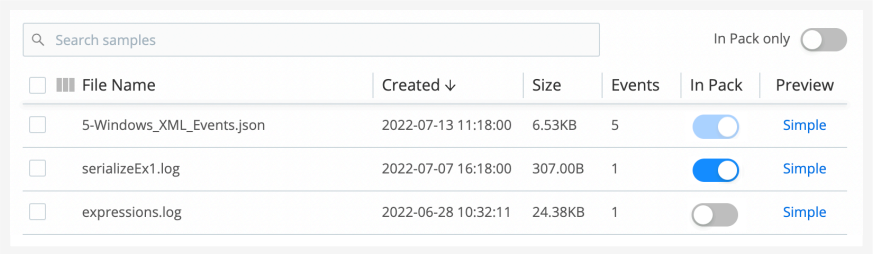
Use Sample Data

Within a Pack, the right pane defaults to displaying all sample data files available on your Cribl Stream instance. If you prefer to filter only sample files internal to the Pack, select the **In Pack only** option.

If you add sample data files via a Pack UI, they will be internal to that Pack. Each sample file here displays its own **In Pack** toggle on its row, which works as follows:

A light-blue toggle is locked, meaning that this sample file is internal to the Pack. It will export with the Pack. If you want to make this sample available across Cribl Stream, you'll need to also add it via the global right preview pane (accessed from **Routing** > **Data Routes** or **Processing** > **Pipelines**).

The toggle indicates whether the sample file is global to Cribl Stream and available to this Pack. Toggle on if you want the sample file to export along with the Pack.

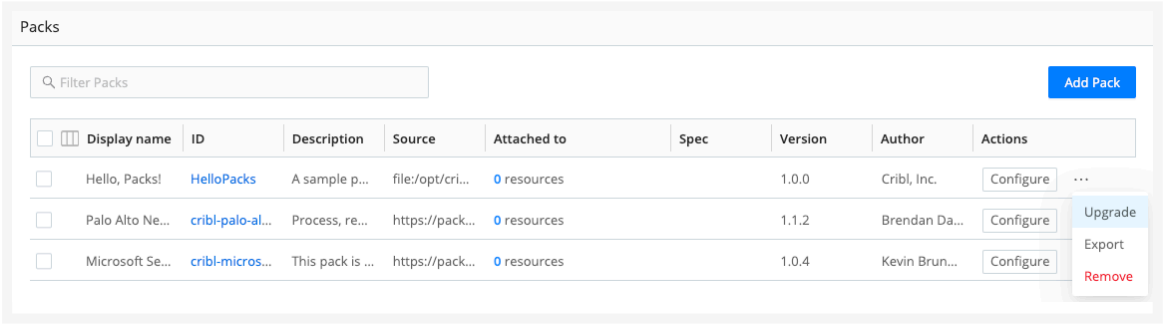


Sample file in Pack

Basically, you can manipulate all the options here as you would work with the equivalent options in the global navigation.

Upgrade an Existing Pack

To upgrade an **existing** Pack, open the Options ••• menu on its row, and then select **Upgrade**.



Upgrading an existing Pack

If you've modified an installed Pack, Cribl Stream will block the overwrite of the Pack, to prevent deletion of your locally created resources.

When upgrading a Pack, Cribl recommends that you:

- Import the updated Pack under a new name that includes the version number (for example, `cribl-syslog-input-120`). This allows you to review and adjust new functionality against currently-deployed configurations.

- Do a side-by-side comparison of the previous and new versions of the Pack - remember to review all comments in the new Pack. Doing this side-by-side comparison allows you to copy Function expressions and other settings from the current version into the same fields in the new version.
- Enable or disable any Functions in the new Pack as necessary.
- Update any Routes, Pipelines, Sources, or Destinations that use the previous Pack version to reference the new Pack.
- If the Pack includes any user-modified versions of default Cribl Stream Knowledge objects (for example, lookup files): Be sure to copy the modified files locally for safekeeping, before upgrading the Pack. After you install the upgrade, copy those files back to the upgraded Pack, overwriting the default versions in the Pack.
- Test, test, and test!
- Commit and Deploy.

Copy a Pack

You can share Packs in a Distributed deployment by copying the Packs between Worker Groups. You can copy one or more Packs from the **Packs** page as described here or using the Cribl API.

To copy a single Pack, your first step is to open its Options ••• menu, then select **Copy to another Worker Group**.

To copy multiple Packs in one operation:

1. Select the check boxes for all Packs you want to copy on the **Packs** page.
2. In the options ••• menu, select **Copy selected Packs to another Worker Group**.
3. Select the Worker Groups to which you want to copy the Pack.
4. Optionally, select the **Overwrite** option to replace existing Packs that have the same **Pack ID**.

Be careful about enabling the **Overwrite** option. Doing so will cause the imported Pack to completely overwrite your existing Pack configuration.

5. Select **Copy**.

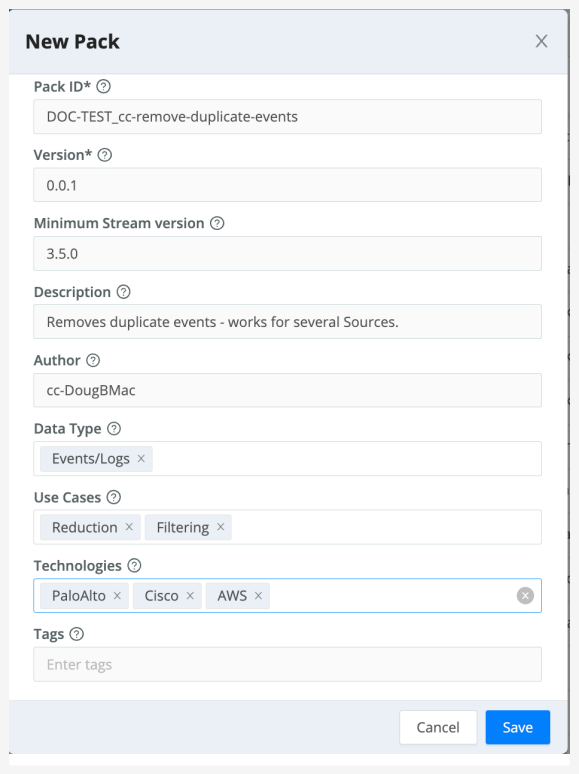
A status modal will either display a success message, or list any Packs that failed to copy.

Create a Pack

You can create a new Pack from scratch, to consolidate and export multiple Cribl Stream configuration objects:

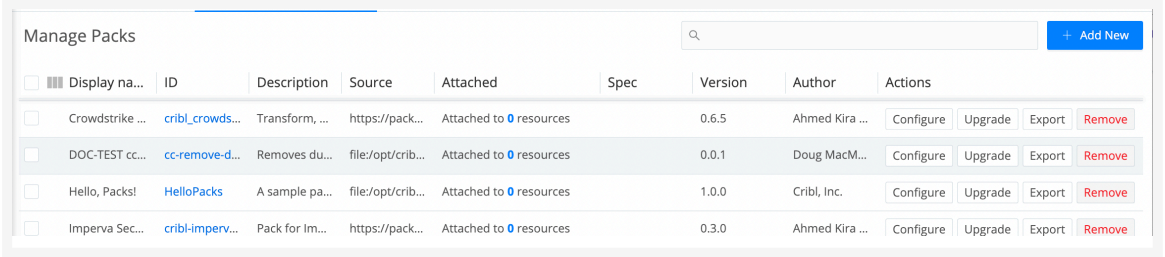
1. Navigate to the **Packs** page and select **Add Pack**.
2. From the submenu, select **Create Pack**.

3. Enter a unique **Pack ID** and other details:
    - Each Pack within a Worker Group must have a separate **Pack ID**, but you can assign arbitrary **Display names**.
    - **Version** is a required field identifying the Pack's own versioning.
    - **Minimum Stream version** is an optional field specifying the lowest compatible version of Cribl Stream software.
    - **Description** and **Author** are optional identifiers.
    - **Data type**, **Use cases**, and **Technologies** are optional combo boxes. You can insert one or multiple keywords to help users filter Packs that you post publicly on the Cribl Packs Dispensary.
    - **Tags** are optional, arbitrary labels that you can use to filter/search and organize Packs.
4. Select **Save**.



Creating a Pack

1. On the **Packs** page, select the new Pack's row to open the Pack.



Manage Packs page

1. Use the standard Cribl Stream controls to configure and save the infrastructure you want to pack up. As you save changes in the UI, they're saved to the Pack.

Pack Variables

To make your Packs portable and reusable across different environments, replace fixed values in your configuration with variables and secrets. This practice, known as templatization, allows your Pack to adapt to different environments without you having to manually edit it each time.

For more information about Pack variables, see:

- Pack Variables
- Create a New Variable
- Use Variables in a Pack
- Update Variables After Import or Update
- Delete Variables

Use Named TLS Certificates

Cribl Stream allows you to optionally authenticate to Sources or Destinations using TLS (Transport Layer Security), which encrypts traffic and helps ensure secure communication with upstream or downstream services. TLS certificates help verify the identity of the remote service and can also support mutual authentication when required. See Secure Cribl Stream Sources and Destinations with Certificates for general information about authenticating with TLS certificates and specific instructions on adding and selecting a certificate to a Source or Destination.

For Sources or Destinations defined at the **global level** (outside of a Pack), Cribl Stream stores TLS certificates at the Worker Group level. Any Worker Nodes in the Worker Group can access those certificates when processing data.

For Sources or Destinations defined at the **Pack level** (inside a Pack), you can reference a TLS certificate by its name. This approach enables portable and secure configurations across environments:

- When you **export** that Pack, it does not include the TLS certificate itself, only the reference to the certificate name.
- When you **import** that Pack to a Worker Group, Cribl Stream checks the local certificate store for that Worker Group. If a certificate with the same name exists, Cribl Stream automatically binds it to the Worker Group.

This approach allows you to:

- Build TLS-secured Sources and Destinations directly within Packs.
- Reuse Pack configurations across environments without exporting certificate data.

- Maintain secure, environment-specific certificate management practices.

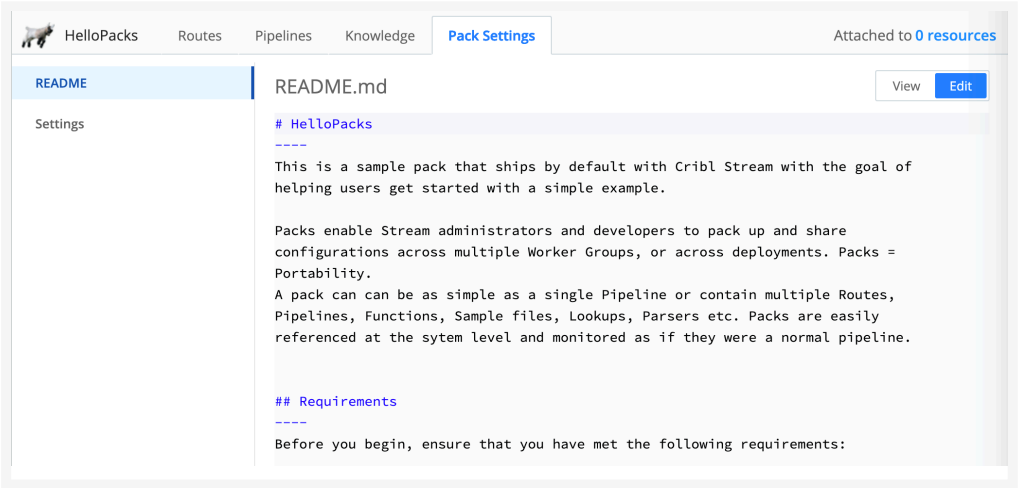
Packs do not include certificate files. Instead, they store the name of the certificate to reference. To ensure a successful deployment, make sure each environment contains a certificate with the expected name for the target Worker Group.

If you create the certificate after importing the Pack, commit and deploy the changes to restart the Worker Nodes. The Worker Nodes will pick up the new certificate after restart.

Modify Pack Settings

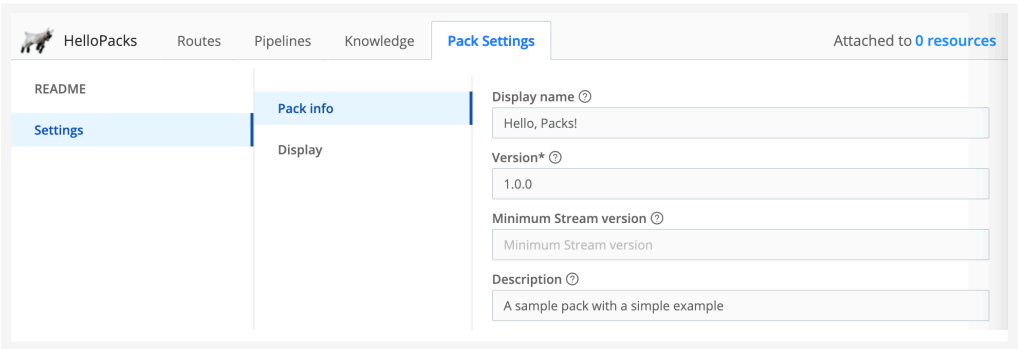
You can update Pack metadata (such as Version, Description, and Author) and the Pack's display settings. If you're developing a new Pack to share, you'll want to use this interface to populate the README and display logo within the Pack:

1. Open a Pack and select **Pack Settings**. The left **README** tab will gain focus.
2. To populate the Pack README file, toggle **View** to **Edit**, replace the placeholder markdown content with your actual Pack description, and **Save**.



Editing a Pack's README

1. To update other metadata, select the left **Settings** tab.



1. You can add a Pack logo on the **Settings** tab.
   Cribl recommends adding a logo to each custom Pack, to visually distinguish the Pack UI from the surrounding Cribl Stream UI (as well as from other Packs). You can upload a `.png` or `.jpg`/`.jpeg` file, up to a maximum size of 2 MB and 350x350 px. Cribl recommends a transparent image, sized approximately 280x50 px.

Share and Publish Packs

To share your Pack with other users, you first need to export it from your deployment.

Then, if you wish, you can publish it to the Cribl Pack Dispensary to make it available for the whole Cribl ecosystem.

Export a Pack

You can also use the Cribl API to export a Pack.

To export a newly created or modified Pack from the **Packs** page, open its Options ••• menu and select **Export**.

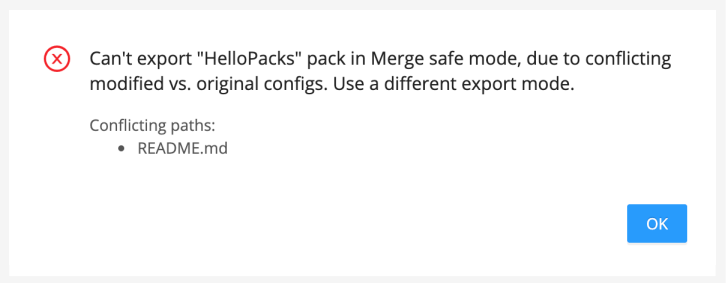
The **Export Pack** modal provides the following options.

Export Mode

Select one of these three buttons to determine the export mode:

- **Merge safe**: (Deprecated) Attempt to safely merge local modifications into the Pack's default layer (original configuration), then export.
- **Merge**: Force-merge local modifications into the Pack's original configuration, then export. When you export Packs in `merge` mode, Cribl Stream deletes all encrypted fields from Pipelines. Upon importing these packs, you will need to re-enter the secrets to make them functional again.
- **Default only**: Export only the Pack's original configuration, without local modifications.

The (deprecated) **Merge safe** option is conservative, and will block the export where Cribl Stream can't readily merge modified contents with the Pack's original contents, due to conflicts. If you encounter an error like the example shown below, use the **Merge** or **Default only** export mode instead.



Merge safe error

Exported Pack ID

Defaults to the Pack's current ID, with the version number appended. This field provides an opportunity to change the Pack's ID, if you're exporting a new version of the Pack.

Publish a Pack

If you'd like to share your Pack with the community of Cribl users, you can publish it on the Cribl Packs Dispensary.

The Cribl Packs Dispensary site is designed for sharing completed Packs. If you want to collaborate with others on iteratively developing a Pack, Cribl recommends relying on our Dispensary GitHub Repo for the development phase.

If you have a Cribl.Cloud account, you can also collaborate there by inviting team members to your Cribl.Cloud Organization. See Invite Members.

Once your Pack is ready to share, we encourage you to submit it to the Cribl Packs Dispensary site. If you already have completed Packs on our GitHub repo, bring them over here!

Change Pack Owner

If you no longer can or wish to maintain a Pack you published to the Cribl Packs Dispensary, you can transfer its ownership to another user. The new owner will have full control over the Pack.

To transfer a Pack to a new owner:

1. Log in to the Cribl Packs Dispensary.
2. Optionally, toggle **View only my Packs** to **Yes** to narrow down the list to the Packs you own.
3. Select **Configure** on your Pack's tile.
4. At the bottom of the Pack pane, select **Change Owner**.
5. Enter the email address of the new owner and, optionally, a message you want to send them.
6. Confirm by selecting **Change Owner**.

The recipient will get an email message with a request to accept ownership of the Pack. They will have 24 hours to respond.

When the new owner either accepts or rejects the request, you will be notified by email. Until the new owner responds, you can withdraw the request. To do it, open the Pack again and select **Cancel Ownership Request**.

Supported Pack Integrations

While all Destinations are supported in Packs, a small subset of Sources cannot be included:

Product	Unsupported Sources
Cribl Stream	• Cribl Internal

9.1. Pack-Based Configuration Management

As your Cribl deployment grows, maintaining consistency across multiple Worker Groups and environments becomes a significant operational challenge. To solve this challenge, use Pack-based configuration management to templatize your entire end-to-end Cribl data streams into reusable units that you can port to other Worker Groups in your deployment. This approach is helpful when you need standardized configuration management workflows to manage multiple Worker Groups, instances, or environments (such as development, testing, and production).

Pack-based configuration management has several advantages over the standard Leader-to-Worker Group configuration management workflow:

- **Composable management:** Define end-to-end data streams as a single unit. Package the configurations for a Source, its associated processing logic, and the final Destination into a reusable Pack that you can deploy across Cribl Stream to different Worker Groups and environments.
- **Git version-control:** You can version-control Packs in an external Git repository to make your Cribl configurations auditable and to flexibly adapt to your company's change management policies.
- **Environmental portability:** Move configurations easily between environments, such as from `dev` to `prod` or more. Packs allow you to bind configuration fields to variables, so that you can apply the same configuration components to multiple Worker Groups and then customize the configuration details that are specific to the local context of a given Worker Group.
- **Scalability:** Onboard new data streams or spin up additional Worker Groups quickly by deploying Packs that have been validated in one context to another.

When To Use Pack-Based Configuration Management

Pack-based configuration management is not required to manage deployments of Cribl Stream. For many deployments, the default Leader-to-Worker configuration management system is sufficient.

However, this workflow is particularly beneficial for organizations that require:

- **The ability for multiple configuration content developers or teams to collaborate on configurations.** Use this workflow if you need developers to work on configurations in parallel. Each content developer can work on isolated feature branches, merge changes systematically, and resolve conflicts before they impact production.
- **Two- or three-branch environment promotion and segregation workflows.** This workflow supports the classic development three-branch model where changes flow from `dev` to `test` to `prod` branches and environments in a controlled manner. You could also use a simpler two-branch model flow from `dev` to `prod` instead.

- **Formal configuration approval from a central Change Advisory Board (CAB).** Git workflows keep configuration changes visible so that stakeholders can review and approve them. An external Git repository can provide additional change traceability and a clear audit trail of who approved what and when.
- **The ability to roll back configurations quickly.** While this is already possible in Cribl Stream, the Pack-based configuration management workflow extends and complements the Leader-to-Worker Group control plane system by allowing you to track changes in an external Git repository. Git's version control capabilities make rollbacks straightforward so that you can easily revert to a previous, stable version. Connecting to an external repository provides additional visibility and auditability that integrates with your company's overall change management system.

Prior experience with Git is helpful for this workflow. For users familiar with Git, this workflow can streamline version control, change management, and CI/CD for your Cribl environments.

Key Components

The recommended Pack-based configuration management workflow for Cribl Stream uses these components:

- Packs: Packs contain the actual configuration content that you will track. This includes dependencies such as the Knowledge objects or (optionally) the sensitive resources (such as certificates or secrets) needed to run the Pack.
- Environments: A logical grouping of Cribl Stream resources that process data, depending on your Worker Group placement strategy. An environment is typically defined by a specific set of Worker Groups.
- Variables: Dynamic placeholders within a Pack configuration that map to a specific values defined at either the Pack or Worker Group level. Variables allow you to port the Pack to another environment, then make small adjustments to the logic to allow for specific differences that are unique to that Worker Group.
- External Git repository: A centralized Git repository serves as the single source of truth, tracking all changes to your Pack configuration. Each repository should track one individual Pack only.

Packs

The core building block of every Cribl Stream configuration management system is the Pack. A Pack contains a bundle of configurations that you want to develop, test, version control, and distribute to multiple Organizations or Worker Groups. Packs can contain configurations for many types of resources:

- Sources
- Destinations
- Event Breakers
- Pipelines
- Routes

- Knowledge Objects (such as lookup files)
- Sample data files

In other words, you can create Packs that contain a full, end-to-end Cribl Stream configuration. Think of a Pack as a configuration object that you can version control and export/import to other environments. It works well in workflows where multiple developers work in a sandbox development environment. Developers can review each other's work and merge their changes to a testing or production environment using Git workflows.

Environments

An Environment is a logical boundary that defines how data is processed, managed, and secured within your architecture. The logical boundary is defined by your Worker Group placement strategy, such as geographic regions or data proximity. It could also refer to content development environments that use the two- or three-branch development strategy, such as `dev`, `staging`, and `prod`.

While typically defined by a specific set of Worker Groups, an environment in the Pack-based workflow encompasses the unique variables and configurations required for that specific environment. This means a single Pack can be deployed across multiple environments while automatically adapting to the new context.

Variables

Variables allow you to copy or import that Pack to multiple Worker Groups, making small adjustments to the logic to adapt to the specific environment context for that Worker Group. For example, you could adjust the Worker Group-level variables that account for regional differences (such as different AWS regions), VPC endpoints, or local port assignments. This ensures that the core data processing business logic remains intact while adapting to the local requirements of its new Worker Group context. It also allows you to easily make updates to Packs and deploy those updates with minimal overhead.

External Git Repositories

Once you've built an end-to-end configuration of Cribl Stream inside a Pack, you can then integrate it with an external Git repository for version control, parallel development, and automated deployment with CI/CD.

Cribl Stream is compatible with all major Git platforms, including GitHub, GitLab, and Bitbucket.

Additional Configuration Management Components

This table explains how standard configuration management components map to Cribl Stream:

Element	Explanation	How it Applies in Cribl
Automation agent or runner	A process or tool that pulls from Git and applies the config to the target environment.	Handled inside Cribl Stream by committing and deploying imported Packs to Worker Groups. Can also be handled using a CI/CD pipeline tool in external repositories, including GitHub Actions, GitLab CI, and Jenkins.
Change workflow	Configuration changes go through a review and approval process.	Handled in an external Git repository linked to Cribl Stream in compliance with your internal change management policies.
Target system(s)	Where the configuration is ultimately applied.	Instances of Cribl Stream.
Secret management	Secrets or encrypted files.	Handled inside Cribl Stream and mapped to variables in external Git repositories.
Environment mapping	Directories or branches per environment.	Handled with variables inside Cribl Stream and mapped to variables in external Git repositories.

Considerations

When working with Packs, keep these considerations in mind:

- Packs cannot be cloned or imported from one Cribl product to another. For example, you cannot clone a Pack from Cribl Stream to Cribl Edge and vice versa.
- Sources and Destinations created in Packs add the Pack name to the `__inputId` and `__outputId` fields

9.1.1. Plan A Pack-Based Strategy

To get started with Pack-based configuration management for Cribl Stream, your first objective is to create a Pack-based configuration management strategy that fits your team's workflows and operational needs. A thoughtful plan helps reduce duplication, simplify maintenance, and make your deployments more predictable and secure.

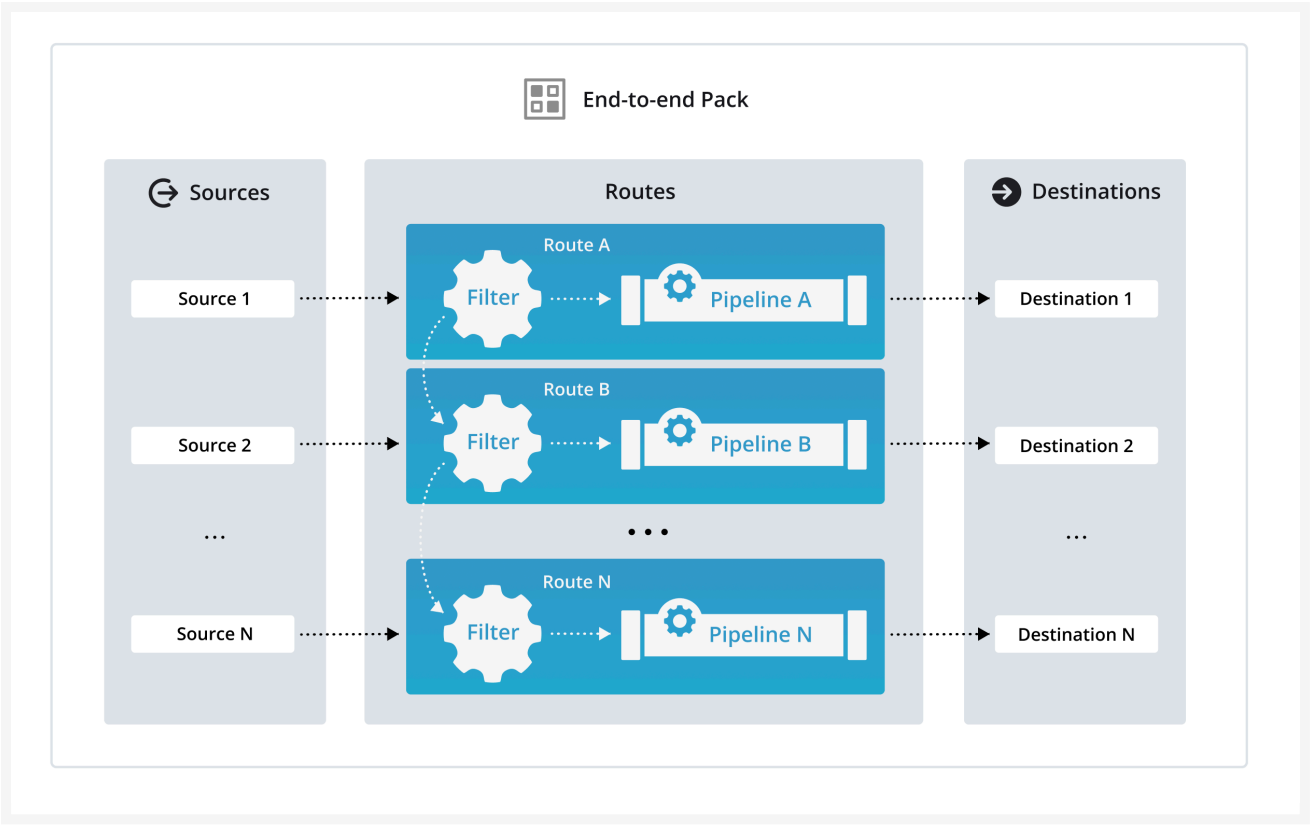
Your configuration plan should define:

- The Purpose and Scope of the Pack: The scope of the configurations in your Pack, which involves deciding whether to include Sources and Destinations. It also includes determining how data will flow in and out of Packs to other resources in Cribl Stream.
- Pack Variables: The variables you want to templatize in your Pack for easy transfer and adaptation across environments.
- Sensitive Data and Secrets: The sensitive data you need to store in Packs, including passwords, API keys, or security certificates, and authentication credentials.
- Pack Versioning System: Your Pack versioning and branch promotion strategies, which ensures your Packs remain consistent and synchronized across your Worker Groups.

The Purpose and Scope of the Pack

The core building block of every configuration management system is the Pack. A Pack contains a bundle of configurations that you want to develop, test, version control, and distribute to multiple Organizations or Worker Groups.

If you're familiar with Packs from prior versions of Cribl Stream before 4.12, it's important to know that Packs have evolved. In previous versions, Packs were mainly used to package processing logic and included Pipelines, Routes, Functions, Knowledge objects, and sample data files. Packs can now contain complete, end-to-end data flow configurations for a single data stream.



Packs can contain end-to-end data flow components

For that reason, carefully plan the scope of the Pack you want to track with version control, which primarily means deciding whether to include Sources and Destinations in your Pack:

Scope	Contains	When to Use
Limited scope	Only include Routes and Pipelines that process data. Do not include Sources and Destinations in these Packs.	When you only want the Pack to process data, meaning it will just contain a Route and Pipeline and related Knowledge objects.
End-to-end scope	• A Source. • A Destination. • All the necessary components to process data from that Source to that Destination (Pipelines, Routes, and related Knowledge objects needed for that configuration to work).	When you want to configure your full end-to-end Cribl Stream deployment inside a Pack.

Scope	Contains	When to Use
Source-side scope	• A Source. • Optionally, a Pipeline and all the necessary components to normalize, reduce, or process data from that Source, including Knowledge objects.	When you want to use a Pack similar to a pre-processing Pipeline.
Destination-side scope	• A Destination. • Optionally, a Pipeline and all the necessary components to prepare or optimize the incoming data for the requirements of the downstream receiver.	When you want to use a Pack similar to a post-processing Pipeline.

Keep each Pack scoped to a single purpose or use case where possible. Avoid mixing too many unrelated Sources or Destinations in one Pack. It increases complexity and makes version control more difficult. Consider these possible approaches for designing your Pack scope:

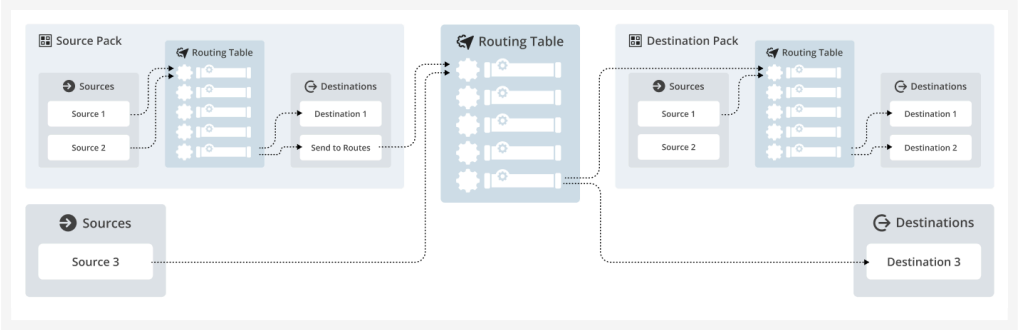
- **One Pack per vendor or technology:** Create a Pack for a specific technology (such as AWS CloudWatch or Syslog messages).
- **One Pack per data lifecycle stage:** Divide Packs by their function: one for normalization, one for enrichment, and one for redaction.
- **Packs for shared services:** Create a Pack that performs a universal task (such as redaction) that multiple other data streams can be routed through before reaching their next Pipeline or Destination.

Packs are fully self-contained, meaning that the objects inside the Pack are not aware of anything outside of the Pack context unless you use Routes to send data outside the Pack context. So, if you include a Source or Destination inside a Pack, **it won't be aware of data from an external Source or Destination** unless you integrate it into your data processing system by connecting it to a Source or Destination.

The rest of this guide assumes you plan to build Packs that include Sources and Destinations. However, you can adapt this guide to Packs with more limited scopes.

Design the Pack Data Flow

Packs allow you to break down repetitive configurations into modular, reusable components that can be chained together into a single data stream flow. Rather than a single, complex Pipeline at the Worker Group level, you can chain specialized Packs together by routing data through Routes at the Worker Group level.



Route data in and out of Packs

You can design the scope of Packs using a few patterns that consider the overall flow of data:

- **Source-side Packs:** These Packs can contain one or more Sources and any specific Pipelines needed to normalize raw data into a standard internal schema. You could also use this Pack to drop or aggregate any unnecessary fields and events to optimize incoming data. From there, the data can flow out of the Pack to Routes at the Worker Group level for further processing.
- **Destination-side Packs:** These Packs can contain one or more Destinations and any specific Pipelines that prepare or optimize the incoming data for the requirements of the downstream receiver. The Pipelines in these Packs can refine, format, and batch data so that the Destination can ingest it as efficiently and cheaply as possible. Routes are the mechanism that sends data from the Worker Group level to these Packs.
- **Pass-through processing Packs:** Data originates from a Source that exists at the Worker Group level, then passes through Routes, which sends the data to a Pack for specific enrichment step before returning to Routes for further routing.

These workflows are ideal for organizations that require functional decoupling of their configuration logic. By segregating Source and Destination logic into independent Packs, individual contributors can iterate on specific data streams in isolation, eliminating the risk of cross-contamination or breaking global configurations.

Avoid processing data twice. When planning your flow, decide exactly where heavy data processing (like masking or Regex operations) should occur. Performing the same transformation in both a Source-side Pack and a Destination-side Pack wastes CPU cycles and increases latency.

See Connect Packs to Global Sources and Destinations for more specific information about how to route data in and out of Packs.

Pack Variables

If you plan to use the same Pack across multiple environments, take some time to identify which configuration values are environment-specific. These may include:

- Endpoint URLs or hostnames.

- Port numbers.
- Region-specific settings.
- Authentication details.

To reduce duplication and make your Packs reusable across environments, use Pack variables to templatize these values. In Cribl Stream, Pack variables let you define values that can be adapted to the local environment. This allows you to:

- Set environment-relative defaults (such as `api_base_url` or `destination_bucket`).
- Avoid hardcoding environment-specific information directly in your configuration.

Keep the following guidelines in mind about variables:

- **Field binding behavior:** Pack variables are bound to specific fields. If you update the variable's value, the corresponding field will reflect the change. However, if you delete the variable, the field retains its last set value rather than reverting or clearing it. See How Field Binding Works for more detailed information.
- **Field compatibility:** Variables are only supported in certain fields, which means you can't use variables universally in every field. Fields that can accept variables have a **Variable** icon next to them.
- **Exporting secrets:** Sensitive values, such as passwords or API keys, are bundled up with a Pack and exported along with the Pack.
- **Use variables and naming conventions consistently:** When Packs include Sources and Destinations, use clear, environment-aware naming (such as `src_aws_logs_dev`, `dst_s3_prod`). Consider using Pack-level variables to template environment-specific values.

See Configure Variables and Secrets for more specific information about how to use variables.

How Field Binding Works

When you bind a Pack variable to a configuration field, you create a dynamic link. Updating the variable's value automatically pushes that change to all bound fields. However, if you delete the variable, the binding is removed and the field retains its last resolved value as a static, hard-coded setting.

To see how Cribl Stream manages this binding, select **Manage as JSON** on a Source or Destination that uses a variable. Notice that any field bound to a variable generates two distinct entries that share related names:

Field type	Description	Examples

Static field	A hard-coded value that uses the last known value. This serves as the fallback value in case of deletion.	`"host": "localhost"`
		
		`"udpPort": 3000`

Variable template field	Prefixed by `__template_`, this lists the field name and the stored variable reference.	`"__template_host": "my-host-variable"`

Field type	Description	Examples
		`"__template_udpPort": "my-port-variable"`

The `__template_` entry takes precedence over the static field. If you delete the variable, Cribl Stream removes the `__template_` entry from the JSON. Because the static field still exists, the configuration reverts to the last set value, ensuring the data stream remains functional even though it is no longer dynamic.

Never delete the static version of a configuration field from the JSON, even if a `__template_` field is present. The static field is a required component of the configuration file stored on the disk.

Sensitive Data and Secrets

Cribl Stream allows you to optionally authenticate to Sources or Destinations using TLS (Transport Layer Security), which encrypts traffic and helps ensure secure communication with upstream or downstream services. TLS certificates help verify the identity of the remote service and can also support mutual authentication when required.

When working with Packs, use the built-in secret store and named TLS certificates to manage all sensitive data. These methods are crucial for making your Packs portable and secure across different environments. You should plan to use the same process for all secrets, whether they are passwords, API keys, or security certificates.

Key considerations for your plan:

- **Secret isolation and referencing:** Unlike other configurations, sensitive values and secrets (such as TLS certificates and authentication credentials) are not bundled within Packs when you export them. Instead, a Pack only stores a reference to a secret by its name. This means:
    - You can build TLS-secured Sources and Destinations directly into your Packs.
    - The sensitive data (such as certificates) remains isolated in each Worker Group's local secret store.
- **Environment-specific secret management:** Plan to manage unique secrets for each environment (such as `dev` or `prod`) while using the same underlying Pack configuration across all environments.
- **Deployment prerequisites:** Cribl Stream stores secrets and certificates at the Worker Group level, so ensure that the secrets and certificates are properly deployed to the correct Worker Groups. For a Pack to deploy successfully with TLS-secured components, a certificate with the matching name must already exist in the target Worker Group's secret store.

Use the Cribl secret store whenever possible. This ensures different Cribl environments can hold their

own distinct secret values, without needing you to redefine the setting multiple times within the Pack.

For Sources or Destinations defined at the **global level** (outside of a Pack), Cribl Stream stores TLS certificates at the Worker Group level. Any Worker Nodes in the Worker Group can access those certificates when processing data.

For Sources or Destinations defined at the **Pack level** (inside a Pack), you can reference a TLS certificate by its name. This approach enables portable and secure configurations across environments:

- When you **export** that Pack, you can choose whether to bundle sensitive resources along with the Pack. If you choose not to bundle the resources, it does not include the TLS certificate itself, only the reference to the certificate name.
- When you **import** that Pack to a Worker Group, you can choose whether to overwrite the local certificate store or not. If you choose not to overwrite it, Cribl Stream checks the local certificate store for that Worker Group. If a certificate with the same name exists, Cribl Stream automatically binds it to the Worker Group.

This approach allows you to:

- Build TLS-secured Sources and Destinations directly within Packs.
- Reuse Pack configurations across environments without exporting certificate data.
- Maintain secure, environment-specific certificate management practices.

Pack Versioning System

The last thing to consider as part of your configuration management plan is what your strategy for updating Packs will be.

Key considerations:

- Packs support semantic versioning structures (`0.0.0`), which correlates to `Major.Minor.Patch`.
- The core principle is that each increment of the Pack's semantic version number signifies a stable, tested state of the configuration suitable for promotion to the next environment.
- You can update Packs to new versions using the **Upgrade** option.
- When you upgrade a Pack, Cribl Stream compares the current Pack version against the Pack version in the external Git repository. If the version number has increased, Cribl Stream imports the latest changes from the Git repository to the Pack in the local environment.

IMPORTANT: Commit and Push Changes to Avoid Data Loss

When you upgrade a Pack with a higher version number, it overwrites any local changes to the Pack. To prevent loss, individual Pack developers should always ensure that they have saved, committed, and

> pushed changes from their local sandbox environment to their feature branch on the external Git repository before upgrading.

With these considerations in mind, the next section describes a recommended version increment strategy

9.1.2. Create A Pack

Now that you have a clear plan for the Cribl Stream elements you want to version control, you can start building an initial Pack to test your basic configuration management workflow.

You can create a new Pack from scratch or you can create a Pack, then copy any configurations you developed in your Cribl Stream deployment into the Pack.

Best Practices for Working With Packs

When creating and working with Packs, keep these considerations in mind:

- **Keep the Pack simple at first.** For the purposes of setting up the initial Pack-based workflow, make your initial Pack simple to start and then test the version control workflow by adding more configurations in later steps.
- **Consider using Pack logos to help you remember the Pack context.** You might understandably lose track of whether you are creating Cribl Stream configurations in a local Pack context or a global context. To give yourself a visual cue that you are working in a local Pack, consider adding a logo to your Pack in the Pack settings. The logo will show up in the top breadcrumb bar so that you can clearly see when you are working in a local Pack context. When you don't see that logo, that will be your cue that you are working in a global context outside of the Pack.

Create a New Pack

Start by creating a new Pack based on the scope and goals you defined in your configuration plan:

1. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
2. Navigate to **Processing**, then **Packs** and select **Add Pack**. From the submenu, select **Create Pack**.
3. Enter a unique **Pack ID** and other details:
    - **Pack ID:** Required. Each Pack within a Worker Group must have a unique alphanumeric **Pack ID**. Some characters like hyphens or underscores are allowed, but spaces are not.

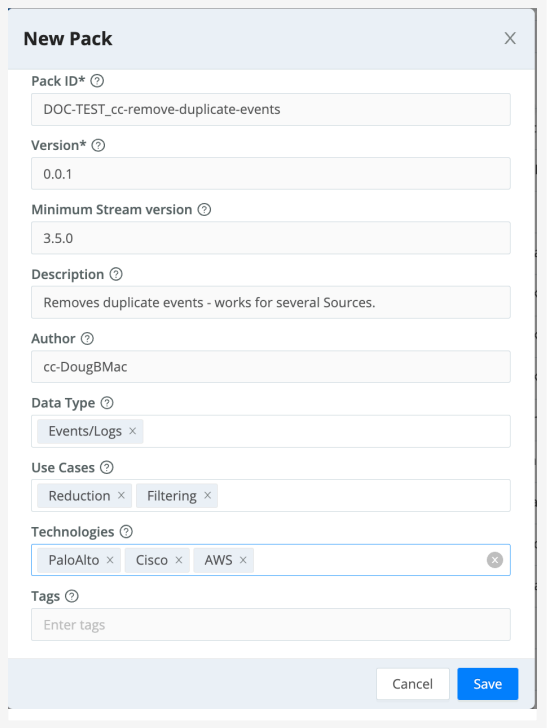
      Pack IDs must be all lowercase. If you enter uppercase letters when creating a Pack ID (such as `MyPack`), Cribl Stream automatically converts the ID to lowercase (`mypack`).

    - **Version:** Required. Set the initial Pack version. The default is `0.0.1`.
    - **Minimum Stream version:** Specify the lowest compatible version of Cribl Stream software. For example, if the Pack uses a feature that was introduced in a specific release, set the version to that

release number.

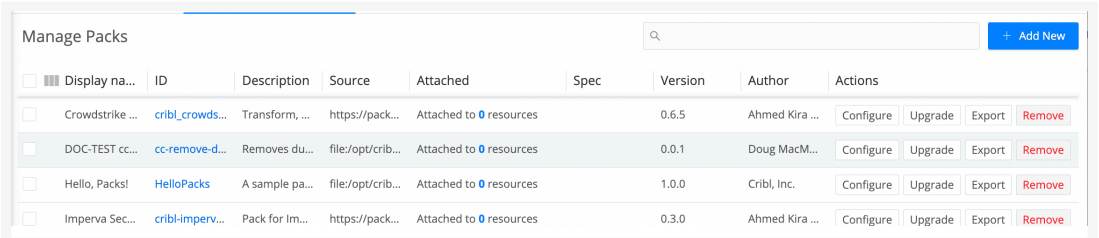
- **Description:** Add an optional description of the Pack's purpose or contents.
- **Author:** Optionally indicate the author of the Pack.
- **Data type**, **Use cases**, and **Technologies** are optional combo boxes. You can insert one or multiple keywords to help users filter Packs that you post publicly on the Cribl Packs Dispensary.
- **Tags** are optional labels that you can use to filter, search, and organize Packs.

4. Select **Save**.



Creating a Pack

5. On the **Packs** page, select the new Pack's row to open the Pack.



Manage Packs page

6. Use the standard Cribl Stream controls to configure the elements you want to include in the Pack. As you save changes in the UI, they're saved to the Pack.

Copy Resources Into the Pack (Optional)

If you developed configurations outside of a Pack context, you can manually copy configurations from one part of Cribl Stream into a Pack. This section provides steps for copying various resources into your newly created Pack. Select the tab for the type of resource you want to copy over:

Pipelines | Routes | Sources | Destinations

To copy a Pipeline into a Pack:

1. In the sidebar, select **Worker Groups**, then select a Worker Group.
2. On the **Worker Groups** submenu, select **Processing**, then **Pipelines**.
3. Locate the Pipeline you want to copy and open the Options ••• menu on its row
4. Select **Copy**.
5. Navigate to your new Pack and, in the Pack submenu, select **Pipelines**.
6. Select **Paste Pipeline**
   on the filter bar, confirm the Pipeline name and select **Save**.

Configure Variables and Secrets

To make your Packs portable and reusable across different environments, replace fixed values in your configuration with variables and secrets. This practice, known as templatization, allows your Pack to adapt to different environments without you having to manually edit it each time. See Pack Variables and Sensitive Data and Secrets for a reminder about how variables and secrets work.

As you build the Pack, design your variables with their ultimate scope in mind. The portability of Packs comes from the relationship between the Pack itself and the Worker Group context. This hierarchy allows a Pack to be environmentally aware, meaning it knows where it is and which local values it should use.

Variable	Description and Use Case	Behavior
Pack	Bundled directly within the Pack. These are best used for internal logic that should remain constant no matter where the Pack is deployed.	When you copy the Pack, these variables move with it automatically.
Worker Group	Allow you to define regional or environment-specific values that the Pack inherits once it is installed. For example, AWS regions (us-west-2 vs. us-east-1), VPC endpoints, or local port assignments.	When you port a Pack into a new Worker Group, the Pack's configuration fields use the Worker Group's context to populate these values.

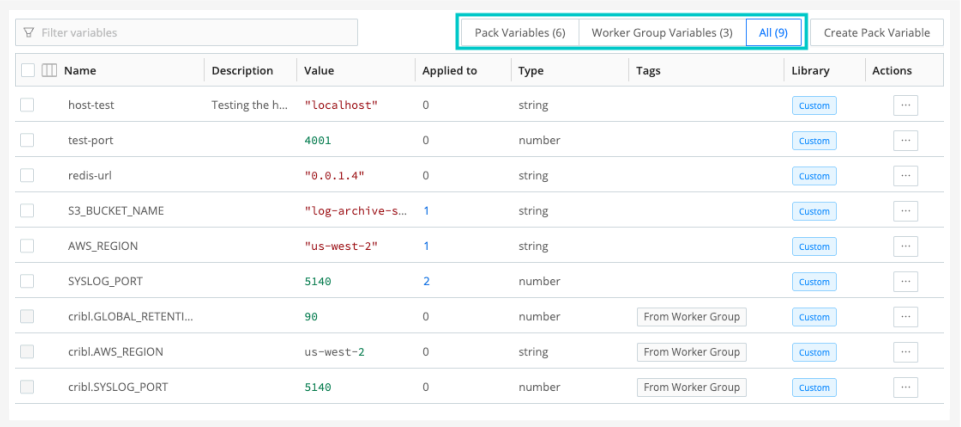
When you reference a Worker Group variable in a Pack, Cribl adds a prefix to indicate that it is a Worker Group variable. In Stream, these variables are prefixed with `cribl.`. In Edge, they are prefixed with

```
edge..
```

Create a New Variable

To create a new variable:

1. From inside the Pack context, select **Knowledge**, then **Variables**. The **Variables** UI has tabs that allow you to select the correct scope for your variable. Select either **Pack Variables** or **Worker Group Variables** depending on the desired variable scope.



Select the correct tab

2. Select **Create Pack Variable** and configure the variable settings:
    - **Name:** Enter a unique name for the variable.
    - **Description**: Optionally, enter a description.
    - **Type:** Select the type of variable you want to create. See the Variables Library reference page for a detailed explanation of variable types.
    - **Value:** Enter the value, data structure, or JavaScript expression for this variable. This input must match the format required by the selected variable type. For example, Expressions should have JavaScript syntax. The system validates the input when you save.
    - **Tags:** Optionally, add tags that you can use for filtering and grouping in the variable list view. Use a tab or hard return between (arbitrary) tag names.
3. Save the variable. The system validates the **Value** field to ensure it matches the specified **Type** when it saves.

You can also add new variables directly in the field where you want to reference that variable.

Use Variables in a Pack

Fields that can accept variables have a **Variable** icon ❖ next to them. Select this icon to open the **Add Variable** modal. You can select an existing variable from this list or create a new one.

Worker Group variables are designed specifically for configuration binding. They are not accessible using the C.vars expression in Pipeline Functions. You can only bind these variables to Source and Destination configuration fields that display the **Variable** icon ◈.

The only exception is the Redis Function, which can resolve Worker Group variables at runtime to manage external state or lookups.

Delete Variables

Before deleting a variable, identify where it is currently applied. When you delete a variable that is bound to a Source or Destination, the system performs the following actions:

- **Converts to static values:** The system replaces the variable reference with its last known value. For example, if a field was bound to an `AWS_REGION` variable with a value of `"us-west-2"`, the field will now contain the hard-coded string `"us-west-2"`.
- **Breaks environmental awareness:** Because the value is now hard-coded, the configuration will no longer adapt if you move the Pack to a different Worker Group or update the Pack or variable at the Worker Group context.
- **Prevents batch updates:** You can no longer update multiple Sources or Destinations by changing a single variable. You must manually edit each component to update its settings.

This action is permanent and cannot be undone. To restore variable binding, you must manually re-bind each field to a new variable.

Copy a Pack

After building and validating Packs, you can then copy or import that Pack to multiple Worker Groups, making small adjustments to the logic to allow for specific differences that are unique to that Worker Group. For example, you could adjust the Worker Group-level variables that account for regional differences (such as different AWS regions), VPC endpoints, or local port assignments.

When you move a Pack, it carries its entire data processing ecosystem with it, including Pipelines, Knowledge objects, and sensitive resources. This ensures that the core data processing business logic remains intact while adapting to the local requirements of its new Worker Group context. It also allows you to easily make updates to Packs and deploy those updates with minimal overhead.

To copy a Pack:

1. In Cribl Stream, navigate to the **Packs** page and open the Options ••• menu on the row of your Pack.
2. Select **Copy to another Worker Group**.
3. In the modal, define the destination and content:

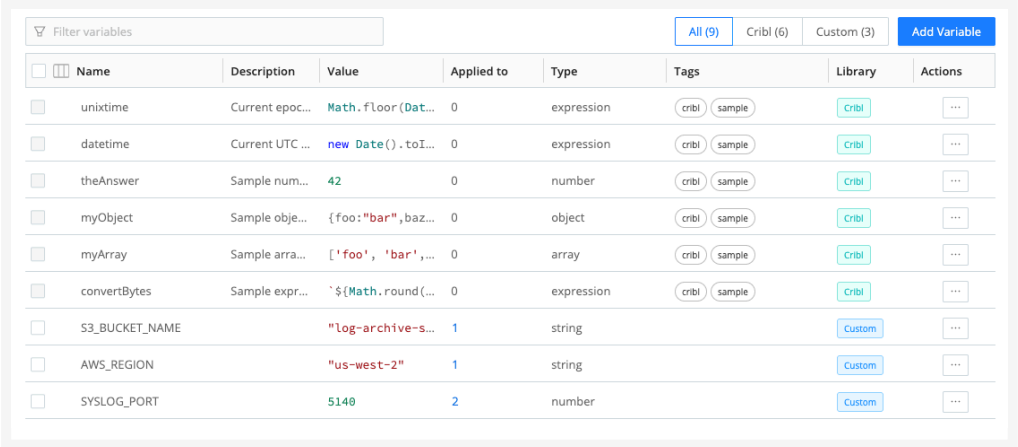
- **Export mode:** Select one of the three available export modes. See Export Mode for more detailed information about each mode.
- **Worker Group:** Type or select the target Worker Group.
- **Overwrite:** Select this option if you want to replace all configurations, Pipelines, and variables in the target Worker Group with the versions from this Pack. Enabling this will permanently overwrite any local modifications made to the Pack in the target Worker Group.
- **Include external dependencies (certificates, secrets, variables):** Bundle all certificates, secrets, and variables required for the Pack to function. This ensures the target Worker Group has the necessary credentials to run the Pack immediately. Existing resources in the target group are maintained, while only new or missing dependencies are added.

4. Select **Copy**.

Update Variables After Import or Update

When you import or copy a Pack into a new Worker Group, Cribl prompts you to configure the variables for that Worker Group context by displaying a message that indicates the import was successful. Select **Configure Variables** in this message to go to the **Variables** list. From here, you can see the list of variables available to the Pack, including its Pack variables. You can then update the variables to their custom value for that Worker Group.

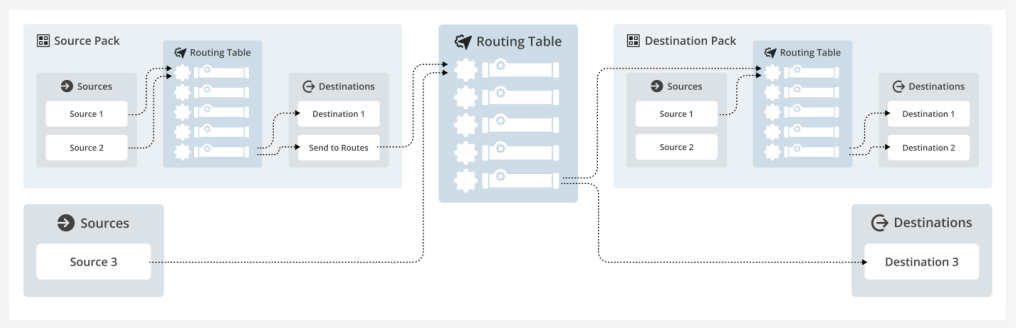
The **Applied to** column displays the number of active references for each variable. Select the value to open a modal listing all Sources and Destinations that reference this variable.



View all Pack variables

Connect Packs to Global Sources and Destinations

After you export a Pack to a new Worker Group, you can optionally set up the ability to transfer data in and out of Packs to other Sources and Destinations that are available at the Worker Group level. See Design the Pack Data Flow for an explanation of how to plan an effective data flow strategy.

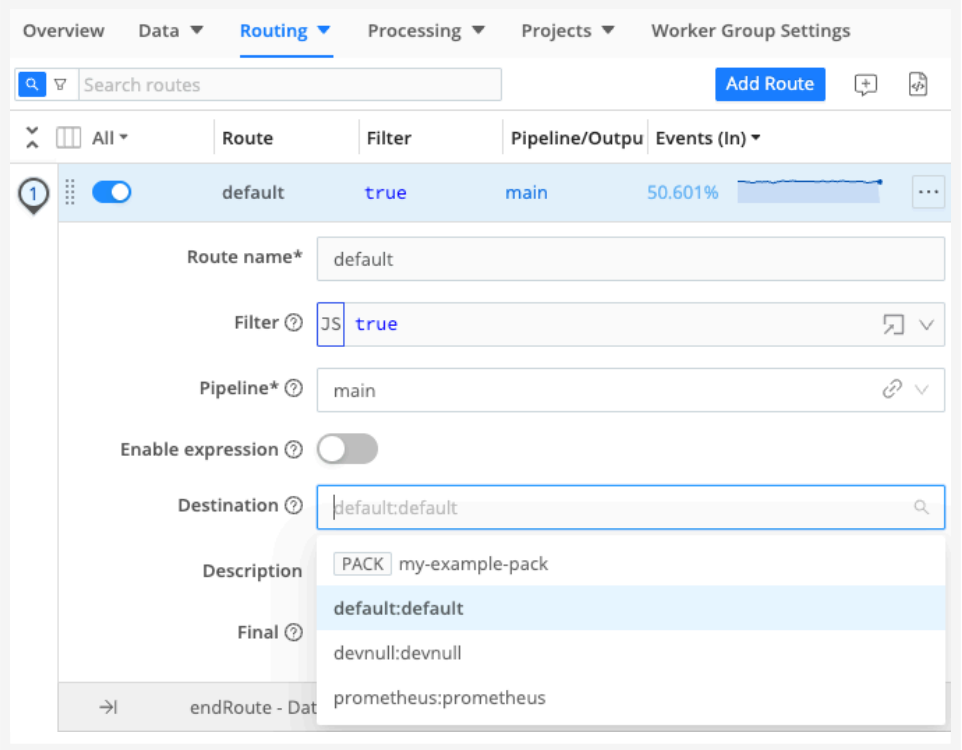


Route data in and out of Packs

Send Data From a Global Source to a Pack

To route data from a Source that exists at the Worker Group level to a Pack containing a Destination:

1. Ensure the target Pack contains a valid Destination.
2. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
3. Navigate to **Routing**, then **Data Routes**.
4. Add a new Route or expand an existing Route.
5. In the **Destination** field, select your Pack from the menu.



Box labels distinguish Packs from other objects in the Destination field

6. Verify that data is flowing as expected.

A Pack only appears as an available target in the **Destination** field if it contains at least one configured Destination. System-provided Destinations, such as Default or DevNull, do not fulfill this requirement and will not make the Pack visible in the selection menu.

Send Data From a Pack to a Global Destination

To route data originating in a Pack to a Destination that exists outside a Pack at the Worker Group level:

1. From inside the Pack context, select **Routes** from the top bar.
2. Add a new Route or expand an existing Route.
3. In the **Destination** field, select **Send to Worker Group Routes**. This sends any data matching the **Route** filter to the global Route table at the Worker Group level.
4. Verify that data is flowing as expected.

Process Data in a Pipeline Contained in a Pack

To route data from a Source that exists at the Worker Group level to a Pack containing a Pipeline:

1. Ensure the target Pack contains a valid Pipeline.
2. On the top bar, select **Products**, and then select **Cribl Stream**. Under **Worker Groups**, select a Worker Group.
3. Navigate to **Routing**, then **Data Routes**.
4. Add a new Route or expand an existing Route.
5. In the **Pipeline** field, select your Pack from the menu.
6. Verify that data is processed as expected using the Data Preview window.

A Pack only appears as an available target in the **Pipeline** field if it contains a Pipeline

9.1.3. Version Control Packs In External Git Repos

You can track and version-control Packs in an external Git repository, such as GitHub, GitLab, and Bitbucket. Tracking a Pack to an external Git repository establishes a single source of truth for Pack configurations, which allows you to track configuration changes and integrate your Cribl deployment into your company's existing change management workflows.

Best Practices for External Pack Repositories

When tracking Packs in an external Git repository, keep these considerations in mind:

- **Each Pack needs its own Git repository.** To easily import and export Packs to an external Git repository, set up a separate Git repository for each Pack you want to version control.
- **Always pay attention to the Pack version.** Whenever you develop or export a Pack, get in the habit of checking the Pack version number in the Pack settings. When you make a change, check that you increased the version of the Pack before exporting. Cribl Stream only updates Packs when the version number has increased.
- **Consider protecting branches to prevent accidental deletions.** Set appropriate merge rights on these branches as needed.
- **Document what the Pack does:** If storing the Pack in an external Git repository, include README files and comments within components that explain what kind of data the Pack handles, which environment is it intended for, and whether there are any Sources/Destinations shared across Packs.

Export a Pack to an External Git Repository

Now that you have a built your initial Pack, you can export it to an external Git repository and begin tracking it with Git version control tools. In this step, you'll set up the connection between the Pack in your local environment and the version-controlled Pack in a Git repository.

Export a Pack

The first step is to export your Pack:

1. In Cribl Stream, navigate to the **Packs** page and open the Options ••• menu on the row of your Pack.
2. Select **Export** and choose the **Merge** export mode.
3. Confirm with **Export** and save the file locally.

Add the Pack To an External Git Repository

Now that you exported your Pack, you can add it to a new Git repository. You can use a public or private repository in any common Git hosting service, such as GitHub, GitLab, or Bitbucket. Regardless of whether your repository is public or private, you need to set up a personal access token for this repository, as explained in a later step: Set up a Personal Access Token and Import the Pack.

To add the Pack to an external repository:

1. Create an empty private repository in any hosting platform.
2. Optionally, create branches for your development, testing, and production branches. For example, `dev`, `test`, `prod`.
3. Clone the new repository to your local computer.
4. Extract the `.crbl` file into the root repository directory.

Because they are gzipped, you can extract `.crbl` files using the `tar` utility. For example: `tar -xvzf WebdataStorage_0.0.1.crbl`.

5. Add, commit, and push the Pack content to your upstream Git repository.

Import and Update a Pack

In this step, you'll set up the connection between the Pack in your local Cribl Stream environment and the version-controlled Pack in a Git repository so that you can push and pull to branches from inside Cribl Stream.

You'll authenticate the connection by setting up a personal access token to connect Cribl Stream. Then, you'll make a change to the original Pack and push those changes to a development branch in your Git repository.

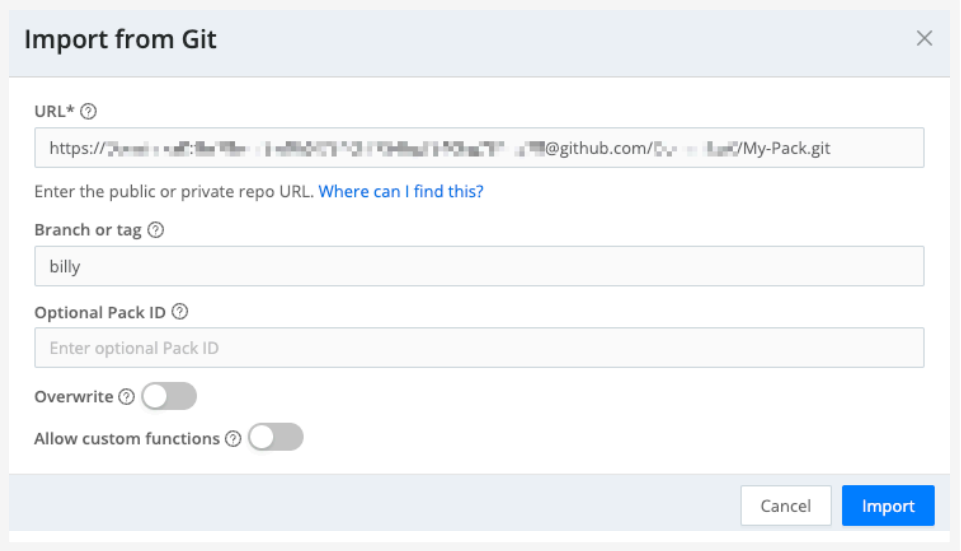
Set Up a Personal Access Token and Import the Pack

To import a Pack from Git into Cribl Stream, generate a personal access token for your Git hosting service:

1. Create a personal access token.
   Using GitHub as an example, create a personal access token with the `repo` scope.
2. In Cribl Stream, select a Worker Group and navigate to the **Packs** page.
3. Select **Add Pack** at the upper right and select **Import from Git**.
4. Enter the repository's URL. Provide your username and the token in this format: `https://<username>:<token>@github.com/<username>/<repo_name>.git`.
5. In **Branch or tag**, provide the name of your branch in the remote repository.

Ensure you import the Pack from the correct branch because this is the same branch you will push changes to. To push changes to a different branch, delete the Pack and re-import it with the

> correct branch.



Importing a Pack from a Git repository

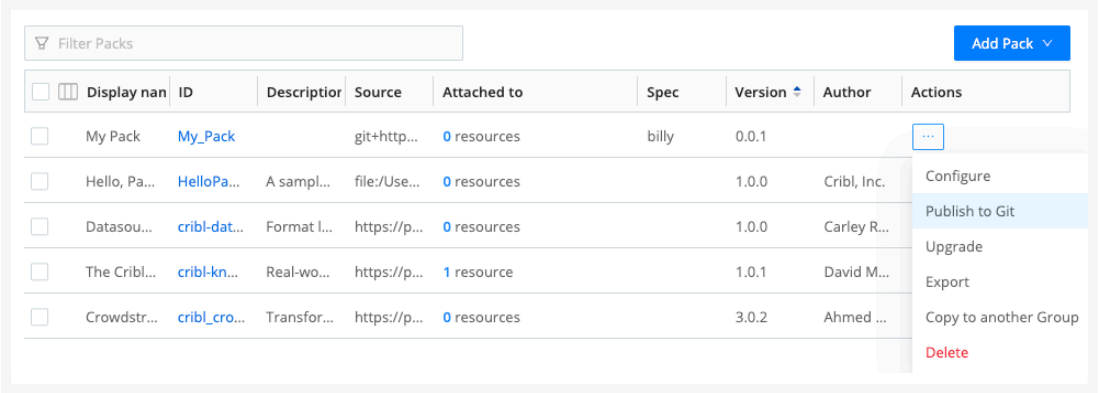
6. Select **Import** to confirm.
7. If the Pack you are importing binds variables to any fields, you will see a **Configure Variables** button in the notification about successful import. Select it to go to the **Variables** page, where you can configure the values for the variables for your deployment.
8. Check that the Pack is working correctly with your current environment based on its scope:
    - If the scope of your Pack contains Sources and Destinations, verify data flow between everything included in the Pack.
    - If your Pack only contains Routes and Pipelines, verify data flow by connecting the Pack to a Worker Group that has the necessary Sources and Destinations.

Verify data flow between Sources and Destinations both inside and outside the Pack.

Modify the Pack and Push Changes to Git

Now you can customize the Pack. Modify the Pack in some way by editing the Pack's configuration objects. For example, try making an edit to a Source, Destination, Route, Pipeline, Event Breaker, or Knowledge Object. For your first time using an external Git repository, keep the changes small. Then, publish your changes to the remote Git repository:

1. Navigate to the **Packs** page and open the Options ••• menu on the row of your Pack.



Publishing a Pack to Git

2. Select **Publish to Git**, and in the modal window, confirm with **Publish to Git** again.
3. Your Pack's branch in the remote repository updates with your changes.

The repository maintainer can now synchronize the main branch with your branch in their preferred way (for example, by creating a pull request).

When you use **Publish to Git**, all files in the repository are overwritten except for hidden files and directories located at the repository root. This means that any non-hidden files not managed by Cribl (such as documentation, scripts, or CI/CD workflows) will be deleted.

To preserve important files, ensure they are stored as hidden files by prefixing their names with a `.` For example use `.github` for GitHub workflows.

9.2. Packs Publication Standards

This page outlines the process for Cribl Community members to publish Cribl Stream, Edge, and Search Packs to the Cribl Packs Dispensary. It also lists standards that apply to all publicly available Community Packs.

Publication Overview

Publishing your Pack is a five-stage process:

1. **Prepare**
    - Prior to starting development, read this topic's Pack Review Summary and Pack Review Checklist. These resources will help you create a Pack that can easily pass the review process. Decide on the problems you're trying to solve, and determine the dataset(s), data senders, and data receivers you're going to work with.
2. **Develop your Pack**
    - You'll generally develop Packs in your own Cribl environment, using saved sample data files to build out Stream/Edge Pipelines, or using Search Datasets to build Search Dashboards. During this stage, you might consider working collaboratively within Cribl's Community Slack `#packs` channel. When the Pack is ready for review, export it as a `.crbl` file, using the Cribl UI.
3. **Test your Pack**
    - Cribl strongly recommends that you test the Pack yourself, in your own environment. To verify that all content was exported correctly, import your Pack into a fresh Cribl environment. For Stream or Edge, this might be a newly created Docker instance, Worker Group, or Fleet. For Search, this might be a different Cribl Search Workspace. Upload the Pack, then use the Pack Review Checklist to verify that the Pack will pass review.
4. **Submit your Pack for review**
    - Sign into, or create an account on, the Cribl Packs Dispensary site. You can use your Cribl.Cloud account's email address to create the Cribl Packs Dispensary account.
    - If you don't have a Cribl.Cloud account, Cribl automatically creates a new one for you when you create your Cribl Packs Dispensary account.
    - Once you sign in, you'll see the Dispensary page with the **Publish Pack** button.

Packs Dispensary controls

- Click **Publish Pack**, and then click **Upload Pack**.
- Select the `.crbl` Pack file you want to publish, and click **Submit**. You will be prompted to agree to the terms of Cribl's Pack Developer Agreement. As a Pack author, you must electronically acknowledge that you have read the Agreement, and that you intend to adhere to its requirements.

Once you've uploaded the Pack, Cribl will review it. The review process typically takes a week or more. Once your Pack is reviewed, you'll receive an email from `packs@cribl.io` either informing you that it was accepted, or providing the reason(s) why it was rejected. (In the latter case, you can address the specified issues and resubmit.)

5. **Once your accepted Pack is published to the Cribl Packs Dispensary, it's time to celebrate!**

Pack Review Summary

To ensure consistency across published Packs, Cribl evaluates all submitted Packs against the Pack Review Checklist. Any Pack that fails to satisfy the checklist's requirements will be rejected. (As noted above, we will notify you of the basis for any rejection, so that you can remedy the issue and resubmit.)

Cribl recommends:

- Check your Pack against the checklist **before** trying to submit the Pack. This way, you can avoid having your Pack rejected for something easily fixed upfront.
- For Stream/Edge Packs, you can install the Syslog Pre-processing Pack as a reference - it includes clear examples of each item on the checklist.
- For Search Packs, you can install the Search Palo Alto Networks Traffic Pack as a reference.

Pack Review Checklist

This is a checklist for Packs you intend to submit to the Cribl Packs Dispensary. The following sections list setup requirements for product-specific Packs, and then for all Packs.

Stream/Edge Packs - Requirements

This section's requirements apply to Packs for Cribl Stream and Cribl Edge.

Sources and Destinations

If you include a Source (or Stream Collector) in your Pack, you must also include the intended Destination, so that the Routes table is wired up to receive and deliver data all within that Pack. If you include Sources (including Collectors) or Destinations inside a Pack, you must clearly state this in the README, and your

README should warn users not to insert this Pack (as a Pipeline) into the global Routes table outside of the Pack.

The benefit of tying everything together in a Pack - Sources, Pipelines, Routes, Destinations - is enabling users to take advantage of an end-to-end telemetry stream configuration. This way, users need to update only selected values (for things like endpoints and authentication details) in order to run the Pack.

Routes

A submitted Stream or Edge Pack must include at least one Route in addition to the default Route.

- A Pack's Routes page must have a Route, with an appropriate filter, routing to a Pipeline.
- The filter should match **only** the set of data that the Pipeline is designed to process. The filter must **not** be set to `true`.
- If the Pack deals with multiple data types, each data type should have its own Route/Pipeline pair. (For a good example, look at the Palo Alto Networks Traffic Pack.)
- The Pack must not use the `default` Route to send events to a Pack-specific custom Pipeline. The `default` Route may have a Pipeline set to `main`, `devnull`, or `passthru` to handle events that do not match the Pack filter(s).
- Each Route must include a short description of the data sample being matched.
- Pairs of Routes and Pipelines should have matching names.

Pipelines

A Pack submitted must include at least one Pipeline in addition to the default Pipelines (which are `apache_sample_pipeline`, `devnull`, `main`, `passthru`, and `prometheus_metrics`.).

- If the Pack deals with multiple data types, there must be one Pipeline (and one Route) for each data type. (For a good example, look at the Palo Alto Networks Traffic Pack.)
- The Pack must not modify the default Pipelines. Leave these Pipelines unchanged.
- Every Pipeline must begin with a Comment Function that gives an overview of the Pipeline's operation as a whole. This Comment is a synopsis of what the entire Pipeline will be doing. For an example of the expected level of detail, refer to the Syslog Pre-processing Pack.
- If Function Groups are used within the Pipeline, each Function Group must begin with a Comment Function that gives an overview of what will happen within that Function Group.
- Every Function must have a Description that briefly explains exactly what is happening in that particular Function. During review, change the Pipeline view option to display the Description for each Function.
- Using the sample files provided with the Pack, validate that the Functions work as detailed in the Comment at the top of the Pipeline.

Sample Data Files

The Pack must include at least one sample data file for each of its Pipelines.

Sample data files:

- Must **not** have an expiration date.
- Must **never** include PII, customer data, or customer hostnames.
- Must be named to make it obvious which sample file(s) apply to each Pipeline.
- Must contain samples that are adequate for a customer to use to review the operation of a Pipeline. To allow performance analysis of the corresponding Pipeline, a sample should include 20 or more events, with 100 or more recommended.
- May be reused across Pipelines, but it is preferable to include multiple sample files, each specific to one Pipeline's Route and filter.
- Must work as expected on a clean installation of the Pack. This applies to all sample data files included in the Pack - Pack developers should test this before submitting the Pack.

Search Packs - Requirements

This section's requirements apply to Packs for Cribl Search.

Lookups

Ensure that lookup files contain no sensitive data. If Cribl review discovers any sensitive information in a submitted lookup file, Cribl will automatically reject the Pack.

When formatting references to the lookup table file, use this syntax:

```
pack(<pack_id>).<lookup_file_name>
```

Copy

When referencing the lookup table in queries, use this syntax:

```
dataset="cribl_search_sample" datatype="aws_vpcflow"
 | lookup pack(cribl-search-aws-vpc-flow-logs).protocol_codes on protocol=protocol_cc
 | summarize count=count() by protocol_name
 | order by count desc
```

Copy

Datatypes

Datatypes are important for Cribl Search Packs, because they provide the rules for how to process and search raw data. Adding Datatypes to Search Packs provides users with important parsing rules that enhance the overall search and Dashboard experience.

Essentially, Datatypes give structure to unstructured data, making it easier to query and explore vendor-specific datasets. Cribl strongly recommends that you implement Datatypes within Search Packs when needed. As a model, refer to the **Palo Alto Datatypes** ruleset that's built into Cribl Search.

All Products - Requirements

This section's naming and metadata requirements apply to all Packs.

Naming, Logo, and Tags

Every Pack must have both a unique **Pack ID** and a **Display name**.

When you view Packs in the Dispensary, the Display name appears above the Pack ID. The Pack ID must follow a specific format of sequenced tokens:

`<source>-<productArea>-<dataVendor>-<dataType>`.

Here:

- `<source>` identifies who contributed the Pack, not a data source. This is either `cribl` for Cribl-developed Packs, or `cc` for Community-contributed Packs. (Packs not satisfying this requirement will be immediately rejected.)
- `<productArea>` is one of `stream`, `edge`, or `search`.
- `<dataVendor>` and `<dataType>` identify (where applicable) the integration; and (where applicable) the focus of the transformations or configurations provided in the Pack. Examples for this pair of tokens might be `okta-rest`, or `palo-alto-networks-traffic`, or `exabeam-fortigate-firewall`.

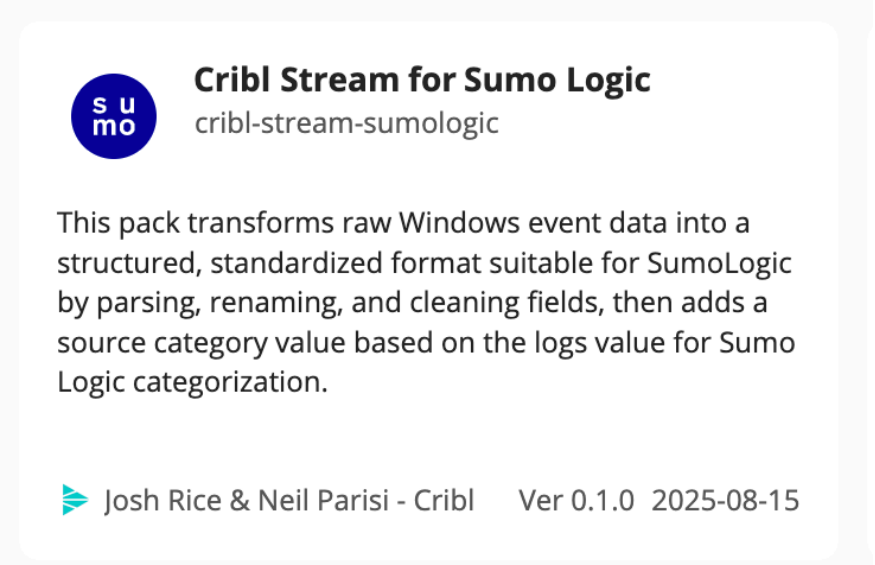
Here are examples of complete Pack IDs from the Cribl Packs Dispensary (where you can find many additional examples):

- `cribl-stream-sumologic`
- `cribl-search-cloudtrail-activity`

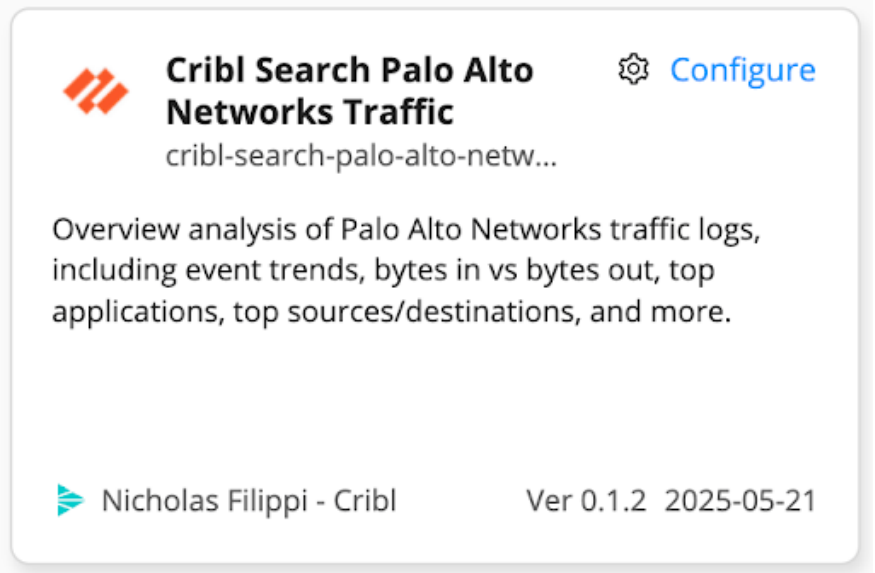
The Display name is a free-text summary of the purpose of the Pack. As examples:

- `Cribl Stream for Sumo Logic` is a Display name, while `cribl-stream-sumologic` is the corresponding Pack ID.
- `Cribl Search CloudTrail Activity` is a Display name, while `cribl-search-cloudtrail-activity` is the corresponding Pack ID:

Detailed requirements for both fields follow these graphical examples from Dispensary tiles.



Display Name and Pack ID - Stream or Edge example



Display Name and Pack ID - Search example

Pack ID Requirements

The Pack ID:

- Must start with either `cc-` (Community-contributed) or `cribl-` (Cribl-contributed). Packs not satisfying this requirement will be immediately rejected.
- Should (by convention) be all lowercase, using hyphens (`-`) as delimiters.
- Should **not** include the word "pack."

Display Name Requirements

The Display name is a friendly, human-readable version of the Pack name that:

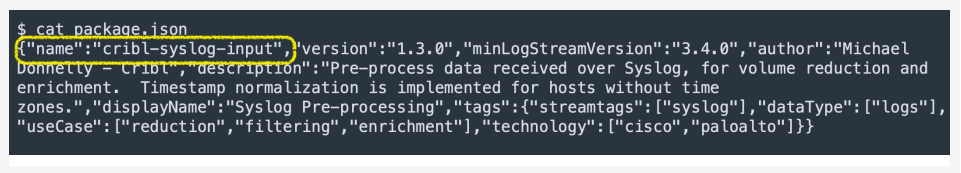
- Omits `cc-` and `cribl-`.
- Uses spaces where appropriate, and uses conventional capitalization.
- Matches the vendor's preferred spelling, spaces, and capitalization of any referenced company or product name.

Within a Pack, you can view or edit the Display name on the **Pack Settings** tab.

How to Review the Pack ID

This is the same procedure that the Cribl will perform when reviewing your Pack:

1. Extract the `.crbl` tarball that was submitted to the Dispensary:
   `tar zxvf <your_filename>.crbl`
2. View the `package.json` file:
   `cat package.json`
3. Check the value of the `name` field. This should be the Pack ID that you want.



Verifying the Pack ID

Pack Settings

You must review your **Pack Settings** to verify that the Pack satisfies all of the following criteria.

Display Name

A friendly, human-readable version of the Pack name, as defined above.

Version

The version number in the submitted Pack must have a corresponding entry in the README Release Notes. This number should be in SemVer (Semantic Versioning) format.

- The minimum version for initial publication is `0.1.0`. A Pack with a lower version **will be rejected**.
- A release that is considered "beta quality" should be designated as version `0.9.0`.

Description

Use this as a brief "hook" of one to two sentences. Example:

`This pack for Syslog inputs will reduce volume, and address timestamp normalization for Syslog senders that omit timezones.`

Author

Community-contributed Packs may include the author's name, a company name, or both.

All three of the following examples are valid:

1. `Jon Smith`
2. `Bazookatron`
3. `Jon Smith - Bazookatron`

Cribl-published Packs must use author's full name, plus `- Cribl`. Example: `Michael Donnelly - Cribl`.

Data Type, Use Cases, Technologies

Select all that apply.

Tags

Tags are optional but recommended, especially where multiple Packs are intended to work together. Tag examples include: `OCSF`, `AWS`.

Logo

The logo is optional, but recommended. You can include a logo associated with the technology addressed by the Pack. As examples: a Windows logo for Windows events, or an AWS Cloudwatch logo for AWS Cloudwatch data.

You can set the logo via **Pack Settings** > **Display**.

README

The `README` is the documentation others will use in selecting and deploying the Pack. The `README` requirements ensure that others will be able to successfully deploy the Pack.

The default sections for the README, as created when you click **Create Pack**, do not match the required and optional sections in the checklist. You will need to delete or replace certain sections.

On the Pack's **Pack Settings** > **README** tab, validate that the README text is correct. The following rules apply to all sections of the README:

- Proper grammar will help the users of your Pack. Use a grammar and spelling checker to make sure the README is clear and readable.

- If the README includes any text that looks like a placeholder, the Pack will be rejected.
- Trademark or product names must be spelled and capitalized exactly as the company that owns them would dictate. This requirement also applies to Pack settings, inline comments, and other uses. Examples: Amazon (not amazon), GCP (not Gcp), CrowdStrike (not Crowdstrike).

The README **must** include all of the sections enumerated here, as items **a** through **f**. The README may also include additional sections, as necessary to explain the resources you've included in the Pack.

a. About This Pack

Think of this initial section as the first paragraph of an essay, or as the only paragraph a person will read when deciding to use the Pack. It must clearly state what the Pack does. Start with the benefits of using the Pack. In sum, this section should answer the question, **Why should anyone use this Pack?**

If the Pack is tied to a specific data source, dataset, or destination, spell that out clearly here - even if that source or destination is in the Pack name.

b. Deployment

This **required** section must include all instructions necessary to get the Pack up and running. If the Pack requires any configuration after initial installation - but before it's put it into production - you must specify the configuration steps here.

Stream/Edge Pack Deployment

Filters: If applicable, say what filter must be used to route data to the Pack, or explain how to configure the filter. A filter based on `__inputId` is much more efficient than a match against `_raw`.

Sources: If the Pack requires a specific Source (or Stream Collector), include information on how to configure that Source. At a minimum, this needs to cover setting up the Source within Cribl. In some cases, this might include information on how to set up an external environment for collection by Cribl as well.

Destinations: If the Pack requires a specific Destination, include information on how to configure that Destination. Does it need a post-processing Pipeline tied to the Pack?

Pipelines: If the Pack includes Functions (within a Pipeline) that need to tailored, enabled, or disabled, include a full explanation and instructions.

Lookups: If the Pack includes Lookup files that need to be tailored, you must specify the configuration steps here.

Search Pack Deployment

Dataset Provider: If the Pack works with a Cribl Search built-in Dataset Provider, specify it. If the Pack requires adding a new Dataset Provider, provide specific instructions.

Dataset: Provide instructions on configuring compatible Datasets.

Datatype: Specify Datatypes to apply - whether built into Cribl Search or included in the Pack.

Macros: Provide instructions on using any custom Macros included in the Pack.

c. Upgrades

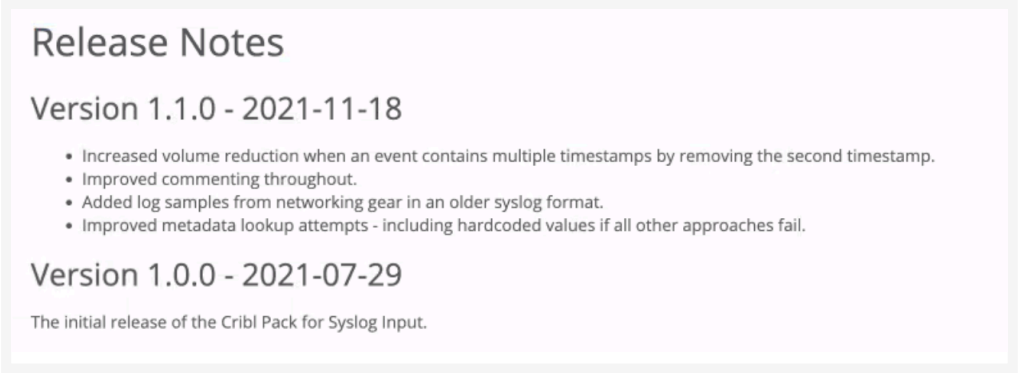
This **optional** section is strongly recommended if you expect others to customize their copy of the Pack and later upgrade the modified Pack.

- If included, this section should cover the usual set of problems with Pack upgrades as described in Upgrading an Existing Pack, and should explain the specifics of upgrading this particular Pack.

d. Release Notes

This **required** section is a list of release notes where each release note:

- Starts with the version number and date of release, separated by a space, a hyphen, and a space. For example: `Version 1.2.0 - 2022-07-11`.
- Continues with bullet points that describe what has changed and/or is new in the release.



Release notes

Release notes must adhere to the following rules:

- Designate the initial release version `Initial release.`
- The most current version listed must exactly match the Pack version in **Pack Settings** > **Pack Info**.
- Release notes versions must be listed starting with the most recent (in descending chronological order).

e. Contributing to the Pack

This **required** section explains how to contribute to the Pack.

- The information provided must be **actionable**. A section reading `[Contributor instructions go here.]` would be rejected.

- Cribl recommends that you refer potential contributors to Cribl's Community Slack. For example, Cribl-created Packs read as follows:

```
To contribute to the Pack, please connect with us on [Cribl Community Slack](htt
```

Copy

f. License

This **required** section must include text that designates **and links to** a legitimate license.

- Community-contributed Packs must include a license that allows for use by the general public, such as Apache 2.0, MIT, or GNU.
- Built-by-Cribl Packs must use the following snippet:

```
This Pack uses the following license: [Apache 2.0](https://github.com/criblio/ap
```

Copy

10. Reference

A reference for Cribl Stream, including:

- Cribl Expressions
- CLI
- Environment variables
- Config files

For reference documentation for the Cribl API, see the API Reference.

10.1. Knowledge Objects

Knowledge objects are reusable, customizable objects that help you scale and maintain your Cribl Stream more easily. You can reference these objects in multiple places, such as Functions or Pipelines. With these objects, you can change a single value in a central location and then populate that change everywhere you use that object. Access these objects in **Processing** > **Knowledge**.

Knowledge Object	Description
Lookups	Reference tables that you can apply to raw data events in your Pipeline for routing and enrichment.
Event Breakers	Event Breaker Rulesets are ordered collections of event-breaking rules that help you define the boundaries and structure of raw log data. Once you have converted unstructured raw data into structured events, you can then send them through Pipelines for further routing and data processing.
Parsers	Parsers for use in the Parser Function.
Global Variables	Store values you can reference elsewhere in Cribl Stream.
Sample Files	A data sample (often called a data capture) is a small, representative subset of live or stored data from a given data stream. When you creating samples as Knowledge objects, these files can be accessed by Routes, Pipelines, Event Breakers, and Datagens at the Worker Group or Pack level.
Regexes	Regular expressions for use in regex-related Functions.
Grok Patterns	Grok pattern files for use in the Grok Function.
Schemas (JSON)	JSON schemas for validating events.
Parquet Schemas	Schemas for writing out data in the Apache Parquet format.
Database Connections	(**Cribl Stream only**) Reusable configuration objects that enable Database Collectors to access and retrieve data from a MySQL, Oracle, Postgres, or SQL Server database.
HMAC Functions	(**Cribl Stream only**) Keyed hashing functions that you use with REST Collectors.
AppScope Configs	Configurations for AppScope (deprecated).

10.1.1. Variables Library

Variables store values you can reference elsewhere in Cribl Stream. These variables allow you to change a single value and then populate everywhere that variable is referenced.

Variables are an important component for scaling your Cribl deployment and improving ease of maintenance. Typical use cases for variables include:

- Storing a constant that you can reference from any Function in any Pipeline.
- Storing a relatively long value expression, or one that uses one or more arguments.
- Converting abstract, complex logical expressions into a reusable variable that has a human-readable name for easier maintenance.

Cribl Stream supports a variety of variable types:

- **Number, String, Boolean:** Store simple, static configuration values for reuse and ease of updating (such as threshold limits or names).
- **Encrypted String:** Store sensitive strings (like API keys or passwords). The value is encrypted at rest but retrieved as plain text when referenced, including when you are logged into the Cribl interface.
- **Object:** Store complex, structured data or key-value dictionaries, often used as an alternative to small lookup files.
- **Array:** Store a list of items (such as IP ranges or required field names).
- **Expression:** Define complex, repeatable logic that requires calculated output, especially when the logic needs to accept arguments.

You access all variables using the standard syntax `C.vars.<variableName>`. See Cribl Expressions for more information about `C.vars` and other Cribl Expressions.

This page explains the general use case for variables. For more information about using variables in Packs, see:

- Pack Variables
- Configure Variables and Secrets

Limitations

When multiple variables share the same name, Cribl Stream resolves the value based on scope, moving from the most specific (inner) scope to the most general (outer) scope. In other words: local scope values win over broader scopes.

For example, a variable defined inside a Pack (which has a local scope) will always override a variable with the same name defined in the Worker Group settings.

Create a Variable

Creating a new variable has three main phases:

1. Define the variable scope.
2. Configure the variable.
3. Reference the variable in a Function or Pipeline.

Define the Variable Scope

Variables are scoped to different levels of the application. You can only use a variable in the scope in which it is defined.

When creating a new variable, you determine its scope based on where you access and create the variable:

Worker Group | Pack | Single-Instance Deployment

To access and create variables specific to a given Worker Group:

1. Select **Worker Groups** from the sidebar, and choose a Worker Group.
2. On the **Worker Groups** submenu, select **Processing**, then **Knowledge**.
3. In the sidebar, select **Variables**.

You cannot access variables from a Pack in a Worker Group.

Configure the Variable

After ensuring you have accessed the variable from the correct scope, create a new variable:

1. Select **Add Variable**.
2. Configure the variable settings:
    - **Name:** Enter a unique name for the variable.
    - **Description**: Optionally, enter a description.
    - **Type:** Select the type of variable you want to create. Options include:

Type	Explanation	Use Cases
String	A sequence of characters (text).	Can centralize environment identifiers, file paths, URL prefixes, or standard naming conventions. If a string or name changes depending on its context (such as a different environment), you only have to update it in the variable and it populates to configurations in that environment.
Number	For numeric values, including integers (whole numbers) and floating-point numbers (decimals).	Ideal for defining thresholds or limits that might need tuning later.
Encrypted string	Stores a string value that is automatically encrypted when saved to the configuration file on disk (YAML). However, within the application at runtime, it is decrypted and treated as a standard string.	Use for storing sensitive credentials like API tokens, secret keys, or passwords. This ensures that if someone inspects your configuration files or git commits, they only see the encrypted hash, not the plaintext secret.
Boolean	Represents a binary logical value that resolves to `true` or `false`.	Use as a switch to enable or disable specific logic paths, detailed logging, or experimental features across your entire deployment without editing individual Pipelines.
Array	An ordered list of values (strings, numbers) stored in a single variable.	Can help maintain long lists of items, such as IP ranges, required field names, known good or bad hostnames, and more.
Object	A structured collection of key-value pairs (dictionaries).	Functions as a lightweight, in-memory lookup table.
Expression	A snippet of JavaScript code that is evaluated only when called. Expressions can accept arguments, allowing them to act like reusable custom logical functions.	Allows you to encapsulate complex, multi-condition logic (like regex matching) into a single function with a human-readable name.
Any	A flexible container that accepts a value of any variable data type (string, number, expression) without enforcing strict validation rules.	While this variable type is flexible, it removes the safety net of UI validation. Use specific types (like number or boolean) whenever possible for better reliability.

- **Value:** Enter the value, data structure, or JavaScript expression for this variable. This input must match the format required by the selected variable type. For example, Expressions should have

JavaScript syntax. The system validates the input when you save.

- **Tags:** Optionally, add tags that you can use for filtering and grouping in the variable list view. Use a tab or hard return between (arbitrary) tag names.

3. Save the variable. The system validates the **Value** field to ensure it matches the specified **Type** when it saves.

Reference the Variable

After creating the variable, you can start calling it in Functions, Pipelines, and Packs. You can reference a variable in fields that let you use custom JavaScript code. See the introduction of Build Custom Logic to Route and Process Your Data for examples and images of JavaScript-enabled fields.

Some variable types require additional inputs or arguments:

Variable Type	How to Reference	Examples
Number, String, Boolean	Use `C.vars.variable_name` in any field that lets you use custom JavaScript code.	`C.vars.my_variable`
Encrypted String	Use `C.vars.variable_name` in any field that lets you use custom JavaScript code.	`C.vars.api_key`
Object	Reference these as you would any other object or array using standard JavaScript object notation.	`C.vars.lookup_map.key` `C.vars.my_config.region`

Array	Reference these as you would any other object or array using standard JavaScript object notation.	`C.vars.varname[0]`
		`C.vars.varname.nested`
		`C.vars.host_list[0]`

Expression	Call it with the arguments in parentheses.	`C.vars.varname(arg1,arg2)`

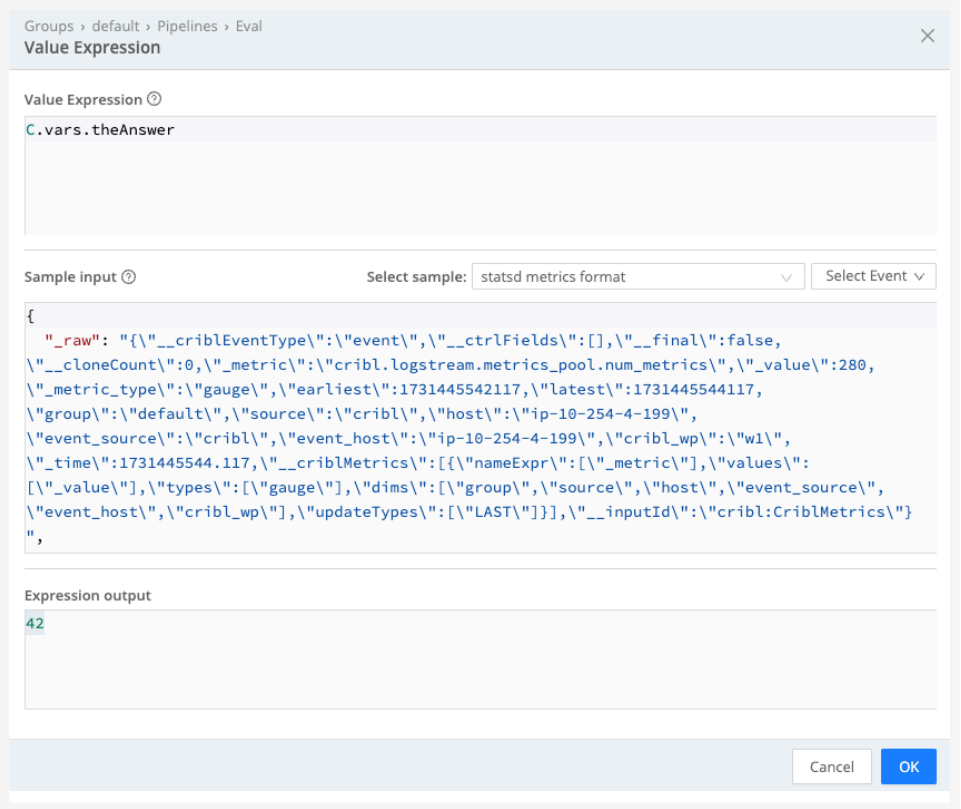
To illustrate the workflow for referencing a variable, imagine a scenario in which you want to add a field named `myField` to all incoming events. This field value will get referenced in multiple Pipelines and may possibly change, which is why you decided to make it a variable named `theAnswer`. For now, you want to set it to a static number value of `42`.

After creating the number variable named `theAnswer` with a value of `42`, you could then reference it in a Pipeline:

1. Add the Eval Function. (The Eval Function adds, modifies, or removes event fields.)
2. Select **Add Field** to add a new field.
3. Configure the new field to reference the `theAnswer` variable:
    - **Name:** myField

- **Value Expression:** `C.vars.theAnswer`

4. To validate the value, select **Advanced Mode** in the **Value Expression**.
5. Confirm that the **Expression output** is `42` as expected.



Value Expression Advanced Mode

You can then reference this variable in multiple Functions or Pipelines as needed. When you change the variable value, it will update anywhere the variable is referenced.

> Worker Group variables are designed specifically for configuration binding. They are not accessible using the C.vars expression in Pipeline Functions. You can only bind these variables to Source and Destination configuration fields that display the **Variable** icon .
>
> The only exception is the Redis Function, which can resolve Worker Group variables at runtime to manage external state or lookups.

Use Case Examples

You can personally test these examples within your own Cribl Stream deployment to understand how Cribl evaluates variables.

Expression Example

This example shows an expression variable that executes JavaScript logic to get the current time and assign it to a field named `nowEpoch`. This expression does not require any input arguments, so it gets left blank.

Variable settings:

- **Name:** `epochTime`
- **Type:** Expression
- **Value:** `Date.now()/1000`

You could then add an Eval Function to a Pipeline:

- **Field Name:** nowEpoch
- **Value Expression:** `C.vars.epochTime()`

Object Example

This example demonstrates how to use an Object variable as a central configuration store (dictionary). You can reference specific keys within the object using dot notation. In this scenario, the goal is to assign the field `contact_email` a value that is stored in the `admin_email` key of a global configuration object.

Variable settings:

- **Name:** `my_app_config`
- **Type:** Object
- **Value:**

```
{
  "env": "production",
  "retry_count": 3,
  "admin_email": "ops@example.com"
}
```

Copy

You could then add an Eval Function to a Pipeline:

- **Field Name:** contact_email
- **Value Expression:** `C.vars.my_app_config.admin_email`

Array Example

This example shows how to use an Array variable to store a list of values (like hostnames) and check if a specific event field exists within that list. In this scenario, the goal is to drop the event if the host field matches any entry in the `blocked_hosts` list. This will be accomplished with logic that returns `false` in a filter.

Variable settings:

- **Name:** `blocked_hosts`
- **Type:** Array
- **Value:** `["bad-host-1", "test-server-99", "retired-asset"]`

You could then add a Drop Function to a Pipeline with a **Filter** value set to: `C.vars.blocked_hosts.includes(host)`.

Additional Examples

There are endless possibilities for how to use variables in Cribl Stream. See more examples in the Goat-to Resources and continue sharing use cases with the rest of the Cribl Community on Slack.

Goat-to Resources

10.1.2. Regexes

Regexes offer predefined regular expressions that you can use in regex-related Functions.

Cribl ships with a collection of commonly-used regexes and gives you the possibility of adding your own custom ones.

Regexes let you quickly identify and filter data that conforms to specific patterns, including commonly-used patterns like credit card numbers, IP addresses, or personal identification numbers. You can use expressions from the regex collection in several Functions, such as Regex Filter, Regex Extract, or Mask, in configuration for certain Sources and Destinations, as well as in other settings.

Prerequisites

You configure regexes per Worker Group, or within a Pack. You can only use the regex within its scope, that is, in the Worker Group or Pack you defined it in.

Regexes defined in an Edge Fleet are subject to Fleet inheritance.

Regex Priority

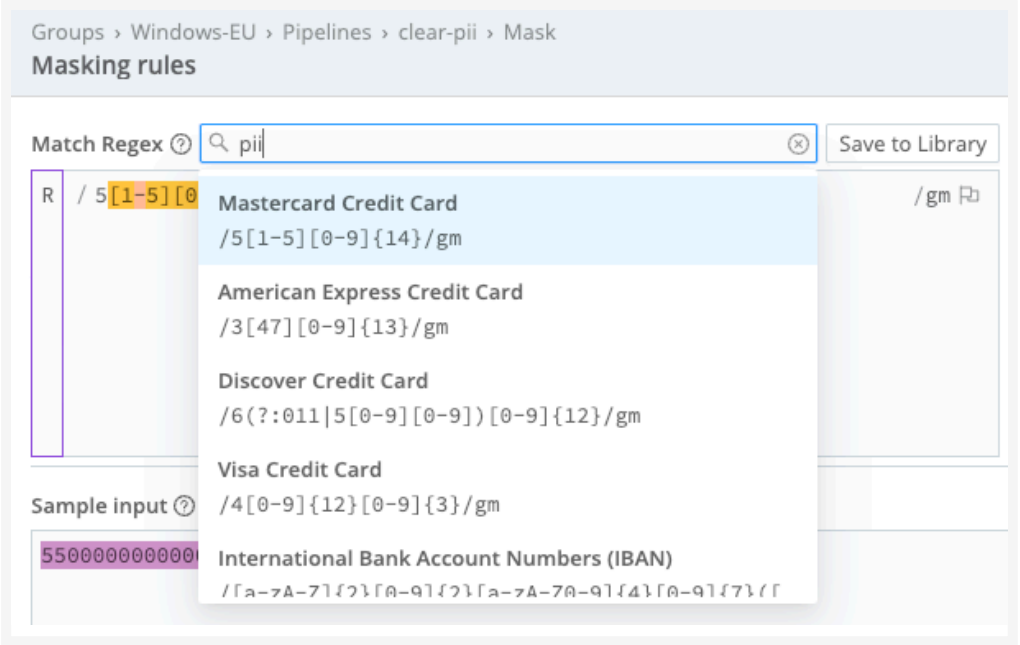
Built-in Regexes are listed with the **Cribl** tag, while those you add are tagged with **Custom**. When you edit any built-in regex, it changes to a **Custom** one as well.

If two regexes with the same ID exist, the custom pattern takes priority in listings and search. For example, if a Cribl-provided pattern and a Custom one are both named `ipv4`, the one from Cribl will not be displayed or delivered as a search result.

Use a Regex

To insert a regex into a field that supports it, for example in a Function, select the Advanced mode or Edit icon beside that field.

In the resulting modal, regexes will appear as typeahead options. Select a pattern to paste it in. You can then use the pattern as-is, or modify it as necessary.



Using a built-in regex

Create a Regex

To create a new regex:

1. In the sidebar, select **Worker Groups** and choose a Worker Group.
2. In the **Worker Groups** submenu, select **Processing**.
3. If you are adding regex to a Pack, select **Packs** and choose your Pack.
4. Select **Knowledge**, then **Regexes**. You will see a list of prebuilt and custom regexes.
5. Select **Add Regex**.
6. Enter an **ID** for the regex and an optional **Description**.
7. In **Regex pattern**, enter the pattern, using the JavaScript (ECMAScript) regex flavor. Use the flag icon to select options for regex matching.
8. In **Sample data**, you can enter a sample text to check the expression against. Matches will be indicated visually.

Adding a custom regex

9. Optionally, add one or more **Tags** that will help you search for this regex.
10. Select **Save**.

Examples

Mask SSN Numbers

If you need to mask personal identification numbers, you can use a build-in regex in conjunction with the Mask Function.

Add a Mask Function to your Pipeline. In the Function's **Masking Rules** modal, search and select the **US Social Security Numbers** regex in the **Match Regex** box. In **Replace Expression**, select the string or expression you

10.1.3. JSON Schemas

Cribl Stream uses JSON schemas for validating JSON events.

You can use the built-in Cribl Schema or create custom ones and use them to validate any fields by using the `C.Schema` expression.

JSON schemas are based on the popular JSON Schema standard, and Cribl Stream supports schemas matching that standard's Drafts 0 through 7.

You can refer to a single JSON schema in multiple expressions. When editing them, be aware that any change will affect all places it is used in.

Cribl Stream supports two kinds of schemas:

- JSON schemas for validating JSON events, described on this page.
- Parquet schemas for writing data from a Cribl Stream Destination to Parquet files.

These schemas serve different purposes. Operations that work for one kind of schema can't be used with the other. For example, the validation method for JSON schemas can't be used to validate Parquet schemas.

Validate an Object Against the Schema

To validate an object against a schema, use the `C.Schema` expression: `C.Schema('<schema_name>').validate(<object_field>)`.

You can call this method anywhere in Cribl Stream that supports JavaScript expressions. Typical use cases for schema validation include:

- Making a decision before sending an event down to a Destination.
- Making a decision before accepting an event (for example, dropping an event if invalid).
- Making a decision to route an event based on the result of validation.

Add New Schema

To add a new JSON schema:

1. In the sidebar, select **Worker Groups** and choose a Worker Group.
2. In the **Worker Groups** submenu, select **Processing**.

3. If you are adding the schema to a Pack, select **Packs** and choose your Pack.
4. Select **Knowledge**, then **Schemas**.
5. Select **Add Schema**.
6. Enter an **ID** for the regex and an optional **Description**.
7. In **Schema**, paste the JSON schema content.
8. Select **Save**.

JSON Schema Example

To add this example JSON Schema, enter the following into **New Schema** fields:

- ID: `schema1`.
- Description: (Enter your own description here.)
- Schema: Paste the following schema.

JSON Schema - Sample

```
{
  "$id": "https://example.com/person.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Person",
  "type": "object",
  "required": ["firstName", "lastName", "age"],
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The person's first name."
    },
    "lastName": {
      "type": "string",
      "description": "The person's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or greater than zero.",
      "type": "integer",
      "minimum": 0,
      "maximum": 42
    }
  }
}
```

Copy

Assume that events look like this:

Events

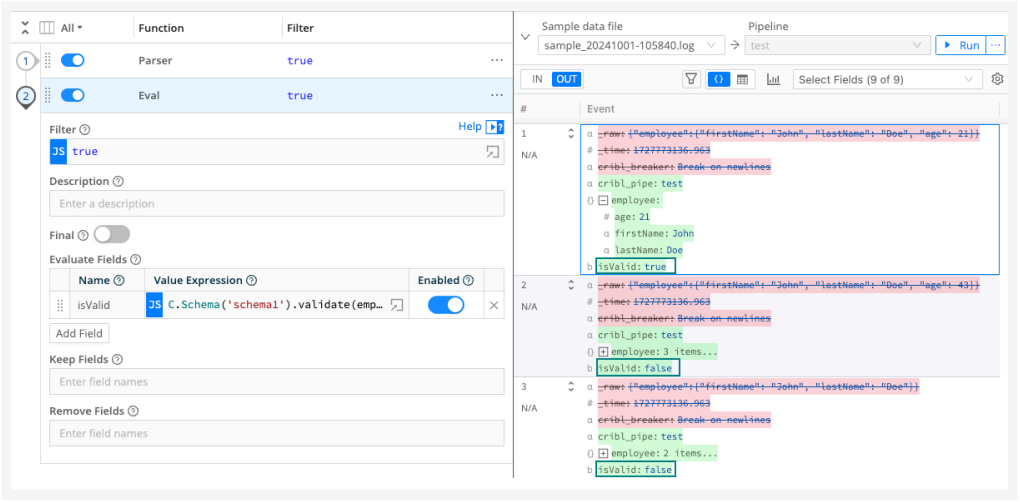
```
{"employee":{"firstName": "John", "lastName": "Doe", "age": 21}}
{"employee":{"firstName": "John", "lastName": "Doe", "age": 43}}
{"employee":{"firstName": "John", "lastName": "Doe"}}
```

Copy

To validate whether the `employee` field is valid per `schema1`, we can use the following:
`C.Schema('schema1').validate(employee)`

Results:

- First event is **valid**.
- Second event is **not valid** because `age` is greater than the maximum of `42`.
- Third event is **not valid** because `age` is missing.



Schema validation results for the above events

10.1.4. Parquet Schemas

Selected Cribl Stream Destinations support writing out data as files in the Apache Parquet columnar storage format.

The following Destinations support the Parquet format:

- Azure Blob Storage
- Azure Data Explorer
- Cloudflare R2
- Databricks
- Filesystem/NFS
- Google Cloud Storage
- MinIO
- Amazon S3 Compatible Stores
- Amazon Security Lake

You can select the Parquet format from the Destination's **General** > **Data format** drop-down. (An exception is the Amazon Security Lake Destination, which does not have the drop-down, because it uses the Parquet format exclusively). When the `Parquet` format is selected, you can configure the Parquet settings for the Destination, including selecting the Parquet schema to use.

You can automatically generate a Parquet schema based on the events of each Parquet file the Destination writes, by toggling on **Parquet Settings** > **Automatic schema** for the Destination. The automatically-generated schema preserves the exact structure of the ingested events, with no data lost. Every field in the generated schema will have a data type of `JSON`, `STRING`, `INT_64`, `UNSIGNED_INT_64`, or `DOUBLE`.

You can refer to a single Parquet schema in multiple Destinations. When editing them, be aware that any change will affect all places it is used in.

When logging is set to `debug`, you will see what schema is associated with the Parquet files that Cribl Stream writes out. On Linux, you can use the Cribl Stream CLI's `parquet` command to view a Parquet file, its metadata, or its schema.

Cribl Stream supports two kinds of schemas:

- Parquet schemas for writing data from a Cribl Stream Destination to Parquet files, described on this page.
- JSON schemas for validating JSON events.

> These schemas serve different purposes. Operations that work for one kind of schema can't be used with the other. For example, the validation method for JSON schemas can't be used to validate Parquet schemas.

Automatic Parquet Schema Generation

When you decide whether or not to generate Parquet schemas automatically, consider these pros and cons.

Advantages:

- Configuration is minimal.
- There's no need to know how events you want to ingest are structured.
- There's no need to change Parquet schema when event structure changes.

Disadvantages:

- Writing Parquet files takes longer than with predefined schemas. Note that schema complexity increases linearly with the number of events used to generate the schema.
- Automatically-generated Parquet schemas cannot be as fine-tuned as predefined ones, to capture certain nuances. For example, encoding will always be `PLAIN`, compression will always be `SNAPPY`.
- Some aspects of event structure (such as maps and sets) cannot be captured in the schema.
- Some aspects of time-related fields cannot be captured in the schema.
- Because the system produces a new Parquet schema for each file written, there may be differences between successive schemas. This can limit your ability to concatenate a collection of Parquet files.

Add Parquet Schema

If you are not using automatic schema generation, or if there's no pre-existing schema that suits your data, before configuring a Destination for Parquet output, you should add an existing Parquet schema, or create a new Parquet schema that suits the data you're working with.

You do not need to start from scratch: Cribl provides sample Parquet schemas for you to clone and then customize as needed.

To add a new Parquet schema:

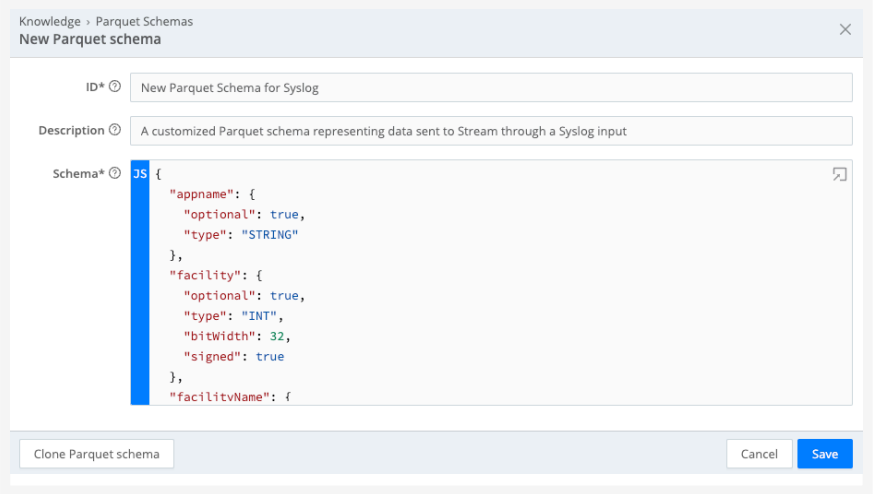
1. In the sidebar, select **Worker Groups** and choose a Worker Group.
2. In the **Worker Groups** submenu, select **Processing**.
3. If you are adding the schema to a Pack, select **Packs** and choose your Pack.
4. Select **Knowledge**, then **Parquet Schemas**.

5. Select **Add Parquet schema**.
6. Enter an **ID** for the regex and an optional **Description**.
7. In **Schema**, paste the Parquet schema content.
8. Select **Save**.

Modify Parquet Schema

You can modify existing Parquet schemas, but Cribl strongly recommends that you first clone the schema, and give the clone its own distinct name.

1. Select the schema you want to start with, to open it in a modal.
2. Select **Clone Parquet Schema** to open the **New Parquet schema** modal.
3. Give the new schema a name and description.



Creating a Parquet schema

Cloning vs Modifying Schemas

Modifying an existing schema does **not** propagate your modifications to its clones. Cloning and renaming schemas is the safest approach, because it ensures that your Destination will use the newly modified version of the schema.

If you do not clone and rename the schema (that is, you leave the schema name unchanged), you still must re-select the schema in the Destination's **Parquet schema** drop-down to bring the modified version into the Destination.

Parquet Format Used by Cribl Stream

Different Parquet readers and writers behave differently. Keep the following guidelines in mind when working with Parquet in Cribl Stream.

In the Parquet Schema editor, you express your Parquet schema in JSON. This does not look like the examples in the Parquet spec, which have their own syntax (like the repetition/name/type triple), but they are functionally equivalent. See the examples below.

The editor provides autocompletion and validation to guide you. (However, Cribl Stream currently does not fully support autocompletion on deeply nested schemas.)

File Extensions

Cribl Stream can read Parquet files only if they have the extension `.parquet`, `.parq`, or `.pqt`.

Field Content

When Cribl Stream writes to a Parquet file:

- If the data contains a field that **is not** present in the schema - i.e, an extra field - Cribl Stream writes out the parent rows, but omits the extra field.
- If the data contains a field that **is** present in the Parquet schema, but whose properties violate the schema, Cribl Stream treats this as a data mismatch. Cribl Stream drops the rows containing that field - it does not write those rows to the output Parquet file at all.
- If the data contains JSON, the JSON must be stringified. Otherwise, Cribl Stream treats this as a data mismatch, and does not write out the row. For example, this valid (but not stringified) JSON will trigger a data mismatch: `{ "name": "test"}`.
  The same JSON in stringified form will work fine: `"{\"name\": \"test\"}"`.

Data Types

Cribl Stream supports:

- All primitive types.
- All logical types.
- All converted types.

Converted types have been superseded by logical types, as described in the Apache Parquet docs. Cribl Stream can read Parquet files that use converted types, but will write out the same data using corresponding logical types.

Repetition Type

You have three alternatives when defining a field's Repetition type:

- Set `optional` to `true`.
- Set `repeated` to `true`.

- Set neither `optional` nor `repeated`. This **implicitly** sets the Repetition type to `required`, and it is the default.

Usage guidelines:

- Do not set both `optional` **and** `repeated` to `true`.
- Do not use the `required` key at all.
- Instead of omitting `optional`, you have the option to include it, but set it to `false`.
- Instead of omitting `repeated`, you have the option to include it, but set it to `false`.
- If any field's Repetition type is `repeated`, Cribl Stream represents this field as a single key whose value is an array - not as separate key-value pairs with identical keys.

Encodings

- Among the `*DICTIONARY` encodings, Cribl Stream supports only `DICTIONARY`. Trying to assign the unsupported encodings `PLAIN_DICTIONARY` or `RLE_DICTIONARY` will produce an error.
- `BYTE_STREAM_SPLIT` encoding can be used only with `DOUBLE` or `FLOAT` types, and otherwise produces errors.
- The `RLE` and all `DELTA*` encodings also produce errors.

Parquet Limitations

Cribl Stream does not support writing Parquet files which employ any of the following:

- Parquet Modular Encryption.
- The Parquet Bloom Filter.
- Separation of metadata and/or column data into multiple files.

Cribl Stream does not support reading or writing Parquet files which employ the deprecated `INT96` data type.

Examples of Parquet Schema as JSON

These examples should give you a hint of how to express Parquet schema in JSON. For a full explanation of the Parquet syntax, see the Parquet spec.

List Example

In this example, the whole schema is named `the_list`, and its structure consists of a container named `list` whose zero or more elements are each named `element`. Note that in Parquet schema, to specify that a field is nested (that is, contains other fields), you just give it the type `group`.

Parquet Schema Example 1

```
message schema {
  REQUIRED group the_list (LIST) {
    REPEATED group list {
      REQUIRED BYTE_ARRAY element (STRING);
    }
  }
}
```

Copy

Expressed as JSON, the same schema is more spread-out, because to specify that a field is nested, you give it a sub-field named `fields`. (This is equivalent to declaring the nested field's type as `group`.)

Parquet Schema Example 1 Expressed as JSON

```
{
  "the_list": {
    "type": "LIST",
    "fields": {
      "list": {
        "repeated": true,
        "fields": {
          "element": {
            "type": "STRING"
          }
        }
      }
    }
  }
}
```

Copy

Map Example

In this example, the whole schema is named `the_map`, and its structure consists of a container named `key_value` whose zero or more pairs of elements each contain a `key` and a `value` element. Both the `key` and the `value` are strings in this case, but they could be other data types.

Parquet Schema Example 2

```
message schema {
  REQUIRED group the_map (MAP) {
    REPEATED group key_value {
      REQUIRED BYTE_ARRAY key (STRING);
      REQUIRED BYTE_ARRAY value (STRING);
    }
  }
}
```

Copy

Parquet Schema Example 2 Expressed as JSON

```
{
  "the_map": {
    "type": "MAP",
    "fields": {
      "key_value": {
        "repeated": true,
        "fields": {
          "key": {
            "type": "STRING"
          },
          "value": {
            "type": "STRING"
          }
        }
      }
    }
  }
}
```

Copy

Array of Arrays Example

In this example, the whole schema is named `array_of_arrays`, and its structure consists of a container named `list` whose zero or more elements - which are themselves lists - are each named `element`. Within each of those `element` lists are one or more additional `list/element` structures.

Parquet Schema Example 3

```
message schema {
  REQUIRED group array_of_arrays (LIST) {
    REPEATED group list {
      REQUIRED group element (LIST) {
        REPEATED group list {
          REQUIRED INT64 element;
        }
      }
    }
  }
}
```

Copy

Expressed as JSON, the same schema is cumbersome to read compared to Parquet, which was designed to express nested structures concisely.

Parquet Schema Example 3 Expressed as JSON

```
{
  "array_of_arrays": {
    "type": "LIST",
    "fields": {
```

10.1.5. AppScope Configs

AppScope is deprecated and will be removed in a future release. Don't use AppScope in production and ensure it is not part of your data flow configuration. For details, please read the announcement in Cribl Community Slack.

You can obtain high-fidelity application data from any Linux command or application using AppScope, an open-source tool from Cribl. This includes data that's difficult or impossible to obtain any other way.

AppScope ships with a set of config files that offer a variety of options for what events and metrics you want AppScope to capture, and whether to enable payload capture. These are: `sample_config`, `default_metrics_events`, `verbose_metrics_events`, and `verbose_metrics_events_payloads`.

To understand precisely what one of these files does, select it to open its **AppScope Configs** modal and select **Manage as JSON**. For explanations and definitions, see the Config File and Schema Reference sections in the AppScope docs.

You can customize the config files, or edit them via **Manage as JSON**. When editing, you can revert to the original state by selecting **Restore Default**. The safer option is to select **Clone AppScope config**, then edit and save your new version with a new file name. The sample configs, and any variants you have created and saved, constitute a reusable set of AppScope configurations in the AppScope Config Library.

When you have assigned an AppScope config to a Rule in an AppScope Source, and you change it in the Knowledge UI, the changes will not take effect until you do the following:

- In the AppScope Source **AppScope Rules** settings, re-assign the config to the Rule. (That is, unselect the config file, then select it again.)
- Save and commit.

Creating AppScope Configs

To add a new Appscope config:

1. In the sidebar, select **Worker Groups** and choose a Worker Group.
2. In the **Worker Groups** submenu, select **Processing**.
3. If you are adding the config to a Pack, select **Packs** and choose your Pack.
4. Select **Knowledge**, then **AppScope Configs**.
5. Select **Add AppScope config** and fill in the options below.

General Settings

ID: Enter a unique name to identify this AppScope Config.

Description: Optionally, enter a description that will remind you what the config is for when you see it listed in the library.

Tags: Optionally, enter any desired tags.

Transports

AppScope offers several options for routing data. See Data Routing in the AppScope docs.

Use the AppScope Source IP/port or UNIX domain socket: Toggle on to apply the transport options defined in the associated AppScope Source. Toggle off to expose the following fields in this section.

Cribl authentication token: Optionally, set a value to add to the `config-event` header (as a top-level `authToken` property) when AppScope establishes a connection.

Send metrics and events over the same connection: Defaults to toggled on, which locks some settings into preconfigured values. Toggle off if you prefer to route metrics and events independently.

Connection type: From the drop-down, choose one of the connection types described below, then enter values in whatever fields the UI exposes.

- Toggle **Send metrics and events over the same connection** on to configure a single connection by choosing `TCP, Edge` (the default), or `Unix`.
- When routing metrics and events independently, you configure one connection for metrics and another for events; the drop-down offers two additional connection types: `UDP` and `File`.

Transports and Connection Types

At the heart of any AppScope config are the transports you define. In AppScope, a transport specifies something to send, how to send it, and where to send it. The things to send can be metrics, events, or logs.

Depending on the choices you make in the **Transport** settings, the UI will make some or all of the following types of connections available.

UDP: The destination is the network server whose hostname or IP address you specify in the **Host** field, along with a UDP **Port** to listen on.

TCP: The destination is the network server whose hostname or IP address you specify in the **Host** field, along with a TCP **Port** to listen on. The **Enable TLS** toggle defaults to off.

Unix: The destination is a Unix domain server listening on the Unix domain **Socket path** you specify.

File: The destination is the file whose directory location you specify in the **File path** field, along with a **Buffering** method.

Edge: The destination is Cribl Edge, listening on a preconfigured Unix domain socket.

TLS Settings (TCP Only)

When the **Connection type** is `TCP`, the UI exposes the **Enable TLS** toggle, which defaults to off.

If you toggle **Enable TLS** on:

- Toggle **Validate server certs** on if you want the connection to fail when the TLS server certificate cannot be validated. Defaults to off.
- Leave **CA certificate path** blank if you toggle **Validate server certs** on and are using the local OS-provided trusted CA certificates to validate the server's certificate. Otherwise, specify the server path containing CA certificates, which must be in PEM format.

Logs Settings

Configure where and how AppScope should send its own logs.

Connection type: From the drop-down, choose one of the connection types described above, then enter values in whatever fields the UI exposes.

Log level: Set logging verbosity. When you toggle **Send metrics and events over the same connection** on, **Log level** is locked at `warning`.

File path: Specify a directory in which to store logs. If not specified, defaults to `/tmp/scope.log`.

Buffering: Set this method to `Line` if there's a chance that multiple scoped processes will be writing to the same file. This prevents the log file from getting scrambled, with interleaved lines. However, in single-writer scenarios, selecting `Full` often improves performance.

Metrics Settings

The Schema Reference in the AppScope docs lists and fully describes every metric that AppScope can emit.

Process (`proc.*`) metrics periodically report information about resource usage at a point in time. All other metrics report on actions taken by a scoped process. For examples, see Metrics and Events in AppScope in the AppScope documentation.

Emit metrics: Defaults to toggled on, which exposes the following options.

Format: AppScope can emit metrics in either of two formats: `NDJSON` (the default), or `StatsD`. If you choose `StatsD`, the UI exposes the following fields:

- **StatsD prefix**: Optionally, enter a prefix for AppScope to add to metrics.
- **StatsD max length**: Optionally, enter a maximum length, in bytes, for StatsD-formatted metrics. AppScope will truncate any metric which exceeds the maximum length. (Defaults to `512`.)

Verbosity: Set a value between `0` and `9`. (Defaults to `4`.) Verbosity controls both the cardinality of metrics that AppScope emits, and which kinds of metrics AppScope aggregates. For details, see Metrics and Events in AppScope in the AppScope documentation. But the general principle is:

- At low verbosity, metrics summarize multiple actions over a reporting period.
- At higher verbosity, metrics provide details about single actions in real time.

Reporting period: Set the metric summary interval in seconds. Defaults to `10`.

Watch list: Select categories of metrics to emit. Defaults to the following categories: `StatsD`, `File System`, `Network`, `HTTP`, `DNS`, `Process`.

Events Settings

The Schema Reference in the AppScope docs lists and fully describes every event that AppScope can emit.

Emit events: Defaults to toggled on, which exposes the following controls in this section.

Event-rate limiter (events per second): AppScope discards events generated in excess of the limiter setting, which defaults to `10000`. To disable the limiter, set it to `0`.

Enhance filesystem event data: If toggled on (default), AppScope enhances `fs.open` events by adding the `uid`, `gid`, and `mode` fields.

Watch list: Specify categories of events to emit, and configure their content and structure. Defaults to the following categories: `Console`, `File`, `Filesystem`, `Network`, `DNS`, and `HTTP`. For details, see Categories.

Events Categories

The **Watch list** is a table providing a row for each enabled category of events. The **Type** column contains the category names. The other columns (**Name**, **Field**, and **Value**) contain regular expressions.

At the right edge of the **Name**, **Field**, and **Value** fields is a Copy button, a Flag button to append Regex flags, and an Expand button to open a validation modal.

AppScope emits an event when its values match all the regexes in the corresponding category's row. For **HTTP**, request and response events must also match the headers listed in the **Header** column.

For **Console** events, toggle **Allow binary** on (default) if you want **Console** events to include binary data along with text data. If you prefer to redact binary data, toggle **Allow binary** off.

The **Type** column enumerates any or all of the following event category names. Switch the toggle in the **Enabled** column to include a category.

- **Console**: Includes writes to standard out and error. Use this to monitor console output, especially in containerized environments where logging to files isn't commonly done.
- **File**: Includes writes to files. Use this to monitor log files and/or any other files of interest.
- **Filesystem**: Includes filesystem operations like `open`, `close`, and `delete`.
- **Network**: Includes `open` and `close` events on network connections.
- **DNS**: Includes DNS request and response events.
- **HTTP**: Includes HTTP request and response events.
- **Granular**: Seldom used, and not emitted by default. Can be useful when tracking down a problem. When this type is toggled on, AppScope sends more events than it otherwise would, giving you a more granular view of the activity of the scoped process.

Some usage examples:

- For `fs.*` events, AppScope normally emits `fs.open` and `fs.close` events. With **Granular** enabled, every time a read or write occurs, AppScope emits `fs.read` or `fs.write`, respectively.
- For `net.*` events, AppScope normally emits `net.open`, `net.close`, and `net.app`.With **Granular** enabled, every time a read or write is done on the socket, AppScope emits `net.rx` or `net.tx`, respectively.

Payloads

AppScope never sends encrypted payloads to disk, to Cribl Stream, or to Cribl Edge.

Capture all payloads: Toggle on to enable capturing all payloads. Defaults is off. Enable this setting with caution: when scoping I/O-intensive programs, it can produce large amounts of data.

Protocol detection: Select **Add Protocol** to specify a protocol (for example, StatsD, HTTP, Redis) for AppScope to detect in network payloads, as well as what AppScope should do upon detecting that protocol. AppScope checks the first packet on a channel against the regular expression for each protocol until it finds a match. Then AppScope treats the matching protocol as the detected protocol for its channel, and ignores any others listed. See Sample Protocols for regexes that detect common protocols.

Name: Enter a short, descriptive name to define the protocol-detect event. AppScope inserts this value into the header for payloads sent to Cribl. Since AppScope sends this as JSON, you cannot include double quotes or backslashes.

Regex: Enter a regular expression to detect the desired protocol.

Generate protocol-detect-events: Toggle on to generate protocol-detect events. Default is toggled off.

Capture payload: Toggle on to send payloads from matching streams to the configured destinations. Default is toggled off.

Convert to hex: Toggle on (default) if you want AppScope to convert part of the payload to a hex-string before applying the regex. Toggle off if you want AppScope to apply the regex to the binary payload.

Bytes to convert: The number of bytes to convert to hex before applying the regex. Defaults to `256`.

Payload directory: Specify the directory where AppScope should store payload files. Consider using a performant filesystem to reduce I/O performance impacts. Defaults to `tmp`. When **General Settings** > **Send metrics and events over the same connection** is toggled off, AppScope ignores this field and sends payloads to the default destination.

Advanced Settings

Send process-start message: Toggle on (default) to enable the process-start message. This message is the first one AppScope sends upon establishing a connection. It describes AppScope's configuration for that session, as well as the application being scoped.

Command directory: If you want to dynamically change AppScope configuration while scoping a running process, save a `scope.<pid>` file to a **command directory** of your choice, where `<pid>` is the PID of the scoped process, and the file contains lines of the form `<ENV_VAR>=<value>`. Once per reporting period, AppScope looks for a `scope.<pid>` file in the **command directory**, which defaults to `/tmp`. You can configure reporting period in the **Metrics** tab.

Metadata: If you want to add a field to events and metrics that AppScope emits, select **Add Metadata** and define it as a key-value pair. Key/value definitions cannot include environment variables.

Per-process configs: Optionally, define a config that, given a process that fits a particular pattern, will partly or completely override the config you have defined already.

Select **Add config** and configure the **Filters** relevant to the process(es) you want to match:

- **Process name**: Enter a string to match the basename of the scoped process.
- **Process argument**: Enter an expression to match the scoped process's full command line including options and arguments.
- **Hostname**: Enter a string to match the hostname of the machine where the scoped process is running.
- **Username**: Enter a string to match the username for the scoped process's UID.
- **Environment**: To match when an environment variable is set, enter its name alone (for example, `FOO`). To match when the variable is set with a particular value, enter both (for example, `FOO=bar`).

- **Ancestor**: Enter a string to match the basename of the scoped process's parent, parent's parent, and so on.

Then, in the **Config** column, select **Edit**. This takes you through the settings, beginning with **General Settings**, but this time you'll be specifying what in the previously-defined settings to override when a process matches your **Filters**.

Protocol Detection Examples for Payloads

Here are some example definitions for detecting protocols in payloads.

Redis

This regex detects the plain-text Redis serialization protocol (RESP).

Name: `Redis`

Regex: `"^[*]\\d+|^[+]\\w+|^[$]\\d+"`

Mongo

This regex detects the MongoDB wire protocol. Since it's binary, we convert to hex.

Name: `Mongo`

Regex: `^2401000000000000000000000d407`

Convert to hex: Toggle on.

Convert length: `14`

HTTP

By default, AppScope uses an internally-defined protocol detector for HTTP similar to this example.

Name: `HTTP`

Regex: `"HTTP\\/1\\.[0-2]|PRI \\* HTTP\\/2\\.0\r\n\r\nSM\r\n\r\n"`

StatsD

By default, AppScope uses an internally-defined protocol detector for StatsD similar to this example.

Name: `STATSD`

Regex: `"^([^:]+):([\\d.]+)\\|(c|g|ms|s|h)"`

TLS

By default, AppScope uses an internally-defined protocol detector for TLS/SSL similar to this example.

Name: TLS

Regex: `"^(?:(?:16030[0-3].{4})|(?:8[0-9a-fA-F]{3}01))"`

Convert to hex: Toggle on.

Convert length: 5

10.2. Cribl Expressions

Native Cribl Stream methods can be found under `C.*`, and can be invoked from any Function that allows for expression evaluations. For example, to create a field that is the SHA1 of a another field's value, you can use the Eval Function with this **Evaluate Fields** pair:

Name	Value Expression
myNewField	`C.Mask.sha1(myOtherField)`

> Where fields' names contain special characters, you can reference them using the `__e['<field-name-here>']` convention. For details, see Build Custom Logic to Route and Process Your Data: Fields with Non-Alphanumeric Characters.

Cribl expressions offer methods and properties in the following classes:

- C.Crypto - data encryption and decryption
- C.Encode and C.Decode - encoding and decoding data
- C.Lookup - running lookups
- C.Mask - masking sensitive data
- C.Net - network methods
- C.Text - text manipulation
- C.Time - time and date manipulation
- C.Schema - schema validation
- C.Secret - secret management
- C.env - environment properties
- C.os - system methods
- C.vars - global variables
- C.version - Cribl Stream versions
- C.Misc - various other methods

10.2.1. C.Crypto - Data Encryption And Decryption

C.Crypto.decrypt()

```
C.Crypto.decrypt(value: string, escape: boolean = false, escapeSeq = '""'): string
```

Copy

Decrypts all occurrences of ciphers in the given value. Instances that cannot be decrypted (for any reason) are left intact.

Returns `value` with ciphers decrypted.

Parameter	Type	Description
`value`	string	String in which to look for ciphers.
`escape`	boolean	Set to `true` to escape double quotes in output after decryption (for example, for data encrypted in Splunk.) Defaults to `false`.
`escapeSeq`	string	String used to escape double quotes. The default `'""'` escapes CSV output.

Examples

To decrypt the contents of the `encrypted` field, use:

```
C.Crypto.decrypt(encrypted)
```

Copy

C.Crypto.encrypt()

```
C.Crypto.encrypt(value: Buffer | string, keyclass: number, keyId?: string, defaultVal
```

Copy

Encrypts the given value with the `keyId`, or with a `keyId` picked up automatically based on `keyclass`.

Returns the encrypted value. If encryption does not succeed, returns `defaultVal` if specified; otherwise, `value`.

Parameter	Type	Description
`value`	string | Buffer	The value to encrypt.
`keyclass`	number	Key Class to pick a key from, if `keyId` isn't specified.
`keyId`	string	Encryption `keyId`, takes precedence over `keyclass`.
`defaultVal`	string	Value to return if encryption fails; if unspecified, the original value is returned.

You can decrypt the encrypted data in Cribl Search, using the `encrypt` and `decrypt` functions in your queries.

Examples

To encrypt the `customer` field using key with `keyId` 2, use:

```
C.Crypto.encrypt(customer, -1, 2)
```

Copy

To encrypt the same field with a key selected from the Key Class 3, without specifying a `keyId`:

```
C.Crypto.encrypt(customer, 3)
```

Copy

C.Crypto.createHmac()

```
C.Crypto.createHmac(value: string | Buffer, secret: string, algorithm: string = 'sha2
```

Copy

Generates an HMAC that can be added to events, or can be used to validate events that contain an HMAC.

Returns the calculated HMAC digest on success; otherwise, `value`.

Parameter	Type	Description
`value`	string | Buffer	The data to encrypt. When the `outputFormat` is invalid or undefined, this parameter is returned as the digest, via a Buffer.
`secret`	string	The secret key used to generate the MAC.
`algorithm`	string	The hash algorithm used to generate the MAC. Defaults to `'sha256'`. Run `openssl list -digest-algorithms` to see the list of available algorithms.
`outputFormat`	`'base64' | 'hex' | 'latin1'`	Output format for the MAC. Defaults to `'hex'`.

Examples

To generate HMAC from the `customer` field, provide the secret as the second argument:

```
C.Crypto.createHmac(customer, 'kETvYtLKZkgIzBvfRdVcZULBITANjOMd')
```

Copy

To additionally specify that you want to use the SHA512 hash algorithm and to output the HMAC in `latin1` format, run:

```
C.Crypto.createHmac(customer, 'kETvYtLKZkgIzBvfRdVcZULBITANjOMd', 'sha512', 'latin1')
```

Copy

10.2.2. C.Decode And C.Encode - Encoding And Decoding

C.Decode - Data Decoding Methods

C.Decode.base64()

```
C.Decode.base64(val: string, resultEnc: string = 'utf8'): string | Buffer | undefinec
```

Copy

Performs Base64 decoding of the given string.

Returns a string or Buffer, depending on the `resultEnc` value.

Parameter	Type	Description
`val`	string	Value to decode.
`resultEnc`	`'utf8' | 'utf8-valid' | 'buffer'`	Encoding to use to convert the binary data to a string. Use `'utf8-valid'` to validate that result is valid UTF8; use `'buffer'` if you need the binary data in a Buffer. Defaults to `'utf8'`.

Examples

To decode an `encoded` field and return the decoded value as a Buffer, use:

```
C.Decode.base64(encoded, 'buffer')
```

Copy

C.Decode.gzip()

```
C.Decode.gzip(value: Buffer | string, encoding?: BufferEncoding) : string
```

Copy

Gunzips the supplied value.

Parameter	Type	Description
`value`	any	Value to gunzip.
`encoding`	string	Encoding of `value`, for example: `'base64'`, `'hex'`, `'utf-8'`, `'binary'`. Default is `'base64'`. If data is received as Buffer (from gzip with encoding: `'none'`), decoding is skipped.

Example

To decode a `hostname` field which had been encoded using `hex`, run:

```
C.Decode.gzip(hostname, 'hex')
```

Copy

C.Decode.hex()

```
C.Decode.hex(val: string): number
```

Copy

Performs hex to number conversion. Returns `NaN` if value cannot be converted to a number.

Parameter	Type	Description
`val`	string	Hex string to parse to a number (for example, `0xcafe`).

C.Decode.inflate()

```
C.Decode.inflate(value: Buffer | string, encoding: BufferEncoding = 'base64', isRaw =
```

Copy

Inflates the supplied value.

Parameter	Type	Description
`value`	string | Buffer	The value to inflate. May be a Buffer or a string.

Parameter	Type	Description
`encoding`	string	If a string `value` is provided, this specifies how it has been encoded. If no `encoding` is provided, `base64` is used. This parameter is ignored if `value` is already a Buffer.
`isRaw`	boolean	Indicates whether the data is in raw deflate format (without zlib headers). Default value is `false`.

Examples

To Base64-decode and inflate the value contained in a `_raw` field, you can use:

```
C.Decode.inflate(_raw)
```

Copy

To additionally specify a different encoding to use, run:

```
C.Decode.inflate(_raw, 'hex')
```

Copy

`C.Decode.mime()`

```
C.Decode.mime(val: string): string
```

Copy

Decode string that has been encoded with MIME RFC 2047 standard. This standard is frequently used in email subject lines within logs to handle non-ASCII characters.

Parameter	Type	Description
`val`	string	The input string to decode. This string must be formatted according to RFC 2047, typically starting and ending with `=?<charset>?<encodingType>?...?=`.

Examples

To decode a MIME-encoded string using the Quoted-Printable format, typically seen when the non-ASCII content is mostly readable text, use:

```
C.Decode.mime('=?UTF-8?Q?R=C3=A9sum=C3=A9?=') // Returns: 'Résumé'
```

Copy

To decode a MIME-encoded string using the Base64 format, often used when the non-ASCII content is complex or binary data, use:

```
C.Decode.mime('=?UTF-8?B?Q2Fmw6k=?=') // Returns: 'Café'.

### `C.Decode.uri()`

```js
C.Decode.uri(val: string): string
```

Copy

Performs URI-decoding of the given string.

Parameter	Type	Description
`val`	string	Value to decode.

C.Encode - Data Encoding Methods

`C.Encode.base64()`

```
C.Encode.base64(val: string | Buffer, trimTrailEq: boolean = false, encoding?: Buffer
```

Copy

Returns a Base64 representation of the given string or Buffer.

Parameter	Type	Description
`val`	any	Value to encode.
`trimTrailEq`	boolean	Whether to trim any trailing `=`.
`encoding`	string	Optional parameter that specifies the encoding of the value. Options include `base64`, `hex`, `utf-8`, and `binary`. The default is `utf-8`.
```

Examples

To Base64-encode the `username` field, and remove the trailing `=`, use:

```
C.Encode.base64(username, true)
```

Copy

To Base64-encode the `hexValue` field, where the value of the field is a hex string:

```
C.Encode.base64('4672616e6b7575757575', undefined, 'hex')
```

Copy

`C.Encode.deflate()`

```
C.Encode.deflate(value: Buffer | string, encoding: BufferEncoding | 'none' = 'base64'
```

Copy

Deflates and optionally Base64-encodes the supplied value.

Parameter	Type	Description
`value`	string or Buffer	Value to deflate.
`encoding`	string	Encoding to apply to the deflated result, for example: `base64`, `hex`, `utf-8`, `binary`, `none`. Default is `base64`. If `none` is specified, data will be returned as a Buffer.
`toRaw`	boolean	Indicates whether to return the raw deflated data, without zlib headers. Default is `false`, which will prepend normal zlib header bytes to the returned value.

Examples

To deflate and hex-decode the value provided in an `inflated` field, run:

```
C.Encode.deflate(inflated, 'hex')
```

Copy

C.Encode.gzip()

```
C.Encode.gzip(value: Buffer | string, encoding?: string) : string | Buffer
```

Copy

Gzips, and optionally Base64-encodes, the supplied value.

Parameter	Type	Description
`value`	string	Value to gzip.
`encoding`	string	Encoding of `value`, for example: `'base64'`, `'hex'`, `'utf-8'`, `'binary'`, `'none'`. Default is `'base64'`. If `'none'` is specified, data will be returned as a Buffer.

Examples

To encode the `hostname` field, for example using `hex` encoding, run:

```
C.Encode.gzip(hostname, 'hex')
```

Copy

C.Encode.hex()

```
C.Encode.hex(val: string | number): string
```

Copy

Rounds the number to an integer and returns its hex representation (lowercase). If a string is provided, it will be parsed into a number or `NaN`.

Parameter	Type	Description
`val`	string | number	Value to convert to hex.

C.Encode.mime()

```
C.Encode.mime(val: string, encoding: string = 'Q', maxLineLength: number = 0, charset
```

Copy

Encode a string with MIME RFC 2047 standard. This method is used to prepare header values that contain non-ASCII characters for transport, such as email subject lines.

Parameter	Type	Description
`val`	string	The raw string to be encoded, including any non-ASCII characters.
`encoding`	MIME Encoding	Defines the encoding type. Use `Q` for Quoted-Printable, which is the default and can be omitted. Use `B` for Base64.
`maxLineLength`	number	Used for folding long lines, which is required by the RFC for headers. If greater than zero, inserts a newline delimiter after this defined number characters to ensure line wrapping. If `0`, no folding is performed.
`charset`	string	The character set of the input string, if it is not UTF-8.

Examples

To encode a MIME-encoded string using the Quoted-Printable format, use:

```
C.Encode.mime(_raw)
```

Copy

To encode a MIME-encoded string using the Base64 format, use:

```
C.Encode.mime(_raw, `B`)
```

Copy

To encode a string using Quoted-Printable and apply line folding to ensure no encoded line exceeds 20 characters:

```
C.Encode.mime('A much longer subject line with a non-ascii character: Español', 'Q',
// Returns `=?UTF-8?Q?A_much_longer_subject_line_with_a_non-ascii_characte?= \n =?UTF
```

Copy

C.Encode.uri()

```
C.Encode.uri(val: string): string
```

Copy

Returns the URI-encoded representation of the given string.

Parameter	Type	Description
`val`	string	Value to encode.

10.2.3. C.Lookup - Inline Lookup Methods

C.Lookup - Exact Lookup

```
C.Lookup: (file: string, primaryKey?: string, otherFields: string[]=[],ignoreCase: bo
```

Copy

Returns an instance of a lookup to use inline.

Examples

In this example, `host` is the name of the primary key field:

```
C.Lookup('lookup_name.csv', 'IP_field_name_in_lookup_file').match(host)
```

Copy

Here, the quoted `'event_field_or_string_to_match'` could be a string to match in the primary key field:

```
C.Lookup('name_of_lookup_file.csv', 'field_in_csv_to_match').match('event_field_or_st
```

Copy

This example checks whether `someValue` is present or not and returns the answer as a boolean.

```
C.Lookup('file', 'primaryKeyCol', additionalCols).match('someValue')
```

Copy

This expression returns a string - the `fieldToReturn` column from the matched lookup row:

```
C.Lookup('file', 'primaryKeyCol', additionalCols).match('someValue','fieldToReturn')
```

Copy

This expression returns an array - the specified `fieldToReturn1` and `fieldToReturn2` columns from the matched lookup row:

```
C.Lookup('file', 'primaryKeyCol').match('someValue', ['fieldToReturn1', 'fieldToRetu
```

Copy

To return an array with all columns from the matched lookup row, you can use:

```
C.Lookup('file', 'primaryKeyCol').match('someValue', [])
```

Copy

`C.Lookup` can load lookup files of up to 10 MB.

All inputs to Lookup methods' `match()` method must be strings. If your lookup file contains numeric fields, convert them to strings, e.g.: `.match(String(<fieldname>))`.

You can use the optional `otherFields[]` argument, shown in the above `C.Lookup()` signatures and examples, to limit which columns of the lookup table will be available in a subsequent `.match()` call. If omitted, or set to `undefined`, all columns will be available.

Cribl Stream 4.1 and newer support returning all columns from a matched row. Use the empty-array convention shown in the above examples: `.match('someValue',[])`.

`C.LookupCIDR` - CIDR Lookup

```
C.LookupCIDR: (file: string, primaryKey?: string, otherFields: string[]=[]) => Inline
```

Copy

Returns an instance of a CIDR lookup to use inline.

`C.LookupIgnoreCase` - Case-insensitive Lookup

```
C.LookupIgnoreCase: (file: string, primaryKey?: string, otherFields: string[]=[]) =>
```

Copy

Returns an instance of a lookup (ignoring case) to use inline. Works identically to `C.Lookup`, except ignores the case of lookup values. (Equivalent to calling `C.Lookup` with its fourth `ignoreCase?` parameter set to `true`).

C.LookupRegex - Regex Lookup

```
C.LookupRegex: (file: string, primaryKey?: string, otherFields: string[]=[]) => Inlir
```

Copy

Returns an instance of a Regex lookup to use inline.

C.Lookup.match()

```
InlineLookup.match(value: string): boolean

InlineLookup.match(value: string, fieldToReturn?: string): any
InlineLookup.match(value: string, fieldToReturn: string[]): {[key: string]: any}
```

Copy

Parameter	Type	Description
`value`	string	the value to look up.
`fieldToReturn`	string | string []	name of the lookup file > field to return.

Examples

To find the last row where the value from column `foo` matches the string `abc`, you can use the following expression. If a matching row is found, the expression returns the row's value for column `bar`.

```
C.Lookup('lookup-exact.csv', 'foo').match('abc', 'bar')
```

Copy

To find the last row where the value from column `transport` matches the string `tcp`, you can use the following expression. If a matching row is found, the expression returns the row's value for column `port_number`.

```
C.Lookup('service_names_port_numbers.csv', 'transport').match('tcp', 'port_number')
```

Copy

You can find the last row where the CIDR range in `cidr` includes `192.168.1.1` with the following expression. If a matching row is found, the expression returns the row's value for column `bar`.

```
C.LookupCIDR('lookup-cidr.csv', 'cidr').match('192.168.1.1', 'bar')
```

Copy

The following expression finds the last row where the value from column `cidr` matches `hostIP`. If a matching row is found, it returns the row's value for column `location`.

```
C.LookupCIDR('lookup-cidr.csv', 'cidr').match(hostIP, 'location')
```

Copy

To find the last row where the regex in column `foo` matches the string `manchester`, use the following expression. If a matching row is found, the expression returns the row's value for column `bar`.

```
C.LookupRegex('lookup-regex.csv', 'foo').match('manchester', 'bar')
```

Copy

With `C.LookupRegex`, ensure that your lookup file contains no empty lines - not even at the bottom. Any empty rows will cause `C.LookupRegex().match()` to always return `true`.

10.2.4. C.Mask - Data Masking Methods

The `C.Mask` expression provides a suite of methods for data masking, which is the process of obscuring sensitive information in your data. This ensures data privacy and maintains compliance with regulations like GDPR and HIPAA. It can protect personally identifiable information (PII) before sending it to a downstream Destination.

Use the methods in this expression to:

- **Redact sensitive information:** Replace data with a literal value like `REDACTED`. For example, replacing a credit card number like `1234-5678-9101-1121` to `REDACTED`.
- **Obfuscate data:** Replace values with a random or repeating character set. For example, obfuscating a username like `jdoe@example.com` by replacing it with a string of repeating characters, such as `XXXXXXXXXXXXXXX`.
- **Pseudonymize data:** Transform values into a non-identifiable, consistent value using a cryptographic hash. This technique allows you to perform analytics on the masked data without risking a privacy breach. For example, psudonymize a user ID like `user:1234567` into a consistent hash, such as `4b4f5d21c9a63c`. This makes it possible to analyze the data without revealing the original value.

By using `C.Mask` expressions, you can ensure that you handle sensitive data securely and that your Pipelines meet the necessary privacy and compliance standards.

For easier navigation, this page organizes the `C.Mask` methods by their purpose:

- Identity Validation and Masking: Methods for masking common identifiers like credit card and IMEI numbers, which also perform a basic validation check.
- Checksums and Validation: Methods that generate checksums and perform validation, which are the building blocks for more complex validation rules.
- Hashing: A collection of cryptographic hashing functions for pseudonymizing data while preserving its uniqueness for analytics.
- String Manipulation: General-purpose functions for creating random or repeating strings.

Identity Validation and Masking

`C.Mask.CC()`

Checks whether a value could be a valid credit card number. If the value matches the respective format, it masks a subset of the value. By default, all digits except the last 4 will be replaced with `X`.

Usage:

```
C.Mask.CC(value: string, unmasked: number = -4, maskChar: string='X'): string
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	The string to mask, **if and only if** it could be a valid credit card number.
`unmasked`	number	Number of digits to leave unmasked: positive for left, negative for right, `0` for none.
`maskChar`	string	String to replace each digit with.

Examples:

Assuming your parsed data contains a `creditCard` field, you can mask this number by using a Mask Function. Apply it to the `cardNumber` field and create the following masking rule:

Match Regex	Replace Expression	Example result
`(.*)` (catches the whole field contents)	`${C.Mask.CC(g1)}`	`cardNumber: XXXXXXXXXX4860`

To control how the masking is performed, you can specify how many digit should be unmasked and what masking character to use:

Match Regex	Replace Expression	Example result
`(.*)` (catches the whole field contents)	`${C.Mask.CC(g1, 4, "Z")}`	`cardNumber: 6011ZZZZZZZZZZZZ`

`C.Mask.IMEI()`

Checks whether a value could be a valid IMEI number. If the value matches the respective format, it masks a subset of the value. By default, all digits except the last 4 will be replaced with `X`.

Usage:

```
C.Mask.IMEI(value: string, unmasked: number = -4, maskChar: string='X'): string
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	The string to mask, **if and only if** it could be a valid IMEI number.
`unmasked`	number	Number of digits to leave unmasked: positive for left, negative for right, `0` for none.
`maskChar`	string	String to replace each digit with.

Examples:

Assuming your parsed data contains an `imei` field, you can mask this number by using a Mask Function. Apply it to the `imei` field and create the following masking rule:

Match Regex	Replace Expression	Example result
`(.*)` (catches the whole field contents)	`${C.Mask.IMEI(g1)}`	`imei: XXXXXXXXXXX7011`

To control how the masking is performed, you can specify how many digit should be unmasked and what masking character to use:

Match Regex	Replace Expression	Example result
`(.*)` (catches the whole field contents)	`${C.Mask.IMEI(g1, 4, "M")}`	`cardNumber: 5356MMMMMMMMMM`

`C.Mask.isCC()`

Checks whether the given value could be a valid credit card number, by computing the string's Luhn's checksum modulo 10 == `0`.

Usage:

```
C.Mask.isCC(value: string): boolean
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	String to check for being a valid credit card number.

Examples:

The following example expression uses `isCC()` to check whether the `cardNumber` field actually contains a credit card number. If it does, the card number is replaced with the `REDACTED` string, otherwise, en empty string is returned.

```
C.Mask.isCC(cardNumber) ? C.Mask.REDACTED : ""
```

Copy

`C.Mask.isIMEI()`

Checks whether the given value could be a valid IMEI number, by computing the string's Luhn's checksum modulo 10 == `0`.

Usage:

```
C.Mask.isIMEI(value: string): boolean
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	String to check for being a valid IMEI number.

Examples:

The following example expression uses `isIMEI()` to check whether the `imei` field actually contains an IMEI number number. If it does, the IMEI number is replaced with the `REDACTED` string, otherwise, en empty string is returned.

```
C.Mask.isIMEI(imei) ? C.Mask.REDACTED : ""
```

Copy

`C.Mask.REDACTED`

`C.Mask.REDACTED` is a literal string (not a function) that evaluates to `REDACTED`. It's used to completely remove sensitive data by replacing it with a clear, non-identifiable placeholder.

Usage:

```
C.Mask.REDACTED: string
```

Copy

Examples:

This example demonstrates using `C.Mask.REDACTED` within a conditional expression. It checks if the `cardNumber` field is a valid credit card number. If it is, the value is replaced with `REDACTED`. Otherwise, it keeps the original value.

```
C.Mask.isCC(cardNumber) ? C.Mask.REDACTED : cardNumber
```

Copy

Checksums and Validation

`C.Mask.luhn()`

Masks a value only if its checksum is valid. This is useful for sanitizing data and ensuring that only correctly formatted numbers are sent downstream.

It checks that value's Luhn checksum mod 10 is `0`, and masks a subset of the value. By default, all digits except the last 4 will be replaced with `X`. If the value's Luhn's checksum mod 10 is not `0`, then the value is returned unmodified.

Usage:

```
C.Mask.luhn(value: string, unmasked: number = -4, maskChar: string='X'): string
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	The string to mask, **if and only if** the value's Luhn checksum mod 10 is `0`.
`unmasked`	number	Number of digits to leave unmasked: positive for left, negative for right, `0` for none.

Parameter	Type	Description
`maskChar`	string	String to replace each digit with.

Examples:

Expression	Input Value	Output
`C.Mask.luhn(value)`	`49927398716` (a valid Luhn number)	`XXXXXXX8716`
`C.Mask.luhn(value)`	`49927398717` (an invalid Luhn number)	`49927398717`
`C.Mask.luhn(value, 4, "Y")`	`49927398716` (a valid Luhn number)	`4992YYYYYYY`

`C.Mask.luhnChecksum()`

Generates a checksum that can be used to validate a number. This is useful for building your own custom validation rules or for debugging.

Generates the Luhn checksum (used to validate certain credit card numbers, IMEIs, and more). By default, the mod 10 of the checksum is returned. Pass mod = `0` to get the actual checksum.

Usage:

```
C.Mask.luhnChecksum(value: string, mod: number=10): number
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	String whose digits you want to perform the Luhn checksum on.
`mod`	number	Return checksum modulo for this number. If `0`, skip modulo. Default is `10`.

Examples:

This example demonstrates how to get the default `mod 10` checksum. This expression returns the last digit of the checksum, which is commonly used to validate a number. A valid Luhn number will return `0`.

```
// This will return the number 0
C.Mask.luhnChecksum("49927398716")
```

Copy

This example demonstrates how to get the full checksum. You can get the full checksum by setting `mod` to `0`.

```
// This will return the number 70, which is the full checksum
C.Mask.luhnChecksum("49927398716", 0)
```

Copy

Hashing

C.Mask.crc32()

Generates a CRC32 hash of given value.

Usage:

```
C.Mask.crc32(value: string): string
```

Copy

Parameter:

Parameter	Type	Description
`value`	string	The string to hash.

Examples:

Expression	Input Value	Output
`C.Mask.crc32(value)`	`example_log_123`	`cf0e3d3a`
`C.Mask.crc32(host + file_path)`	`webserver01/var/log/app.log`	`b9b661b6`

C.Mask.md5()

Generates an MD5 hash of a given value.

`C.Mask.md5()` doesn't work when Cribl Stream is running in FIPS mode.

Usage:

```
C.Mask.md5(value: string, len?: string | number, encoding?: BufferEncoding): string
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	The string to hash.
`len`	string or number	Length of hash to return: `0` for full hash, a +number for left or a -number for right substring. If a string is passed, its length will be used.
`encoding`	string	Optional parameter that specifies the encoding of the value. Options include `base64`, `hex`, `utf-8`, and `binary`. The default is `utf-8`.

Examples:

To hash the contents of a `username` field with the MD5 algorithm, run:

```
C.Mask.md5(username)
```

Copy

To hash the contents of an `authToken` field, where the value of the field is a base64-encoded string, with the MD5 algorithm, run:

```
C.Mask.md5(authToken, undefined, 'base64')
```

Copy

C.Mask.sha1(), C.Mask.sha256(), C.Mask.sha512(), C.Mask.sha3_256(), C.Mask.sha3_512()

Generates a hash of a given value. Supported algorithms:

- SHA1
- SHA256
- SHA512

- SHA3-256
- SHA3-512

Usage:

```
C.Mask.sha1(value: string, len?: string | number, encoding?: BufferEncoding): string
C.Mask.sha256(value: string, len?: string | number, encoding?: BufferEncoding): strir
C.Mask.sha512(value: string, len?: string | number, encoding?: BufferEncoding): strir
C.Mask.sha3_256(value: string, len?: string | number, encoding?: BufferEncoding): str
C.Mask.sha3_512(value: string, len?: string | number, encoding?: BufferEncoding): str
```

Copy

Parameters:

Parameter	Type	Description
`value`	string	The string to hash.
`len`	string or number	Length of hash to return: `0` for full hash, a +number for left, or a -number for right substring. If a string is passed, its length will be used.
`encoding`	string	Optional parameter that specifies the encoding of the value. Options include `base64`, `hex`, `utf-8`, and `binary`. The default is `utf-8`.

Examples:

To hash the contents of a `secret` field with the SHA256 algorithm, run:

```
C.Mask.sha256(secret)
```

Copy

To hash the contents of an `authToken` field, where the value of the field is a base64-encoded string, with the SHA256 algorithm, run:

```
C.Mask.sha256(authToken, undefined, 'base64')
```

Copy

String Manipulation

C.Mask.random

Generates a random alphanumeric string.

Usage:

```
C.Mask.random(len?: string | number): string
```

Copy

Parameters:

Parameter	Type	Description
`len`	string or number	Length of the result; or, if a string, use its length.

Examples:

Assuming you have some identifying user information in a `username` field, you can replace it with a random string of the same length with:

```
C.Mask.random(username)
```

Copy

Input	Output
`McGoatFace`	`xY2KLHkyTg`

C.Mask.repeat()

Generates a repeating character or string pattern, such as `XXXX`.

Usage:

```
C.Mask.repeat(len?: number | string, char: string = 'X'): string
```

Copy

Parameters:

Parameter	Type	Description
`len`	string or number	Length of the result; or, if a string, use its length.
`char`	string	Pattern to repeat `len` times.

Examples:

The following example will repeat the `GOAT` string as many time as there are characters in the first parameter, `goat`, so four times:

```
C.Mask.repeat("goat", "GOAT")
```

Copy

10.2.5. C.Net - Network Methods

C.Net.cidrMatch()

```
C.Net.cidrMatch(cidrIpRange: string, ipAddress: string): boolean
```

Copy

Determines whether the supplied IPv4 or IPv6 `ipAddress` is inside the range of addresses identified by `cidrIpRange`.

Parameter	Type	Description
`cidrIpRange`	string	IP address range in CIDR format. E.g., `10.0.0.0/24`.
`ipAddress`	string	The IP address to test for inclusion in `cidrIpRange`.

Examples

(IPv4): `C.Net.cidrMatch('10.0.0.0/24', '10.0.0.100')` returns `true`.

(IPv6): `C.Net.cidrMatch('2001:db8::/32', '2001:db8:0:0:0:ffff:0:0')` returns `true`.

C.Net.communityIDv1()

```
C.Net.communityIDv1(sourceIP: string, destIP: string, sourcePort: number | undefined,
```

Copy

Calculates the Community ID for a given network flow. A Community ID is a unique identifier that is reversible - it is the same for both sides of a connection. It is calculated based on source and destination IP addresses, ports, a protocol, and (optionally) a seed (explained below). When the protocol is ICMP, the Community ID will be reversible provided that the network flow uses types with a clear request/response relationship (for example, Echo and Echo Reply).

Usage

This method returns a Base64-encoded string.

Provide IP addresses - whether IPv4 or IPv6 - as strings.

When the protocol is ICMP (v4 or v6), `sourcePort` corresponds to message type and `destPort` corresponds to message code. For other Layer 3 protocols, `sourcePort` and `destPort` should be `undefined`.

You can specify `proto` either as a number (recommended, to avoid ambiguity) or a string.

- If a string, `proto` must be the desired protocol's keyword as specified in the IANA protocol registry.
- For network flows coming from the Cribl Stream NetFlow & IPFIX Source, use the protocol name as-is - the Source will provide it in correct form.

The optional `seed` parameter defaults to `0`. If you want to generate different Community IDs for different use cases within the same flow, change `seed` to a different value for each use case.

Parameter	Type	Description
sourceIP	string	Source IPv4 or IPv6 address
destIP	string	Destination IPv4 or IPv6 address
sourcePort	number or undefined	Source port. For ICMP, use the ICMP type. For other portless protocols, use `undefined`.
destPort	number or undefined	Destination port. For ICMP, use the ICMP code. For other portless protocols, use `undefined`.
proto	number or string	IP protocol number, such as `6`, `17`, or `1`. May also be the protocol name, such as `'tcp'`, `'udp'`, or `'icmp'`.
seed	number	Optional seed value; defaults to `0`

Examples

Bidirectional TCP flow using the protocol number.

```
// Protocol number 6 is TCP

C.Net.communityIDv1('1.2.3.4','5.6.7.8',1122,3344,6)
// '1:wCb3OG7yAFWelaUydu0D+125CLM='

C.Net.communityIDv1('5.6.7.8','1.2.3.4',3344,1122,6)
// '1:wCb3OG7yAFWelaUydu0D+125CLM='
```

Copy

Bidirectional UDP flow using the protocol name.

```
C.Net.communityIDv1('1.2.3.4','5.6.7.8',1122,3344,'udp')
// '1:0Mu9InQx6z4ZiCZM/7HXi2WMhOg='

C.Net.communityIDv1('5.6.7.8','1.2.3.4',3344,1122,'udp')
// '1:0Mu9InQx6z4ZiCZM/7HXi2WMhOg='
```

Copy

ICMP Echo/Echo Reply flow.

```
// Protocol number 1 is ICMP
// Echo is Type=8, Code=0
// Echo Reply is Type=0, Code=0

// Echo (equivalent of request)
C.Net.communityIDv1('1.2.3.4','5.6.7.8',8,0,1)
// '1:crodRHL2FEsHjbv3UkRrfbs4bZ0='

// Echo Reply (equivalent of response)
C.Net.communityIDv1('5.6.7.8','1.2.3.4',0,0,1)
// '1:crodRHL2FEsHjbv3UkRrfbs4bZ0='
```

Copy

C.Net.isIp()

```
C.Net.isIp(value: string): boolean
```

Copy

Determines whether the value supplied is an IP address (either IPv4 or IPv6).

Parameter	Type	Description
`value`	string	The IP address to test.

C.Net.isIpV4()

```
C.Net.isIpV4(value: string): boolean
```

Copy

Determines whether the value supplied is an IPv4 address.

Parameter	Type	Description
`value`	string	The IP address to test.

C.Net.isIpV6()

```
C.Net.isIpV6(value: string): boolean
```

Copy

Determines whether the value supplied is an IPv6 address.

Parameter	Type	Description
`value`	string	The IP address to test.

C.Net.isIPV4Cidr()

```
C.Net.isIPV4Cidr(value: string): boolean
```

Copy

Determines whether a value supplied is an IPv4 CIDR range.

Returns `true` if the value is a valid IPv4 CIDR range; otherwise, `false`.

Parameter	Type	Description
`value`	string	String value that may be an IPv4 CIDR range.

C.Net.isIPV6Cidr()

```
C.Net.isIPV6Cidr(value: string): boolean
```

Copy

Determine whether a value supplied is an IPv6 CIDR range.

Returns `true` if the value is a valid IPv6 CIDR range; otherwise, `false`.

Parameter	Type	Description
`value`	string	String value that may be an IPv6 CIDR range.

C.Net.isIpV4AllInterfaces()

```
C.Net.isIpV4AllInterfaces(value): boolean
```

Copy

Determines whether the value supplied is the IPv4 all-interfaces address, typically used to bind to all IPv4 interfaces - i.e., '0.0.0.0'.

Parameter	Type	Description
`value`	any	The IP address to test.

C.Net.isIpV6AllInterfaces()

```
C.Net.isIpV6AllInterfaces(value): boolean
```

Copy

Determines whether the value supplied is an IPv6 all-interfaces address, typically used to bind to all IPv6 interfaces - one of: '::', '0:0:0:0:0:0:0:0', or '0000:0000:0000:0000:0000:0000:0000:0000'.

Parameter	Type	Description
`value`	any	the IP address to test.

C.Net.ipv6Normalize()

```
C.Net.ipv6Normalize(address: string): string
```

Copy

Normalizes an IPV6 address based on RFC draft-ietf-6man-text-addr-representation-04.

Parameter	Type	Description
`address`	string	The IPV6 address to normalize.

C.Net.isPrivate()

```
C.Net.isPrivate(address: string): boolean
```

Copy

Determines whether the supplied IPv4 address is in the range of private addresses per RFC1918.

Parameter	Type	Description
`address`	string	The IP address to test.

C.Net.parseAddressOrRangeString()

```
C.Net.parseAddressOrRangeString(source: string, mask?: string | number)
```

Copy

Parses a string representation of an IP or IP range.

Returns an `IPv4` or `IPv6` object. When the `source` argument is an IP range, this method returns the first IP in the range.

Parameter	Type	Description
`source`	string	The string to parse.
`mask`	string | number	Optional parameter containing the CIDR mask. Used only when `source` is a valid IP string.

C.Net.IPv4_CIDR_REGEX

```
C.Net.IPv4_CIDR_REGEX: RegExp
```

Copy

The regular expressions for matching IP CIDR.

10.2.6. C.Text - Text Methods

C.Text.entropy()

```
C.Text.entropy(bytes: Buffer | string): number
```

Copy

Computes the Shannon entropy of the given Buffer or string.

Returns the entropy value; or `-1` in case of an error.

Parameter	Type	Description
`bytes`	Buffer | string	value to undergo Shannon entropy computation.

C.Text.hashCode()

```
C.Text.hashCode(val: string | Buffer | number): number
```

Copy

Computes hashcode (djb2) of the given value.

Returns hashcode value.

Parameter	Type	Description
`val`	string | Buffer | number	value to be hashed.

C.Text.isASCII()

```
C.Text.isASCII(bytes: Buffer | string): boolean
```

Copy

Checks whether all bytes or chars are in the ASCII printable range.

Returns `true` if all chars/bytes are within ASCII printable range; otherwise, `false`.

Parameter	Type	Description
`bytes`	string | Buffer	value to check for character range.

`C.Text.isUTF8()`

```
C.Text.isUTF8(bytes: Buffer | string): boolean
```

Copy

Checks whether the given Buffer contains valid UTF8.

Returns `true` if bytes are UTF8; otherwise, `false`.

Parameter	Type	Description
`bytes`	Buffer | string	bytes to check.

`C.Text.parseWinEvent()`

```
C.Text.parseWinEvent(xml: string, nonValues: string[] = C.Text._WIN_EVENT_NON_VALUES)
```

Copy

Parses an XML string representing a Windows event into a compact, prettified JSON object. Works like `C.Text.parseXml`, but with Windows events, produces more-compact output. For a usage example, see Reducing Windows XML Events.

Returns an object representing the parsed Windows Event; or `undefined` if the input could not be parsed.

Parameter	Type	Description
`xml`	string	an XML string; or an event field containing the XML.
`nonValues`	string[]	array of string values. Elements whose value equals any of the values in this array will be omitted from the returned object. Defaults to `['-']`, meaning that elements whose value equals `-` will be discarded.

C.Text.parseXml()

```
C.Text.parseXml(xml: string, keepAttr: boolean = true, keepMetadata: boolean = false,
```

Copy

Parses an XML string and returns a JSON object. Can be used with Eval Function to parse XML fields contained in an event, or with ad hoc XML.

Returns an object representing the parsed XML; or `undefined` if the input could not be parsed. An input collection of elements will be parsed into an array of objects.

Parameter	Type	Description
`xml`	string	XML string, or an event field containing the XML.
`keepAttr`	boolean	whether or not to include attributes in the returned object. Defaults to `true`.
`keepMetadata`	boolean	whether or not to include metadata found in the XML. The `keepAttr` parameter must be set to `true` for this to work. Defaults to `false`. (Eligible metadata includes namespace definitions and prefixes, and XML declaration attributes such as encoding, version, etc.)
`nonValues`	string[]	array of string values. Elements whose value equals any of the values in this array will be omitted from the returned object. Defaults to `[]` (empty array), meaning discard no elements.

C.Text.relativeEntropy()

```
C.Text.relativeEntropy(bytes: Buffer | string, modelName: string = 'top_domains'): nu
```

Copy

Computes the relative entropy of the given Buffer or string.

Returns the relative entropy value, or `-1` in case of an error.

Parameter	Type	Description
`bytes`	Buffer | string	Value whose relative entropy to compute.

Parameter	Type	Description
`modelName`	string	Optionally, override the default `$CRIBL_HOME/data/lookups/model_relative_entropy_top_domains.csv` model used to test the input. Create a custom lookup file with the same column and value structure as the default, and store it in the same path, as `model_relative_entropy_<custom-name>.csv`. To reference it, pass your `<custom-name>` substring as the `modelName` parameter.

When using `modelName` in a distributed deployment, the corresponding paths are `$CRIBL_HOME/groups/<worker-group-name>/data/lookups/`. Creating your custom lookup file via Cribl Stream's UI will automatically set the appropriate paths.

10.2.7. C.Time - Time Methods

C.Time.adjustTZ()

```
C.Time.adjustTZ(epochTime: number, tzTo: string, tzFrom: string = 'UTC'): number | ur
```

Copy

Adjusts a timestamp from one timezone to another.

Returns the adjusted timestamp, in Unix time (ms).

Parameter	Type	Description
`epochTime`	number	Timestamp to adjust, in Unix time.
`tzTo`	string	Timezone to adjust to (full name).
`tzFrom` (optional)	string	Timezone of the timestamp.

Cribl relies on this list of TZ Database Time Zones.

Note that these are timezone identifiers (full names), not abbreviations such as "CET", "PST", "EST", and so on.

Examples

To adjust a timestamp found in the `_time` field to a selected timezone (here, Australia/Melbourne), use:

```
C.Time.adjustTZ(_time, "Australia/Melbourne")
```

Copy

The method returns the timestamp in milliseconds. If you want to further operate on it using methods such as `strftime()`, which takes as parameter Unix time in seconds, you can divide it by 1000:

```
C.Time.strftime(C.Time.adjustTZ(_time, "Australia/Melbourne") / 1000, "%d %B %Y %H %M
```

Copy

C.Time.clamp()

```
C.Time.clamp(date: T, earliest: T, latest: T, defaultDate?: T ): T  | undefined
```

Copy

Constrains a parsed timestamp to realistic earliest and latest boundaries.

Returns a JavaScript Date object or timestamp, depending on the format of the `date` parameter.

Parameter	Type	Description
`date`	Date | number	Timestamp to constrain, in Unix time (ms) or JavaScript Date format.
`earliest`	Date | number	Earliest allowable timestamp, in Unix time (ms) or JavaScript Date format.
`latest`	Date | number	Latest allowable timestamp, in Unix time (ms) or JavaScript Date format.
`defaultDate` (optional)	Date | number	Default date, in Unix time (ms) or JavaScript Date format. This date is used for values that fall outside the `earliest` or `latest` boundaries.

Examples

Suppose you have a date in a `"YYYY-MM-DD"` format in a `time` field. To make sure this date does not exceed the range of 1 Jan 2020 - 1 Jan 2025, you can use:

```
C.Time.clamp(time, "2020-01-01", "2025-01-01")
```

Copy

Input	Output
`"2016-02-11"`	`"2020-01-01"` (set to the earliest value)
`"2026-01-01"`	`"2025-01-01"` (set to the latest value)
`"2023-12-24"`	`"2023-12-24"` (unchanged, because it fits inside the defined range)

To make sure that all dates that fall outside the bounds are changed to a specific value, add the third parameter:

```
C.Time.clamp(time, "2020-01-01", "2025-01-01", "2022-02-22")
```

Copy

Input	Output
`"2016-02-11"`	`"2022-02-22"` (set to the default value)
`"2026-01-01"`	`"2022-02-22"` (set to the default value)
`"2023-12-24"`	`"2023-12-24"` (unchanged, because it fits inside the defined range)

C.Time.strftime()

```
C.Time.strftime(date: number | Date | string, format: string, utc: boolean = true): s
```

Copy

Formats a Date object or timestamp number as a time string, using d3js time format.

Returns representation of the given date in the specified format.

Parameter	Type	Description
`date`	number | Date | string	Date to format, in Unix time (in seconds) or JavaScript Date format.
`format`	string	New format for the date.
`utc` (optional)	boolean	Whether to output the time in UTC (`true`), rather than in local timezone.

Examples

To transform a `_time` field containing a timestamp (in Unix time) to a string with time formatted, for example: `15 November 2023, 08:03`, you can use:

```
C.Time.strftime(_time, "%d %B %Y, %H:%M")
```

Copy

Some common formats (assuming timestamp in Unix time of 1702460134) are:

Format	Output
`"%Y-%m-%d"` (ISO date)	`"2023-12-13"`
`"%d %B %Y %H:%M:%S"`	`"13 December 2023 09:35:34"`
`"%x %X"` (the locale's date and time format)	`"12/13/2023 9:35:34 AM"`

For detailed reference of the date and time tokens, see d3js reference or Auto Timestamp.

C.Time.strptime()

```
C.Time.strptime(str: string, format: string, utc: boolean = true, strict: boolean = f
```

Copy

Extracts time from a string using d3js time format that defines the format of the incoming string.

Returns a parsed JavaScript Date object, or `null` if the specified `format` did not match.

Parameter	Type	Description
`str`	string	Timestamp to parse.
`format`	string	Format to parse.
`utc` (optional)	boolean	Whether to interpret times as UTC (`true`), rather than as local time.
`strict` (optional)	boolean	Whether to return `null` if there are any extra characters after timestamp.

Examples

To convert a string in the `"YYYY-MM-DD"` format, for example `"2023-12-12"`, to a Date object, use:

```
C.Time.strptime(F, "%Y-%m-%d")
```

Copy

Additionally, to interpret the time in UTC, and make sure null is returned if the timestamp contains any trailing characters, use:

```
C.Time.strptime(F, "%Y-%m-%d", true, true)
```

Copy

For detailed reference of the date and time tokens, see d3js reference or Auto Timestamp.

C.Time.timestampFinder()

```
C.Time.timestampFinder(utc?: boolean).find(str: string): IAutoTimeParser
```

Copy

Extracts time from the specified field, using the same algorithm as the Auto Timestamp Function and Event Breakers.

Returns representation of the extracted time; truncates timestamps to three-digit (milliseconds) resolution, omitting trailing zeros.

Parameter	Type	Description
`utc`	boolean	Whether to output the time in UTC (`true`), rather than in local timezone.
`str`	string	The field in which to search for the time.

Examples

To find timestamps in the `_raw` field and output them in UTC, you can use:

```
C.Time.timestampFinder(true).find(_raw)
```

Copy

You can then format the string output with `strftime()`:

```
C.Time.strftime(C.Time.timestampFinder(true).find(_raw), "%d.%m.%Y, %H.%M")
```

Copy

C.Time.timePartition()

```
C.Time.timePartition(date: number | Date | string, level: 'd' | 'h'): string | undefi
```

Copy

Provides the part of the file path prefix that contains the date.

Returns a string in the format `YYYY/MM/DD` for `d` and `YYYY/MM/DD/HH` for `h`.

Parameter	Type	Description
`date`	number | Date | string	Datetime to extract the file path prefix for.
`level`	string	Granularity of the partitioning, `d` for daily, and `h` for hourly.

Examples

To get the file path prefix from the `_time` field:

```
C.Time.timePartition(_time)
```

Copy

This will return the prefix in the form of, for example: `2024/04/23`.

C.Time.s3TimePartition()

```
C.Time.s3TimePartition(date: number | Date | string, level: 'd' | 'h'): string | unde
```

Copy

Provides the full S3 file path prefix for a given datetime.

Returns the file path prefix as a string.

Parameter	Type	Description
`date`	number | Date | string	Datetime to extract the file path prefix for.
`level`	string	Granularity of the partitioning, `d` for daily, and `h` for hourly.

Examples

To get the S3 file path prefix from the `_time` field:

```
C.Time.s3TimePartition(_time)
```

Copy

This will return the prefix in the form of, for example: `hjhg-2024/aj-04/ai-23`.

10.2.8. Miscellaneous Expression Methods

C.Schema - Schema Methods

`C.Schema`

```
C.Schema: (id: string) => SchemaValidator
```

Copy

`C.Schema.validate()`

```
C.SchemaValidator.validate(data: any): boolean
```

Copy

Validates the given object against the schema.

Returns `true` when schema is valid; otherwise, `false`.

Parameter	Type	Description
`data`	any	Object to be validated.

Examples

To validate whether a `myField` conforms to `schema1`, you can use:

```
C.Schema('schema1').validate(myField)
```

Copy

See Schema Library for more details.

C.Secret - Secrets-Management Methods

C.Secret()

```
C.Secret: (id: string, type?: string): ISecret
C.Secret(id: string, type: 'keypair') => IPairSecret
C.Secret(id: string, type: 'text') => ITextSecret
C.Secret(id: string, type: 'credentials') => ICredentialsSecret
```

Copy

Returns a secret matching the specified ID.

Parameter	Type	Description
`id`	string	ID of the secret.
`type` (optional)	'text' | 'keypair' | 'credentials'	Type of the secret.

Examples

To return a text secret with ID `victorias` (or with undefined, if no such secret exists), use:

```
C.Secret('victorias', 'text')
```

Copy

You can also get attributes of secrets with the following expressions:

```
C.Secret('api_key', 'keypair').secretKey
C.Secret('secret_hash', 'text').value
C.Secret('user_pass', 'credentials').password
```

Copy

Common returned attributes for `ISecret` objects:

- `secretType` - one of `keypair`, `text`, or `credentials`.
- `description` (optional) - the secret description.
- `tags` (optional) - a comma separated list of tags.

Additional returned attributes for `IPairSecret` objects:

- `apiKey` - the API key value
- `secretKey` - the Secret key value

Additional returned attributes for `ITextSecret` objects:

- `value` - the text value

Additional returned attributes for `ICredentialsSecret` objects:

- `username` - the username value
- `password` - the password value

See Securing Cribl Stream > Secrets for more details.

C.env - Environment

`C.env`

```
C.env: {[key: string]: string;}
```

Copy

Returns an object containing Cribl Stream's environment variables. All internal variables, that is, those starting with `CRIBL_`, are accessible in this method.

Examples

To return the parent of Cribl's `bin` directory (generally `/opt/cribl`), use:

```
C.env.CRIBL_HOME
```

Copy

To return the hostname of the machine where Cribl Stream is running, use:

```
C.env.HOSTNAME
```

Copy

C.os - System Methods

`C.os.hostname()`

Returns hostname of the system running this Cribl Stream instance.

C.vars - Global Variables

See Global Variables Library for more details.

> Worker Group variables are designed specifically for configuration binding. They are not accessible using the `C.vars` expression in Pipeline Functions. You can only bind these variables to Source and Destination configuration fields that display the **Variable** icon ◈.
>
> The only exception is the Redis Function, which can resolve Worker Group variables at runtime to manage external state or lookups.

C.version - Cribl Stream Versions

`C.version`

Returns the Cribl Stream version currently running.

`C.confVersion`

Returns the commit hash of the Worker Node's current config version. (Evaluates only against live data sent through Worker Nodes. Values will be undefined in the Leader's Preview pane.)

C.Misc - Miscellaneous Utility Methods

`C.Misc.zip()`

```
C.Misc.zip(keys: string[], values: any[], dest?: any): any
```

Copy

Sets the given keys to the corresponding values on the given `dest` object. If `dest` is not provided, a new object will be constructed.

Returns object on which the fields were set.

Parameter	Type	Description
`keys`	string[]	Field names corresponding to keys.
`values`	any[]	Values corresponding to values.
`dest`	any	Object on which to set field values.

Examples

Let's take the following expression:

```
people = C.Misc.zip(titles, names)
```

Copy

If sample data contains: `titles=['ceo', 'svp', 'vp']`, `names=['foo', 'bar', 'baz']`, this expression create an object called `people`, with key names from elements in `titles`, and with corresponding values from elements in `names`.

Result: `"people": {"ceo": "foo", "svp": "bar", "vp": "baz"}`

`C.Misc.uuidv4()`

```
C.Misc.uuidv4(): string
```

Copy

Returns a version 4 (random) UUID in accordance with RFC-4122.

Examples

To create a UUIDv4, use:

```
C.Misc.uuidv4()
```

Copy

Result: a conforming UUIDv4, such as `58d8be36-4db0-4b1c-ac80-28bb03c45e0d`. It is highly improbable that two version 4 UUIDs will ever have the same value.

C.Misc.uuidv5()

```
C.Misc.uuidv5(name: string, namespace: string): string
```

Copy

Returns a version 5 (namespaced) UUID in accordance with RFC-4122 for the given `name` and `namespace`. If `namespace` is not a valid UUID, this function will fail.

Parameter	Type	Description
`name`	string	Any arbitrary name to use in UUID generation.
`namespace`	string	One of `DNS`, `URL`, `OID`, or `X500` to use a predefined namespace, or else a valid UUID.

Examples

To create a UUIDv5 with the predefined DNS namespace, use:

```
C.Misc.uuidv5('example', 'DNS')
```

Copy

Result: a UUID that is highly likely to be the same for the same name and namespace. In this case, the result would be `7cb48787-6d91-5b9f-bc60-f30298ea5736`.

C.Misc.validateUUID()

```
C.Misc.validateUUID(maybeUUID: string): boolean
```

Copy

Returns `true` if `maybeUUID` is a valid UUID of any version, and false otherwise.

Parameter	Type	Description
`maybeUUID`	string	A string to test for a valid UUID.

Examples

```
C.Misc.validateUUID(C.Misc.uuidv4())
```

Copy

Result: returns `true`

```
C.Misc.validateUUID('clearly not a UUID')
```

Copy

Result: returns `false`

C.Misc.getUUIDVersion()

```
C.Misc.getUUIDVersion(uuid: string): number
```

Copy

Returns a `number` of the UUID version given by `uuid` if it is a valid UUID, otherwise `undefined`.

Parameter	Type	Description
`uuid`	string	A UUID for which to determine the version.

Examples

```
C.Misc.getUUIDVersion(C.Misc.uuidv4())
```

Copy

Result: `4`

```
C.Misc.getUUIDVersion(C.Misc.uuidv5('example', 'X500'))
```

Copy

Result: `5`

C.WorkerGroupId

```
C.WorkerGroupId: string
```

Copy

Returns the name of the current Worker Group. Use this expression to enrich logs or metrics with the Worker Group identity, enabling filtering and grouping in dashboards and analytics tools. This method always returns a string, even in single-node deployments.

Prefer `C.WorkerGroupId` over `C.env.CRIBL_GROUP_ID` where possible. The environment variable may be unset or stale, whereas `C.WorkerGroupId` always reflects the current Worker Group assignment from your configuration.

Example

To add the current Worker Group name to an event field:

```
__e.worker_group = C.WorkerGroupId
```

10.3. CLI Reference

Command line interface basics and reference

In addition to starting and stopping the Cribl Stream server, the Cribl Stream command line interface enables you to initiate many configuration and administrative tasks directly from your terminal.

The command-line interface is only available to on-prem or hybrid customers.

To use the CLI:

1. First check that you have read/write access to the directory where Cribl Stream is installed. Contact your Cribl administrator to request access if needed.
2. Navigate to the directory where Cribl Stream is installed:

```
cd $CRIBL_HOME/bin
```

Copy

3. Execute a command. CLI commands have this basic syntax:

```
./cribl <command> <sub-command> <options> <arguments>
```

Copy

Not all commands have sub-commands.

4. To see the documentation for any command, append the `--help` option. For example:

```
./cribl vars --help

./cribl vars get --help

./cribl vars get -i myArray --help
```

Copy

Prevent and Troubleshoot CLI Problems

Immediate Execution

As indicated in the command reference and sample output, some commands take effect immediately.

Commands that require additional input will echo the sub-commands, options, and arguments they expect.

Persistent Volumes

If you start Cribl Stream with the `CRIBL_VOLUME_DIR` variable, all subsequent CLI commands should have this variable defined. Otherwise, those commands will apply the Cribl Stream default directories, yielding misleading results.

You can set `CRIBL_VOLUME_DIR` as an environment variable, or you can explicitly include it in each command, as in this example:

```
CRIBL_VOLUME_DIR=<writable-path-name> /opt/cribl/bin/cribl status
```

When set, `$CRIBL_VOLUME_DIR` overrides `$CRIBL_HOME`.

Avoid setting `$CRIBL_VOLUME_DIR` to an existing folder because it creates predefined folders in the specified directory. If that directory already contains folders with those names, Cribl Stream will overwrite those directories.

Failover Mode

In failover mode, set the `CRIBL_CONF_DIR` variable to your failover directory, ensuring your CLI commands access the correct configuration directory. For example, `CRIBL_CONF_DIR=/<path_to_failover_volume> cribl diag create`.

Your failover directory is either defined by the environment variable `CRIBL_DIST_LEADER_FAILOVER_VOLUME` (alternatively `CRIBL_DIST_MASTER_FAILOVER_VOLUME`), or defined in the instance.yml configuration file.

Commands Available

To see a list of available commands, enter `./cribl` alone (or the equivalent `./cribl help`). To execute a command, or to see its required parameters, enter `./cribl <command>`.

Command	Description
auth	Command to log into or out of the product
boot-start	Command to enable or disable product boot-start

Command	Description
cloud-workspace	Command to update a Leader with new config that allows it to connect to the Cribl.Cloud Leader and send it usage metrics
decrypt	Command to decrypt data with a secret key
diag	Command to manage diagnostic bundles
encrypt	Command to encrypt data with a secret key
git	Command to manage Worker Group or Fleet configuration
help	Displays a list of commands and their help
keys	Command to manage encryption keys
limits	Command to control the availability of Cribl features
mode-master	Command to configure an instance as a Leader
mode-single	Command to configure an instance as a Single-instance deployment
mode-edge	Command to configure Cribl Edge as a Single-instance deployment
mode-worker	Command to configure Cribl Stream as a Worker instance
mode-managed-edge	Command to configure Cribl Edge as an Edge Node
mode-outpost	Command to configure an instance as a Cribl Outpost
nc	Command to listen a port for traffic and output stats and data
node	Command to execute a JavaScript file
pack	Command to mange Cribl packs
parquet	Command to view a Parquet file, its metadata, or its schema
pipe	Command to feed stdin to a Pipeline
pq	Command for persistent queue performance benchmarking
reload	Command to reload the product
restart	Command to restart the product
scope	Scope a Linux Command via AppScope

Command	Description
start	Command to start the product
status	Command to display the product status
stop	Command to stop the product
vars	Command to manage global variables
version	Command to display product version

10.3.1. Auth

Log into or out of Cribl Stream.

Sub-commands:

- `auth login`
- `auth logout`
- `auth mf`

Sub-commands and Options

`auth login`

Login with a username and password or use a file with credentials.

Usage:

```
./cribl auth login -H http://localhost:9000 -u example-username -p example-password
./cribl auth login -H http://localhost:9000 -f example-credentials-file.txt
```

Copy

See additional `auth login` Examples.

Arguments:

Option	Definition
`-H <host>`	Host URL. Example: `http://localhost:9000`.
`-u <username>`	Username.
`-p <password>`	Password.
`-f <file>`	File with credentials.

Arguments of the `login` subcommand correspond to the following environment variables:

Argument	Variable
<username>	CRIBL_USERNAME
<password>	CRIBL_PASSWORD
<host>	CRIBL_HOST

`auth login` Examples

Launch interactive login:

```
$CRIBL_HOME/bin/cribl auth login
```

Append credentials as command arguments:

```
$CRIBL_HOME/bin/cribl auth login -h <url> -u <username> -p <password>
```

All `-h` and `host` arguments are optional, provided that the `cribl.yml` file's `api:` section lists the API host and port.

Provide credentials in environment variables:

```
CRIBL_HOST=<url> CRIBL_USERNAME=<username> CRIBL_PASSWORD=<password>
$CRIBL_HOME/bin/cribl auth login
```

Provide credentials in a file:

```
$CRIBL_HOME/bin/cribl auth login -f <path/to/file>
```

Corresponding file contents:

```
host=<url>
username=<username>
password=<password>
```

Copy

`auth logout`

Logout from the current session.

Usage:

```
./cribl auth logout
```

Copy

auth mf

Control multi-factor authentication (PIV).

Usage:

```
./cribl auth mf -e true
```

Copy

See additional `auth mf` Examples.

Arguments:

Option	Definition
`-e <enable>`	Disable or enable multi-factor authentication using PIV. Accepts: [“true”, “false”].
`-u <usernameRegex>`	Username regex pattern (defaults to ‘/CN=(?:\w+.)+(\d+)/”).
`-f <usernameField>`	Field from which username is to be extracted (defaults to ‘subject’). Accepts: [subject,san]
`-o <accessControlAllowOrigin>`	Access control allow origin header. Example: “https://127.0.0.1:9000” or “https://fqdn”
`-a <apiServerUrl>`	API server internal FQDN or IP:PORT combo. Make sure this is also part of server TLS certificate SAN->IP or SAN->DNS (such as “https://[IP]:[PORT]” or “https://internal-fqdn”).

auth mf Examples

Check the status of multi-factor authentication using Personal Identity Verification (PIV):

```
$CRIBL_HOME/bin/cribl auth mf
```

The response includes the status (disabled or enabled) and the current configuration, along with notifications for unconfigured values and issues like failure to restart or enable TLS. For example:

```
PIV multi-factor authentication is disabled.
Username field not defined. Please make sure to configure it.
Username regex pattern: /(?:UPN:)([^,]+)(?:,|$)/
Access control allow origin header: https://127.0.0.1:9000
Api server url: https://127.0.0.1:9000
You will need to restart instance before your changes take full effect.
Warning: TLS is not enabled. PIV will not function until TLS is successfully configur
```

Copy

Enable multi-factor authentication login using PIV:

```
$CRIBL_HOME/bin/cribl auth mf -e true
```

The response confirms that PIV authentication is enabled and lists the current configuration. For example:

```
PIV multi-factor authentication is enabled.
Username field: san
Username regex pattern: /(?:UPN:)([^,]+)(?:,|$)/
Access control allow origin header: https://127.0.0.1:9000
Api server url: https://127.0.0.1:9000
```

Copy

Disable multi-factor authentication login using PIV:

```
$CRIBL_HOME/bin/cribl auth mf -e false
```

The response confirms that PIV authentication is disabled and lists the current configuration. For example:

```
PIV multi-factor authentication is disabled.
Username field: san
Username regex pattern: /(?:UPN:)([^,]+)(?:,|$)/
Access control allow origin header: https://127.0.0.1:9000
Api server url: https://127.0.0.1:9000
```

Copy

For a detailed example that uses the mf sub-command to configure and enable PIV authentication, see Personal Identity Verification (PIV) Authentication.

10.3.2. Boot-Start

Enables or disables Cribl Stream boot-start.

Sub-commands:

- `boot-start disable`
- `boot-start enable`

Sub-commands and Options

`boot-start disable`

Disable boot-start.

Usage:

```
./cribl boot-start disable -m initd
```

Copy

Arguments:

Option	Definition
`-m <manager>`	Init manager. Accepts: `systemd` or `initd`.
`-c <configDir>`	Config directory for the init manager.

`boot-start enable`

Enable boot-start.

Usage:

```
./cribl boot-start enable -m initd
```

Copy

Sample response:

```
Enabling Cribl to be managed by initd...
boot-start enable command needs root privileges...
Enabled Cribl to be managed by initd as user=root.
```

Copy

Arguments:

Option	Definition
`-m <manager>`	Init manager. Accepts: `systemd` or `initd`.
`-u <user>`	User to run the instance as.
`-c <configDir>`	Config directory for the init manager.

10.3.3. Cloud-Workspace

Updates an on-prem Leader Node with a new config in the `instance.yml` file called `cloudWorkspace`. With this config, the local Leader will connect to the Cribl.Cloud Leader and send it usage metrics.

Usage:

```
./cribl cloud-workspace -H <host> -p <port> # etc: flags
```

Copy

Arguments:

Option	Definition
`-d <deploymentId>`	Deployment ID used to report telemetry on multiple deployments.
`-H <host>`	Hostname/IP. Defaults to `0.0.0.0`.
`-p <port>`	Port. Defaults to `4200`.
`-n <certName>`	Name of the saved certificate to use. Mutually exclusive with `-k` or `-c`.
`-k <privKeyPath>`	Path on server to the private key to use. PEM format. Can reference `$ENV_VARS`.
`-c <certPath>`	Path on server to the certificate to use. PEM format. Can reference `$ENV_VARS`.
`-u <authToken>`	Authentication token to include as part of the connection header.
`-S <tls>`	Sets, or disables, secure communication between Leader and Worker/Edge Nodes via TLS. Accepts: ["true", "false"].
`-i <ipWhitelistRegex>`	Regex matching IP addresses that are allowed to establish a connection.
`-r <resiliency>`	Resiliency mode on the Leader. Accepts: ["none", "failover"].
`-v <failoverVolume>`	For resiliency=failover, the volume to use as shared storage.

10.3.4. Decrypt

Decrypts data with a secret key. See the related encrypt command.

Usage:

```
./cribl decrypt -v 12345 -s my-path-name
```

Copy

Arguments:

Option	Definition
`-v <value>`	Value to decrypt.
`-s <criblSecretPath>`	The cribl.secret decrypt key path. If you do not specify a cribl.secret file with -s options, the default cribl.secret is used.

10.3.5. Diag

Manages diagnostic bundles. For failover mode, see how to prevent and troubleshoot CLI problems.

Sub-commands:

- `diag create`
- `diag cpuprofile`
- `diag heapsnapshot`
- `diag list`
- `diag perf`
- `diag send`

Sub-commands and Options

`diag create`

Create a diagnostic bundle for this instance.

Usage:

```
./cribl diag create
```

Copy

Arguments:

Option	Definition
`-d`	Run create in debug mode.
`-j`	Do not append '.txt' to js files.
`-k`	Include metrics for the top 10 Sources, Destinations, Pipelines, Routes, and Packs from the last 24 hours. This matches the **Include top talkers** option in the UI and is separate from the main metrics store controlled by -M.
`-t <maxIncludeJobs>`	Maximum number of jobs (and all associated tasks) to include in the bundle.

Option	Definition
`-M`	Exclude the persistent internal metrics store from the bundle. This reduces bundle size, especially on Leaders and single-instance deployments, but does not remove other metric-related data (such as logs or data added with -k).
`-g`	Exclude git from bundle.
`-i`	Include install logs.

Windows PowerShell usage:

```
PS > $env:CRIBL_VOLUME_DIR='c:\ProgramData\Cribl'; & 'c:\Program
Files\Cribl\bin\cribl.exe' diag create
```

The Windows command requires the `CRIBL_VOLUME_DIR` environment variable. If your deployment doesn't use the default path (`c:\ProgramData\Cribl`), verify that `CRIBL_VOLUME_DIR` points to the correct directory. Otherwise, you might get empty diag files.

Sample response:

```
Created diagnostic bundle at c:\ProgramData\Cribl\diag\edge-DESKTOP-C9BEMQ2-202409251
```

Copy

Windows CMD usage:

```
> set "CRIBL_VOLUME_DIR=c:\ProgramData\Cribl" && "c:\Program
Files\Cribl\bin\cribl.exe" diag create
```

Sample response:

```
Created diagnostic bundle at c:\ProgramData\Cribl\diag\edge-DESKTOP-C9BEMQ2-202409251
```

Copy

`diag cpuprofile`

Collect a 30-second CPU profile and place in the `diag` directory.

Usage:

```
./cribl diag cpuprofile -p 12345
```

Copy

Sample response:

```
Created a Cribl diagnostic bundle at /opt/cribl/diag/<product>-zedborcdb72f-202108201
```

Copy

Arguments:

Option	Definition
`-p <pid>`	The pid of the process to dump the CPU profile.

`diag heapsnapshot`

Generate heap snapshot of a Cribl process and place in the `diag` directory.

Usage:

```
./cribl diag heapsnapshot -p 12345
```

Copy

Sample response:

```
Heap-1672574400000-12345.heapsnapshot
```

Copy

The response format is `Heap-<epoch-timestamp>-<pid>.heapsnapshot`.

Arguments:

Option	Definition
`-p <pid>`	The pid of the process to dump the heap snapshot.

`diag list`

List existing diagnostic bundles.

`diag perf`

Collect a 30-second CPU profile and a heapsnapshot and place both in the `diag` directory.

Option	Definition
`-p <pid>`	The pid of the process to dump both the CPU profile and heap snapshot.

`diag send`

Send diagnostics bundle to Cribl Support.

```
./cribl diag send -c 00001234
```

Copy

Arguments:

Option	Definition
`-c <caseNumber>`	Cribl Support Case Number.
`-p <path>`	Diagnostic bundle path. If empty, it creates a new bundle.

10.3.6. Encrypt

Encrypts data with a secret key. See the related decrypt command.

Usage:

```
./cribl encrypt -v my_token -s my-path-name
```

Copy

Sample response:

```
LzmJ3uqA6WXtXjaLPwuM++WX84o3fPxBaubA6Q0look=
```

Copy

Arguments:

Option	Definition
`-v <value>`	Value to encrypt.
`-s <criblSecretPath>`	The cribl.secret encrypt key path. If you do not specify a cribl.secret file with -s options, the default cribl.secret is used.

10.3.7. Git

Manages Worker Group configuration.

Sub-commands:

- `git commit`
- `git commit-deploy`
- `git deploy`
- `git list-groups`

Sub-commands and Options

`git commit`

Commit Worker Group or Fleet configuration changes.

Usage:

```
./cribl git commit -g production_logs -m "Add syslog pipeline"
./cribl git commit -g webserver_data -m "Remove Parser function from webhook pipeline
```

Copy

Arguments:

Option	Definition
`-g <group>`	Group ID.
`-m <message>`	Commit message.

`git commit-deploy`

Commit and also deploy Worker Group or Fleet configuration changes.

Usage:

```
./cribl git commit-deploy -g production_logs -m "Add syslog pipeline"
./cribl git commit-deploy -g webserver_data -m "Remove Parser function from webhook p
```

Copy

Option	Definition
`-g <group>`	Group ID.
`-m <message>`	Commit message.

`git deploy`

Deploy Worker Group or Fleet configuration changes.

Usage:

```
./cribl git commit-deploy -g production_logs -v 7c04de1
```

Copy

Sample response:

```
Successfully committed version 7c04de1
```

Copy

Arguments:

Option	Definition
`-g <group>`	Group ID.
`-v <version>`	Deploy version.

`git list-groups`

List Worker Groups or Fleets.

10.3.8. Help

Displays a list of commands with a description (help) for each. Defaults to a selection of generally useful commands.

Usage:

```
./cribl help [-a]
```

Copy

Sample response:

```
Software version: 4.4.2
Usage: [sub-command] [options] [args]

Commands:
help              - Display help
mode-edge         - Configure instance in Edge mode
mode-managed-edge - Configure instance in Managed-Edge mode
mode-master       - Configure instance in Leader mode
mode-single       - Configure instance in Single-Instance mode
mode-worker       - Configure instance in Worker mode
reload            - Reload instance
restart           - Restart instance
start             - Start instance
status            - Display status
stop              - Stop instance
version           - Print version
```

Copy

Arguments:

Option	Definition
`-a`	Display the list of all commands, except for `scope`.

10.3.9. Keys

Manages encryption keys.

You must append the `-g <group>` argument to specify a Worker Group/Fleet. As a fallback, append the argument `-g default`. For example: `./cribl keys list -g default`.

Sub-commands:

- `keys add`
- `keys list`

Sub-commands and Options

`keys add`

Add encryption keys.

Usage:

```
./cribl keys add -g default
./cribl keys add -g production_logs
```

Copy

Sample response:

```
Adding key succeeded. Key count=1
```

Copy

Arguments:

Option	Definition
`-a <algorithm>`	Encryption algorithm. Supported values: aes-256-cbc (default), aes-256-gcm.
`-c <keyclass>`	Key class to set for the key.

Option	Definition
`-k <kms>`	KMS to use. Must be configured; see cribl.yml.
`-e <expires>`	Expiration time, in Unix time.
`-i`	Use an initialization vector. (IV size configurable for algorithm aes-256-gcm & fixed to 16 for algorithm aes-256-cbc)
`-s <ivSize>`	(For algo 'aes-256-gcm' only) Initialization vector size (bytes). Supported values: 12 (default), 13, 14, 15, 16
`-g <group>`	Worker Group ID.

`keys list`

List encryption keys.

Usage:

```
./cribl keys add -g default
./cribl keys add -g production_logs
```

Copy

Arguments:

Option	Definition
`-g <group>`	Worker Group ID.

10.3.10. Limits

Controls the availability of Cribl features. See Scripts for more information.

Usage:

```
./cribl limits -s true
./cribl limits -s false
```

Copy

Arguments:

Option	Definition
`-s <enable>`	Enable custom scripts API. Accepts: [“true”, “false”].

10.3.11. Mode-Master

Configures instance as a Leader.

For more information, see:

- Leader High Availability/Failover
- Set Up Leader and Worker Nodes

Usage:

```
./cribl mode-master -r failover -v /tmp/shared
```

Copy

Arguments:

Option	Definition
`-d <deploymentId>`	Deployment ID for reporting telemetry on multiple deployments.
`-H <host>`	Hostname/IP. Defaults to `0.0.0.0`.
`-p <port>`	Port. Defaults to `4200`.
`-f <uiAccess>`	Controls whether the distributed port responds to requests on behalf of the API process. The default port is `4200`. When disabled, `4200` only responds to Worker/Edge Node queries for config bundles and distributed upgrades. Accepts: ["true", "false"]. Defaults to `true` for backward compatibility.
`-n <certName>`	Name of the saved certificate to use. Mutually exclusive with -k or -c.
`-k <privKeyPath>`	Path on server to the private key to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-c <certPath>`	Path on server to the certificate to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-u <authToken>`	Authentication token to include as part of the connection header.
`-S <tls>`	Sets, or disables, secure communication between Leader and Worker/Edge Nodes via TLS. Accepts: ["true", "false"].

Option	Definition
`-i <ipWhitelistRegex>`	Regex matching IP addresses that are allowed to establish a connection.
`-r <resiliency>`	Resiliency mode for the Leader. Accepts: [“none”, “failover”].
`-v <failoverVolume>`	For resiliency=failover, the volume to use as shared storage.

10.3.12. Mode-Single

Configures node as a Single-instance Cribl Stream deployment.

Usage:

```
./cribl mode-single -d d0e301e4-f961-4f50-956a-5153224b34e3
```

Copy

Arguments:

Option	Definition
`-d <deploymentId>`	Deployment ID for reporting telemetry on multiple deployments.

10.3.13. Mode-Edge

Configures node as a Single-instance Cribl Edge deployment.

Usage:

```
./cribl mode-edge -d d0e301e4-f961-4f50-956a-5153224b34e3
```

Copy

Arguments:

Option	Definition
`-d <deploymentId>`	Deployment ID for reporting telemetry on multiple deployments.
`-H <host>`	Hostname/IP. Defaults to `127.0.0.1`.
`-p <port>`	Port. Defaults to `9420`.
`-s <socket>`	Location of the AppScope socket.

10.3.14. Mode-Worker

Configures instance as a Cribl Stream Worker Node.

Running this command resets all the related options. This means that even if you want to change only one option, you also need to provide values for all the other arguments.

When you generate bootstrap scripts through the UI to add or update Workers, these scripts automatically set the `-S` option according to the Leader TLS configuration.

For more information, see Set Up Leader and Worker Nodes.

Usage:

```
./cribl mode-worker -H 127.0.0.1 -p 9420 -u myToken
```

Copy

The `-H <host> -p <port> -u <authToken>` parameters are required for successful connection to the Leader.

Arguments:

Option	Definition
`-d <deploymentId>`	Deployment ID for reporting telemetry on multiple deployments.
`-H <host>`	Leader's hostname or IP address. Required.
`-p <port>`	Leader's cluster communications port. Example: `4200`. Required.
`-n <certName>`	Name of the saved certificate to use. Mutually exclusive with `-k` or `-c`.
`-k <privKeyPath>`	Path on server to the private key to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-c <certPath>`	Path on server to the certificate to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-u <authToken>`	Authentication token to include as part of the connection header. Required.

Option	Definition
`-S <tls>`	Sets, or disables, secure communication between Leader and Worker/Edge Nodes via TLS. Accepts: [“true”, “false”].
`-e <envRegex>`	Regex to select environment variables to report to Leader. For example: `"/test/"`.
`-t <tags>`	Tag values to report to Leader.
`-g <group>`	Worker Group to report to Leader.

In Cribl.Cloud, get the token for your Workspace from the Worker Node modal (**Add/Update Worker Node**) from the **Auth token** field.

In an on-prem deployment, get the token from **Settings** > **Global** > **System** > **Distributed Settings** > **Leader Settings** > **Auth token**.

10.3.15. Mode-Managed-Edge

Configures instance as a Cribl Edge Node.

Running this command resets all the related options. This means that even if you want to change only one option, you also need to provide values for all the other arguments.

When you generate bootstrap scripts via the UI to add or update Workers, these scripts automatically set the `-S` option according to the Leader TLS configuration.

Usage:

```
./cribl mode-managed-edge 127.0.0.1 -p 9420 -u myToken && ./cribl restart
```

Copy

The `-H <host> -p <port> -u <authToken>` parameters are required for successful connection to the Leader.

Arguments:

Option	Definition
`-d <deploymentId>`	Deployment ID for reporting telemetry on multiple deployments.
`-H <host>`	Leader's hostname or IP address. Required.
`-p <port>`	Leader's cluster communications port Example: `4200`. Required.
`-n <certName>`	Name of the saved certificate to use. Mutually exclusive with `-k` or `-c`.
`-k <privKeyPath>`	Path on server to the private key to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-c <certPath>`	Path on server to the certificate to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-u <authToken>`	Authentication token to include as part of the connection header. Required.
`-S <tls>`	Sets, or disables, secure communication between Leader and Worker/Edge Nodes via TLS. Accepts: ["true", "false"].

Option	Definition
`-e <envRegex>`	Regex to select environment variables to report to Leader. Example: `"/test/"`.
`-t <tags>`	Tag values to report to Leader.
`-g <group>`	Fleet to report to Leader.

In Cribl.Cloud, get the token for your Workspace from the Worker Node modal (**Add/Update Worker Node**) from the **Auth token** field.

In an on-prem deployment, get the token from **Settings** > **Global** > **System** > **Distributed Settings** > **Leader Settings** > **Auth token**.

10.3.16. Mode-Outpost

Configures instance as a Cribl Outpost.

For more information, see:

- Cribl Outpost
- Set up Outpost

Usage:

```
./cribl mode-outpost \
  -H leader.example.com \
  -p 4200 \
  -u myToken \
  -S 1
```

Copy

The `-H <host> -p <port> -u <authToken>` parameters are required for successful connection to the Leader.

Arguments:

Option	Definition
`-d <deploymentId>`	Deployment ID for reporting telemetry on multiple deployments.
`-H <host>`	Leader's hostname or IP address.
`-p <port>`	Leader's cluster communications port.
`-n <certName>`	Name of the saved certificate to use. Mutually exclusive with -k or -c.
`-k <privKeyPath>`	Path on server to the private key to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-c <certPath>`	Path on server to the certificate to use. Accepts PEM format. Can reference `$ENV_VARS`.
`-u <authToken>`	Authentication token to include as part of the connection header. Required.

Option	Definition
`-S <tls>`	Sets, or disables, secure communication between Leader and the Outpost via TLS. Accepts: [“true”, “false”].
`-e <envRegex>`	Regex to select environment variables to report to Leader. Example: `"/test/"`.
`-t <tags>`	Tag values to report to Leader.

In Cribl.Cloud, get the token for your Workspace from the Worker Node modal (**Add/Update Worker Node**) from the **Auth token** field.

In an on-prem deployment, get the token from **Settings** > **Global** > **System** > **Distributed Settings** > **Leader Settings** > **Auth token**.

10.3.17. Nc

Listens on a port for traffic, and outputs stats and data. (Netcat-like utility.)

Usage:

```
./cribl nc -p 4200
```

Copy

Sample response:

```
2021-08-20T22:44:30.457Z - starting server on 0.0.0.0:9999
2021-08-20T22:44:30.462Z - server listening 0.0.0.0:9999
2021-08-20T22:44:31.461Z - messages: 0, socks: 0, thruput: 0MBps
2021-08-20T22:44:32.466Z - messages: 0, socks: 0, thruput: 0MBps
...
2021-08-20T22:44:39.212Z - got connection: 127.0.0.1:37190
2021-08-20T22:44:39.213Z - got connection: 127.0.0.1:37192
```

Copy

Arguments:

Option	Definition
`-p <port>`	Port to listen on.
`-f <family>`	If this is `6` then nc will listen on `::1`. Otherwise, it listens on `0.0.0.0`.
`-s <statsInterval>`	Stats output interval (ms), use `0` to disable.
`-u`	Listen on UDP port instead.
`-o`	Output received data to stdout.
`-t <throttle>`	Throttle rate in (unit)/sec, where units can be KB, MB, GB, and TB.

10.3.18. Node

When you run `node` with no options, it displays a command prompt, as shown here:

```
>
```

Copy

To execute a JavaScript file, you can enter path/file name at the prompt.

With the `-v` option, prints the version of NodeJS that is running.

With `-e`, evaluates a string. Write to console to see the output. For example:

Usage:

```
./cribl node -e 'console.log(Date.now())'
./cribl node -v
```

Copy

Sample response:

```
./cribl node -e 'console.log(Date.now())'
1629740667695
```

Copy

```
v14.15.1
```

Copy

Arguments:

Option	Definition
`-e <eval>`	String to evaluate.
`-v`	Prints NodeJS version.

10.3.19. Pack

Manages Cribl Packs.

Sub-commands:

- `pack export`
- `pack install`
- `pack list`
- `pack uninstall`
- `pack upgrade`

Sub-commands and Options

`pack export`

Exports Cribl packs.

Usage:

```
./cribl pack export -m merge
```

Copy

Arguments:

Option	Definition
`-m <mode>`	Mode to export. Accepts: merge_safe, merge, default_only.
`-o <filename>`	Where to export the Pack on disk.
`-n <name>`	Name to override the installed Pack's name on export.
`-g <group>`	The Worker Group/Fleet to execute within.

`pack install`

Install a Cribl Pack.

Usage:

```
./cribl pack install -g default https://packs.cribl.io/dl/cribl-sysdig-events/1.0.3/c
```

Copy

In this command, `default` Worker Group ID.

Sample response:

```
{"time":"2025-11-13T08:14:05.599Z","cid":"api","channel":"ConfFeatureFlagsMgr","level
{"time":"2025-11-13T08:14:05.600Z","cid":"api","channel":"ConfFeatureFlagsMgr","level
{"time":"2025-11-13T08:14:05.661Z","cid":"api","channel":"collectors","level":"info",
{"time":"2025-11-13T08:14:05.671Z","cid":"api","channel":"LookupDiskUsageLimiterFactc
{"time":"2025-11-13T08:14:05.672Z","cid":"api","channel":"LookupDiskUsageLimiterFactc
{"time":"2025-11-13T08:14:05.687Z","cid":"api","channel":"collectors","level":"info",
{"time":"2025-11-13T08:14:05.691Z","cid":"api","channel":"PackMgr","level":"info","me
{"time":"2025-11-13T08:14:05.695Z","cid":"api","channel":"PackMgr","level":"info","me
{"time":"2025-11-13T08:14:05.695Z","cid":"api","channel":"PackMgr","level":"info","me
{"time":"2025-11-13T08:14:06.330Z","cid":"api","channel":"PackMgr","level":"info","me
https://packs.cribl.io/dl/cribl-sysdig-events/1.0.3/cribl-sysdig-events-1.0.3.crbl: s
{"time":"2025-11-13T08:14:06.993Z","cid":"api","channel":"PackMgr","level":"info","me
```

Copy

Arguments:

Option	Definition
`-d`	Run install in debug.
`-f`	Force install.
`-c`	Disallow installation of Packs with custom functions.
`-n <name>`	Name of the Pack to install, defaults to source.
`-g <group>`	The Worker Group/Fleet to execute within. Required for distributed deployments.

`pack list`

Lists Cribl Packs.

Usage:

```
./cribl pack list
```

Copy

Arguments:

Option	Definition
`-v`	Display all Pack information in verbose mode.
`-g <group>`	The Worker Group/Fleet to execute within.

`pack uninstall`

Uninstall a Cribl Pack.

Usage:

```
./cribl pack uninstall
```

Copy

Arguments:

Option	Definition
`-d`	Run uninstall in debug.
`-g <group>`	The Worker Group/Fleet to execute within.

`pack upgrade`

Upgrade a Cribl Pack.

Usage:

```
./cribl pack upgrade -s https://packs.cribl.io/packs/cribl-sysdig-events
```

Copy

Arguments:

Option	Definition
`-d`	Run upgrade in debug.
`-s <source>`	Provide the Pack source.
`-m <minor>`	Only upgrade to minor version.
`-g <group>`	The Worker Group/Fleet to execute within.

10.3.20. Parquet

For any Parquet file: view (`cat`) the file, its metadata (`details`), or its `schema`.

This command is available only on Linux.

Sub-commands:

- `parquet cat`
- `parquet details`
- `parquet schema`

Sub-commands and Options

`parquet cat`

Cat Parquet events to console.

Usage:

To view a Parquet file:

```
./cribl parquet cat -f /some_parquet_file
```

Copy

Sample response:

```
{"__bytes":274,"Unnamed: 0":0,"name":"apples","quantity":10,"price":2.6,"date":"2018-
...
```

Copy

Arguments:

Option	Definition
`-f <file>`	Path to Parquet file.

Option	Definition
`-m <max>`	Maximum events to print.

`parquet details`

View all Parquet details (schema and statistics) and metadata, including encoding and compression.

Usage:

```
./cribl parquet details -f /some_parquet_file
```

Copy

Sample response:


```
{
  "schema": {
    "Unnamed: 0": {
      "optional": true,
      "type": "INT64"
    },
    ...
  },
  "path": "../../../src/sluice/js/util/__tests__/data/parquet/single/small_fruits.par
  "num_rows": 40000,
  "num_cols": 11,
  "num_real_cols": 11,
  "num_row_groups": 1,
  "created_by": "parquet-cpp-arrow version 11.0.0",
  "schema_root_name": "schema",
  "metadata": {
    "pandas": "{\"index_columns\": [{\"kind\": \"range\", \"name\": null, \"start\":
    ...
  },
  "version": "PARQUET_2_6",
  "metadataSize": 6072,
  ...
    }
  ]
}
```

Copy

Arguments:

Option	Definition
`-f <file>`	Path to Parquet file.

`parquet schema`

View a Parquet file's schema (not including encoding or compression), expressed as JSON like in the Parquet Schema editor.

Usage:

```
./cribl parquet schema -f /some_parquet_file
```

Copy

Sample response:

```
{
  "Unnamed: 0": {
    "optional": true,
    "type": "INT64"
  },
  "name": {
    "optional": true,
    "type": "STRING"
  },
  "quantity": {
    "optional": true,
    "type": "DOUBLE"
  },
  "price": {
    "optional": true,
    "type": "DOUBLE"
...
  }
}
```

Copy

Arguments:

Option	Definition
`-f <file>`	Path to Parquet file.

10.3.21. Pipe

Feeds standard input (stdin) to a Pipeline.

Usage:

```
cat sample.log |  ./cribl pipe -p pipelineName
cat sample.log |  ./cribl pipe -p pipelineName 2>/dev/null
```

Copy

Sample response:

```
...
{"time":"2021-08-20T20:37:00.017Z","cid":"api","channel":"commands","level":"info","m
{"time":"2021-08-20T20:37:00.019Z","cid":"api","channel":"pipe:main","level":"info","
{"time":"2021-08-20T20:37:00.021Z","cid":"api","channel":"pipe:main","level":"info","
{"time":"2021-08-20T20:37:00.022Z","cid":"api","channel":"commands","level":"info","m
{"time":"2021-08-20T20:37:00.028Z","cid":"api","channel":"GrokMgr","level":"info","me
...
```

Copy

Arguments:

Option	Definition
`-p <pipeline>`	Pipeline to feed data through.
`-d`	Include dropped events.
`-c <cpuProfile>`	Perform CPU profiling.
`-t`	Perform pipeline tracing.
`-a <pack>`	Optional Cribl Pack context. Mutually exclusive with -b.
`-b <project>`	Optional Cribl Project context.
`-i`	Runs the pipe command without sandboxing javascript expressions from potential attackers.

10.3.22. Pq

Benchmark persistent queue (PQ) performance. Use `pq write` to simulate high-throughput event ingestion and `pq read` to simulate PQ consumption. This command is useful for internal benchmarking and evaluating disk performance in customer environments.

Sub-commands:

- `pq read`
- `pq write`

Sub-commands and Options

`pq-read`

Read data from a persistent queue and discard it. Use this command to test PQ deserialization and consumer performance.

Usage:

```
./cribl pq read -o /tmp/pq
```

Copy

Sample response:

This response prints every second during operation:

```
{"time":"2025-07-15T22:38:37.978Z","cid":"api","channel":"IPersistentQueue","level":'
{"time":"2025-07-15T22:38:37.982Z","cid":"api","channel":"IPersistentQueue","level":'
{"time":"2025-07-15T22:38:38.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:38.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:39.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:39.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:40.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:40.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:41.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:41.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:42.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:42.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:43.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:43.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:44.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:44.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:45.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:45.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:46.978Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:38:46.982Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:38:47.146Z","cid":"api","channel":"IPersistentQueue","level":'
{"time":"2025-07-15T22:38:47.148Z","cid":"api","channel":"commands:pq","level":"info'
```


Copy

Arguments:

Option	Definition
`-o <path>`	**(Required)** Directory path to read persistent queue files from.

This command does not commit or remove files from the persistent queue. It's intended for read-only performance testing.

`pq-write`

Read data from `stdin` and write to a persistent queue. Use this command to test PQ serialization and disk write performance.

Usage:

```
cat ~/work/syslog_large.log | ./cribl pq write -o /tmp/pq -b 1000 -s 10485760 -c gzip
```

Copy

Sample response:

This response prints every second during operation:

```
{"time":"2025-07-15T22:36:45.543Z","cid":"api","channel":"IPersistentQueue","level":'
{"time":"2025-07-15T22:36:45.545Z","cid":"api","channel":"IPersistentQueue","level":'
{"time":"2025-07-15T22:36:46.543Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:36:46.545Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:36:47.543Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:36:47.545Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:36:48.543Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:36:48.545Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:36:49.542Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:36:49.545Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:36:50.543Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:36:50.545Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:36:51.543Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:36:51.545Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:36:52.542Z","cid":"api","channel":"ProcessMetrics","level":"ir
{"time":"2025-07-15T22:36:52.545Z","cid":"api","channel":"commands:pq","level":"info'
{"time":"2025-07-15T22:36:52.563Z","cid":"api","channel":"commands:pq","level":"info'
```

Copy

Output includes the number of events and bytes processed every second, and a final summary on completion.

Arguments:

Option	Definition
-o	(Required) Directory that stores persistent queue files

10.3.23. Reload

Reloads Cribl Stream. Executes immediately.

Usage:

```
`./cribl reload`
```

Copy

Sample response:

```
Reload request submitted to Cribl
```

Copy

10.3.24. Restart

Restarts Cribl Stream. Executes immediately.

Executing this command cancels any running collection jobs.

Usage:

```
./cribl restart
```

Copy

Sample response:

```
Stopping Cribl, process 17
..............
Cribl is not running
Starting...
....
Cribl started
```

Copy

10.3.25. Scope

Greps your apps by the syscalls. Executes immediately.

See the AppScope CLI Reference for usage and examples.

10.3.26. Start

Starts Cribl Stream. Executes immediately. Upon first run, echoes the Cribl Stream default login credentials.

Usage:

```
./cribl start
```

Sample response:

```
Starting...
...
Cribl started
```

Copy

Arguments:

Option	Definition
`-d <dir>`	Configuration directory.
`-r <role>`	Process role.

10.3.27. Status

Displays status of Cribl Stream, including the API Server address, instance's mode (Leader or Worker), process ID, and GUID. Executes immediately.

Usage:

```
./cribl status
```

Copy

Sample response:

```
Address: http://172.17.0.3:9000
Mode: master
Status: Up
Software Version: 3.1.0-f765e418
Config Version: 347079c
Master: 0.0.0.0:4200
PID: 4100
GUID: e706052a-ace9-4511-a7c7-b58a414a07d3
```

Copy

10.3.28. Stop

Stops Cribl Stream. Executes immediately.

Executing this command cancels any running collection jobs.

Usage:

```
./cribl stop
```

Copy

Sample response:

```
Stopping Cribl, process 3951
............
Cribl is not running
```

Copy

10.3.29. Vars

Manages Cribl Stream global variables.

See Global Variables for more information.

Sub-commands:

- `vars add`
- `vars get`
- `vars remove`
- `vars update`

Sub-commands and Options

`vars add`

Add a global variable.

Usage:

```
./cribl vars add -i exampleVariable -v 42 -t number -d "Sample number variable" -c cr
```

Copy

Sample response:

```
[
  {
    "type": "number",
    "lib": "cribl",
    "description": "Sample number variable",
    "value": "42",
    "tags": "cribl,sample",
    "id": "exampleVariable"
  }
]
```

Copy

Arguments:

Option	Definition
`-i <id>`	Global variable ID.
`-t <type>`	Type.
`-v <value>`	Value.
`-a <args>`	Arguments.
`-d <description>`	Description.
`-c <tags>`	Custom Tags (comma separated list).
`-g <group>`	Group ID.

`vars get`

List global variables.

Usage:

```
./cribl vars get
```

Copy

Arguments:

Option	Definition
`-i <id>`	Global variable ID.
`-g <group>`	Group ID.

`vars remove`

Remove global variable.

Usage:

```
./cribl vars remove -i exampleVariable
```

Copy

Arguments:

Option	Definition
`-i <id>`	Global variable ID.
`-g <group>`	Group ID.

`vars update`

Update global variable.

Usage:

```
./cribl vars update -i exampleVariable -v 2001
```

Copy

Arguments:

Option	Definition
`-i <id>`	Global variable ID.
`-t <type>`	Type.
`v <value>`	Value.
`-a <args>`	Arguments.
`-d <description>`	Description.
`-c <tags>`	Custom Tags (comma separated list).
`-g <group>`	Group ID.

10.3.30. Version

Displays Cribl Stream version. Executes immediately.

Usage:

```
./cribl version
```

Copy

Sample response:

```
Software Version: 4.7.3-6f48361f
```

Copy

10.4. Environment Variables

You can use the following environment variables to configure your Cribl Stream installation.

Name	Purpose
`CRIBL_TMP_DIR`	Defines the root of a temporary directory. See Usage Notes below.
`CRIBL_VOLUME_DIR`	Sets a directory that persists modified data between different containers or ephemeral instances. When set, this environment variable overrides `CRIBL_HOME`. It also creates predefined subdirectories in the specified directory. If that directory already contains subdirectories with those names, they will be overwritten.
`CRIBL_BOOTSTRAP`	Quickstart a Cribl instance by configuring this variable.
`CRIBL_BOOTSTRAP_HOST`	Host name for connecting to the Leader Node when setting up a new Worker Node.
`CRIBL_API_HOST`	Overrides the **Host** setting.
`CRIBL_API_PORT`	Overrides the **Port** setting.
`CRIBL_EDGE`	(Cribl Edge only) When set to any value, runs this command at container start: `cribl mode-edge -H 0.0.0.0`. This launches the instance as an Edge Node, listening on a host at `0.0.0.0`.
`CRIBL_EDGE_FS_ROOT`	(Cribl Edge running in a container only) Location of the host OS filesystem when mounting in a container. Defaults to `/hostfs`.
`CRIBL_K8S_POD`	(Cribl Edge on Kubernetes only) Sets the name of the Kubernetes Pod in which Cribl Edge is deployed.
`CRIBL_K8S_TLS_REJECT_UNAUTHORIZED`	(Cribl Edge on Kubernetes only) Set to `0` to disable certificates validation when connecting to the Kubernetes APIs. When you disable this environment variable, all Kubernetes features (including Metadata, Metrics, Logs, and AppScope metadata) will tolerate invalid TLS certificates (such as, expired, self-signed, and so forth) when connecting to the Kubernetes APIs.
`CRIBL_SERVICEACCOUNT_PATH`	(Cribl Edge on Kubernetes only) Path to the `ServiceAccount` to use to query the Kubernetes API.

Name	Purpose
	Defaults to `/var/run/secrets/kubernetes.io/serviceaccount`.
`CRIBL_K8S_CPU_LIMIT`	(Cribl Stream on Kubernetes only) Automatically sets CPU shares for Cribl Stream based on the `resources.limits.cpu` value, configured by default in the Helm chart.
`CRIBL_MAX_WORKERS`	(Cribl Stream on Kubernetes only) Setting a maximum amount of Worker Processes on a Kubernetes node helps to prevent overloading the node with more CPUs than it can handle.
`CRIBL_NOEXEC`	When set to `1`, prevents the Exec Source from processing any data, effectively putting it into a "disabled" state. To re-enable, remove or unset the `CRIBL_NOEXEC` variable.

Distributed Deployment Variables

You can use the following environment variables to configure your distributed Cribl Stream instance.

Name	Purpose
`CRIBL_DEPLOYMENT`	Use this variable to set a human-readable name for your on-prem Leader. Example: `linux_production`. The value you set using this variable will appear on the Connected Environments page.
`CRIBL_CLOUD_WORKSPACE_URL`	URL of the Leader Node in a Cribl.Cloud deployment. Example: `CRIBL_DIST_LEADER_URL=tls://<authToken>@leader:4200`. See Usage Notes.
`CRIBL_DIST_LEADER_URL`	URL of the Leader Node. Example: `CRIBL_DIST_LEADER_URL=tls://<authToken>@leader:4200`. See Usage Notes.
`CRIBL_DIST_MODE`	`worker` or `leader`. Defaults to `worker`, if (and only if) `CRIBL_DIST_LEADER_URL` is present.
`CRIBL_DIST_WORKER_PROXY`	Communicate to the Leader Node via a SOCKS proxy. See Usage Notes below.
`CRIBL_DIST_LEADER_BUNDLE_URL`	Specifies the location of configuration bundles. When set, Cribl Stream {@node}s will attempt to retrieve their configuration bundles from the URL specified by this variable, rather than directly from the Leader Node.

Usage Notes

This section explains how to use certain environment variables.

`CRIBL_DIST_LEADER_URL`

Use this format:

```
<tls|tcp>://<authToken>@<host>:<port>?group=defaultGroup&tag=tag1&tag=tag2&tls.
<tls_settings>
```

> To generate a random authentication token, leave `<authToken>` unchanged. You can define it to add your own token instead, but make sure it's secure enough.
>
> The auth token can include only alphanumeric characters or the underscore (`_`). Do not include `<`, `>`, `"`, `` ` ``, `\r`, `\n`, `\t`, `{`, `}`, `|`, `\`, `^`, or `'`.

When configuring the node as a Leader, use `0.0.0.0` (or another suitable binding address) for `host`. Then, set `CRIBL_DIST_MODE` to `leader`.

Here are the components:

- `group` - The preferred Worker Group assignment.
- `resiliency` - The preferred Leader failover mode.
- `volume` - The location of the NFS directory to support Leader failover.
- `tag` - A list of tags that you can use to map (Stream, Edge) the Worker to a Worker Group.
- `disableSNIRouting` - Whether Server Name Indicator (SNI) routing is enabled on the Worker Node. Boolean; defaults to `false`.

  > Do not edit this advanced setting without supervision by Cribl Support. Changing this setting can affect the scalability of your system because it determines how Cribl Stream routes connections between Leader Nodes and Worker Nodes. See the Known Issue that resulted in this setting for more details.

- `tls.privKeyPath` - Private Key Path.
- `tls.passphrase` - Key Passphrase.
- `tls.caPath` - CA Certificate Path.
- `tls.certPath` - Certificate Path.
- `tls.rejectUnauthorized` - Validate Client Certs. Boolean; defaults to `false`.

- `tls.requestCert` - Authenticate Client (mutual auth). Boolean; defaults to `false`.
- `tls.commonNameRegex` - Regex matching peer certificate > subject > common names allowed to connect. Used only if `tls.requestCert` is set to `true`.
- `compression` - Codec used for compressing the data before sending. One of: `gzip` or `none`.

`CRIBL_DIST_LEADER_URL` was previously called `CRIBL_DIST_MASTER_URL`. This name is still in use in certain contexts, for example in Helm charts.

CRIBL_TMP_DIR

Sources use this variable to construct temporary directories in which to stage downloaded Parquet data. If `CRIBL_TMP_DIR` is not set (the default), Cribl applications create subdirectories within your operating system's default temporary directory:

- For Cribl Stream: `<OS_default_temporary_directory>/stream/`.
- For Cribl Edge: `<OS_default_temporary_directory>/edge/`.

For example, on Linux, Stream's default staging directory would be `/tmp/stream/`.

If you explicitly set this `CRIBL_TMP_DIR` environment variable, its value replaces this OS-specific default parent directory.

CRIBL_DIST_WORKER_PROXY

`CRIBL_DIST_WORKER_PROXY` lets you configure a SOCKS proxy for communication between the Leader Node and Worker Nodes.

The default protocol is SOCKS5, which uses the format: `socks5://<username>:<password>@<host>:<port>`. Only `<host>:<port>` are required.

You can alternatively specify SOCKS4 as the protocol with the format: `socks4://<proxyhost>:<port>`. To authenticate on a SOCKS4 proxy with username and password, use this format: `socks4://<username>:<password>@<proxyhost>:<port>`. The `<proxyhost>` can be a hostname, IPv4, or IPv6.

To configure a SOCKS proxy via the UI, see Configure SOCKS Proxy. To configure an HTTP(S) proxy to use for your Sources and Destinations, refer to System Proxy Configuration.

CRIBL_BOOTSTRAP_HOST

`CRIBL_BOOTSTRAP_HOST` overrides the Add/Bootstrap New Worker script generator's default hostname.

For example, if you set `CRIBL_BOOTSTRAP=myhost`, then `myhost` will appear in the script modal's **Leader or Outpost hostname/IP** field, instead of the URL used in the browser.

`CRIBL_BOOTSTRAP`

`CRIBL_BOOTSTRAP` enables specifying a URL, an absolute disk file path, or a YAML string, in order to bootstrap a configuration to the `$CRIBL_HOME/local` directory. Cribl Stream applies this configuration only upon its first startup.

For any method, Cribl Stream expects each targeted config file to be YAML-formatted. Each file's top-level keys should be the paths to config files inside the `$CRIBL_HOME/local/...` subdirectory.

Below is an example of a bootstrap file. Its output, when Cribl Stream starts, would be to create three files inside the `$CRIBL_HOME/local/cribl` path: `inputs.yml`, `outputs.yml`, and `pipelines/route.yml`.

```
cribl/inputs.yml:
  inputs:
    <id>:
      <config>
cribl/outputs.yml:
  outputs:
    <id>:
      <config>
cribl/pipelines/route.yml:
  id: default
  groups: {}
  comments: []
  routes:
    ...
```

Copy

For details about each file's syntax, see Config Files and its child topics.

`CRIBL_GIT_REMOTE` and `CRIBL_GIT_REMOTE_URL`

These two GitOps bootstrap variables are closely related but behave very differently. Use them with the following in mind.

`CRIBL_GIT_REMOTE` (Bootstrap/Force Sync)

When `CRIBL_GIT_REMOTE` is set on a Leader at startup, it triggers a one-time forced synchronization from the Git repo, rather than simply setting the remote URL.

On Leader startup with `CRIBL_GIT_REMOTE` set:

- Treats the remote Git repo as the source of truth for configuration.
- Replaces any uncommitted local configuration in the Leader's Git working copy with the contents of the configured remote and branch. Effectively a `git reset --hard`.
- Restarts the Leader after the sync is complete.
- Removes `CRIBL_GIT_REMOTE` from the running process so that this forced sync does not recur on subsequent restarts.

This bootstrap process runs based solely on the presence of `CRIBL_GIT_REMOTE`, even if `CRIBL_GIT_OPS` is set to `none`. If the remote repo is unreachable or misconfigured, the Leader can fail to start until the environment variable is removed or corrected.

Recommended use: For one-time bootstrap or recovery scenarios, where you explicitly want to discard uncommitted local configuration in favor of a remote branch, and you have a backup of `$CRIBL_HOME` before enabling this variable.

`CRIBL_GIT_REMOTE_URL` (Remote URL Configuration)

`CRIBL_GIT_REMOTE_URL` exists to configure the Git remote in a way that matches the Git Settings UI without triggering a forced synchronization or restart.

- It sets the Git remote URL used by the Leader.
- It does not wipe uncommitted local configuration.
- When both variables are set, `CRIBL_GIT_REMOTE_URL` takes precedence for the URL value, while `CRIBL_GIT_REMOTE` acts as the switch to enable the forced bootstrap behavior.

Recommended use: For standard Git integration where you want to maintain local changes until you explicitly commit and push them. This is best for keeping the UI's remote URL in sync via environment variables without risking data loss.

Adding Fallback Leaders

You can configure fallback Leader Nodes to support high availability and failover. Use the following environment variables.

Name	Purpose
`CRIBL_DIST_MASTER_RESILIENCY=failover`	Sets the Leader's `Resiliency` to `Failover` mo
`CRIBL_DIST_MASTER_FAILOVER_VOLUME=/tmp/shared`	Sets the location of the NFS directory to support failover. For example: `/mnt/cribl`.

Name	Purpose
`CRIBL_DIST_MASTER_FAILOVER_MISSED_HB_LIMIT`	Determines how many Lease refresh periods ca before the standby Leader Nodes attempt to pr themselves to primary. Cribl recommends settin
`CRIBL_DIST_MASTER_FAILOVER_PERIOD`	Determines how often the primary Leader refre on the Lease file. Cribl recommends setting this
`CRIBL_INSTANCE_HOME`	In `Failover` mode, this variable points to the L `root` directory, as opposed to the shared volu to access `$CRIBL_INSTANCE_HOME/local/_system/in` (`C:\ProgramData\Cribl\local\_system.y` Edge on Windows).

GitOps Variables

Cribl Stream provides the following environment variables to facilitate using GitOps in Cribl Stream.

You can use these variables to override the defaults in the UI if you are not using systemd to manage the `cribl` service.

Bootstrap Variables

Name	Purpose
`CRIBL_GIT_REMOTE`	Location of the remote repo to track. Can contain username and password for HTTPS auth. See Usage Notes.
`CRIBL_GIT_REMOTE_URL`	Configure simple Git remote URL. Takes precedence over `CRIBL_GIT_REMOTE` for URL setting. See Usage Notes.
`GIT_SSH` / `GIT_SSH_COMMAND`	See Git's documentation.
`CRIBL_GIT_BRANCH`	Git ref (branch, tag, commit) to track/check out.
`CRIBL_GIT_AUTH`	One of: `none`, `basic`, or `ssh`.
`CRIBL_GIT_USER`	Used for `basic` auth.
`CRIBL_GIT_PASSWORD`	Used for `basic` auth.
`CRIBL_GIT_OPS`	Controls which GitOps workflow to use. One of: `push` to enable the GitOps push workflow, or `none` to disable GitOps.

Name	Purpose
`CRIBL_INTERACTIVE`	Controls whether git commands called by Cribl CLI at startup are interactive.
`CRIBL_GIT_SSH_KEY`	Content of the SSH key used to access git remote.
`CRIBL_GIT_STRICT_HOST_KEY_CHECKING`	Controls whether to check the host key strictly.

Additional Environment Variables

Cribl Stream uses the following variables for multiple purposes.

Name	Purpose
`CRIBL_SPOOL_DIR`	Specifies the base path where events from various Sources and Destinations are spooled. Defaults to `$CRIBL_HOME/state/spool` or `$CRIBL_VOLUME_DIR/state/spool`.

Cribl.Cloud Certificate Variables

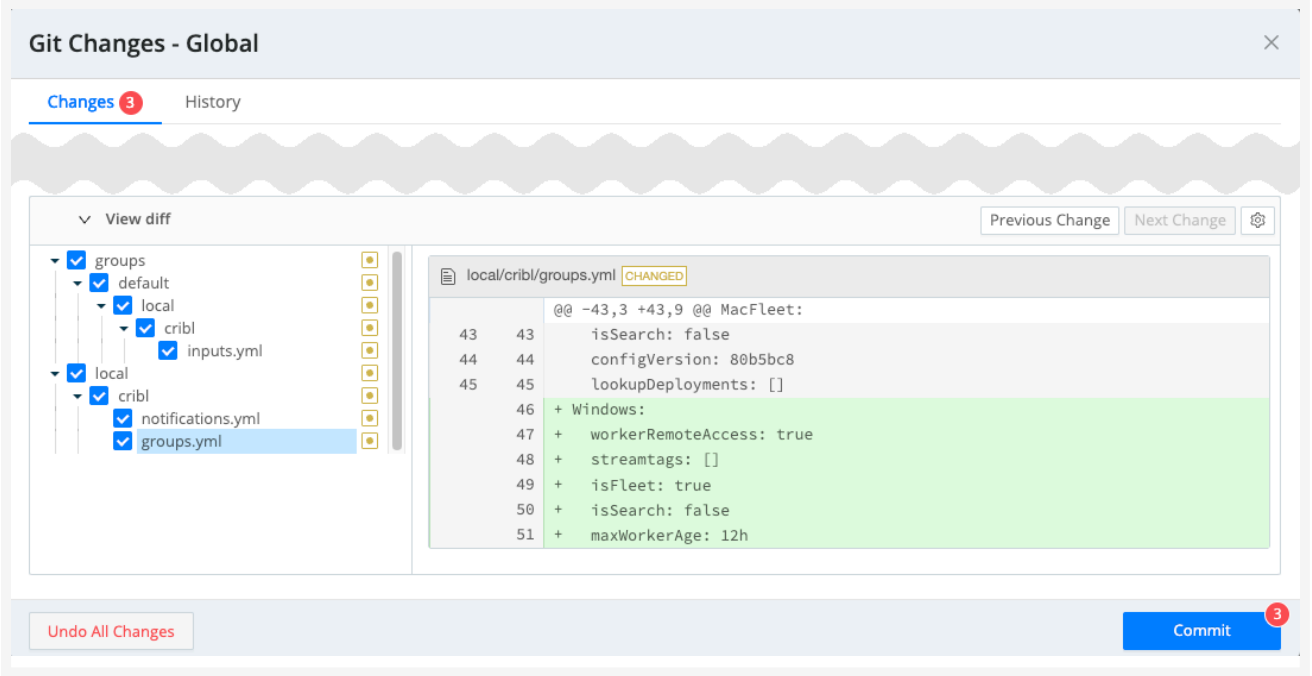
The following variables point to the location of default certificate files on Cribl.Cloud:

Name	Purpose
`CRIBL_CLOUD_KEY`	Path to the default private key on Cribl.Cloud.

10.5. Configuration Files

Configuration files store the configuration changes locally on your system. They reflect the changes you make to configuration settings in the UI.

You can preview how settings are translated into configuration changes in the Git Changes modal whenever you commit your changes via the Version Control option.



Git Changes screen showing Git diff with changes to configuration files.

Typically, you do not modify configuration files manually. If you need to edit the configuration outside the UI, Cribl recommends using the API. However, if your particular use case requires it, in an on-prem deployment the files are open for editing.

Configuration File Locations

In an on-prem deployment, configuration files are stored in the `$CRIBL_HOME/local/cribl` directory (where `$CRIBL_HOME` is your installation directory, for example, `/opt/cribl/`). Configuration for specific Worker Groups or Fleets is located `$CRIBL_HOME/groups/<group-or-fleet-name>/local/cribl`.

You can view the default configurations by taking a look in the `default` directory, instead of `local`.

Pack configuration resides within a separate subdirectory for each Pack: `$CRIBL_HOME/default/<pack_name>`.

Configuration file/folder	Description
`ai.yml`	Cribl Copilot settings.
`app-limits.yml`	Internal limit configuration.
`appscope.yml`	Appscope configuration (deprecated).
`breakers-search.yml`	Cribl Search Event Breaker settings.
`breakers.yml`	Event Breakers configuration.
`certificates.yml`	TLS/SSL certificates.
`collectors`	Directory storing individual Collector configurations.
`conditions`	Directory storing configurations for Notification conditions.
`cribl.yml`	General system configuration.
`dataset-providers.yml`	Cribl Search Dataset Providers.
`datasets.yml`	Cribl Search Datasets.
`email-templates.yml`	Templates for Email Notifications.
`executors`	Executor job configurations.
`fleet-mappings.yml`	Mapping rulesets for Edge Nodes.
`functions`	Directory storing individual Function configurations.
`grok-patterns`	Directory storing individual Grok Pattern configurations.
`groups.yml`	Worker Group and Edge Fleet settings.
`inputs.yml`	Sources configuration.
`instance.yml`	Configuration for the current instance.
`iometrics.yml`	Metrics levels for individual Sources and Destinations in Cribl Stream.
`ipfix-information-elements.yml`	Predefined IPFIX fields for handling and decoding IPFIX fields from the Cribl Stream NetFlow & IPFIX Source.
`jobs.yml`	Collector configuration.
`job-limits.yml`	Parameters for Collection jobs and system tasks.

Configuration file/folder	Description
`kms.yml`	KMS provider configuration.
`lake-config.yml`	Configuration for Cribl Lake.
`lakes.yml`	Configuration for individual Cribl Lake datasets.
`leader.yml`	Secondary Leader configuration.
`licenses.yml`	Licenses.
`limits.yml`	Limit configuration.
`logger.yml`	Logging levels and redactions.
`mappings.yml`	Mapping rulesets for Stream Workers.
`messages.yml`	Messages displayed in the UI's `Messages` fly-out.
`mdt-devices.yml`	Configuration for MDT (Model Driven Telemetry) devices.
`notifications.yml`	Notifications and Notification targets.
`notification-templates.yml`	Templates for email Notifications.
`outpost.yml`	Configuration for Cribl Outpost.
`outputs.yml`	Destinations configuration.
`parquet-schemas.yml`	Parquet Schemas.
`parquet-schemas`	Directory storing individual Parquet Schema configurations.
`parsers.yml`	Parsers Library.
`perms.yml`	Permission configuration.
`persistent-queue.yml`	Persistent queue configuration.
`pipelines`	Directory storing individual Pipeline configurations.
`policies.yml`	RBAC Policies.
`protobuf-libraries.yml`	Configuration for Protobuf libraries.
`redis-cache-limits.yml`	Redis cache settings.

Configuration file/folder	Description
`redis-limits.yml`	Redis connection settings.
`regexes.yml`	Regex Library.
`roles.yml`	RBAC Roles.
`route.yml`	Route configuration. Located in `local/cribl/pipelines/routes.yml`.
`samples.yml`	Metadata about about stored sample data files.
`saved-queries.yml`	Saved searches in Cribl Search.
`schemas.yml`	Schema Library.
`schemas`	Directory storing individual schema configurations.
`scripts.yml`	Scripts configured at **Settings** > **Global** > **Scripts**.
`sds-rules.yml`	Cribl Guard rules.
`sds-rulesets.yml`	Cribl Guard rulesets.
`search-limits.yml`	Limit configuration for Cribl Search.
`search-usage-groups`	Cribl Search Usage Group configuration.
`secrets.yml`	Secrets.
`scope_protocol.yml`	Appscope (deprecated) configuration for protocol detection.
`scope.yml`	Appscope (deprecated) configuration.
`search-limits.yml`	Cribl Search limits.
`search-usage-groups.yml`	Cribl Search Usage Groups.
`service.yml`	Service processes.
`vars.yml`	Variables Library.

Configurations and Restart

You can **Restart** and **Reload** via the UI. In the sidebar, select **Settings**, then **Global**. Under **System** > **Controls** select **Reload** or **Restart**.

In a distributed environment, Worker Nodes poll the Leader for configuration changes. Many of these changes require a quick reload to read the new configuration, while others require a restart of the Cribl processes on the Worker Node.

Upon restarts, be aware of the following:

- Syslog data still being received over UDP might be dropped.
- Worker Nodes will temporarily disappear from the Leader's **Workers** or **Edge Nodes** page.
- Aggregation and suppression operations will start over.
- Worker Nodes' local copies of Monitoring metrics will be dropped.
- Cribl Stream will drop any events still in RAM that were bound for persistent queues. (However, PQ data already written to disk will persist through the restart.)

Changes that require reloads include configuration changes to:

- Functions
- Pipelines
- Packs
- Routes
- Lookups
- Parquet schemas
- Global variables
- Group Settings > Limits
- Group Settings > Logging > Levels

Changes that require restarts include configuration changes to:

- Distributed mode (Leader versus Managed Worker Node or Single instance)
- Worker Group assignment
- Event Breakers
- QuickConnect configs
- Sources
- Destinations
- Group Settings > General Settings > TLS

- Group Settings > General Settings > Advanced
- Group Settings > Worker Processes > Process count and Memory

Some general guidelines to keep in mind:

- Configuration changes generated by most UI interactions - for instance, changing the order of Functions in a Pipeline, or changing the order of Routes - **do not require restarts**.
- Some configuration changes in the **Settings** UI **do require restarts**. These will prompt you for confirmation before restarting.
- All direct edits of configuration files in `$CRIBL_HOME/local/` **will require restarts**.
- Worker Nodes might temporarily disappear from the Leader's **Workers** or **Edge Nodes** tab while restarting.
- A `git commit` command on the Leader Node's host (using a freestanding `git` client not embedded in

10.5.1. Breakers.Yml

Event Breaker configuration is stored in `breakers.yml`.

breakers.yml

```
# ID - ID can contain only letters (A-Z, a-z), numbers (0-9), dashes, underscores, ar
# [string; required]
id:
# Library - Library classification for the ruleset
# One of: custom | cribl-custom
# [string; default: custom]
lib:
# Description - Description of the event breaker ruleset
# [string]
description:
# Tags - Tags associated with the ruleset
# [string]
tags:
# Min raw length - The minimum number of characters in _raw to determine which rule t
# [number; min: 50; max: 100000; default: 256]
minRawLength:
# Rules - A list of rules that will be applied, in order, to the input data stream
rules:
  # Name - Name of the rule
  # [string; required]
  name:
  # Filter condition - JavaScript expression applied to the beginning of a file or ob
  # whether the rule applies to all contained events.
  # [string; required]
  condition:
  # Event Breaker type - Type of event breaker to use
  # One of: regex | json | json_array | header | timestamp | csv | aws_cloudtrail | a
  # [string; required]
  type:
  # Index - Rule index for ordering
  # [number]
  index:
  # Timestamp anchor - The regex to match before attempting timestamp extraction. Use
  # anchor) to prevent extraction.
  # [string; required]
  timestampAnchorRegex:
  # Default timezone - Default timezone for timestamp parsing
  # [string; required]
  timestampTimezone:
  # Earliest timestamp - Earliest allowable timestamp
  # [string]
  timestampEarliest:
```

```
# Latest timestamp - Latest allowable timestamp
# [string]
timestampLatest:
# Timestamp configuration - Timestamp extraction configuration
# [required]
timestamp:
  # Timestamp type - Type of timestamp extraction
  # One of: auto | format | current
  # [string; required]
  type:
  # Timestamp length - Length of timestamp to extract
  # [number]
  length:
  # Timestamp format - Format string for timestamp parsing
  # [string]
  format:
# Fields - Field definitions for extraction
fields:
# Event byte limit - The maximum number of bytes that an event can be before being
# [number; required]
maxEventBytes:
# Disabled - Whether this rule is disabled
# [boolean]
disabled:
# Parser enabled - Whether to enable parser for this rule
# [boolean]
parserEnabled:
# Parser configuration - Parser settings for this rule
parser:
# Should use data raw - Use the field in the data called _raw for post processors
# [boolean]
shouldUseDataRaw:
# Event Breaker - The regex used to break the stream into events at the beginning o
# [string]
eventBreakerRegex:
# JSON Array Field - Name of the Array field to parse events from (json_array type
# [string]
jsonArrayField:
# JSON Extract All - Whether json_array breaker should extract fields (json_array t
# [boolean]
jsonExtractAll:
# JSON Time Field - Name of the field to use as _time (json_array type only)
# [string]
```

```
jsonTimeField:
```

10.5.2. Certificates.Yml

`certificates.yml` maintains a list of configured certificates and their parameters.

certificates.yml

```
# Name - Unique identifier for this certificate
# [string; required]
id:
# Description - Optional description for the certificate
# [string]
description:
# Certificate - Drag/drop or upload host certificate in PEM/Base64 format, or paste i
# here
# [string; required]
cert:
# Private key - Certificate private key in PEM format
# [string; required]
privKey:
# Passphrase - Optional passphrase for the private key
# [string]
passphrase:
# CA certificate - Optionally, drag/drop or upload all CA certificates in PEM/Base64
# certificate contents here. Certificates can be used for client and/or server auther
# [string]
ca:
# Referenced - List of configurations that reference this certificate
inUse:
```

Copy

10.5.3. Cribl.Yml

`cribl.yml` contains settings for configuring API and other system properties.

cribl.yml

```
# API Server Settings
api:
  # Host - Hostname or address to bind API server to. Defaults to 0.0.0.0. Using $CRI
  # overrides this setting.
  # [string; default: 0.0.0.0; required]
  host:
  # Port - API port to listen to. Defaults to 9000. Using $CRIBL_API_PORT overrides t
  # [number; max: 65535; default: 9000; required]
  port:
  # Protocol - Protocol that API server speaks, defaults to http1.1
  # [string; default: http1.1]
  protocol:
  # Retry count - Number of times to retry binding to API port
  # [number; min: 0; default: 120]
  retryCount:
  # Retry period - Period, in seconds, between consecutive port binding retries
  # [number; min: 0; default: 5]
  retrySleepSecs:
  # URL base path - URL base path from which to serve all assets (useful when behind
  # [string; default: '']
  baseUrl:
  # Local UI access - Enable to allow direct browser access to the Cribl Stream Worke
  # [boolean; default: false]
  disabled:
  # Listen on port - Expose the API service to the network on the configured port.
  # [boolean; default: true]
  listenOnPort:
  # Enable teleporting to Worker Nodes - Enables authenticated access to each Worker
  # Leader
  # [boolean; default: false]
  workerRemoteAccess:
  # Log out on roles change - Log users out when their roles change
  # [boolean; default: true]
  revokeOnRoleChange:
  # Auth token TTL - How long (in seconds) authentication tokens remain valid. Defaul
  # minimum is 1 sec.
  # [number; min: 1; default: 3600]
  authTokenTTL:
  # Session idle time limit - How long (in seconds) Cribl Stream will observe no user
  # before invalidating user's session tokens. Default is 1 hr.; minimum is 60 sec.
  # [number; min: 60; default: 3600]
  idleSessionTTL:
```

```
  # Login rate limit - Rate limit, expressed as maximum number of requests per interv
  # minute, hour, day). Examples: 3/second, 5/minute
  # [string]
  loginRateLimit:
  # SSO/SLO callback rate limit - Rate limit for SSO and SLO callback endpoints. Expr
  # maximum number of requests per second, minute, hour, or day. Examples: 3/second,
  # When limit is reached, the Cribl Stream sends 429 Too Many Requests HTTP response
  # [string]
  ssoRateLimit:
  # HTTP headers - Custom HTTP headers to be sent with every response
  headers:
  apiCache:
    # Enable API cache - Enable caching of browser's frequent API requests. (Disablir
    # the response time.)
    # [boolean; default: false]
    disabled:
  # SSL - Enable SSL on API. Defaults to disabled.
  ssl:
    # Disabled
    # [boolean; default: true; required]
    disabled:
    # Certificate - The name of the predefined certificate
    # [string]
    certificateName:
    # Private key path - Path on server in which to find the private key to use. PEM
    # reference $ENV_VARS.
    # [string; required]
    privKeyPath:
    # Passphrase - Passphrase to use to decrypt private key
    # [string]
    passphrase:
    # Certificate path - Path on server in which to find certificates to use. PEM for
    # reference $ENV_VARS.
    # [string; required]
    certPath:
    # CA certificate path - Path on server where to find CA certificates to use. PEM
    # reference $ENV_VARS.
    # [string]
    caPath:
# Support
support:
  # Feature Flag Overrides
  featureFlagOverrides:
```

```
    # Flag ID
    # [string; min: 1; max: 100; required]
    flagId:
    # Enabled
    # [boolean; default: true; required]
    disabled:
# Authentication Settings
auth:
  # Type - Select from this list of supported authentication providers
  # One of: local | splunk | ldap | openid | saas | saml
  # [string; required]
  type:
  # LDAP servers - List of LDAP servers. Each entry should contain host:port (example
  # localhost:389).
  # [array; min: 1; required]
  ldapServers:
  # Bind DN - Distinguished Name of entity to authenticate with LDAP server. Example:
  # 'cn=admin,dc=example,dc=org'
  # [string; required]
  bindDN:
  # Password - Distinguished Name password used to authenticate with LDAP server
  # [string; required]
  bindCredentials:
  # Username field - LDAP user search field, such as cn or uid
  # [string; default: cn; required]
  usernameField:
  # User search base - Starting point to search LDAP for users. Example: 'dc=example,
  # [string; required]
  searchBase:
  # Secure - Enable to use a secure LDAP connection (ldaps://); disable for unsecure
  # connection
  # [boolean; default: false]
  secure:
  # User search filter - LDAP search filter to apply when finding user. Example:
  # (&(group=admin)(!(department=123*)))
  # [string]
  searchFilter:
  # Group search base - Starting point to search LDAP for groups. Example: 'dc=exampl
  # [string]
  groupSearchBase:
  # Group member field - LDAP group search field, such as member
  # [string; default: member]
  groupMemberField:
```

```
# Group membership attribute - Attribute name of LDAP user object, such as dn. Dete
# member attribute's value, which defines group's allowed users.
# [string; default: dn]
groupMemberValField:
# Group search filter - LDAP search filter to apply when finding group. Example:
# (&(cn=cribl*)(objectclass=group))
# [string]
groupSearchFilter:
# Group name field - LDAP group field, such as cn
# [string; default: cn]
groupField:
# Connection timeout (ms)
# [number; default: 5000]
connectTimeout:
# Validate server certs - Validate server certificates globally. Disable to allow s
# certificates.
# [boolean; default: true]
rejectUnauthorized:
# Host - Hostname or IP address of Splunk instance
# [string]
host:
# Port - Management port of Splunk instance
# [number; default: 8089]
port:
# Issuer - The OIDC provider's issuer URL, e.g., https://accounts.google.com
# [string]
issuer:
# Authorization URL - The OIDC provider's authorization URL
# [string]
authorizationURL:
# Token URL - The OIDC provider's token URL
# [string]
tokenURL:
# UserInfo URL - The OIDC provider's UserInfo URL
# [string]
userInfoURL:
# Client ID - The client ID provided by the OIDC provider
# [string]
clientID:
# Client Secret - The client secret provided by the OIDC provider
# [string]
clientSecret:
# Callback URL - The callback URL where the OIDC provider redirects after authentic
```

```
  # [string]
  callbackURL:
  # Scope - The scopes to request from the OIDC provider
  # [string]
  scope:
system:
  # One of: api | auto | false
  # [string]
  upgrade:
  # One of: api | false
  # [string]
  restart:
  # One of: splunk-app | standalone
  # [string]
  installType:
  # Share telemetry with Cribl - Help improve Cribl's products and capabilities by sh
  # anonymized telemetry data with Cribl
  # [boolean]
  intercom:
  rollback:
    # Enable automatic rollback - Automatically roll back the upgrade if it fails to
    # the Worker Node fails to connect to the Leader
    # [boolean; default: true; required]
    rollbackEnabled:
    # Rollback timeout (ms) - Time to wait before checking node health to perform an
    # rollback
    # [number; min: 1; default: 30000]
    rollbackTimeout:
    # Rollback condition retries - Number of times to retry health check before perfo
    # rollback
    # [number; min: 0; default: 5]
    rollbackRetries:
    # Check interval (ms) - Time to wait between retries
    # [number; min: 100; default: 1000]
    checkInterval:
  backups:
    # Backups directory - Directory where backups will be stored
    # [string; default: $CRIBL_STATE_DIR/backups]
    backupsDirectory:
    # Backup persistence - Relative time expression for how long to keep the backups.
    # [string; default: 24h]
    backupPersistence:
workers:
```

```
  # number of worker processes to spawn, if less than 1 the value is added to CPU cou
  # [number; default: 1]
  count:
  # [number; min: 1024; default: 2048]
  memory:
# Default TLS Settings
tls:
  # Minimum TLS version - Minimum TLS version. Defaults to TLS 1.2.
  # One of: TLSv1 | TLSv1.1 | TLSv1.2 | TLSv1.3
  # [string]
  minVersion:
  # Maximum TLS version - Maximum TLS version. Defaults to TLS 1.3.
  # One of: TLSv1 | TLSv1.1 | TLSv1.2 | TLSv1.3
  # [string]
  maxVersion:
  # Default cipher list - Default suite of enabled and disabled TLS ciphers. Defaults
  # ECDHE-RSA-AES128-GCM-SHA256:
  # ECDHE-ECDSA-AES128-GCM-SHA256:
  # ECDHE-RSA-AES256-GCM-SHA384:
  # ECDHE-ECDSA-AES256-GCM-SHA384:
  # DHE-RSA-AES128-GCM-SHA256:
  # ECDHE-RSA-AES128-SHA256:
  # DHE-RSA-AES128-SHA256:
  # ECDHE-RSA-AES256-SHA384:
  # DHE-RSA-AES256-SHA384:
  # ECDHE-RSA-AES256-SHA256:
  # DHE-RSA-AES256-SHA256:
  # HIGH:
  # !aNULL:
  # !eNULL:
  # !EXPORT:
  # !DES:
  # !RC4:
  # !MD5:
  # !PSK:
  # !SRP:
  # !CAMELLIA
  # [string]
  defaultCipherList:
  # ECDH curve - The curve name, or a colon-separated list of curve NIDs or names, to
  # key agreement. For example: 'P-521:P-384:P-256'. Defaults to 'auto'.
  # [string; default: auto]
  defaultEcdhCurve:
```

```
  # Validate server certs - Validate server certificates globally. Disable to allow s
  # certificates.
  # [boolean; default: true]
  rejectUnauthorized:
# Proxy Settings
proxy:
  # Use proxy env vars - Honor HTTP_PROXY/HTTPS_PROXY environment variables
  # [boolean; default: true; required]
  useEnvVars:
git:
  # Branch - The branch to track in your Stream deployment's git repository
  # [string; default: master]
  branch:
  # GitOps workflow - The GitOps workflow for managing Cribl configuration
  # One of: none | push
  # [string; default: none]
  gitOps:
  # Collapse actions - Collapse Commit & Deploy into a single action. If you've confi
  # remote, Commit & Git Push will also be collapsed. Your default commit message bel
  # used for all commits.
  # [boolean]
  commitDeploySingleAction:
  # Default commit message - Enter a default message to use for all commits
  # [string; required]
  defaultCommitMessage:
  # Remote URL
  # [string]
  remote:
  # Git authentication type
  # One of: none | ssh | basic
  # [string; default: ssh]
  authType:
  # Scheduled global actions - Global git actions to run automatically on a schedule
  # One of: none | commit | push | commitPush
  # [string; default: none]
  autoAction:
  # Git timeout - Maximum time (in milliseconds) to wait for git processes before enc
  # Enter 0 to wait indefinitely.
  # [number; min: 0; default: 600000]
  timeout:
  # Generate commit messages automatically with Cribl Copilot - Cribl Copilot-generat
  # override any default commit message you may have set
  # [boolean]
```

```
  copilotAutoGitCommitMessages:
  # SSH private key - SSH private key (without passphrase) to use for authentication
  # repo
  # [string]
  sshKey:
  # SSH strict host key checking - Validate key against known hosts, to prevent spoof
  # impersonation attacks. For details, see "Verifying Host Keys" [on the Linux man
  # page](https://linux.die.net/man/1/ssh).
  # [boolean; default: true]
  strictHostKeyChecking:
  # Username - Username for authentication on remote git repo
  # [string; required]
  user:
  # Password/Token - Password or Personal Access Token for authentication on remote g
  # [string; required]
  password:
# [boolean]
fips:
# SNI Routing
sni:
  # Disable SNI-based connection routing - Advanced setting - do not change without c
  # Cribl Support. This setting affects how connections are routed within the Cribl c
  # Changing this setting could affect the scalability of your system.
  # [boolean; default: false]
  disableSNIRouting:
# Sockets for IPC
sockets:
  # Directory - Directory to hold sockets for inter-process communication (IPC), such
  # balancing and Worker Processes. Defaults to your system's temp directory.
  # [string; max: 64]
  directory:
# Shutdown Settings
shutdown:
  # Drain timeout (sec) - How many seconds to wait for writes to complete before Crib
  # down on Workers
  # [number; min: 1; max: 600; default: 10]
  drainTimeout:
# PII Detection
pii:
  # Enable PII detection - Enable periodic detection of PII in this group
  # [boolean; default: false]
  enablePiiDetection:
# Upgrade Group Settings
```

```
  upgradeGroupSettings:
    # Quantity - Percentage of workers to upgrade
    # [number; default: 100]
    quantity:
    # Rolling upgrade - Upgrade workers one at a time
    # [boolean; default: true]
    isRolling:
    # Retry delay (ms) - Number of milliseconds to wait in between upgrade attempts
    # [number; default: 1000]
    retryDelay:
    # Retry count - Number of times to retry failed upgrade
    # [number; default: 5]
    retryCount:
  # Upgrade Settings
  upgradeSettings:
    # Package source - Select download package source
    # One of: cdn | path
    # [string; default: cdn]
    upgradeSource:
    # Enable automatic upgrades - Disable to opt out of automatic upgrades when Cribl r
    # versions
    # [boolean; default: true]
    disableAutomaticUpgrade:
    # Enable Legacy Edge upgrades - Opt in legacy upgrades
    # [boolean; default: false]
    enableLegacyEdgeUpgrade:
    # Upgrade check period - Interval over which to check for out-of-date Nodes
    # [string; default: 1h; required]
    automaticUpgradeCheckPeriod:
    # Custom Path - Provide your own URLs or local paths for platform-specific Cribl pa
    # [required]
    packageUrls:
      # Platform-Specific Package Location - Package HTTP URL or local path
      # [string; min: 1; required]
      packageUrl:
      # Package Hash Location - Package's MD5 or SHA256 hash HTTP URL or local path
      # [string; min: 1]
      packageHashUrl:
```

Copy

Example `cribl.yml`:

$CRIBL_HOME/default/cribl/cribl.yml

```
api:
  host: 0.0.0.0
  port: 9000
```

10.5.4. Groups.Yml

`groups.yml` maintains a list of groups and their configuration versions.

groups.yml

```
# Worker Group type - Whether this group accepts Hybrid (on-prem/customer-managed) nc
# Cribl-managed Cribl.Cloud nodes. (This field is only available in Cribl.Cloud.)
# [boolean; default: true]
onPrem:
# ID - Unique identifier for the group
# [string; required]
id:
# Enable teleporting to nodes - Enable remote access to nodes in this group
# [boolean; default: false]
workerRemoteAccess:
```

10.5.5. Inputs.Yml

`inputs.yml` contains settings for configuring inputs into Cribl.

inputs.yml

```
inputs: # [object]
  collection_input: # [object]
    type: # [string] Input Type
    breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
    staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
    sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

    # -------------- if sendToRoutes is false ---------------

    connections: # [array] Use QuickConnect - Direct connections to Destinations, and
      - pipeline: # [string] Pipeline or Pack
        output: # [string] Destination

    # ------------------------------------------------------

    preprocess: # [object]
      disabled: # [boolean] Disabled

      # -------------- if disabled is false ---------------

      command: # [string] Command - Command to feed the data through (via stdin) and
      args: # [array of strings] Arguments - Arguments to be added to the custom comm

      # ------------------------------------------------------

    throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
    metadata: # [array] Fields - Fields to add to events from this input
      - name: # [string] Field Name
        value: # [string] Value - JavaScript expression to compute field's value, enc
    disabled: # [boolean] Disabled
    pipeline: # [string] Pipeline - Pipeline to process data from this Source before
    environment: # [string] Environment - Optionally, enable this config only on a sp
    pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

    # -------------- if pqEnabled is true ---------------

    pq: # [object]
      mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
      maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
      commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
      maxFileSize: # [string] File size limit - The maximum size to store in each que
      maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
```

```
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
kafka_input: # [object]
  type: # [string] Input Type
  brokers: # [array of strings] Bootstrap servers - Enter each Kafka bootstrap serv
  topics: # [array of strings] [required] Topic - Topic to subscribe to. Warning: T
  groupId: # [string] Group ID - The consumer group to which this instance belongs.
  fromBeginning: # [boolean] From beginning - Leave enabled if you want the Source,
  kafkaSchemaRegistry: # [object] Kafka Schema Registry Authentication
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    schemaRegistryURL: # [string] Schema Registry URL - URL for accessing the Confl
    schemaType: # [string] Schema type - The schema format used to encode and decod
    connectionTimeout: # [number; minimum: 1000, maximum: 60000] Connection timeout
    requestTimeout: # [number; minimum: 1000, maximum: 60000] Request timeout (ms)
    maxRetries: # [number; maximum: 100] Retry limit - Maximum number of times to t
    auth: # [object]  - Credentials to use when authenticating with the schema regi
      disabled: # [boolean] Disabled

      # -------------- if disabled is false ---------------

      credentialsSecret: # [string] Credentials secret - Select or create a secret

      # ------------------------------------------------------

    tls: # [object] TLS settings (client side)
      disabled: # [boolean] Disabled

      # -------------- if disabled is false ---------------

      rejectUnauthorized: # [boolean] Validate server certs - Reject certificates t
                  trusted CA (such as the system's). Defaults to Enabled. Overrides
      servername: # [string] Server name (SNI) - Server name for the SNI (Server Na
      certificateName: # [string] Certificate - The name of the predefined certific
      caPath: # [string] CA certificate path - Path on client in which to find CA c
      privKeyPath: # [string] Private key path (mutual auth) - Path on client in wh
      certPath: # [string] Certificate path (mutual auth) - Path on client in which
```

```
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------


  # --------------------------------------------------------

connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
sasl: # [object] Authentication - Authentication parameters to use when connectir
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  mechanism: # [string] SASL mechanism
  oauthEnabled: # [boolean] Enable OAuth - Enable OAuth authentication

  # ------------------------------------------------------

tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version
```

```
  # ------------------------------------------------------

sessionTimeout: # [number; minimum: 1000, maximum: 3600000] Session timeout (ms)
  Timeout used to detect client failures when using Kafka's group-management faci
  If the client sends no heartbeats to the broker before the timeout expires,
  the broker will remove the client from the group and initiate a rebalance.
  Value must be between the broker's configured group.min.session.timeout.ms and
  See [Kafka's documentation](https://kafka.apache.org/documentation/#consumercor
rebalanceTimeout: # [number; minimum: 1000, maximum: 3600000] Rebalance timeout (
  Maximum allowed time for each worker to join the group after a rebalance begins
  If the timeout is exceeded, the coordinator broker will remove the worker from
  See [Kafka's documentation](https://kafka.apache.org/documentation/#connectconf
heartbeatInterval: # [number; minimum: 1000, maximum: 3600000] Heartbeat interval
  Expected time between heartbeats to the consumer coordinator when using Kafka's
  Value must be lower than sessionTimeout and typically should not exceed 1/3 of
  See [Kafka's documentation](https://kafka.apache.org/documentation/#consumercor
autoCommitInterval: # [number; minimum: 1000, maximum: 3600000] Offset commit int
autoCommitThreshold: # [number; minimum: 1, maximum: 10000] Offset commit thresho
maxBytesPerPartition: # [number; minimum: 1, maximum: 10000000] Byte limit, per p
maxBytes: # [number; minimum: 1, maximum: 1000000000] Byte limit - Maximum number
maxSocketErrors: # [number; maximum: 100] Error limit, per socket - Maximum numbe
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
```

```
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
msk_input: # [object]
  type: # [string] Input Type
  brokers: # [array of strings] Bootstrap servers - Enter each Kafka bootstrap serv
  topics: # [array of strings] [required] Topic - Topic to subscribe to. Warning: T
  groupId: # [string] Group ID - The consumer group to which this instance belongs.
  fromBeginning: # [boolean] From beginning - Leave enabled if you want the Source,
  sessionTimeout: # [number; minimum: 1000, maximum: 3600000] Session timeout (ms)
    Timeout used to detect client failures when using Kafka's group-management faci
    If the client sends no heartbeats to the broker before the timeout expires,
    the broker will remove the client from the group and initiate a rebalance.
    Value must be between the broker's configured group.min.session.timeout.ms and
    See [Kafka's documentation](https://kafka.apache.org/documentation/#consumercor
  rebalanceTimeout: # [number; minimum: 1000, maximum: 3600000] Rebalance timeout (
    Maximum allowed time for each worker to join the group after a rebalance begins
    If the timeout is exceeded, the coordinator broker will remove the worker from
    See [Kafka's documentation](https://kafka.apache.org/documentation/#connectconf
  heartbeatInterval: # [number; minimum: 1000, maximum: 3600000] Heartbeat interval
    Expected time between heartbeats to the consumer coordinator when using Kafka's
    Value must be lower than sessionTimeout and typically should not exceed 1/3 of
    See [Kafka's documentation](https://kafka.apache.org/documentation/#consumercor
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  kafkaSchemaRegistry: # [object] Kafka Schema Registry Authentication
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    schemaRegistryURL: # [string] Schema Registry URL - URL for accessing the Confl
    schemaType: # [string] Schema type - The schema format used to encode and decod
    connectionTimeout: # [number; minimum: 1000, maximum: 60000] Connection timeout
    requestTimeout: # [number; minimum: 1000, maximum: 60000] Request timeout (ms)
```

```
  maxRetries: # [number; maximum: 100] Retry limit - Maximum number of times to t
  auth: # [object]  - Credentials to use when authenticating with the schema regi
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    credentialsSecret: # [string] Credentials secret - Select or create a secret

    # ----------------------------------------------------

  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates t
                trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Na
    certificateName: # [string] Certificate - The name of the predefined certific
    caPath: # [string] CA certificate path - Path on client in which to find CA c
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in wh
    certPath: # [string] Certificate path (mutual auth) - Path on client in which
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ----------------------------------------------------


  # ------------------------------------------------------

connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
awsAuthenticationMethod: # [string] [required] Authentication method - AWS auther

# -------------- if awsAuthenticationMethod is manual ---------------
```

```
awsApiKey: # [string] Access key

# -------------------------------------------------------


# -------------- if awsAuthenticationMethod is secret ---------------

awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

# -------------------------------------------------------

awsSecretKey: # [string] Secret key
region: # [string] [required] Region - Region where the MSK cluster is located
endpoint: # [string] Endpoint - MSK cluster service endpoint. If empty, defaults
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for MSK - Use Assume Role credentials to acc
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # -------------------------------------------------------

autoCommitInterval: # [number; minimum: 1000, maximum: 3600000] Offset commit int
autoCommitThreshold: # [number; minimum: 1, maximum: 10000] Offset commit thresho
maxBytesPerPartition: # [number; minimum: 1, maximum: 10000000] Byte limit, per p
maxBytes: # [number; minimum: 1, maximum: 1000000000] Byte limit - Maximum numbe
```

```
  maxSocketErrors: # [number; maximum: 100] Error limit, per socket - Maximum numbe
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
http_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array of strings] Auth tokens - Shared secrets to be provided by a
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
```

```
  privKeyPath: # [string] Private key path - Path on server containing the privat
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------

maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an in
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
criblAPI: # [string] Cribl HTTP event API - Absolute path on which to listen for
elasticAPI: # [string] Elasticsearch API endpoint (Bulk API) - Absolute path on w
splunkHecAPI: # [string] Splunk HEC endpoint - Absolute path on which listen for
splunkHecAcks: # [boolean] Enable Splunk HEC acknowledgements
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
authTokensExt: # [array] Auth tokens - Shared secrets to be provided by any clien
  - token: # [string] Token - Shared secret to be provided by any client (Authori
    description: # [string] Description
    metadata: # [array] Fields - Fields to add to events referencing this token
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, e
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination
```

```
  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
splunk_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
  maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
```

```
socketIdleTimeout: # [number] Socket idle timeout (seconds) - How long Cribl Stre
socketEndingMaxWait: # [number] Forced socket termination timeout (seconds) - How
socketMaxLifespan: # [number] Socket max lifespan (seconds) - The maximum duratic
enableProxyHeader: # [boolean] Enable proxy protocol - Enable if the connection i
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
authTokens: # [array] Auth tokens - Shared secrets to be provided by any Splunk f
  - token: # [string] Token - Shared secrets to be provided by any Splunk forwarc
    description: # [string] Description
maxS2Sversion: # [string] Max S2S version - The highest S2S protocol version to a

# -------------- if maxS2Sversion is v4 ---------------

useFwdTimezone: # [boolean] Use Universal Forwarder time zone - Event Breakers wi
dropControlFields: # [boolean] Drop control fields - Drop Splunk control fields s
extractMetrics: # [boolean] Extract metrics - Extract and process Splunk-generate
compress: # [string] Compression - Controls whether to support reading compressed

# ----------------------------------------------------------

description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ----------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
```

```
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
splunk_search_input: # [object]
  type: # [string] Input Type
  searchHead: # [string] Search head - Search head base URL. Can be an expression.
  search: # [string] [required] Search - Enter Splunk search here. Examples: 'index
  earliest: # [string] Earliest - The earliest time boundary for the search. Can be
  latest: # [string] Latest - The latest time boundary for the search. Can be an ex
  cronSchedule: # [string] [required] Cron schedule - A cron schedule on which to r
  endpoint: # [string] [required] Search endpoint - REST API used to create a searc
  outputMode: # [string] [required] Output mode - Format of the returned output
  endpointParams: # [array] Endpoint parameters - Optional request parameters to se
    - name: # [string] Parameter Name
      value: # [string] Value - JavaScript expression to compute the parameter's va
  endpointHeaders: # [array] Endpoint headers - Optional request headers to send to
    - name: # [string] Header Name
      value: # [string] Value - JavaScript expression to compute the header's value
  logLevel: # [string] Log level - Collector runtime log level (verbosity)
  requestTimeout: # [number; maximum: 2400] Request timeout (seconds) - HTTP reques
  useRoundRobinDns: # [boolean] Round-robin DNS - When a DNS server returns multipl
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
  encoding: # [string] Encoding - Character encoding to use when parsing ingested d
  keepAliveTime: # [number; minimum: 10] Keep alive time (seconds) - How often work
  jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run (e.g.
  maxMissedKeepAlives: # [number; minimum: 2] Worker timeout (periods) - The number
  ttl: # [string] Time to live - Time to keep the job's artifacts on disk after job
  ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled,
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  retryRules: # [object]
    type: # [string] Retry type - The algorithm to use when performing HTTP retries

    # -------------- if type is static ---------------

    codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigger
```

```
    interval: # [number; maximum: 20000] Wait (ms) - Time interval between failed r
    limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
    enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
    retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
    retryConnectReset: # [boolean] Retry connection reset - Retry request when a co

    # ------------------------------------------------------


    # -------------- if type is backoff ---------------

    interval: # [number; maximum: 20000] Initial retry interval (ms) - Time interva
    limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
    multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for e
    codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigger
    enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
    retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
    retryConnectReset: # [boolean] Retry connection reset - Retry request when a co

    # ------------------------------------------------------

breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
authType: # [string] Authentication type - Splunk Search authentication type

# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# ------------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# ------------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that
```

```
# ------------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# ------------------------------------------------------


# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth t
  - name: # [string] Name - OAuth header name
    value: # [string] Value - OAuth header value

# ------------------------------------------------------

description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
```

```
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ---------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
splunk_hec_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array] Auth tokens - Shared secrets to be provided by any client (
    - authType: # [string] Authentication method - Select Manual to enter an auth t
      enabled: # [boolean] Enable token
      description: # [string] Description - Optional token description
      allowedIndexesAtToken: # [array of strings] Allowed indexes - Enter the value
      metadata: # [array] Fields - Fields to add to events referencing this token
      - name: # [string] Field Name
        value: # [string] Value - JavaScript expression to compute field's value, e
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version
```

```
  # ------------------------------------------------------

maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
splunkHecAPI: # [string] [required] Splunk HEC endpoint - Absolute path on which
metadata: # [array] Fields - Fields to add to every event. Overrides fields added
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
allowedIndexes: # [array of strings] Allowed indexes - List values allowed in HEC
splunkHecAcks: # [boolean] Splunk HEC acks - Enable Splunk HEC acknowledgements
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
useFwdTimezone: # [boolean] Use Universal Forwarder time zone (S2S only) - Event
dropControlFields: # [boolean] Drop control fields (S2S only) - Drop Splunk contr
extractMetrics: # [boolean] Extract metrics (S2S only) - Extract and process Splu
accessControlAllowOrigin: # [array of strings] CORS allowed origins - Optionally,
accessControlAllowHeaders: # [array of strings] CORS allowed headers - Optionally
emitTokenMetrics: # [boolean] Emit per-token request metrics - Emit per-token (<p
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------
```

```
  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
azure_blob_input: # [object]
  type: # [string] Input Type
  queueName: # [string] Queue - The storage account queue name blob notifications w
  fileFilter: # [string] Filename filter - Regex matching file names to download ar
  visibilityTimeout: # [number; maximum: 604800] Visibility timeout (secs) - The du
  numReceivers: # [number; minimum: 1, maximum: 100] Number of receivers - How many
  maxMessages: # [number; minimum: 1, maximum: 32] Message limit - The maximum numb
  servicePeriodSecs: # [number; minimum: 1, maximum: 10] Service period (secs) - Th
  skipOnError: # [boolean] Skip file on error - Skip files that trigger a processin
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
  staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
  parquetChunkSizeMB: # [number; minimum: 1, maximum: 100] Parquet chunk size limit
  parquetChunkDownloadTimeout: # [number; minimum: 1, maximum: 3600] Parquet chunk
  authType: # [string] Authentication method

  # -------------- if authType is manual ---------------

  connectionString: # [string] Connection string - Enter your Azure Storage account

  # ------------------------------------------------------


  # -------------- if authType is secret ---------------

  textSecret: # [string] Connection string (text secret) - Select or create a store

  # ------------------------------------------------------
```

```
# -------------- if authType is clientSecret ---------------

storageAccountName: # [string] Storage account name - The name of your Azure stor
tenantId: # [string] Tenant ID - The service principal's tenant ID
clientId: # [string] Client ID - The service principal's client ID
azureCloud: # [string] Azure Cloud - The Azure cloud to use. Defaults to Azure Pu
endpointSuffix: # [string] Endpoint suffix - Endpoint suffix for the service URL.
clientTextSecret: # [string] Client secret (text secret) - Select or create a stc

# ----------------------------------------------------------


# -------------- if authType is clientCert ---------------

storageAccountName: # [string] Storage account name - The name of your Azure stor
tenantId: # [string] Tenant ID - The service principal's tenant ID
clientId: # [string] Client ID - The service principal's client ID
azureCloud: # [string] Azure Cloud - The Azure cloud to use. Defaults to Azure Pu
endpointSuffix: # [string] Endpoint suffix - Endpoint suffix for the service URL.
certificate: # [object]
  certificateName: # [string] Certificate - The certificate you registered as cre

# ----------------------------------------------------------

description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ----------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------
```

```
  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
elastic_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  maxActiveReq: # [number] Active request limit - Maximum number of active requests
  maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
  enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
  captureHeaders: # [boolean] Capture request headers - Add request headers to even
  activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
  requestTimeout: # [number] Request timeout (seconds) - How long to wait for an in
  socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
  keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
  enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
  ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
```

```
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
elasticAPI: # [string] [required] Elasticsearch API endpoint - Absolute path on w
authType: # [string] Authentication type

# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# ---------------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ---------------------------------------------------------


# -------------- if authType is authTokens ---------------

authTokens: # [array of strings] Token - Bearer tokens to include in the authoriz

# ---------------------------------------------------------

apiVersion: # [string] API version - The API version to use for communicating wit

# -------------- if apiVersion is custom ---------------

customAPIVersion: # [string] Custom API Version - Custom version information to r

# ---------------------------------------------------------

extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
proxyMode: # [object]
  enabled: # [boolean] Enable proxy mode - Enable proxying of non-bulk API reques

  # -------------- if enabled is true ---------------
```

```
    url: # [string] Proxy URL - URL of the Elastic server to proxy non-bulk request
    rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certi
    removeHeaders: # [array of strings] Remove headers - List of headers to remove
    timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Proxy request tim
    authType: # [string] Authentication method - Enter credentials directly, or sel

    # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
confluent_cloud_input: # [object]
  type: # [string] Input Type
  brokers: # [array of strings] Bootstrap servers - List of Confluent Cloud bootstr
  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled
```

```
  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------

topics: # [array of strings] [required] Topic - Topic to subscribe to. Warning: T
groupId: # [string] Group ID - The consumer group to which this instance belongs.
fromBeginning: # [boolean] From beginning - Leave enabled if you want the Source,
kafkaSchemaRegistry: # [object] Kafka Schema Registry Authentication
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  schemaRegistryURL: # [string] Schema Registry URL - URL for accessing the Confl
  schemaType: # [string] Schema type - The schema format used to encode and decod
  connectionTimeout: # [number; minimum: 1000, maximum: 60000] Connection timeout
  requestTimeout: # [number; minimum: 1000, maximum: 60000] Request timeout (ms)
  maxRetries: # [number; maximum: 100] Retry limit - Maximum number of times to t
  auth: # [object]  - Credentials to use when authenticating with the schema regi
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    credentialsSecret: # [string] Credentials secret - Select or create a secret

    # ------------------------------------------------------

  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------
```

```
    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates t
                trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Na
    certificateName: # [string] Certificate - The name of the predefined certific
    caPath: # [string] CA certificate path - Path on client in which to find CA c
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in wh
    certPath: # [string] Certificate path (mutual auth) - Path on client in which
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # -------------------------------------------------------


  # -------------------------------------------------------

connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
sasl: # [object] Authentication - Authentication parameters to use when connectin
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  mechanism: # [string] SASL mechanism
  oauthEnabled: # [boolean] Enable OAuth - Enable OAuth authentication

  # -------------------------------------------------------

sessionTimeout: # [number; minimum: 1000, maximum: 3600000] Session timeout (ms)
  Timeout used to detect client failures when using Kafka's group-management faci
  If the client sends no heartbeats to the broker before the timeout expires,
  the broker will remove the client from the group and initiate a rebalance.
  Value must be between the broker's configured group.min.session.timeout.ms and
  See [Kafka's documentation](https://kafka.apache.org/documentation/#consumercon
rebalanceTimeout: # [number; minimum: 1000, maximum: 3600000] Rebalance timeout (
  Maximum allowed time for each worker to join the group after a rebalance begins
  If the timeout is exceeded, the coordinator broker will remove the worker from
```

```
  See [Kafka's documentation](https://kafka.apache.org/documentation/#connectconf
heartbeatInterval: # [number; minimum: 1000, maximum: 3600000] Heartbeat interval
  Expected time between heartbeats to the consumer coordinator when using Kafka's
  Value must be lower than sessionTimeout and typically should not exceed 1/3 of
  See [Kafka's documentation](https://kafka.apache.org/documentation/#consumercon
autoCommitInterval: # [number; minimum: 1000, maximum: 3600000] Offset commit int
autoCommitThreshold: # [number; minimum: 1, maximum: 10000] Offset commit thresho
maxBytesPerPartition: # [number; minimum: 1, maximum: 10000000] Byte limit, per p
maxBytes: # [number; minimum: 1, maximum: 1000000000] Byte limit - Maximum number
maxSocketErrors: # [number; maximum: 100] Error limit, per socket - Maximum numbe
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# --------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data

# --------------------------------------------------------

streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
```

```
grafana_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ---------------------------------------------------------

  maxActiveReq: # [number] Active request limit - Maximum number of active requests
  maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
  enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
  captureHeaders: # [boolean] Capture request headers - Add request headers to ever
  activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
  requestTimeout: # [number] Request timeout (seconds) - How long to wait for an in
  socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
  keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep alive timeout (second
  enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
  ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
  ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
  prometheusAPI: # [string] Remote Write API endpoint - Absolute path on which to l
  lokiAPI: # [string] Logs API endpoint - Absolute path on which to listen for Loki
  prometheusAuth: # [object]
    authType: # [string] Authentication type - Remote Write authentication type

    # -------------- if authType is basic ---------------

    username: # [string] Username
    password: # [string] Password

    # ---------------------------------------------------------
```

```
    # -------------- if authType is token ---------------

    token: # [string] Token - Bearer token to include in the authorization header

    # ----------------------------------------------------


    # -------------- if authType is credentialsSecret ---------------

    credentialsSecret: # [string] Credentials secret - Select or create a secret th

    # ----------------------------------------------------


    # -------------- if authType is textSecret ---------------

    textSecret: # [string] Token (text secret) - Select or create a stored text sec

    # ----------------------------------------------------


    # -------------- if authType is oauth ---------------

    loginUrl: # [string] Login URL - URL for OAuth
    secretParamName: # [string] OAuth Secret parameter name - Secret parameter name
    secret: # [string] OAuth secret - Secret parameter value to pass in request bod
    tokenAttributeName: # [string] Token attribute name - Name of the auth token at
    spacer: # [null]
    authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp
    tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (sec
    oauthParams: # [array] OAuth parameters - Additional parameters to send in the
      - name: # [string] Name - OAuth parameter name
        value: # [string] Value - OAuth parameter value
    oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth
      - name: # [string] Name - OAuth header name
        value: # [string] Value - OAuth header value

    # ----------------------------------------------------

  lokiAuth: # [object]
    authType: # [string] Authentication type - Loki logs authentication type
```

```
# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# ---------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# ---------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret th

# ---------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text sec

# ---------------------------------------------------


# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name
secret: # [string] OAuth secret - Secret parameter value to pass in request bod
tokenAttributeName: # [string] Token attribute name - Name of the auth token at
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (sec
oauthParams: # [array] OAuth parameters - Additional parameters to send in the
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth
```

```
      - name: # [string] Name - OAuth header name
        value: # [string] Value - OAuth header value

    # -------------------------------------------------------

  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
loki_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
```

```
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # -------------------------------------------------------

maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
lokiAPI: # [string] [required] Logs API endpoint - Absolute path on which to list
authType: # [string] Authentication type - Loki logs authentication type

# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# -------------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# -------------------------------------------------------
```

```
# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ----------------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# ----------------------------------------------------------


# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth t
  - name: # [string] Name - OAuth header name
    value: # [string] Value - OAuth header value

# ----------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
```

```
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
prometheus_rw_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------
```

```
maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
prometheusAPI: # [string] [required] Remote Write API endpoint - Absolute path or
authType: # [string] Authentication type - Remote Write authentication type

# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# ---------------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# ---------------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ---------------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# ---------------------------------------------------------
```

```
# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth l
  - name: # [string] Name - OAuth header name
    value: # [string] Value - OAuth header value

# --------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# --------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
```

```
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
prometheus_input: # [object]
  type: # [string] Input Type
  dimensionList: # [array of strings] Extra Dimensions - Other dimensions to includ
  discoveryType: # [string] Discovery Type - Target discovery mechanism. Use static

  # -------------- if discoveryType is static ---------------

  targetList: # [array of strings] Targets - List of Prometheus targets to pull met

  # ------------------------------------------------------


  # -------------- if discoveryType is dns ---------------

  nameList: # [array of strings] DNS Names - List of DNS names to resolve
  recordType: # [string] Record Type - DNS Record type to resolve
  scrapeProtocol: # [string] Metrics Protocol - Protocol to use when collecting met
  scrapePath: # [string] Metrics Path - Path to use when collecting metrics from di

  # ------------------------------------------------------


  # -------------- if discoveryType is ec2 ---------------

  usePublicIp: # [boolean] Use Public IP - Use public IP address for discovered tar
  scrapeProtocol: # [string] Metrics Protocol - Protocol to use when collecting met
  scrapePort: # [number; minimum: 1, maximum: 65535] Metrics Port - The port number
  scrapePath: # [string] Metrics Path - Path to use when collecting metrics from di
  searchFilter: # [array] Search Filter - EC2 Instance Search Filter
    - Name: # [string] Filter Name - Search filter attribute name, see: https://doc
      Values: # [array of strings] Filter Values - Search Filter Values, if empty d
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me
  awsSecretKey: # [string] Secret key
  region: # [string] Region - Region where the EC2 is located
  endpoint: # [string] Endpoint - EC2 service endpoint. If empty, defaults to the A
  signatureVersion: # [string] Signature version - Signature version to use for sig
```

```
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for EC2 - Use Assume Role credentials to acc
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du

# -------------------------------------------------------

interval: # [number; minimum: 1, maximum: 60] Poll Interval - How often in minute
logLevel: # [string] [required] Log Level - Collector runtime Log Level
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
keepAliveTime: # [number; minimum: 10] Keep alive time (seconds) - How often work
jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run (e.g.
maxMissedKeepAlives: # [number; minimum: 2] Worker timeout (periods) - The number
ttl: # [string] Time to live - Time to keep the job's artifacts on disk after job
ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled,
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
authType: # [string] Authentication method - Enter credentials directly, or selec
username: # [string] Username - Username for Prometheus Basic authentication
password: # [string] Password - Password for Prometheus Basic authentication

# -------------- if authType is manual ---------------


# -------------------------------------------------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# -------------- if authType is secret ---------------


# -------------------------------------------------------

description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------
```

```
  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
edge_prometheus_input: # [object]
  type: # [string] Input Type
  dimensionList: # [array of strings] Extra Dimensions - Other dimensions to includ
  discoveryType: # [string] [required] Discovery Type - Target discovery mechanism.

  # -------------- if discoveryType is static ---------------

  targets: # [array] Targets
    - protocol: # [string] Protocol - Protocol to use when collecting metrics
      host: # [string] Host - Name of host from which to pull metrics.
      port: # [number; minimum: 1, maximum: 65535] Port - The port number in the me
      path: # [string] Path - Path to use when collecting metrics from discovered t

  # ------------------------------------------------------



  # -------------- if discoveryType is dns ---------------

  nameList: # [array of strings] DNS Names - List of DNS names to resolve
  recordType: # [string] Record Type - DNS Record type to resolve
```

```
scrapeProtocol: # [string] Metrics Protocol - Protocol to use when collecting met
scrapePath: # [string] Metrics Path - Path to use when collecting metrics from di

# ------------------------------------------------------


# -------------- if discoveryType is ec2 ---------------

usePublicIp: # [boolean] Use Public IP - Use public IP address for discovered tar
scrapeProtocol: # [string] Metrics Protocol - Protocol to use when collecting met
scrapePort: # [number; minimum: 1, maximum: 65535] Metrics Port - The port numbe
scrapePath: # [string] Metrics Path - Path to use when collecting metrics from di
searchFilter: # [array] Search Filter - EC2 Instance Search Filter
  - Name: # [string] Filter Name - Search filter attribute name, see: https://doc
    Values: # [array of strings] Filter Values - Search Filter Values, if empty c
awsAuthenticationMethod: # [string] Authentication method - AWS authentication me
awsSecretKey: # [string] Secret key
region: # [string] Region - Region where the EC2 is located
endpoint: # [string] Endpoint - EC2 service endpoint. If empty, defaults to the A
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for EC2 - Use Assume Role credentials to acc
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du

# ------------------------------------------------------


# -------------- if discoveryType is k8s-node ---------------

scrapeProtocol: # [string] Protocol - Protocol to use when collecting metrics
scrapePort: # [number; minimum: 1, maximum: 65535] Port - The port number in the
scrapePath: # [string] Path - Path to use when collecting metrics from discovered

# ------------------------------------------------------


# -------------- if discoveryType is k8s-pods ---------------

scrapeProtocolExpr: # [string] Protocol - Protocol to use when collecting metrics
scrapePortExpr: # [string] Port - The port number in the metrics URL for discover
```

```
  scrapePathExpr: # [string] Path - Path to use when collecting metrics from discov
  podFilter: # [array] Filter Rules -
Add rules to decide which pods to discover for metrics.
Pods are searched if no rules are given or of all the rules'
expressions evaluate to true.

    - filter: # [string] Filter Expression - JavaScript expression applied to pods
      description: # [string] Description - Optional description of this rule's pur

  # ------------------------------------------------------

  interval: # [number; minimum: 2] Poll Interval - How often in seconds to scrape t
  timeout: # [number; maximum: 60000] HTTP Connection Timeout - Timeout, in millise
  persistence: # [object] Disk Spooling
    enable: # [boolean] Enable disk spooling - Spool events on disk for Cribl Edge

    # -------------- if enable is true ---------------

    timeWindow: # [string] Bucket time span - Time period for grouping spooled ever
    maxDataSize: # [string] Data size limit - Maximum disk space that can be consum
    maxDataTime: # [string] Data age limit - Maximum amount of time to retain data
    compress: # [string] Compression - Data compression format. Default is gzip.

    # ------------------------------------------------------

  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  authType: # [string] Authentication method - Enter credentials directly, or selec
  username: # [string] Username - Username for Prometheus Basic authentication
  password: # [string] Password - Password for Prometheus Basic authentication

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  credentialsSecret: # [string] Credentials secret - Select or create a secret that

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------
```

```
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # --------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # --------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
office365_mgmt_input: # [object]
  type: # [string] Input Type
  planType: # [string] [required] Subscription plan - Office 365 subscription plan
  tenantId: # [string] Tenant ID - Office 365 Azure Tenant ID
  appId: # [string] [required] App ID - Office 365 Azure Application ID
  timeout: # [number; maximum: 2400] Request timeout (seconds) - HTTP request inact
  keepAliveTime: # [number; minimum: 10] Keep alive time (seconds) - How often work
  jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run (e.g.
  maxMissedKeepAlives: # [number; minimum: 2] Worker timeout (periods) - The number
  ttl: # [string] Time to live - Time to keep the job's artifacts on disk after job
  ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled,
  metadata: # [array] Fields - Fields to add to events from this input
```

```
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
publisherIdentifier: # [string] Publisher Identifier - Optional Publisher Identif
contentConfig: # [array] Content Types - Enable Office 365 Management Activity AP
  - contentType: # [string] Content Type - Office 365 Management Activity API Cor
    description: # [string] Interval Description - If interval type is minutes th
    interval: # [number; minimum: 1, maximum: 60] Interval
    logLevel: # [string] Log Level - Collector runtime Log Level
    enabled: # [boolean] Enabled
ingestionLag: # [number; maximum: 7200] Ingestion lag (minutes) - Use this settir
retryRules: # [object]
  type: # [string] Retry type - The algorithm to use when performing HTTP retries

  # -------------- if type is static ---------------

  codes: # [array of numbers] Retry HTTP codes - List of http codes that trigger
  interval: # [number; maximum: 20000] Wait (ms) - Time interval between failed r
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a co

  # ------------------------------------------------------


  # -------------- if type is backoff ---------------

  interval: # [number; maximum: 20000] Initial retry interval (ms) - Time interva
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
  multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for e
  codes: # [array of numbers] Retry HTTP codes - List of http codes that trigger
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a co

  # ------------------------------------------------------

authType: # [string] Authentication method - Enter client secret directly, or sel
clientSecret: # [string] Client secret - Office 365 Azure client secret

# -------------- if authType is manual ---------------
```

```
  # ------------------------------------------------------

  textSecret: # [string] Client secret (text secret) - Select or create a stored te

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
office365_service_input: # [object]
  type: # [string] Input Type
  planType: # [string] Subscription plan - Office 365 subscription plan for your or
  tenantId: # [string] Tenant ID - Office 365 Azure Tenant ID
```

```
appId: # [string] [required] App ID - Office 365 Azure Application ID
timeout: # [number; maximum: 2400] Request timeout (seconds) - HTTP request inact
keepAliveTime: # [number; minimum: 10] Keep alive time (seconds) - How often work
jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run (e.g.
maxMissedKeepAlives: # [number; minimum: 2] Worker timeout (periods) - The numbe
ttl: # [string] Time to live - Time to keep the job's artifacts on disk after jo
ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled,
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
contentConfig: # [array] Content Types - Enable Office 365 Service Communication
  - contentType: # [string] Content Type - Office 365 Services API Content Type
    description: # [string] Interval Description - If interval type is minutes th
    interval: # [number; maximum: 60] Interval
    logLevel: # [string] Log Level - Collector runtime Log Level
    enabled: # [boolean] Enabled
retryRules: # [object]
  type: # [string] Retry type - The algorithm to use when performing HTTP retries

  # -------------- if type is static ---------------

  codes: # [array of numbers] Retry HTTP codes - List of http codes that trigger
  interval: # [number; maximum: 20000] Wait (ms) - Time interval between failed r
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a co

  # -----------------------------------------------------


  # -------------- if type is backoff ---------------

  interval: # [number; maximum: 20000] Initial retry interval (ms) - Time interva
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
  multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for e
  codes: # [array of numbers] Retry HTTP codes - List of http codes that trigger
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a co

  # -----------------------------------------------------
```

```
authType: # [string] Authentication method - Enter client secret directly, or sel
clientSecret: # [string] Client secret - Office 365 Azure client secret

# -------------- if authType is manual ---------------


# -----------------------------------------------------

textSecret: # [string] Client secret (text secret) - Select or create a stored te

# -------------- if authType is secret ---------------


# -----------------------------------------------------

description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# -----------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data

# -----------------------------------------------------
```

```
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
office365_msg_trace_input: # [object]
  type: # [string] Input Type
  url: # [string] Report URL - URL to use when retrieving report data.
  interval: # [number; minimum: 1, maximum: 60] [required] Poll interval - How ofte
  startDate: # [string] Date range start - Backward offset for the search range's h
  endDate: # [string] Date range end - Backward offset for the search range's tail.
  timeout: # [number; maximum: 2400] Request timeout (seconds) - HTTP request inact
  disableTimeFilter: # [boolean] Disable time filter - Disables time filtering of e
  authType: # [string] Authentication method - Select authentication method.

  # -------------- if authType is manual ---------------

  username: # [string] Username - Username to run Message Trace API call.
  password: # [string] Password - Password to run Message Trace API call.

  # ----------------------------------------------------


  # -------------- if authType is secret ---------------

  credentialsSecret: # [string] Credentials secret - Select or create a secret that

  # ----------------------------------------------------


  # -------------- if authType is oauth ---------------

  clientSecret: # [string] Client secret - client_secret to pass in the OAuth reque
  tenantId: # [string] Tenant identifier - Directory ID (tenant identifier) in Azur
  clientId: # [string] Client ID - client_id to pass in the OAuth request parameter
  resource: # [string] Resource - Resource to pass in the OAuth request parameter.
  planType: # [string] Subscription plan - Office 365 subscription plan for your or

  # ----------------------------------------------------


  # -------------- if authType is oauthSecret ---------------

  textSecret: # [string] Client secret - Select or create a secret that references
  tenantId: # [string] Tenant identifier - Directory ID (tenant identifier) in Azur
  clientId: # [string] Client ID - client_id to pass in the OAuth request parameter
```

```
resource: # [string] Resource - Resource to pass in the OAuth request parameter.
planType: # [string] Subscription plan - Office 365 subscription plan for your or

# ------------------------------------------------------


# -------------- if authType is oauthCert ---------------

certOptions: # [object]
  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path to the private key to use. Key
  passphrase: # [string] Passphrase - Passphrase to use to decrypt the private ke
  certPath: # [string] [required] Certificate path - Path to the certificate to u
tenantId: # [string] Tenant identifier - Directory ID (tenant identifier) in Azur
clientId: # [string] Client ID - client_id to pass in the OAuth request parameter
resource: # [string] Resource - Resource to pass in the OAuth request parameter.
planType: # [string] Subscription plan - Office 365 subscription plan for your or

# ------------------------------------------------------

rescheduleDroppedTasks: # [boolean] Reschedule tasks - Reschedule tasks that fail
maxTaskReschedule: # [number; minimum: 1] Task reschedule limit - Maximum number
logLevel: # [string] Log level - Log Level (verbosity) for collection runtime beh
jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run (e.g.
keepAliveTime: # [number; minimum: 10] Keep alive time (seconds) - How often work
maxMissedKeepAlives: # [number; minimum: 2] Worker timeout (periods) - The number
ttl: # [string] Time to live - Time to keep the job's artifacts on disk after job
ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled,
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
retryRules: # [object]
  type: # [string] Retry type - The algorithm to use when performing HTTP retries

  # -------------- if type is static ---------------

  codes: # [array of numbers] Retry HTTP codes - List of http codes that trigger
  interval: # [number; maximum: 20000] Wait (ms) - Time interval between failed r
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a co
```

```
  # ------------------------------------------------------


  # -------------- if type is backoff ---------------

  interval: # [number; maximum: 20000] Initial retry interval (ms) - Time interva
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to ret
  multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for e
  codes: # [array of numbers] Retry HTTP codes - List of http codes that trigger
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After head
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retry
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a co

  # -------------------------------------------------------

description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# -------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data

# -------------------------------------------------------
```

```
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
eventhub_input: # [object]
  type: # [string] Input Type
  brokers: # [array of strings] Brokers - List of Event Hubs Kafka brokers to conne
  topics: # [array of strings] [required] Event Hub name - The name of the Event Hu
  groupId: # [string] Group ID - The consumer group this instance belongs to. Defau
  fromBeginning: # [boolean] From beginning - Start reading from earliest available
  connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
  requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
  maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
  maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
  initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
  backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
  authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
  reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
  sasl: # [object] Authentication - Authentication parameters to use when connecti
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    mechanism: # [string] SASL mechanism

    # ----------------------------------------------------

  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha

    # ----------------------------------------------------

  sessionTimeout: # [number; minimum: 6000, maximum: 300000] Session timeout (ms) -
    Timeout (session.timeout.ms in Kafka domain) used to detect client failures whe
    If the client sends no heartbeats to the broker before the timeout expires, the
    Value must be lower than rebalanceTimeout.
    See details [here](https://github.com/Azure/azure-event-hubs-for-kafka/blob/mas
  rebalanceTimeout: # [number; minimum: 1000, maximum: 3600000] Rebalance timeout (
    Maximum allowed time (rebalance.timeout.ms in Kafka domain) for each worker to
    If the timeout is exceeded, the coordinator broker will remove the worker from
    See [Recommended configurations](https://github.com/Azure/azure-event-hubs-for-
```

```
heartbeatInterval: # [number; minimum: 1000, maximum: 3600000] Heartbeat interval
  Expected time (heartbeat.interval.ms in Kafka domain) between heartbeats to the
  Value must be lower than sessionTimeout and typically should not exceed 1/3 of
  See [Recommended configurations](https://github.com/Azure/azure-event-hubs-for-
autoCommitInterval: # [number; minimum: 1000, maximum: 3600000] Offset commit int
autoCommitThreshold: # [number; minimum: 1, maximum: 10000] Offset commit thresho
maxBytesPerPartition: # [number; minimum: 1, maximum: 10000000] Byte limit, per p
maxBytes: # [number; minimum: 1, maximum: 1000000000] Byte limit - Maximum number
maxSocketErrors: # [number; maximum: 100] Error limit, per socket - Maximum numbe
minimizeDuplicates: # [boolean] Minimize duplicates - Minimize duplicate events b
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# --------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data

# --------------------------------------------------------

streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
```

```
exec_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  command: # [string] Command - Command to execute; supports Bourne shell (or CMD c
  retries: # [number] Retry limit - Maximum number of retry attempts in the event t
  scheduleType: # [string] Schedule type - Select a schedule type; either an interv

  # -------------- if scheduleType is interval ---------------

  interval: # [number; minimum: 1] Interval - Interval between command executions i

  # -----------------------------------------------------



  # -------------- if scheduleType is cronSchedule ---------------

  cronSchedule: # [string] Schedule - Cron schedule to execute the command on.

  # -----------------------------------------------------

  breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
  staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -----------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
```

```
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
firehose_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array of strings] Auth tokens - Shared secrets to be provided by a
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  maxActiveReq: # [number] Active request limit - Maximum number of active requests
  maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
  enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
  captureHeaders: # [boolean] Capture request headers - Add request headers to even
  activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
  requestTimeout: # [number] Request timeout (seconds) - How long to wait for an in
  socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
  keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
  enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
  ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
```

```
  ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
google_pubsub_input: # [object]
  type: # [string] Input Type
  topicName: # [string] [required] Topic ID - ID of the topic to receive events fro
  subscriptionName: # [string] Subscription ID - ID of the subscription to use wher
  monitorSubscription: # [boolean] Monitor subscription for new messages - Use wher
  createTopic: # [boolean] Create topic - Create topic if it does not exist
  createSubscription: # [boolean] Create subscription - Create subscription if it o

  # -------------- if createSubscription is true ---------------
```

```
orderedDelivery: # [boolean] Ordered delivery - Receive events in the order they

# ------------------------------------------------------

region: # [string] Region - Region to retrieve messages from. Select 'default' to
googleAuthMethod: # [string] Google authentication method - Choose Auto to use Go

# -------------- if googleAuthMethod is manual ---------------

serviceAccountCredentials: # [string] Service account credentials - Contents of s

# ------------------------------------------------------


# -------------- if googleAuthMethod is secret ---------------

secret: # [string] Service account credentials (text secret) - Select or create a

# ------------------------------------------------------

maxBacklog: # [number; minimum: 1] Backlog limit - If Destination exerts backpres
concurrency: # [number; minimum: 1, maximum: 100] Number of concurrent streams -
requestTimeout: # [number; minimum: 10000] Request timeout (ms) - Pull request ti
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------
```

```
  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
cribl_input: # [object]
  type: # [string] Input Type
  filter: # [string]
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
```

```
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
cribl_tcp_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
  socketIdleTimeout: # [number] Socket idle timeout (seconds) - How long Cribl Stre
  socketEndingMaxWait: # [number] Forced socket termination timeout (seconds) - How
  socketMaxLifespan: # [number] Socket max lifespan (seconds) - The maximum duratio
  enableProxyHeader: # [boolean] Enable proxy protocol - Enable if the connection i
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  enableLoadBalancing: # [boolean] Enable load balancing - Load balance traffic acr
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
```

```
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
cribl_http_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array of strings] Auth tokens - Shared secrets to be provided by a
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------
```

```
maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data

# ------------------------------------------------------
```

```
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
cribl_lake_http_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array of strings] Auth tokens - Shared secrets to be provided by a
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ---------------------------------------------------------

  maxActiveReq: # [number] Active request limit - Maximum number of active requests
  maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
  enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
  captureHeaders: # [boolean] Capture request headers - Add request headers to ever
  activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
  requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
  socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
  keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
  enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
  ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
  ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
  criblAPI: # [string] Cribl HTTP event API - Absolute path on which to listen for
  elasticAPI: # [string] Elasticsearch API endpoint (Bulk API) - Absolute path on w
  splunkHecAPI: # [string] Splunk HEC endpoint - Absolute path on which listen for
  splunkHecAcks: # [boolean] Enable Splunk HEC acknowledgements
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  authTokensExt: # [array] Auth tokens
    - token: # [string] Token
```

```
    description: # [string]
    metadata: # [array] Fields - Fields to add to events referencing this token
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, e
    splunkHecMetadata: # [object]
    enabled: # [boolean]

    # -------------- if enabled is true ---------------

    defaultDataset: # [string]
    allowedIndexesAtToken: # [array of strings]

    # --------------------------------------------------------

    elasticsearchMetadata: # [object]
    enabled: # [boolean]

    # -------------- if enabled is true ---------------

    defaultDataset: # [string]

    # --------------------------------------------------------

description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# --------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
```

```
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
tcpjson_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
  maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
  socketIdleTimeout: # [number] Socket idle timeout (seconds) - How long Cribl Stre
  socketEndingMaxWait: # [number] Forced socket termination timeout (seconds) - How
  socketMaxLifespan: # [number] Socket max lifespan (seconds) - The maximum duratio
  enableProxyHeader: # [boolean] Enable proxy protocol - Enable if the connection i
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  enableLoadBalancing: # [boolean] Enable load balancing - Load balance traffic acr
  authType: # [string] Authentication method - Select Manual to enter an auth token
  authToken: # [string] Auth token - Shared secret to be provided by any client (ir
```

```
  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored text

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
system_metrics_input: # [object]
  type: # [string] Input Type
```

```
interval: # [number; minimum: 1] Polling interval - Time, in seconds, between cor
host: # [object]
  mode: # [string]  - Select level of detail for host metrics

  # -------------- if mode is custom ---------------

  custom: # [object]
    system: # [object]
      mode: # [string]  - Select the level of detail for system metrics

      # -------------- if mode is custom ---------------

      processes: # [boolean] Process metrics - Generate metrics for the numbers c

      # ------------------------------------------------------

    cpu: # [object]
      mode: # [string]  - Select the level of detail for CPU metrics

      # -------------- if mode is custom ---------------

      perCpu: # [boolean] Per-CPU metrics - Generate metrics for each CPU
      detail: # [boolean] Detailed metrics - Generate metrics for all CPU states
      time: # [boolean] CPU time metrics - Generate raw, monotonic CPU time count

      # ------------------------------------------------------

    memory: # [object]
      mode: # [string]  - Select the level of detail for memory metrics

      # -------------- if mode is custom ---------------

      detail: # [boolean] Detailed metrics - Generate metrics for all memory stat

      # ------------------------------------------------------

    network: # [object]
      mode: # [string]  - Select the level of detail for network metrics

      # -------------- if mode is custom ---------------

      devices: # [array of strings] Interface filter - Network interfaces to incl
      perInterface: # [boolean] Per-interface metrics - Generate separate metrics
```

```
      detail: # [boolean] Detailed metrics - Generate full network metrics

      # ------------------------------------------------------

    disk: # [object]
      mode: # [string]  - Select the level of detail for disk metrics

      # -------------- if mode is custom ---------------

      devices: # [array of strings] Device filter - Block devices to include/excl
      mountpoints: # [array of strings] Mountpoint filter - Filesystem mountpoint
      fstypes: # [array of strings] Filesystem type filter - Filesystem types to
      perDevice: # [boolean] Per-device metrics - Generate separate metrics for e
      detail: # [boolean] Detailed metrics - Generate full disk metrics

      # ------------------------------------------------------


  # --------------------------------------------------------

process: # [object]
  sets: # [array] Process sets - Configure sets to collect process metrics
    - name: # [string] Set Name
      filter: # [string] Filter Expression
      includeChildren: # [boolean] Include Child Processes
container: # [object]
  mode: # [string]  - Select the level of detail for container metrics

  # -------------- if mode is basic ---------------

  dockerSocket: # [array of strings] Docker socket - Full paths for Docker's UNI
  dockerTimeout: # [number; minimum: 1] Docker timeout - Timeout, in seconds, for

  # ------------------------------------------------------


  # -------------- if mode is all ---------------

  dockerSocket: # [array of strings] Docker socket - Full paths for Docker's UNI
  dockerTimeout: # [number; minimum: 1] Docker timeout - Timeout, in seconds, for

  # ------------------------------------------------------
```

```
  # -------------- if mode is custom ---------------

  filters: # [array] Container filters - Containers matching any of these will be
    - expr: # [string] Expression
  allContainers: # [boolean] All containers - Include stopped and paused containe
  perDevice: # [boolean] Per-device metrics - Generate separate metrics for each
  detail: # [boolean] Detailed metrics - Generate full container metrics
  dockerSocket: # [array of strings] Docker socket - Full paths for Docker's UNI
  dockerTimeout: # [number; minimum: 1] Docker timeout - Timeout, in seconds, for

  # ------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
persistence: # [object] persistence
  enable: # [boolean] Enable disk spooling - Spool metrics to disk for Cribl Edge

  # -------------- if enable is true ---------------

  timeWindow: # [string] Bucket time span - Time span for each file bucket
  maxDataSize: # [string] Data size limit - Maximum disk space allowed to be cons
  maxDataTime: # [string] Data age limit - Maximum amount of time to retain data
  compress: # [string] Data compression format
  destPath: # [string] Path location - Path to use to write metrics. Defaults to

  # ------------------------------------------------------

description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# -------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
```

```
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ---------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
system_state_input: # [object]
  type: # [string] Input Type
  interval: # [number; minimum: 1] Polling interval - Time, in seconds, between cor
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  collectors: # [object]
    hostsfile: # [object] Hosts File - Creates events based on entries collected fr
      enable: # [boolean] Enabled
    interfaces: # [object] Interfaces - Creates events for each of the host's netwo
      enable: # [boolean] Enabled
    disk: # [object] Disks & File Systems - Creates events for physical disks, part
      enable: # [boolean] Enabled
    metadata: # [object] Host Info - Creates events based on the host system's curr
      enable: # [boolean] Enabled
    routes: # [object] Routes - Creates events based on entries collected from the
      enable: # [boolean] Enabled
    dns: # [object] DNS - Creates events for DNS resolvers and search entries
      enable: # [boolean] Enabled
    user: # [object] Users & Groups - Creates events for local users and groups
      enable: # [boolean] Enabled
    firewall: # [object] Firewall - Creates events for Firewall rules entries
      enable: # [boolean] Enabled
    services: # [object] Services - Creates events from the list of services
      enable: # [boolean] Enabled
    ports: # [object] Listening Ports - Creates events from list of listening ports
      enable: # [boolean] Enabled
```

```
  loginUsers: # [object] Logged-In Users - Creates events from list of logged-in
    enable: # [boolean] Enabled
persistence: # [object]
  enable: # [boolean] Enable disk spooling - Spool metrics to disk for Cribl Edge

  # -------------- if enable is true ---------------

  timeWindow: # [string] Bucket time span - Time span for each file bucket
  maxDataSize: # [string] Data size limit - Maximum disk space allowed to be cons
  maxDataTime: # [string] Data age limit - Maximum amount of time to retain data
  compress: # [string] Data compression format
  destPath: # [string] Path location - Path to use to write metrics. Defaults to

  # ------------------------------------------------------

disableNativeModule: # [boolean] Use Windows Tools - Enable to use built-in tools
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# -------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data
```

```
  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
kube_metrics_input: # [object]
  type: # [string] Input Type
  interval: # [number; minimum: 1] Polling interval - Time, in seconds, between cor
  rules: # [array] Filter Rules - Add rules to decide which Kubernetes objects to g
    - filter: # [string] Filter Expression - JavaScript expression applied to Kuber
      description: # [string] Description - Optional description of this rule's pur
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  persistence: # [object] persistence
    enable: # [boolean] Enable disk spooling - Spool metrics on disk for Cribl Sear

    # -------------- if enable is true ---------------

    timeWindow: # [string] Bucket time span - Time span for each file bucket
    maxDataSize: # [string] Data size limit - Maximum disk space allowed to be cons
    maxDataTime: # [string] Data age limit - Maximum amount of time to retain data
    compress: # [string] Data compression format
    destPath: # [string] Path location - Path to use to write metrics. Defaults to

    # ------------------------------------------------------

  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------
```

```
  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
kube_logs_input: # [object]
  type: # [string] Input Type
  interval: # [number; minimum: 1] Polling interval - Time, in seconds, between che
  rules: # [array] Filter Rules - Add rules to decide which Pods to collect logs fr
    - filter: # [string] Filter Expression - JavaScript expression applied to Pod o
      description: # [string] Description - Optional description of this rule's pur
  timestamps: # [boolean] Enable timestamps - For use when containers do not emit a
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  persistence: # [object] Disk Spooling
    enable: # [boolean] Enable disk spooling - Spool events on disk for Cribl Edge

    # -------------- if enable is true ---------------

    timeWindow: # [string] Bucket time span - Time period for grouping spooled even
    maxDataSize: # [string] Data size limit - Maximum disk space that can be consum
    maxDataTime: # [string] Data age limit - Maximum amount of time to retain data
    compress: # [string] Compression - Data compression format. Default is gzip.

    # ------------------------------------------------------

  breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
  staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
  enableLoadBalancing: # [boolean] Enable load balancing - Load balance traffic acr
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------
```

```
  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
kube_events_input: # [object]
  type: # [string] Input Type
  rules: # [array] Filter Rules - Filtering on event fields
    - filter: # [string] Filter Expression - JavaScript expression applied to Kuber
      description: # [string] Description - Optional description of this rule's pur
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination
```

```
  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
windows_metrics_input: # [object]
  type: # [string] Input Type
  interval: # [number; minimum: 1] Polling interval - Time, in seconds, between cor
  host: # [object]
    mode: # [string]  - Select level of detail for host metrics

    # -------------- if mode is custom ---------------

    custom: # [object]
      system: # [object]
        mode: # [string]  - Select the level of details for system metrics

        # -------------- if mode is custom ---------------

        detail: # [boolean] Detailed metrics - Generate metrics for all system info

        # ------------------------------------------------------

      cpu: # [object]
        mode: # [string]  - Select the level of details for CPU metrics

        # -------------- if mode is custom ---------------

        perCpu: # [boolean] Per-CPU metrics - Generate metrics for each CPU
```

```
      detail: # [boolean] Detailed metrics - Generate metrics for all CPU states
      time: # [boolean] CPU time metrics - Generate raw, monotonic CPU time count

      # ------------------------------------------------------

    memory: # [object]
      mode: # [string]  - Select the level of details for memory metrics

      # -------------- if mode is custom ---------------

      detail: # [boolean] Detailed metrics - Generate metrics for all memory stat

      # ------------------------------------------------------

    network: # [object]
      mode: # [string]  - Select the level of details for network metrics

      # -------------- if mode is custom ---------------

      devices: # [array of strings] Interface filter - Network interfaces to incl
      perInterface: # [boolean] Per interface metrics - Generate separate metrics
      detail: # [boolean] Detailed metrics - Generate full network metrics

      # ------------------------------------------------------

    disk: # [object]
      mode: # [string]  - Select the level of details for disk metrics

      # -------------- if mode is custom ---------------

      volumes: # [array of strings] Volume filter - Windows volumes to include/ex
      perVolume: # [boolean] Per volume metrics - Generate separate metrics for e

      # ------------------------------------------------------


  # --------------------------------------------------------

process: # [object]
  sets: # [array] Process sets - Configure sets to collect process metrics
    - name: # [string] Set Name
      filter: # [string] Filter Expression
      includeChildren: # [boolean] Include Child Processes
```

```
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
persistence: # [object] persistence
  enable: # [boolean] Enable disk spooling - Spool metrics to disk for Cribl Edge

  # -------------- if enable is true ---------------

  timeWindow: # [string] Bucket time span - Time span for each file bucket
  maxDataSize: # [string] Data size limit - Maximum disk space allowed to be cons
  maxDataTime: # [string] Data age limit - Maximum amount of time to retain data
  compress: # [string] Data compression format
  destPath: # [string] Path location - Path to use to write metrics. Defaults to

  # -------------------------------------------------------

disableNativeModule: # [boolean] Use Windows Tools - Enable to use built-in tools
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# -------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data
```

```
  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
crowdstrike_input: # [object]
  type: # [string] Input Type
  queueName: # [string] Queue - The name, URL, or ARN of the SQS queue to read noti
  fileFilter: # [string] Filename filter - Regex matching file names to download ar
  awsAccountId: # [string] AWS account ID - SQS queue owner's AWS account ID. Leave
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key

  # ------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

  # ------------------------------------------------------

  awsSecretKey: # [string] Secret key
  region: # [string] Region - AWS Region where the S3 bucket and SQS queue are loca
  endpoint: # [string] Endpoint - S3 service endpoint. If empty, defaults to the AW
  signatureVersion: # [string] Signature version - Signature version to use for sig
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
  breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
  staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
  maxMessages: # [number; minimum: 1, maximum: 10] Message limit - The maximum numb
  visibilityTimeout: # [number; maximum: 43200] Visibility timeout seconds - After
  numReceivers: # [number; minimum: 1, maximum: 100] Number of receivers - How many
  socketTimeout: # [number; minimum: 1, maximum: 43200] Socket timeout - Socket ina
  skipOnError: # [boolean] Skip file on error - Skip files that trigger a processin
  includeSqsMetadata: # [boolean] Include notification metadata - Attach SQS notifi
  enableAssumeRole: # [boolean] Enable for Amazon S3 - Use Assume Role credentials
  assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
  assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
  durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
  enableSQSAssumeRole: # [boolean] Enable for Amazon SQS - Use Assume Role credenti
```

```
preprocess: # [object]
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  command: # [string] Command - Command to feed the data through (via stdin) and
  args: # [array of strings] Arguments - Arguments to be added to the custom comm

  # ---------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
checkpointing: # [object]
  enabled: # [boolean] Enable checkpointing - Resume processing files after an ir

  # -------------- if enabled is true ---------------

  retries: # [number; maximum: 100] Retries - The number of times to retry proces

  # ---------------------------------------------------------

pollTimeout: # [number; minimum: 1, maximum: 20] Poll timeout (secs) - How long t
encoding: # [string] Encoding - Character encoding to use when parsing ingested d
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ---------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
```

```
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
datadog_agent_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  maxActiveReq: # [number] Active request limit - Maximum number of active requests
  maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
  enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
  captureHeaders: # [boolean] Capture request headers - Add request headers to even
  activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
  requestTimeout: # [number] Request timeout (seconds) - How long to wait for an in
  socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
  keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
  enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
  ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
  ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
```

```
  extractMetrics: # [boolean] Extract metrics - Toggle to Yes to extract each incom
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  proxyMode: # [object]
    enabled: # [boolean] Forward API key validation requests - Toggle to Yes to ser

    # -------------- if enabled is true ---------------

    rejectUnauthorized: # [boolean] Reject unauthorized certificates - Whether to r

    # -------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
datagen_input: # [object]
```

```
  type: # [string] Input Type
  samples: # [array] Datagens
    - sample: # [string] Data Generator File Name
      eventsPerSec: # [number; minimum: 1] Events Per Second Per Worker Node - Maxi
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # --------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # --------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
http_raw_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array of strings] Auth tokens - Shared secrets to be provided by a
```

```
tls: # [object] TLS settings (server side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # -------------------------------------------------------

maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
allowedPaths: # [array of strings] Allowed URI paths - List of URI paths accepted
allowedMethods: # [array of strings] Allowed HTTP methods - List of HTTP methods
authTokensExt: # [array] Auth tokens - Shared secrets to be provided by any clien
  - token: # [string] Token - Shared secret to be provided by any client (Authori
    description: # [string] Description
    metadata: # [array] Fields - Fields to add to events referencing this token
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, e
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t
```

```
  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # --------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # --------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
kinesis_input: # [object]
  type: # [string] Input Type
  streamName: # [string] Stream name - Kinesis Data Stream to read data from
  serviceInterval: # [number; minimum: 1, maximum: 5] Service period - Time interva
  shardExpr: # [string] Shard selection expression - A JavaScript expression to be
  shardIteratorType: # [string] Shard iterator start - Location at which to start r
  payloadFormat: # [string] Record data format - Format of data inside the Kinesis
  getRecordsLimit: # [number; minimum: 5000, maximum: 10000] Records limit per call
  getRecordsLimitTotal: # [number; minimum: 20000] Total records limit - Maximum nu
  loadBalancingAlgorithm: # [string] Shard load balancing - The load-balancing algo
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key

  # --------------------------------------------------------
```

```
# -------------- if awsAuthenticationMethod is secret ---------------

awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

# -------------------------------------------------------

awsSecretKey: # [string] Secret key
region: # [string] [required] Region - Region where the Kinesis stream is located
endpoint: # [string] Endpoint - Kinesis stream service endpoint. If empty, defaul
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for Kinesis stream - Use Assume Role credent
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
verifyKPLCheckSums: # [boolean] Verify KPL checksums - Verify Kinesis Producer Li
avoidDuplicates: # [boolean] Avoid duplicate records - When resuming streaming fr
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# -------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
```

```
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
criblmetrics_input: # [object]
  type: # [string] Input Type
  prefix: # [string] Metric name prefix - A prefix that is applied to the metrics p
  fullFidelity: # [boolean] Full fidelity - Include granular metrics. Disabling thi
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data
```

```
  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
metrics_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. For IPv4 (all addresses), use the
  udpPort: # [number; maximum: 65535] UDP Port - Enter UDP port number to listen or
  tcpPort: # [number; maximum: 65535] TCP Port - Enter TCP port number to listen or
  maxBufferSize: # [number] Buffer size limit (events) - Maximum number of events t
  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
  enableProxyHeader: # [boolean] Enable proxy protocol - Enable if the connection i
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  udpSocketRxBufSize: # [number; minimum: 256, maximum: 4294967295] UDP socket buff
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination
```

```
  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
s3_input: # [object]
  type: # [string] Input Type
  queueName: # [string] Queue - The name, URL, or ARN of the SQS queue to read noti
  fileFilter: # [string] Filename filter - Regex matching file names to download ar
  awsAccountId: # [string] AWS account ID - SQS queue owner's AWS account ID. Leave
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key

  # ------------------------------------------------------



  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

  # ------------------------------------------------------

  awsSecretKey: # [string] Secret key
  region: # [string] Region - AWS Region where the S3 bucket and SQS queue are loca
  endpoint: # [string] Endpoint - S3 service endpoint. If empty, defaults to the AW
  signatureVersion: # [string] Signature version - Signature version to use for sig
```

```
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
maxMessages: # [number; minimum: 1, maximum: 10] Message limit - The maximum numb
visibilityTimeout: # [number; maximum: 43200] Visibility timeout seconds - After
numReceivers: # [number; minimum: 1, maximum: 100] Number of receivers - How many
socketTimeout: # [number; minimum: 1, maximum: 43200] Socket timeout - Socket ina
skipOnError: # [boolean] Skip file on error - Skip files that trigger a processir
includeSqsMetadata: # [boolean] Include notification metadata - Attach SQS notifi
enableAssumeRole: # [boolean] Enable for Amazon S3 - Use Assume Role credentials
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
enableSQSAssumeRole: # [boolean] Enable for Amazon SQS - Use Assume Role credenti
preprocess: # [object]
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  command: # [string] Command - Command to feed the data through (via stdin) and
  args: # [array of strings] Arguments - Arguments to be added to the custom comm

  # ---------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
parquetChunkSizeMB: # [number; minimum: 1, maximum: 100] Parquet chunk size limit
parquetChunkDownloadTimeout: # [number; minimum: 1, maximum: 3600] Parquet chunk
checkpointing: # [object]
  enabled: # [boolean] Enable checkpointing - Resume processing files after an ir

  # -------------- if enabled is true ---------------

  retries: # [number; maximum: 100] Retries - The number of times to retry proces

  # ---------------------------------------------------

pollTimeout: # [number; minimum: 1, maximum: 20] Poll timeout (secs) - How long t
encoding: # [string] Encoding - Character encoding to use when parsing ingested d
tagAfterProcessing: # [boolean] Tag after processing - Add a tag to processed S3
```

```
  # -------------- if tagAfterProcessing is true ---------------

  processedTagKey: # [string] Tag key - The key for the S3 object tag applied after
  processedTagValue: # [string] Tag value - The value for the S3 object tag applied

  # ------------------------------------------------------

  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
s3_inventory_input: # [object]
  queueName: # [string] Queue - The name, URL, or ARN of the SQS queue to read noti
  fileFilter: # [string] Filename filter - Regex matching file names to download ar
  awsAccountId: # [string] AWS account ID - SQS queue owner's AWS account ID. Leave
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me
```

```
# -------------- if awsAuthenticationMethod is manual ---------------

awsApiKey: # [string] Access key

# -------------------------------------------------------


# -------------- if awsAuthenticationMethod is secret ---------------

awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

# -------------------------------------------------------


awsSecretKey: # [string] Secret key
region: # [string] Region - AWS Region where the S3 bucket and SQS queue are loca
endpoint: # [string] Endpoint - S3 service endpoint. If empty, defaults to the AW
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
maxMessages: # [number; minimum: 1, maximum: 10] Message limit - The maximum numb
visibilityTimeout: # [number; maximum: 43200] Visibility timeout seconds - After
numReceivers: # [number; minimum: 1, maximum: 100] Number of receivers - How many
socketTimeout: # [number; minimum: 1, maximum: 43200] Socket timeout - Socket ina
skipOnError: # [boolean] Skip file on error - Skip files that trigger a processin
includeSqsMetadata: # [boolean] Include notification metadata - Attach SQS notifi
enableAssumeRole: # [boolean] Enable for Amazon S3 - Use Assume Role credentials
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
enableSQSAssumeRole: # [boolean] Enable for Amazon SQS - Use Assume Role credenti
preprocess: # [object]
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  command: # [string] Command - Command to feed the data through (via stdin) and
  args: # [array of strings] Arguments - Arguments to be added to the custom comm

  # -------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
```

```
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
parquetChunkSizeMB: # [number; minimum: 1, maximum: 100] Parquet chunk size limit
parquetChunkDownloadTimeout: # [number; minimum: 1, maximum: 3600] Parquet chunk
checkpointing: # [object]
  enabled: # [boolean] Enable checkpointing - Resume processing files after an ir

  # -------------- if enabled is true ---------------

  retries: # [number; maximum: 100] Retries - The number of times to retry proces

  # -------------------------------------------------------

pollTimeout: # [number; minimum: 1, maximum: 20] Poll timeout (secs) - How long t
type: # [string] Input Type
checksumSuffix: # [string] Checksum Suffix - Filename suffix of the manifest chec
maxManifestSizeKB: # [integer; minimum: 1] Manifest size limit (KB) - Maximum dow
validateInventoryFiles: # [boolean] Validate inventory files - If set to Yes, eac
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# -------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
```

```
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
snmp_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. For IPv4 (all addresses), use the
  port: # [number; maximum: 65535] [required] UDP port - UDP port to receive SNMP t
  snmpV3Auth: # [object] SNMPv3 authentication - Authentication parameters for SNMF
    v3AuthEnabled: # [boolean] Enabled

    # -------------- if v3AuthEnabled is true ---------------

    allowUnmatchedTrap: # [boolean] Allow unmatched traps - Pass through traps that
    v3Users: # [array] SNMP v3 users - User credentials for receiving v3 traps
      - name: # [string] V3 name
        authProtocol: # [string] Authentication protocol

    # ------------------------------------------------------

  maxBufferSize: # [number] Buffer size limit (events) - Maximum number of events t
  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  udpSocketRxBufSize: # [number; minimum: 256, maximum: 4294967295] UDP socket buff
  varbindsWithTypes: # [boolean] Include varbind types - If enabled, parses varbinc
  bestEffortParsing: # [boolean] Best effort parsing - If enabled, the parser will
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
```

```
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ---------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
open_telemetry_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  protocol: # [string] Protocol - Select whether to leverage gRPC or HTTP for OpenT

  # -------------- if protocol is http ---------------

  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ---------------------------------------------------------

  maxActiveReq: # [number] Active request limit - Maximum number of active requests
  maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
```

```
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (seconc
enableHealthCheck: # [boolean] Health check endpoint - Enable to expose the /crit
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should

# ------------------------------------------------------


# -------------- if protocol is grpc ---------------

maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
tls: # [object] TLS settings (server side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------


# ------------------------------------------------------

extractSpans: # [boolean] Extract spans - Enable to extract each incoming span to
extractMetrics: # [boolean] Extract metrics - Enable to extract each incoming Gau
otlpVersion: # [string] OTLP version - The version of OTLP Protobuf definitions t

# -------------- if otlpVersion is 1.3.1 ---------------

extractLogs: # [boolean] Extract logs - Enable to extract each incoming log recor

# ------------------------------------------------------

authType: # [string] Authentication type - OpenTelemetry authentication type
```

```
# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# ---------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# ---------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ---------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# ---------------------------------------------------


# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth l
  - name: # [string] Name - OAuth header name
```

```
    value: # [string] Value - OAuth header value

  # ------------------------------------------------------

  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
model_driven_telemetry_input: # [object]
  type: # [string] Input Type
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled
```

```
  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # --------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
shutdownTimeoutMs: # [number; minimum: 1] Shutdown timeout - Time in milliseconds
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# --------------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data
```

```
  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
sqs_input: # [object]
  type: # [string] Input Type
  queueName: # [string] Queue - The name, URL, or ARN of the SQS queue to read ever
  queueType: # [string] [required] Queue type - The queue type used (or created)

  # -------------- if queueType is standard ---------------

  numReceivers: # [number; minimum: 1, maximum: 100] Number of receivers - How many

  # ------------------------------------------------------

  awsAccountId: # [string] AWS account ID - SQS queue owner's AWS account ID. Leave
  createQueue: # [boolean] Create queue - Create queue if it does not exist
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key

  # ------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

  # ------------------------------------------------------

  awsSecretKey: # [string] Secret key
  region: # [string] Region - AWS Region where the SQS queue is located. Required,
  endpoint: # [string] Endpoint - SQS service endpoint. If empty, defaults to the A
  signatureVersion: # [string] Signature version - Signature version to use for sig
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
  enableAssumeRole: # [boolean] Enable for SQS - Use Assume Role credentials to acc
  assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
  assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
  durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
  maxMessages: # [number; minimum: 1, maximum: 10] Message limit - The maximum numb
```

```
  visibilityTimeout: # [number; maximum: 43200] Visibility Timeout Seconds - After
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  pollTimeout: # [number; minimum: 1, maximum: 20] Poll timeout (secs) - How long t
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
syslog_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. For IPv4 (all addresses), use the
  udpPort: # [number; maximum: 65535] UDP port - Enter UDP port number to listen or
  tcpPort: # [number; maximum: 65535] TCP port - Enter TCP port number to listen or
  maxBufferSize: # [number] Buffer size limit (events) - Maximum number of events t
  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
```

```
timestampTimezone: # [string] Default timezone - Timezone to assign to timestamps
singleMsgUdpPackets: # [boolean] Single msg per UDP - Treat UDP packet data recei
enableProxyHeader: # [boolean] Enable proxy protocol - Enable if the connection i

# -------------- if enableProxyHeader is true ---------------

enableEnhancedProxyHeaderParsing: # [boolean] Enable enhanced TLS handshake for p

# ------------------------------------------------------

keepFieldsList: # [array of strings] Fields to keep - Wildcard list of fields to
octetCounting: # [boolean] Octet count framing - Enable if incoming messages use
inferFraming: # [boolean] Infer Syslog framing - Enable if we should infer the sy
strictlyInferOctetCounting: # [boolean] Strictly infer octet count framing - Enab
allowNonStandardAppName: # [boolean] Allow non-standard app name - Enable if RFC
maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
socketIdleTimeout: # [number] TCP socket idle timeout (seconds) - How long Cribl
socketEndingMaxWait: # [number] TCP forced socket termination timeout (seconds) -
socketMaxLifespan: # [number] TCP Socket max lifespan (seconds) - The maximum dur
tls: # [object] TLS settings (server side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
udpSocketRxBufSize: # [number; minimum: 256, maximum: 4294967295] UDP socket buff
enableLoadBalancing: # [boolean] Enable TCP load balancing - Load balance traffic
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t
```

```
  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
file_input: # [object]
  type: # [string] Input Type
  mode: # [string]  - Choose how to discover files to monitor

  # -------------- if mode is manual ---------------

  path: # [string] Search path - Directory path to search for files. Environment va
  depth: # [number] Max depth - Set how many subdirectories deep to search. Use 0 t
  suppressMissingPathErrors: # [boolean] Suppress errors when search path does not
  deleteFiles: # [boolean] Delete files - Delete files after they have been collect

  # -------------------------------------------------------

  interval: # [number; minimum: 1] Polling interval - Time, in seconds, between sca
  filenames: # [array of strings] Filename allowlist - The full path of discovered
  tailOnly: # [boolean] Collect from end - Read only new entries at the end of all
  idleTimeout: # [number; minimum: 1] Idle timeout - Time, in seconds, before an ic
```

```
maxAgeDur: # [string] Age duration limit - The maximum age of files to monitor. F
checkFileModTime: # [boolean] Check file modification times - Skip files with mod
forceText: # [boolean] Force text format - Forces files containing binary data to

# -------------- if forceText is false ---------------

includeUnidentifiableBinary: # [boolean] Enable binary files - Stream binary file

# ----------------------------------------------------

hashLen: # [number; minimum: 1] Hash length - Length of file header bytes to use
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
description: # [string] Description
disabled: # [boolean] Disabled
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination

# ----------------------------------------------------

environment: # [string] Environment - Optionally, enable this config only on a sp
pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

# -------------- if pqEnabled is true ---------------

pq: # [object]
  mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
  maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
  commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
  maxFileSize: # [string] File size limit - The maximum size to store in each que
  maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  path: # [string] Queue file path - The location for the persistent queue files.
  compress: # [string] Compression - Codec to use to compress the persisted data
```

```
  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
tcp_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    certificateName: # [string] Certificate - The name of the predefined certificat
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate path - Path on server containing certificates
    caPath: # [string] CA certificate path - Path on server containing CA certifica
    requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
  maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
  socketIdleTimeout: # [number] Socket idle timeout (seconds) - How long Cribl Stre
  socketEndingMaxWait: # [number] Forced socket termination timeout (seconds) - How
  socketMaxLifespan: # [number] Socket max lifespan (seconds) - The maximum duratio
  enableProxyHeader: # [boolean] Enable proxy protocol - Enable if the connection i
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
  staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
  enableHeader: # [boolean] Enable header - Client will pass the header record with

  # -------------- if enableHeader is true ---------------

  authType: # [string] Authentication method - Select Manual to enter an auth token

  # ------------------------------------------------------
```

```
  preprocess: # [object]
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    command: # [string] Command - Command to feed the data through (via stdin) and
    args: # [array of strings] Arguments - Arguments to be added to the custom comm

    # ----------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -----------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -----------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
appscope_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
```

```
maxActiveCxn: # [number] Active connection limit - Maximum number of active conne
socketIdleTimeout: # [number] Socket idle timeout (seconds) - How long Cribl Stre
socketEndingMaxWait: # [number] Forced socket termination timeout (seconds) - How
socketMaxLifespan: # [number] Socket max lifespan (seconds) - The maximum duratic
enableProxyHeader: # [boolean] Enable proxy protocol - Enable if the connection i
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
enableUnixPath: # [boolean] UNIX domain socket - Toggle to Yes to specify a file-

# -------------- if enableUnixPath is false ---------------

host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
port: # [number; maximum: 65535] Port - Port to listen on
tls: # [object] TLS settings (server side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # -------------------------------------------------------


# -------------------------------------------------------


# -------------- if enableUnixPath is true ---------------

unixSocketPath: # [string] UNIX socket path - Path to the UNIX domain socket to l
unixSocketPerms: # [string,number] UNIX socket permissions - Permissions to set f

# -------------------------------------------------------
```

```
filter: # [object]
  allow: # [array] Rules - Specify processes that AppScope should be loaded into,
    - procname: # [string] Process name - Specify the name of a process or family
      arg: # [string] Process argument - Specify a string to substring-match agai
      config: # [string] AppScope config - Choose a config to apply to processes
  transportURL: # [string] Transport override - To override the UNIX domain socke
persistence: # [object] Persistence
  enable: # [boolean] Enable disk spooling - Spool events and metrics on disk for

  # -------------- if enable is true ---------------

  timeWindow: # [string] Bucket time span - Time span for each file bucket
  maxDataSize: # [string] Data size limit - Maximum disk space allowed to be cons
  maxDataTime: # [string] Data age limit - Maximum amount of time to retain data
  compress: # [string] Data compression format
  destPath: # [string] Path location - Path to use to write metrics. Defaults to

  # ------------------------------------------------------

authType: # [string] Authentication method - Select Manual to enter an auth token
authToken: # [string] Auth token - Shared secret to be provided by any client (ir

# -------------- if authType is manual ---------------


# ------------------------------------------------------

textSecret: # [string] Auth token (text secret) - Select or create a stored text

# -------------- if authType is secret ---------------


# ------------------------------------------------------

description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------

connections: # [array] Use QuickConnect - Direct connections to Destinations, and
  - pipeline: # [string] Pipeline or Pack
    output: # [string] Destination
```

```
  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
wef_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authMethod: # [string] Authentication method - How to authenticate incoming clien

  # -------------- if authMethod is clientCert ---------------

  tls: # [object] mTLS settings
    disabled: # [boolean] Disabled - Enable TLS
    rejectUnauthorized: # [boolean] Validate client certs - Required for WEF certif
    requestCert: # [boolean] Authenticate client - Required for WEF certificate aut
    certificateName: # [string] Certificate - Name of the predefined certificate
    privKeyPath: # [string] Private key path - Path on server containing the privat
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] [required] Certificate path - Path on server containing ce
    caPath: # [string] [required] CA certificate path - Server path containing CA c
    commonNameRegex: # [string] Common name - Regex matching allowable common names
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version
    ocspCheck: # [boolean] Verify certificate via OCSP - Enable OCSP check of certi
```

```
  # -------------- if ocspCheck is true ---------------

  ocspCheckFailClose: # [boolean] Strict validation - If enabled, checks will fai

  # ----------------------------------------------------------

caFingerprint: # [string] CA fingerprint override - SHA1 fingerprint expected by
allowMachineIdMismatch: # [boolean] Allow MachineID mismatch - Allow events to be
logFingerprintMismatch: # [boolean] Log CA fingerprint mismatch warning - Log a w

# ----------------------------------------------------------


# -------------- if authMethod is kerberos ---------------

keytab: # [string] Keytab location - Path to the keytab file containing the servi
principal: # [string] Service principal name - Kerberos principal used for auther

# ----------------------------------------------------------

maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Preserve the client's origir
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
subscriptions: # [array] [required] Subscriptions - Subscriptions to events on fo
  - subscriptionName: # [string] Subscription name
    version: # [string] Version - Version UUID for this subscription. If any subs
    contentFormat: # [string] Format - Content format in which the endpoint shoul
    heartbeatInterval: # [number; minimum: 1] Heartbeat - Maximum time (in second
    batchTimeout: # [number] Batch timeout - Interval (in seconds) over which the
    readExistingEvents: # [boolean] Read existing events - Newly subscribed endpo
    sendBookmarks: # [boolean] Use bookmarks - Keep track of which events have be
    compress: # [boolean] Compression - Receive compressed events from the source
    targets: # [array of strings] Targets - The DNS names of the endpoints that s
    locale: # [string] Locale - The RFC-3066 locale the Windows clients should us
    querySelector: # [string] Query builder mode
    metadata: # [array] Fields - Fields to add to events ingested under this subs
    - name: # [string] Field Name
```

```
      value: # [string] Value - JavaScript expression to compute field's value, e
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # --------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # --------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
win_event_logs_input: # [object]
  type: # [string] Input Type
  logNames: # [array of strings] Event logs - Enter the event logs to collect. Run
  readMode: # [string] Read mode - Read all stored and future event logs, or only f
  eventFormat: # [string] Event format - Format of individual events
  disableNativeModule: # [boolean] Use Windows Tools - Enable to use built-in tools
  interval: # [number; minimum: 1] Polling interval - Time, in seconds, between che
  batchSize: # [number; minimum: 1] Batch size - The maximum number of events to re
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
```

```
    value: # [string] Value - JavaScript expression to compute field's value, enc
  maxEventBytes: # [number; minimum: 1, maximum: 134217728] Max event bytes - The m
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # --------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # --------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
raw_udp_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. For IPv4 (all addresses), use the
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  maxBufferSize: # [number] Buffer size limit (events) - Maximum number of events t
  ipWhitelistRegex: # [string] IP allowlist regex - Regex matching IP addresses tha
  singleMsgUdpPackets: # [boolean] Single msg per UDP - If true, each UDP packet is
  ingestRawBytes: # [boolean] Ingest raw bytes - If true, a __rawBytes field will b
  udpSocketRxBufSize: # [number; minimum: 256, maximum: 4294967295] UDP socket buff
  metadata: # [array] Fields - Fields to add to events from this input
```

```
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # --------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # --------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
journal_files_input: # [object]
  type: # [string] Input Type
  path: # [string] Search path - Directory path to search for journals. Environment
  interval: # [number; minimum: 1] Polling interval - Time, in seconds, between sca
  journals: # [array of strings] [required] Journal allowlist - The full path of di
  rules: # [array] Filter Rules - Add rules to decide which journal objects to allo
    - filter: # [string] Filter Expression - JavaScript expression applied to Journ
      description: # [string] Description - Optional description of this rule's pur
  currentBoot: # [boolean] Current boot only - Skip log messages that are not part
  maxAgeDur: # [string] Age duration limit - The maximum log message age, in durati
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
```

```
    value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # --------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # --------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
wiz_input: # [object]
  type: # [string] Input Type
  endpoint: # [string] GraphQL endpoint - The Wiz GraphQL API endpoint. Example: ht
  authUrl: # [string] [required] Authentication URL - The authentication URL to ger
  authAudienceOverride: # [string] Authentication audience - The audience to use wh
  clientId: # [string] [required] Client ID - The client ID of the Wiz application
  contentConfig: # [array] [required] Content types
    - contentType: # [string] Content name - The name of the Wiz query
      contentDescription: # [string] Description
      enabled: # [boolean] Enable content
  requestTimeout: # [number; maximum: 2400] Request timeout (seconds) - HTTP reques
  keepAliveTime: # [number; minimum: 10] Keep alive time (seconds) - How often work
```

```
maxMissedKeepAlives: # [number; minimum: 2] Worker timeout (periods) - The numbe
ttl: # [string] Time to live - Time to keep the job's artifacts on disk after jo
ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled,
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, en
retryRules: # [object]
  type: # [string] Retry type - The algorithm to use when performing HTTP retrie

  # -------------- if type is static ---------------

  codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigger
  interval: # [number; maximum: 20000] Wait (ms) - Time interval between failed r
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to re
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After hea
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retr
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a c

  # -------------------------------------------------------


  # -------------- if type is backoff ---------------

  interval: # [number; maximum: 20000] Initial retry interval (ms) - Time interva
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to re
  multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for e
  codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigger
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After hea
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single retr
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a c

  # -------------------------------------------------------

authType: # [string] Authentication method - Enter client secret directly, or sel
clientSecret: # [string] Client secret - The client secret of the Wiz applicatio

# -------------- if authType is manual ---------------


# -------------------------------------------------------

textSecret: # [string] Client Secret (text secret) - Select or create a stored te
```

```
  # -------------- if authType is secret ---------------


  # -------------------------------------------------------

  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
wiz_webhook_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array of strings] Auth tokens - Shared secrets to be provided by a
  tls: # [object] TLS settings (server side)
    disabled: # [boolean] Disabled
```

```
  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # -------------------------------------------------------

maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an ir
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
enableHealthCheck: # [boolean] Health check endpoint - Expose the /cribl_health e
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
allowedPaths: # [array of strings] Allowed URI paths - List of URI paths accepted
allowedMethods: # [array of strings] Allowed HTTP methods - List of HTTP methods
authTokensExt: # [array] Auth tokens - Shared secrets to be provided by any clien
  - token: # [string] Token - Shared secret to be provided by any client (Authori
    description: # [string] Description
    metadata: # [array] Fields - Fields to add to events referencing this token
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, e
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data from this Source before
sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

# -------------- if sendToRoutes is false ---------------
```

```
  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # ------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
netflow_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. For IPv4 (all addresses), use the
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  enablePassThrough: # [boolean] Enable pass-through - Allow forwarding of events t
  ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
  ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
  udpSocketRxBufSize: # [number; minimum: 256, maximum: 4294967295] UDP socket buff
  templateCacheMinutes: # [number; minimum: 1, maximum: 3600] Template cache minute
  v5Enabled: # [boolean] V5 - Accept messages in Netflow V5 format.
  v9Enabled: # [boolean] V9 - Accept messages in Netflow V9 format.
  ipfixEnabled: # [boolean] IPFIX - Accept messages in IPFIX format.
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, enc
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t
```

```
  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
security_lake_input: # [object]
  type: # [string] Input Type
  queueName: # [string] Queue - The name, URL, or ARN of the SQS queue to read noti
  fileFilter: # [string] Filename filter - Regex matching file names to download ar
  awsAccountId: # [string] AWS account ID - SQS queue owner's AWS account ID. Leave
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key

  # -------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref
```

```
# ------------------------------------------------------

awsSecretKey: # [string] Secret key
region: # [string] Region - AWS Region where the S3 bucket and SQS queue are loca
endpoint: # [string] Endpoint - S3 service endpoint. If empty, defaults to the AW
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
maxMessages: # [number; minimum: 1, maximum: 10] Message limit - The maximum numb
visibilityTimeout: # [number; maximum: 43200] Visibility timeout seconds - After
numReceivers: # [number; minimum: 1, maximum: 100] Number of receivers - How many
socketTimeout: # [number; minimum: 1, maximum: 43200] Socket timeout - Socket ina
skipOnError: # [boolean] Skip file on error - Skip files that trigger a processin
includeSqsMetadata: # [boolean] Include notification metadata - Attach SQS notifi
enableAssumeRole: # [boolean] Enable for Amazon S3 - Use Assume Role credentials
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
enableSQSAssumeRole: # [boolean] Enable for Amazon SQS - Use Assume Role credenti
preprocess: # [object]
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  command: # [string] Command - Command to feed the data through (via stdin) and
  args: # [array of strings] Arguments - Arguments to be added to the custom comm

  # ------------------------------------------------------

metadata: # [array] Fields - Fields to add to events from this input
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
parquetChunkSizeMB: # [number; minimum: 1, maximum: 100] Parquet chunk size limit
parquetChunkDownloadTimeout: # [number; minimum: 1, maximum: 3600] Parquet chunk
checkpointing: # [object]
  enabled: # [boolean] Enable checkpointing - Resume processing files after an in

  # -------------- if enabled is true ---------------

  retries: # [number; maximum: 100] Retries - The number of times to retry proces
```

```
    # ------------------------------------------------------

  pollTimeout: # [number; minimum: 1, maximum: 20] Poll timeout (secs) - How long t
  encoding: # [string] Encoding - Character encoding to use when parsing ingested d
  description: # [string] Description
  disabled: # [boolean] Disabled
  pipeline: # [string] Pipeline - Pipeline to process data from this Source before
  sendToRoutes: # [boolean]  - Select whether to send data to Routes, or directly t

  # -------------- if sendToRoutes is false ---------------

  connections: # [array] Use QuickConnect - Direct connections to Destinations, and
    - pipeline: # [string] Pipeline or Pack
      output: # [string] Destination

  # -------------------------------------------------------

  environment: # [string] Environment - Optionally, enable this config only on a sp
  pqEnabled: # [boolean] Enable persistent queue - Use a disk queue to minimize dat

  # -------------- if pqEnabled is true ---------------

  pq: # [object]
    mode: # [string] Mode - With Smart mode, PQ will write events to the filesystem
    maxBufferSize: # [number; minimum: 42] Buffer size limit - The maximum number o
    commitFrequency: # [number; minimum: 1] Commit frequency - The number of events
    maxFileSize: # [string] File size limit - The maximum size to store in each que
    maxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    path: # [string] Queue file path - The location for the persistent queue files.
    compress: # [string] Compression - Codec to use to compress the persisted data

  # -------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
zscaler_hec_input: # [object]
  type: # [string] Input Type
  disabled: # [boolean] Disabled
  host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all addresses
  port: # [number; maximum: 65535] [required] Port - Port to listen on
  authTokens: # [array] Auth tokens - Shared secrets to be provided by any client (
    - authType: # [string] Authentication method - Select Manual to enter an auth t
      enabled: # [boolean] Enable token
```

```
    description: # [string] Description
    allowedIndexesAtToken: # [array of strings] Allowed indexes - Enter the value
    metadata: # [array] Fields - Fields to add to events referencing this token
    - name: # [string] Field Name
      value: # [string] Value - JavaScript expression to compute field's value, e
tls: # [object] TLS settings (server side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  certificateName: # [string] Certificate - The name of the predefined certificat
  privKeyPath: # [string] Private key path - Path on server containing the privat
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  certPath: # [string] Certificate path - Path on server containing certificates
  caPath: # [string] CA certificate path - Path on server containing CA certifica
  requestCert: # [boolean] Authenticate client (mutual auth) - Require clients to
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------

maxActiveReq: # [number] Active request limit - Maximum number of active requests
maxRequestsPerSocket: # [integer] Requests-per-socket limit - Maximum number of r
enableProxyHeader: # [boolean] Show originating IP - Extract the client IP and po
captureHeaders: # [boolean] Capture request headers - Add request headers to ever
activityLogSampleRate: # [number; minimum: 1] Activity log sample rate - How ofte
requestTimeout: # [number] Request timeout (seconds) - How long to wait for an in
socketTimeout: # [number] Socket timeout (seconds) - How long Cribl Stream should
keepAliveTimeout: # [number; minimum: 1, maximum: 600] Keep-alive timeout (second
ipAllowlistRegex: # [string] IP allowlist regex - Messages from matched IP addres
ipDenylistRegex: # [string] IP denylist regex - Messages from matched IP addresse
hecAPI: # [string] [required] HEC endpoint - Absolute path on which to listen for
metadata: # [array] Fields - Fields to add to every event. May be overridden by f
  - name: # [string] Field Name
    value: # [string] Value - JavaScript expression to compute field's value, enc
allowedIndexes: # [array of strings] Allowed indexes - List values allowed in HEC
hecAcks: # [boolean] Zscaler HEC Acks - Whether to enable Zscaler HEC acknowledge
accessControlAllowOrigin: # [array of strings] CORS allowed origins - Optionally,
accessControlAllowHeaders: # [array of strings] CORS allowed headers - Optionally
emitTokenMetrics: # [boolean] Emit per-token request metrics - Enable to emit per
```

10.5.6. Instance.Yml

Instance configuration is located under `$CRIBL_HOME/local/_system/instance.yml` (`C:\ProgramData\Cribl\local\system\instance.yml` for Cribl Edge on Windows).

$CRIBL_HOME/local/_system/instance.yml or C:\ProgramData\Cribl\local_system.yml for Cribl Edge on Windows

```
distributed:
  # Mode - Choose one.
  # One of: single | worker | edge | managed-edge | master | outpost | search-supervi
  # [string; default: single; required]
  mode:
  # Deployment ID - Deployment ID for reporting telemetry on multiple deployments.
  # [string]
  reportedDeploymentId:
  # Default Worker Group - Worker Group to report to leader
  # [string; default: default]
  group:
  # Tags - Tag values to report to leader
  tags:
  # Env Var Regex - Regex to select environment variables to report to leader
  # [string; default: /^CRIBL_/]
  envRegex:
  master:
    # Address - Address to bind on. Defaults to 0.0.0.0 (all addresses).
    # [string; default: 0.0.0.0; required]
    host:
    # Port - Port to listen on
    # [number; max: 65535; default: 4200; required]
    port:
    # Forward to Leader API - Respond to API requests from the distributed port. True
    # [boolean; default: true]
    forwardToLeaderApi:
    # Auth token - Authentication token for Worker Nodes to communicate with Leader.
    # [string; default: criblmaster]
    authToken:
    # IP allowlist regex - Regex matching IP addresses that are allowed to establish
    # [string; default: /.*/]
    ipWhitelistRegex:
    # Active connection limit - Maximum number of active connections allowed from Wor
    # Use 0 for unlimited.
    # [number; min: 0; default: 0]
    maxActiveCxn:
    # Helper processes socket dir - Directory to hold sockets for inter-process commu
    # (IPC) between Leader and processes like Config Helpers and services. Defaults t
    # system's temp directory.
    # [string]
    configHelperSocketDir:
    # Resiliency - Enable or disable failover.
```

```
# One of: none | failover
# [string; default: none]
resiliency:
configBundles:
  # S3 Bundle Bucket URL - Bucket to use for remote bundle storage, in s3://${buc
  # [string]
  remoteUrl:
failover:
  # Failover volume - NFS (network file system) directory to support Leader failc
  # [string]
  volume:
  # Lease refresh period - How often the primary leader will refresh its hold on
  # file
  # [string; default: 5s]
  period:
  # Missed refresh limit - How many Lease refresh periods that can be missed befc
  # standby nodes attempt to elect themselves to primary
  # [number; default: 3]
  missedHBLimit:
tls:
  # Disabled
  # [boolean; default: true; required]
  disabled:
  # Certificate name - The name of the predefined certificate
  # [string]
  certificateName:
  # Private key path - Path on server in which to find the private key to use. PE
  # reference $ENV_VARS.
  # [string; required]
  privKeyPath:
  # Passphrase - Passphrase to use to decrypt private key
  # [string]
  passphrase:
  # Certificate path - Path on server in which to find certificates to use. PEM f
  # reference $ENV_VARS.
  # [string; required]
  certPath:
  # CA certificate path - Path on server where to find CA certificates to use. PE
  # reference $ENV_VARS.
  # [string]
  caPath:
  # Minimum TLS version - Minimum TLS version. Defaults to TLS 1.2.
  # One of: TLSv1 | TLSv1.1 | TLSv1.2 | TLSv1.3
```

```
  # [string]
  minVersion:
  # Maximum TLS version - Maximum TLS version. Defaults to TLS 1.3.
  # One of: TLSv1 | TLSv1.1 | TLSv1.2 | TLSv1.3
  # [string]
  maxVersion:
  # Default cipher list - Default suite of enabled and disabled TLS ciphers. Defa
  # ECDHE-RSA-AES128-GCM-SHA256:
  # ECDHE-ECDSA-AES128-GCM-SHA256:
  # ECDHE-RSA-AES256-GCM-SHA384:
  # ECDHE-ECDSA-AES256-GCM-SHA384:
  # DHE-RSA-AES128-GCM-SHA256:
  # ECDHE-RSA-AES128-SHA256:
  # DHE-RSA-AES128-SHA256:
  # ECDHE-RSA-AES256-SHA384:
  # DHE-RSA-AES256-SHA384:
  # ECDHE-RSA-AES256-SHA256:
  # DHE-RSA-AES256-SHA256:
  # HIGH:
  # !aNULL:
  # !eNULL:
  # !EXPORT:
  # !DES:
  # !RC4:
  # !MD5:
  # !PSK:
  # !SRP:
  # !CAMELLIA
  # [string]
  defaultCipherList:
  # ECDH curve - The curve name, or a colon-separated list of curve NIDs or names
  # ECDH key agreement. For example: 'P-521:P-384:P-256'. Defaults to 'auto'.
  # [string; default: auto]
  defaultEcdhCurve:
  # Validate server certs - Validate server certificates globally. Disable to all
  # self-signed certificates.
  # [boolean; default: true]
  rejectUnauthorized:
# Disable SNI-based connection routing - This is an advanced setting - do not tou
# under supervision of Cribl Support. This affects how connections are routed wit
# Control Plane. Changing this setting could affect the scalability of your syste
# [boolean; default: false]
disableSNIRouting:
```

```
# Compression - Codec to use to compress the data before sending
# One of: none | gzip
# [string; default: gzip]
compression:
# Connection timeout (ms) - Amount of time (milliseconds) to wait before assuming
# connection has failed
# [number; min: 0; default: 10000]
connectionTimeout:
# Write timeout (ms) - Amount of time (milliseconds) to wait for a write to compl
# assuming connection has failed
# [number; min: 0; default: 60000]
writeTimeout:
# Protocol
# One of: tcp | http2
# [string; default: tcp]
protocol:
proxy:
  # Disabled - Whether to proxy Leader comms through SOCKS
  # [boolean; default: true; required]
  disabled:
  # User ID - Username for SOCKS proxy authentication
  # [string]
  userId:
  # Password - Password for SOCKS proxy authentication
  # [string]
  password:
  # Port - SOCKS proxy port
  # [number; default: 1080]
```

10.5.7. IOMETRICS.YML

`iometrics.yml` is a configuration file that defines the metrics levels for individual Sources and Destinations in Cribl Stream. It allows you to control the granularity of metrics emitted by each integration. These correspond to the **Metrics Levels** settings of a Worker Group.

iometrics.yml

```
# Default metrics level - Default level for I/O metrics collection if not otherwise s
# One of: minimal | basic | detailed | comprehensive
# [string]
DEFAULT:
# Input/Output specific metrics levels - Maps specific input/output IDs to their metr
# One of: minimal | basic | detailed | comprehensive
<metric>:
```

Copy

Example `iometrics.yml` configuration:

Example iometrics.yml configuration

```
DEFAULT:
  level: detailed
input:in_splunk_tcp:
  level: minimal
input:in_syslog:
  level: detailed
output:syslog_loopback_dest_udp:
  level: detailed
input:eh_in:
  level: basic
input:in_cribl_http:
  level: minimal
```

Copy

10.5.8. JOBS.YML

`jobs.yml` maintains parameters for configured Collectors, corresponding to those listed on the **Collectors** page. Each collection job is listed according to the pattern shown below.

jobs.yml

```
collection_job: # [object]
  workerAffinity: # [boolean] Worker affinity - If enabled, tasks are created and run
  collector: # [object]
    type: # [string] Collector type - The type of collector to run

    # -------------- if type is azure_blob ---------------

    conf: # [object] [required]
      type: # [string]
      outputName: # [string] Auto-populate from - An optional predefined Destination
      authType: # [string] Authentication method - Enter authentication data directly

      # -------------- if authType is manual ---------------

      connectionString: # [string] Connection string - Enter your Azure storage accou

      # -----------------------------------------------------


      # -------------- if authType is secret ---------------

      textSecret: # [string] Connection string (text secret) - Text secret

      # -----------------------------------------------------


      # -------------- if authType is clientSecret ---------------

      storageAccountName: # [string] Storage account name - The name of your Azure st
      tenantId: # [string] Tenant ID - The service principal's tenant ID
      clientId: # [string] Client ID - The service principal's client ID
      clientTextSecret: # [string] Client secret (text secret) - Text secret containi
      endpointSuffix: # [string] Endpoint suffix - The endpoint suffix for the servic
      azureCloud: # [string] Azure Cloud - The Azure cloud to use. Defaults to Azure

      # -----------------------------------------------------


      # -------------- if authType is clientCert ---------------

      storageAccountName: # [string] Storage account name - The name of your Azure st
      tenantId: # [string] Tenant ID - The service principal's tenant ID
```

```
  clientId: # [string] Client ID - The service principal's client ID
  certificate: # [object]
    certificateName: # [string] Certificate - The certificate you registered as c
  azureCloud: # [string] Azure Cloud - The Azure cloud to use. Defaults to Azure
  endpointSuffix: # [string] Endpoint suffix - The endpoint suffix for the servic

  # ------------------------------------------------------

  containerName: # [string] Container name - Container to collect from. This valu
  path: # [string] Path - The directory from which to collect data. Templating is
  extractors: # [array] Path extractors - Extractors allow use of template tokens
    - key: # [string] Token - A token from the template path, such as epoch
      expression: # [string] Extractor Expression - A JavaScript expression that
  recurse: # [boolean] Recursive - Recurse through subdirectories
  includeMetadata: # [boolean] Include metadata - Include Azure Blob metadata in
  includeTags: # [boolean] Include tags - Include Azure Blob tags in collected ev
  maxBatchSize: # [number; minimum: 1] Batch size limit - Maximum number of metad
  parquetChunkSizeMB: # [number; minimum: 1, maximum: 100] Parquet chunk size lim
  parquetChunkDownloadTimeout: # [number; minimum: 1, maximum: 3600] Parquet chun

# ------------------------------------------------------


# -------------- if type is cribl_lake ---------------

conf: # [object] [required]
  type: # [string]
  dataset: # [string] Lake dataset - Lake dataset to collect data from.

# ------------------------------------------------------


# -------------- if type is database ---------------

conf: # [object] [required]
  type: # [string]
  connectionId: # [string] Connection - Select an existing Connection, or go to K
  query: # [string] [required] SQL Query - An expression that resolves to the que
  queryValidationEnabled: # [boolean] Validate Query - Enforces a basic query val
  defaultBreakers: # [string] Hidden Default Breakers
  __scheduling: # [object]
    stateTracking: # [object]
      enabled: # [boolean] Enabled - Enable tracking of collection progress betwe
```

```
      # -------------- if enabled is true ---------------

      trackingColumn: # [string] Tracking Column - A numeric database column whos

      # ------------------------------------------------------


# ------------------------------------------------------


# -------------- if type is filesystem ---------------

conf: # [object] [required]
  type: # [string]
  outputName: # [string] Auto-populate from - Select a predefined configuration (
  path: # [string] Directory - The directory from which to collect data. Templati
  extractors: # [array] Path extractors - Allows using template tokens as context
    - key: # [string] Token - A token from the template directory, such as epoch
      expression: # [string] Extractor expression - JavaScript expression that re
  recurse: # [boolean] Recursive - Recurse through subdirectories
  maxBatchSize: # [number; minimum: 1] Batch size limit (files) - Maximum number

# ------------------------------------------------------


# -------------- if type is google_cloud_storage ---------------

conf: # [object] [required]
  type: # [string]
  outputName: # [string] Auto-populate from - Name of the predefined Destination
  bucket: # [string] Bucket name - Name of the bucket to collect from. This value
  path: # [string] Path - The directory from which to collect data. Templating is
  extractors: # [array] Path extractors - Allows using template tokens as context
    - key: # [string] Token - A token from the template path, such as epoch
      expression: # [string] Extractor Expression - JavaScript expression that re
  endpoint: # [string] Endpoint - Google Cloud Storage service endpoint. If empty
  disableTimeFilter: # [boolean] Disable time filter - Used to disable Collector
  recurse: # [boolean] Recursive - Recurse through subdirectories
  maxBatchSize: # [number; minimum: 1] Batch size limit (objects) - Maximum numbe
  authType: # [string] Authentication method - Enter account credentials manually

  # -------------- if authType is manual ---------------
```

```
  serviceAccountCredentials: # [string] Service account credentials - Contents of

  # ------------------------------------------------------


  # -------------- if authType is secret ---------------

  textSecret: # [string] Service account credentials (text secret) - Select or cr

  # ------------------------------------------------------

  parquetChunkSizeMB: # [number; minimum: 1, maximum: 100] Parquet chunk size lim
  parquetChunkDownloadTimeout: # [number; minimum: 1, maximum: 3600] Parquet chun

# ------------------------------------------------------


# -------------- if type is health_check ---------------

conf: # [object] [required]
  discovery: # [object]
    discoverType: # [string] Discover Type - Defines how task discovery will be p

    # -------------- if discoverType is http ---------------

    discoverUrl: # [string] Discover URL - Expression to derive URL to use for th
    discoverMethod: # [string] Discover method - Discover HTTP method.
    discoverRequestHeaders: # [array] Discover Headers - Optional discover reques
      - name: # [string] Name - Header name.
        value: # [string] Value - JavaScript expression to compute the header val
    discoverDataField: # [string] Discover Data Field - Path to field in the resp

    # ------------------------------------------------------


    # -------------- if discoverType is json ---------------

    manualDiscoverResult: # [string] Discover result - Allows hard-coding the Dis
    discoverDataField: # [string] Discover data field - Within the response JSON,

    # ------------------------------------------------------
```

```
  # -------------- if discoverType is list ---------------

  itemList: # [array of strings] Discover items - Comma-separated list of items

  # ------------------------------------------------------

collectUrl: # [string] Health check URL - Expression to derive URL to use for t
collectMethod: # [string] [required] Health check method - Health check HTTP me

# -------------- if collectMethod is get ---------------

collectRequestParams: # [array] Health check parameters - Optional health check
  - name: # [string] Name - Parameter name
    value: # [string] Value - JavaScript expression to compute the parameter va

# ------------------------------------------------------


# -------------- if collectMethod is post ---------------

collectRequestParams: # [array] Health check parameters - Optional health check
  - name: # [string] Name - Parameter name.
    value: # [string] Value - JavaScript expression to compute the parameter va

# ------------------------------------------------------


# -------------- if collectMethod is post_with_body ---------------

collectBody: # [string] Health check POST Body - Template for POST body to send

# ------------------------------------------------------

collectRequestHeaders: # [array] Health check headers - Optional health check r
  - name: # [string] Name - Header Name
    value: # [string] Value - JavaScript expression to compute the header value
authenticateCollect: # [boolean] Authenticate health check - Enable to make aut
authentication: # [string] [required] Authentication - Authentication method fo

# -------------- if authentication is basic ---------------

username: # [string] Username - Basic authentication username
```

```
password: # [string] Password - Basic authentication password

# ------------------------------------------------------


# -------------- if authentication is basicSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a stored se

# ------------------------------------------------------


# -------------- if authentication is login ---------------

loginUrl: # [string] Login URL - URL to use for login API call. This call is ex
username: # [string] Username - Login username
password: # [string] Password - Login password
loginBody: # [string] POST body - Template for POST body to send with login req
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderExpr: # [string] Authorize Expression - JavaScript expression to comp
authRequestHeaders: # [array] Authentication Headers - Optional authentication
  - name: # [string] Name - Header name.
    value: # [string] Value - JavaScript expression to compute the header value

# ------------------------------------------------------


# -------------- if authentication is loginSecret ---------------

loginUrl: # [string] Login URL - URL to use for login API call, this call is ex
credentialsSecret: # [string] Credentials secret - Select or create a stored se
loginBody: # [string] POST body - Template for POST body to send with login req
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderExpr: # [string] Authorize Expression - JavaScript expression to comp
authRequestHeaders: # [array] Authentication Headers - Optional authentication
  - name: # [string] Name - Header name.
    value: # [string] Value - JavaScript expression to compute the header value

# ------------------------------------------------------


# -------------- if authentication is oauth ---------------
```

```
loginUrl: # [string] Login URL - URL to use for the OAuth API call. This call i
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp
clientSecretParamName: # [string] Client secret parameter - Parameter name that
clientSecretParamValue: # [string] Client secret value - Secret value to add to
authRequestParams: # [array] Extra authentication parameters - OAuth request pa
  - name: # [string] Name - Parameter name.
    value: # [string] Value - JavaScript expression to compute the parameter's
authRequestHeaders: # [array] Authentication headers - Optional authentication
  - name: # [string] Name - Header name.
    value: # [string] Value - JavaScript expression to compute the header's val

# -------------------------------------------------------


# -------------- if authentication is oauthSecret ---------------

loginUrl: # [string] Login URL - URL to use for the OAuth API call. This call i
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp
clientSecretParamName: # [string] Client secret parameter - Parameter name that
textSecret: # [string] Client secret value (text secret) - Select or create a t
authRequestParams: # [array] Extra authentication parameters - OAuth request pa
  - name: # [string] Name - Parameter name.
    value: # [string] Value - JavaScript expression to compute the parameter's
authRequestHeaders: # [array] Authentication headers - Optional authentication
  - name: # [string] Name - Header name.
    value: # [string] Value - JavaScript expression to compute the header's val

# -------------------------------------------------------

timeout: # [number; maximum: 1800] Request Timeout (secs) - HTTP request inacti
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Whether to r
defaultBreakers: # [string] Hidden Default Breakers
safeHeaders: # [array of strings] Safe headers - List of headers that are safe
retryRules: # [object]
  type: # [string] Retry type - The algorithm to use when performing HTTP retri

  # -------------- if type is static ---------------

  interval: # [number; maximum: 20000] Wait (ms) - Time interval between retrie
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to r
  codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigge
```

```
    enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After he

    # ---------------------------------------------------------


    # -------------- if type is backoff ---------------

    interval: # [number; maximum: 20000] Initial retry interval (ms) - Time inter
    limit: # [number; maximum: 20] Retry limit - The maximum number of times to r
    multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigge
    enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After he

    # ---------------------------------------------------------


# ---------------------------------------------------------


# -------------- if type is office365_mgmt ---------------

conf: # [object] [required]
  plan_type: # [string] Subscription plan - Office 365 subscription plan for your
  tenant_id: # [string] [required] Tenant identifier - Directory ID (tenant ident
  app_id: # [string] [required] Application identifier - Identifier of the regist
  client_secret: # [string] [required] Client secret - Application key of the reg
  publisher_identifier: # [string] Publisher identifier - Optional PublisherIdent
  content_type: # [string] [required] Content type - The type of content to retri
  ingestionLag: # [number] Ingestion lag (minutes) - This is necessary because th

# ---------------------------------------------------------


# -------------- if type is office365_service ---------------

conf: # [object] [required]
  tenant_id: # [string] Tenant identifier - Directory ID (tenant identifier) in A
  app_id: # [string] [required] Application identifier - Identifier of the regist
  client_secret: # [string] [required] Client secret - Application key of the reg
  content_type: # [string] [required] Content type - The type of content to retri

# ---------------------------------------------------------
```

```
# -------------- if type is prometheus ---------------

conf: # [object] [required]
  dimensionList: # [array] Extra Dimensions - Other dimensions to include in ever
  username: # [string] Username - Optional username for Basic authentication
  password: # [string] Password - Optional password for Basic authentication
  discoveryType: # [string] Discovery Type - Target discovery mechanism, use stat

  # -------------- if discoveryType is static ---------------

  targetList: # [array] Targets - List of Prometheus targets to pull metrics from

  # ----------------------------------------------------------


  # -------------- if discoveryType is dns ---------------

  nameList: # [array] DNS Names - List of DNS names to resolve
  recordType: # [string] Record Type - DNS Record type to resolve
  scrapeProtocol: # [string] Metrics Protocol - Protocol to use when collecting m
  scrapePath: # [string] Metrics Path - Path to use when collecting metrics from

  # ----------------------------------------------------------


  # -------------- if discoveryType is ec2 ---------------

  usePublicIp: # [boolean] Use Public IP - Use public IP address for discovered t
  scrapeProtocol: # [string] Metrics Protocol - Protocol to use when collecting m
  scrapePort: # [number; minimum: 1, maximum: 65535] Metrics Port - The port numb
  scrapePath: # [string] Metrics Path - Path to use when collecting metrics from
  searchFilter: # [array] Search Filter - EC2 Instance Search Filter
    - Name: # [string] Filter Name - Search filter attribute name, see: https://d
      Values: # [array of strings] Filter Values - Search Filter Values
  region: # [string] Region - Region from which to retrieve data.
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication
  enableAssumeRole: # [boolean] Enable Assume Role - Use Assume Role credentials
  endpoint: # [string] Endpoint - EC2 service endpoint. If empty, defaults to AWS
  signatureVersion: # [string] Signature version - Signature version to use for s

  # ----------------------------------------------------------
```

```
# ------------------------------------------------------


# -------------- if type is rest ---------------

conf: # [object] [required]
  type: # [string]
  discovery: # [object]
    discoverType: # [string] Discover type - Defines how task discovery will be p

    # -------------- if discoverType is http ---------------

    discoverUrl: # [string] Discover URL - URL to use for the Discover operation.
    discoverMethod: # [string] Discover method
    discoverRequestHeaders: # [array] Discover headers
      - name: # [string] Name
        value: # [string] Value - JavaScript expression to compute parameter valu
    pagination: # [object]
      type: # [string] Pagination

      # -------------- if type is response_body ---------------

      attribute: # [array,string] Response attributes - Names of attributes withi
      maxPages: # [number] Page limit - Maximum number of pages to retrieve for t
      lastPageExpr: # [string] Last-page expression - JavaScript expression used

      # ------------------------------------------------------


      # -------------- if type is response_header ---------------

      attribute: # [array,string] Response attributes - Names of attributes withi
      maxPages: # [number] Page limit - Maximum number of pages to retrieve for t

      # ------------------------------------------------------


      # -------------- if type is response_header_link ---------------

      nextRelationAttribute: # [string] Next page relation name - Relation name u
      curRelationAttribute: # [string] Current page relation name - Relation name
      maxPages: # [number] Page limit - Maximum number of pages to retrieve for t
```

```
  # ------------------------------------------------------


  # -------------- if type is request_offset ---------------

  offsetField: # [string] Offset field name - Query string parameter that set
  offset: # [number] Starting offset - Offset index from which to start reque
  offsetSpacer: # [null]
  limitField: # [string] Limit field name - Query string parameter that sets
  limit: # [number; minimum: 1] Record limit - Maximum number of records to r
  limitSpacer: # [null]
  totalRecordField: # [string] Total record count field name - Name of the at
  maxPages: # [number] Page limit - Maximum number of pages to retrieve for t
  zeroIndexed: # [boolean] Zero-based index - Enable to indicate that the fir

  # ------------------------------------------------------


  # -------------- if type is request_page ---------------

  pageField: # [string] Page number field name - Query string parameter that
  page: # [number] Starting page number - Page number from which to start rec
  offsetSpacer: # [null]
  sizeField: # [string] Page size field name - Query string parameter that se
  size: # [number; minimum: 1] Record limit - Maximum number of records to re
  limitSpacer: # [null]
  totalPageField: # [string] Total page count field name - Name of the attrib
  totalRecordField: # [string] Total record count field name - Name of the at
  maxPages: # [number] Page limit - Maximum number of pages to retrieve for t
  zeroIndexed: # [boolean] Zero-based index - Enable to indicate that the fir

  # ------------------------------------------------------

discoverDataField: # [string] Discover data field - Path to field in the resp
enableDiscoverCode: # [boolean] Format discover result with custom code

# ------------------------------------------------------


# -------------- if discoverType is json ---------------

manualDiscoverResult: # [string] Discover result - Allows hard-coding the Dis
```

```
  discoverDataField: # [string] Discover data field - Within the response JSON,

  # ------------------------------------------------------

  # -------------- if discoverType is list ---------------

  itemList: # [array of strings] Discover items - Comma-separated list of items

  # ------------------------------------------------------

collectUrl: # [string] Collect URL - URL (constant or JavaScript expression) to
collectMethod: # [string] [required] Collect method

# -------------- if collectMethod is get ---------------

collectRequestParams: # [array] Collect parameters
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,

# ------------------------------------------------------

# -------------- if collectMethod is post ---------------

collectRequestParams: # [array] Collect parameters
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,

# ------------------------------------------------------

# -------------- if collectMethod is post_with_body ---------------

collectBody: # [string] Collect POST body - Template for POST body to send with

# ------------------------------------------------------

# -------------- if collectMethod is other ---------------

collectVerb: # [string] Collect verb - Custom HTTP method to use for the Collec
collectBody: # [string] Collect body - Template for body to send with the Colle
```

```
collectRequestParams: # [array] Collect parameters
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,

# -----------------------------------------------------

collectRequestHeaders: # [array] Collect headers
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,
pagination: # [object]
  type: # [string] Pagination

  # -------------- if type is response_body ---------------

  attribute: # [array,string] Response attributes - Names of attributes within
  maxPages: # [number] Page limit - Maximum number of pages to retrieve per col
  lastPageExpr: # [string] Last-page expression - JavaScript expression used to

  # -----------------------------------------------------


  # -------------- if type is response_header ---------------

  attribute: # [array,string] Response attributes - Names of attributes within
  maxPages: # [number] Page limit - Maximum number of pages to retrieve per col

  # -----------------------------------------------------


  # -------------- if type is response_header_link ---------------

  nextRelationAttribute: # [string] Next page relation name - Relation name use
  curRelationAttribute: # [string] Current page relation name - Relation name u
  maxPages: # [number] Page limit - Maximum number of pages to retrieve per col

  # -----------------------------------------------------


  # -------------- if type is request_offset ---------------

  offsetField: # [string] Offset field name - Query string parameter that sets
  offset: # [number] Starting offset - Offset index from which to start request
  offsetSpacer: # [null]
```

```
  limitField: # [string] Limit field name - Query string parameter that sets th
  limit: # [number; minimum: 1] Record limit - Maximum number of records to col
  limitSpacer: # [null]
  totalRecordField: # [string] Total record count field name - Name of the attr
  maxPages: # [number] Page limit - Maximum number of pages to retrieve per col
  zeroIndexed: # [boolean] Zero-based index - Enable to indicate that the first

  # ------------------------------------------------------


  # -------------- if type is request_page ---------------

  pageField: # [string] Page number field name - Query string parameter that se
  page: # [number] Starting page number - Page number from which to start reque
  offsetSpacer: # [null]
  sizeField: # [string] Page size field name - Query string parameter that sets
  size: # [number; minimum: 1] Record limit - Maximum number of records to coll
  limitSpacer: # [null]
  totalPageField: # [string] Total page count field name - Name of the attribut
  totalRecordField: # [string] Total record count field name - Name of the attr
  maxPages: # [number] Page limit - Maximum number of pages to retrieve per col
  zeroIndexed: # [boolean] Zero-based index - Enable to indicate that the first

  # ------------------------------------------------------

authentication: # [string] [required] Authentication - Authentication method fo

# -------------- if authentication is basic ---------------

username: # [string] Username
password: # [string] Password

# ------------------------------------------------------


# -------------- if authentication is basicSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a stored se

# ------------------------------------------------------


# -------------- if authentication is login ---------------
```

```
loginUrl: # [string] Login URL - URL to use for login API call. This call is ex
username: # [string] Login username
password: # [string] Login password
loginBody: # [string] POST body - Template for POST body to send with login req
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderKey: # [string] Authorization header - Authorization header key to pa
authHeaderExpr: # [string] Authorize expression - JavaScript expression used to
authRequestHeaders: # [array] Authentication headers
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,

# -------------------------------------------------------


# -------------- if authentication is loginSecret ---------------

loginUrl: # [string] Login URL - URL to use for login API call. This call is ex
credentialsSecret: # [string] Credentials secret - Select or create a stored se
loginBody: # [string] POST body - Template for POST body to send with login req
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderKey: # [string] Authorization header - Authorization header key to pa
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp
authRequestHeaders: # [array] Authentication headers
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,

# -------------------------------------------------------


# -------------- if authentication is oauth ---------------

loginUrl: # [string] Login URL - URL to use for the OAuth API call. This call i
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderKey: # [string] Authorization header - Authorization header key to pa
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp
clientSecretParamName: # [string] Client secret parameter - Defaults to 'client
clientSecretParamValue: # [string] Client secret value - Secret value to add to
authRequestParams: # [array] Extra authentication parameters - OAuth request pa
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,
authRequestHeaders: # [array] Authentication headers
  - name: # [string] Name
```

```
    value: # [string] Value - JavaScript expression to compute parameter value,

# ------------------------------------------------------


# -------------- if authentication is oauthSecret ---------------

loginUrl: # [string] Login URL - URL to use for the OAuth API call. This call i
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderKey: # [string] Authorization header - Authorization header key to pa
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp
clientSecretParamName: # [string] Client secret parameter - Defaults to 'client
textSecret: # [string] Client secret value (text secret) - Select or create a t
authRequestParams: # [array] Extra authentication parameters - OAuth request pa
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,
authRequestHeaders: # [array] Authentication headers
  - name: # [string] Name
    value: # [string] Value - JavaScript expression to compute parameter value,

# ------------------------------------------------------


# -------------- if authentication is google_oauth ---------------

scopes: # [array of strings] Scopes - Scopes to use during authentication. See
serviceAccountCredentials: # [string] Service account credentials - Contents of
subject: # [string] Impersonated account's email address - Email address of a u

# ------------------------------------------------------


# -------------- if authentication is google_oauthSecret ---------------

scopes: # [array of strings] Scopes - Scopes to use during authentication. See
textSecret: # [string] Service account credentials (text secret) - Select or cr
subject: # [string] Impersonated account's email address - Email address of a u

# ------------------------------------------------------


# -------------- if authentication is hmac ---------------
```

```
hmacFunctionId: # [string] HMAC Function - Select or create an HMAC Function to

# ------------------------------------------------------

timeout: # [number; maximum: 1800] Request timeout (secs) - HTTP request inacti
useRoundRobinDns: # [boolean] Round-robin DNS - Use round-robin DNS lookup. Sui
disableTimeFilter: # [boolean] Disable time filter - Disable Collector event ti
decodeUrl: # [boolean] Decode URL - Decode the URL before sending requests (inc
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certi
captureHeaders: # [boolean] Capture response headers - Enable to add response h
safeHeaders: # [array of strings] Safe headers - List of headers that are safe
retryRules: # [object]
  type: # [string] Retry type - Algorithm to use when performing HTTP retries

  # -------------- if type is static ---------------

  interval: # [number; maximum: 20000] Wait (ms) - Time interval between retrie
  limit: # [number; maximum: 20] Retry limit - Maximum number of times to retry
  codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigge
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After he
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single ret
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a

  # ------------------------------------------------------


  # -------------- if type is backoff ---------------

  interval: # [number; maximum: 20000] Initial retry interval (ms) - Time inter
  limit: # [number; maximum: 20] Retry limit - Maximum number of times to retry
  multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxIntervalMs: # [number] Longest interval between retries (ms)
  codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigge
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After he
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single ret
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a

  # ------------------------------------------------------

__scheduling: # [object]
  stateTracking: # [object]
    enabled: # [boolean] Enabled - Track collection progress between consecutiv
```

```
      # -------------- if enabled is true ---------------

      stateUpdateExpression: # [string] State update expression - JavaScript expr
      stateMergeExpression: # [string] State merge expression - JavaScript expres

      # --------------------------------------------------------


# -----------------------------------------------------


# -------------- if type is s3 ---------------

conf: # [object] [required]
  type: # [string]
  outputName: # [string] Auto-populate from - Name of the predefined Destination
  bucket: # [string] S3 bucket - S3 Bucket from which to collect data
  parquetChunkSizeMB: # [number; minimum: 1, maximum: 100] Parquet chunk size lim
  parquetChunkDownloadTimeout: # [number; minimum: 1, maximum: 3600] Parquet chun
  region: # [string] Region - Region from which to retrieve data
  path: # [string] Path - Directory where data will be collected. Templating (suc
  partitioningScheme: # [string] Partitioning scheme - Partitioning scheme used f

  # -------------- if partitioningScheme is none ---------------

  recurse: # [boolean] Recursive - Traverse and include files from subdirectories

  # --------------------------------------------------------

  extractors: # [array] Path extractors - Allows using template tokens as context
    - key: # [string] Token - A token from the template path, such as epoch
      expression: # [string] Extractor Expression - JavaScript expression that re
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key - Access key. If not present, will fall back t
  awsSecretKey: # [string] Secret key - Secret key. If not present, will fall bac

  # --------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------
```

```
  awsSecret: # [string] Secret key pair - Select or create a stored secret that r

  # ------------------------------------------------------

  endpoint: # [string] Endpoint - Must point to an S3-compatible endpoint. If emp
  signatureVersion: # [string] Signature version - Signature version to use for s
  enableAssumeRole: # [boolean] Enable Assume Role - Use AssumeRole credentials
  assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the ro
  assumeRoleExternalId: # [string] External ID - External ID to use when assuming
  durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) -
  maxBatchSize: # [number; minimum: 1] Batch size limit (objects) - Maximum numbe
  reuseConnections: # [boolean] Reuse connections - Reuse connections between rec
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certi
  verifyPermissions: # [boolean] Verify bucket permissions - Disable if you can a
  disableTimeFilter: # [boolean] Disable time filter - Disable Collector event ti

# ------------------------------------------------------


# -------------- if type is script ---------------

conf: # [object] [required]
  type: # [string]
  discoverScript: # [string] Discover Script - Script to discover what to collect
  collectScript: # [string] [required] Collect Script - Script to run to perform
  shell: # [string] Shell - Shell to use to execute scripts.
  envVars: # [array] Environment Variables - Environment variables to expose to t
    - name: # [string] Name - Environment variable name
      value: # [string] Value - JavaScript expression to compute environment vari

# ------------------------------------------------------


# -------------- if type is splunk ---------------

conf: # [object] [required]
  type: # [string]
  searchHead: # [string] [required] Search head - Search head base URL. Can be ar
  search: # [string] Search - Examples: 'index=myAppLogs level=error channel=myAp
  earliest: # [string] Earliest - The earliest time boundary for the search. Can
  latest: # [string] Latest - The latest time boundary for the search. Can be an
  endpoint: # [string] [required] Search endpoint - REST API used to create a sea
```

```
outputMode: # [string] [required] Output mode - Format of the returned output
collectRequestParams: # [array] Extra parameters - Optional collect request par
  - name: # [string] Parameter Name
    value: # [string] Value - JavaScript expression to compute the parameter's
collectRequestHeaders: # [array] Extra headers - Optional collect request heade
  - name: # [string] Header Name
    value: # [string] Value - JavaScript expression to compute the header's val
authentication: # [string] [required] Authentication - Authentication method fo

# -------------- if authentication is basic ---------------

username: # [string] Username - Basic authentication username
password: # [string] Password - Basic authentication password

# ---------------------------------------------------------


# -------------- if authentication is basicSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a stored se

# ---------------------------------------------------------


# -------------- if authentication is token ---------------

token: # [string] Bearer token

# ---------------------------------------------------------


# -------------- if authentication is tokenSecret ---------------

tokenSecret: # [string] Bearer token secret - Select or create a stored secret

# ---------------------------------------------------------


# -------------- if authentication is login ---------------

loginUrl: # [string] Login URL - URL to use for login API call. This call is ex
username: # [string] Username
password: # [string] Password
```

```
loginBody: # [string] POST body - Template for POST body to send with login rec
tokenRespAttribute: # [string] Token attribute - Path to token attribute in log
authHeaderExpr: # [string] Authorize Expression - JavaScript expression to comp

# -------------------------------------------------------


# -------------- if authentication is loginSecret ---------------

loginUrl: # [string] Login URL - URL to use for login API call, this call is ex
credentialsSecret: # [string] Credentials secret - Select or create a stored se
loginBody: # [string] POST Body - Template for POST body to send with login rec
tokenRespAttribute: # [string] Token Attribute - Path to token attribute in log
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comp

# -------------------------------------------------------

timeout: # [number; maximum: 1800] Request timeout (secs) - HTTP request inacti
useRoundRobinDns: # [boolean] Round-robin DNS - Use round-robin DNS lookup. Sui
disableTimeFilter: # [boolean] Disable time filter - Disable collector event ti
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certi
retryRules: # [object]
  type: # [string] Retry type - Algorithm to use when performing HTTP retries

  # -------------- if type is static ---------------

  interval: # [number; maximum: 20000] Wait (ms) - Time interval between retrie
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to r
  codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigge
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After he
  retryConnectTimeout: # [boolean] Retry connection timeout - Make a single ret
  retryConnectReset: # [boolean] Retry connection reset - Retry request when a

  # -------------------------------------------------------


  # -------------- if type is backoff ---------------

  interval: # [number; maximum: 20000] Initial retry interval (ms) - Time inter
  limit: # [number; maximum: 20] Retry limit - The maximum number of times to r
  multiplier: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  codes: # [array of numbers] Retry HTTP codes - List of HTTP codes that trigge
  enableHeader: # [boolean] Honor Retry-After header - Honor any Retry-After he
```

```
      retryConnectTimeout: # [boolean] Retry connection timeout - Make a single ret
      retryConnectReset: # [boolean] Retry connection reset - Retry request when a

      # ------------------------------------------------------


  # ------------------------------------------------------

  destructive: # [boolean] Destructive - Delete any files collected (where applicab
  encoding: # [string] Encoding - Character encoding to use when parsing ingested d
input: # [object]
  type: # [string]
  breakerRulesets: # [array of strings] Event Breaker rulesets - A list of event-br
  staleChannelFlushMs: # [number; minimum: 10, maximum: 43200000] Event Breaker buf
  sendToRoutes: # [boolean] Send to Routes - Send events to normal routing and even

  # -------------- if sendToRoutes is true ---------------

  pipeline: # [string] Pre-Processing Pipeline - An optional Pipeline to process re

  # ------------------------------------------------------


  # -------------- if sendToRoutes is false ---------------

  pipeline: # [string] Pipeline - Pipeline to process results
  output: # [string] Destination - Destination to send results to

  # ------------------------------------------------------

  preprocess: # [object]
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    command: # [string] Command - Command to feed the data through (via stdin) and
    args: # [array of strings] Arguments - Arguments to be added to the custom comm

    # ------------------------------------------------------

  throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
```

```
      value: # [string] Value - JavaScript expression to compute field's value, enc
description: # [string] Description
type: # [string] Job type
ttl: # [string] Time to live - Time to keep the job's artifacts on disk after job c
ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled, th
removeFields: # [array of strings] Remove Discover fields - List of fields to remov
resumeOnBoot: # [boolean] Resume job on boot - Resume the ad hoc job if a failure c
environment: # [string] Environment - Optionally, enable this config only on a spec
schedule: # [object] Schedule - Configuration for a scheduled job
  enabled: # [boolean] Enabled - Enable to configure scheduling for this Collector

  # -------------- if enabled is true ---------------

  cronSchedule: # [string] Cron schedule - A cron schedule on which to run this job
  maxConcurrentRuns: # [number; minimum: 1] Concurrent run limit - The maximum numb
  skippable: # [boolean] Skippable - Skippable jobs can be delayed, up to their nex
  run: # [object] Run settings

    # -------------- if type is collection ---------------

    rescheduleDroppedTasks: # [boolean] Reschedule tasks - Reschedule tasks that fa
    maxTaskReschedule: # [number; minimum: 1] Task reschedule limit - Maximum numbe
    logLevel: # [string] Log level - Level at which to set task logging
    jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run. Ti
    mode: # [string] Mode - Job run mode. Preview will either return up to N matchi

    # -------------- if mode is list ---------------

    discoverToRoutes: # [boolean] Send to Routes - Send discover results to Routes

    # -----------------------------------------------------



    # -------------- if mode is preview ---------------

    capture: # [object] Capture Settings
      duration: # [number] Capture time (sec) - Amount of time to keep capture oper
      maxEvents: # [number; minimum: 1, maximum: 10000] Capture up to N events - Ma
      level: # [string] Where to capture

    # -----------------------------------------------------

    timeRangeType: # [string] Time range
```

```
      # -------------- if timeRangeType is absolute ---------------

      timestampTimezone: # [string] Range timezone - Timezone to use for Earliest and

      # ------------------------------------------------------

      earliest: # [number,string] Earliest - Earliest time to collect data for the se
      latest: # [number,string] Latest - Latest time to collect data for the selected
      timeWarning: # [object]
      expression: # [string] Filter - A filter for tokens in the provided collect pat
      minTaskSize: # [string] Lower task bundle size - Limits the bundle size for sma
        if your lower bundle size is 1MB, you can bundle up to five 200KB files into
      maxTaskSize: # [string] Upper task bundle size - Limits the bundle size for fil
        you can bundle up to five 2MB files into one task. Files greater than this si

    # ------------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl St
executor_job: # [object]
  executor: # [object]
    type: # [string] Executor type - The type of executor to run
    storeTaskResults: # [boolean] Store task results - Determines whether or not to w
    conf: # [object] Executor-specific settings
  description: # [string] Description
  type: # [string] Job type
  ttl: # [string] Time to live - Time to keep the job's artifacts on disk after job c
  ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled, th
  removeFields: # [array of strings] Remove Discover fields - List of fields to remov
  resumeOnBoot: # [boolean] Resume job on boot - Resume the ad hoc job if a failure c
  environment: # [string] Environment - Optionally, enable this config only on a spec
  schedule: # [object] Schedule - Configuration for a scheduled job
    enabled: # [boolean] Enabled - Enable to configure scheduling for this Collector

    # -------------- if enabled is true ---------------

    cronSchedule: # [string] Cron schedule - A cron schedule on which to run this job
    maxConcurrentRuns: # [number; minimum: 1] Concurrent run limit - The maximum numb
    skippable: # [boolean] Skippable - Skippable jobs can be delayed, up to their nex
    run: # [object] Run settings

      # -------------- if type is collection ---------------
```

```
      rescheduleDroppedTasks: # [boolean] Reschedule tasks - Reschedule tasks that fa
      maxTaskReschedule: # [number; minimum: 1] Task reschedule limit - Maximum numbe
      logLevel: # [string] Log level - Level at which to set task logging
      jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run. Ti
      mode: # [string] Mode - Job run mode. Preview will either return up to N matchi

      # -------------- if mode is list ---------------

      discoverToRoutes: # [boolean] Send to Routes - Send discover results to Routes

      # -----------------------------------------------------


      # -------------- if mode is preview ---------------

      capture: # [object] Capture Settings
        duration: # [number] Capture time (sec) - Amount of time to keep capture oper
        maxEvents: # [number; minimum: 1, maximum: 10000] Capture up to N events - Ma
        level: # [string] Where to capture

      # -----------------------------------------------------

      timeRangeType: # [string] Time range

      # -------------- if timeRangeType is absolute ---------------

      timestampTimezone: # [string] Range timezone - Timezone to use for Earliest and

      # -----------------------------------------------------

      earliest: # [number,string] Earliest - Earliest time to collect data for the se
      latest: # [number,string] Latest - Latest time to collect data for the selected
      timeWarning: # [object]
      expression: # [string] Filter - A filter for tokens in the provided collect pat
      minTaskSize: # [string] Lower task bundle size - Limits the bundle size for sma
        if your lower bundle size is 1MB, you can bundle up to five 200KB files into
      maxTaskSize: # [string] Upper task bundle size - Limits the bundle size for fil
        you can bundle up to five 2MB files into one task. Files greater than this si

    # -----------------------------------------------------

  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl St
scheduledSearch_job: # [object]
```

```
savedQueryId: # [string] ID of the SavedQuery - Identifies which search query to ru
description: # [string] Description
type: # [string] Job type
ttl: # [string] Time to live - Time to keep the job's artifacts on disk after job c
ignoreGroupJobsLimit: # [boolean] Ignore Worker Group job limits - When enabled, th
removeFields: # [array of strings] Remove Discover fields - List of fields to remov
resumeOnBoot: # [boolean] Resume job on boot - Resume the ad hoc job if a failure c
environment: # [string] Environment - Optionally, enable this config only on a spec
schedule: # [object] Schedule - Configuration for a scheduled job
  enabled: # [boolean] Enabled - Enable to configure scheduling for this Collector

  # -------------- if enabled is true ---------------

  cronSchedule: # [string] Cron schedule - A cron schedule on which to run this job
  maxConcurrentRuns: # [number; minimum: 1] Concurrent run limit - The maximum numb
  skippable: # [boolean] Skippable - Skippable jobs can be delayed, up to their nex
  run: # [object] Run settings

    # -------------- if type is collection ---------------

    rescheduleDroppedTasks: # [boolean] Reschedule tasks - Reschedule tasks that fa
    maxTaskReschedule: # [number; minimum: 1] Task reschedule limit - Maximum numbe
    logLevel: # [string] Log level - Level at which to set task logging
    jobTimeout: # [string] Job timeout - Maximum time the job is allowed to run. Ti
    mode: # [string] Mode - Job run mode. Preview will either return up to N matchi

    # -------------- if mode is list ---------------

    discoverToRoutes: # [boolean] Send to Routes - Send discover results to Routes

    # -----------------------------------------------------



    # -------------- if mode is preview ---------------

    capture: # [object] Capture Settings
      duration: # [number] Capture time (sec) - Amount of time to keep capture oper
      maxEvents: # [number; minimum: 1, maximum: 10000] Capture up to N events - Ma
```

10.5.9. Job-Limits.Yml

`job-limits.yml` maintains parameters for Collection jobs and system tasks. In the UI, you can configure these per Worker Group at **Group Settings** > **General Settings** > **Limits** > **Jobs**.

Where we do not specify or validate a minimum or maximum value, Cribl provides a prudent default. Adjust this initial value to match your architecture and needs.

job-limits.yml

```
# Disable jobs/tasks - When jobs/tasks are disabled, Worker and Edge Nodes on versior
# poll the Leader for tasks. Nodes running 4.4.4 and older will honor the Jobs settir
# are disabled.
# [boolean; default: false]
disableTasks:
# Concurrent job limit - The total number of jobs that may run concurrently
# [number; min: 1; default: 10]
concurrentJobLimit:
# Concurrent system job limit - The total number of system jobs that may run concurre
# [number; min: 1; default: 10]
concurrentSystemJobLimit:
# Concurrent scheduled job limit - The total number of scheduled jobs that can run cc
# relative to concurrent job limit.
# [number; default: -2]
concurrentScheduledJobLimit:
# Concurrent task limit - The total number of tasks that a Worker Process can run cor
# [number; min: 0; default: 2]
concurrentTaskLimit:
# Concurrent system task limit - The number of system tasks that a Worker Process car
# [number; min: 0; default: 1]
concurrentSystemTaskLimit:
# Task usage percentage limit - Value from 0 to 1 representing the percentage of tota
# system a job may consume
# [number; min: 0; max: 1; default: 0.5]
maxTaskPerc:
# Task poll timeout - The number of milliseconds the Worker's task handler will wait
# before retrying the request for a task
# [number; min: 10000; default: 60000]
taskPollTimeoutMs:
# Artifact reaper period - Time period at which the system attempts to reap stale dis
# to the jobs
# [string; default: 30m]
jobArtifactsReaperPeriod:
# Finished job artifacts limit - Maximum number of finished job artifacts to keep on
# [number; min: 0; default: 100]
finishedJobArtifactsLimit:
```

10.5.10. Leader.Yml

`leader.yml` contains configuration for a secondary Leader and is located under `$CRIBL_HOME/local/cribl` in the volume of that Leader.

This configuration file is created when you have configured a Leader for failover and the new Leader becomes active in case of failover.

leader.yml

```
# Host/IP address to bind to - The host address for the instance
# [string; default: 0.0.0.0; required]
host:
# Port number to bind to - The port number for the instance
# [number; default: 4200; required]
port:
# Protocol - Communication protocol
# [string]
protocol:
# Auth Token - Authentication token for secure communication
# [string]
authToken:
# IP allowlist regex - Regex matching IP addresses that are allowed to establish a cc
# [string; default: /.*/]
ipWhitelistRegex:
# Proxy settings - SOCKS proxy configuration
proxy:
# Compression - Data compression method
# [string]
compression:
# Connection timeout - Timeout for establishing connections
# [number]
connectionTimeout:
# Write timeout - Timeout for write operations
# [number]
writeTimeout:
# Max buffer bytes - Maximum buffer size in bytes
# [number]
maxBufferBytes:
# Forward to Leader API - Controls whether distributed traffic gets proxied to Leader
# [boolean; default: true]
forwardToLeaderApi:
# TLS configuration - TLS/SSL settings
tls:
  # Disabled - Whether TLS is disabled
  # [boolean]
  disabled:
  # Certificate Name - Name of the certificate to use
  # [string]
  certificateName:
  # Private Key Path - Path to private key file
  # [string]
```

```
privKeyPath:
# Passphrase - Passphrase for private key
# [string]
passphrase:
# Certificate Path - Path to certificate file
# [string]
certPath:
# CA Path - Path to certificate authority file
# [string]
caPath:
# Request Certificate - Whether to request client certificates
# [boolean]
requestCert:
# Reject Unauthorized - Whether to reject unauthorized certificates
```

10.5.11. LIMITS.YML

`limits.yml` maintains parameters for the system. In a distributed deployment, these parameters are configured on both the Leader and the Workers.

In the UI, you can configure them:

- On the Leader at **Settings** > **Global** > **General Settings > Limits**.
- On a single instance at **Settings** > **General Settings > Limits**.
- On Worker Groups at `<Group/Fleet-name>` > **Group Settings** > **General Settings > Limits**.

Where we do not specify or validate a minimum or maximum value, Cribl provides a prudent default. Adjust this initial value to match your architecture and needs.

limits.yml

```
# Default managed node heartbeat period - How many seconds a managed node will wait t
# the Cribl control plane
# [number; min: 1]
defaultHeartbeatSeconds:
# Samples
samples:
  # Sample size limit - Maximum file size for the sample, in binary units (KB, MB). (
  # [string]
  maxSize:
# Min free disk space - The minimum amount of disk space in the system before various
# measures to prevent disk usage (KB, MB, etc.)
# [string; default: 5 GB]
minFreeSpace:
# Worker Process PQ size limit - Highest accepted value for individual integrations'
# queue size' parameters. Consult Cribl Support before increasing the default 1 TB.
# [string; default: 1TB]
maxPQSize:
# Metrics GC period - The interval at which the system attempts to free memory by pru
# the Stream system metrics store
# [string; default: 60s]
metricsGCPeriod:
# Number of metrics allowed - The system's total allowed number of metric series. Cor
# before updating this value.
# [number; default: 1000000]
maxMetrics:
# Worker metrics limit - The maximum number of metric series allowed per worker. Cons
# before updating this value.
# [number; default: 100000]
workerMaxMetrics:
# Metrics cardinality limit - The system's allowed number of permutations of a given
# [number; default: 1000]
metricsMaxCardinality:
# Metrics disk space limit - Maximum allowed disk space for persisting metrics to dis
# [string; default: 64GB]
metricsMaxDiskSpace:
# Metrics worker tracking - List of metric names for which to disable tracking of Wor
# Supports wildcards.
# [array; default: host.*, source.*, sourcetype.*, index.*]
metricsWorkerIdBlacklist:
# Metrics never-drop list - List metric names for which to ensure delivery. Supports
# [array; default: total.*]
metricsNeverDropList:
```

```
# Disable field metrics - List of event fields for which to disable metric collectior
# [array; default: source, host, index, sourcetype]
metricsFieldsBlacklist:
# Metrics directory - Directory to store metrics on disk
# [string; default: $CRIBL_STATE_DIR/metrics]
metricsDirectory:
# CPU profile TTL - The time to live for collected CPU profiles
# [string; default: 30m]
cpuProfileTTL:
# Event metadata sources - List of event metadata sources to enable
eventsMetadataSources:
# Metrics to send from Edge Nodes - Choose a set of metrics for Edge Nodes to send to
# a custom set. Details about pre-defined sets are in [the docs](https://docs.cribl.i
# One of: basic | detailed | none | custom
# [string; default: basic]
edgeMetricsMode:
# Metrics expression - JavaScript expression to filter metrics
# [string; required]
edgeMetricsCustomExpression:
# Persist metrics - Disable to stop writing metrics to disk
# [boolean; default: true]
enableMetricsPersistence:
# Edge Nodes estimated count - Used to pre-allocate connection handlers. Changing thi
# currently running connection handlers, causing all Edge Nodes to reconnect. A blank
# default, which starts a handler on demand when an Edge Node attempts to connect.
# [number; min: 0; max: 250000; default: 0]
edgeNodesCount:
# Disable Node persistence - Do not persist connection data across restarts. Caution:
# [boolean; default: false]
enableWorkerInMemory:
# Enable KVStore WAL - Enable the write-ahead log for the KVStore
# [boolean; default: false]
kvStoreEnableWAL:
# KVStore persist period - How often the KVStore syncs itself to disk. When WAL mode
# the KVStore are synced to disk immediately as they happen; however, the WAL is merg
# KVStore on this interval.
# [number; min: 10000]
kvStoreServicePeriod:
# KVStore WAL file bytes limit - The amount of bytes to allow a WAL file to get to be
# back into the KVStore. This may result in performance degradation if adjusted.
# [number; min: 1024; default: 5242880]
kvStoreWALMaxBytes:
# Config bundle download timeout (secs) - The maximum time a Worker will wait for a s
```

```
# connection before canceling a download of a new configuration bundle. A 0 value mea
# which could cause Workers to hang.
# [number; min: 0; default: 0]
bundleDownloadTimeoutSeconds:
# Minimum reconnect interval - The minimum time to wait before attempting to reconnec
# being disconnected. The interval is doubled with each failed reconnect attempt unti
# reached.
# [string; default: 2s]
minReconnectInterval:
# Maximum reconnect interval - The maximum time to wait before attempting to reconnec
# being disconnected.
# [string; default: 1m]
maxReconnectInterval:
# Random reconnect interval - Upper limit on the random time added to each reconnect
# non-zero interval to randomly spread out reconnection attempts. Useful for Leaders
# Worker or Edge nodes.
# [string; default: 0s]
randomReconnectInterval:
# Consult Cribl Support before adjusting this advanced setting. The maximum number of
# a worker node is allowed to ingest. If 0 or not configured, no rate limiting is app
# means automatic determination based on configurations.
# [string; default: '']
rateLimitBytesPerSecond:
# Use Windows tools to collect process info - Enable PowerShell to collect process ir
# using the native API
# [boolean; default: false]
enableProcessInfoUsingWindowsTools:
# Lookup file max size - Maximum allowed disk space for persisting single lookup data
# [string; default: 500MB]
lookupMaxSize:
# Lookup files disk limit - Maximum allowed disk space for persisting lookup data and
# [string; default: 2GB]
lookupMaxTotalSize:
```

10.5.12. Logger.Yml

`logger.yml` maintains logging levels and redactions, per channel. In the UI, you can configure these at **Settings** > **Global** > **System** > **Logging**.

logger.yml

```
# Logger Configuration ID
# [string]
id:
# Default redact fields - List of default field names to redact in internal logs
defaultRedactFields:
# Additional redact fields - List of additional field names to redact in internal log
redactFields:
# Custom redact string - Literal redaction string to apply on selected field values.
# [string]
redactLabel:
# Log channels - Configuration for individual log channels
channels:
# TTL configuration - Time-to-live configuration for log levels
ttlConfig:
  # Log level - Level at which to log messages
  # One of: error | warn | info | http | verbose | debug | silly
  # [string; required]
  level:
  # Time to live - Duration in seconds after which log level will revert to default
  # [number; required]
  ttl:
# Max log file size - Maximum log file size in bytes
# [number]
maxSizeBytes:
# Rate limit - Rate limit for log messages (0 to disable)
# [number]
limitRate:
```

Copy

10.5.13. Mappings.Yml And Fleet-Mappings.Yml

Mapping ruleset configurations for Stream Workers and Edge Node are stored in `$CRIBL_HOME/local/cribl/mappings.yml` and `$CRIBL_HOME/local/cribl/fleet-mappings.yml`, respectively.

mappings.yml or fleet-mappings.yml

```
# Mapping ID - Unique identifier for the mapping ruleset
# [string; required]
id:
# Configuration - Mapping ruleset configuration
conf:
  # Functions - List of functions to pass data through
  functions:
# Active - Whether this mapping ruleset is active
# [boolean]
active:
```

Copy

10.5.14. Messages.Yml

`messages.yml` stores messages displayed in the UI's `Messages` fly-out.

logger.yml

```
# Message ID - Unique identifier for the message
# [string]
id:
# Message Type - Type of the bulletin message
# [string]
type:
# Message Text - Content of the message
# [string]
message:
# Severity - Severity level of the message
# [string]
severity:
# Created - Timestamp when the message was created
# [number]
created:
# TTL - Time to live for the message
# [number]
ttl:
```

Copy

10.5.15. Notifications.Yml

`notifications.yml` contains settings for Notifications.

notifications.yml

```
# Notification Targets - Configuration for notification targets
targets:
  # Output ID - Unique identifier for this notification target
  # [string; required]
  id:
  # Output Type - Type of notification target
  # [string; required]
  type:
  # System fields - Fields to automatically add to events, such as cribl_pipe. Suppor
  # [array; default: cribl_host]
  systemFields:
  # Webhook configuration (webhook type only)
  # URL - Webhook endpoint URL
  # [string; required for webhook]
  url:
  # Format - Data format for webhook payload
  # [string; default: ndjson for webhook]
  format:
  # Method - HTTP method for webhook requests
  # [string; default: POST for webhook]
  method:
  # Authentication type - Authentication method for webhook requests
  # [string; default: none for webhook]
  authType:
  # Token - Authentication token (when authType is token)
  # [string]
  token:
  # Username - Username for basic authentication (when authType is basic)
  # [string]
  username:
  # Password - Password for basic authentication (when authType is basic)
  # [string]
  password:
  # Source expression - Expression to evaluate on events to generate output notificat
  # [string]
  srcExpression:
  # Slack configuration (slack type only)
  # Slack webhook URL - Slack incoming webhook URL
  # [string; required for slack]
  slackUrl:
  # PagerDuty configuration (pager_duty type only)
  # Routing key - PagerDuty integration key
```

```
# [string; required for pager_duty]
routingKey:
# Severity - Alert severity level
# [string; required for pager_duty]
severity:
# Group - Alert group identifier
# [string]
group:
# Class - Alert class identifier
# [string]
class:
# Component - Alert component identifier
# [string]
component:
# SNS configuration (sns type only)
# Topic ARN - AWS SNS topic ARN
# [string; required for sns]
topicArn:
# Region - AWS region
# [string; required for sns]
region:
# AWS authentication method - Method for AWS authentication
# [string; default: auto for sns]
awsAuthenticationMethod:
# Destination type - Type of SNS destination
# [string; required for sns]
destinationType:
# Topic type - SNS topic type
# [string; default: standard for sns]
topicType:
# Message group ID - Message group ID for FIFO topics
# [string; required for sns]
messageGroupId:
# Allow list - List of allowed phone numbers or email addresses
allowList:
# Assume role ARN - ARN of the role to assume
# [string]
assumeRoleArn:
# Enable assume role - Whether to enable role assumption
# [boolean]
enableAssumeRole:
# SMTP configuration (smtp type only)
# Host - SMTP server hostname
```

```
    # [string; required for smtp]
    host:
    # Port - SMTP server port
    # [number; required for smtp]
    port:
    # SMTP username - Username for SMTP authentication
    # [string; required for smtp]
    smtpUsername:
    # SMTP password - Password for SMTP authentication
    # [string; required for smtp]
    smtpPassword:
    # From - Sender email address
    # [string; required for smtp]
    from:
    # TLS - TLS configuration for SMTP
    tls:
      # Reject unauthorized - Whether to reject unauthorized certificates
      # [boolean; default: true]
      rejectUnauthorized:
      # Minimum TLS version - Minimum TLS version
      # [string]
      minVersion:
      # Maximum TLS version - Maximum TLS version
      # [string]
      maxVersion:
    # Router configuration (router type only)
    # Routes - Routing rules for notifications
    routes:
    # Bulletin message configuration (bulletin_message type only)
    # Message - Bulletin message content
    # [string]
    message:
    # Notifications log configuration (notifications_log type only)
    # Log notifications to internal log
    logNotifications:
# Notifications - Individual notification configurations
notifications:
    # Notification ID - Unique identifier for this notification
    # [string; required]
    id:
    # Disabled - Whether this notification is disabled
    # [boolean; default: false]
    disabled:
```

```
# Condıtıon - When to trıgger thıs notıfıcatıon
# [string; required]
condition:
# Notification targets - Targets to send any Notifications to
targets:
# Target configuration - Configuration for specific targets
```

10.5.16. Outpost.Yml

`outpost.yml` contains configuration for a Cribl Outpost instance.

outpost.yml

```
# Settings for the internal distributed-management interface
listener:
  # Host - Host/IP address to bind to
  # [string; required]
  host:
  # Port - Port number to bind to
  # [number; required]
  port:
```

10.5.17. Outputs.Yml

`outputs.yml` contains configuration settings for Cribl Stream Destinations.

outputs.yml

```
outputs: # [object]
  default_output: # [object]
    type: # [string] Output Type
    defaultId: # [string,null] Default Output ID - ID of the default output. This wil
    pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
    systemFields: # [array of strings] System fields - Fields to automatically add to
    environment: # [string] Environment - Optionally, enable this config only on a sp
    streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
  webhook_output: # [object]
    type: # [string] Output Type
    method: # [string] Method - The method to use when sending events
    format: # [string] Format - How to format events before sending out

    # -------------- if format is custom ---------------

    customSourceExpression: # [string] Source expression - Expression to evaluate on
    customDropWhenNull: # [boolean] Drop when null - Whether to drop events when the
    customEventDelimiter: # [string] Event delimiter - Delimiter string to insert bet
    customContentType: # [string] Content type - Content type to use for request. Def
    customPayloadExpression: # [string] Batch expression - Expression specifying how

    # ---------------------------------------------------------


    # -------------- if format is advanced ---------------

    advancedContentType: # [string] Content type - HTTP content-type header value
    formatEventCode: # [string] Format inbound event - Custom JavaScript code to form
    formatPayloadCode: # [string] Format outbound payload - Optional JavaScript code

    # ---------------------------------------------------------


    keepAlive: # [boolean] Keep alive - Disable to close the connection immediately a
    concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
    maxPayloadSizeKB: # [number; minimum: 1024, maximum: 512000] Body size limit (KB)
    maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
    compress: # [boolean] Compress - Compress the payload body before sending
    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
        Enabled by default. When this setting is also present in TLS Settings (Client
        that value will take precedence.
    timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
    flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
```

```
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events. Yo
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automatically
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------

authType: # [string] Authentication type - Authentication method to use for the H

# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password
```

```
# ------------------------------------------------------

# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# ------------------------------------------------------

# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ------------------------------------------------------

# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# ------------------------------------------------------

# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth l
  - name: # [string] Name - OAuth header name
    value: # [string] Value - OAuth header value

# ------------------------------------------------------

tls: # [object] TLS settings (client side)
```

```
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
    certificateName: # [string] Certificate - The name of the predefined certificat
    caPath: # [string] CA certificate path - Path on client in which to find CA cer
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
    certPath: # [string] Certificate path (mutual auth) - Path on client in which t
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ---------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  loadBalanced: # [boolean] Load balancing - Enable for optimal performance. Even i

  # -------------- if loadBalanced is false ---------------

  url: # [string] Webhook URL - URL of a webhook endpoint to send events to, such a
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh

  # ---------------------------------------------------------


  # -------------- if loadBalanced is true ---------------

  excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the curren
  urls: # [array] Webhook URLs
    - url: # [string] Webhook LB URL - URL of a webhook endpoint to send events to,
      weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
  loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec

  # ---------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
sentinel_output: # [object]
```

```
keepAlive: # [boolean] Keep alive - Disable to close the connection immediately a
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 100, maximum: 1000] Body size limit (KB) -
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events. Yo
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th
```

```
  # ------------------------------------------------------

  loginUrl: # [string] [required] Login URL - URL for OAuth
  secret: # [string] [required] OAuth secret - Secret parameter value to pass in re
  spacer: # [null]
  client_id: # [string] [required] Client ID - JavaScript expression to compute the
  scope: # [string] Scope - Scope to pass in the OAuth request
  type: # [string] Output Type
  endpointURLConfiguration: # [string] Endpoint configuration - Enter the data coll

  # -------------- if endpointURLConfiguration is url ---------------

  url: # [string] URL - URL to send events to. Can be overwritten by an event's __u

  # ------------------------------------------------------


  # -------------- if endpointURLConfiguration is ID ---------------

  dcrID: # [string] Data collection rule ID - Immutable ID for the Data Collection
  dceEndpoint: # [string] Data collection endpoint - Data collection endpoint (DCE)
  streamName: # [string] Stream name - The name of the stream (Sentinel table) in w

  # ------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
devnull_output: # [object]
  type: # [string] Output Type
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
syslog_output: # [object]
  type: # [string] Output Type
  protocol: # [string] Protocol - The network protocol to use for sending out syslo

  # -------------- if protocol is tcp ---------------
```

```
loadBalanced: # [boolean] Load balancing - For optimal performance, enable load b
connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------

onBackpressure: # [string] Backpressure behavior - How to handle events when all

# ------------------------------------------------------


# -------------- if protocol is udp ---------------

host: # [string] Address - The hostname of the receiver
port: # [number; maximum: 65535] Port - The port to connect to on the provided ho
maxRecordSize: # [number; minimum: 1, maximum: 65535] Record size limit - Maximum
udpDnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (sec) -

# ------------------------------------------------------

facility: # [integer] Facility - Default value for message facility. Will be over
severity: # [integer] Severity - Default value for message severity. Will be over
appName: # [string] App name - Default name for device or application that origin
messageFormat: # [string] Message format - The syslog message format depending on
timestampFormat: # [string] Timestamp format - Timestamp format to use when seria
throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
octetCountFraming: # [boolean] Octet count framing - Prefix messages with the byt
```

```
  logFailedRequests: # [boolean] Log failed requests to disk - Use to troubleshoot
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
splunk_output: # [object]
  type: # [string] Output Type
  host: # [string] Address - The hostname of the receiver
  port: # [number; maximum: 65535] [required] Port - The port to connect to on the
  nestedFields: # [string] Nested field serialization - How to serialize nested fie
  throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
  connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
  writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                  trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
    certificateName: # [string] Certificate - The name of the predefined certificat
    caPath: # [string] CA certificate path - Path on client in which to find CA cer
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
    certPath: # [string] Certificate path (mutual auth) - Path on client in which t
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ---------------------------------------------------------

  enableMultiMetrics: # [boolean] Output multiple metrics - Output metrics in multi
  enableACK: # [boolean] Minimize in-flight data loss - Check if indexer is shuttin

  # -------------- if enableACK is false ---------------

  maxFailedHealthChecks: # [number] Failed health check limit - Maximum number of t

  # ---------------------------------------------------------

  logFailedRequests: # [boolean] Log failed requests to disk - Use to troubleshoot
  maxS2Sversion: # [string] Max S2S version - The highest S2S protocol version to a
```

```
  # -------------- if maxS2Sversion is v4 ---------------

  compress: # [string] Compression - Controls whether the sender should send compre

  # ------------------------------------------------------

  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  authType: # [string] Authentication method - Select Manual to enter an auth token
  authToken: # [string] Auth token - Shared secret token to use when establishing a

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored text

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
splunk_lb_output: # [object]
  type: # [string] Output Type
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
```

```
loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec
maxConcurrentSenders: # [number] Connection limit - Maximum number of concurrent
nestedFields: # [string] Nested field serialization - How to serialize nested fie
throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ---------------------------------------------------------

enableMultiMetrics: # [boolean] Output multiple metrics - Output metrics in multi
enableACK: # [boolean] Minimize in-flight data loss - Check if indexer is shuttin

# -------------- if enableACK is false ---------------

maxFailedHealthChecks: # [number] Failed health check limit - Maximum number of t

# ----------------------------------------------------------

logFailedRequests: # [boolean] Log failed requests to disk - Use to troubleshoot
maxS2Sversion: # [string] Max S2S version - The highest S2S protocol version to a

# -------------- if maxS2Sversion is v4 ---------------

compress: # [string] Compression - Controls whether the sender should send compre

# ----------------------------------------------------------

onBackpressure: # [string] Backpressure behavior - How to handle events when all
```

```
# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# ------------------------------------------------------

indexerDiscovery: # [boolean] Indexer Discovery - Automatically discover indexers

# -------------- if indexerDiscovery is true ---------------

indexerDiscoveryConfigs: # [object]  - List of configurations to set up indexer d
  site: # [string] [required] Site - Clustering site of the indexers from where i
  masterUri: # [string] Cluster manager URI - Full URI of Splunk cluster manager
  refreshIntervalSec: # [number; minimum: 60, maximum: 86400] [required] Refresh
  rejectUnauthorized: # [boolean] Validate cluster manager certificates - During
  authTokens: # [array] Authentication tokens - Tokens required to authenticate t
    - authType: # [string] Authentication method - Select Manual to enter an auth
  authType: # [string] Authentication method - Select Manual to enter an auth tok
  authToken: # [string] Auth token - Shared secret to be provided by any client (

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored tex

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------


# ------------------------------------------------------


# -------------- if indexerDiscovery is false ---------------
```

```
  excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the currer
  hosts: # [array] Destinations - Set of Splunk indexers to load-balance data to.
    - host: # [string] Address - The hostname of the receiver
      port: # [number; maximum: 65535] Port - The port to connect to on the provide
      tls: # [string] TLS - Whether to inherit TLS configs from group setting or di
      servername: # [string] TLS Servername - Servername to use if establishing a T
      weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
  loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec
  maxConcurrentSenders: # [number] Connection limit - Maximum number of concurrent

  # ------------------------------------------------------

  senderUnhealthyTimeAllowance: # [number; maximum: 60000] Endpoint health fluctuat
  authType: # [string] Authentication method - Select Manual to enter an auth token
  authToken: # [string] Auth token - Shared secret token to use when establishing a

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored text

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
splunk_hec_output: # [object]
  type: # [string] Output Type
  loadBalanced: # [boolean] Load balancing - Enable for optimal performance. Even i

  # -------------- if loadBalanced is false ---------------

  url: # [string] Splunk HEC Endpoint - URL to a Splunk HEC endpoint to send events
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
```

```
# -------------------------------------------------------


# -------------- if loadBalanced is true ---------------

excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the currer
urls: # [array] Splunk HEC Endpoints
  - url: # [string] HEC Endpoint - URL to a Splunk HEC endpoint to send events to
    weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec

# -------------------------------------------------------

nextQueue: # [string] Next Processing Queue - In the Splunk app, define which Spl
tcpRouting: # [string] Default _TCP_ROUTING - In the Splunk app, set the value of
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 2097152] Body size limit (KB
throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
enableMultiMetrics: # [boolean] Output multi-metrics - Output metrics in multiple
authType: # [string] Authentication method - Select Manual to enter an auth token

# -------------- if authType is manual ---------------

token: # [string] HEC Auth token - Splunk HEC authentication token

# -------------------------------------------------------


# -------------- if authType is secret ---------------
```

```
  textSecret: # [string] HEC Auth token (text secret) - Select or create a stored t

  # ------------------------------------------------------

  responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # ------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
tcpjson_output: # [object]
  type: # [string] Output Type
  loadBalanced: # [boolean] Load balancing - Use load-balanced destinations
```

```
# -------------- if loadBalanced is false ---------------

host: # [string] Address - The hostname of the receiver
port: # [number; maximum: 65535] Port - The port to connect to on the provided ho

# -------------------------------------------------------


# -------------- if loadBalanced is true ---------------

excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the curren
hosts: # [array] Destinations - Set of hosts to load-balance data to
  - host: # [string] Address - The hostname of the receiver
    port: # [number; maximum: 65535] Port - The port to connect to on the provide
    tls: # [string] TLS - Whether to inherit TLS configs from group setting or di
    servername: # [string] TLS Servername - Servername to use if establishing a T
    weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec
maxConcurrentSenders: # [number] Connection limit - Maximum number of concurrent

# -------------------------------------------------------

compression: # [string] Compression - Codec to use to compress the data before se
logFailedRequests: # [boolean] Log failed requests to disk - Use to troubleshoot
throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version
```

```
    # ------------------------------------------------------

  connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
  writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
  tokenTTLMinutes: # [number; minimum: 1, maximum: 60] Auth Token TTL minutes - The
  sendHeader: # [boolean] Send auth token in initial record - Upon connection, send
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  authType: # [string] Authentication method - Select Manual to enter an auth token
  authToken: # [string] Auth token - Optional authentication token to include as pa

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored text

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
wavefront_output: # [object]
  type: # [string] Output Type
  authType: # [string] Authentication method - Select Manual to enter an auth token
```

```
# -------------- if authType is manual ---------------

token: # [string] Auth token - WaveFront API authentication token (see [here](htt

# ----------------------------------------------------


# -------------- if authType is secret ---------------

textSecret: # [string] Auth token (text secret) - Select or create a stored text

# ----------------------------------------------------

domain: # [string] Domain name - WaveFront domain name, e.g. "longboard"
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th
```

```
    # ------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # -------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
signalfx_output: # [object]
  type: # [string] Output Type
  authType: # [string] Authentication method - Select Manual to enter an auth token

  # -------------- if authType is manual ---------------

  token: # [string] Auth token - SignalFx API access token (see [here](https://docs

  # -------------------------------------------------------


  # -------------- if authType is secret ---------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored text

  # -------------------------------------------------------

  realm: # [string] Realm - SignalFx realm name, e.g. "us0". For a complete list of
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean] Compress - Compress the payload body before sending
```

```
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------

description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
```

```
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
filesystem_output: # [object]
  type: # [string] Output Type
  destPath: # [string] Output location - Final destination for the output files
  stagePath: # [string] Staging location - Filesystem location in which to buffer f
  addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
  removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

  # -------------- if removeEmptyDirs is true ---------------

  emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

  # ----------------------------------------------------------

  partitionExpr: # [string] Partitioning expression - JavaScript expression definin
  format: # [string] Data format - Format of the output data

  # -------------- if format is json ---------------

  compress: # [string] Compression - Data compression format to apply to HTTP conte
  compressionLevel: # [string] Compression level - Compression level to apply befor

  # ----------------------------------------------------------


  # -------------- if format is parquet ---------------

  automaticSchema: # [boolean] Automatic schema - Automatically calculate the schem
  parquetVersion: # [string] Parquet version - Determines which data types are supp
  parquetDataPageVersion: # [string] Data page version - Serialization format of da
  parquetRowGroupLength: # [number; minimum: 1, maximum: 67108864] Group row limit
  parquetPageSize: # [string] Page size - Target memory size for page segments, suc
  shouldLogInvalidRows: # [boolean] Log invalid rows - Log up to 3 rows that Cribl
  keyValueMetadata: # [array] Metadata (optional) - The metadata of files the Desti
    - key: # [string] Key
      value: # [string] Value
  enableStatistics: # [boolean] Write statistics - Statistics profile an entire fil
  enableWritePageIndex: # [boolean] Write page indexes - One page index contains st
  enablePageChecksum: # [boolean] Write page checksum - Parquet tools can use the c

  # ----------------------------------------------------------
```

```
  baseFileName: # [string] File name prefix expression - JavaScript expression to c
  fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
  maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 1800] File open time limit (
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 1800] Idle time limit (sec) -
  maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
  headerLine: # [string] Header line - If set, this line will be written to the beg
  writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
  deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

  # -------------- if deadletterEnabled is true ---------------

  deadletterPath: # [string] Dead-letter location - Storage location for files that
  maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

  # ------------------------------------------------------

  onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
s3_output: # [object]
  type: # [string] Output Type
  bucket: # [string] S3 bucket name - Name of the destination S3 bucket. Must be a
  region: # [string] Region - Region where the S3 bucket is located
  awsSecretKey: # [string] Secret key - Secret key. This value can be a constant or
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key - This value can be a constant or a JavaScript e

  # ------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref
```

```
# ----------------------------------------------------------

endpoint: # [string] Endpoint - S3 service endpoint. If empty, defaults to the AW
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for S3 - Use Assume Role credentials to acce
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
stagePath: # [string] [required] Staging location - Filesystem location in which
addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
destPath: # [string] Key prefix - Prefix to prepend to files before uploading. Mu
objectACL: # [string] Object ACL - Object ACL to assign to uploaded objects
storageClass: # [string] Storage class - Storage class to select for uploaded obj
serverSideEncryption: # [string] Server-side encryption for uploaded objects

# -------------- if serverSideEncryption is aws:kms ---------------

kmsKeyId: # [string] KMS key ID - ID or ARN of the KMS customer-managed key to us

# ----------------------------------------------------------

removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

# -------------- if removeEmptyDirs is true ---------------

emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

# ----------------------------------------------------------

partitionExpr: # [string] Partitioning expression - JavaScript expression definir
format: # [string] Data format - Format of the output data

# -------------- if format is json ---------------

compress: # [string] Compression - Data compression format to apply to HTTP conte
compressionLevel: # [string] Compression level - Compression level to apply befor

# ----------------------------------------------------------


# -------------- if format is parquet ---------------
```

```
automaticSchema: # [boolean] Automatic schema - Automatically calculate the schem
parquetVersion: # [string] Parquet version - Determines which data types are supp
parquetDataPageVersion: # [string] Data page version - Serialization format of da
parquetRowGroupLength: # [number; minimum: 1, maximum: 67108864] Group row limit
parquetPageSize: # [string] Page size - Target memory size for page segments, suc
shouldLogInvalidRows: # [boolean] Log invalid rows - Log up to 3 rows that Cribl
keyValueMetadata: # [array] Metadata (optional) - The metadata of files the Desti
  - key: # [string] Key
    value: # [string] Value
enableStatistics: # [boolean] Write statistics - Statistics profile an entire fil
enableWritePageIndex: # [boolean] Write page indexes - One page index contains st
enablePageChecksum: # [boolean] Write page checksum - Parquet tools can use the c

# ------------------------------------------------------

baseFileName: # [string] File name prefix expression - JavaScript expression to c
fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
headerLine: # [string] Header line - If set, this line will be written to the beg
writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
onBackpressure: # [string] Backpressure behavior - How to handle events when all
deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

# -------------- if deadletterEnabled is true ---------------

deadletterPath: # [string] Dead-letter location - Storage location for files that
maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

# ------------------------------------------------------

onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
maxFileOpenTimeSec: # [number; minimum: 10, maximum: 86400] File open time limit
maxFileIdleTimeSec: # [number; minimum: 5, maximum: 86400] Idle time limit (sec)
maxConcurrentFileParts: # [number; minimum: 1, maximum: 10] Concurrent file parts
verifyPermissions: # [boolean] Verify if bucket exists - Disable if you can acces
maxClosingFilesToBackpressure: # [number; minimum: 10, maximum: 4200] Staging fil
description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
systemFields: # [array of strings] System fields - Fields to automatically add to
environment: # [string] Environment - Optionally, enable this config only on a sp
streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
```

```
azure_blob_output: # [object]
  type: # [string] Output Type
  containerName: # [string] Container name - The Azure Blob Storage container name.
  createContainer: # [boolean] Create container - Create the configured container i
  destPath: # [string] Blob prefix - Root directory prepended to path before uploac
  stagePath: # [string] [required] Staging location - Filesystem location in which
  addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
  maxConcurrentFileParts: # [number; minimum: 1, maximum: 10] Concurrent file parts
  removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

  # -------------- if removeEmptyDirs is true ---------------

  emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

  # ----------------------------------------------------------

  partitionExpr: # [string] Partitioning expression - JavaScript expression definir
  format: # [string] Data format - Format of the output data

  # -------------- if format is json ---------------

  compress: # [string] Compression - Data compression format to apply to HTTP conte
  compressionLevel: # [string] Compression level - Compression level to apply befor

  # ----------------------------------------------------------


  # -------------- if format is parquet ---------------

  automaticSchema: # [boolean] Automatic schema - Automatically calculate the schem
  parquetVersion: # [string] Parquet version - Determines which data types are supp
  parquetDataPageVersion: # [string] Data page version - Serialization format of da
  parquetRowGroupLength: # [number; minimum: 1, maximum: 67108864] Group row limit
  parquetPageSize: # [string] Page size - Target memory size for page segments, suc
  shouldLogInvalidRows: # [boolean] Log invalid rows - Log up to 3 rows that Cribl
  keyValueMetadata: # [array] Metadata (optional) - The metadata of files the Desti
    - key: # [string] Key
      value: # [string] Value
  enableStatistics: # [boolean] Write statistics - Statistics profile an entire fil
  enableWritePageIndex: # [boolean] Write page indexes - One page index contains st
  enablePageChecksum: # [boolean] Write page checksum - Parquet tools can use the c

  # ----------------------------------------------------------
```

```
baseFileName: # [string] File name prefix expression - JavaScript expression to c
fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
maxFileOpenTimeSec: # [number; minimum: 10, maximum: 1800] File open time limit (
maxFileIdleTimeSec: # [number; minimum: 5, maximum: 1800] Idle time limit (sec) -
maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
headerLine: # [string] Header line - If set, this line will be written to the beg
writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
onBackpressure: # [string] Backpressure behavior - How to handle events when all
deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

# -------------- if deadletterEnabled is true ---------------

deadletterPath: # [string] Dead-letter location - Storage location for files that
maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

# ------------------------------------------------------------

onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
authType: # [string] Authentication method

# -------------- if authType is manual ---------------

connectionString: # [string] Connection string - Enter your Azure Storage account

# ------------------------------------------------------------


# -------------- if authType is secret ---------------

textSecret: # [string] Connection string (text secret) - Select or create a store

# ------------------------------------------------------------


# -------------- if authType is clientSecret ---------------

storageAccountName: # [string] Storage account name - The name of your Azure stor
tenantId: # [string] Tenant ID - The service principal's tenant ID
clientId: # [string] Client ID - The service principal's client ID
azureCloud: # [string] Azure Cloud - The Azure cloud to use. Defaults to Azure Pu
endpointSuffix: # [string] Endpoint suffix - Endpoint suffix for the service URL.
```

```
  clientTextSecret: # [string] Client secret (text secret) - Select or create a stc

  # -------------------------------------------------------


  # -------------- if authType is clientCert ---------------

  storageAccountName: # [string] Storage account name - The name of your Azure stor
  tenantId: # [string] Tenant ID - The service principal's tenant ID
  clientId: # [string] Client ID - The service principal's client ID
  azureCloud: # [string] Azure Cloud - The Azure cloud to use. Defaults to Azure Pu
  endpointSuffix: # [string] Endpoint suffix - Endpoint suffix for the service URL.
  certificate: # [object]
    certificateName: # [string] Certificate - The certificate you registered as cre

  # -------------------------------------------------------

  storageClass: # [string] Blob access tier
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
azure_data_explorer_output: # [object]
  type: # [string] Output Type
  clusterUrl: # [string] Cluster base URI - The base URI for your cluster. Typicall
  database: # [string] [required] Database name - Name of the database containing t
  table: # [string] [required] Table name - Name of the table to ingest data into
  validateDatabaseSettings: # [boolean] Validate database settings - When saving or
  ingestMode: # [string] Ingestion mode

  # -------------- if ingestMode is batching ---------------

  ingestUrl: # [string] Ingestion service URI - The ingestion service URI for your
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
  isMappingObj: # [boolean] Add mapping object - Send a JSON mapping object instead
  format: # [string] Data format - Format of the output data
  stagePath: # [string] Staging location - Filesystem location in which to buffer f
  fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
  maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 1800] File open time limit (
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 1800] Idle time limit (sec) -
  maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
```

```
maxConcurrentFileParts: # [number; minimum: 1, maximum: 10] Concurrent file parts
onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag
deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushImmediately: # [boolean] Flush immediately - Bypass the data management serv
retainBlobOnSuccess: # [boolean] Retain blob on success - Prevent blob deletion a
extentTags: # [array] Extent tags - Strings or tags associated with the extent (i
  - prefix: # [string] Prefix (optional)
    value: # [string] Value
ingestIfNotExists: # [array] Enforce uniqueness via tag values - Prevents duplica
  - value: # [string] Value
reportLevel: # [string] Report level - Level of ingestion status reporting. Defau
reportMethod: # [string] Report method - Target of the ingestion status reporting
additionalProperties: # [array] Additional fields - Optionally, enter additional
  - key: # [string] Key
    value: # [string] Value

# ------------------------------------------------------


# -------------- if ingestMode is streaming ---------------

responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all
format: # [string] Data format - Format of the output data
```

```
compress: # [string] [required] Compression - Data compression format to apply to
mappingRef: # [string] Data mapping - Enter the name of a data mapping associated
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 4096] Body size limit (KB) -
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
keepAlive: # [boolean] Keep alive - Disable to close the connection immediately a

# ------------------------------------------------------

oauthEndpoint: # [string] [required] Microsoft Entra ID authentication endpoint -
tenantId: # [string] [required] Tenant ID - Directory ID (tenant identifier) in A
clientId: # [string] [required] Client ID - client_id to pass in the OAuth reques
scope: # [string] [required] Scope - Scope to pass in the OAuth request parameter
oauthType: # [string] [required] Authentication method - The type of OAuth 2.0 cl

# -------------- if oauthType is clientSecret ---------------

clientSecret: # [string] Client secret - The client secret that you generated for

# ------------------------------------------------------


# -------------- if oauthType is clientTextSecret ---------------

textSecret: # [string] Client secret (text secret) - Select or create a stored te

# ------------------------------------------------------


# -------------- if oauthType is certificate ---------------

certificate: # [object]
  certificateName: # [string] Certificate - The certificate you registered as cre

# ------------------------------------------------------

advancedSettingsNote: # [null]
```

```
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
azure_logs_output: # [object]
  type: # [string] Output Type
  logType: # [string] Log Type - The Log Type of events sent to this LogAnalytics w
  resourceId: # [string] Resource ID - Optional Resource ID of the Azure resource t
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1, maximum: 10240] Body size limit (KB) - M
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean]
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  apiUrl: # [string] DNS name of API endpoint - The DNS name of the Log API endpoin
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automatical
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # -----------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
```

```
  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ---------------------------------------------------------

  authType: # [string] Authentication method - Enter workspace ID and workspace key
  workspaceId: # [string] Workspace ID - Azure Log Analytics Workspace ID. See Azur
  workspaceKey: # [string] Workspace key - Azure Log Analytics Workspace Primary or

  # -------------- if authType is manual ---------------


  # ---------------------------------------------------------

  keypairSecret: # [string] Secret key pair - Select or create a stored secret that

  # -------------- if authType is secret ---------------


  # ---------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
kinesis_output: # [object]
  type: # [string] Output Type
  streamName: # [string] Stream Name - Kinesis stream name to send events to.
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key

  # ---------------------------------------------------------
```

```
# -------------- if awsAuthenticationMethod is secret ---------------

awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

# ------------------------------------------------------

awsSecretKey: # [string] Secret key
region: # [string] [required] Region - Region where the Kinesis stream is located
endpoint: # [string] Endpoint - Kinesis stream service endpoint. If empty, defaul
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for Kinesis stream - Use Assume Role credent
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
concurrency: # [number; minimum: 1, maximum: 32] Put request concurrency - Maximu
maxRecordSizeKB: # [number; minimum: 1, maximum: 10240] Record size limit (KB, ur
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
compression: # [string] Compression - Compression type to use for records

# -------------- if compression is none ---------------

asNdjson: # [boolean] Send batched - Batch events into a single record as NDJSON

# ------------------------------------------------------

useListShards: # [boolean] ListShards API - Provides higher stream rate limits, i
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# ------------------------------------------------------
```

```
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
honeycomb_output: # [object]
  type: # [string] Output Type
  dataset: # [string] Dataset name - Name of the dataset to send events to - e.g.,
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean] Compress - Compress the payload body before sending
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automatical
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # -------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------
```

```
  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  authType: # [string] Authentication method - Enter API key directly, or select a
  team: # [string] API key - Team API key where the dataset belongs

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] API key (text secret) - Select or create a stored text sec

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
azure_eventhub_output: # [object]
  type: # [string] Output Type
  brokers: # [array of strings] Brokers - List of Event Hubs Kafka brokers to conne
  topic: # [string] [required] Event Hub name - The name of the Event Hub (Kafka To
  ack: # [integer] Acknowledgments - Control the number of required acknowledgments
  format: # [string] Record data format - Format to use to serialize events before
  maxRecordSizeKB: # [number; minimum: 1] Record size limit (KB, uncompressed) - Ma
  flushEventCount: # [number; minimum: 1, maximum: 10000] Events-per-batch limit -
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
  requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
  maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
  maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
```

```
  initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
  backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
  authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
  reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
  sasl: # [object] Authentication - Authentication parameters to use when connectin
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    mechanism: # [string] SASL mechanism

    # ---------------------------------------------------

  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha

    # ---------------------------------------------------

  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ---------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
google_chronicle_output: # [object]
  type: # [string] Output Type
  apiVersion: # [string] API version
```

```
# -------------- if apiVersion is v1 ---------------

authenticationMethod: # [string] Authentication method

# --------------------------------------------------------


# -------------- if apiVersion is v2 ---------------

authenticationMethod: # [string] Authentication method

# --------------------------------------------------------


authenticationMethod: # [string] Authentication method

# -------------- if authenticationMethod is manual ---------------

apiKey: # [string] API key - Organization's API key in Google SecOps

# --------------------------------------------------------


# -------------- if authenticationMethod is secret ---------------

apiKeySecret: # [string] API key (text secret) - Select or create a stored text s

# --------------------------------------------------------


# -------------- if authenticationMethod is serviceAccount ---------------

serviceAccountCredentials: # [string] Service account credentials - Contents of s

# --------------------------------------------------------


# -------------- if authenticationMethod is serviceAccountSecret ---------------

serviceAccountCredentialsSecret: # [string] Service account credentials (text sec

# --------------------------------------------------------
```

```
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
logFormatType: # [string] Send events as

# -------------- if logFormatType is unstructured ---------------

extraLogTypes: # [array] Custom log types - Custom log types. If the value "Custo
  - logType: # [string] Log Type
    description: # [string] Description
logType: # [string] Default log type - Default log type value to send to SecOps.
logTextField: # [string] Log text field - Name of the event field that contains t
customerId: # [string] Customer ID - A unique identifier (UUID) for your Google S
namespace: # [string] Namespace - User-configured environment namespace to identi
customLabels: # [array] Custom labels - Custom labels to be added to every batch
  - key: # [string] Key
    value: # [string] Value

# -------------------------------------------------------

region: # [string] Region - Regional endpoint to send events to
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1, maximum: 1024] Body size limit (KB) - Ma
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
```

```
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ---------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
google_cloud_storage_output: # [object]
  type: # [string] Output Type
  bucket: # [string] Bucket name - Name of the destination bucket. This value can b
  region: # [string] [required] Region - Region where the bucket is located
  endpoint: # [string] [required] Endpoint - Google Cloud Storage service endpoint
  signatureVersion: # [string] Signature version - Signature version to use for sig
  awsAuthenticationMethod: # [string] Authentication method

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key - HMAC access key. This value can be a constant
  awsSecretKey: # [string] Secret - HMAC secret. This value can be a constant or a

  # ---------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------
```

```
awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

# ------------------------------------------------------

stagePath: # [string] [required] Staging location - Filesystem location in which
destPath: # [string] Key prefix - Prefix to prepend to files before uploading. Mu
verifyPermissions: # [boolean] Verify if bucket exists - Disable if you can acces
objectACL: # [string] Object ACL - Object ACL to assign to uploaded objects
storageClass: # [string] Storage class - Storage class to select for uploaded obj
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

# -------------- if removeEmptyDirs is true ---------------

emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

# ------------------------------------------------------

partitionExpr: # [string] Partitioning expression - JavaScript expression definin
format: # [string] Data format - Format of the output data

# -------------- if format is json ---------------

compress: # [string] Compression - Data compression format to apply to HTTP conte
compressionLevel: # [string] Compression level - Compression level to apply befor

# ------------------------------------------------------


# -------------- if format is parquet ---------------

automaticSchema: # [boolean] Automatic schema - Automatically calculate the schem
parquetVersion: # [string] Parquet version - Determines which data types are supp
parquetDataPageVersion: # [string] Data page version - Serialization format of da
parquetRowGroupLength: # [number; minimum: 1, maximum: 67108864] Group row limit
parquetPageSize: # [string] Page size - Target memory size for page segments, suc
shouldLogInvalidRows: # [boolean] Log invalid rows - Log up to 3 rows that Cribl
keyValueMetadata: # [array] Metadata (optional) - The metadata of files the Desti
  - key: # [string] Key
    value: # [string] Value
```

```
  enableStatistics: # [boolean] Write statistics - Statistics profile an entire fil
  enableWritePageIndex: # [boolean] Write page indexes - One page index contains st
  enablePageChecksum: # [boolean] Write page checksum - Parquet tools can use the c

  # ------------------------------------------------------

  baseFileName: # [string] File name prefix expression - JavaScript expression to c
  fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
  maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 1800] File open time limit (
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 1800] Idle time limit (sec) -
  maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
  headerLine: # [string] Header line - If set, this line will be written to the beg
  writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
  deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

  # -------------- if deadletterEnabled is true ---------------

  deadletterPath: # [string] Dead-letter location - Storage location for files that
  maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

  # ------------------------------------------------------

  onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
google_cloud_logging_output: # [object]
  type: # [string] Output Type
  logLocationType: # [string] Log location type

  # -------------- if logLocationType is project ---------------

  logLocationExpression: # [string] [required] Project ID expression - JavaScript e

  # ------------------------------------------------------


  # -------------- if logLocationType is organization ---------------
```

```
logLocationExpression: # [string] [required] Organization ID expression - JavaScr

# ---------------------------------------------------------


# -------------- if logLocationType is billingAccount ---------------

logLocationExpression: # [string] [required] Billing account ID expression - Java

# ---------------------------------------------------------


# -------------- if logLocationType is folder ---------------

logLocationExpression: # [string] [required] Folder ID expression - JavaScript ex

# ---------------------------------------------------------

logNameExpression: # [string] [required] Log name expression - JavaScript express
payloadFormat: # [string] Payload format - Format to use when sending payload. De

# -------------- if payloadFormat is text ---------------

payloadExpression: # [string] Payload text expression - JavaScript expression to

# ---------------------------------------------------------


# -------------- if payloadFormat is json ---------------

payloadExpression: # [string] Payload object expression - JavaScript expression t

# ---------------------------------------------------------

logLabels: # [array] Log labels - Labels to apply to the log entry
  - label: # [string] Label - Label name
    valueExpression: # [string] Value - JavaScript expression to compute the labe
resourceTypeExpression: # [string] Resource type expression - JavaScript expressi
resourceTypeLabels: # [array] Resource labels - Labels to apply to the managed re
  - label: # [string] Label - Label name
    valueExpression: # [string] Value - JavaScript expression to compute the labe
severityExpression: # [string] Severity expression - JavaScript expression to com
insertIdExpression: # [string] Insert ID expression - JavaScript expression to co
```

```
googleAuthMethod: # [string] Google authentication method - Choose Auto to use Go

# -------------- if googleAuthMethod is manual ---------------

serviceAccountCredentials: # [string] Service account credentials - Contents of s

# ------------------------------------------------------


# -------------- if googleAuthMethod is secret ---------------

secret: # [string] Service account credentials (text secret) - Select or create a

# ------------------------------------------------------


maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Max number of events to i
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
throttleRateReqPerSec: # [integer; maximum: 2000] Throttle request rate - Maximum
requestMethodExpression: # [string] Request method expression - A JavaScript expr
requestUrlExpression: # [string] Request URL expression - A JavaScript expressior
requestSizeExpression: # [string] Request size expression - A JavaScript expressi
statusExpression: # [string] Request status expression - A JavaScript expression
responseSizeExpression: # [string] Response size expression - A JavaScript expres
userAgentExpression: # [string] Request user agent expression - A JavaScript expr
remoteIpExpression: # [string] Remote IP expression - A JavaScript expression tha
serverIpExpression: # [string] Server IP expression - A JavaScript expression tha
refererExpression: # [string] Referer expression - A JavaScript expression that e
latencyExpression: # [string] Latency expression - A JavaScript expression that e
cacheLookupExpression: # [string] Cache lookup expression - A JavaScript expressi
cacheHitExpression: # [string] Cache hit expression - A JavaScript expression tha
cacheValidatedExpression: # [string] Cache validated with origin server expressio
cacheFillBytesExpression: # [string] Cache fill bytes expression - A JavaScript e
protocolExpression: # [string] Protocol expression - A JavaScript expression that
idExpression: # [string] ID expression - A JavaScript expression that evaluates t
producerExpression: # [string] Producer expression - A JavaScript expression that
firstExpression: # [string] First expression - A JavaScript expression that evalu
lastExpression: # [string] Last expression - A JavaScript expression that evaluat
fileExpression: # [string] File expression - A JavaScript expression that evaluat
lineExpression: # [string] Line expression - A JavaScript expression that evaluat
```

```
  functionExpression: # [string] Function expression - A JavaScript expression that
  uidExpression: # [string] UID expression - A JavaScript expression that evaluates
  indexExpression: # [string] Index expression - A JavaScript expression that evalu
  totalSplitsExpression: # [string] Total splits expression - A JavaScript expressi
  traceExpression: # [string] Trace expression - A JavaScript expression that evalu
  spanIdExpression: # [string] Span ID expression - A JavaScript expression that ev
  traceSampledExpression: # [string] Trace sampled expression - A JavaScript expres
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ---------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
google_pubsub_output: # [object]
  type: # [string] Output Type
  topicName: # [string] Topic ID - ID of the topic to send events to.
  createTopic: # [boolean] Create topic - If enabled, create topic if it does not e
  orderedDelivery: # [boolean] Ordered delivery - If enabled, send events in the or
  region: # [string] Region - Region to publish messages to. Select 'default' to al
  googleAuthMethod: # [string] Google authentication method - Choose Auto to use Go

  # -------------- if googleAuthMethod is manual ---------------

  serviceAccountCredentials: # [string] Service account credentials - Contents of s

  # ---------------------------------------------------------


  # -------------- if googleAuthMethod is secret ---------------
```

```
  secret: # [string] Service account credentials (text secret) - Select or create a

  # ------------------------------------------------------

  batchSize: # [number; minimum: 1, maximum: 10000] Batch size - The maximum number
  batchTimeout: # [number; minimum: 1, maximum: 100000] Batch timeout (ms) - The ma
  maxQueueSize: # [number; minimum: 1] Queue size limit - Maximum number of queued
  maxRecordSizeKB: # [number; minimum: 1, maximum: 256] Batch size limit (KB) - Max
  flushPeriod: # [number] Flush period (sec) - Maximum time to wait before sending
  maxInProgress: # [number; minimum: 1, maximum: 100] Concurrent request limit - Th
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
exabeam_output: # [object]
  bucket: # [string] Bucket name - Name of the destination bucket. A constant or a
  region: # [string] [required] Region - Region where the bucket is located
  stagePath: # [string] [required] Staging location - Filesystem location in which
  endpoint: # [string] [required] Endpoint - Google Cloud Storage service endpoint
  signatureVersion: # [string] Signature version - Signature version to use for sig
  objectACL: # [string] Object ACL - Object ACL to assign to uploaded objects
  storageClass: # [string] Storage class - Storage class to select for uploaded obj
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
  addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
  removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

  # -------------- if removeEmptyDirs is true ---------------
```

```
  emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

  # ------------------------------------------------------

  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 1800] File open time limit (
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 1800] Idle time limit (sec) -
  maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
  deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

  # -------------- if deadletterEnabled is true ---------------

  deadletterPath: # [string] Dead-letter location - Storage location for files that
  maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

  # ------------------------------------------------------

  onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
  type: # [string] Output Type
  maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
  encodedConfiguration: # [string] Exabeam connection string - Enter an encoded str
  collectorInstanceId: # [string] [required] Collector instance ID - ID of the Exab

  siteName: # [string] Site name - Constant or JavaScript expression to create an E
  siteId: # [string] Site ID - Exabeam site ID. If left blank, Cribl Stream will us
  timezoneOffset: # [string] Timezone offset
  awsApiKey: # [string] Access key - HMAC access key. Can be a constant or a JavaSc
  awsSecretKey: # [string] Secret - HMAC secret. Can be a constant or a JavaScript
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
kafka_output: # [object]
  type: # [string] Output Type
  brokers: # [array of strings] Bootstrap servers - Enter each Kafka bootstrap serv
  topic: # [string] [required] Topic - The topic to publish events to. Can be overr
  ack: # [integer] Acknowledgments - Control the number of required acknowledgments
  format: # [string] Record data format - Format to use to serialize events before

  # -------------- if format is protobuf ---------------

  protobufLibraryId: # [string] Definition set - Select a set of Protobuf definitio
```

```
# -----------------------------------------------------

compression: # [string] Compression - Codec to use to compress the data before se
maxRecordSizeKB: # [number; minimum: 1] Record size limit (KB, uncompressed) - Ma
flushEventCount: # [number; minimum: 1, maximum: 10000] Events-per-batch limit -
flushPeriodSec: # [number] Flush period (sec) - The maximum amount of time you wa
kafkaSchemaRegistry: # [object] Kafka Schema Registry Authentication
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  schemaRegistryURL: # [string] Schema Registry URL - URL for accessing the Confl
  schemaType: # [string] Schema type - The schema format used to encode and decoc
  connectionTimeout: # [number; minimum: 1000, maximum: 60000] Connection timeout
  requestTimeout: # [number; minimum: 1000, maximum: 60000] Request timeout (ms)
  maxRetries: # [number; maximum: 100] Retry limit - Maximum number of times to t
  auth: # [object]  - Credentials to use when authenticating with the schema regi
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    credentialsSecret: # [string] Credentials secret - Select or create a secret

    # -------------------------------------------------------

  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates t
                trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Na
    certificateName: # [string] Certificate - The name of the predefined certific
    caPath: # [string] CA certificate path - Path on client in which to find CA c
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in wh
    certPath: # [string] Certificate path (mutual auth) - Path on client in which
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # -------------------------------------------------------
```

```
  defaultKeySchemaId: # [number] Default key schema ID - Used when __keySchemaIdC
  defaultValueSchemaId: # [number] Default value schema ID - Used when __valueSch

  # ------------------------------------------------------

connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
sasl: # [object] Authentication - Authentication parameters to use when connectir
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  mechanism: # [string] SASL mechanism
  oauthEnabled: # [boolean] Enable OAuth - Enable OAuth authentication

  # ------------------------------------------------------

tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------

onBackpressure: # [string] Backpressure behavior - How to handle events when all
```

```
  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ---------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
confluent_cloud_output: # [object]
  type: # [string] Output Type
  brokers: # [array of strings] Bootstrap servers - List of Confluent Cloud bootstr
  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                  trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
    certificateName: # [string] Certificate - The name of the predefined certificat
    caPath: # [string] CA certificate path - Path on client in which to find CA cer
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
    certPath: # [string] Certificate path (mutual auth) - Path on client in which t
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ---------------------------------------------------------

  topic: # [string] [required] Topic - The topic to publish events to. Can be overr
  ack: # [integer] Acknowledgments - Control the number of required acknowledgments
  format: # [string] Record data format - Format to use to serialize events before

  # -------------- if format is protobuf ---------------
```

```
protobufLibraryId: # [string] Definition set - Select a set of Protobuf definitic

# ------------------------------------------------------

compression: # [string] Compression - Codec to use to compress the data before se
maxRecordSizeKB: # [number; minimum: 1] Record size limit (KB, uncompressed) - Ma
flushEventCount: # [number; minimum: 1, maximum: 10000] Events-per-batch limit -
flushPeriodSec: # [number] Flush period (sec) - The maximum amount of time you wa
kafkaSchemaRegistry: # [object] Kafka Schema Registry Authentication
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  schemaRegistryURL: # [string] Schema Registry URL - URL for accessing the Confl
  schemaType: # [string] Schema type - The schema format used to encode and decoc
  connectionTimeout: # [number; minimum: 1000, maximum: 60000] Connection timeout
  requestTimeout: # [number; minimum: 1000, maximum: 60000] Request timeout (ms)
  maxRetries: # [number; maximum: 100] Retry limit - Maximum number of times to t
  auth: # [object]  - Credentials to use when authenticating with the schema regi
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    credentialsSecret: # [string] Credentials secret - Select or create a secret

    # ------------------------------------------------------

  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates t
                trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Na
    certificateName: # [string] Certificate - The name of the predefined certific
    caPath: # [string] CA certificate path - Path on client in which to find CA c
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in wh
    certPath: # [string] Certificate path (mutual auth) - Path on client in which
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version
```

```
    # ------------------------------------------------------

  defaultKeySchemaId: # [number] Default key schema ID - Used when __keySchemaIdC
  defaultValueSchemaId: # [number] Default value schema ID - Used when __valueSch

  # ------------------------------------------------------

connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
sasl: # [object] Authentication - Authentication parameters to use when connectir
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  mechanism: # [string] SASL mechanism
  oauthEnabled: # [boolean] Enable OAuth - Enable OAuth authentication

  # ------------------------------------------------------

onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------

description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
systemFields: # [array of strings] System fields - Fields to automatically add to
environment: # [string] Environment - Optionally, enable this config only on a sp
```

```
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
msk_output: # [object]
  type: # [string] Output Type
  brokers: # [array of strings] Bootstrap servers - Enter each Kafka bootstrap serv
  topic: # [string] [required] Topic - The topic to publish events to. Can be overr
  ack: # [integer] Acknowledgments - Control the number of required acknowledgments
  format: # [string] Record data format - Format to use to serialize events before

  # -------------- if format is protobuf ---------------

  protobufLibraryId: # [string] Definition set - Select a set of Protobuf definitio

  # ------------------------------------------------------

  compression: # [string] Compression - Codec to use to compress the data before se
  maxRecordSizeKB: # [number; minimum: 1] Record size limit (KB, uncompressed) - Ma
  flushEventCount: # [number; minimum: 1, maximum: 10000] Events-per-batch limit -
  flushPeriodSec: # [number] Flush period (sec) - The maximum amount of time you wa
  kafkaSchemaRegistry: # [object] Kafka Schema Registry Authentication
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------

    schemaRegistryURL: # [string] Schema Registry URL - URL for accessing the Confl
    schemaType: # [string] Schema type - The schema format used to encode and decod
    connectionTimeout: # [number; minimum: 1000, maximum: 60000] Connection timeout
    requestTimeout: # [number; minimum: 1000, maximum: 60000] Request timeout (ms)
    maxRetries: # [number; maximum: 100] Retry limit - Maximum number of times to t
    auth: # [object]  - Credentials to use when authenticating with the schema regi
      disabled: # [boolean] Disabled

      # -------------- if disabled is false ---------------

      credentialsSecret: # [string] Credentials secret - Select or create a secret

      # ------------------------------------------------------

    tls: # [object] TLS settings (client side)
      disabled: # [boolean] Disabled

      # -------------- if disabled is false ---------------

      rejectUnauthorized: # [boolean] Validate server certs - Reject certificates t
```

```
                trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Na
    certificateName: # [string] Certificate - The name of the predefined certific
    caPath: # [string] CA certificate path - Path on client in which to find CA c
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in wh
    certPath: # [string] Certificate path (mutual auth) - Path on client in which
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  defaultKeySchemaId: # [number] Default key schema ID - Used when __keySchemaIdC
  defaultValueSchemaId: # [number] Default value schema ID - Used when __valueSch

  # ------------------------------------------------------

connectionTimeout: # [number; minimum: 1000, maximum: 3600000] Connection timeout
requestTimeout: # [number; minimum: 1000, maximum: 3600000] Request timeout (ms)
maxRetries: # [number; maximum: 100] Retry limit - If messages are failing, you c
maxBackOff: # [number; minimum: 30000, maximum: 180000] Backoff limit (ms) - The
initialBackoff: # [number; minimum: 300, maximum: 600000] Initial retry interval
backoffRate: # [number; minimum: 2, maximum: 20] Backoff multiplier - Set the bac
authenticationTimeout: # [number; minimum: 1000, maximum: 3600000] Authentication
reauthenticationThreshold: # [number; minimum: 1000, maximum: 1800000] Reauthenti
awsAuthenticationMethod: # [string] [required] Authentication method - AWS auther

# -------------- if awsAuthenticationMethod is manual ---------------

awsApiKey: # [string] Access key

# ------------------------------------------------------


# -------------- if awsAuthenticationMethod is secret ---------------

awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

# ------------------------------------------------------

awsSecretKey: # [string] Secret key
region: # [string] [required] Region - Region where the MSK cluster is located
endpoint: # [string] Endpoint - MSK cluster service endpoint. If empty, defaults
```

```
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for MSK - Use Assume Role credentials to acc
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ---------------------------------------------------------

onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -----------------------------------------------------------

description: # [string] Description
pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
systemFields: # [array of strings] System fields - Fields to automatically add to
environment: # [string] Environment - Optionally, enable this config only on a sp
streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
```

```
elastic_output: # [object]
  type: # [string] Output Type
  loadBalanced: # [boolean] Load balancing - Enable for optimal performance. Even i

  # -------------- if loadBalanced is false ---------------

  url: # [string] Bulk API URL or Cloud ID - The Cloud ID or URL to an Elastic clus
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh

  # ------------------------------------------------------



  # -------------- if loadBalanced is true ---------------

  excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the curren
  urls: # [array] Bulk API URLs
    - url: # [string] URL - The URL to an Elastic node to send events to. Example:
      weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
  loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec

  # ------------------------------------------------------

  index: # [string] Index or data stream - Index or data stream to send events to.
  docType: # [string] Type - Document type to use for events. Can be overwritten by
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1024, maximum: 102400] Body size limit (KB)
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean] Compress - Compress the payload body before sending
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
```

```
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any F
extraParams: # [array] Extra parameters
  - name: # [string] Field Name
    value: # [string] Field Value
auth: # [object]
  disabled: # [boolean] Authentication Disabled

  # -------------- if disabled is false ---------------

  authType: # [string] Authentication method - Enter credentials directly, or sel

  # -------------------------------------------------------

elasticVersion: # [string] Elastic version - Optional Elasticsearch version, usec
elasticPipeline: # [string] Elastic pipeline - Optional Elasticsearch destinatior
includeDocId: # [boolean] Include document _id - Include the `document_id` field
writeAction: # [string] Write action - Action to use when writing events. Must be
retryPartialErrors: # [boolean] Retry partial errors - Retry failed events when a
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------
```

```
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
elastic_cloud_output: # [object]
  type: # [string] Output Type
  url: # [string] Cloud ID - Enter Cloud ID of the Elastic Cloud environment to ser
  index: # [string] [required] Data stream or index - Data stream or index to send
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1024, maximum: 102400] Body size limit (KB)
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean] Compress - Compress the payload body before sending
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  extraParams: # [array] Extra parameters - Extra parameters to use in HTTP request
    - name: # [string] Field Name
      value: # [string] Field Value
  auth: # [object]
    disabled: # [boolean] Authentication Disabled

    # -------------- if disabled is false ---------------

    authType: # [string] Authentication method - Enter credentials directly, or sel

    # ---------------------------------------------------

  elasticPipeline: # [string] Elastic pipeline - Optional Elastic Cloud Destinatior
  includeDocId: # [boolean] Include document _id - Include the `document_id` field
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
```

```
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # -------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # -------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
newrelic_output: # [object]
  type: # [string] Output Type
  region: # [string] Region - Which New Relic region endpoint to use.

  # -------------- if region is Custom ---------------

  customUrl: # [string]

  # -------------------------------------------------------

  logType: # [string] Log type - Name of the logtype to send with events, e.g.: obs
  messageField: # [string] Log message field - Name of field to send as log message
  metadata: # [array] Fields - Fields to add to events from this input
    - name: # [string] Field Name
```

```
    value: # [string] Value - JavaScript expression to compute field's value, enc
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1, maximum: 1024] Body size limit (KB) - Ma
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th
```

```
  # ------------------------------------------------------

  authType: # [string] Authentication method - Enter API key directly, or select a
  apiKey: # [string] API key - New Relic API key. Can be overridden using __newReli

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] API key (text secret) - Select or create a stored text sec

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
newrelic_events_output: # [object]
  type: # [string] Output Type
  region: # [string] Region - Which New Relic region endpoint to use.

  # -------------- if region is Custom ---------------

  customUrl: # [string]

  # ------------------------------------------------------

  accountId: # [string] Account ID - New Relic account ID
  eventType: # [string] [required] Event type - Default eventType to use when not p
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1, maximum: 1024] Body size limit (KB) - Ma
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean] Compress - Compress the payload body before sending
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
```

```
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# ------------------------------------------------------

authType: # [string] Authentication method - Enter API key directly, or select a
apiKey: # [string] API key - New Relic API key. Can be overridden using __newReli

# -------------- if authType is manual ---------------
```

```
  # ------------------------------------------------------

  textSecret: # [string] API key (text secret) - Select or create a stored text sec

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
influxdb_output: # [object]
  type: # [string] Output Type
  url: # [string] Write API URL - URL of an InfluxDB cluster to send events to, e.g
  useV2API: # [boolean] Use v2 API - The v2 API can be enabled with InfluxDB versio

  # -------------- if useV2API is false ---------------

  database: # [string] Database - Database to write to.

  # ------------------------------------------------------


  # -------------- if useV2API is true ---------------

  bucket: # [string] Bucket - Bucket to write to.
  org: # [string] Organization - Organization ID for this bucket.

  # ------------------------------------------------------

  timestampPrecision: # [string] Timestamp precision - Sets the precision for the s
  dynamicValueFieldName: # [boolean] Dynamic value fields - Enabling this will pull
  valueFieldName: # [string] Value field name - Name of the field in which to store
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1024, maximum: 51200] Body size limit (KB)
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean] Compress - Compress the payload body before sending
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
```

```
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any F
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------

authType: # [string] Authentication type - InfluxDB authentication type

# -------------- if authType is basic ---------------
```

```
username: # [string] Username
password: # [string] Password

# ------------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# ------------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ------------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# ------------------------------------------------------


# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth l
  - name: # [string] Name - OAuth header name
    value: # [string] Value - OAuth header value
```

```
  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
cloudwatch_output: # [object]
  type: # [string] Output Type
  logGroupName: # [string] Log group name - CloudWatch log group to associate event
  logStreamName: # [string] [required] Log stream prefix - Prefix for CloudWatch lo
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key

  # ------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

  # ------------------------------------------------------

  awsSecretKey: # [string] Secret key
  region: # [string] [required] Region - Region where the CloudWatchLogs is located
  endpoint: # [string] Endpoint - CloudWatchLogs service endpoint. If empty, defaul
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
  enableAssumeRole: # [boolean] Enable for CloudWatchLogs - Use Assume Role credent
  assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
  assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
  durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
  maxQueueSize: # [number; minimum: 1, maximum: 32] Queue size limit - Maximum numb
  maxRecordSizeKB: # [number; minimum: 1, maximum: 10240] Record size limit (KB, ur
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
```

```
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
minio_output: # [object]
  type: # [string] Output Type
  endpoint: # [string] [required] MinIO endpoint - MinIO service url (e.g. http://m
  bucket: # [string] MinIO bucket name - Name of the destination MinIO bucket. This
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key - This value can be a constant or a JavaScript e

  # ------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

  # ------------------------------------------------------

  awsSecretKey: # [string] Secret key - Secret key. This value can be a constant or
  region: # [string] Region - Region where the MinIO service/cluster is located
  stagePath: # [string] [required] Staging location - Filesystem location in which
  addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
  destPath: # [string] Key prefix - Root directory to prepend to path before upload
  signatureVersion: # [string] Signature version - Signature version to use for sig
  objectACL: # [string] Object ACL - Object ACL to assign to uploaded objects
  storageClass: # [string] Storage class - Storage class to select for uploaded obj
  serverSideEncryption: # [string] Server-side encryption - Server-side encryption
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
```

```
verifyPermissions: # [boolean] Verify if bucket exists - Disable if you can acces
removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

# -------------- if removeEmptyDirs is true ---------------

emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

# ---------------------------------------------------------

partitionExpr: # [string] Partitioning expression - JavaScript expression definin
format: # [string] Data format - Format of the output data

# -------------- if format is json ---------------

compress: # [string] Compression - Data compression format to apply to HTTP conte
compressionLevel: # [string] Compression level - Compression level to apply befor

# ---------------------------------------------------------


# -------------- if format is parquet ---------------

automaticSchema: # [boolean] Automatic schema - Automatically calculate the schem
parquetVersion: # [string] Parquet version - Determines which data types are supp
parquetDataPageVersion: # [string] Data page version - Serialization format of da
parquetRowGroupLength: # [number; minimum: 1, maximum: 67108864] Group row limit
parquetPageSize: # [string] Page size - Target memory size for page segments, suc
shouldLogInvalidRows: # [boolean] Log invalid rows - Log up to 3 rows that Cribl
keyValueMetadata: # [array] Metadata (optional) - The metadata of files the Desti
  - key: # [string] Key
    value: # [string] Value
enableStatistics: # [boolean] Write statistics - Statistics profile an entire fil
enableWritePageIndex: # [boolean] Write page indexes - One page index contains st
enablePageChecksum: # [boolean] Write page checksum - Parquet tools can use the c

# ---------------------------------------------------------

baseFileName: # [string] File name prefix expression - JavaScript expression to c
fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
headerLine: # [string] Header line - If set, this line will be written to the beg
writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
```

```
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
  deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

  # -------------- if deadletterEnabled is true ---------------

  deadletterPath: # [string] Dead-letter location - Storage location for files that
  maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

  # ------------------------------------------------------

  onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 86400] File open time limit
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 86400] Idle time limit (sec)
  maxConcurrentFileParts: # [number; minimum: 1, maximum: 10] Concurrent file parts
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
statsd_output: # [object]
  type: # [string] Output Type
  protocol: # [string] Destination protocol - Protocol to use when communicating wi

  # -------------- if protocol is udp ---------------

  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (sec) - How

  # ------------------------------------------------------


  # -------------- if protocol is tcp ---------------

  throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
  connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
  writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # ------------------------------------------------------

  host: # [string] [required] Host - The hostname of the destination.
  port: # [number; minimum: 1, maximum: 65535] [required] Port - Destination port.
  mtu: # [number; minimum: 1, maximum: 65535] Record size limit (bytes) - When prot
  flushPeriodSec: # [number] Flush period (sec) - When protocol is TCP, specifies h
```

```
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
statsd_ext_output: # [object]
  type: # [string] Output Type
  protocol: # [string] Destination protocol - Protocol to use when communicating wi

  # -------------- if protocol is udp ---------------

  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (sec) - How

  # -------------------------------------------------------


  # -------------- if protocol is tcp ---------------

  throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
  connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
  writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------------------------------------------------

  host: # [string] [required] Host - The hostname of the destination.
  port: # [number; minimum: 1, maximum: 65535] [required] Port - Destination port.
  mtu: # [number; minimum: 1, maximum: 65535] Record size limit (bytes) - When prot
  flushPeriodSec: # [number] Flush period (sec) - When protocol is TCP, specifies h
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
graphite_output: # [object]
  type: # [string] Output Type
  protocol: # [string] Destination protocol - Protocol to use when communicating wi

  # -------------- if protocol is udp ---------------

  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (sec) - How

  # -------------------------------------------------------
```

```
  # -------------- if protocol is tcp ---------------

  throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
  connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
  writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # --------------------------------------------------------

  host: # [string] [required] Host - The hostname of the destination.
  port: # [number; minimum: 1, maximum: 65535] [required] Port - Destination port.
  mtu: # [number; minimum: 1, maximum: 65535] Record size limit (bytes) - When prot
  flushPeriodSec: # [number] Flush period (sec) - When protocol is TCP, specifies h
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
router_output: # [object]
  type: # [string] Output Type
  rules: # [array] Rules - Event routing rules
    - filter: # [string] Filter Expression - JavaScript expression to select events
      output: # [string] Output - Output to send matching events to
      description: # [string] Description - Description of this rule's purpose
      final: # [boolean] Final - Flag to control whether to stop the event from bei
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
sns_output: # [object]
  type: # [string] Output Type
  topicArn: # [string] Topic ARN - The ARN of the SNS topic to send events to. When
  messageGroupId: # [string] [required] Message Group ID - Messages in the same gro
  maxRetries: # [number] Maximum number of retries - Maximum number of retries befo
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key
```

```
  # ------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

  # ------------------------------------------------------


  awsSecretKey: # [string] Secret key
  region: # [string] Region - Region where the SNS is located
  endpoint: # [string] Endpoint - SNS service endpoint. If empty, defaults to the A
  signatureVersion: # [string] Signature version - Signature version to use for sig
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
  enableAssumeRole: # [boolean] Enable for SNS - Use Assume Role credentials to acc
  assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
  assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
  durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------


  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
sqs_output: # [object]
  type: # [string] Output Type
  queueName: # [string] Queue Name - The name, URL, or ARN of the SQS queue to send
  queueType: # [string] [required] Queue Type - The queue type used (or created). D
  awsAccountId: # [string] AWS account ID - SQS queue owner's AWS account ID. Leave
  messageGroupId: # [string] Message Group ID - This parameter applies only to FIFO
```

```
createQueue: # [boolean] Create Queue - Create queue if it does not exist.
awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

# -------------- if awsAuthenticationMethod is manual ---------------

awsApiKey: # [string] Access key

# -------------------------------------------------------


# -------------- if awsAuthenticationMethod is secret ---------------

awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

# -------------------------------------------------------

awsSecretKey: # [string] Secret key
region: # [string] Region - AWS Region where the SQS queue is located. Required,
endpoint: # [string] Endpoint - SQS service endpoint. If empty, defaults to the A
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for SQS - Use Assume Role credentials to acc
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
maxQueueSize: # [number; minimum: 1] Queue size limit - Maximum number of queued
maxRecordSizeKB: # [number; minimum: 1, maximum: 256] Record size limit (KB) - Ma
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
maxInProgress: # [number; minimum: 1, maximum: 100] Concurrent request limit - Th
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------
```

```
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
snmp_output: # [object]
  type: # [string] Output Type
  hosts: # [array] SNMP Trap Destinations - One or more SNMP destinations to forwar
    - host: # [string] Address - Destination host
      port: # [number; maximum: 65535] Port - Destination port, default is 162
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (sec) - How
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
sumo_logic_output: # [object]
  type: # [string] Output Type
  url: # [string] API URL - Sumo Logic HTTP collector URL to which events should be
  customSource: # [string] Custom source name - Override the source name configured
  customCategory: # [string] Custom source category - Override the source category
  format: # [string] Data format - Preserve the raw event format instead of JSONify
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1, maximum: 1024] Body size limit (KB) - Ma
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  compress: # [boolean] Compress - Compress the payload body before sending
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automatical
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
```

```
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # ------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any F
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
datadog_output: # [object]
  type: # [string] Output Type
  contentType: # [string] Send logs as - The content type to use when sending logs
  message: # [string] Message field - Name of the event field that contains the mes
  source: # [string] Source - Name of the source to send with logs. When you send l
  host: # [string] Host - Name of the host to send with logs. When you send logs as
  service: # [string] Service - Name of the service to send with logs. When you ser
  tags: # [array of strings] Datadog tags - List of tags to send with logs, such as
  batchByTags: # [boolean] Batch by tags - Batch events by API key and the ddtags f
  allowApiKeyFromEvents: # [boolean] Allow API key from events - Allow API key to b
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
  severity: # [string] Severity - Default value for message severity. When you send
  site: # [string] Datadog site - Datadog site to which events should be sent
```

```
# -------------- if site is custom ---------------

customUrl: # [string]

# ------------------------------------------------------

sendCountersAsCount: # [boolean] Send counter metrics as 'count' - If not enablec
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------
```

```
  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  authType: # [string] Authentication method - Enter API key directly, or select a
  apiKey: # [string] API key - Organization's API key in Datadog

  # -------------- if authType is manual ---------------


  # ------------------------------------------------------

  textSecret: # [string] API key (text secret) - Select or create a stored text sec

  # -------------- if authType is secret ---------------


  # ------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
grafana_cloud_output: # [object]
  type: # [string] Output Type
  lokiUrl: # [string] Loki URL - The endpoint to send logs to, such as https://logs
  prometheusUrl: # [string] Prometheus URL - The remote_write endpoint to send Prom
  message: # [string] Logs message field - Name of the event field that contains th
  messageFormat: # [string] Message format - Format to use when sending logs to Lok

  # -------------- if messageFormat is json ---------------

  compress: # [boolean] Compress - Compress the payload body before sending. Applie

  # ------------------------------------------------------
```

```
labels: # [array] Logs labels - List of labels to send with logs. Labels define l
  - name: # [string] Name
    value: # [string] Value
metricRenameExpr: # [string] Metrics renaming expression - JavaScript expression
prometheusAuth: # [object]
  authType: # [string] Authentication type

  # -------------- if authType is token ---------------

  token: # [string] Auth token - Bearer token to include in the authorization hea

  # ----------------------------------------------------


  # -------------- if authType is textSecret ---------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored tex

  # ----------------------------------------------------


  # -------------- if authType is basic ---------------

  username: # [string] Username - Username for authentication
  password: # [string] Password - Password (API key in Grafana Cloud domain) for

  # ----------------------------------------------------


  # -------------- if authType is credentialsSecret ---------------

  credentialsSecret: # [string] Credentials secret - Select or create a secret th

  # ----------------------------------------------------

lokiAuth: # [object]
  authType: # [string] Authentication type

  # -------------- if authType is token ---------------

  token: # [string] Auth token - Bearer token to include in the authorization hea

  # ----------------------------------------------------
```

```
  # -------------- if authType is textSecret ---------------

  textSecret: # [string] Auth token (text secret) - Select or create a stored tex

  # ---------------------------------------------------------


  # -------------- if authType is basic ---------------

  username: # [string] Username - Username for authentication
  password: # [string] Password - Password (API key in Grafana Cloud domain) for

  # ---------------------------------------------------------


  # -------------- if authType is credentialsSecret ---------------

  credentialsSecret: # [string] Credentials secret - Select or create a secret th

  # ---------------------------------------------------------

concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
systemFields: # [array of strings] System fields - Fields to automatically add to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automatical
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
```

```
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # -----------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any F
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # -----------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
loki_output: # [object]
  type: # [string] Output Type
  url: # [string] Loki URL - The endpoint to send logs to
  message: # [string] Logs message field - Name of the event field that contains th
  messageFormat: # [string] Message format - Format to use when sending logs to Lok

  # -------------- if messageFormat is json ---------------

  compress: # [boolean] Compress - Compress the payload body before sending

  # -----------------------------------------------------

  labels: # [array] Logs labels - List of labels to send with logs. Labels define l
    - name: # [string] Name
```

```
    value: # [string] Value
authType: # [string] Authentication type

# -------------- if authType is token ---------------

token: # [string] Auth token - Bearer token to include in the authorization heade

# ---------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Auth token (text secret) - Select or create a stored text

# ---------------------------------------------------


# -------------- if authType is basic ---------------

username: # [string] Username - Username for authentication
password: # [string] Password - Password (API key in Grafana Cloud domain) for au

# ---------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ---------------------------------------------------

concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
```

```
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automatically
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # -------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  systemFields: # [array of strings] System fields - Fields to automatically add to
  enableDynamicHeaders: # [boolean] Enable dynamic headers - Add per-event HTTP hea
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # -------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
prometheus_output: # [object]
  type: # [string] Output Type
  url: # [string] Remote Write URL - The endpoint to send metrics to
  metricRenameExpr: # [string] Metric renaming expression - JavaScript expression t
```

```
sendMetadata: # [boolean] Send metadata - Generate and send metadata (`type` and

# -------------- if sendMetadata is true ---------------

metricsFlushPeriodSec: # [number] Metadata flush period (sec) - How frequently me

# ------------------------------------------------------

systemFields: # [array of strings] System fields - Fields to automatically add to
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue --------------
```

```
pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------

authType: # [string] Authentication type - Remote Write authentication type

# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# -------------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# -------------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# -------------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# -------------------------------------------------------


# -------------- if authType is oauth ---------------
```

```
  loginUrl: # [string] Login URL - URL for OAuth
  secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
  secret: # [string] OAuth secret - Secret parameter value to pass in request body
  tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
  spacer: # [null]
  authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
  tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
  oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
    - name: # [string] Name - OAuth parameter name
      value: # [string] Value - OAuth parameter value
  oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth l
    - name: # [string] Name - OAuth header name
      value: # [string] Value - OAuth header value

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
ring_output: # [object]
  type: # [string] Output Type
  format: # [string] Data format - Format of the output data.
  partitionExpr: # [string] Partitioning expression - JS expression to define how t
  maxDataSize: # [string] Data size limit - Maximum disk space allowed to be consum
  maxDataTime: # [string] Data age limit - Maximum amount of time to retain data (e
  compress: # [string] Data compression format
  destPath: # [string] Path location - Path to use to write metrics. Defaults to $C
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
open_telemetry_output: # [object]
  type: # [string] Output Type
  protocol: # [string] Protocol - Select a transport option for OpenTelemetry

  # -------------- if protocol is http ---------------

  httpTracesEndpointOverride: # [string] Traces endpoint override - If you want to
  httpMetricsEndpointOverride: # [string] Metrics endpoint override - If you want t
  httpLogsEndpointOverride: # [string] Logs endpoint override - If you want to send
```

```
httpCompress: # [string] Compression - Type of compression to apply to messages s
keepAlive: # [boolean] Keep alive - Disable to close the connection immediately a
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automatically
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R

# -------------------------------------------------------


# -------------- if protocol is grpc ---------------

connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
keepAliveTime: # [number; minimum: 1] Keep alive time (seconds) - How often the s
metadata: # [array] Metadata - List of key-value pairs to send with each gRPC req
  - key: # [string] Key
    value: # [string] Value
tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------
```

```
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ----------------------------------------------------------

compress: # [string] Compression - Type of compression to apply to messages sent

# ----------------------------------------------------------

endpoint: # [string] Endpoint - The endpoint where OTel events will be sent. Ente
otlpVersion: # [string] OTLP version - The version of OTLP Protobuf definitions t
authType: # [string] Authentication type - OpenTelemetry authentication type

# -------------- if authType is basic ---------------

username: # [string] Username
password: # [string] Password

# ----------------------------------------------------------


# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# ----------------------------------------------------------


# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ----------------------------------------------------------


# -------------- if authType is textSecret ---------------
```

```
textSecret: # [string] Token (text secret) - Select or create a stored text secre

# -------------------------------------------------------


# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth t
  - name: # [string] Name - OAuth header name
    value: # [string] Value - OAuth header value

# -------------------------------------------------------

concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -------------------------------------------------------

description: # [string] Description
```

```
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
service_now_output: # [object]
  type: # [string] Output Type
  endpoint: # [string] Endpoint - The endpoint where ServiceNow events will be sent
  tokenSecret: # [string] [required] Auth token (text secret) - Select or create a
  authTokenName: # [string] Auth token name
  otlpVersion: # [string] [required] OTLP version - The version of OTLP Protobuf de
  maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
  protocol: # [string] [required] Protocol - Select a transport option for OpenTele

  # -------------- if protocol is http ---------------

  httpTracesEndpointOverride: # [string] Traces endpoint override - If you want to
  httpMetricsEndpointOverride: # [string] Metrics endpoint override - If you want t
  httpLogsEndpointOverride: # [string] Logs endpoint override - If you want to send
  httpCompress: # [string] Compression - Type of compression to apply to messages s
  keepAlive: # [boolean] Keep alive - Disable to close the connection immediately a
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automatically
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # -------------------------------------------------------
```

```
responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R

# -------------------------------------------------------


# -------------- if protocol is grpc ---------------

connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
keepAliveTime: # [number; minimum: 1] Keep alive time (seconds) - How often the s
metadata: # [array] Metadata - List of key-value pairs to send with each gRPC req
  - key: # [string] Key
    value: # [string] Value
tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # -------------------------------------------------------

compress: # [string] Compression - Type of compression to apply to messages sent

# -------------------------------------------------------

concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
```

```
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
dataset_output: # [object]
  type: # [string] Output Type
  messageField: # [string] Message field - Name of the event field that contains th
  excludeFields: # [array of strings] Exclude fields - Fields to exclude from the e
  serverHostField: # [string] Server/host field - Name of the event field that cont
  timestampField: # [string] Timestamp field - Name of the event field that contain
  defaultSeverity: # [string] Severity - Default value for event severity. If the `
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automatically
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # ------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  site: # [string] DataSet site - DataSet site to which events should be sent

  # -------------- if site is custom ---------------

  customUrl: # [string]
```

```
# ------------------------------------------------------

concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 6144] Body size limit (KB) -
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# ------------------------------------------------------

authType: # [string] Authentication method - Enter API key directly, or select a
apiKey: # [string] API key - A 'Log Write Access' API key for the DataSet account

# -------------- if authType is manual ---------------


# ------------------------------------------------------

textSecret: # [string] API key (text secret) - Select or create a stored text sec

# -------------- if authType is secret ---------------
```

```
  # ------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
cribl_tcp_output: # [object]
  type: # [string] Output Type
  loadBalanced: # [boolean] Load balancing - Use load-balanced destinations

  # -------------- if loadBalanced is false ---------------

  host: # [string] Address - The hostname of the receiver
  port: # [number; maximum: 65535] Port - The port to connect to on the provided ho

  # ------------------------------------------------------


  # -------------- if loadBalanced is true ---------------

  excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the curren
  hosts: # [array] Destinations - Set of hosts to load-balance data to
    - host: # [string] Address - The hostname of the receiver
      port: # [number; maximum: 65535] Port - The port to connect to on the provide
      tls: # [string] TLS - Whether to inherit TLS configs from group setting or di
      servername: # [string] TLS Servername - Servername to use if establishing a T
      weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
  loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec
  maxConcurrentSenders: # [number] Connection limit - Maximum number of concurrent

  # ------------------------------------------------------

  compression: # [string] Compression - Codec to use to compress the data before se
  logFailedRequests: # [boolean] Log failed requests to disk - Use to troubleshoot
  throttleRatePerSec: # [string] Throttling - Rate (in bytes per second) to throttl
  tls: # [object] TLS settings (client side)
    disabled: # [boolean] Disabled

    # -------------- if disabled is false ---------------
```

```
    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                        trusted CA (such as the system's). Defaults to Enabled. Overrides
    servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
    certificateName: # [string] Certificate - The name of the predefined certificat
    caPath: # [string] CA certificate path - Path on client in which to find CA cer
    privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
    certPath: # [string] Certificate path (mutual auth) - Path on client in which t
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    minVersion: # [string] Minimum TLS version
    maxVersion: # [string] Maximum TLS version

    # ------------------------------------------------------

  connectionTimeout: # [number] Connection timeout - Amount of time (milliseconds)
  writeTimeout: # [number] Write timeout - Amount of time (milliseconds) to wait fo
  tokenTTLMinutes: # [number; minimum: 1, maximum: 60] Auth Token TTL minutes - The
  excludeFields: # [array of strings] Exclude fields - Fields to exclude from the e
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
cribl_http_output: # [object]
  type: # [string] Output Type
  loadBalanced: # [boolean] Load balancing - For optimal performance, enable load b

  # -------------- if loadBalanced is false ---------------

  url: # [string] Cribl endpoint - URL of a Cribl Worker to send events to, such as
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
```

```
# ------------------------------------------------------


# -------------- if loadBalanced is true ---------------

excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the currer
urls: # [array] Cribl Worker endpoints
  - url: # [string] Cribl Endpoint - URL of a Cribl Worker to send events to, suc
    weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec

# ------------------------------------------------------

tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates tha
                trusted CA (such as the system's). Defaults to Enabled. Overrides
  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # ------------------------------------------------------

tokenTTLMinutes: # [number; minimum: 1, maximum: 60] Auth Token TTL minutes - The
excludeFields: # [array of strings] Exclude fields - Fields to exclude from the e
compression: # [string] Compression - Codec to use to compress the data before se
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
```

```
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
  timeoutRetrySettings: # [object]
    timeoutRetry: # [boolean] Retry timed-out HTTP requests

    # -------------- if timeoutRetry is true ---------------

    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

    # ------------------------------------------------------

  responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
  onBackpressure: # [string] Backpressure behavior - How to handle events when all

  # -------------- if onBackpressure is queue ---------------

  pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
  pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
  pqPath: # [string] Queue file path - The location for the persistent queue files.
  pqCompress: # [string] Compression - Codec to use to compress the persisted data
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
humio_hec_output: # [object]
  type: # [string] Output Type
```

```
url: # [string] [required] LogScale endpoint - URL to a CrowdStrike Falcon LogSca
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 32768] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
format: # [string] Request format - When set to JSON, the event is automatically
authType: # [string] Authentication method - Select Manual to enter an auth token

# -------------- if authType is manual ---------------

token: # [string] LogScale auth token - CrowdStrike Falcon LogScale authenticatio

# ----------------------------------------------------


# -------------- if authType is secret ---------------

textSecret: # [string] LogScale auth token (text secret) - Select or create a sto

# ----------------------------------------------------

responseRetrySettings: # [array] Settings for failed HTTP requests - Automatical
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
```

```
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
      maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

      # ------------------------------------------------------

    responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
    onBackpressure: # [string] Backpressure behavior - How to handle events when all

    # -------------- if onBackpressure is queue ---------------

    pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
    pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
    pqPath: # [string] Queue file path - The location for the persistent queue files.
    pqCompress: # [string] Compression - Codec to use to compress the persisted data
    pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
    pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

    # -------------------------------------------------------

    description: # [string] Description
    pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
    systemFields: # [array of strings] System fields - Fields to automatically add to
    environment: # [string] Environment - Optionally, enable this config only on a sp
    streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
  crowdstrike_next_gen_siem_output: # [object]
    type: # [string] Output Type
    url: # [string] [required] Next-Gen SIEM endpoint - URL provided from a CrowdStri
Example: https://ingest.<region>.crowdstrike.com/api/ingest/hec/<connection-id>/v1/se
    concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
    maxPayloadSizeKB: # [number; minimum: 1024, maximum: 32768] Body size limit (KB)
    maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
    compress: # [boolean] Compress - Compress the payload body before sending
    rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
        Enabled by default. When this setting is also present in TLS Settings (Client
        that value will take precedence.
    timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
    flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
    extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
      - name: # [string] Field Name
        value: # [string] Field Value
    useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
    failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
    safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
```

```
format: # [string] Request format - When set to JSON, the event is automatically
authType: # [string] Authentication method - Select Manual to enter an auth toker

# -------------- if authType is manual ---------------

token: # [string] Next-Gen SIEM authentication token

# ----------------------------------------------------


# -------------- if authType is secret ---------------

textSecret: # [string] Next-Gen SIEM authentication token (text secret) - Select

# ----------------------------------------------------

responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ----------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th
```

```
  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
dl_s3_output: # [object]
  type: # [string] Output Type
  bucket: # [string] S3 bucket name - Name of the destination S3 bucket. Must be a
  region: # [string] Region - Region where the S3 bucket is located
  awsSecretKey: # [string] Secret key - Secret key. This value can be a constant or
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key - This value can be a constant or a JavaScript e

  # -------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref

  # -------------------------------------------------------

  endpoint: # [string] Endpoint - S3 service endpoint. If empty, defaults to the AW
  signatureVersion: # [string] Signature version - Signature version to use for sig
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
  rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
  enableAssumeRole: # [boolean] Enable for S3 - Use Assume Role credentials to acce
  assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
  assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
  durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
  stagePath: # [string] [required] Staging location - Filesystem location in which
  addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
  destPath: # [string] Key prefix - Prefix to prepend to files before uploading. Mu
  objectACL: # [string] Object ACL - Object ACL to assign to uploaded objects
  storageClass: # [string] Storage class - Storage class to select for uploaded obj
  serverSideEncryption: # [string] Server-side encryption for uploaded objects
```

```
# -------------- if serverSideEncryption is aws:kms ---------------

kmsKeyId: # [string] KMS key ID - ID or ARN of the KMS customer-managed key to us

# ------------------------------------------------------

removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

# -------------- if removeEmptyDirs is true ---------------

emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

# ------------------------------------------------------

format: # [string] Data format - Format of the output data

# -------------- if format is json ---------------

compress: # [string] Compression - Data compression format to apply to HTTP conte
compressionLevel: # [string] Compression level - Compression level to apply befor

# ------------------------------------------------------


# -------------- if format is parquet ---------------

automaticSchema: # [boolean] Automatic schema - Automatically calculate the schem
parquetVersion: # [string] Parquet version - Determines which data types are supp
parquetDataPageVersion: # [string] Data page version - Serialization format of da
parquetRowGroupLength: # [number; minimum: 1, maximum: 67108864] Group row limit
parquetPageSize: # [string] Page size - Target memory size for page segments, suc
shouldLogInvalidRows: # [boolean] Log invalid rows - Log up to 3 rows that Cribl
keyValueMetadata: # [array] Metadata (optional) - The metadata of files the Desti
  - key: # [string] Key
    value: # [string] Value
enableStatistics: # [boolean] Write statistics - Statistics profile an entire fil
enableWritePageIndex: # [boolean] Write page indexes - One page index contains st
enablePageChecksum: # [boolean] Write page checksum - Parquet tools can use the c

# ------------------------------------------------------

baseFileName: # [string] File name prefix expression - JavaScript expression to c
fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
```

```
  maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
  maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
  headerLine: # [string] Header line - If set, this line will be written to the beg
  writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
  onBackpressure: # [string] Backpressure behavior - How to handle events when all
  deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

  # -------------- if deadletterEnabled is true ---------------

  deadletterPath: # [string] Dead-letter location - Storage location for files that
  maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

  # ------------------------------------------------------------

  onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 86400] File open time limit
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 86400] Idle time limit (sec)
  maxConcurrentFileParts: # [number; minimum: 1, maximum: 10] Concurrent file parts
  verifyPermissions: # [boolean] Verify if bucket exists - Disable if you can acces
  maxClosingFilesToBackpressure: # [number; minimum: 10, maximum: 4200] Staging fil
  partitioningFields: # [array of strings] Partition by fields - List of fields to
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
security_lake_output: # [object]
  type: # [string] Output Type
  bucket: # [string] S3 bucket name - Name of the destination S3 bucket. Must be a
  region: # [string] [required] Region - Region where the Amazon Security Lake is l
  awsSecretKey: # [string] Secret key
  awsAuthenticationMethod: # [string] Authentication method - AWS authentication me

  # -------------- if awsAuthenticationMethod is manual ---------------

  awsApiKey: # [string] Access key - This value can be a constant or a JavaScript e

  # ------------------------------------------------------------


  # -------------- if awsAuthenticationMethod is secret ---------------

  awsSecret: # [string] Secret key pair - Select or create a stored secret that ref
```

```
# ----------------------------------------------------------

endpoint: # [string] Endpoint - Amazon Security Lake service endpoint. If empty,
signatureVersion: # [string] Signature version - Signature version to use for sig
reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for S3 - Use Assume Role credentials to acce
assumeRoleArn: # [string] [required] AssumeRole ARN - Amazon Resource Name (ARN)
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
stagePath: # [string] [required] Staging location - Filesystem location in which
addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
objectACL: # [string] Object ACL - Object ACL to assign to uploaded objects
storageClass: # [string] Storage class - Storage class to select for uploaded obj
serverSideEncryption: # [string] Server-side encryption for uploaded objects

# -------------- if serverSideEncryption is aws:kms ---------------

kmsKeyId: # [string] KMS key ID - ID or ARN of the KMS customer-managed key to us

# ----------------------------------------------------------

removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

# -------------- if removeEmptyDirs is true ---------------

emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

# ----------------------------------------------------------

baseFileName: # [string] File name prefix expression - JavaScript expression to c
maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
headerLine: # [string] Header line - If set, this line will be written to the beg
writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
onBackpressure: # [string] Backpressure behavior - How to handle events when all
deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

# -------------- if deadletterEnabled is true ---------------

deadletterPath: # [string] Dead-letter location - Storage location for files that
maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f
```

```
  # ------------------------------------------------------

  onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 86400] File open time limit
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 86400] Idle time limit (sec)
  maxConcurrentFileParts: # [number; minimum: 1, maximum: 10] Concurrent file parts
  verifyPermissions: # [boolean] Verify if bucket exists - Disable if you can acces
  maxClosingFilesToBackpressure: # [number; minimum: 10, maximum: 4200] Staging fil
  accountId: # [string] [required] Account ID - ID of the AWS account whose data th
  customSource: # [string] [required] Custom source name - Name of the custom sourc
  automaticSchema: # [boolean] Automatic schema - Automatically calculate the schem

  # -------------- if automaticSchema is false ---------------

  parquetSchema: # [string] Parquet schema - To add a new schema, navigate to Proce

  # ------------------------------------------------------

  parquetVersion: # [string] Parquet version - Determines which data types are supp
  parquetDataPageVersion: # [string] Data page version - Serialization format of da
  parquetRowGroupLength: # [number; minimum: 1, maximum: 67108864] Group row limit
  parquetPageSize: # [string] Page size - Target memory size for page segments, suc
  shouldLogInvalidRows: # [boolean] Log invalid rows - Log up to 3 rows that Cribl
  keyValueMetadata: # [array] Metadata (optional) - The metadata of files the Desti
    - key: # [string] Key
      value: # [string] Value
  enableStatistics: # [boolean] Write statistics - Statistics profile an entire fil
  enableWritePageIndex: # [boolean] Write page indexes - One page index contains st
  enablePageChecksum: # [boolean] Write page checksum - Parquet tools can use the c
  systemFields: # [array of strings] System fields - Fields to automatically add to
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
cribl_lake_output: # [object]
  type: # [string] Output Type
  bucket: # [string] S3 bucket name - Name of the destination S3 bucket. Must be a
  region: # [string] Region - Region where the S3 bucket is located
  awsSecretKey: # [string] Secret key - Secret key. This value can be a constant or
  endpoint: # [string] Endpoint - S3 service endpoint. If empty, defaults to the AW
  signatureVersion: # [string] Signature version - Signature version to use for sig
  reuseConnections: # [boolean] Reuse connections - Reuse connections between reque
```

```
rejectUnauthorized: # [boolean] Reject unauthorized certificates - Reject certifi
enableAssumeRole: # [boolean] Enable for S3 - Use Assume Role credentials to acce
assumeRoleArn: # [string] AssumeRole ARN - Amazon Resource Name (ARN) of the role
assumeRoleExternalId: # [string] External ID - External ID to use when assuming r
durationSeconds: # [number; minimum: 900, maximum: 43200] Duration (seconds) - Du
stagePath: # [string] Staging location - Filesystem location in which to buffer f
addIdToStagePath: # [boolean] Add output ID - Add the Output ID value to staging
destPath: # [string] Lake dataset - Lake dataset to send the data to.
objectACL: # [string] Object ACL - Object ACL to assign to uploaded objects
storageClass: # [string] Storage class - Storage class to select for uploaded obj
serverSideEncryption: # [string] Server-side encryption for uploaded objects

# -------------- if serverSideEncryption is aws:kms ---------------

kmsKeyId: # [string] KMS key ID - ID or ARN of the KMS customer-managed key to us

# ------------------------------------------------------

removeEmptyDirs: # [boolean] Remove empty staging directories - Remove empty stag

# -------------- if removeEmptyDirs is true ---------------

emptyDirCleanupSec: # [number; minimum: 10, maximum: 86400] Staging cleanup perio

# ------------------------------------------------------

baseFileName: # [string] File name prefix expression - JavaScript expression to c
fileNameSuffix: # [string] File name suffix expression - JavaScript expression to
maxFileSizeMB: # [number; minimum: 5, maximum: 1024] File size limit (MB) - Maxim
maxOpenFiles: # [number; minimum: 10, maximum: 2000] Open file limit - Maximum nu
headerLine: # [string] Header line - If set, this line will be written to the beg
writeHighWaterMark: # [number; minimum: 16, maximum: 4096] Writing high watermark
onBackpressure: # [string] Backpressure behavior - How to handle events when all
deadletterEnabled: # [boolean] Enable dead-lettering - If a file fails to move to

# -------------- if deadletterEnabled is true ---------------

deadletterPath: # [string] Dead-letter location - Storage location for files that
maxRetryNum: # [number; minimum: 1] Retry limit - The maximum number of times a f

# ------------------------------------------------------

onDiskFullBackpressure: # [string] Disk space protection - How to handle events w
```

```
  maxFileOpenTimeSec: # [number; minimum: 10, maximum: 86400] File open time limit
  maxFileIdleTimeSec: # [number; minimum: 5, maximum: 86400] Idle time limit (sec)
  verifyPermissions: # [boolean] Verify if bucket exists - Disable if you can acces
  maxClosingFilesToBackpressure: # [number; minimum: 10, maximum: 4200] Staging fil
  awsAuthenticationMethod: # [string]
  format: # [string]
  maxConcurrentFileParts: # [number; minimum: 1, maximum: 10] Concurrent file parts
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
disk_spool_output: # [object]
  type: # [string] Output Type
  timeWindow: # [string] Bucket time span - Time period for grouping spooled events
  maxDataSize: # [string] Data size limit - Maximum disk space that can be consumed
  maxDataTime: # [string] Data age limit - Maximum amount of time to retain data be
  compress: # [string] Compression - Data compression format. Default is gzip.
  partitionExpr: # [string] Partitioning expression - JavaScript expression definin
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
click_house_output: # [object]
  type: # [string] Output Type
  url: # [string] [required] URL - URL of the ClickHouse instance. Example: http://
  authType: # [string] Authentication type

  # -------------- if authType is basic ---------------

  username: # [string] Username
  password: # [string] Password

  # ---------------------------------------------------


  # -------------- if authType is token ---------------

  token: # [string] Token - Bearer token to include in the authorization header

  # ---------------------------------------------------
```

```
# -------------- if authType is credentialsSecret ---------------

credentialsSecret: # [string] Credentials secret - Select or create a secret that

# ----------------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# ----------------------------------------------------------


# -------------- if authType is oauth ---------------

loginUrl: # [string] Login URL - URL for OAuth
secretParamName: # [string] OAuth Secret parameter name - Secret parameter name t
secret: # [string] OAuth secret - Secret parameter value to pass in request body
tokenAttributeName: # [string] Token attribute name - Name of the auth token attr
spacer: # [null]
authHeaderExpr: # [string] Authorize expression - JavaScript expression to comput
tokenTimeoutSecs: # [number; minimum: 1, maximum: 300000] Refresh interval (secs.
oauthParams: # [array] OAuth parameters - Additional parameters to send in the OA
  - name: # [string] Name - OAuth parameter name
    value: # [string] Value - OAuth parameter value
oauthHeaders: # [array] OAuth headers - Additional headers to send in the OAuth l
  - name: # [string] Name - OAuth header name
    value: # [string] Value - OAuth header value

# ----------------------------------------------------------


# -------------- if authType is sslUserCertificate ---------------

sqlUsername: # [string] Username - Username for certificate authentication

# ----------------------------------------------------------

database: # [string] ClickHouse database
tableName: # [string] [required] ClickHouse table - Name of the ClickHouse table
format: # [string] Format - Data format to use when sending data to ClickHouse. D
```

```
mappingType: # [string] Mapping type - How event fields are mapped to ClickHouse

# -------------- if mappingType is automatic ---------------

excludeMappingFields: # [array of strings] Exclude fields - Fields to exclude fro

# -----------------------------------------------------------


# -------------- if mappingType is custom ---------------

describeTable: # [string] Retrieve table columns - Retrieves the table schema fro
columnMappings: # [array] Column Mapping
  - columnName: # [string] Column name - Name of the column in ClickHouse that wi
    columnType: # [string] Column type - Type of the column in the ClickHouse dat
    columnValueExpression: # [string] Column value - JavaScript expression to com

# -----------------------------------------------------------

asyncInserts: # [boolean] Async inserts - Collect data into batches for later pro

# -------------- if asyncInserts is true ---------------

waitForAsyncInserts: # [boolean] Wait for async inserts - Cribl will wait for cor

# -----------------------------------------------------------

tls: # [object] TLS settings (client side)
  disabled: # [boolean] Disabled

  # -------------- if disabled is false ---------------

  servername: # [string] Server name (SNI) - Server name for the SNI (Server Name
  certificateName: # [string] Certificate - The name of the predefined certificat
  caPath: # [string] CA certificate path - Path on client in which to find CA cer
  privKeyPath: # [string] Private key path (mutual auth) - Path on client in whic
  certPath: # [string] Certificate path (mutual auth) - Path on client in which t
  passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
  minVersion: # [string] Minimum TLS version
  maxVersion: # [string] Maximum TLS version

  # -----------------------------------------------------------
```

```
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 10240] Body size limit (KB)
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
dumpFormatErrorsToDisk: # [boolean] Log last schema mismatch - Log the most recen
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th
```

```
  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
xsiam_output: # [object]
  type: # [string] Output Type
  loadBalanced: # [boolean] Load balancing - Enable for optimal performance. Even i

  # -------------- if loadBalanced is false ---------------

  url: # [string] XSIAM endpoint - XSIAM endpoint URL to send events to, such as ht
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh

  # ------------------------------------------------------


  # -------------- if loadBalanced is true ---------------

  excludeSelf: # [boolean] Exclude current host IPs - Exclude all IPs of the curren
  urls: # [array] XSIAM Endpoints
      weight: # [number] Load Weight - Assign a weight (>0) to each endpoint to inc
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (seconds) -
  loadBalanceStatsPeriodSec: # [number; minimum: 10] Load balance stats period (sec

  # ------------------------------------------------------

  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 100, maximum: 10000] Body size limit (KB) -
  maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
    - name: # [string] Field Name
      value: # [string] Field Value
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
```

```
authType: # [string] Authentication method - Enter a token directly, or provide a

# -------------- if authType is token ---------------

token: # [string] Auth token - XSIAM authentication token

# ---------------------------------------------------


# -------------- if authType is secret ---------------

textSecret: # [string] Auth token (text secret) - Select or create a stored text

# ---------------------------------------------------

responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ---------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R
throttleRateReqPerSec: # [integer; maximum: 2000] Throttle request rate limit - M
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th
```

```
  # ------------------------------------------------------

  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
netflow_output: # [object]
  type: # [string] Output Type
  hosts: # [array] NetFlow Destinations - One or more NetFlow destinations to forwa
    - host: # [string] Address - Destination host
      port: # [number; maximum: 65535] Port - Destination port, default is 2055
  dnsResolvePeriodSec: # [number; maximum: 86400] DNS resolution period (sec) - How
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
dynatrace_http_output: # [object]
  type: # [string] Output Type
  method: # [string] Method - The method to use when sending events
  keepAlive: # [boolean] Keep alive - Disable to close the connection immediately a
  concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
  maxPayloadSizeKB: # [number; minimum: 1024, maximum: 5000] Body size limit (KB) -
  maxPayloadEvents: # [number; maximum: 50000] Events-per-request limit - Maximum r
  compress: # [boolean] Compress - Compress the payload body before sending
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
  timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
  flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
  extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events. Yo
    - name: # [string] Field Name
      value: # [string] Field Value
  useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
  failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
  safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
  responseRetrySettings: # [array] Settings for failed HTTP requests - Automatical
    - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
      initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
      backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
```

```
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lor
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # -----------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any F
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

# -----------------------------------------------------

authType: # [string] Authentication type

# -------------- if authType is token ---------------

token: # [string] Token - Bearer token to include in the authorization header

# -----------------------------------------------------


# -------------- if authType is textSecret ---------------

textSecret: # [string] Token (text secret) - Select or create a stored text secre

# -----------------------------------------------------

format: # [string] Format - How to format events before sending. Defaults to JSON
endpoint: # [string] [required] Endpoint
```

```
  # -------------- if endpoint is cloud ---------------

  environmentId: # [string] Environment ID - ID of the environment to send to

  # ----------------------------------------------------


  # -------------- if endpoint is activeGate ---------------

  activeGateDomain: # [string] ActiveGate domain - ActiveGate domain with Log analy
  environmentId: # [string] Environment ID - ID of the environment to send to

  # ----------------------------------------------------


  # -------------- if endpoint is manual ---------------

  url: # [string] URL - URL to send events to. Can be overwritten by an event's __u

  # ----------------------------------------------------

  telemetryType: # [string] [required] Telemetry type
  totalMemoryLimitKB: # [number] Buffer memory limit (KB) - Maximum total size of t
  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
dynatrace_otlp_output: # [object]
  type: # [string] Output Type
  protocol: # [string] [required] Protocol - Select a transport option for Dynatrac

  # -------------- if protocol is http ---------------

  httpTracesEndpointOverride: # [string] Traces endpoint override - If you want to
  httpMetricsEndpointOverride: # [string] Metrics endpoint override - If you want t
  httpLogsEndpointOverride: # [string] Logs endpoint override - If you want to send
  httpCompress: # [string] Compression - Type of compression to apply to messages s
  keepAlive: # [boolean] Keep alive - Disable to close the connection immediately a
  rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
      Enabled by default. When this setting is also present in TLS Settings (Client
      that value will take precedence.
```

```
useRoundRobinDns: # [boolean] Round-robin DNS - Enable round-robin DNS lookup. Wh
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
  timeoutRetry: # [boolean] Retry timed-out HTTP requests

  # -------------- if timeoutRetry is true ---------------

  initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How lon
  backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base for
  maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) - Th

  # ------------------------------------------------------

responseHonorRetryAfterHeader: # [boolean] Honor Retry-After header - Honor any R

# -------------------------------------------------------

endpoint: # [string] Endpoint - The endpoint where Dynatrace events will be sent.
otlpVersion: # [string] [required] OTLP version - The version of OTLP Protobuf de
concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 4096] Body size limit - Maxi
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
endpointType: # [string] [required] Endpoint type - Select the type of Dynatrace
tokenSecret: # [string] [required] Auth token (text secret) - Select or create a
authTokenName: # [string] Api-Token name
onBackpressure: # [string] Backpressure behavior - How to handle events when all

# -------------- if onBackpressure is queue ---------------

pqMaxFileSize: # [string] File size limit - The maximum size to store in each que
pqMaxSize: # [string] Queue size limit - The maximum disk space that the queue ca
pqPath: # [string] Queue file path - The location for the persistent queue files.
pqCompress: # [string] Compression - Codec to use to compress the persisted data
```

```
  pqOnBackpressure: # [string] Queue-full behavior - How to handle events when the
  pqMode: # [string] Mode - In Error mode, PQ writes events to the filesystem if th

  # ------------------------------------------------------

  description: # [string] Description
  pipeline: # [string] Pipeline - Pipeline to process data before sending out to th
  systemFields: # [array of strings] System fields - Fields to automatically add to
  environment: # [string] Environment - Optionally, enable this config only on a sp
  streamtags: # [array of strings] Tags - Tags for filtering and grouping in Cribl
sentinel_one_ai_siem_output: # [object]
  type: # [string] Output Type
  region: # [string] Region - The SentinelOne region to send events to. In most cas

  # -------------- if region is Custom ---------------

  baseUrl: # [string] Base AI SIEM endpoint URL - Base URL of the endpoint used to

  # ------------------------------------------------------

  endpoint: # [string] [required] AI SIEM endpoint path - Endpoint to send events t

  # -------------- if endpoint is /services/collector/event ---------------

  hostExpression: # [string] serverHost expression - Define serverHost for events u
  sourceExpression: # [string] logFile expression - Define logFile for events using
  sourceTypeExpression: # [string] parser expression - Define the parser for events
  dataSourceCategoryExpression: # [string] dataSource.category expression - Define
  dataSourceNameExpression: # [string] dataSource.name expression - Define the data
  dataSourceVendorExpression: # [string] dataSource.vendor expression - Define the
  eventTypeExpression: # [string] event.type expression - Optionally, define the ev

  # ------------------------------------------------------



  # -------------- if endpoint is /services/collector/raw ---------------

  host: # [string] serverHost expression - Define the serverHost for events using a
  source: # [string] logFile - Specify the logFile value to pass as a parameter to
  sourceType: # [string] parser - Specify the sourcetype parameter for SentinelOne
  dataSourceCategory: # [string] dataSource.category - Specify the dataSource.categ
  dataSourceName: # [string] dataSource.name - Specify the dataSource.name value to
  dataSourceVendor: # [string] dataSource.vendor - Specify the dataSource.vendorval
```

```
eventType: # [string] event.type - Specify the event.type value to pass as an opt

# ------------------------------------------------------

concurrency: # [number; minimum: 1, maximum: 32] Request concurrency - Maximum nu
maxPayloadSizeKB: # [number; minimum: 1024, maximum: 2097152] Body size limit (KB
maxPayloadEvents: # [number] Events-per-request limit - Maximum number of events
compress: # [boolean] Compress - Compress the payload body before sending
rejectUnauthorized: # [boolean] Validate server certs - Reject certificates not a
    Enabled by default. When this setting is also present in TLS Settings (Client
    that value will take precedence.
timeoutSec: # [number; minimum: 1, maximum: 9007199254740991] Request timeout - A
flushPeriodSec: # [number] Flush period (sec) - Maximum time between requests. Sm
extraHttpHeaders: # [array] Extra HTTP headers - Headers to add to all events
  - name: # [string] Field Name
    value: # [string] Field Value
failedRequestLoggingMode: # [string] Failed request logging mode - Data to log wh
safeHeaders: # [array of strings] Safe headers - List of headers that are safe to
authType: # [string] Authentication method - Select Manual to enter an auth token

# -------------- if authType is manual ---------------

token: # [string] AI SIEM API Key - In the SentinelOne Console select Policy & Se

# ------------------------------------------------------


# -------------- if authType is secret ---------------

textSecret: # [string] AI SIEM API Key (text secret) - Select or create a stored

# ------------------------------------------------------

responseRetrySettings: # [array] Settings for failed HTTP requests - Automaticall
  - httpStatus: # [number; minimum: 100, maximum: 599] HTTP status code - The HTT
    initialBackoff: # [number; maximum: 600000] Pre-backoff interval (ms) - How l
    backoffRate: # [number; minimum: 1, maximum: 20] Backoff multiplier - Base fo
    maxBackoff: # [number; minimum: 10000, maximum: 180000] Backoff limit (ms) -
timeoutRetrySettings: # [object]
```

10.5.18. Parsers.Yml

`parsers.yml` stores configuration data for the Knowledge > Parsers Library.

parsers.yml

```
# Parser ID - Unique identifier for this parser
# [string; required]
id:
# Library - Library classification for the parser
# One of: cribl | cribl-custom | custom
# [string]
lib:
# Description - Description of the parser
# [string]
description:
# Tags - Tags associated with the parser
# [string]
tags:
# Type - Parser or formatter type to use
# One of: csv | elff | clf | kvp | json | delim | regex | grok
# [string; required]
type:
# Fields - Field names to extract
fields:
# Mode - Operation mode - Extract creates new fields. Reserialize extracts and filter
# reserializes.
# One of: extract | reserialize
# [string; default: extract]
mode:
# Source field - Field containing text to be parsed
# [string; default: _raw]
srcField:
# Destination field - Destination field name
# [string]
dstField:
# Clean fields - Clean field names by replacing non [a-zA-Z0-9] characters with _
# [boolean]
cleanFields:
# Field filter expression
# [string]
fieldFilterExpr:
# Keep - Field names to keep
keep:
# Remove - Field names to remove
remove:
# Regular expression - Regular expression for parsing
# [string]
```

```
regex:
# Regular expression list - List of regular expressions for parsing
regexList:
# Field name expression
# [string]
fieldNameExpression:
# Pattern - Grok pattern for parsing
# [string]
pattern:
# Pattern list - List of grok patterns
```

10.5.19. Persistent-Queue.Yml

`persistent-queue.yml` contains settings for configuring persistent queues.

persistent-queue.yml

```
# Buffer size limit - The maximum number of events to hold in memory before writing t
# [number; default: 1000]
maxBufferSize:
# File size limit - The maximum size to store in each queue file before closing and c
# Enter a numeral with units of KB, MB, etc.
# [string; default: 10MB]
maxFileSize:
# Queue size limit - The maximum disk space that the queue can consume before queueir
# [string; default: 1GB]
maxSize:
# Mode - Queue operation mode
# One of: always | never | smart
# [string; default: always]
mode:
# Queue file path - The location for the persistent queue files
# [string]
path:
# Compression - Codec to use to compress the persisted data
# One of: gzip | none
# [string]
compress:
# Commit frequency - The number of events to send downstream before committing that S
# [number]
commitFrequency:
```

Copy

10.5.20. Regexes.Yml

`regexes.yml` maintains a list of regexes. Cribl's Regex Library ships under `default`. Each regex is listed according to the following pattern:

regexes.yml

```
# Regex ID - Unique identifier for the regex pattern
# [string; required]
id:
# Library - Library classification for the regex
# One of: cribl | cribl-custom | custom
# [string]
lib:
# Description - Description of the regex pattern
# [string]
description:
# Regex pattern - The regular expression pattern
# [string; required]
regex:
# Sample data - Optionally, paste in sample data to match against this regex
# [string]
sampleData:
# Tags - Tags associated with the regex
# [string]
tags:
```

Copy

10.5.21. SAMPLES.YML

`samples.yml` contains metadata about about stored sample data files (size, number of events, date created, name, and so on).

samples.yml

```
# ID - Unique identifier for the sample
# [string; required]
id:
# Sample Name - Name of the sample file
# [string; required]
sampleName:
# Pipeline ID - Associate with Pipeline - Select a pipeline to associate with sample
# sure. Deprecated.
# [string]
pipelineId:
# Pack ID - ID of the pack this sample belongs to
# [string]
packId:
# Is Pack Only - Whether this sample only exists within this pack and cannot be turne
# [boolean]
isPackOnly:
# Description - Brief description of this sample file. Optional.
# [string]
description:
# Tags - One or more tags related to this sample file. Optional.
# [string]
tags:
# Library - Library classification for the sample
# [string]
lib:
# Created - Timestamp when the sample was created
# [number]
created:
# Modified - Timestamp when the sample was last modified
# [number]
```

10.5.22. Schemas.Yml

`schemas.yml` stores configuration data for the Knowledge > Schema Library.

schemas.yml

```
# Schema ID - Unique identifier for the schema
# [string; required]
id:
# Description - Description of the schema
# [string]
description:
# Schema - JSON schema matching standards of draft version 2019-09
# [string; required]
schema:
# Tags - Tags associated with the schema
# [string]
tags:
```

Copy

10.5.23. Scripts.Yml

`scripts.yml` stores configuration data for scripts configured at **Settings** > **Global** > **Scripts**:

scripts.yml

```
# Script ID - Unique identifier for the script (cannot contain forward slashes)
# [string; required]
id:
# Command - Command to execute for this script
# [string; required]
command:
# Description - Description of the script
# [string]
description:
# Arguments - Arguments to pass when executing this script
args:
# Env variables - Extra environment variables to set when executing script
env:
```

Copy

10.5.24. Service.Yml

`service.yml` maintains configuration for Cribl Stream service processes.

In the UI, you can configure some of these settings on the Leader at **Settings** > **Global** > **System** > **Service Processes** > **Services**.

Be careful about reducing the predefined `memoryLimit` values. Extremely low values can prevent some Cribl Stream components from functioning.

service.yml

```
# Service configuration - Configuration for various Cribl services
# Available service types: connections, metrics, notifications, stream_connections, e
connections:
  # Connections listener number of processes - Negative values are offsets from the n
  # on the host.
  # [number; default: 1]
  procs:
  # Connections listener heap memory limit - Heap memory limit for the connection lis
  # [string; default: 2GB]
  memoryLimit:
metrics:
  # Metrics number of processes - Single process; values above 1 will be treated as 1
  # offsets from the number of available cores on the host.
  # [number; default: 1]
  procs:
  # Metrics heap memory limit - Heap memory limit for the metrics process
  # [string; default: 2GB]
  memoryLimit:
notifications:
  # Notifications number of processes - Single process; values above 1 will be treate
  # are offsets from the number of available cores on the host.
  # [number; default: 1]
  procs:
  # Notifications heap memory limit - Heap memory limit for the notifications process
  # [string; default: 2GB]
  memoryLimit:
stream_connections:
  # Stream connection processes - Negative values are offsets from the number of avai
  # [number; default: 1]
  procs:
  # Auto scale - Do not modify this setting. Defines whether Leader can start the ser
  # Worker node connects to it.
  # [boolean]
  autoScale:
  # Stream connection memory limit - Heap memory limit for the Stream connection proc
  # [string; default: 2GB]
  memoryLimit:
edge_connections:
  # Edge connection processes - Negative values are offsets from the number of availa
  # [number; default: 1]
  procs:
  # Edge auto scale - Do not modify this setting. Defines whether Leader can start th
```

```
  # qualifying Edge node connects to it.
  # [boolean]
  autoScale:
  # Edge connection memory limit - Heap memory limit for the Edge connection processe
  # [string; default: 2GB]
  memoryLimit:
# Connection proxy service - Proxy service that runs on a standby Leader node to forw
# from port 4200 to the primary Leader.
connection_proxy:
  # Connection proxy processes - Number of processes to spawn for connection proxy se
  # Set to 0 to disable the service. Negative values are offsets from the number of a
  # [number; default: 0]
```

10.5.25. VARS.YML

`vars.yml` stores configuration data for the Knowledge > Global Variables Library.

As of Cribl Stream 4.16.0, new Worker Groups do not have sample variables by default.

vars.yml

```
# Name - Global variable name
# [string; required]
id:
# Library - Library classification for the variable
# One of: cribl | cribl-custom | custom
# [string]
lib:
# Description - Description of the global variable
# [string]
description:
# Type - Data type of the variable
# One of: string | number | encryptedString | boolean | array | object | expression |
# [string; default: any; required]
type:
# Value - Value or expression for the variable
# [string; required]
value:
# Tags - Tags associated with the variable
# [string]
tags:
# Arguments - For expression type variables, define function arguments
args:
```

Copy

11. Troubleshoot Cribl Stream

Use the resources in this section to diagnose and resolve issues in Cribl Stream:

- Known Issues: Get information about issues that affect Cribl Stream, along with temporary workarounds and fix versions when applicable.
- Diagnosing Issues: Create and export diagnostic bundles to share with Cribl Support.
- Common Errors and Warnings: Learn how to resolve common errors and warning messages in Cribl Stream.
- Git Push Errors: Learn how to resolve common errors when pushing a `git` commit.
- Git Remote Repos & Trusted CAs: Learn how to resolve errors related to self-signed certificates when pushing or pulling a `git` commit.
- Sample Logs for Login Scenarios: Get examples of the event series that Cribl Stream records in common login scenarios.

11.1. Known Issues

View filterable Known Issues for Cribl Stream. Check if issues are resolved or find temporary workarounds.

11.2. Diagnose Issues

To help diagnose Cribl Stream problems, you can share a diagnostic bundle with Cribl Support. The bundle contains a snapshot of configuration files and logs at the time the bundle was created, and gives troubleshooters insights into how Cribl Stream was configured and operating at that time.

What's in the Diagnostic Bundle

The bundle includes the following subdirectories (and their contents) of `$CRIBL_HOME`:

- `.../default/*`
- `.../local/*` - except for `/local/cribl/auth/`, to exclude sensitive files.
- `.../log/*`
- `.../groups/*`
- `.../state/jobs/*` - will return all jobs if left empty.

As a security measure, the bundle excludes all `.crt`, `.pem`, `.cer`, and `.key` files from all `$CRIBL_HOME` subdirectories.

Create and Export a Diagnostic Bundle

If you're managing your own Cribl Stream deployment (single-instance Stream/Edge or distributed Stream/Edge), you can create and securely share bundles with Cribl Support, either from the UI or via the CLI.

With a Cribl.Cloud deployment, create bundles and securely share them with Cribl Support directly from the UI. In the Using the UI instructions below, **Send to Cribl Support** will always be enabled, and the **Export**/download option will be unavailable.

Requirements

Using either medium, you'll need outbound internet access to `https://diag-upload.cribl.io` and a valid support case number. The above URL works only when using the `cribl diag` command or uploading using the Cribl Stream UI. (So connecting directly to it with your web browser will fail.)

If your organization doesn't permit outbound access to `https://diag-upload.cribl.io` to upload from within Cribl, you can also submit diags directly via the Support Portal (login required). If none of

> these options work with your organization's policies, please work directly with your Support engineer to find a solution.

Open your support case before sending a diagnostic bundle to Cribl Support. Then, when creating the bundle, be sure to use the case number provided. Case numbers are 8 digits in length, and you must include any leading zeros. For example, `00001234`.

Use the UI

Depending on the issue you're debugging, you can create a bundle either from the Leader or from an individual Worker Node.

UI Options

The **Create & Export Diag Bundle** modal provides these additional options:

- **Rename .js files**: When enabled, appends `.txt` to JavaScript files in the bundle. Defaults to toggled on.
- **Include metrics**: When enabled, includes the persistent internal metrics store used by Leaders and single-instance deployments in the bundle. This metrics store can grow to many gigabytes over time. Individual Worker Node do not maintain this metrics store, their metric-related information still appears in logs, diag artifacts, and other files regardless of this setting. Defaults to toggled on.
- **Include git**: Includes the last 14 days of git log in the bundle, identifying each commit's message, author, timestamp, and files changed. Defaults to toggled on. Unavailable on Leader-managed Stream Workers and Edge Nodes, as well as on Outpost Nodes.
- **Include top talkers**: Adds detailed metrics for the top 10 Sources, Destinations, Pipelines, Routes, and Packs (based on the last 24 hours) to the diagnostic bundle. These JSON files (such as, `top_sources.json`) contain in/out events and bytes, aiding in troubleshooting and performance analysis. Disabled by default to minimize bundle size, this requires Leader Node access to the `/system/metrics/query` API.
- **Latest number of jobs**: The number of jobs (and all tasks) the bundle will include. Defaults to empty, which means unlimited jobs.
- **Include installation logs**: Includes Windows MSI logs, only available on Windows deployments of Cribl Edge. Default is toggled off.
    - This option is not available for Linux deployments.
    - If MSI logs are available, the diag bundle will contain them. If no MSI logs are present, the bundle will not contain them even with this option toggled on.

Cribl Support will advise you on changing the defaults here. Because the Leader metrics store can be large, they often recommend disabling **Include metrics** unless they specifically request metrics store contents for your case.

From the Leader Node

To create a bundle from the Leader Node, go to **Settings** > **Global** > **Diagnostics** and select **Create Diagnostic Bundle**.

- To share the bundle with Cribl Support, toggle **Send to Cribl Support** on. Enter your case number, and then select **Export**.
- To download the bundle locally to your machine (on-prem only), select **Export**.

From Worker Nodes

To create a bundle from individual Worker Nodes, you must be able to access the Worker Node's UI. You can easily use the Leader to get authenticated access to each Worker Node, if you first enable the parent Worker Group's **Teleport** setting:

1. In the Cribl Stream sidebar, select **Worker Groups**.
2. Make sure the relevant Worker Group's **Teleport** toggle is on.
3. Select the displayed number of Worker Nodes to access the **Worker Nodes** page.
4. In the **GUID** column, select the link for the Worker Node you're debugging.
5. Look for the purple border to confirm that you've teleported to that Worker Node.
6. Select **Worker Node Settings** (top right) > **Diagnostics** (lower left) > **Diagnostic Bundle**, and select **Create Diagnostic Bundle**.
7. Use the appropriate **Export** options listed above for the Leader.

On on-prem deployments, new and previously created bundles are stored in `$CRIBL_HOME/diag`. They're also listed in the UI, where you can re-download them or share them with Cribl Support.

Use the CLI

To create a bundle using the CLI, use the `diag` command. (This option is on-prem only.) Depending on the issue you're debugging, you can create a bundle either from the Leader host or from an individual Worker Node host.

diag command CLI

```
# $CRIBL_HOME/bin/cribl diag
Usage: [sub-command] [options] [args]

Commands:
create       - Creates diagnostic bundle for this instance, args:
  [-d]                  - Run create in debug mode.
  [-j]                  - Do not append '.txt' to js files.
  [-k]                  - Include metrics for the top 10 Sources, Destinations, Pipel
  [-t <maxIncludeJobs>] - Maximum number of jobs (and all associated tasks) to includ
  [-M]                  - Exclude the persistent internal metrics store from the bund
  [-g]                  - Exclude git log from bundle.
  [-i]                  - Include install logs.
heapsnapshot - Generate heap snapshot of a Cribl process, args:
  [-p <pid>] - The pid of the process to dump the heap snapshot
list         - List existing diagnostic bundles
send         - Send diagnostics bundle to Cribl Support, args:
   -c <caseNumber> - Cribl Support Case Number
  [-p <path>]      - Diagnostic bundle path (if empty then new bundle will be created


## Creating a diagnostic bundle
# $CRIBL_HOME/bin/cribl diag create
Created Cribl Stream/Edge diagnostic bundle at /opt/cribl/diag/ <product>-<hostname>-

## Creating and sending a diagnostic bundle
# $CRIBL_HOME/bin/cribl diag send -c <caseNumber>
Sent Cribl diagnostic bundle to Cribl Support

## Sending a previously created diagnostic bundle
# $CRIBL_HOME/bin/cribl diag send -p /opt/cribl/diag/ <product>-<hostname>-<datetime>
Sent Cribl diagnostic bundle to Cribl Support
```

Copy

For complete command syntax, see diag.

Taking a diag from a Cribl Edge Node running on Windows requires setting the `CRIBL_VOLUME_DIR` environment variable. See `diag` for more information.

Include CPU Profiles

This section is applicable only to Cribl Stream.

If Cribl Support asks you to grab CPU profiles of Worker Processes, follow these steps:

1. Use `top` or `htop` on the Worker Node to identify Worker Node PIDs consuming a lot of CPU.
2. See Measuring CPU or API Load for instructions on accessing the UI's **Profile** options (for your deployment type), and generating and saving profiles.
3. Find the Worker Processes matching the PIDs you identified above.
4. Select **Profile** on each. Start with the default 10-second **Duration**.
5. Once the profile is displayed, save it to a `.json` file. (See details at the above link.)
6. Repeat steps 3-5 for other CPU-intensive Worker Processes.
7. Upload the profile `.json` files to Cribl Support.

On an already CPU-starved Worker Node, profiling might fail with an error message, or just hang. In this case, you might need a few retries to get a successful profile.

Include Memory Snapshots

Cribl.Cloud does not support taking memory snapshots. This feature is for on-prem deployments only.

If you encounter a memory leak, Cribl Support might request a memory snapshot of a `cribl` process to better understand where the leak is occurring.

There are two ways to generate a memory snapshot:

- Automatically
- Manually

Automatically Capture a Memory Snapshot

To automatically capture memory snapshots when Worker Processes approach or exceed memory limits, turn on the **Enable heap snapshots setting** in the Worker Group's **General Settings**. Navigate to **Worker Group Settings** > **System** > **Worker Processes** and toggle on. Only the two most recent heap snapshots are retained. Older snapshots are automatically deleted. This behavior cannot be modified.

This setting should only be enabled if recommended by support or if you are experiencing out-of-memory issues. Be aware that enabling this setting may impact system performance and storage usage.

For details, see Worker Processes.

Manually Capture a Memory Snapshot

Prerequisites

To manually generate a reliable memory snapshot for Cribl Stream diagnostics, ensure two key conditions are met:

1. **Correct `CRIBL_HOME` Environment Variable**: This variable points to the Cribl Stream installation directory. Open a new terminal session and verify its value using `echo $CRIBL_HOME`. If empty, set it appropriately for your operating system (for example, `export CRIBL_HOME=/path/to/cribl` on Linux, `setex cribl C:\Program Files\Cribl /m` on Windows).
2. **Run as the `cribl` User**: The `diag heapsnapshot` command requires permissions to create the snapshot file within the run Cribl Stream directory. Use `sudo su - cribl` to switch to the cribl user before running the command. Remember to exercise caution with `sudo`. On Windows, if Cribl is set to run under a specific user account, log in as that user before running Cribl. Alternatively, you can run Cribl Stream as an `Administrator`.

Capture the Memory Snapshot

To capture the memory snapshot:

1. Note the PID of the process of interest.
2. In a terminal, `cd` to the directory where Cribl Stream is installed.

```
cd $CRIBL_HOME/bin
```

Copy

3. Run the `diag heapsnapshot` command with the PID you noted in step 1.

- For Linux:

```
./cribl diag heapsnapshot -p <pid>
```

Copy

Locate the PID on Windows

There are three methods to identify the Process ID (PID) for `cribl.exe`:

1. **Task Manager**: Open the Task Manager application. Locate the **Processes** tab and search for `cribl.exe` in the list. The PID will be displayed in the corresponding column.
2. **Command Line Interface (CLI)**: Open a Command Prompt window. Run the following command:

```
tasklist /FI "IMAGENAME eq cribl.exe"
```

Copy

This command will list all running processes. The line containing `cribl.exe` will also display the PID in the corresponding column (usually the second column).

3. **PowerShell**: Open a PowerShell window. Run the following command:

```
PowerShell
Get-Process -Name cribl
```

Copy

This command will list details about running processes. The output will include the PID for any process named `cribl`.

Once you locate the PID, run the `diag heapsnapshot` command:

1. Navigate to the Cribl Stream installation directory: For example, `C:\Program Files\Cribl\bin` or `$CRIBL_HOME\bin`.
2. Run the `diag heapsnapshot` command with the PID.

```
./cribl diag heapsnapshot -p <pid>
```

Copy

11.3. Common Errors And Warnings

This page lists common error and warning messages that you might find in Cribl Stream's internal logs and/or UI. It includes recommendations for resolving the errors/warnings. Messages are grouped by component (Sources, Destinations, and so on).

Whole message texts are not listed in every case. They may have different capitalization than the messages you may encounter in the product.

Where events are written to log files, they might reside in different logs variously devoted to data processing, cluster communication, or the REST API. (For details, see Types of Logs.) We note the log type where necessary.

NodeJS serves as the Cribl Stream backend and has a large collection of well documented errors of its own. Some of system-level are listed on this page with the appropriate action to take. The remainder are documented at https://nodejs.org/api/errors.html, specifically in the `Common system errors` section of that page.

Web UI

Duplicate GUID

When you deploy and first run Cribl Stream software, it generates a GUID (globally unique identifier) and stores it in a `.dat` file located in `CRIBL_HOME/local/cribl/auth/`. For example:

```
# cat CRIBL_HOME/local/cribl/auth/676f6174733432.dat

{"it":1570724418,"phf":0,"guid":"48f7b21a-0c03-45e0-a699-01e0b7a1e061"}
```

Copy

When deploying Cribl Stream as part of a host image or VM, be sure to remove this file, so that you don't end up with duplicate GUIDs. The file will regenerate on the next run.

"Failed to fetch"

Occurs only when accessing **Data** > **Collectors**. It can occur with ad blockers, or in the Brave browser (because the newly loaded page's URL includes the string `collector`).

Recommendation

If you have an ad blocker, allowlist the `https://<hostname>:9000/jobs/collectors` (single-instance) URL or the `https://<leadernode>:9000/m/<group_name>/jobs/collectors` (Leader) URL.

"No workers registered"

Occurs during a live data capture, when there are no Workers available to fulfill the capture request. This error can be triggered if you attempt to capture events too quickly following a commit & deploy.

Recommendation

- If you recently committed and deployed changes, wait 30 seconds and retry the live capture.
- Navigate to the **Workers** page and confirm there are Workers currently registered with the Leader. Note that the error appears on a per-Group basis, which means that even if you have Workers registered, they may not be in the Group you're capturing live data from.

"The Config Helper service is not available..."

Full Text: "The Config Helper service is not available because a configuration file doesn't exist or the settings are invalid. Please fix it and restart Cribl server."

Occurs only on Leader Nodes. Each Worker Group relies on a config helper process that runs on its behalf on a Leader Node, and that process is not currently running. This error is usually triggered by restarting Cribl Stream on the Leader Node, and then trying to access one of these features before the Cribl Stream server is fully up:

- **Groups** or **Configure** link (3.x).
- **Worker Groups** menu (2.x).
- **Manage** > **Groups** or **Manage** > **Workers** tabs (4.x).

Recommendation

Try again in a few seconds. If you've made changes to the backend or filesystem permissions, stop Cribl Stream, and make sure the `cribl` user owns all relevant assets, before restarting: `sudo chown -R cribl.cribl /opt/cribl`

"KMS saved, but secret exists in destination location. No data was migrated."

This will appear when you attempt to save KMS Settings with a **Secret Path** entry that references a secret that already exists in HashiCorp Vault. It can also occur with OpenID Connect remote authentication. Cribl Stream has aborted the remote write to avoid overwriting an external shared secret.

Recommendation

Edit the Leader Node's `kms.yml` file to use `provider: local`. Then restart Cribl Stream to correct the path conflict.

"The number of Worker Processes has been limited by the license."

Workers use their local license until they connect to the Leader and download their current license data, which is why the message appears in Logs.

Recommendation

Ignore the message. Once the Worker checks in with the Leader and receives the production license, the rest of the Workers will start.

Sources

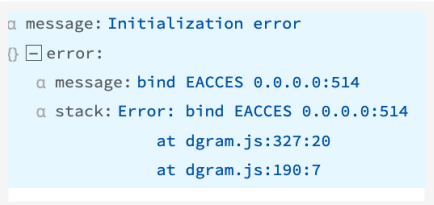
Here are some general errors that can occur with many Sources. When you encounter a problem with a specific Source, check the "Troubleshooting" section in its documentation as well for potential solutions.

"bind EACCES 0.0.0.0:514"

Cribl Stream doesn't have proper privileges to bind to the specified IP and port. Usually, this simply indicates that Cribl Stream is running as a non-root user, but a Source was accidentally configured to use a privileged port below 1024.

Recommendation

Check your Cribl Stream Sources' configuration. The error's `channel` field will indicate which input is affected. If you choose to enable access to the port range below 1024, see our configuration instructions for systemd and initd.



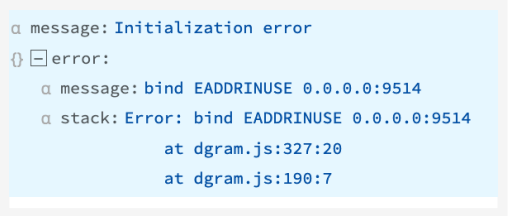
EACCES error

"bind EADDRINUSE 0.0.0.0:9514"

The interface and port combination is already in use by another process, which might or might not be Cribl Stream.

Recommendation

Check your Cribl Stream Sources' configuration. The error's `channel` field will indicate which Source is affected, but this error means that one or more other inputs are also using the same IP/port combination.



EADDRINUSE error

Destinations

Here are some general errors that can occur with many Destinations. When you encounter a problem with a specific Destination, check the "Troubleshooting" section in its documentation as well for potential solutions.

"Sending is blocked"

The Destination is blocking. This can occur with any TCP-based Destination, and is logged only after 1 second of blocking. This can also occur between Worker and Leader when the Worker can't connect to the Leader Node to send metrics data. When triggered by cluster communication, the warning will be in the Worker's API log.

"Begin backpressure" and "end backpressure"

The Destination is blocking. Like the "sending is blocked" message, "begin backpressure" is logged only after 1 second of blocking. It is logged only once while backpressure is occurring (at the start), and it will always be followed by the "end backpressure" message.

"Unable to write to the filesystem, disk space low"

Full Text: "Unable to write to the filesystem, disk space low, see destination status and logs for more information"

This error can occur if the available disk space on the host is below the specified minimum storage limit.

Recommendation

To check this limit, in **Worker Group Settings**, select **General Settings** > **Limits**, and then select **Storage**. Make sure the available disk space in the location you're writing to is above the **Min free disk space** listed. See Storage in Worker Group Settings for more information on storage limits.

To prevent Cribl service disruptions due to a full data partition, we recommend setting your staging and PQ directories to a partition other than `/opt/cribl` (or `$CRIBL_HOME` installation).

TLS

“Certificate has expired”

Validate server certs or **Validate client certs** is enabled, and the peer’s certificate has expired.

Recommendation

Disable **Validate server certs** or **Validate client certs** (depending on whether Cribl Stream is serving as the client or server), so that encryption can still occur without authentication. Or renew the expired certificate.

“Client network socket disconnected…”

Full Text: “Client network socket disconnected before secure TLS connection was established”

Typically caused by a TLS protocol version mismatch between the client and server.

Recommendation

Verify that client’s and server’s TLS settings use the same minimum/maximum TLS version.

“Unable to get local issuer certificate”

Client can’t validate the server certificate’s CA (that is, the issuer) because it doesn’t trust the CA certificate. Or vice versa (server can’t validate client’s certificate CA) if mutual auth is enabled.

Recommendation

The CA certificates used by the server’s leaf certificate must be trusted by the client. See Configuring CA certs.

“Self signed certificate”

The client or server was presented with a self-signed cert from the peer, but that cert is not trusted.

Recommendation

The self-signed certificate must be trusted by the peer to which it’s presented. See Configuring CA certs.

“Hostname/IP does not match certificate’s altnames”

Full Text: “Hostname/IP does not match certificate’s altnames: host: <server hostname> is not in the cert’s list” or: “Hostname/IP does not match certificate’s altnames: IP: <server IP> is not in the cert’s list”

The server’s hostname/FQDN used on the client is not found in the CN or SAN attribute of the server’s certificate.

Recommendation

Examine the CN and/or SAN attribute, to see which names are listed that can be used as the server hostname/FQDN on the client. **CN values with spaces are not supported as hostnames/FQDNS.** If there isn't a SAN attribute, then a new cert will need to be issued.

"No protocols available"

Full Text: "140244944713600:error:141E70BF:SSL routines:tls_construct_client_hello:no protocols available:"

The highest TLS protocol available by the client is still too low for the server to support.

Recommendation

Review the minimum/maximum TLS version settings on the client and server, to ensure that they overlap.

"Wrong version number"

Full Text: "140251374995328:error:1408F10B:SSL routines:ssl3_get_record:wrong version number"

The client is using TLS but the server is probably not configured for TLS.

Recommendation

Review the TLS settings on the server.

Pipelines/Functions

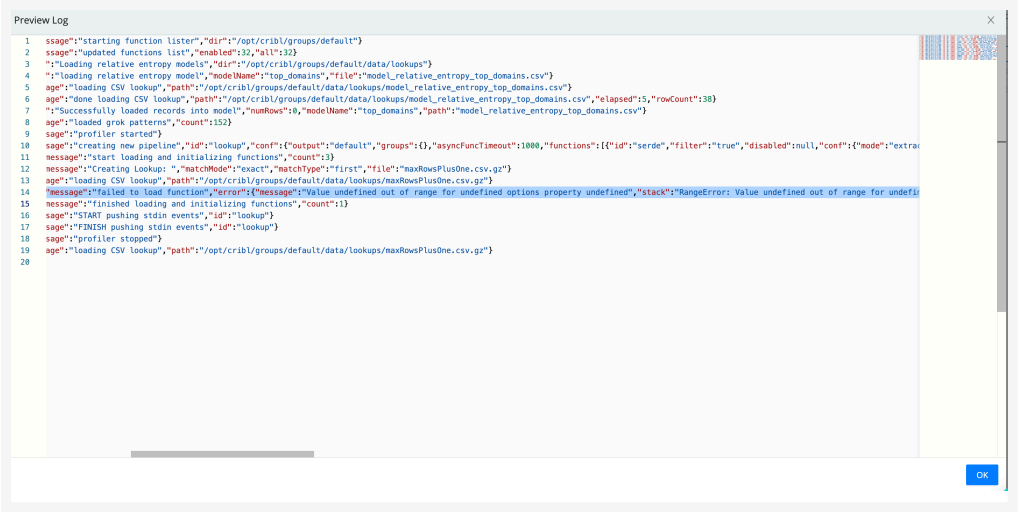
These events will be in the Worker Process logs.

"Failed to load function...Value undefined out of range..."

A Lookup Function attempted to load a Lookup table that exceeded Node.js' hard size limit of 16,777,216 (that is, 2^{24}) rows.

Recommendation

Split the Lookup table to smaller tables, or use the Redis Function.



Oversized Lookup table error

Diag Command

“You are running Cribl Stream CLI as user=root…”

Full Text: “WARNING: You are running Cribl Stream CLI as user=root, while the binary is owned by the user=cribl. This may change the ownership of some files under CRIBL_HOME=/opt/cribl. Please make sure all files under CRIBL_HOME=/opt/cribl are owned by the user=cribl.”

This is caused by improper ownership on `$CRIBL_HOME`, and will cause some files to be missing from the diag.

Recommendation

Execute the `chown` command on the entire `$CRIBL_HOME` directory, so that everything can be owned by the proper user. Afterward, run the `./cribl diag create` command again.

Distributed Deployment

11.4. Git Push Errors

This page anticipates common errors you might see in Cribl Stream's UI, or in the `git` CLI, when pushing a commit.

Failed to Push Some Refs

Your first push to a remote repo might fail with one of several `failed to push some refs` errors.

As a first step in debugging these errors, edit the `$CRIBL_HOME/.git/config` file to make sure that its `name` and `email` key values match the credentials you've set on your repo provider or git server.

Also make sure that the `remote "origin"` key value matches the remote you set when you connected to the remote repo. This example shows all three keys, with placeholder values:

```
[user]
        name = <your-login-name>
        email = <email@example.com>
[remote "origin"]
        url = https://<user-name>:<token>@github.com/<username>/<repo-name>
```

Copy

Next, verify the remote repo from the command line, as follows:

```
cd $CRIBL_HOME/.git
git remote -v
```

Copy

In response, `git` should echo your configured remote twice - once for `fetch` and once for `push` operations.

If all of the above settings are correct, the `push` is very likely blocking because the remote repo has some commit history, or was simply created with a `readme.md` file. For command-line instructions to remedy this - by syncing your local repo to its remote - see GitHub's Dealing with Non-Fast-Forward Errors topic.

Large Files Detected

A push command might also trigger "large file" warnings or, more seriously, errors of this form (CLI/GitHub example):

```
remote: warning: File data/lookups/geo.mmdb is 60.12 MB; this is larger than GitHub's
remote: error: GH001: Large files detected. You may want to try Git Large File Storag
remote: error: Trace: [############################################################
remote: error: See http://git.io/iEPt8g for more information.
remote: error: File groups/default/data/lookups/largelookup.csv is 313.91 MB; this ex
```

Copy

Cribl recommends adding such large files to `.gitignore`, to exclude them from subsequent `push` commands. As the above examples show, typical culprits are large `.csv` or `.mmdb` lookup files. A simple option is to place these files in a `$CRIBL_HOME` subdirectory that's already listed in `.gitignore` - for details, see About Lookups.

Other available workarounds include staging such files **outside** `$CRIBL_HOME`, or using plugins to accommodate the large files. For GitHub-specific options, see Working with Large Files.

See Also

- Git Remote Repos with Trusted CAs

11.5. Git Remote Repos & Trusted CAs

If you are using an internal Git server, a self-signed certificate might prevent Cribl Stream from successfully pushing commits to the origin. You might see errors like these when pushing (or pulling) via the CLI:

```
SSL certificate problem: self signed certificate in certificate chain
SSL certificate problem: unable to get local issuer certificate
```

Copy

Resolving the Errors

To ensure that Git trusts your self-signed certificate, follow these steps:

1. Obtain the certificate chain (root, intermediates, and leaf) for the Git server.
2. As the `cribl` user, run this command: `git config http.sslCAInfo /path/to/certs.pem`
3. Test with this command: `git push origin` Verify that this throws no errors.

Obtain the Certificate Chain (TLS/SSL)

Use these steps to enable Worker-to-Leader mutual authentication:

A. Validate the Client Certs

If you are using an internal certificate authority, obtain a copy of the CA public certificate, then add it to `/etc/systemd/system/cribl.service`:

```
...
[Service]
Environment="NODE_EXTRA_CA_CERTS=/opt/cribl/local/cribl/auth/certs/ca.pem"
...
```

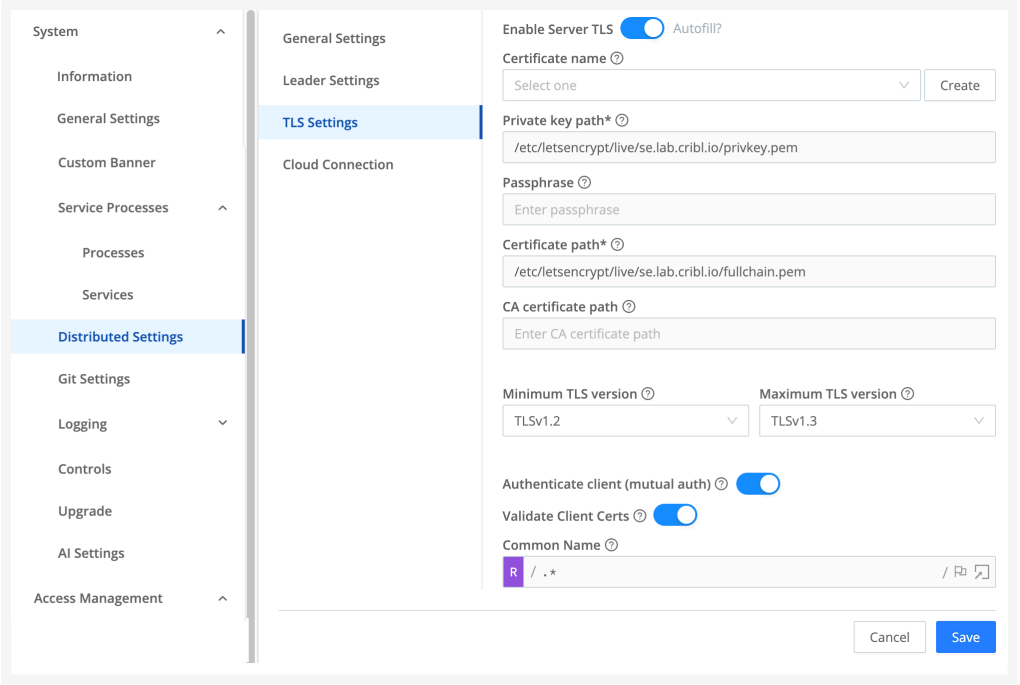
Copy

For details, see CA Certificates and Environment Variables.

B. Simplify the Common-Name Regex

The common-name regex (if required) should omit the `CN=` at the beginning of the **Common Name** field. The following example will match all immediate subdomains of `se.lab.cribl.io`, like `madsci.se.lab.cribl.io`.

If you disable **Validate Client Certs**, Cribl Stream will match only on common names.



Common Name example

C. Extract SSL Certificate Info

As in this example:

```
openssl x509 -in certificate.pem -text -noout
```

Copy

D. Dump the Certificate Chain from the Server

As in this example:

```
echo "" | openssl s_client -host www.google.com -port 443 -showcerts 2>&1 | sed -n '/
```

Copy

11.6. Sample Logs For Login Scenarios

This page lists sample event series that Cribl Stream records in common login scenarios. Cribl Stream captures event series for both successful and failed login attempts. Note that these samples omit the following repeating fields that are captured in each log:

- `cid:api`
- `channel:cribl`
- `level:info`

Successful Authentication Using the `memberOf` Attribute


```
{"time":"2022-03-30T03:09:55.569Z","message":"Running LDAP
search","searchBase":"dc=mydomain,dc=com","filter":"(sAMAccountName=admin)","options":
{"scope":"sub","filter":"(sAMAccountName=admin)"}}

{"time":"2022-03-30T03:09:55.575Z","message":"LDAP user search
results","count":1,"username":"REDACTED","user":
{"dn":"CN=admin,CN=Users,DC=mydomain,DC=com","memberOf":
["CN=admingrp,CN=Users,DC=mydomain,DC=com"]}}

{"time":"2022-03-
30T03:09:55.582Z","channel":"LDAPMapper","level":"debug","message":"External groups
found","groups":["admingrp","Users"]}

{"time":"2022-03-30T03:09:55.583Z","message":"Successful
login","user":"admin","provider":"ldap:win2008ad1.mydomain.com:389"}
```


Successful Authentication With Fallback When a Bad Login Is Triggered


```
{"time":"2022-03-30T03:10:48.389Z","message":"Running LDAP
search","searchBase":"dc=mydomain,dc=com","filter":"(sAMAccountName=admin)","options":
{"scope":"sub","filter":"(sAMAccountName=admin)"}}

{"time":"2022-03-30T03:10:48.393Z","message":"LDAP user search
results","count":1,"username":"REDACTED","user":
{"dn":"CN=admin,CN=Users,DC=mydomain,DC=com","memberOf":
["CN=admingrp,CN=Users,DC=mydomain,DC=com"]}}
```

{"time":"2022-03-30T03:10:48.399Z","message":"Attempting fallback to local authentication","reason":"failed login","provider":"ldap:win2008ad1.mydomain.com:389","user":"admin"}

{"time":"2022-03-30T03:10:48.400Z","message":"Successful login","user":"admin","provider":"local"}

Failed Authentication with Fallback When a Bad Login Is Disabled

{"time":"2022-03-30T03:11:54.175Z","message":"Running LDAP search","searchBase":"dc=mydomain,dc=com","filter":"(sAMAccountName=admin)","options":{"scope":"sub","filter":"(sAMAccountName=admin)"}}

{"time":"2022-03-30T03:11:54.178Z","message":"LDAP user search results","count":1,"username":"REDACTED","user":{"dn":"CN=admin,CN=Users,DC=mydomain,DC=com","memberOf":["CN=admingrp,CN=Users,DC=mydomain,DC=com"]}}

{"time":"2022-03-30T03:11:54.184Z","level":"warn","message":"Failed login","user":"admin","provider":"ldap:win2008ad1.mydomain.com:389","details":{"message":"80090308: LdapErr: DSID-0C0903A9, comment: AcceptSecurityContext error, data 52e, v1db1\u0000 Code: 0x31","stack":"Error: 80090308: LdapErr: DSID-0C0903A9, comment: AcceptSecurityContext error, data 52e, v1db1\u0000 Code: 0x31

Incorrect Proxy Bind Password With Fallback When a Fatal Error Is Triggered

{"time":"2022-03-30T03:04:16.400Z","level":"warn","message":"Authentication Provider Error","provider":"ldap:win2008ad1.mydomain.com:389","fatal":{"message":"80090308: LdapErr: DSID-0C0903A9, comment: AcceptSecurityContext error, data 52e, v1db1\u0000 Code: 0x31","stack":"Error: 80090308: LdapErr: DSID-0C0903A9, comment: AcceptSecurityContext error, data 52e, v1db1\u0000 Code: 0x31\n at Function.parse [snip]

{"time":"2022-03-30T03:04:16.401Z","message":"Attempting fallback to local authentication","reason":"provider error","provider":"ldap:win2008ad1.mydomain.com:389","user":"admin","error":{"message":"80090308: LdapErr: DSID-0C0903A9, comment: AcceptSecurityContext error, data 52e, v1db1\u0000 Code: 0x31","stack":"Error: 80090308: LdapErr: DSID-0C0903A9, comment: AcceptSecurityContext error, data 52e, v1db1\u0000 Code: 0x31

Successful Authentication Without `memberOf` to Trigger Group Search but Without Group Memberships


```
{"time":"2022-03-30T03:15:21.410Z","message":"Running LDAP
search","searchBase":"dc=mydomain,dc=com","filter":"(sAMAccountName=admin)","options":
{"scope":"sub","filter":"(sAMAccountName=admin)"}}

{"time":"2022-03-30T03:15:21.416Z","message":"LDAP user search
results","count":1,"username":"REDACTED","user":
{"dn":"CN=admin,CN=Users,DC=mydomain,DC=com"}}

{"time":"2022-03-30T03:15:21.422Z","message":"Running LDAP
search","searchBase":"dc=mydomain,dc=com","filter":"
(member=CN=admin,CN=Users,DC=mydomain,DC=com)","options":{"scope":"sub","filter":"
(member=CN=admin,CN=Users,DC=mydomain,DC=com)"}}

{"time":"2022-03-30T03:15:21.424Z","message":"LDAP groups search
results","count":0,"opts":{"groupSearchBase":"dc=mydomain,dc=com","memberSearch":
{"memberField":"member","memberDN":"CN=admin,CN=Users,DC=mydomain,DC=com"}},"groups":
[]}

{"time":"2022-03-
30T03:15:21.425Z","channel":"LDAPMapper","level":"debug","message":"External groups
found","groups":[]}

{"time":"2022-03-30T03:15:21.427Z","message":"Successful
login","user":"admin","provider":"ldap:win2008ad1.mydomain.com:389"}
```

12. CrowdStream

CrowdStream is a special Cloud-hosted version of Cribl Stream, available through CrowdStrike Falcon LogScale. This collaboration is tailored to forward data from a wide range of sources into LogScale for XDR analysis and log management.

En route, you can process events using all of Cribl Stream's options for aggregating, sampling, redacting, cloning, and otherwise shaping and directing your data. CrowdStream provides parallel options to route data to cost-effective Amazon S3 storage, and to S3 data lakes, for later replay and analysis.

Getting Started

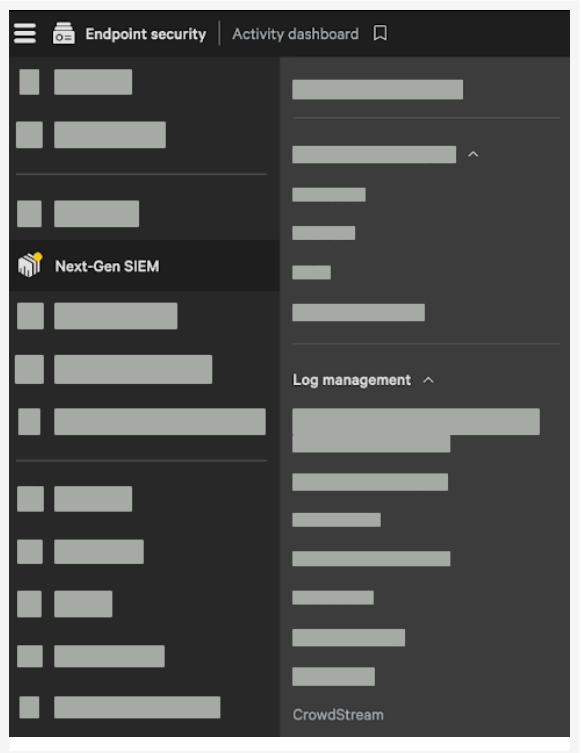
These steps assume that you've set up your CrowdStrike Falcon LogScale account, obtained your corresponding API token.

You can get access to CrowdStream in two ways:

- Through Falcon Next-Gen SIEM interface
- Through Falcon LogScale interface

To log into CrowdStream from Next-Gen SIEM interface:

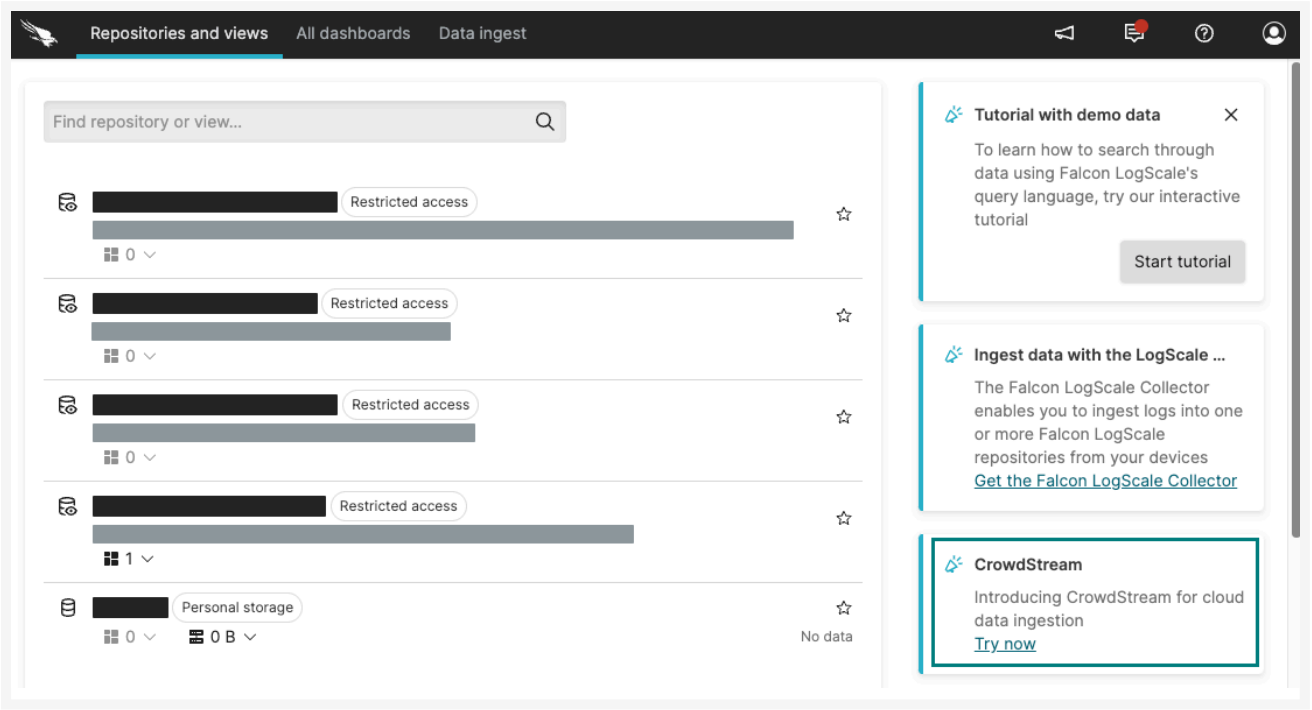
1. In the left menu, select **Next-Gen SIEM**
2. Then, under **Log management** select **CrowdStream**.



Navigating from Next-Gen SIEM to CrowdStream

To log into CrowdStream from LogScale interface:

1. Log into LogScale.
2. On the **CrowdStream** tile at right, select **Try now**.

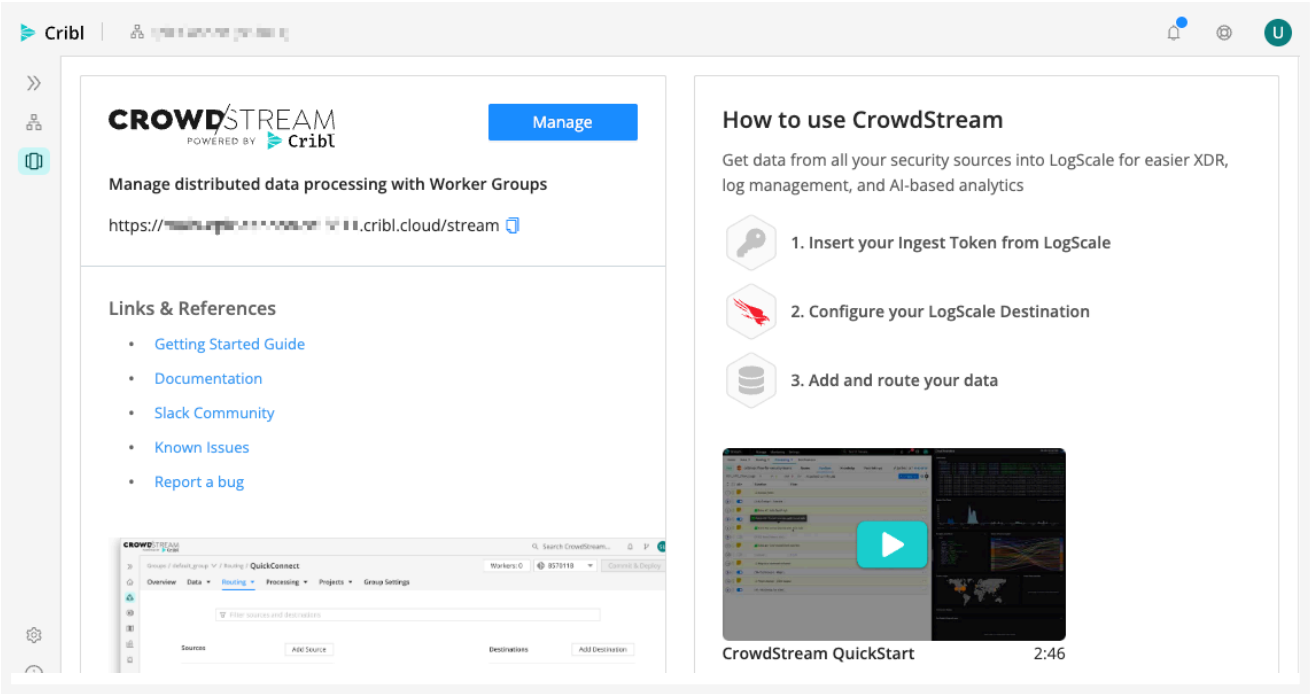


Navigating from LogScale to CrowdStream

3. On the resulting page, select **Log in**.

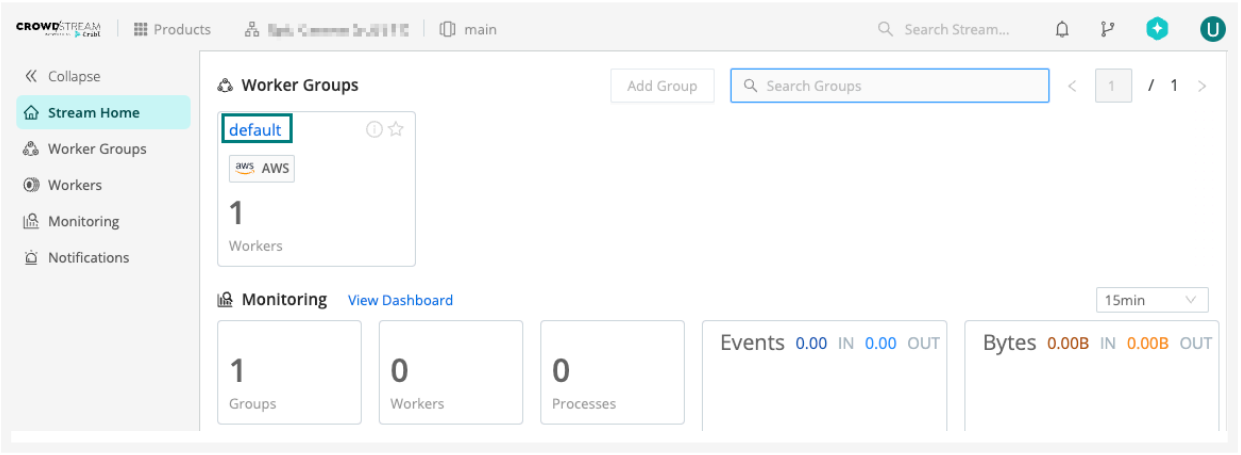
Regardless of your login method, you should now see the CrowdStream home page.

4. Select **Manage** on the **CrowdStream** tile.



Selecting the CrowdStream app

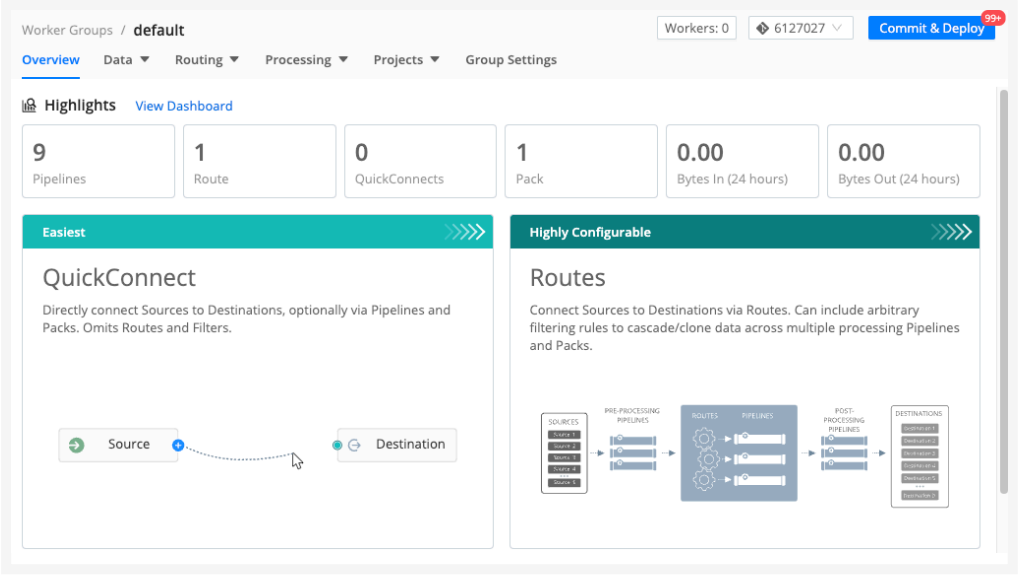
5. On CrowdStream's home page, under **Worker Groups**, select the **default** Group.
   (A Worker Group is a collection of processing instances that share the same configuration. For details, see Cribl Stream's Distributed Deployment topic.)



Selecting the **default** Group

If you see a **Worker Group is not provisioned** warning, select the **Provision Now** button to create Workers before proceeding.

6. To configure this Group, you can choose between the **QuickConnect** or **Routes** tiles. How to choose? See the next section.



Group **Overview** page

For More Information

- Cribl.Cloud Launch Guide

Simple or Complex Routing?

Your first choice is how simply or intricately you want to configure data flow from the first data Source you set up. CrowdStream offers two user interfaces:

- QuickConnect (the left tile shown above) provides a simple visual UI. You drag and drop connections from Sources on the left to Destinations on the right. Each connection can include as much or as little intermediate processing as you like. To remove a connection, just drag it off the Destination (you'll get a confirmation modal). The only practical limitation here is that data flow through every Source -> Destination connection must be parallel and independent.
- Data Routing (the right tile shown above) provides fine-grained control, via a more traditional UI that you configure across separate modals and pages. You build a Routing table with conditional processing. This can cascade and clone data across multiple processing Routs, according to the arbitrary filtering rules and sequence that you define.

So the trade-off between these two options is ease of use versus flexibility. These two options are separate, but not mutually exclusive: You can set up simultaneous data flows through QuickConnect and Data Routes.

You can also move Sources, and their connections to Destinations, between the two UIs. So one obvious option is to configure your first data flow in QuickConnect, and then move to the Routing UI as your familiarity and needs expand.

Here is a lightning-fast (one-minute) preview of **all** the configuration steps below, in QuickConnect:

Navigate Between User Interfaces

To work in QuickConnect, select the left tile shown above, or open the **Worker Groups** and select **Routing** > **QuickConnect**.

To work in the Routing UI, select the right tile shown above, or open the **Worker Groups** submenu and select **Routing** > **Data Routes** .

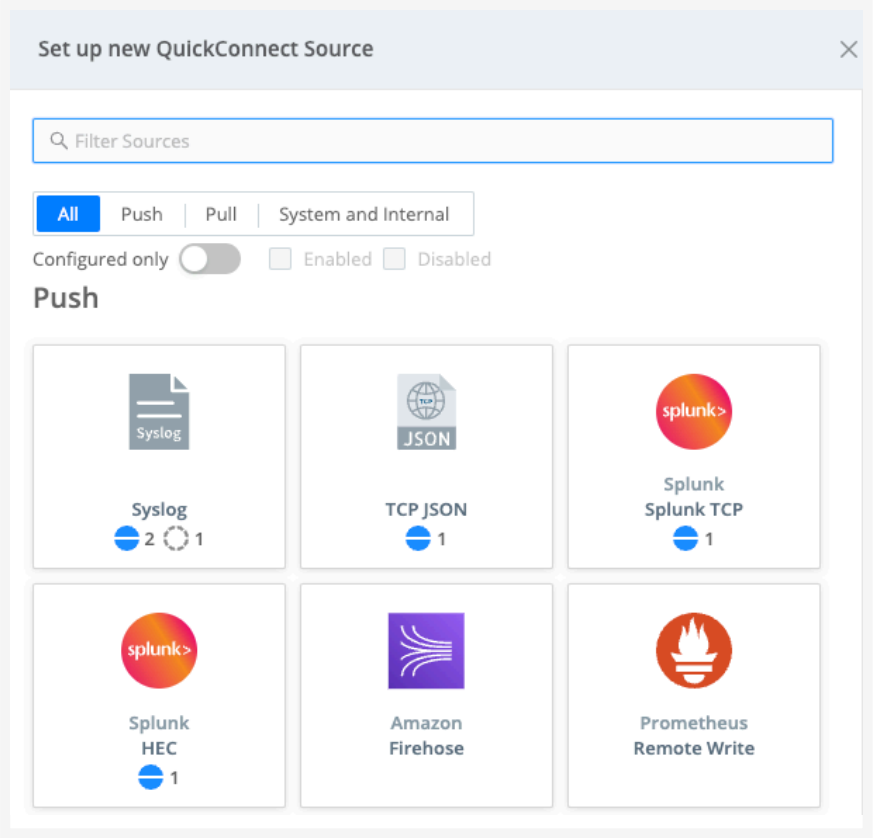
Configure a Source

A CrowdStream Source is a model of an upstream service sending data into the system. In this model, you configure authentication and/or other needed parameters.

You can configure multiple instances of a given Source type, each with different parameters. You can also attach processing that's specific to each instance. CrowdStream provides the Datagen Source type to generate sample inbound events for development and testing.

Using QuickConnect

In QuickConnect, select **Add Source** on the left (Sources) side. In the resulting left drawer, search for and select your desired Source type to begin configuring a new instance of that Source.



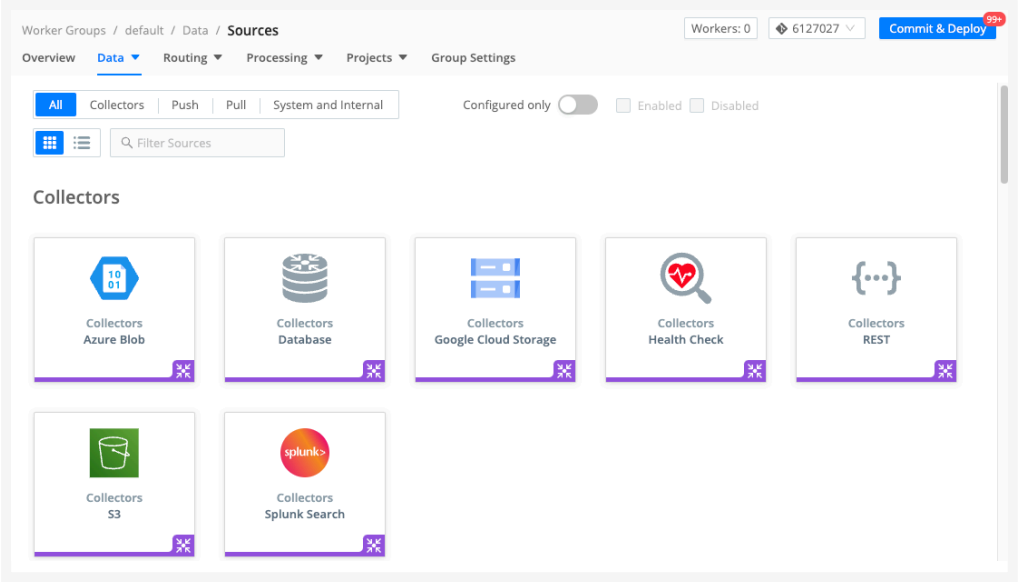
Selecting a Source through the QuickConnect UI

To configure (or reconfigure) a Source that's already present in QuickConnect, hover over its tile and select **Configure**. Note that for multiple Sources of the same type, tiles will display stacked until you hover over the stack.

Using Data Routes

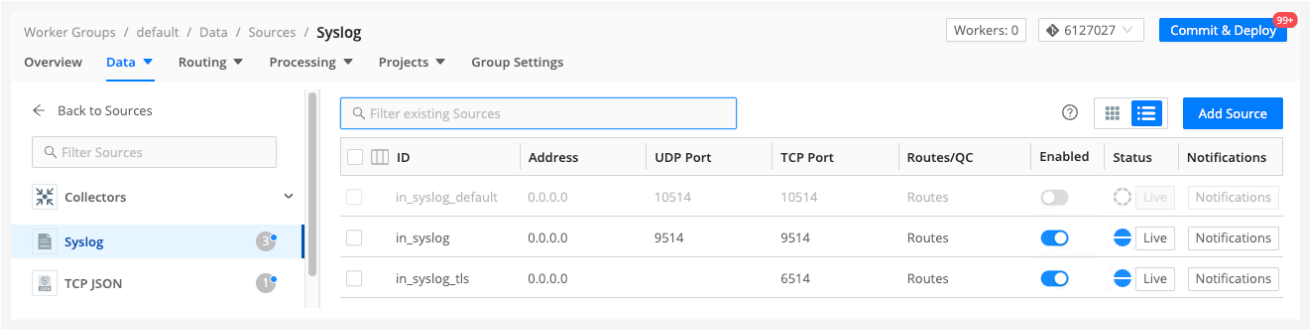
To use the Routing UI, on the **Worker Groups** submenu select **Data**, then **Sources**. On the resulting **Sources** page, search for your desired Source type, then select its tile to begin configuring a new instance of that Source.

(After you click through to a Source, the same set of peer Sources will be available in the left nav.)



Sources page

Select an existing Source's row here to access its configuration, or select **Add Source** to configure a new instance of that Source.



Manage Syslog Sources UI

To get data flowing, jump ahead to commit and deploy your new configuration.

For More Information

- Sources
- Using Datagens
- QuickConnect
- Routes

Configure a Destination

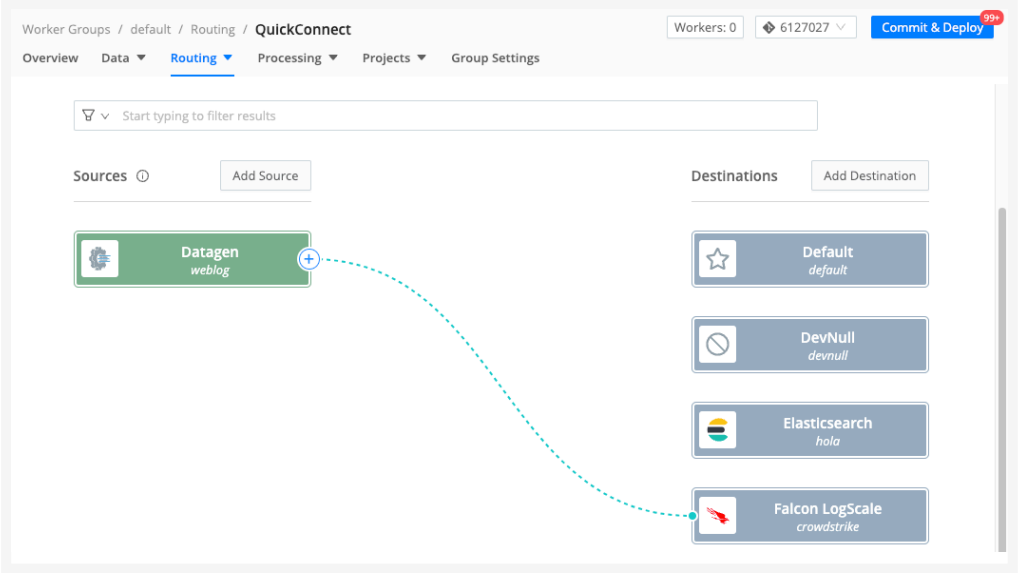
A CrowdStream Destination is a model of a downstream receiver of events, enabling you to configure authentication and/or other needed parameters.

As with Sources, you can configure multiple instances of a given Destination type, each with different parameters. You can attach processing that's specific to each instance. CrowdStream provides the following Destinations:

- CrowdStrike Falcon LogScale - Set up your LogScale endpoint, token, and other parameters here.
- Amazon S3 Compatible Stores - Route full-fidelity data to cost-effective storage for later replay and analysis.
- Data Lakes > Amazon S3 - Store full-fidelity data for later retrieval or analysis. Compatible with Cribl Search (a separate product).
- DevNull - This output simply consumes and discards events, to facilitate development and testing.

Using QuickConnect

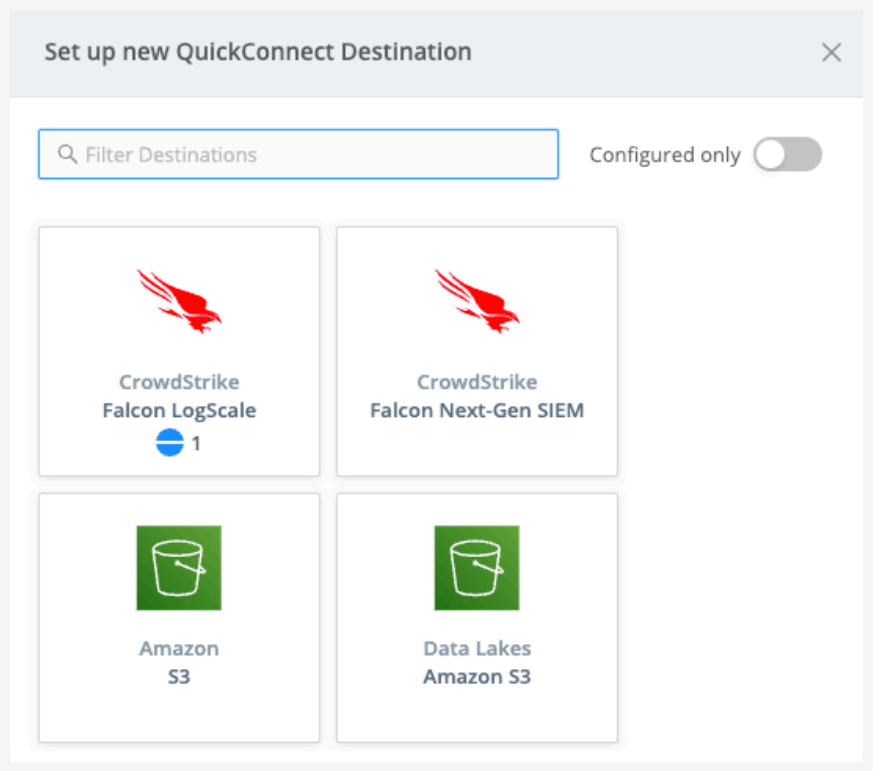
To connect a configured Source to a Destination tile already present on the right, just drag from the Source and drop the connector onto the Destination. A pop-up will prompt you to configure the connection via a Passthru, Pipeline or Pack (explained below).



Multiple Destinations added to QuickConnect

To configure (or reconfigure) the Destination, hover over its tile and select **Configure**. For multiple Destinations of the same type, tiles will display stacked until you hover over the stack.

To add a new Destination, select **Add Destination** on the right (Destinations) side. In the resulting right drawer, search for and select your desired Destination type to begin configuring a new instance of that Destination.

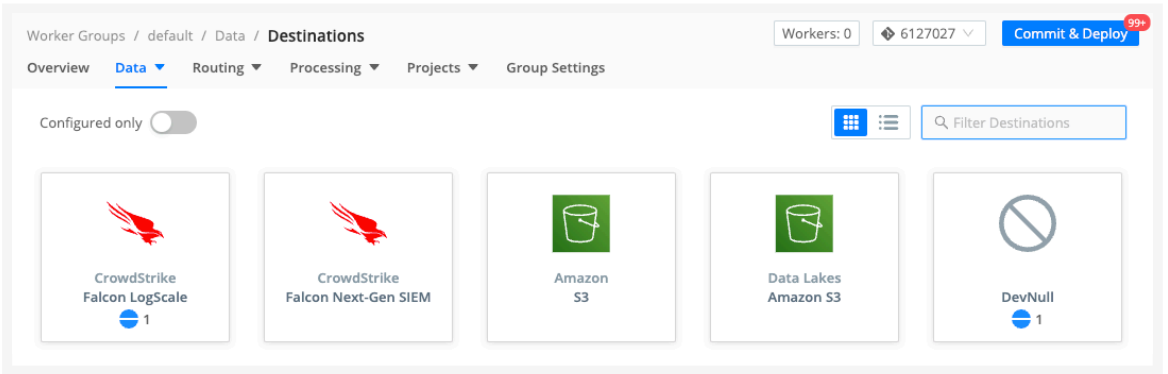


Selecting additional Destinations in QuickConnect

Using Data Routes

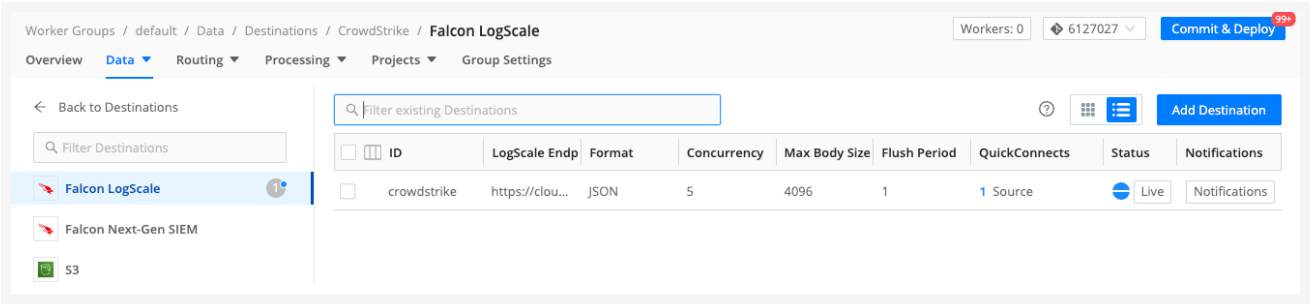
To use the Routing UI, on the **Worker Groups** submenu select **Data**, then **Destinations**. On the resulting **Destinations** page, search for your desired Destination type, then select its tile to begin configuring it.

(After you click through to a Destination, the same set of peer Destinations will be available in the left nav.)



Destinations page

Select an existing Destination's row here to access its configuration, or slect **Add Destination** to configure a new instance of that Destination.



Manage LogScale Destinations UI

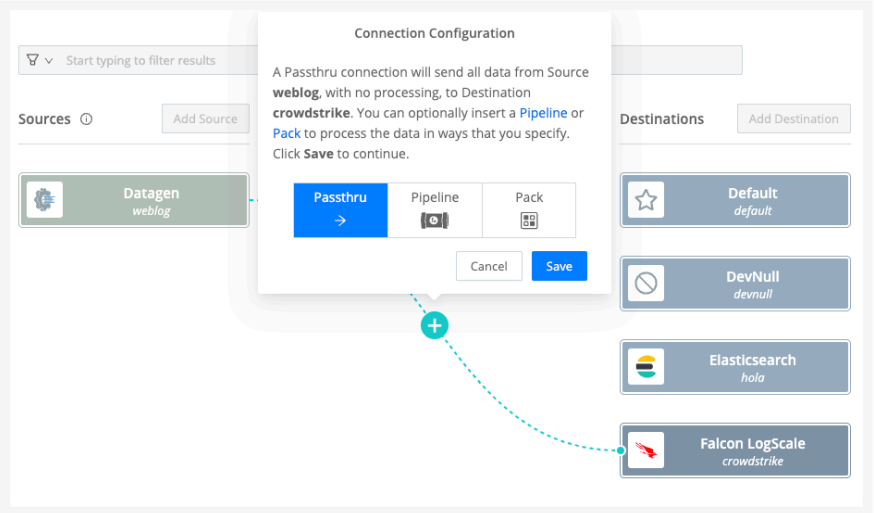
To get data flowing, jump ahead to commit and deploy your new configuration.

For More Information

- See the individual Destination topics linked at the head of this section.

Connect: Passthru, Pipeline, or Pack

In QuickConnect, as soon as you drag and drop a Source -> Destination connection, a pop-up dialog will prompt you to select a connection type: **Passthru**, **Pipeline**, or **Pack**. (In Data Routes, you'll make the same choices elsewhere.)



Connection-type selector in QuickConnect

What are these options?

- A **Passthru** establishes a simple direct connection between a Source -> Destination pair.
- A Pipeline hosts a stack of Functions to process your data, in the order you arrange them.
- A Pack is a microcosm of a whole CrowdStream configuration, designed to share configs between Worker Groups or deployments. A Pack can encapsulate its own Pipelines, Routes, Knowledge libraries, and

sample data (but not data Sources or Destinations). A Pack can snap in wherever you can use a Pipeline.

So let's unpack Pipelines, where you'll define most of your event processing.

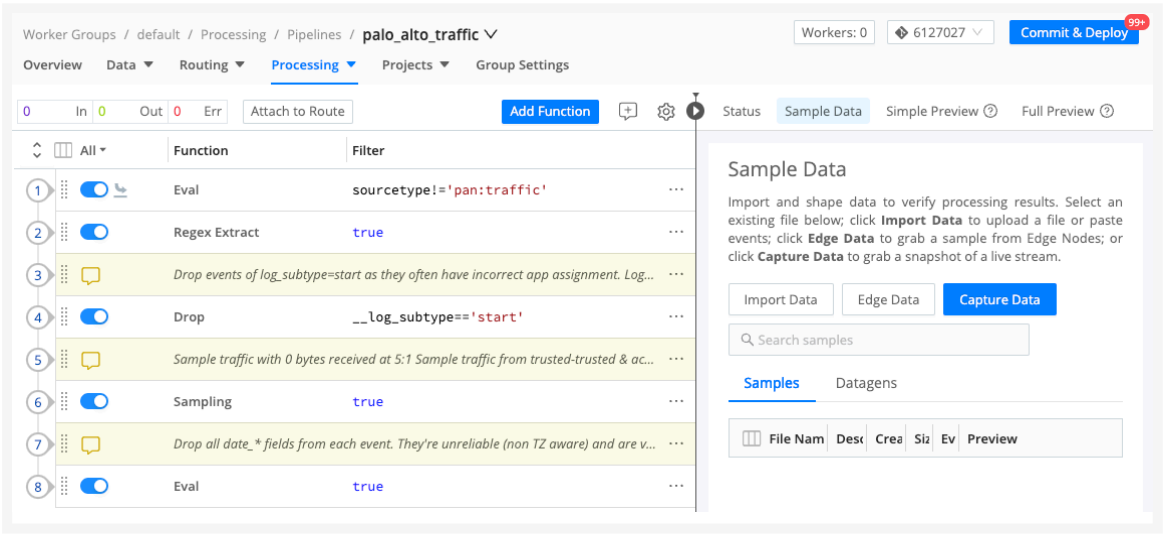
Pipeline Options

When you first create a new Pipeline, it's empty, making it equivalent to **Passthru**. CrowdStream provides a set of out-of-the-box Pipelines for specific purposes. As you can see below, one of these is a **Passthru** Pipeline. This similarly contains no Functions - making it the Data Routes UI's equivalent of QuickConnect's **Passthru** button.

For these reasons, it's easy to start with an empty Pipeline, and then add Functions as you need them.

A Pipeline, as you populate it, is a stack of processing Functions and optional comments. Use the accordions on the left to expand or collapse each Function. Use the toggles on the left to enable/disable each Function - preserving structure as you develop and debug.

Use the right **Sample Data** pane to preview how the Pipeline's current state transforms a set of events. (Controls here enable you to capture live data, save it to sample or datagen files, and reload those files.)



A Pipeline's Functions stack

Although you'll configure most data shaping in processing Pipelines between Sources and Destinations, you can also attach a pre-processing Pipeline to any Source instance, and a post-processing Pipeline to any Destination instance. Use these options when you need to "condition" the specific data type handled by a particular inbound or outbound integration.

Connect via QuickConnect

To select a connection type in QuickConnect, select the appropriate button in the pop-up dialog shown above. To accept the default **Passthru** option, select **Save**. Your Source and Destination will now be connected.

If you select **Pipeline**, you'll see an **Add Pipeline to Connection** modal. Here, you can choose an existing Pipeline, or select **Add Pipeline** to define a new one. Then select **Save** here to establish the connection.

Selecting **Pack** opens an **Add Pack to Connection** modal that works exactly the same way.

Whichever option you choose here, your connection between this Source and Destination is now complete.

To get data flowing, jump ahead to commit and deploy your new configuration.

Connect via Data Routes

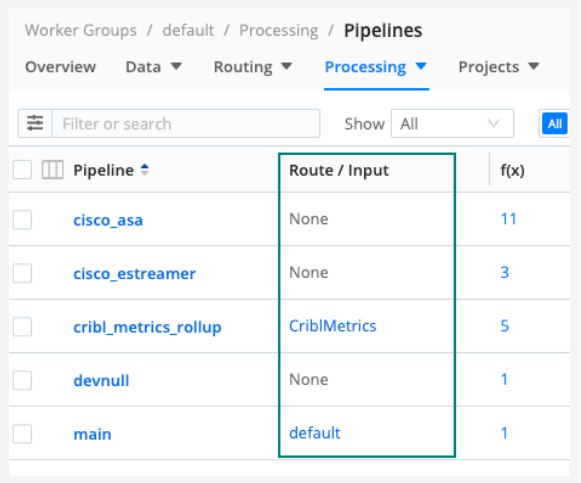
The Data Routes UI enables you to finely configure multiple, interdependent connections between Sources and Destinations. Events clone and cascade across a Routing table, based on the filtering rules and sequence you define.

In exchange for this flexibility, you sacrifice QuickConnect's one-stop, snap-together UI metaphor. After configuring your Sources and Destinations - which you've already done above - you need to connect them by separately setting up at least one Pipeline and one Route.

Pipelines Access

On the **Worker Groups** submenu, select **Processing**, then **Pipelines**. In addition to the minimal **Passthru** Pipeline identified above, other simple building blocks here include **main** (which initially contains only a simple Eval Function) and **devnull** (which contains a single Function that just drops events).

Out-of-the-box Pipelines are already connected to Routes. On the **Pipelines** page, the **Route/Input** columns displays the connected Route, as shown here. But as our final connection step, let's next look at how to connect up everything (Source, Pipeline, Destination) in a Route.



Checking Pipelines Route connections

For More Information

- Refine Data with Functions walks you through configuring CrowdStream's built-in Pipeline Functions to perform common processing tasks like parsing, masking, adding, deleting, and

renaming event fields.

Configure a Route

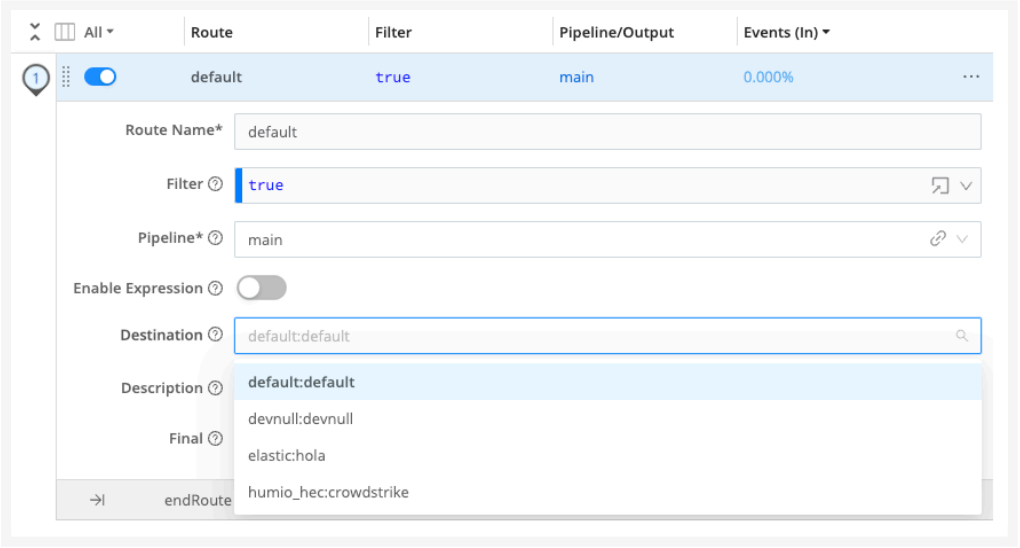
In the Data Routes UI, a Route is where you complete a data-flow connection. Each Route connects exactly one Pipeline with exactly one Destination.

This combination can consume data from one, multiple, or all the Sources you've configured on the Data Routes side. (QuickConnect Sources are separate, but you can open those Sources' configs to move them to the Data Routes UI.) This is governed by the filtering we'll see below.

To open the Routing table, on the **Worker Groups** submenu, select **Routing**, then **Data Routes**.

CrowdStream ships with one starter Route named **default**. It's configured as follows:

- The **Filter** expression is set to **true**. This simple, default expression makes data from **all** active Sources available to this Route. (See finer filtering options below.)
- The **Pipeline** drop-down is where you set the 1:1 Pipeline:Route connection. This starter Route's Pipeline defaults to **main**, but you can select any one other configured Pipeline.
- The **Destination** drop-down defines the single Destination for this Pipeline:Route connection. If this isn't already set to your LogScale Destination, you'll most likely want to select that here. (Note that each option displayed on this list corresponds to the unique, arbitrary **Output ID** set on one **instance** of a Destination type. So some names might initially be unfamiliar.)



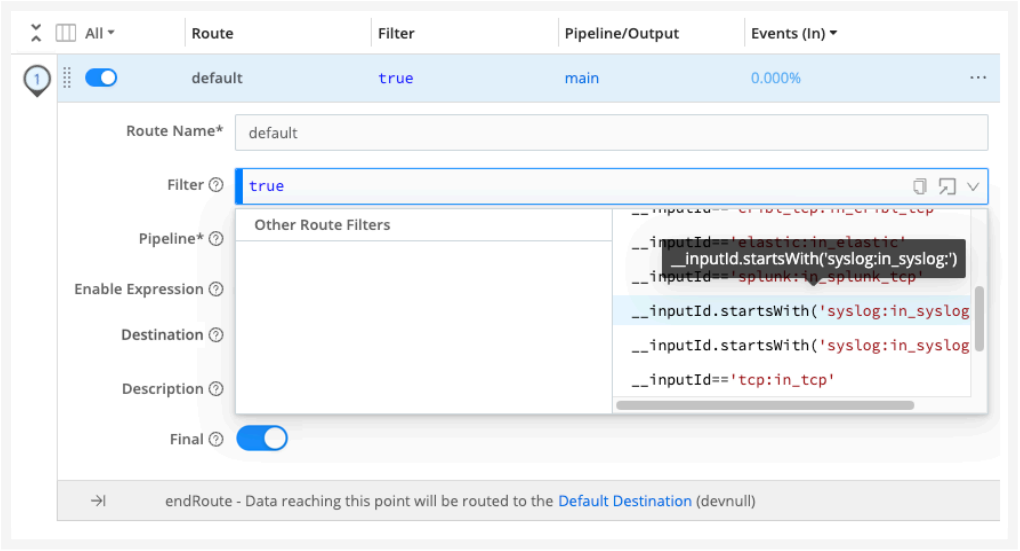
Changing a Route's Destination

To get data flowing, jump ahead to commit and deploy your new configuration.

Filter a Route's Input Data

When you want a Route to ingest data from only one Source, or a subset of Sources, you expand its **Filter** expression beyond the default **true** catch-all.

Filter is itself a drop-down. Open it to select among typeahead suggestions for several common inbound data types.



Tuning a Route's Filter expression

You can select one of these expressions and then modify it, using the same `__inputId` syntax. In the example expression below, note that the identifier after the `:` is - as with the **Destination** options we saw above - the arbitrary name of one Syslog Source instance:

```
__inputId.startsWith('syslog:in_syslog:')
```

For More Information

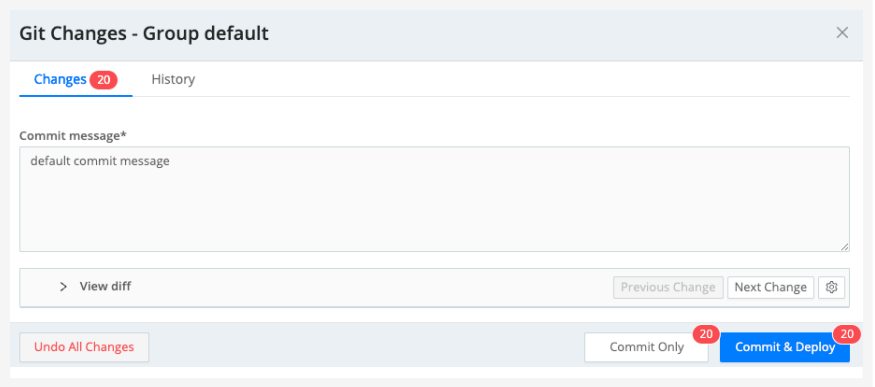
- Build Custom Logic to Route and Process Your Data: Filter Expressions

Commit/Deploy Config Changes

To get data flowing as you expect, you'll always need to use CrowdStream's built-in Git version-control integration to commit your configuration changes and deploy them to your event-processing Worker instances. After any significant configuration change, you'll be prompted with Group-level **Commit & Deploy** button at the upper right

Depending on your version control settings, **Commit** and **Deploy** might appear as separate buttons.

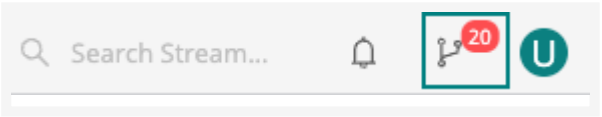
Selecting either flavor of the **Commit** button will display a confirmation modal like this. You can either accept the default commit message, or enter a specific message. Then be sure to select the appropriate buttons to both commit **and** deploy your changes.



Confirming a commit

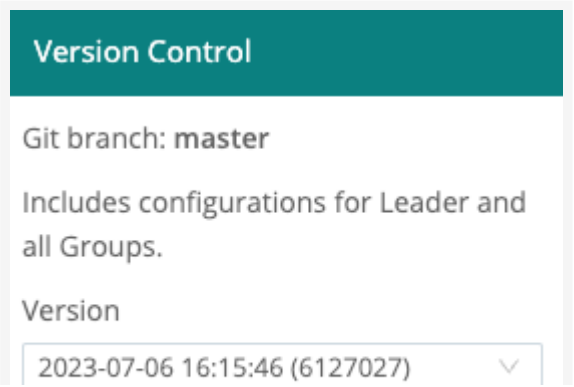
Data should now start flowing through the Workers. (You can check this by selecting the top nav's **Monitoring** link. This takes you to read-only dashboards, so remember to select **Manage** when you want to return to configuration options.)

In return for clearing all the red-dot prompts, your reward will be one more red dot, on the top nav's CrowdStream-wide Version Control button. This is because CrowdStream sees the preceding deploy to Workers as a change to record (commit) on the **Leader's** configuration.



Leader commit prompt

This top-level commit isn't necessary to enable data flow, but it backs up and safeguards your overall configuration. Select the button to display one final confirmation prompt:



13. Third-Party Credits

Various components in Cribl Stream are built and enhanced with software under free or open source licenses. We thank those projects' contributors!

Package name	Version	License	Copyright