Cribl LogStream Documentation Manual

Version: v3.0.1

Generated: 2021-05-25 21:29:41

INTRODUCTION 8

About Cribl LogStream 8

Basic Concepts 10

Getting Started Guide 15

DEPLOYMENT 34

Deployment Types 34

Single-Instance Deployment 36

Distributed Deployment 42

Splunk App Deployment * 59

Bootstrap Workers from Leader 63

Kubernetes/Helm Deployment 67

K8s Leader Deployment 70

K8s Worker Deployment 86

Docker Deployment 97

Sizing and Scaling 99

Config Files 103

cribl.yml 105

inputs.yml 107

outputs.yml 108

licenses.yml 110

regexes.yml 111

breakers.yml 112

mappings.yml 113

instance.yml 114

Licensing 115

Access Management 122

Authentication 123

Local Users 131

Roles 133

Version Control 141

Persistent Queues 153

Securing 156

Monitoring 164

Internal Metrics 173

Upgrading 179

Uninstalling 187

WORKING WITH DATA 188
Event Model 188
Event Processing Order 190
Routes 192
Pipelines 199
Data Onboarding 205
Functions 209
Auto Timestamp 215
Aggregations 220
CEF Serializer 227
Clone 229
Comment 230
DNS Lookup 232
Drop 236
Dynamic Sampling 237
Eval 241
Flatten 244
GeoIP 246
Grok 249
JSON Unroll 251
Lookup 253
Mask 257
Numerify 263
Parser 265
Publish Metrics 273
Regex Extract 279
Redis 283
Regex Filter 286
Rename 287
Rollup Metrics 290
Sampling 292
Serialize 294
Suppress 297
Tee 300
Trim Timestamp 303
Unroll 305
XML Unroll 307
Prometheus Publisher (Deprecated) 310

Reverse DNS (deprecated) 312
Sources 314
Syslog 318
Splunk TCP 322
Splunk HEC 326
Amazon Kinesis Firehose 331
Amazon Kinesis 334
Amazon SQS 338
Amazon S3 343
Google Cloud Pub/Sub 350
Azure Event Hubs 353
Azure Blob Storage 356
Office 365 Services 366
Office 365 Activity 369
Office 365 Message Trace 372
TCP JSON 375
TCP (RAW) 379
HTTP/S (Bulk API) 383
Raw HTTP/S 388
Elasticsearch API 391
Kafka 395
Metrics 399
SNMP Trap 402
Prometheus 405
Datagens 408
AppScope 410
Cribl Internal 413
Collectors 417
Filesystem/NFS 421
Azure Blob Storage 425
Google Cloud Storage 429
S3 433
Script 438
REST / API Endpoint 442
Scheduling and Running 451
Destinations 457
Output Router 461
Splunk Single Instance 463

- Splunk Load Balanced 467
- Splunk HEC 475
- Amazon S3 Compatible Stores 479
- Amazon Kinesis Streams 484
- Amazon CloudWatch Logs 487
- Amazon SQS 490
- Azure Blob Storage 494
- Azure Monitor Logs 497
- Azure Event Hubs 500
- Google Cloud Storage 503
- Google Cloud Pub/Sub 507
- StatsD 510
- StatsD Extended 513
- Graphite 516
- TCP JSON 519
- Syslog 523
- Filesystem/NFS 527
- Kafka 530
- Elasticsearch 535
- Honeycomb 539
- New Relic 542
- SNMP Trap 545
- InfluxDB 547
- MinIO 550
- Prometheus 556
- Wavefront 560
- SignalFx 563
- Sumo Logic 566
- Datadog 569
- Webhook 573
- DevNull 577
- Default 578

Packs 579

Data Preview 591

Securing Data 600

- Encryption 601
- Decryption 606

Scripts 609

Using Datagens 611
CLI Reference 616
EXPRESSION REFERENCE 624
Introduction to Expression Syntax 624
Cribl Expressions 627
KNOWLEDGE 638
Regex Library 638
Grok Patterns Library 641
Event Breakers 643
Lookups Library 653
Parsers Library 657
Schema Library 659
Global Variables Library 661
TECHNIQUES & TIPS 663
Tips and Tricks 663
Ingest-time Fields 667
Ingest-time Lookups 670
Sampling 674
Access Logs: Apache, ELB, CDN, S3, etc. 676
Firewall Logs: VPC Flow Logs, Cisco ASA, Etc. 679
Masking and Obfuscation 682
Managing Large Lookups 686
Lookups as Filters for Masks 690
Lookups and Regex Magic 694
Regex Filtering 703
Encrypting Sensitive Data 705
Syslog Data Reduction 711
Splunk to Elasticsearch 719
Reducing Windows XML Events 727
Using REST/API Collectors 739
System Proxy Configuration 758
VIDEOS 763
Videos 763
TROUBLESHOOTING 773
Diagnosing Issues 773
Working with Cribl Support 775
Known Issues 779

Common Errors and Warnings 792
Git Push Errors 801
Git Remote Repos & Trusted CAs 803
THIRD-PARTY SOFTWARE 806
Credits 806

INTRODUCTION

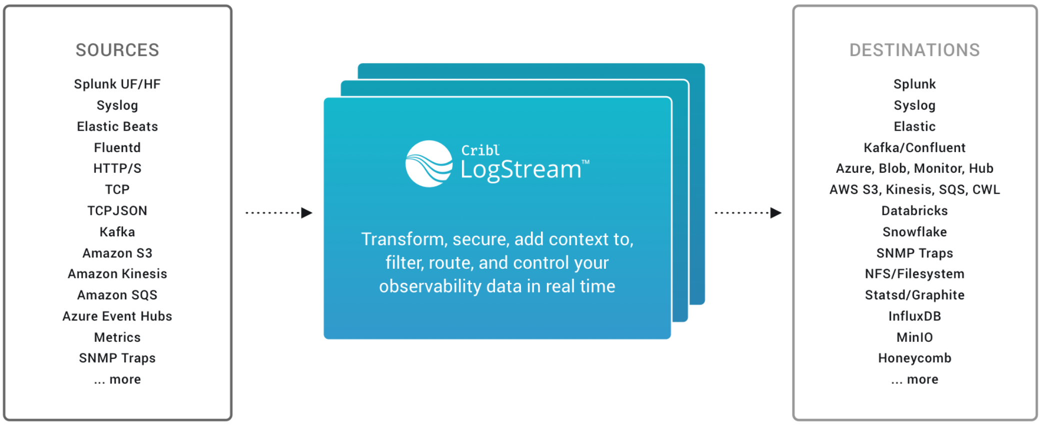
About Cribl LogStream

Getting started with Cribl LogStream

What Is Cribl LogStream?

Cribl LogStream 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, LogStream, destinations

Cribl LogStream 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.

Who Is Cribl LogStream For?

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

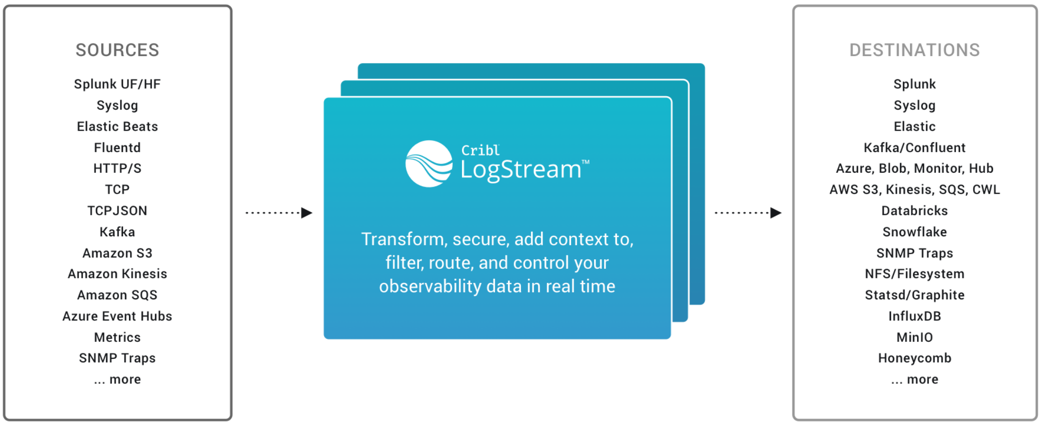
What's Next

› Basic Concepts

Basic Concepts

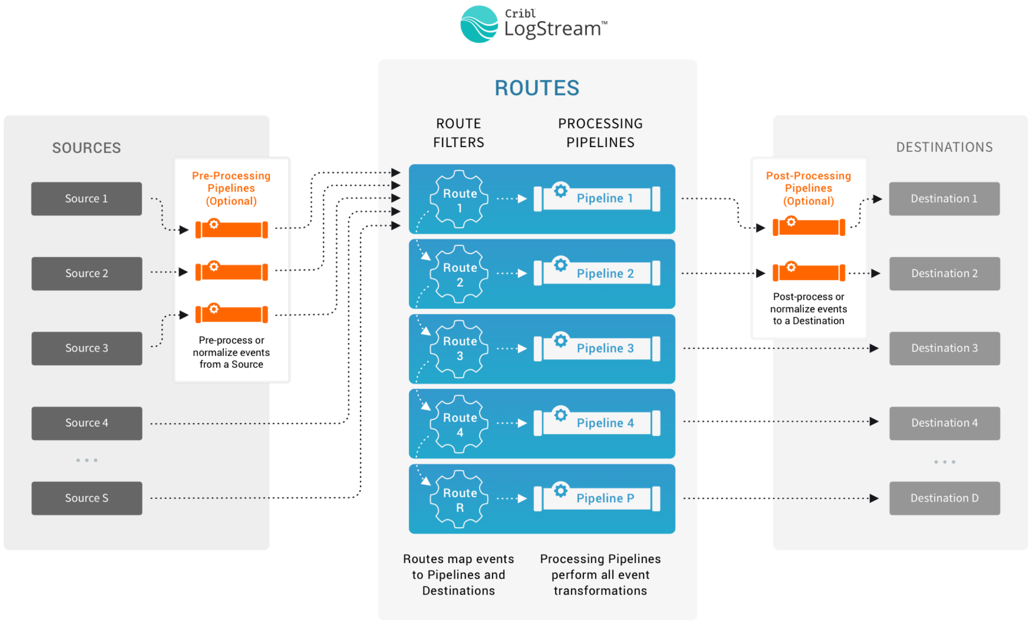
Notable features and concepts to get a fundamental understanding of Cribl LogStream

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



Sources, LogStream, destinations

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


Cribl LogStream™
ROUTES
ROUTE FILTERS
PROCESSING PIPELINES
SOURCES
Source 1
Source 2
Source 3
Source 4
Source S
Pre-Processing Pipelines (Optional)
Pre-process or normalize events from a Source
Route 1
Route 2
Route 3
Route 4
Route R
Pipeline 1
Pipeline 2
Pipeline 3
Pipeline 4
Pipeline P
Routes map events to Pipelines and Destinations
Processing Pipelines perform all event transformations
DESTINATIONS
Post-Processing Pipelines (Optional)
Post-process or normalize events to a Destination
Destination 1
Destination 2
Destination 3
Destination 4
Destination D

Routes, Pipelines, Functions

Routes

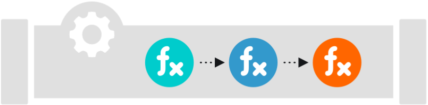
Routes evaluate incoming events against **filter** expressions to find the appropriate Pipeline to send them to. Routes are **evaluated in order**. Each Route can be associated **with only one** Pipeline and one output (configured as a LogStream **Destination**).

By default, each Route is created with its `Final` flag set to `Yes`. With this setting, a Route-Pipeline-Destination set will consume events that match its filter, and that's that.

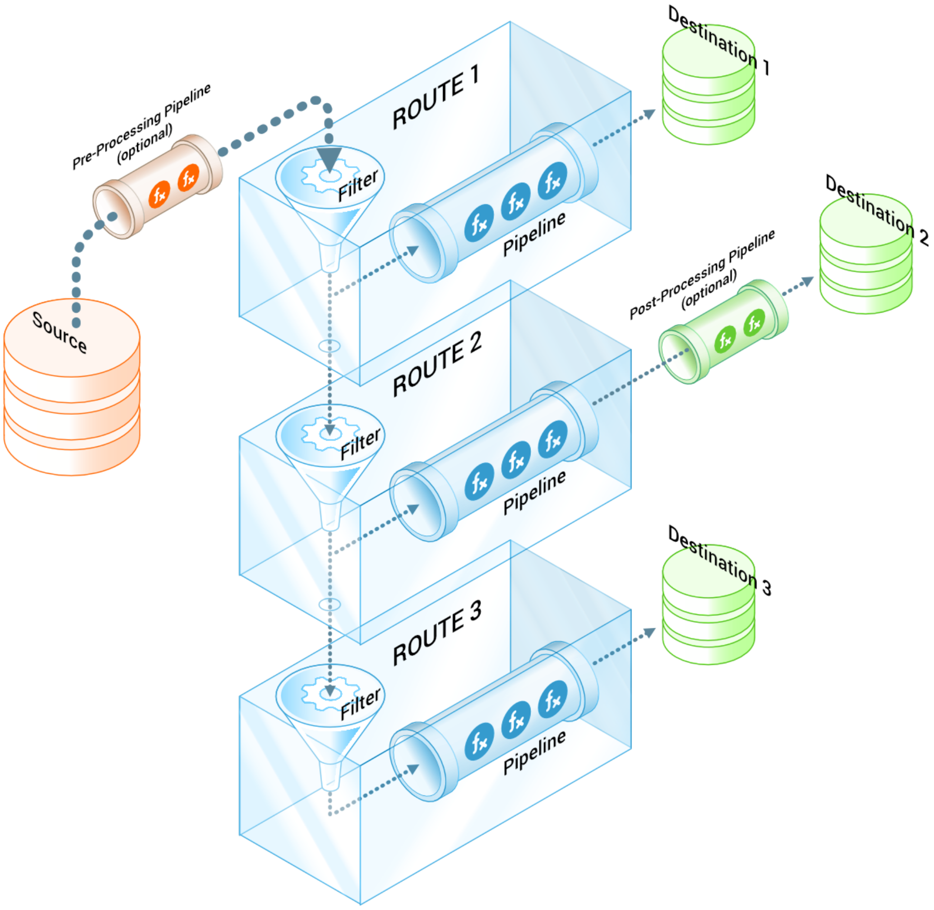
However, if you disable the Route's `Final` flag, one or more event **clones** will be sent down the associated Pipeline, while the original event continues down LogStream's Routing table to be evaluated against other configured Routes. This is very useful in cases where the same set of events needs 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 LogStream Sources, and attach **post-processing Pipelines** to LogStream Destinations. All Pipelines are configured through the same UI – these three designations are determined only by a Pipeline's placement in LogStream's data flow.

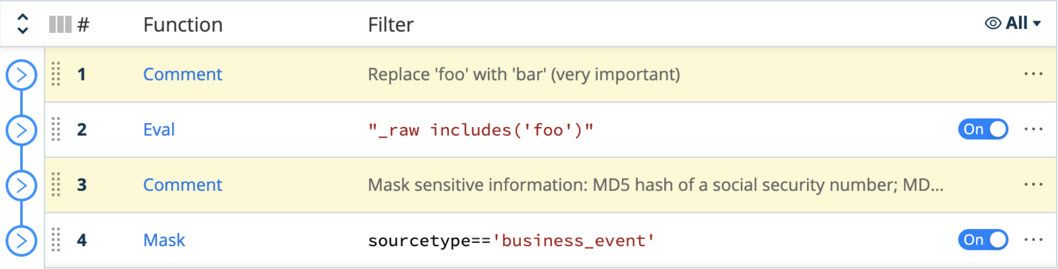


Pipelines categorized by position

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`.



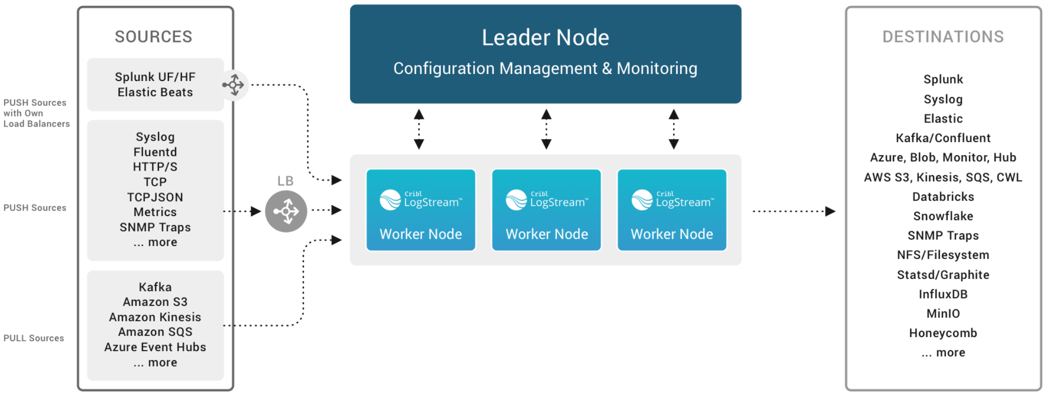
#	Function	Filter	All
1	Comment	Replace 'foo' with 'bar' (very important)	···
2	Eval	`"_raw includes('foo')"`	On ···
3	Comment	Mask sensitive information: MD5 hash of a social security number; MD...	···
4	Mask	`sourcetype=='business_event'`	On ···

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.

A Scalable Model

You can scale LogStream up to meet enterprise needs in a distributed deployment. Here, multiple LogStream 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

What's Next

› Getting Started Guide

› Deployment Types

Getting Started Guide

This guide walks you through planning, installing, and configuring a single-instance deployment of Cribl LogStream. You'll capture some realistic sample log data, and then use LogStream's built-in Functions to redact, parse, refine, and shrink the data.

By the end of this guide, you'll have assembled all of LogStream's basic building blocks: a Source, Route, Pipeline, several Functions, and a Destination. You can complete this tutorial using LogStream's included sample data, without connections to – or licenses on – any inbound or outbound services.

Assuming a cold start (from initial LogStream download and installation), this guide might take about an hour. But you can work through it in chunks, and LogStream will persist your work between sessions.

> If you've already downloaded, installed, and launched LogStream, skip ahead to Add a Source.

Requirements for this Tutorial

The minimum requirements for running this tutorial are the same as for a LogStream production single-instance deployment.

OS

- Linux 64-bit kernel >= 3.10 and glibc >= 2.17
- Examples: Ubuntu 16.04, Debian 9, RHEL 7, CentOS 7, SUSE Linux Enterprise Server 12+, Amazon Linux 2014.03+

System

- +4 physical cores = +8 vCPUs; +8GB RAM – all beyond your basic OS/VM requirements
- 5GB free disk space (more if persistent queuing is enabled)

> ℹ We assume that 1 physical core is equivalent to 2 virtual/hyperthreaded CPUs (vCPUs). For details, see Recommended AWS, Azure, and GCP Instance Types.

Browser Support

- Firefox 65+, Chrome 70+, Safari 12+, Microsoft Edge

Network Ports

By default, LogStream listens on the following ports:

Component	Default Port
UI default	`9000`
HTTP Inbound, default	`10080`
User options	+ Other data ports as required.

You can override these defaults as needed.

Plan for Production

For higher processing volumes, users typically enable LogStream's Distributed Deployment option. While beyond the scope of this tutorial, that option has a few additional requirements, which we list here for planning purposes:

- Port `9000` or `4200` must be available on the Leader Node for Workers' communications.
- Git (1.8.3.1 or higher) must be installed on the Leader Node, to manage configuration changes.

See Sizing and Scaling for further details about configuring LogStream to handle large data streams.

Download and Install LogStream

Download the latest version of LogStream at https://cribl.io/download/.

Un-tar the resulting `.tgz` file in a directory of your choice (e.g., `/opt/`). Here's general syntax, then a specific example:

```
tar xvzf cribl-<version>-<build>-<arch>.tgz
tar xvzf cribl-2.3.1-1d4e05c5-linux-x64.tgz
```

You'll now have LogStream installed in a `cribl` subdirectory, by default: `/opt/cribl/`. We'll refer to this `cribl` subdirectory throughout this documentation as `$CRIBL_HOME`.

Run LogStream

In your terminal, switch to the `$CRIBL_HOME/bin` directory (e.g,: `/opt/cribl/bin`). Here, you can start, top, and verify the LogStream server using these basic `./cribl` CLI commands:

- **Start**: `./cribl start`
- **Stop**: `./cribl stop`
- **Get status**: `./cribl status`

> For other available commands, see CLI Reference.

Next, in your browser, open `http://<hostname>:9000` (e.g., `http://localhost:9000`) and log in with default credentials (`admin`, `admin`).

Register your copy of LogStream if desired.

After registering, you'll be prompted to change the default password.

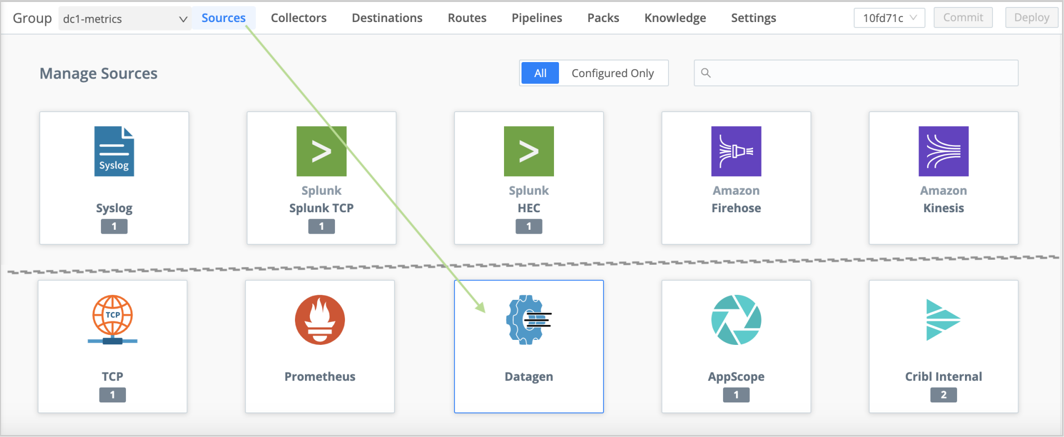
You are now ready to configure a working LogStream installation – with a Source, Destination, Pipeline, and Route – and to assemble several built-in Functions to refine sample log data.

Get Data Flowing

Add a Source

Each LogStream Source represents a data input. Options include Splunk, Elastic Beats, Kinesis, Kafka, syslog, HTTP, TCP JSON, and others.

For this tutorial, we'll enable a LogStream built-in datagen (i.e., data generator) that generates a stream of realistic sample log data.



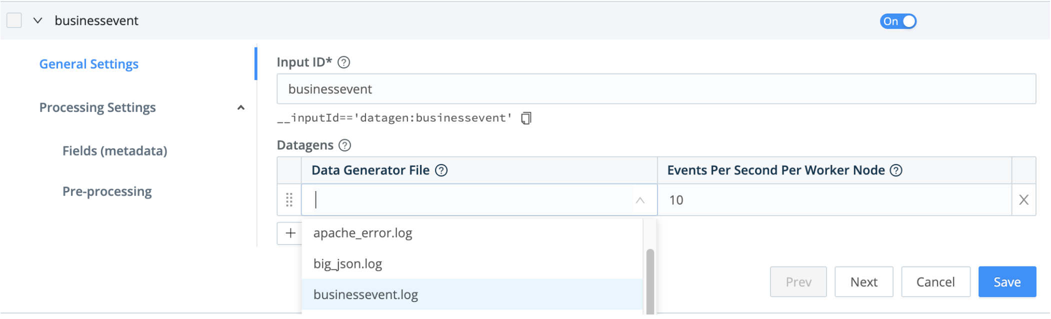
Adding a datagen Source

1. From LogStream's top menu, select **Data Sources**.
2. From the **Data Sources** page's tiles or left menu, select **Datagens**.

   (You can use the search box to jump to the **Datagens** tile.)
3. Click **+ Add New** to open the **New Datagen source** pane.
4. In the **Input ID** field, name this Source `businessevent`.
5. In the **Data Generator File** drop-down, select `businessevent.log`.

   This generates...log events for a business scenario. We'll look at their structure shortly, in Capture and Filter Sample Data.
6. Click **Save**.

The **On** slider in the **Enabled** column indicates that your datagen Source has started generating sample data.



Input ID*: businessevent

```
__inputId=='datagen:businessevent'
```

Events Per Second Per Worker Node: 10

Configuring a datagen Source

Add a Destination

Each LogStream Destination represents a data output. Options include Splunk, Kafka, Kinesis, InfluxDB, Snowflake, Databricks, TCP JSON, and others.

For this tutorial, we'll use LogStream's built-in **DevNull** Destination. This simply discards events – not very exciting! But it simulates a real output, so it provides a configuration-free quick start for testing LogStream setups. It's ideal for our purposes.

To verify that **DevNull** is enabled, let's walk through setting up a Destination, then setting it up as LogStream's default output:

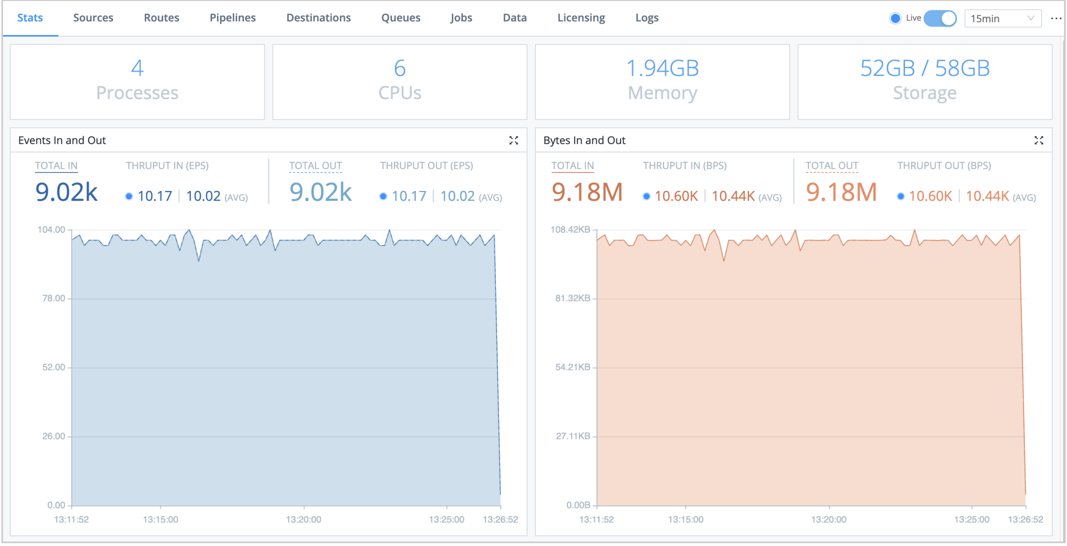
1. From LogStream's top menu, select **Destinations**.
2. Select **DevNull** from the **Data Destinations** page's tiles or left menu.

   (You can use the search box to jump to the **DevNull** tile.)
3. On the resulting **devnull** row, look for the **Live** indicator under **Enabled**. This confirms that the **DevNull** Destination is ready to accept events.
4. From the **Data Destinations** page's left nav, select the **Default** Destination at the top.
5. On the resulting **Manage Default Destination** page, verify that the **Default Output ID** drop-down points to the **devnull** Destination we just examined.

We've now set up data flow on both sides. Is data flowing? Let's check.

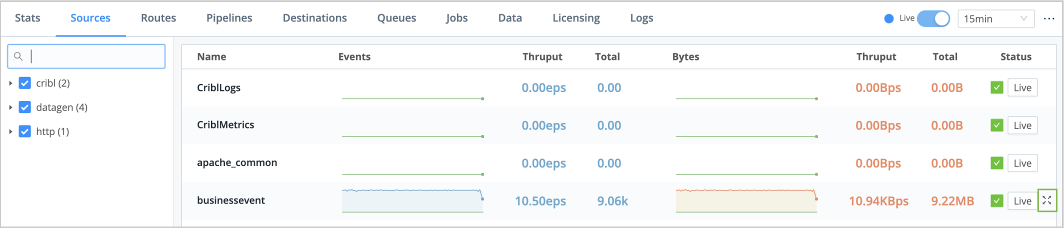
Monitor Data Throughput

From the top menu, select **Monitoring**. (On very narrow displays, you might need to select it from the ••• overflow menu.) This opens a summary dashboard, where you should see a steady flow of data in and out of LogStream. The left graph shows events in/out. The right graph shows bytes in/out.



Monitoring dashboard

Monitoring displays data from the preceding 24 hours. You can use the **Monitoring** submenu to open detailed displays of LogStream components, collection jobs and tasks, and LogStream's own internallogs. Click **Sources** on the lower submenu to switch to this view:

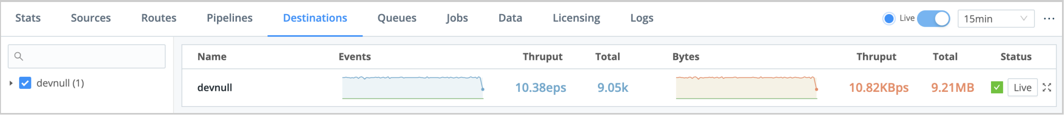


Name	Events	Thruput	Total	Bytes	Thruput	Total	Status
CriblLogs		0.00eps	0.00		0.00Bps	0.00B	☑ Live
CriblMetrics		0.00eps	0.00		0.00Bps	0.00B	☑ Live
apache_common		0.00eps	0.00		0.00Bps	0.00B	☑ Live
businessevent		10.50eps	9.06k		10.94KBps	9.22MB	☑ Live

Monitoring Sources

This is a compact display of each Source's inbound events and bytes as a sparkline. You can click each Source's Expand button (highlighted at right) to zoom up detailed graphs.

Click **Destinations** on the lower submenu. This displays a similar sparklines view, where you can confirm data flow out to the `devnull` Destination:



Name	Events	Thruput	Total	Bytes	Thruput	Total	Status
devnull		10.38eps	9.05k		10.82KBps	9.21MB	✓ Live

Monitoring Destinations

With confidence that we've got data flowing, let's send it through a LogStream Pipeline, where we can add Functions to refine the raw data.

Create a Pipeline

A Pipeline is a stack of LogStream Functions that process data. Pipelines are central to refining your data, and also provide a central LogStream workspace – so let's get one going.

1. From the top menu, select **Pipelines**.

   You now have a two-pane view, with ~~business on the left and party on the right~~ a **Pipelines** list on the left and **Sample Data** controls on the right. (We'll capture some sample data momentarily.)

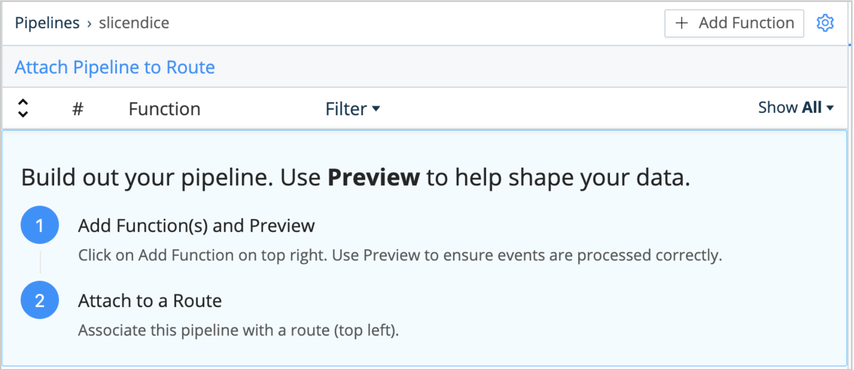
2. At the **Pipelines** pane's upper right, click **+ Pipeline**, then select **Create Pipeline**.

3. In the new Pipeline's **ID** field, enter a unique identifier. (For this tutorial, you might use `slicendice` .)

4. Optionally, enter a **Description** of this Pipeline's purpose.

5. Click **Save**.

Your empty Pipeline now prompts you to preview data, add Functions, and attach a Route. So let's capture some data to preview.



Build out your pipeline. Use **Preview** to help shape your data.

1 Add Function(s) and Preview

Click on Add Function on top right. Use Preview to ensure events are processed correctly.

2 Attach to a Route

Associate this pipeline with a route (top left).

Pipeline prompt to add Functions

Capture and Filter Sample Data

The right **Sample Data** pane provides multiple tools for grabbing data from multiple places (inbound streams, copy/paste, and uploaded files); for previewing and testing data transformations as you build them; and for saving and reloading sample files.

Since we've already got live (simulated) data flowing in from the datagen Source we built, let's grab some of that data.

Capture New Data

1. In the right pane, click **Capture New**.
2. Click **Capture**, then accept the drop-down's defaults – click **Start**.
3. When the modal finishes populating with events, click **Save as Sample File**.
4. In the **SAMPLE FILE SETTINGS** pop-up, change the generated **File Name** to a name you'll recognize, like `be_raw.log` .
5. Click **Save**. This saves to the **File Name** you entered above, and closes the modal. You're now previewing the captured events in the right pane. (Note that this pane's **Preview Simple** tab now has focus.)
6. Click **Show more** to expand one or more events.

By skimming the key-value pairs within the data's `_raw` fields, you'll notice the scenario underlying this preview data (provided by the `businessevents.log` datagen): these are business logs from a mobile-phone provider.

To set up our next step, find at least one `marketState` K=V pair. Having captured and examined this raw data, let's use this K=V pair to crack open LogStream's most basic data-transformation tool, Filtering.

Filter Data and Manage Sample Files

1. Click the right pane's **Sample Data** tab.
2. Again click **Capture New**.
3. In the **Capture Sample Data** modal, replace the **Filter Expression** field's default `true` value with this simple regex:

```
_raw.match(/marketState=TX/)
```

We're going to Texas! If you type this in, rather than pasting it, notice how LogStream provides typeahead assist to complete a well-formed JavaScript expression.

You can also click the Expand button at the **Filter Expression** field's right edge to open a modal to validate your expression. The adjacent drop-down enables you to restore previously used expressions

4. Click **Capture**, then **Start**.

   Using the **Capture** drop-down's default limits of 10 seconds and 10 events, you'll notice that with this filter applied, it takes much longer for LogStream to capture 10 matching events.

5. Click **Cancel** (and confirm your selection) to discard this filtered data and close the modal.

6. On the right pane's **Sample Data** tab, click **Simple** beside `be_raw.log` .

This restores our preview of our original, unfiltered capture. We're ready to transform this sample data in more interesting ways, by building out our Pipeline's Functions.

Refine Data with Functions

Functions are pieces of JavaScript code that LogStream invokes on each event that passes through them. By default, this means all events – each Function has a **Filter** field whose value defaults to `true` . As we just saw with data capture, you can replace this value with an expression that scopes the Function down to particular matching events.

In this Pipeline, we'll use some of LogStream's core Functions to:

- Redact (mask) sensitive data
- Extract (parse) the `_raw` field's key-value pairs as separate fields.
- Add a new field.
- Delete the original `_raw` field, now that we've extracted its contents.

- Rename a field for better legibility..

Mask: Redact Sensitive Data

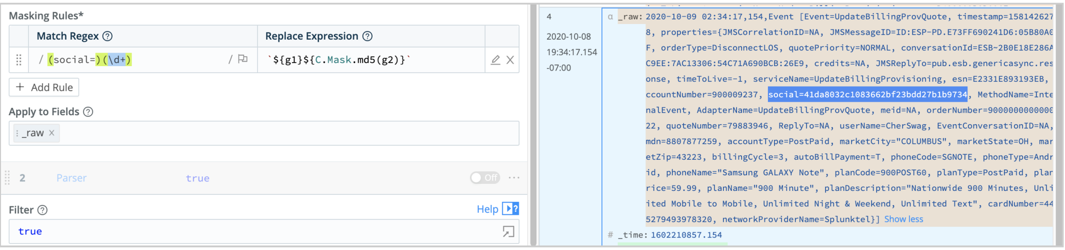
In the right **Preview** pane, notice each that event includes a **social** key, whose value is a (fictitious) raw Social Security number. Before this data goes any further through our Pipeline, let's use LogStream's `Mask` Function to swap in an md5 hash of each SSN.

1. In the left **Pipelines** pane, click `+ + Function`.
2. Search for `Mask`, then click it.
3. In the new Function's **Masking Rules**, click the into **Match Regex** field.
4. Enter or paste this regex, which simply looks for digits following `social=`: `(social=)(\d+)`
5. In **Replace Expression**, paste the following hash function. The backticks are literal: `` `${g1}${C.Mask.md5(g2)}` ``
6. Note that **Apply to Fields** defaults to `_raw`. This is what we want to target, so we'll accept this default.
7. Click **Save**.

You'll immediately notice some obvious changes:

- The **Preview** pane has switched from its **IN** to its **OUT** tab, to show you the outbound effect of the Pipeline you just saved.
- Each event's `_raw` field has changed color, to indicate that it's undergone some redactions.

Now locate at least one event's **Show more** link, and click to expand it. You can verify that the `social` values have now been hashed.



Masking Rules*

Match Regex	Replace Expression
`/ (social=)(\d+) /`	`` `${g1}${C.Mask.md5(g2)}` ``

Apply to Fields: _raw

Filter: `true`

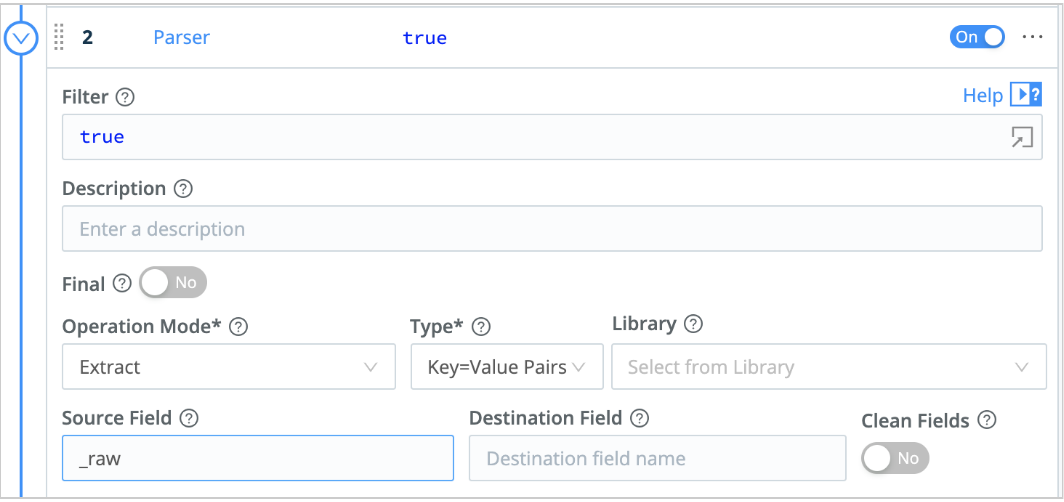
```
4
2020-10-08
19:34:17.154
-07:00
_raw: 2020-10-09 02:34:17,154,Event [Event=UpdateBillingProvQuote, timestamp=158142627
      8, properties={JMSCorrelationID=NA, JMSMessageID=ID:ESP-PD.E73FF690241D6:05B80A0
      F, orderType=DisconnectLOS, quotePriority=NORMAL, conversationId=ESB~2B0E18E286A
      C9EE:7AC13306:54C71A690BCB:26E9, credits=NA, JMSReplyTo=pub.esb.genericasync.res
      onse, timeToLive=-1, serviceName=UpdateBillingProvisioning, esn=E2331E893193EB,
      ccountNumber=900009237, social=41da8032c1083662bf23bdd27b1b9734, MethodName=Inte
      nalEvent, AdapterName=UpdateBillingProvQuote, meid=NA, orderNumber=900000000000
      22, quoteNumber=79883946, ReplyTo=NA, userName=CherSwag, EventConversationID=NA,
      mdn=8807877259, accountType=PostPaid, marketCity="COLUMBUS", marketState=OH, mar
      etZip=43223, billingCycle=3, autoBillPayment=T, phoneCode=SGNOTE, phoneType=Andr
      id, phoneName="Samsung GALAXY Note", planCode=900POST60, planType=PostPaid, plar
      rice=59.99, planName="900 Minute", planDescription="Nationwide 900 Minutes, Unli
      ited Mobile to Mobile, Unlimited Night & Weekend, Unlimited Text", cardNumber=44
      5279493978320, networkProviderName=Splunktel}] Show less
# _time: 1602210857.154
```

Mask Function and hashed result

Parser: Extract Events

Having redacted sensitive data, we'll next use a Parser function to lift up all the `_raw` field's key-value pairs as fields:

1. In the left **Pipelines** pane, click `+ Function`.
2. Search for `Parser`, then click it.
3. Leave the **Operation Mode** set to its `Extract` default.
4. Set the **Type** to `Key=Value Pairs`.
5. Leave the **Source Field** set to its `_raw` default.
6. Click **Save**.



2 Parser true On

Filter: true

Description: Enter a description

Final: No

Operation Mode*: Extract

Type*: Key=Value Pairs

Library: Select from Library

Source Field: _raw

Destination Field: Destination field name

Clean Fields: No

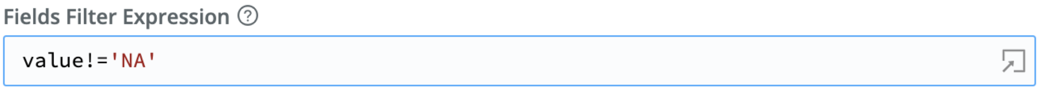
Parser configured to extract K=V pairs from `_raw`

You should see the **Preview** pane instantly light up with a lot more fields, parsed from `_raw`. You now have rich structured data, but not all of this data is particularly interesting: Note how many fields have `NA` ("Not Applicable") values. We can enhance the **Parser** Function to ignore fields with `NA` values.

1. In the Function's **Fields Filter Expression** field (near the bottom), enter this negation expression: `value≠'NA'`

   Note the single-quoted value. If you type (rather than paste) this expression, watch how typeahead matches the first quote you type.

2. Click **Save**, and watch the **Preview** pane.



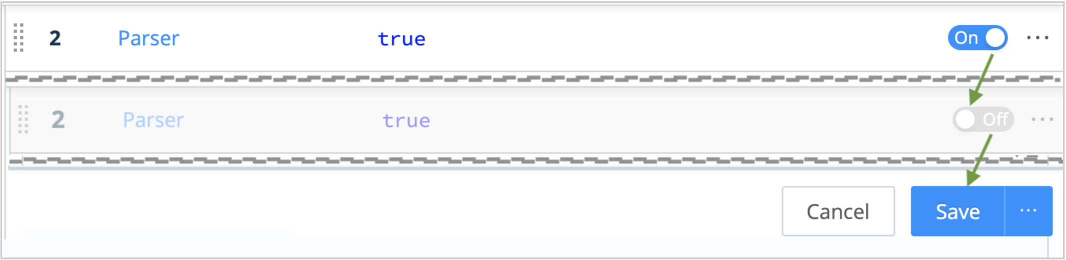
Fields Filter Expression

```
value!='NA'
```

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

Several fields should disappear – such as `credits` , `EventConversationID` , and `ReplyTo` . The remaining fields should display meaningful values. Congratulations! Your log data is already starting to look better-organized and less bloated.

> Missed It?
>
> If you didn't see the fields change, slide the Parser Function **Off**, click **Save** below, and watch the **Preview** pane change. Using these toggles, you can preserve structure as you test and troubleshoot each Function's effect.
>
> Note that each Function also has a **Final** toggle, defaulting to **Off**. Enabling **Final** anywhere in the Functions stack will prevent data from flowing to any Functions lower in the UI.
>
> Be sure to toggle the Function back **On**, and click **Save** again, before you proceed!



Toggling a Function off and on

Next, let's add an extra field, and conditionally infer its value from existing values. We'll also remove the `_raw` field, now that it's redundant. To add and remove fields, the **Eval** Function is our pal.

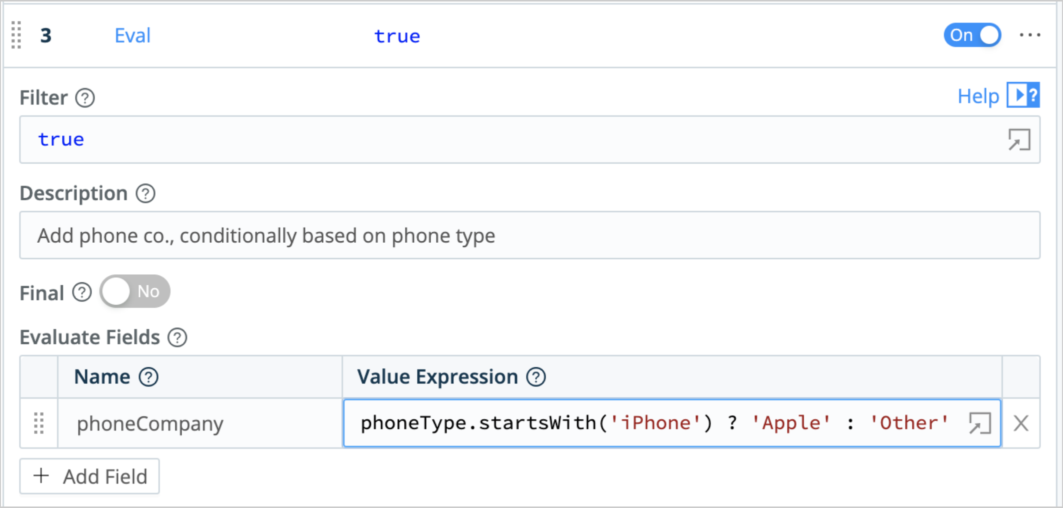
Eval: Add and Remove Fields

Let's assume we want to enrich our data by identifying the manufacturer of a certain popular phone handset. We can infer this from the existing `phoneType` field that we've lifted up for each event.

Add Field (Enrich)

1. In the left **Pipelines** pane, click `+ Function`.
2. Search for `Eval`, then click it.
3. Click into the new Function's **Evaluate Fields** table.

   Here you add new fields to events, defining each field as a key-value pair. If we needed more key-value pairs, we could click `+ Add Field` for more rows.
4. In **Name**, enter: `phoneCompany`.
5. In **Value Expression**, enter this JS ternary expression that tests `phoneType`'s value: `phoneType.startsWith('iPhone') ? 'Apple' : 'Other'` (Note the `?` and `:` operators, and the single-quoted values.)
6. Click **Save**. Examine some events in the **Preview** pane, and each should now contain a `phoneCompany` field that matches its `phoneType`.



true

Filter: true

Description: Add phone co., conditionally based on phone type

Final: No

Evaluate Fields

Name	Value Expression
phoneCompany	`phoneType.startsWith('iPhone') ? 'Apple' : 'Other'`

Adding a field to enrich data

Remove Field (Shrink Data)

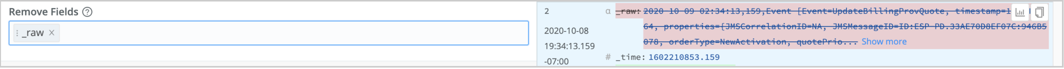
Now that we've parsed out all of the `_raw` field's data – it can go. Deleting a (large) redundant field will give us cleaner events, and reduced load on downstream resources.

1. Still in the **Eval** Function, click into **Remove Fields**

2. Type: `_raw` and press **Tab** or **Enter**.

3. Click **Save**.

The **Preview** pane's diff view should now show each event's `_raw` field stripped out.



Remove Fields: _raw

2
2020-10-08 19:34:13.159 -07:00

~~_raw: 2020-10-09 02:34:13,159,Event [Event=UpdateBillingProvQuote, timestamp=1 64, properties={JMSCorrelationID=NA, JMSMessageID=ID:ESP-PD.33AE70D8EF07C:946B5 078, orderType=NewActivation, quotePrio...~~ Show more

_time: 1602210853.159

Removing a field to streamline data

Our log data has now been cleansed, structured, enriched, and slimmed-down. Let's next look at how to make it more legible, by giving fields simpler names.

Rename: Refine Field Names

1. In the left **Pipelines** pane, click `+ Function`.

   This rhythm should now be familiar to you.

2. Search for `Rename`, then click it.

3. Click into the new Function's **Rename Fields** table.

   This has the same structure you saw above in Eval: Each row defines a key-value pair.

4. In **Current Name**, enter the longhaired existing field name: `conversationId`.

5. In **New Name**, enter the simplified field name: `ID`.

6. Watch any event's `conversationId` field in the **Preview** pane as you click **Save** at left. This field should change to `ID` in all events.

Drop: Remove Unneeded Events

We've already refined our data substantially. To further slim it down, a Pipeline can entirely remove events that aren't of interest for a particular downstream service.

> As the "Pipeline" name implies, your LogStream installation can have multiple Pipelines, each configured to send out a data stream

> tailored to a particular Destination. This helps you get the right data in the right places most efficiently.

Here, let's drop all events for customers who use prepaid monthly phone service (i.e., **not** postpaid):

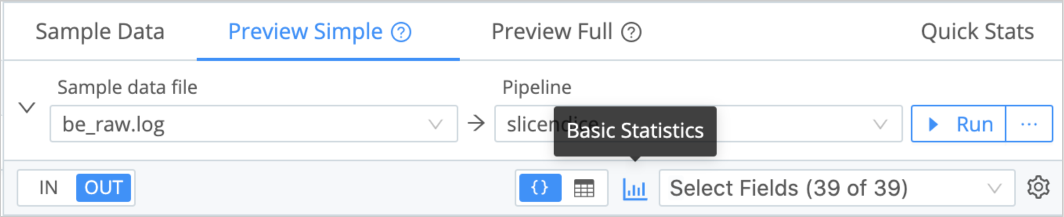
1. In the left **Pipelines** pane, click `+ Function`.
2. Search for `Drop`, then click it.
3. Click into the new Function's **Filter** field.
4. Replace the default `true` value with this JS negation expression: `accountType≠'PostPaid'`
5. Click **Save**.

Now scroll through the right **Preview** pane. Depending on your data sample, you should now see multiple events struck out and faded – indicating that LogStream will drop them before forwarding the data.

A Second Look at Our Data

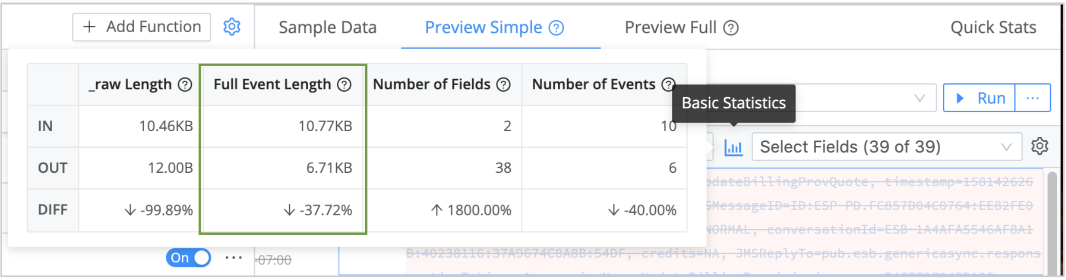
Torture the data enough, and it will confess. By what factor have our transformations refined our data's volume? Let's check.

In the right **Preview** pane, click the **Basic Statistics** button:



Displaying Basic Statistics

Even without the removal of the `_raw` field (back in Eval) and the dropped events, you should see a substantial % reduction in the **Full Event Length**.



	_raw Length	Full Event Length	Number of Fields	Number of Events
IN	10.46KB	10.77KB	2	10
OUT	12.00B	6.71KB	38	6
DIFF	↓ -99.89%	↓ -37.72%	↑ 1800.00%	↓ -40.00%

Data reduction quantified

Woo hoo! Before we wrap up our configuration: If you're curious about individual Functions' independent contribution to the data reduction shown here, you can test it now. Use the toggle **Off** > **Save** > **Basic Statistics** sequence to check various changes.

Add and Attach a Route

We've now built a complete, functional Pipeline. But so far, we've tested its effects only on the static data sample we captured earlier. To get dynamic data flowing through a Pipeline, we need to filter that data in, by defining a LogStream Route.

1. At the **Pipelines** page's top left, click **Attach Pipeline to Route**.

   This displays the **Routes** page. It's structured very similarly to the **Pipelines** page, so the rhythm here should feel familiar.

2. Click `+ Route`.

3. Enter a unique, meaningful **Route Name**, like `demo`.

4. Leave the **Filter** field set to its `true` default, allowing it to deliver all events.

   Because a Route delivers events to a Pipeline, it offers a first stage of filtering. In production, you'd typically configure each Route to filter events by appropriate `source`, `sourcetype`, `index`, `host`, `_time`, or other characteristics. The **Filter** field accepts JavaScript expressions, including AND (`&&`) and OR (`||`) operators.

5. Set the **Pipeline** drop-down to our configured `slicendice` Pipeline.

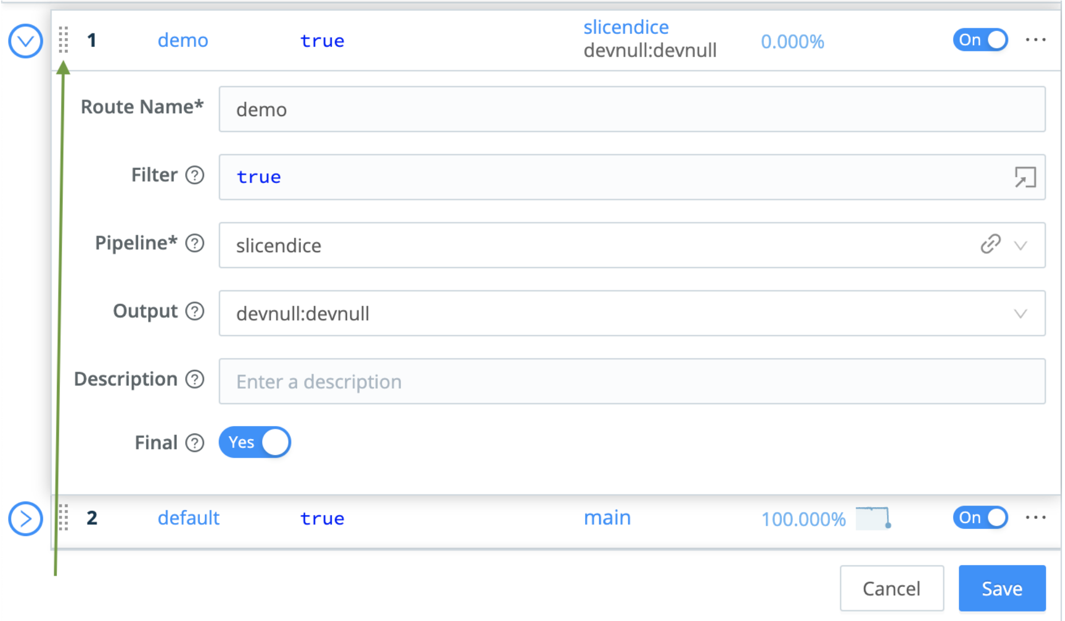
6. Set the **Output** drop-down to either `devnull` or `default`.

This doesn't matter, because we've set `default` as a pointer to `devnull`. In production, you'd set this carefully.

7. You can leave the **Description** empty, and leave **Final** set to **Yes**.

8. Grab the new Route by its left handle, and drag it above the `default` Route, so that our new Route will process events first. You should see something like the screenshot below.

9. Click **Save** to save the new Route to the Routing table.



#	Route	Filter	Pipeline/Output	Events	
1	demo	`true`	slicendice devnull:devnull	0.000%	On

Route Name*: demo

Filter: `true`

Pipeline*: slicendice

Output: devnull:devnull

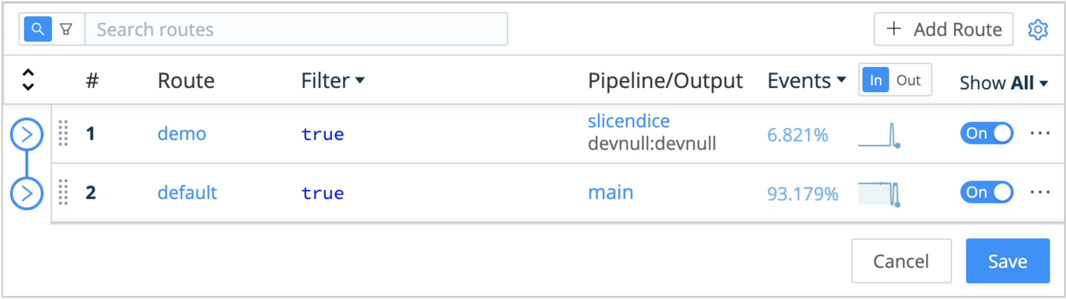
Description: Enter a description

Final: Yes

#	Route	Filter	Pipeline/Output	Events	
2	default	`true`	main	100.000%	On

Configuring and adding a Route

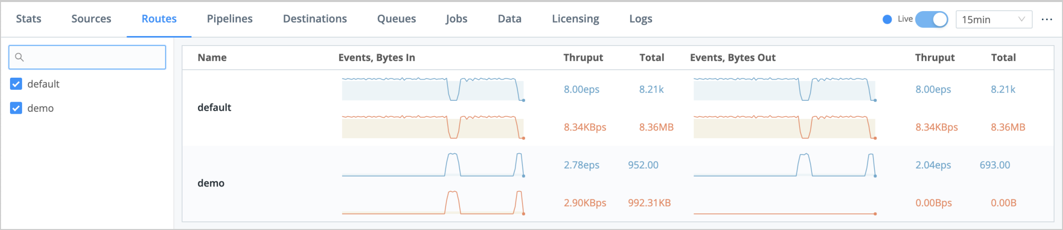
The sparklines should immediately confirm that data is flowing through your new Route:



#	Route	Filter	Pipeline/Output	Events (In / Out)	Show All
1	demo	`true`	slicendice devnull:devnull	6.821%	On
2	default	`true`	main	93.179%	On

Live Routes

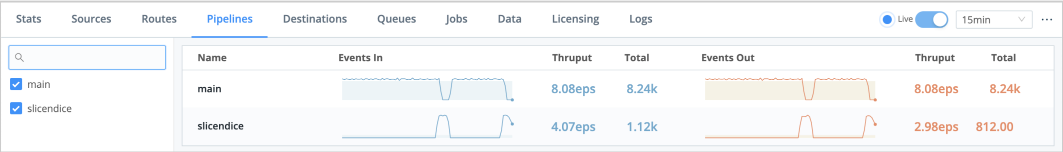
To confirm data flow through the whole system we've built, select **Monitoring > Routes** from LogStream's top menu and examine `demo`.



Name	Events, Bytes In	Thruput	Total	Events, Bytes Out	Thruput	Total
default		8.00eps	8.21k		8.00eps	8.21k
		8.34KBps	8.36MB		8.34KBps	8.36MB
demo		2.78eps	952.00		2.04eps	693.00
		2.90KBps	992.31KB		0.00Bps	0.00B

Monitoring data flow through Routes

Also select **Monitoring > Pipelines** and examine `slicendice` .



Name	Events In	Thruput	Total	Events Out	Thruput	Total
main		8.08eps	8.24k		8.08eps	8.24k
slicendice		4.07eps	1.12k		2.98eps	812.00

Monitoring data flow through Pipelines

What Have We Done?

Look at you! Give yourself a pat on the back! In this short, scenic tour – with no hit to your cloud-services charges – you've build a simple but complete LogStream system, exercising all of its basic components:

- Downloaded, installed, and run LogStream.
- Configured a Source to hook up an input.
- Configured a Destination to feed an output.
- Monitored data throughput, and checked it twice.
- Built a Pipeline.
- Configured LogStream Functions to redact, parse, enrich, trim, rename, and drop event data.
- Added and attached a Route to get data flowing through our Pipeline.

Next Steps

Interested in guided walk-throughs of more-advanced LogStream features? We suggest that next, you check out:

- LogStream Sandboxes: Work through general and specific scenarios in containers. with terminal access and free, hosted data inputs and outputs.

- Use Cases documentation: Bring your own services to build solutions to specific challenges.
- Cribl Concept: Pipelines – Video showing how to build and use Pipelines at multiple LogStream stages.
- Cribl Concept: Routing – Video about using Routes to send different data through different paths.

Cleaning Up

Oh yeah, you've still got the LogStream server running, with its `businessevent.log` datagen wtill firing events. If you'd like to shut these down for now, in reverse order:

1. Go to **Data > Sources > Datagens**.
2. Slide `businessevent` to **Off**, and click **Save**. (Refer back to the screenshot above.)
3. In your terminal's `$CRIBL_HOME/bin` directory, shut down the server with:

   ```
   ./cribl stop
   ```

That's it! Enjoy using LogStream.

What's Next

› Deployment Types

DEPLOYMENT

Deployment Types

Deployment guide to get you started with Cribl

There are at least **two** key factors that will determine the type of Cribl LogStream 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?

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. See the note below before you plan.

> ⚠ Cribl App for Splunk Deprecation Notice

> Click here.

Kubernetes/Helm Deployment

You can deploy LogStream Leader Nodes (or single instances) and Worker Nodes via Cribl's Helm charts.

Docker Deployment

You can deploy LogStream instances using images from Cribl's public Docker Hub.

What's Next

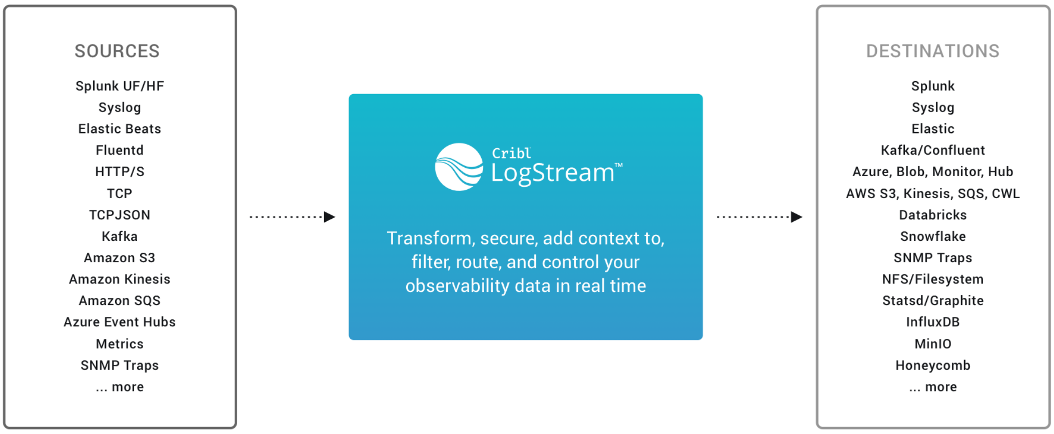
- Single-Instance Deployment
- Distributed Deployment
- Splunk App Deployment *
- Bootstrap Workers from Master
- Kubernetes/Helm Deployment
- Docker Deployment

Single-Instance Deployment

Getting started with Cribl LogStream on a single instance

For small-volume or light processing environments – or for test or evaluation use cases – a single instance of Cribl LogStream might be sufficient to serve all inputs, event processing, and outputs. This page outlines how to implement a single-instance deployment.

Architecture



Requirements

- **OS:**
  - Linux 64-bit kernel >= 3.10 and glibc >= 2.17
  - Examples: Ubuntu 16.04, Debian 9, RHEL 7, CentOS 7, SUSE Linux Enterprise Server 12+, Amazon Linux 2014.03+

  > ⚠ Mac OS is no longer supported as of v. 2.3, due to LogStream's incorporation of Linux-native components.

- **System:**
  - +4 physical cores, +8GB RAM

  - 5GB free disk space (more if persistent queuing is enabled)

    > i We assume that 1 physical core is equivalent to 2 virtual/hyperthreaded CPUs (vCPUs). All quantities listed above are minimum requirements. To fulfill the above requirements using cloud-based virtual machines, see Recommended AWS, Azure, and GCP Instance Types.

- **Browser Support**: Firefox 65+, Chrome 70+, Safari 12+, Microsoft Edge

Network Ports

By default, LogStream listens on the following ports:

Component	Default Port
UI	`9000`
HTTP In	`10080`
Splunk to Cribl LogStream data port	`localhost:10000` (Cribl App for Splunk)
`| criblstream` Splunk search command to Cribl LogStream	`localhost:10420` (Cribl App for Splunk)
User options	+ Other data ports as required.

Overriding Default Ports

The above ports can be overridden in the following configuration files:

- Cribl UI port (`9000`): Default definitions for `host` , `port` , and other settings are set in `$CRIBL_HOME/default/cribl/cribl.yml` , and can be overridden by defining alternatives in `$CRIBL_HOME/local/cribl/cribl.yml` .

- Data Ports: HTTP In (`10080`), TCPJSON in (`10420`) Splunk to Cribl (`10000`) : Default definitions for `host` , `port` and other settings are set in `$CRIBL_HOME/default/cribl/inputs.yml` , and can be overridden by defining alternatives in `$CRIBL_HOME/local/cribl/inputs.yml` .

Installing on Linux

- 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., `/opt/` :
  - `tar xvzf cribl-<version>-<build>-<arch>.tgz`

Running

Go to the `$CRIBL_HOME/bin` directory, where the package was extracted (e.g.: `/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`)

> i 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 LogStream with Sources and Destinations, or start creating Routes and Pipelines.

> i In the case of an API port conflict, the process will retry binding for 10 minutes before exiting.

Enabling Start on Boot

Cribl LogStream ships with a CLI utility that can update your system's configuration to start LogStream at system boot time. The basic format to invoke this utility is:

```
[sudo] $CRIBL_HOME/bin/cribl boot-start [enable|disable] [options] [args]
```

> You will need to run this command as root, or with `sudo`. For options and arguments, see the CLI Reference.

Most, if not all, popular Linux distributions use `systemd` now to start processes at boot, while older or more obscure distributions may still use `initd`. Verify with your Linux distribution vendor if you aren't sure which method your systems use in order to know which procedure listed below to follow.

Using systemd

To **enable** Cribl LogStream 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 LogStream. E.g., to run LogStream 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 below) and start Cribl LogStream 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 LogStream 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
```

Note the file's default `65536` hard limit on maximum open file descriptors (known as a ulimit). The minimum recommended 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 LogStream.
After=network.target
```

```
[Service]
Type=forking
User=cribl
Restart=on-failure
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
ExecStopPost='/bin/rm -f /install/path/to/cribl/pid/cribl.pid'
ExecReload=/install/path/to/cribl/bin/cribl reload
TimeoutSec=60

[Install]
WantedBy=multi-user.target
```

> ⚠ Do NOT Run LogStream as Root!
>
> If LogStream is required to listen on ports 1–1024, it will need privileged access. You can enable this on systemd by adding this configuration key:
>
> ```
> [Service]
> AmbientCapabilities=CAP_NET_BIND_SERVICE
> ```

Using initd

To **enable** Cribl LogStream to start at boot time with **initd**, you need to run the `boot-start` command. If the user that you want to run LogStreams does not exist, create it prior to executing. E.g., running LogStream 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 start Cribl LogStream 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 LogStream 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
```

> ⚠ Do NOT Run LogStream as Root!
>
> If LogStream is required to listen on ports 1–1024, it will need privileged access. On a Linux system with POSIX capabilities, you can achieve this 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`
>
> Upon starting the LogStream server, a `bind EACCES 0.0.0.0:<port>` error in the API/worker logs (depending on the service) might indicate that `setcap` did not successfully execute.

System Proxy Configuration

For details on configuring LogStream 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.

Distributed Deployment

Getting started with Cribl LogStream on a distributed deployment

Distributed Deployment

To sustain higher incoming data volumes, and/or increased processing, you can scale from a single instance up to a multi-instance, distributed deployment. Instances in the deployment independently serve all inputs, process events, and send to outputs.

The instances are managed centrally by a single Leader Node, which is responsible for keeping configurations in sync, and for tracking and monitoring the instances' activity metrics.

> For some use cases for distributed deployments, see Worker Groups – What Are They and Why You Should Care.
>
> As of version 3.0, LogStream'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.

Concepts

Single Instance – a single Cribl LogStream instance, running as a standalone (not distributed) installation on one server.

Leader Node – a LogStream instance running in **Leader** mode, used to centrally author configurations and monitor Worker Nodes in a distributed deployment.

Worker Node – a LogStream instance running as a **managed Worker**, whose configuration is fully managed by a Leader Node. (By default, will poll the

Leader for configuration changes every 10 seconds.)

Worker Group – a collection of Worker Nodes that share the same configuration. You map Nodes to a Worker Group using a Mapping Ruleset.

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
>
> A Worker Node's local running config can be manually overridden/changed, but changes won't persist on the filesystem. To permanently modify a Worker Node's config: Save, commit, and deploy it from the Leader. See Deploying Configurations below.
>
> With an Enterprise license, you can configure role-based access control at the Worker Group level. Non-administrator users will then be able to access Workers only within those Worker Groups on which they're authorized.

Aggregating Workers

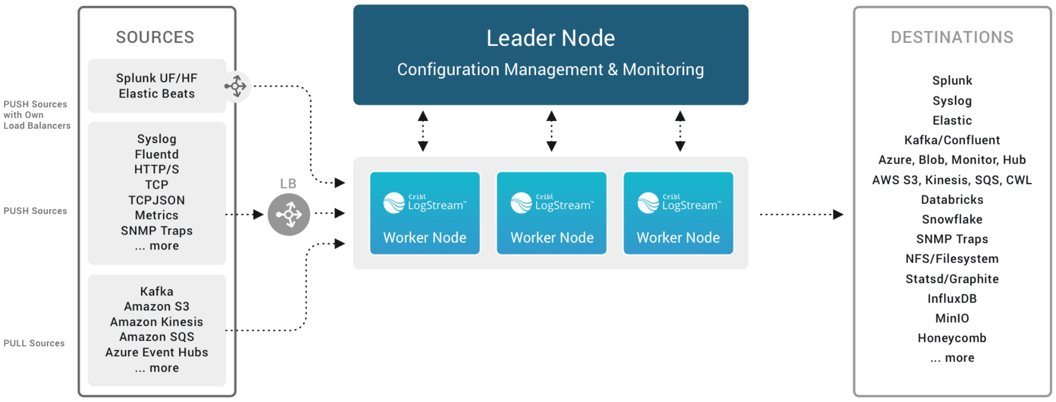
To clarify how the above concepts add up hierarchically, let's use a military metaphor involving toy soldiers:

- Worker Process = soldier.
- Worker Node = multiple Worker Processes = squad.
- Worker Group = multiple Worker Nodes = platoon.

Multiple Worker Groups are very useful in making your configuration reflect organizational or geographic constraints. E.g., you might have a U.S. Worker Group with certain TLS certificates and output settings, versus an APAC Worker Group and an EMEA Worker Group, each with their own distinct certs and settings.

Architecture

This is an overview of a distributed LogStream deployment's components.



Distributed deployment architecture

Here is the division of labor among components of the Leader Node and Worker Node.

Leader Node

- API Process – Handles all the API interactions.
- *N* Config Helpers – One process per Worker Group. Helps with maintaining configs, previews, etc.

Worker Node

- API Process – Handles communication with the Leader Node (i.e., with its API Process) and handles other API requests.
- *N* Worker Processes – Handle all the data processing.

Single-Instance Architecture

For comparison, here's the simpler division of labor on a single-instance deployment, where the separate Leader versus Worker Nodes are essentially condensed into one stack:

- API Process – Handles all the API interactions.
- *N* Worker Process – Handle all data processing,
- One of the Worker Processes is called the leader Worker Process. (Not to be confused with the Leader Node.) This is responsible for writing configs to disk, in addition to data processing.

So here, the API Process handles the same responsibilities as a Leader Node's API Process, while the Worker Processes correspond to the Worker Nodes' Worker Processes. The exception is that one Worker Process does double duty, also filling in for one of the Leader Node's Config Helpers.

Leader Node Requirements

- **OS**:
  - Linux: RedHat, CentOS, Ubuntu, AWS Linux (64bit)
- **System**:
  - +4 physical cores, +8GB RAM
  - 5GB free disk space
- **Git**: `git` must be available on the Leader Node. See details below.
- **Browser Support**: Firefox 65+, Chrome 70+, Safari 12+, Microsoft Edge

> ℹ We assume that 1 physical core is equivalent to 2 virtual/hyperthreaded CPUs (vCPUs). All quantities listed above are minimum requirements.

> ⚠ Mac OS is no longer supported as of v. 2.3, due to LogStream's incorporation of Linux-native features.

Worker Node Requirements

See Single-Instance Deployment for requirements and Sizing and Scaling for capacity planning details.

Network Ports – Leader Node

In a distributed deployment, Workers communicate with the Leader Node on these ports. Ensure that the Leader is reachable on those ports from **all** Workers.

Component	Default Port

Heartbeat	`4200`

Network Ports – Worker Nodes

By default, all LogStream Worker instances listen on the following ports:

Component	Default Port
UI	`9000`
User options	+ Other data ports as required.

Installing on Linux

See Single-Instance Deployment, as the installation procedures are identical.

Version Control with `git`

LogStream 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, check here 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.

Setting up Leader and Worker Nodes

1. Configuring a Leader Node

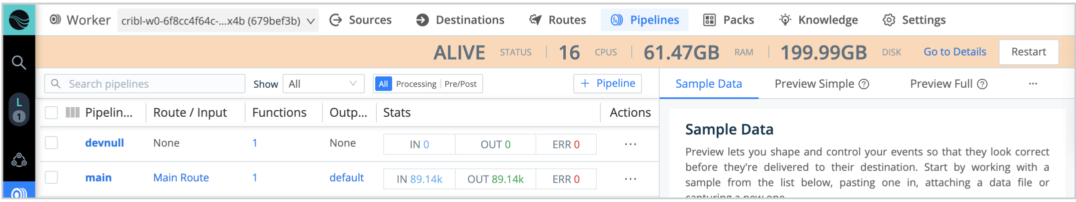
You can configure a Leader Node through the UI, through the `instance.yml` config file, or through the command line.

Using the UI

In global ⚙ **Settings** (lower left) > **Distributed Settings** > **Distributed Management** > **General Settings**, select **Mode**: **Leader**.

Next, on the **Leader Settings** left tab, confirm or enter the required Leader settings (**Address** and **Port**). Customize the optional settings if desired. Then click **Save** to restart.

> Worker UI Access
>
> If you enable the nearby global ⚙ **Settings** > **Distributed Settings** > **Leader Settings** > **Worker UI access** option (which corresponds to the `enabledWorkerRemoteAccess` key), you will be able to click through from the Leader's **Manage Worker Nodes** screen to an authenticated view of each Worker's UI. An orange header labeled **Viewing Worker:** `<host/GUID>` will be added, to confirm that you are remotely viewing a Worker's UI.



Pipelin...	Route / Input	Functions	Outp...	Stats			Actions
devnull	None	1	None	IN 0	OUT 0	ERR 0	···
main	Main Route	1	default	IN 89.14k	OUT 89.14k	ERR 0	···

Sample Data

Preview lets you shape and control your events so that they look correct before they're delivered to their destination. Start by working with a sample from the list below, pasting one in, attaching a data file or capturing a new one

Worker UI access

Using YAML Config File

In `$CRIBL_HOME/local/_system/instance.yml`, under the `distributed` section, set `mode` to `master`:

$CRIBL_HOME/local/_system/instance.yml

```
distributed:
  mode: master
  master:
    host: <IP or 0.0.0.0>
    port: 4200
    tls:
      disabled: true
    ipWhitelistRegex: /.*/
    authToken: <auth token>
    enabledWorkerRemoteAccess: false
    compression: none
    connectionTimeout: 5000
    writeTimeout: 10000
```

Using the Command Line

You can configure a Leader Node using a CLI command of this form:

```
./cribl mode-master [options] [args]
```

For all options, see the CLI Reference.

2. Configuring a Worker Node

On each LogStream instance you designate as a Worker Node, you can configure the Worker through the UI, the `instance.yml` config file, environment variables, or the command line.

Using the UI

In global ⚙ **Settings** (lower left) > **Distributed Settings** > **Distributed Management** > **General Settings**, select **Mode**: **Worker**.

Next, confirm or enter the required Leader settings (**Address** and **Port**). Customize the optional settings if desired. Then click **Save** to restart.

Using YAML Config File

In `$CRIBL_HOME/local/_system/instance.yml` , under the `distributed` section, set `mode` to `worker` :

$CRIBL_HOME/local/_system/instance.yml

```
distributed:
  mode: worker
  envRegex: /^CRIBL_/
  master:
    host: <master address>
    port: 4200
    authToken: <token here>
    compression: none
    tls:
      disabled: true
    connectionTimeout: 5000
    writeTimeout: 10000
  tags:
       - tag1
       - tag2
       - tag42
  group: teamsters
```

Using Environment Variables

You can configure Worker Nodes via environment variables, as in this example:

```
 CRIBL_DIST_MASTER_URL=tcp://criblmaster@masterHostname:4203
./cribl start
```

See the Environment Variables section for more details.

Using the Command Line

You can configure a Worker Node using CLI commands of this form:

```
./cribl mode-worker -H <master-hostname-or-IP> -p <port> [options] [args]
```

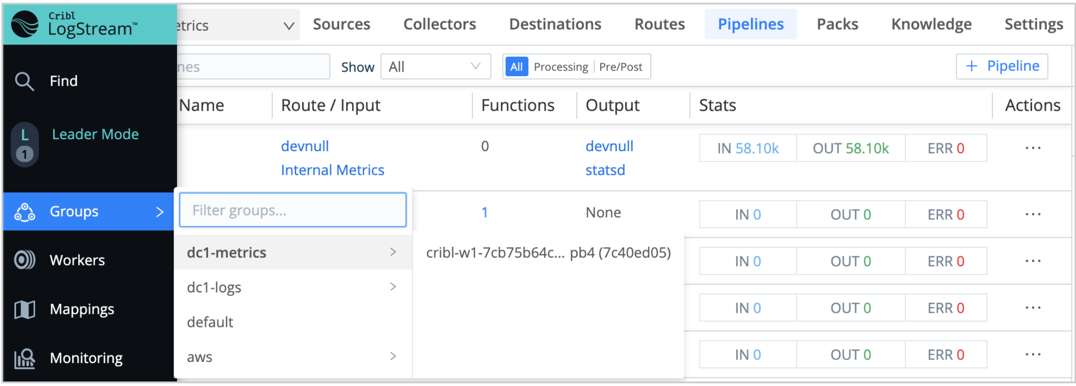
The `-H` and `-p` parameters are required. For other options, see the CLI Reference. Here is an example command:

```
./cribl mode-worker -H 192.0.2.1 -p 4200 -u myAuthToken
```

LogStream will need to restart after this command is issued.

Menu Changes in Distributed Mode

Compared to a single-instance deployment, deploying in distributed mode changes LogStream's menu structure in a few ways. The left nav adds **Leader Mode**, **Groups**, **Workers**, and **Mappings** tabs – all to manage Workers and their assignments. Also, the global **Monitoring** link moves from the top to the left nav.

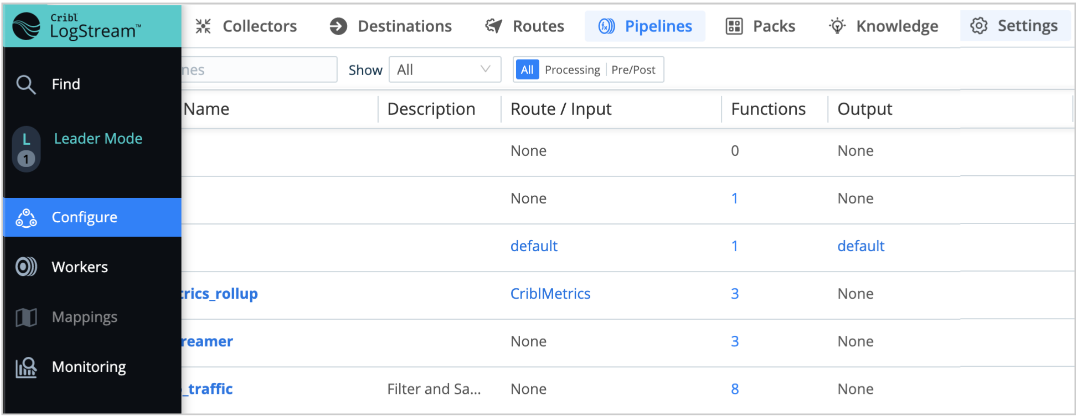


Name	Route / Input	Functions	Output	Stats	Actions
	devnull Internal Metrics	0	devnull statsd	IN 58.10k OUT 58.10k ERR 0	⋯
		1	None	IN 0 OUT 0 ERR 0	⋯
				IN 0 OUT 0 ERR 0	⋯
				IN 0 OUT 0 ERR 0	⋯
				IN 0 OUT 0 ERR 0	⋯

Distributed deployment: menu structure

To access the Group-specific top nav shown above, click **Groups**, then click into your desired Worker Group. This contextual menu also adds a **Settings** tab, through which you can manage configuration per Worker Group.

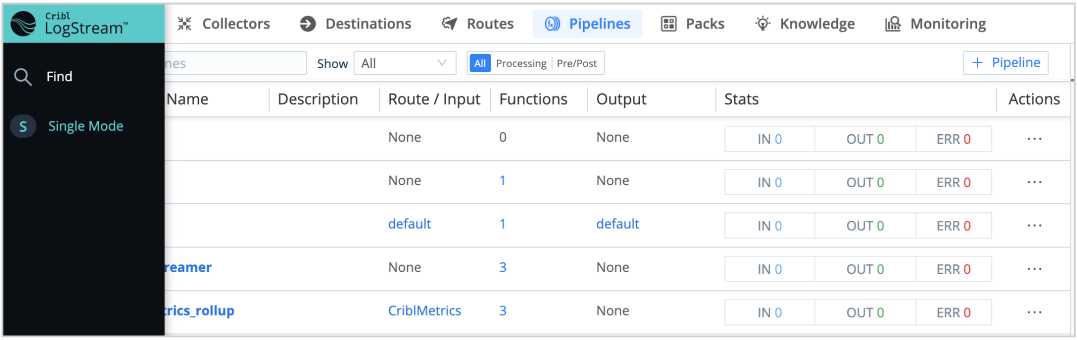
If you have a LogStream Free or LogStream One license and use distributed mode, the left nav's **Groups** link instead reads **Configure**, because these license types allow only one group. Therefore, throughout this documentation, interpret any reference to the "**Groups** link" as "**Configure** link" in your installation. Here, the top nav's added **Settings** link opens configuration specific to the same `default` group.



Name	Description	Route / Input	Functions	Output
		None	0	None
		None	1	None
		default	1	default
rics_rollup		CriblMetrics	3	None
reamer		None	3	None
_traffic	Filter and Sa...	None	8	None

Distributed deployment with LogStream Free/One license

For comparison, here is a single-instance deployment's consolidated top-menu structure:



Name	Description	Route / Input	Functions	Output	Stats			Actions
		None	0	None	IN 0	OUT 0	ERR 0	···
		None	1	None	IN 0	OUT 0	ERR 0	···
		default	1	default	IN 0	OUT 0	ERR 0	···
reamer		None	3	None	IN 0	OUT 0	ERR 0	···
rics_rollup		CriblMetrics	3	None	IN 0	OUT 0	ERR 0	···

Single-instance deployment: anchored top menu

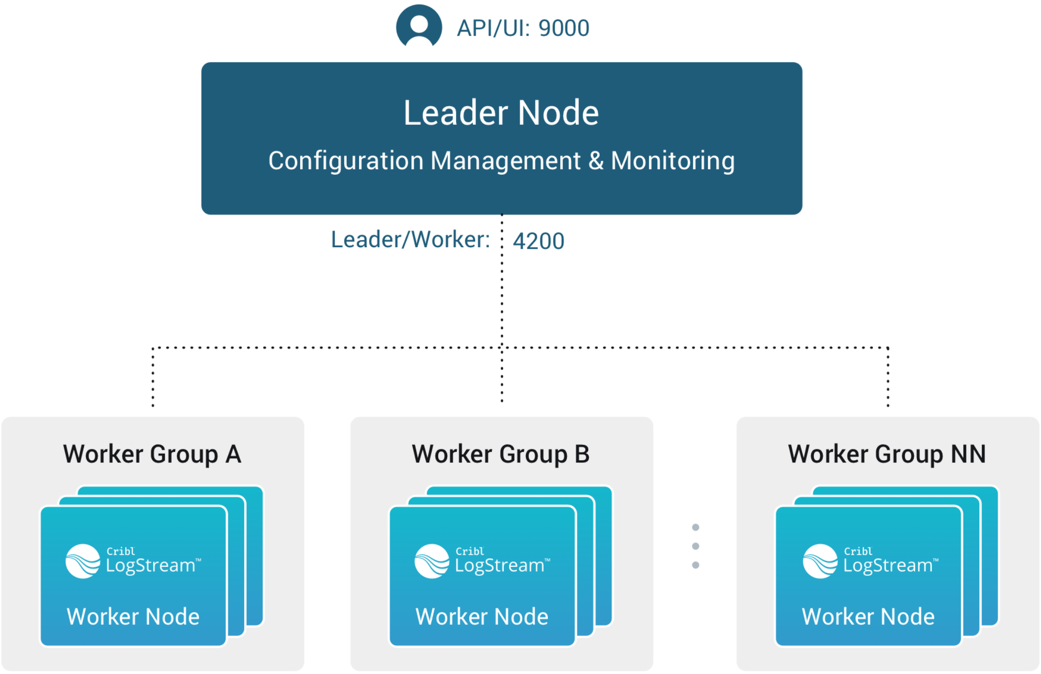
> ⚠ Distributed mode's repositioning of navigation/menu links also applies to several instructions and screenshots that you'll see throughout this documentation.
>
> Where procedures are written around a single-instance scenario, just click into your appropriate Group to access the corresponding navigation links.

How Do Workers and Leader Work Together

The Leader Node has two primary roles:

1. Serves as a central location for Workers' operational metrics. The Leader ships with a monitoring console that has a number of dashboards, covering almost every operational aspect of the deployment.

2. Serves as a central location for authoring, validating, deploying, and synchronizing configurations across Worker Groups.



Leader Node/Worker Nodes relationship

Network Port Requirements (Defaults)

- UI access to Leader Node: TCP 9000.
- Worker Node to Leader Node: TCP 4200 (Heartbeat/Metrics/other).

Leader/Worker Node Communication

Workers will periodically (every 10 seconds) send a heartbeat to the Leader. This heartbeat includes information about themselves, 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 tracks with the connection.

The failure of a Worker Node to successfully send two consecutive heartbeat messages to the Leader will cause the respective Worker to be removed from the Workers page in the Leader's UI until the Leader receives a heartbeat message from the affected Worker.

When a Worker Node checks in with the Leader:

- The Worker sends heartbeat to Leader.
- The Leader uses the Worker’s facts and Mapping Rules to map it to a Worker Group.
- The Worker Node pulls its Group's updated configuration bundle, if necessary.

Config Bundle Management

Config bundles are compressed archives of all config files and associated data that a Worker needs to operate. The Leader creates bundles upon Deploy, and manages them as follows:

- Bundles are wiped clean on startup.
- While running, at most 5 bundles per group are kept.
- Bundle cleanup is invoked when a new bundle is created.

The Worker pulls bundles from the Leader and manages them as follows:

- Last 5 bundles and backup files are kept.
- At any point in time, all files created in the last 10 minutes are kept.
- Bundle cleanup is invoked after a reconfigure.

Worker Groups

Worker Groups facilitate authoring and management of configuration settings for a particular set of Workers. To create a new Worker Group, click **Groups** from the left nav and, from the resulting **Manage Groups** page, click **+ Add New**.

> Configuring multiple Worker Groups requires a LogStream Enterprise or Standard license, and configuring or more than 10 Worker Processes requires at least a LogStream One license.

Configuring a Worker Group

Click on newly created Group's **Configure** button to display an interface for **authoring and validating** its configuration. You can configure everything for this Group as if it were a single LogStream instance – using a similar visual interface for Routes, Pipelines, Sources, Destinations, and Group-specific **Settings**.

> ⚠ Can't Log into the Worker Node as Admin User?
>
> To explicitly set passwords for Worker Groups, see User Authentication.

Mapping Workers to Worker Groups

Mapping Rulesets are used to map Workers to Worker Groups. Within a ruleset, a list of rules evaluate Filter expressions on the information that Workers send to the Leader.

Only one Mapping Ruleset can be active at any one time, although a ruleset can contain multiple rules. At least one Worker Group should be defined and present in the system.

The ruleset behavior is similar to Routes, where the order matters, and the **Filter** section supports full JS expressions. The ruleset matching strategy is first-match, and one Worker can belong to only one Worker Group.

Creating a Mapping Ruleset

To create a Mapping Ruleset, Click **Mappings** from the left nav and then, from the resulting **Manage Mapping Rulesets** page, click **+ Add New**. Give the resulting **New Ruleset* a unique** ID **and click** Save**.

> i The **Mappings** left-nav link appears only when you have started LogStream with global ⚙ **Settings** (lower left) > **Distributed Settings** > **Mode** set to **Leader**.

On the resulting **Manage Mapping Rulesets** page, click your new ruleset's **Configure** button, and start adding rules by clicking on **+ Rule**. While you build and refine rules, the Preview in the right pane will show which currently reporting and tracked workers map to which Worker Groups.

A ruleset must be activated before it can be used by the Leader. To activate it, go to **Mappings** and click **Activate** on the required ruleset. The **Activate** button will then change to an **Active** toggle. Using the adjacent buttons, you can also **Configure** or **Delete** a ruleset, or **Clone** a ruleset if you'd like to work on it offline, test different filters, etc.

Although not required, Workers can be configured to send a Group with their payload. See below how this ranks in mapping priority.

Add a Mapping Rule – Example

Within a Mapping Ruleset, click **+ Add Rule** to define a new rule. Assume that you want to define a rule for all hosts that satisfy this set of conditions:

- IP address starts with `10.10.42` , AND:
- More than 6 CPUs OR `CRIBL_HOME` environment variable contains `w0` , AND:
- Belongs to `Group420` .

Rule Configuration

- **Rule Name**: `myFirstRule`
- **Filter**: `(conn_ip.startsWith('10.10.42.') && cpus > 6) || env.CRIBL_HOME.match('w0')`
- **Group**: `Group420`

Default Worker Group and Mapping

When a LogStream instance runs as Leader, the following are created automatically:

- A `default` Worker Group.
- A `default` Mapping Ruleset,
  - with a `default` Rule matching all (`true`).

Mapping Order of Priority

Priority for mapping to a group is as follows: Mapping Rules > Group sent by Worker > `default` Group.

- If a Filter matches, use that Group.
- Else, if a Worker has a Group defined, use that.

- Else, map to the `default` Group.

Deploying Configurations

Your typical workflow for deploying LogStream configurations is the following:

1. Work on configs.
2. Save your changes.
3. Commit (and optionally push).
4. Deploy.

Deployment is the last step after configuration changes have been saved and committed. **Deploying here means propagating updated configs to Workers.** You deploy new configurations at the Group level: Locate your desired Group and click on **Deploy**. Workers that belong to the group will start **pulling** updated configurations on their next check-in with the Leader.

> ⚠ Can't Log into the the Worker Node as Admin User?
>
> When a Worker Node pulls its first configs, 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 User Authentication.

Configuration Files

On the Leader, a Worker Group's configuration lives under: `$CRIBL_HOME/groups/<groupName>/local/cribl/` .

On the managed Worker, after configs have been pulled, they're extracted under: `$CRIBL_HOME/local/cribl/` .

Lookup Files

On the Leader, a Group's lookup files live under: `$CRIBL_HOME/groups/<groupName>/data/lookups` .

On the managed Worker, after configs have been pulled, lookups are extracted under: `$CRIBL_HOME/data/lookups` . When deployed via the Leader, lookup files are distributed to Workers as part of a configuration deployment.

If you want your lookup files to be part of the LogStream configuration's version control process, we recommended deploying using the Leader Node. Otherwise, you can update your lookup file out-of-band on the individual Workers. The latter is especially useful for larger lookup files (> 10 MB, for example), or for lookup files maintained using some other mechanism, or for lookup files that are updated frequently.

For other options, see Managing Large Lookups.

> i Some configuration changes will require restarts, while many others require only reloads. See here for details.
>
> Restarts/reloads of each Worker Process are handled automatically by the Worker. Note that individual Worker Nodes might temporarily disappear from the Leader's **Workers** tab while restarting.

Worker Process Rolling Restart

During a restart, to minimize ingestion disruption and increase availability of network ports, Worker Processes on a Worker Node are restarted in a rolling fashion. **20% of running processes – with a minimum of one process – are restarted at a time.** A Worker Process must come up and report as **started** before the next one is restarted. This rolling restart continues until all processes have restarted. If a Worker Process fails to restart, configurations will be rolled back.

Auto-Scaling Workers and Load-Balancing Incoming Data

If data flows in via Load Balancers, make sure to register all instances. Each Cribl LogStream 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"}`

Environment Variables

- `CRIBL_DIST_MASTER_URL` – URL of the Master Node. Format: `<tls|tcp>://<authToken>@host:port?group=defaultGroup&tag=tag1&tag=tag2&tls.<tls-settings below>`.
    - `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`.
- `CRIBL_DIST_MODE – worker | master`. Defaults to `worker` **iff** `CRIBL_DIST_MASTER_URL` is present.
- `CRIBL_HOME` – Auto setup on startup. Defaults to parent of `bin` directory.
- `CRIBL_CONF_DIR` – Auto setup on startup. Defaults to parent of `bin` directory.
- `CRIBL_NOAUTH` – Disables authentication. Careful here!!
- `CRIBL_VOLUME_DIR` – Sets a directory that persists modified data between different containers or ephemeral instances.

Deprecated Variables

These will be removed as of LogStream 3.0:

- `CRIBL_CONFIG_LOCATION`.
- `CRIBL_SCRIPTS_LOCATION`.

Workers GUID

When you install and first run the software, a GUID is generated and stored in a `.dat` file located in `CRIBL_HOME/bin/`, e.g.:

```
# cat CRIBL_HOME/bin/676f6174733432.dat
{"it":1570724418,"phf":0,"guid":"48f7b21a-0c03-45e0-a699-01e0b7a1e061"}
```

When deploying Cribl LogStream 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 be regenerated on next run.

Splunk App Deployment *

Getting started with Cribl App for Splunk

> ⚠ Cribl App for Splunk for HFs Is Deprecated as of Cribl LogStream v.2.1
>
> Cribl will continue to support this package, but **customers are advised to begin planning now for the eventual removal of support**.
>
> See Single-Instance Deployment and Distributed Deployment for alternatives.

Deploying Cribl App for Splunk

In a Splunk environment, Cribl LogStream can be installed and configured as a Splunk app (Cribl App for Splunk). Depending on your requirements and architecture, it can run either on a Search Head or on a Heavy Forwarder. Cribl App for Splunk **cannot** be used in a Cribl LogStream Distributed Deployment as a Leader or managed as Worker.

Running on a Search Head (SH)

When running on an SH, Cribl LogStream 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 LogStream instance's TCP JSON input on port `10420`, but it's also capable of sending to any other LogStream 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

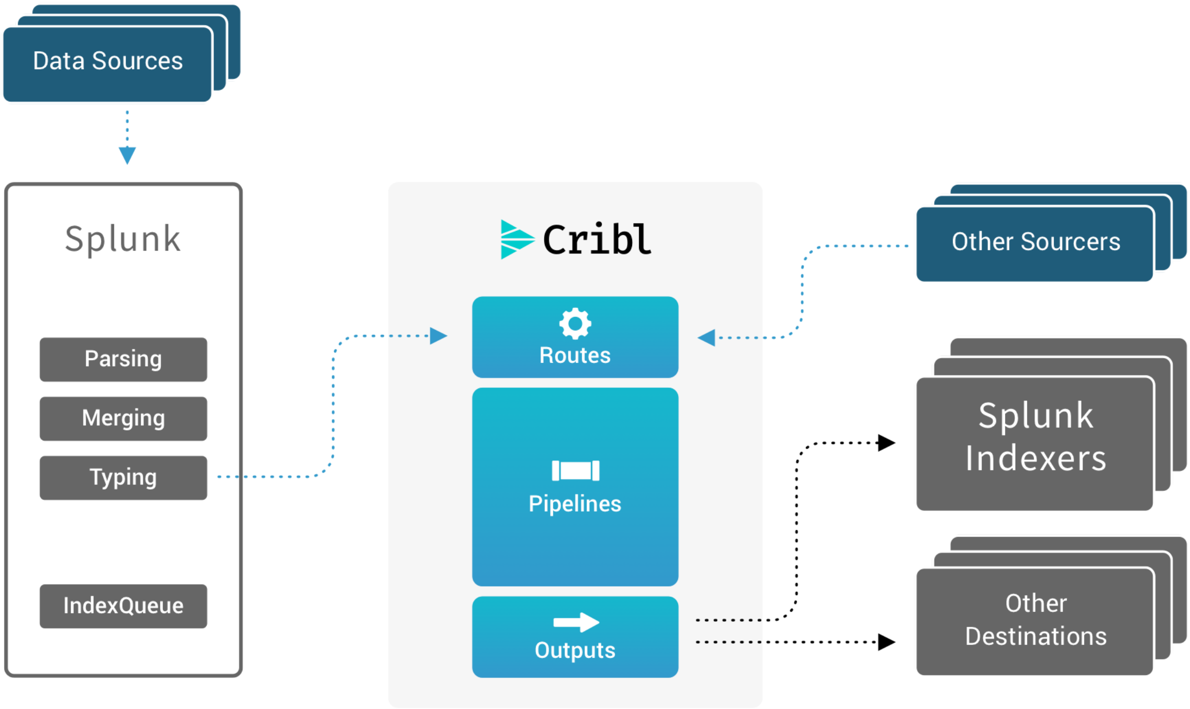
- Select an instance on which to install.
- Ensure that ports `10000`, `10420`, and `9000` are available. See the Requirements section for more info.
- Get the bits here, and install as a regular Splunk app.
- Restart the Splunk instance.
- Go to `https://<instance>/en-US/app/cribl` or `https://<instance>:9000`, and log in with Splunk **admin** role credentials.

Typical Use Cases for Search Head Mode

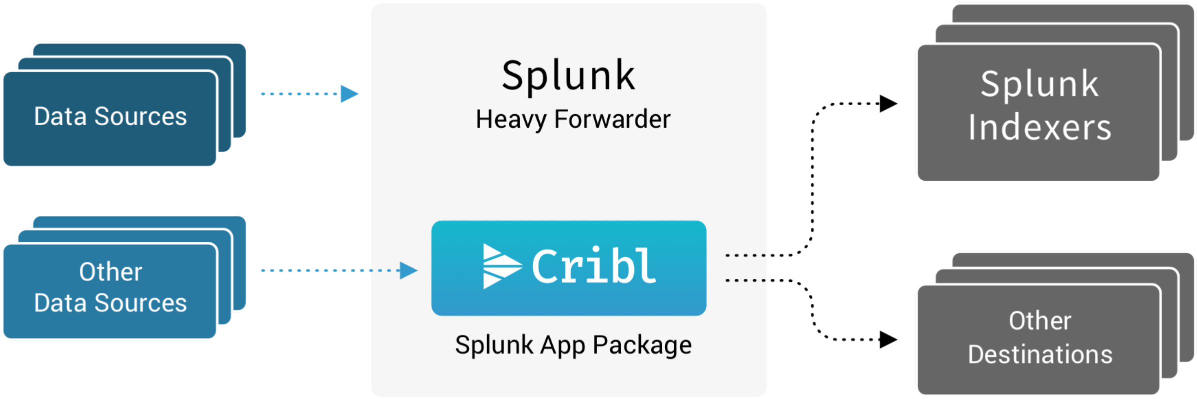
- Working with search results in a Cribl LogStream pipeline.
- Sending search results to any Destination supported by Cribl LogStream.

Running on a Heavy Forwarder (HF)

When running on an HF, Cribl LogStream is set to **mode-hwf**. It receives events from the local Splunk process per routing configurations in `props.conf` and `transforms.conf`. Data is parsed and processed first by Splunk pipelines, and then by LogStream. By default, all data except internal indexes is routed out right after the Typing pipeline.



Cribl LogStream is capable of accepting data **streams** (unbroken events) or **events** from other sources. In this case, the HF will deliver **events** locally to LogStream, which processes them and sends them to one or more destinations downstream. When receivers are Splunk indexers, LogStream can also load-balance across them.



Installing the Cribl App for Splunk on an HF

- Select an instance on which to install.
- Ensure that ports `10000` , `10420` , and `9000` are available. See here.
- Get the bits here, and install as a regular Splunk app.
- Set Cribl to **mode-hwf**: `$SPLUNK_HOME/etc/apps/cribl/bin/cribl mode-hwf` .

> ⚠ The `SPLUNK_HOME` environment variable must be defined.

- Restart the Splunk instance.
- Go to `https://<instance>:9000` and log in with Splunk **admin** role credentials.

> ### Note About Splunk Warnings
>
> If you come across messages similar to the following example, on startup or in logs, please ignore them. They are benign warnings.
>
> `Invalid value in stanza [route2criblQueue]/[hecCriblQueue] in /opt/splunk/etc/apps/cribl/default/transforms.conf, line 11: (key: DEST_KEY, value: criblQueue) / line 24: (key: DEST_KEY, value: $1)`

Relevant configurations in Cribl App for Splunk on an HF

When Cribl App for Splunk is installed on an HF (in `mode-hwf`), below are the **relevant sections** in configuration files that enable Splunk to send data to Cribl LogStream:

apps/cribl/default/outputs.conf

```
[tcpout]
disabled = false
defaultGroup = cribl

[tcpout:cribl]
server=127.0.0.1:10000
sendCookedData=true
useACK = false
negotiateNewProtocol = false
negotiateProtocolLevel = 0
```

apps/cribl/default/inputs.conf

```
[splunktcp]
route=has_key:_replicationBucketUUID:replicationQueue;has_key:_dstrx:typingQueue;has_key:__CRIB
```

apps/cribl/default/transforms.conf

```
[route2cribl]
SOURCE_KEY = _MetaData:Index
```

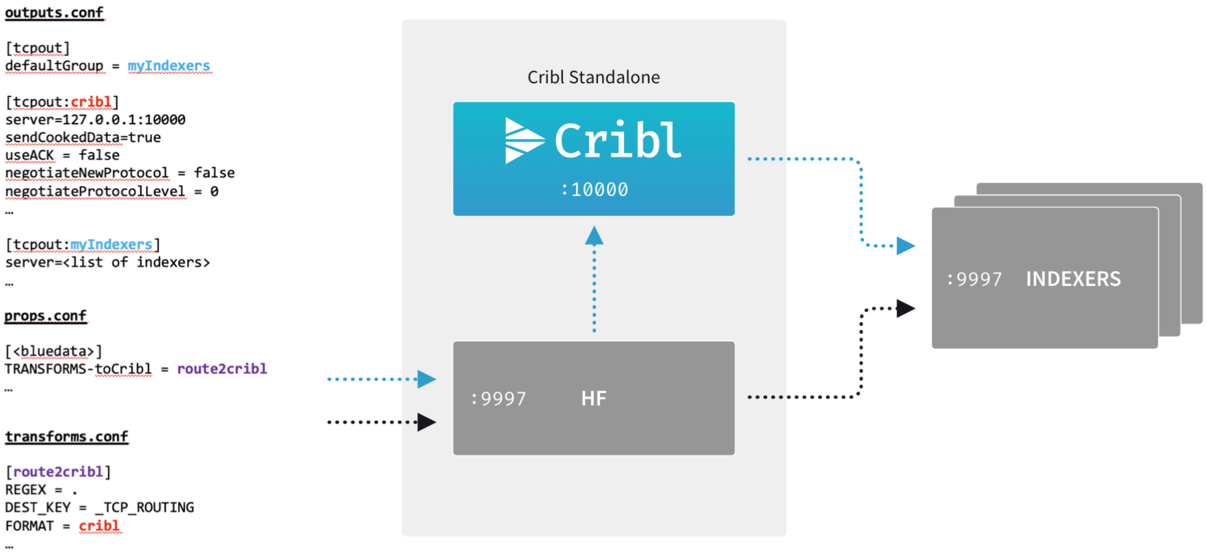
```
REGEX = ^[^_]
DEST_KEY = _TCP_ROUTING
FORMAT = cribl

[route2criblQueue]
SOURCE_KEY = _MetaData:Index
REGEX = ^[^_]
DEST_KEY = queue
FORMAT = criblQueue
```

apps/cribl/default/props.conf

```
[default]
TRANSFORMS-cribl = route2criblQueue, route2cribl
```

Configuring Cribl LogStream with a Subset of Your Data

The `props.conf` stanza above will apply the above transforms to **everything**. Depending on your requirements, you might want to target only a subset of your sources, sourcetypes, or hosts. For example, the diagram below shows the **effective** configurations of `outputs.conf`, `props.conf`, and `transforms.conf` to send `<bluedata>` events through Cribl LogStream.



outputs.conf

```
[tcpout]
defaultGroup = myIndexers

[tcpout:cribl]
server=127.0.0.1:10000
sendCookedData=true
useACK = false
negotiateNewProtocol = false
negotiateProtocolLevel = 0
…

[tcpout:myIndexers]
server=<list of indexers>
…
```

props.conf

```
[<bluedata>]
TRANSFORMS-toCribl = route2cribl
…
```

transforms.conf

```
[route2cribl]
REGEX = .
DEST_KEY = _TCP_ROUTING
FORMAT = cribl
…
```

Configure Cribl LogStream to Send Data to Splunk Indexers

To send data from Cribl LogStream to a set of Splunk indexers, use the LogStream UI to go to **Destinations** > Splunk Load Balanced, then enter the required information.

Bootstrap Workers from Leader

Boot fully provisioned workers

ThisLogStream feature allows workers to completely provision themselves on initial boot, directly from the Leader. It allows a fleet of any number of nodes to launch, and to be fully functional within the cluster, in seconds.

How Does It Work?

A LogStream Leader Node (v2.2 or higher) 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 LogStream installed, fully provisioning the machine as a Worker Node.

Although you can specify the download URL when you execute the initial curl command, the LogStream package is not downloaded until the script is generated by the API, and then later executed.

> ⚠ Root Access or sudo
>
> Note that the script will install LogStream 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` command when executing the script.

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 `criblmaster` . You can find this secret in the the Leader Node's **Distributed Settings** section.
`group`	optional	Name of the cluster's work group. If not specified, falls back to `default` .

`download_url`	optional	Provide the complete URL to a Cribl LogStream installation binary. This is especially useful if the Worker Nodes don’t have access to the Internet to download from cribl.io.

Example HTTP Request

HTTP

```
GET http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-beef-4c6e-554
```

> As of version 3.0, LogStream'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

Shell

```
#!/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 ${CRIBL_VERSION} | cut -d '-' -f 1)/${FIL
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 LogStream User" --disabled-password
fi
if  [ $CENTOS -eq 1 ] || [ $AMAZON -eq 1 ]; then
    adduser cribl -d /home/cribl -c "Cribl LogStream User" -m
    usermod -aG wheel cribl
fi

echo "Installing LogStream"
```

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:

Shell

```
curl http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-beef-4c6e-5
```

Chaining 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 the command below. This passes the script's contents to the `sh` shell to immediately execute. As noted above, on Ubuntu and similar systems, you might need to insert `sudo` before the `sh`.

Shell

```
curl http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-beef-4c6e-5
```

Adding Download URL

We'll now graduate to the next level by adding more to the above commands. All the preceding commands excluded the `download_url` parameter so, by default, the script gets configured to download the LogStream package from the public Cribl repository.

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. Quoting the URL, as shown in this example, prevents this.

Shell

```
curl "http://<leader hostname or IP>:9000/init/install-worker.sh?token=79364d6e-dead-beef-4c6e-
```

Status Codes

Status Code	Reason
200 – OK	All is well. You should have received the script as a response.
403 – Forbidden	Either the node is not configured as a Leader, or the token provided is invalid.

Restrictions

Keep the following in mind when using this feature:

- Each Worker must normally have access to the internet in order to download the Cribl LogStream installation binary from cribl.io. Where this isn't feasible, you can use the `download_url` switch to point to a binary in a restricted location.

- By default, Worker Nodes communicate with the Leader on port 4200. Ensure that access between all Workers and the Leader is open on this port.

- TLS is not enabled by default. If enabled and configured, access to this feature will be over `https` instead of `http`.

- Red Hat, Ubuntu, CentOS, and Amazon Linux are the only supported Worker platforms.

User Data

For public-cloud customers, an easy way to use this feature is in an instance's user data. First, be sure to set the Leader Node to mode = 'leader'. Then use the following script (changing the command as needed. based on the information above). Upon launch, the Worker Node will reach out to the Leader, download the script, download the LogStream package from the specified location, and then install and configure LogStream:

Shell

```
#!/bin/bash
curl http://<leader-node-ip/host-address>:9000/init/install-worker.sh?token=<auth-token> | sh -
```

Kubernetes/Helm Deployment

Cribl's master and workergroup Helm charts provide a fast way to deploy a distributed LogStream environment to a Kubernetes cluster.

> ℹ As of version 3.0, LogStream's former "master" application components are renamed "leader." While some legacy terminology remains within the Helm charts, this document will match the legacy naming, to avoid ambiguity.

Prerequisites

Helm version 3 is required to use these charts.

To install Helm on (e.g.) a Mac, using Homebrew:

```
brew install helm
```

Find instructions for other operation systems in Helm's installation documentation.

Deploying

If you haven't done so already, create a namespace. Our documentation example uses `logstream`.

```
kubectl create namespace logstream
```

Add the Cribl Helm repo.

```
helm repo add cribl https://criblio.github.io/helm-charts/
```

The following example creates a distributed deployment with two autoscaled worker groups, `pcilogs` and `system-metrics`. It uses an auth token of `ABCDEF01-1234-5678-ABCD-ABCDEF012345`, sets an admin password, and installs our license:

```
helm install ls-master cribl/logstream-master \
  --set "config.groups={pcilogs,system-metrics}" \
  --set config.token="ABCDEF01-1234-5678-ABCD-ABCDEF012345" \
  --set config.adminPassword="<admin password>" \
  --set config.license="<license key>" \
  -n logstream

helm install ls-wg-pci cribl/logstream-workergroup \
  --set config.host="ls-master-internal" \
  --set config.tag="pcilogs" \
  --set config.token="ABCDEF01-1234-5678-ABCD-ABCDEF012345" \
  -n logstream

helm install ls-wg-system-metrics cribl/logstream-workergroup \
  --set config.host="ls-master-internal" \
  --set config.tag="system-metrics" \
  --set config.token="ABCDEF01-1234-5678-ABCD-ABCDEF012345" \
  -n logstream
```

Running Distributed on a Free License

If you do not specify a license in your install with `config.license`, and you want to run distributed, you'll need to go to LogStream's **Settings > Licensing** UI page and accept the Free license. (The Free license allows one worker group.) If your Helm configuration includes the `config.groups` option, the LogStream Master Node will be configured as a distributed Master. If you omit that option, it will be configured as a single instance. (You can later use LogStream's **Settings > Distributed** page to select **Mode:** `Master`.)

Upgrading

Upgrading LogStream to new bits via Helm is easy. Sync up your repo to the origin, and then upgrade each chart version. The example below updates to the current version, but you can append `--version X.Y.Z` if you want to specify a particular version.

```
helm repo update
helm upgrade ls-master cribl/logstream-master -n logstream
helm upgrade ls-wg-pci cribl/logstream-workergroup -n logstream
helm upgrade ls-wg-system-metrics cribl/logstream-workergroup -n logstream
```

What's Next

- Kubernetes Leader Deployment
- Kubernetes Worker Deployment

K8s Leader Deployment

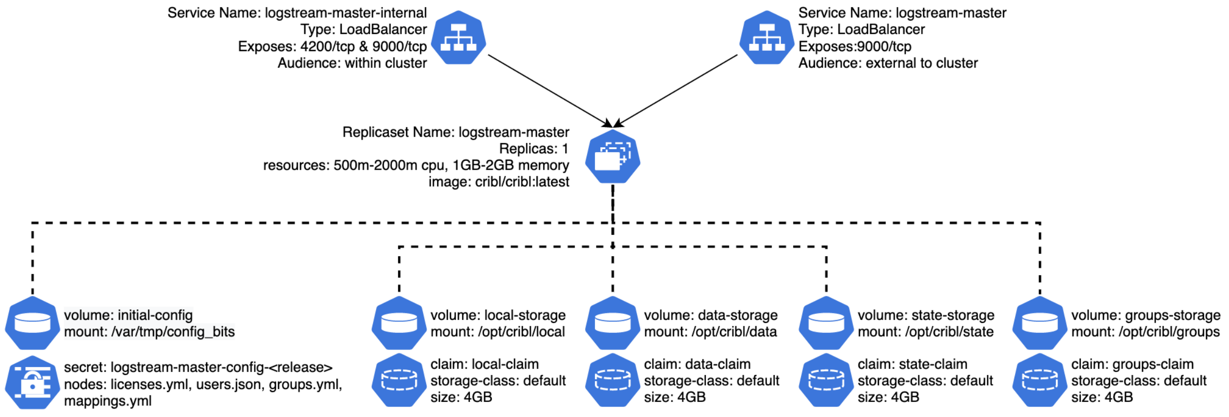
Boot a fully provisioned Leader Node via Helm

This page outlines how to deploy a Cribl LogStream Leader Node (or single instance) to AWS via Kubernetes, using a Cribl-provided Helm chart.

> ⚠ This chart is a work in progress, provided as-is. Cribl expects to further develop and refine it.
>
> As of version 3.0, LogStream's former "master" application components are renamed "leader." While some legacy terminology remains within the Helm chart, this document will match the legacy naming, to avoid ambiguity.

Deployment

As built, Cribl's chart will deploy a Master Server for LogStream, 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-master communication.

> By default, this chart installs only a LogStream Leader Node. To also deploy LogStream Worker Groups 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 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>
```

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
[ ... ]
```

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>
```

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-master/values.yaml
2. Save this as a local file, e.g.: `/bar/values.yaml`
3. Modify values as necessary (see Values to Override below).
4. Install your updated values to Helm, using this command:
   `helm install -f /bar/values.yaml`

Values to Override

This section covers the most likely values to override. To see the full scope of values available, run: `helm show values cribl/logstream-master`

Key	Type	Default Value	Description
`config.adminPassword`	String	[No default]	The password you want to assign to the admin user.
`config.token`	String	[No default]	The auth key you want to set up for Worker access. If you set this value, the LogStream instance will be configured only as a Leader server for a distributed deployment. (You can also configure this later via the LogStream UI, after launching the instance in single-instance mode.)

`config.license`	String	[No default]	The license for your LogStream instance. If you do not set this, it will default to the Free license. You can change this in the LogStream UI as well.
`config.groups`	List	[No default]	Array of Worker Group names to configure for the Leader instance. This will create a mapping for each Group, which looks for the tag `<groupname>` , and will create the basic structure of each Group's configuration.
`config.scName`	String	<default StorageClass name>	The StorageClass name for all of the persistent volumes.
`config.rejectSelfSignedCerts`	Number	`0`	Either `0` (allow self-signed certificates) or `1` (deny self-signed certs).
`config.healthPort`	Number	`9000`	The port to use for health checks (readiness/live).
`config.healthScheme`	String	`HTTP`	The scheme to use for health checks. Supports `HTTP` or `HTTPS` .
`service.internalType`	ClusterIP	[No default]	The type to use for the `<release>-master-internal` service. In 2.4.5+, this is set to `ClusterIP` by default. If you have any Worker Groups outside of the Kubernetes cluster where the Leader lives, you'll need to change this to `NodePort` or `LoadBalancer` to expose it outside of the cluster.
`service.externalType`	Load Balancer	[No default]	The type to use for the user-facing `<release>-master` service. If `ingress.enable` is set, this will be force-set to `NodePort` , to work with the ingress.

`service.ports`	Array of Maps	`- name: api`
		`  port: 9000`
		`  protocol: TCP`
		`  external: true`
		`- name: mastercomm`
		`  port: 4200`
		`  protocol: TCP`
		`  external: false`
			The ports to make available, both in the deployment and in the service. Each "map" in this list needs the following values set:
			name
			A descriptive name, identifying what the port is being used for.
			port
			The container port to be made available.
			protocol
			The protocol in use for this port (UDP or TCP).

			external Set to `true` to `expose` the port on the external service, or `false` to not expose it.
`service.annotations`	String	[No default]	Annotations for the the service component – this is where you'll want to put load-balancer–specific configuration directives.
`criblImage.tag`	String	`latest`	The container image tag to pull from. Cribl will increment this tag per LogStream version. By default, this will use a version equivalent to the chart's `appVersion` value. You can override this with `latest` to get the latest LogStream version, or with a specific LogStream version number (like "2.3.3").
consolidate_volumes	boolean	[No default]	If this value exists, and the `helm` command is `upgrade` , this will use the split volumes that we created in charts before 2.4 and consolidate them down to one config volume. This is a **one-time** event.

Extra Configuration Options

The links here point to configuration details on our GitHub repo.

Key	Type	Default Value	Description
extraVolumeMounts	Object	[No default]	Additional volumes to mount in the container.
extraSecretMounts	Array	[No default]	Pre-existing Secrets to mount within the container.
extraConfigmapMounts	Object	[No default]	Pre-existing ConfigMaps to mount within the container.
extraInitContainers	Object	[No default]	Additional containers to run ahead of the primary container in the pod.
securityContext.runAsUser	Number	0	User ID to run the container processes under.
securityContext.runAsGroup	Number	0	Group ID to run the container processes under.
envValueFrom	Object	[No default]	Environment variables to be exposed from the Downward API.
env	Array	[No	Additional static environment variables.

		default]	
ingress.enable	boolean	false	Enable Ingress in front of the external service. Setting this to `true` changes the external service to type `NodePort` , and creates an ingress that connects to it.
ingress.annotations	Object	[No default]	If `ingress.enable` is set to `true` , this is where annotations to configure the specific ingress controller. (NOTE: Ingress is supported only on Kubernetes 1.19 and later clusters).

Match Versions

Cribl recommends that you use the same LogStream version on Worker 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 name>"
    service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix: "ELB"
```

For an exhaustive list of annotations you can use with AWS's Elastic Load Balancers, see the Kubernetes Service documentation.

> ⚠ More options are coming here!

Basic Chart Installation

With the above prerequisites and configuration completed, you're ready to install our chart to deploy a LogStream Leader Node. Here are some example commands:

- To install the chart with the release name `logstream-master` :

```
helm install logstream-master cribl/logstream-master
```

- To install the chart using the storage class `ebs-sc` :

  ```
  helm install logstream-master cribl/logstream-master --set config.scName='lebs-
  sc
  ```

Post-Install/Post-Upgrade

LogStream will not automatically deploy changes to the Worker Nodes. You'll need to commit and deploy changes to all of your Worker Groups.

Change the Configuration

If you don't override its default values, this Helm chart effectively creates a single-instance deployment of LogStream, using the standard container image. You can later configure distributed mode, licensing, user passwords, etc., all from the LogStream 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-master cribl/logstream-master --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 LogStream's global ⚙ **Settings** (lower left) > **Licensing** UI page and accept the Free license. (The Free license allows one Worker Group.)

If your Helm configuration includes the `config.groups` option, the LogStream 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 LogStream's global ⚙ **Settings** (lower left) > **Distributed** page to select **Mode:** `Leader` .)

Set the Admin Password

Normally, when you first install LogStream 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-master cribl/logstream-master --set config.adminPassword="
<new password>"
```

Set Up Worker Groups/Mappings

As mentioned above, the chart's default is to install a vanilla deployment of LogStream. If you are deploying as a Leader, you can use the `config.groups` parameter to define the Worker Groups you want created and mapped. Each group in the list you provide will be created as a Worker Group, with a Mapping Rule to seek a tag with that Worker Group's name in it:

```
helm install logstream-master cribl/logstream-master --set config.groups={group1,group2,group3}
```

The example above will create three Worker Groups – `group1` , `group2` , and `group3` – and a Mapping Rule for each.

Persistent Volumes

LogStream 2.4.0 and above support the `$CRIBL_VOLUME_DIR` environment variable, which simplifies the persistent-storage requirement for `logstream-master` .

Instead of maintaining multiple volumes (one each for `$CRIBL_HOME/{.git, data, state, local, groups, log}`), you can persist data using a single volume. `$CRIBL_VOLUME_DIR` instructs LogStream where to place these persistent directories, by overriding `$CRIBL_HOME` .

Using Persistent Volumes

To use this feature, pass the `$CRIBL_VOLUME_DIR` variable to your container's environment. Make sure it points to the same value as the volume's mount point.

To start out simple, here is a minimal working example using Docker:

[Docker CLI example]

```
docker volume create example-volume
docker run -e CRIBL_VOLUME_DIR=/mount/point -v example-volume:/mount/point cribl/cribl
```

For Kubernetes, as shown in the example below, you’ll need to create:

- A persistent volume claim.
- An environment variable definition.
- A volume mount definition.
- A volume definition.

[Helm manifest excerpt]

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: config-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: logstream-master
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: cribl-master
        image: cribl/cribl
        env:
        - name: CRIBL_VOLUME_DIR
          value: /mount/point
        volumeMounts:
        - name: example-volume
          mountPath: /mount/point
  volumes:
    - name: config-storage
      persistentVolumeClaim:
        claimName: config-claim
```

Notes on Persistent Volumes

- See your Kubernetes service's documentation for appropriate details on setting up persistent storage and default storage classes.
- If you use `$CRIBL_VOLUME_DIR`, you **must** set it on the Leader Node or single instance.
- Also set `$CRIBL_VOLUME_DIR` on Worker Nodes if you are using Persistent Queue on any Destination. Doing so will retain data if the container is shut down while data is still queued.
- Setting this variable for **existing** instances will leave existing data intact in the original directories. (Note that data loss is still possible if you don't persist the data by other means.)
- For more usage details, see the CLI Reference.

Upgrading Pre-2.4.0 Versions to Use Persistent Volumes

To enable a persistent volume using the `$CRIBL_VOLUME_DIR` environment variable, you'll need to upgrade any pre-2.4.0 version of LogStream. In the Helm chart, we handle this via the `helm upgrade` command.

If you are upgrading from a pre-2.4 version of the chart, you'll want to set the `consolidate_volumes` value, which will create a new, larger volume, and consolidate the data from the original volumes into that volume. An `initContainer` handles the logistics. When this process completes, the `logstream-master` pod will come back up with a single consolidated volume.

> Back Up Your Data First

> While we've tested this upgrade repeatedly, differences in environments can always cause problems. Therefore, we recommend that you back up your data before running the upgrade command. This is best done with a combination of `kubectl` and `tar` :
>
> ```
> kubectl exec <pod name> -n <namespace> -- bash -c "cd /opt/cribl; tar cf - {state,data
> ```
>
> This command executes the `tar` -based backup of all four volumes, and outputs it to a local `.tar` file (`cribl_backup.tar`).

Running the Upgrade

Helm makes upgrades easy. You simply need to run `helm repo update` to ensure you have the latest repo updates available, followed by `helm upgrade` to actually upgrade the containers.

For example, if you've installed the Helm charts in the `logstream` namespace, named your release `ls-master` , and set up your Helm repo according to the prerequisites section above (i.e., named it `cribl`), run the following:

```
helm repo update
helm upgrade ls-master --set consolidate_volumes=true -n logstream cribl/logstream-master
```

Upgrade Order of Operations

While there should be no major problems running a 2.4.0 master and 2.3.4 workers, Cribl does not recommend this. Instead, upgrade the master Helm chart to 2.4.0 first, and then upgrade the workers. (For details, see Kubernetes Worker Deployment.)

Idempotency of Upgrade

The upgrade operation performs a potentially destructive action in coalescing the 4 volumes to a single volume. But that operation proceeds only if the single volume has no data on it. Once the upgrade is performed the first time, any further upgrade operations will effectively skip that coalescence operation, without causing any additional issues.

Recovering from a Failed Upgrade

If the upgrade fails, the suggested recovery path is to remov the Helm chart, reinstall it, and then run this command to restore the data from the backup:

```
cat cribl_backup.tar| kubectl -n <namespace> exec --stdin <pod name> -- bash -c "cd /opt/cribl/c
```

This will restore the data into the "new" volume (which is mounted as `/opt/cribl/config-volume`). If you want to double-check that:

```
kubectl -n <namespace> exec <pod name> -- bash -c "ls -alR /opt/cribl/config-volume"
```

Preloading Configuration

The `extraConfigmapMounts` and `extraSecretMounts` options enable you to preload configuration files into the master 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.

> ℹ 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 configuration lives under the `groups/` subdirectory. If you have a worker group named `datacenter_a`, its configuration will live in `/opt/cribl/config-volume/groups/datacenter_a`. See Configuration Files section for details on file locations.

Using 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 LogStream startup (`CRIBL_BEFORE_START_CMD_[1-30]`). It also looks for up to 30 environment variables to execute after LogStream 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.

Figuring 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 (i.e., 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 ../master-values.yaml -n logstream-ht cribl/logstream-master
```

You can now get the pod's name:

```
% kubectl get pods -n logstream-ht
NAME                                               READY   STATUS    RESTARTS   AGE
lsms-master-659bfccdd6-xsz67                       1/1     Running   0          52m
```

And then you can use `kubectl describe` to get the relevant environment variables:

```
% kubectl describe  pod/lsms-master-659bfccdd6-xsz67 -n logstream-ht  | egrep "CRIBL_.*START"
CRIBL_BEFORE_START_CMD_1:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/licenses.yml ]; then mkdi
CRIBL_BEFORE_START_CMD_2:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/mappings.yml ]; then mkdi
CRIBL_AFTER_START_CMD_1:       [ ! -f $CRIBL_VOLUME_DIR/users_imported ] && sleep 20 && cp /var/
```

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`.

Preloading 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}/${_time:%Y}/${_time:%
      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
```

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 master 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
```

This example will mount the files in the ConfigMap into the pod's `/var/tmp/job-config` directory.

Copying 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/group1/local
```

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 tree exists
  cp /var/tmp/job-config/jobs.yml /opt/cribl/config-volume/groups/group1/local/cribl # copy the
  touch $FLAG_FILE
fi
```

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 $FLAG_FILE
```

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' `service > ports` configuration. This restriction might be removed in future versions.
- The upgrade process from pre-2.4.0 versions creates an `initContainer`, which will run prior to any instance of the LogStream 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 2.3.* -> 2.4.0. In the next iteration, we'll remove the `initContainer` from the upgrade path.

- The upgrade process leaves the old `PersistentVolume` s and `PersistentVolumeClaim` s around. This is, unfortunately, necessary for this upgrade path. In follow-on versions, we will remove these volumes from the chart.
- See EKS-specific issues on our GitHub repo.

K8s Worker Deployment

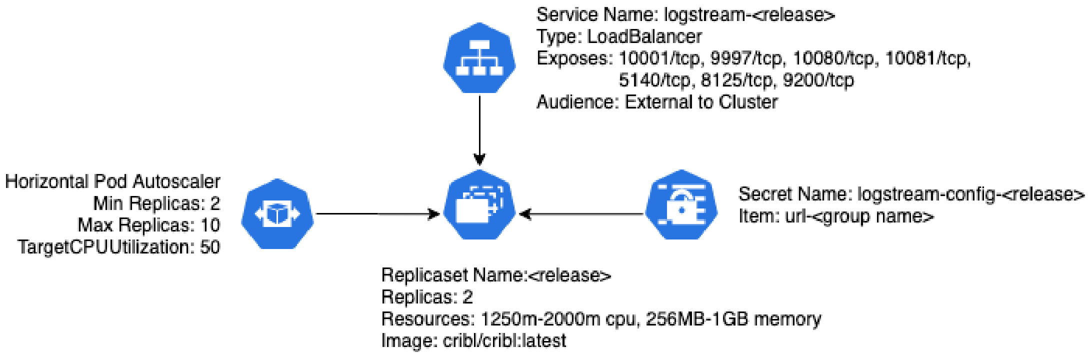
Boot a fully provisioned Worker Group via Helm

This page outlines how to deploy a Cribl LogStream Worker Group to AWS via Kubernetes, using a Cribl-provided Helm chart.

> This chart will deploy only a LogStream Worker Group, which depends on the presence of a LogStream Leader Node. To deploy the Leader, see Kubernetes Leader Deployment.

Deployment

As built, Cribl's chart will deploy a simple Worker Group for LogStream, consisting of a deployment, a service, a horizontal pod autoscaler configuration, and a secret used for configuration.


Service Name: logstream-<release>
Type: LoadBalancer
Exposes: 10001/tcp, 9997/tcp, 10080/tcp, 10081/tcp, 5140/tcp, 8125/tcp, 9200/tcp
Audience: External to Cluster
Horizontal Pod Autoscaler
Min Replicas: 2
Max Replicas: 10
TargetCPUUtilization: 50
Secret Name: logstream-config-<release>
Item: url-<group name>
Replicaset Name:<release>
Replicas: 2
Resources: 1250m-2000m cpu, 256MB-1GB memory
Image: cribl/cribl:latest

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>
```

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
[ … ]
```

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>
```

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/`

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, e.g.: `/foo/values.yaml`

3. Modify values as necessary (see Values to Override below).

4. Install your updated values to Helm, using this command:
   `helm install -f /foo/values.yaml`

Values to Override

This section covers the most likely values to override. To see the full scope of values available, run: `helm show values cribl/logstream-workergroup` .

> i As of version 3.0, LogStream's former "master" application components are renamed "leader." While some legacy terminology remains within the Helm charts, this document will match the legacy naming, to avoid ambiguity.

Key	Type	Default Value	Description
`config.group`	String	`criblmaster`	Tag/group to include in the URL (included as both a group value and a tag value).
`config.tag`	[Deprecated]	[Deprecated]	The tag/group to include in the URL. (This option is deprecated, but is still supported for backward compatibility.
`config.token`	String	`criblmaster`	The authentication token for your

			LogStream Leader.
`config.host`	String	`logstream-master`	The resolvable hostname of your LogStream Leader.
`config.rejectSelfSignedCerts`	Number	`0`	One of: `0` – allow self-signed certs, or `1` – deny self-signed certs.

`service.ports`	Array of Maps	`- name: tcpjson`
		`  port: 10001`
		`  protocol: TCP`
		`- name: s2s`
		`  port: 9997`
		`  protocol: TCP`
		`- name: http`
		`  port: 10080`
		`  protocol: TCP`
		`- name: https`
		`  port: 10081`
		`  protocol: TCP`
		`- name: syslog`
		`  port: 5140`
		`  protocol: TCP`
		`- name: metrics`
		`  port: 8125`
		`  protocol: TCP`
		`- name: elastic`
		`  port: 9200`
		`  protocol: TCP`
			The ports to make available, both in the deployment and in the service. Each "map" in this list needs the following values set:
			name
			A descriptive name, identifying what the port is being used for.
			port
			The container port to make available.
			protocol
			The protocol in use for this port (UDP or TCP).

`service.annotations`	String	[No default]	Annotations for the the service component – this is where you'll want to put load-balancer–specific configuration directives.
`criblImage.tag`	String	`2.4.5`	The container image tag to pull from. Cribl will increment this tag per LogStream version. By default, this will use a version equivalent to the chart's `appVersion` value. You can override this with `latest` to get the latest LogStream version, or with a specific LogStream version number.
`autoscaling.minReplicas`	Number	`2`	The minimum number of LogStream pods to run.
`autoscaling.maxReplicas`	Number	`10`	The maximum number of LogStream pods to scale up to.
`autoscaling.targetCPUUtilizationPercentage`	Number	`50`	The CPU utilization percentage that

			triggers scaling action.
rbac.create	Boolean	`false`	Enable Service Account, Cluster Role, and Role Binding creation.
rbac.resources	List	`["pods"]`	Set the resource boundary for the role being created (K8s resources).
rbac.verbs	List	`["get", "list"]`	Set the API verbs allowed the role (default: `read ops`).

Extra Configuration Options

The links here point to configuration details on our GitHub repo.

Key	Type	Default Value	Description
extraVolumeMounts	Object	[No default]	Additional volumes to mount in the container.
extraSecretMounts	Array	[No default]	Pre-existing Secrets to mount within the container.
extraConfigmapMounts	Object	[No default]	Pre-existing ConfigMaps to mount within the container.
extraInitContainers	Object	[No default]	Additional containers to run ahead of the primary container in the pod.
securityContext.runAsUser	Number	0	User ID to run the container processes under.
securityContext.runAsGroup	Number	0	Group ID to run the

			container processes under.
envValueFrom	Object	[No default]	Environment variables to be exposed from the Downward API.
env	Array	[No default]	Additional static environment variables.
deployment	String	`deployment`	One of: `deployment` to deploy as a Deployment Set; or `daemonset` to deploy as a DaemonSet.
rbac.extraRules	Object	[No default]	Additional RBAC rules to put in place.

Match Versions

Cribl recommends that you use the same LogStream version on Leader Nodes versus Worker Group 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.

Install the Chart

With the above prerequisites and configuration completed, you're ready to install our chart to deploy a LogStream Worker Group. 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 LogStream 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 LogStream Leadermast `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
  ```

Upgrading

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
```

After this step, invoke:

```
helm upgrade <release> -n <namespace> cribl/logstream-workergroup
```

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
```

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
```

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 LogStream Leader, and commit and deploy it to your Worker Group.

Using Persistent Storage for Persistent Queueing

The `extraVolumeMounts` option makes it feasible to use persistent volumes for LogStream 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. However, if you choose to implement persistent volumes for queueing, please consider these suggestions:

1. Use a shared-storage-volume mechanism. We've worked with the EFS CSI driver for AWS, and it works fairly well (although it can be tedious to configure).

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 pods for volume issues. The faster you can see such issues and react, the more likely that you'll be able to resolve thema.

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 LogStream Leader, and then commit and deploy it to your Worker Group(s).

Known Issues

- The chart currently supports **only** TCP ports in `service > ports` for Worker Groups. This limitation might be removed in future versions.
- See EKS-specific issues on our GitHub repo.

Docker Deployment

Use this `docker-compose.yml` to stand up a LogStream distributed deployment:

docker-compose.yml

```
version: '3.5'
services:
  master:
    image: ${CRIBL_IMAGE:-cribl/cribl:latest}
    environment:
      - CRIBL_DIST_MODE=master
      - CRIBL_DIST_MASTER_URL=tcp://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_IMAGE:-cribl/cribl:latest}
    depends_on:
      - master
    environment:
      - CRIBL_DIST_MODE=worker
      - CRIBL_DIST_MASTER_URL=tcp://criblmaster@master:4200
    ports:
      - "29000-29200:9000"
```

This uses a local directory, `~/cribl-config`, as the configuration store for LogStream. This directory must exist before you run the `docker-compose` command.

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.

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
```

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:29000-29200

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

Sizing and Scaling

A Cribl LogStream installation can be scaled **up** within a single instance and/or scaled **out** across multiple instances. Scaling allows for:

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

Scale Up

A LogStream installation can be configured to scale up and utilize as many resources on the host as required. In a single-instance deployment, you govern resource allocation through the global ⚙ **Settings** (lower left) > **System** > **Worker Processes** section.

In a distributed deployment, you allocate resources per Worker Group. Navigate to **Groups >** `group-name` **> Settings > Worker Processes**.

Either way, these controls are available:

- **Process count**: Indicates the number of Worker Processes to spawn. Positive numbers specify an absolute number of Workers. Negative numbers specify the number of Workers relative to the number of CPUs in the system. like this: { `number of CPUs available` minus `this setting` }. The default is `-2` . A `0` setting is interpreted as `1` Worker Process. (LogStream corrects for excessive negative offsets by guaranteeing at least `1` Process.)

- **Memory (MB):** Amount of memory available to each Worker Process, in MB. Defaults to `2048` . (See Estimating Memory Requirements below.)

> ℹ Throughout these guidelines, we assume that 1 physical core is equivalent to 2 virtual/hyperthreaded CPUs (vCPUs). Each LogStream instance requires the following resources to run:
>
> - +4 physical cores, +8GB RAM
> - 5GB free disk space (more if persistent queuing is enabled)

For example, assuming a Cribl LogStream system with 6 physical cores (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).

See Capacity and Performance Considerations below for CPU utilization.

> ℹ LogStream incorporates guardrails that prevent spawning more processes than available vCPUs.

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.**

   E.g., it's better to have 5 connections to TCP 514, each bringing in 200GB/day, than one at 1TB/day.

2. Each Worker Process will maintain and manage its own outputs. E.g., if an instance with 2 Worker Processes is configured with a Splunk output, then the Splunk destination will see 2 inbound connections.

Capacity and Performance Considerations

As with most data processing applications, Cribl LogStream's expected resource utilization will be commensurate with 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 finding and replacing a string. At the time of this writing:

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

Estimating Core Requirements

Current guidance for capacity planning is: **Allocate 1 physical core for each 400GB/day of IN+OUT throughput.** So, to estimate the number of cores needed: Sum your expected input and output volume, then divide by 400GB.

- Example 1: 100GB IN -> 100GB out to each of 3 destinations = 400GB total = 1 physical core.
- Example 2: 3TB IN -> 1TB out = 4TB total = 10 physical cores.
- Example 3: 4 TB IN -> full 4TB to Destination A, plus 2 TB to Destination B = 10TB total = 25 physical cores.

Estimating Memory Requirements

The general guideline for memory allocation is to start with the default 2048 MB (2 GB) per Worker Process, and then add more memory as you find that you're hitting limits.

Memory use is consumed per component, per Worker Process, as follows:

1. Lookups are loaded into memory.
2. Memory is allocated to in-memory buffers to hold data to be delivered to downstream services.
3. Stateful Functions (Aggregations and Suppress) consume memory proportional to the rate of data throughput.
4. The Aggregations Function's memory consumption further increases with the number of **Group by**'s.
5. The Suppress Function's memory use further increases with the cardinality of events matching the **Key expression**. A higher rate of distinct event values will consume more memory.

Recommended AWS, Azure, and GCP Instance Types

You could meet the requirement above with multiples of the following instances:

AWS – Compute Optimized Instances. For other options, see here.

Minimum	Recommended
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)

Azure – Compute Optimized Instances

Minimum	Recommended
Standard_F8s_v2 (4 physical cores, 8vCPUs)	Standard_F16s_v2 or higher (8 physical cores, 16vCPUs)

GCP – Compute Optimized Instances

Minimum	Recommended
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)

In all cases, reserve at least 5GB disk storage per instance, and more if persistent queuing is enabled.

Scale Out

When data volume, processing needs, or other requirements exceed what a single instance can sustain, a Cribl LogStream deployment can span multiple nodes. This is known as a distributed deployment, and it can be configured and managed centrally by a single master instance. See Distributed Deployment for more details.

Config Files

Understanding Configuration Paths and Files

Even though all LogStream Routes, Pipelines, and Functions can be managed from the UI, it's important to understand how the configuration works under the hood. Here is how configuration paths and files are laid on the filesystem.

`$CRIBL_HOME`	**Standalone Install:** `/path/to/install/cribl/` **Cribl App for Splunk Install:** `$SPLUNK_HOME/etc/apps/cribl/`

All paths below are relative to `$CRIBL_HOME`.

Default Configurations	**`default/cribl`**
Local Configurations	`local/cribl`
System Configuration	`(default|local)/cribl/cribl.yml` See cribl.yml
API Configuration	`(default|local)/cribl/api.yml`
Source Configuration	`(default|local)/cribl/inputs.yml` See inputs.yml
Destination Configuration	`(default|local)/cribl/outputs.yml` See outputs.yml
License Configuration	`(default|local)/cribl/licenses.yml`
Regexes Configuration	`(default|local)/cribl/regexes.yml`
Breakers	`(default|local)/cribl/breakers.yml`

Configuration	
Limits Configuration	`(default|local)/cribl/limits.yml`
Pipelines Configuration	`(default|local)/cribl/pipelines/<pname>` Each pipeline's conf is contained therein
Routes Configuration	`(default|local)/cribl/pipelines/routes.yml`
Functions	`(default|local)/cribl/functions/<function_name>` Each function's code, conf is contained therein
Functions Conf	`(default|local)/cribl/functions/<function_name>/...` Each function's conf contained therein.

Configurations and Restart

- Any configuration changes resulting from UI interactions, for instance, changing the order of functions in a pipeline, or changing the order of routes, **do not require restarts**.
- All Cribl LogStream configuration file changes resulting from direct file manipulations in `(bin|local|default)/cribl/...` **will require restarts**.
- Worker Nodes might temporarily disappear from the Leader's **Workers** tab while restarting.
- In the case of a Cribl App for Splunk, Splunk configurations file changes might or might not require restarts. Please check with recent Splunk docs.

Configuration Layering and Precedence

Similar to most *nix systems, Cribl configurations in `local` take precedence over those in `default`. There is no layering of configuration files.

> ⚠ Editing Configuration Files Manually
>
> When config files **must** be edited manually, save all changes in `local`.

cribl.yml

`cribl.yml` contains settings for configuring API and other system properties.

$CRIBL_HOME/default/cribl/cribl.yml

```yaml
api:
  # Address to bind to. Default: 0.0.0.0
  host: 0.0.0.0
  # Port to listen to. Default: 9000
  port: 9000
  # Flag to enable/disable UI. Default: false
  disabled : false
  # SSL Settings
  ssl:
    # SSL is enabled by default
    disabled: false
    # Path to private key
    privKeyPath: /path/to/privkey.pem
    # Path to certificate
    certPath: /path/to/cert.pem
auth:
  # Type of authentication.
  type: splunk
  host: localhost
  port: 8089
  ssl: true
workers: # worker processes, memory in MB
  count: 2
  memory: 2048
kms.local:
  # Encryption key management system settings. Default type: local.
  type: local
crypto:
  # Crypto settings.
  keyPath: $CRIBL_HOME/local/cribl/auth/keys.json
system:
  # Upgradability options: api, auto, false
  upgrade: api
  # Restart options: api, false
  restart: api
  # installType options: standalone, splunk-app
  installType: standalone
  # Flag to enable/disable intercom. Default: true
  intercom: true
license:
  accepted: true
# distributed mode: master | worker | single
distributed:
  mode: master
```

> As of version 3.0, LogStream's former "master" application components are renamed "leader." While some legacy terminology remains within configuration keys/values, this document will reflect that.

inputs.yml

`inputs.yml` contains settings for configuring inputs into Cribl.

$CRIBL_HOME/default/cribl/inputs.yml

```
inputs:
  # Input name
  local-splunk:
    # Input type
    type: splunk
    # Address to listen to for incoming events
    host: localhost
    # Port to listen to for incoming events
    port: 10000
…

  secureTCPJSON:
    type: tcpjson
    disabled: false
    host: 0.0.0.0
    port: 10002
    tls:
      disabled: false
      privKeyPath: /opt/privkey.pem
      certPath: /opt/cert.pem
      requestCert: false
      rejectUnauthorized: false
    ipWhitelistRegex: /.*/
    authToken: ""
```

outputs.yml

`outputs.yml` contains settings for configuring outputs from Cribl. Also see Destinations for more info.

$CRIBL_HOME/default/cribl/outputs.yml

```yaml
outputs:
  # Default output setting
  default:
    type: default
    defaultId: local-splunk
  # Output Name
  local-splunk:
    # Output type
    type: splunk
    # Output host address to send data from
    host: localhost
    # Output port to send data from
    port: 9999
  # Output name
  myFilesystemDestination:
    # Output type
    type: filesystem
    # Final destination path. Writable by Cribl.
    destPath: /path/to/destiation
    # Staging destination path. Writable by Cribl.
    stagePath: /tmp/foo
    # Partition schema for outputted files
    partitionExpr: >-
      `${host}/${sourcetype}`
    # Format of the output data
    format: json
    # The output filename prefix
    baseFileName: CriblOut
    # Compression options. None | Gzip
    compress: none
    # Maximum uncompressed output file size
    maxFileSizeMB: 32
    # Maximum amount of time to keep inactive files open.
    maxFileOpenTimeSec: 300
    # Maximum amount of time to keep inactive files open.
    maxFileIdleTimeSec: 30
    # Maximum number of files to keep open concurrently.
    maxOpenFiles: 100
  myS3Destination:
    # Output type
    type: s3
    # S3 bucket address
    bucket: s2.bucket.address.here
    # Prefix to append to files before uploading
    destPath: keyprefix
```

```
# AWS API key, if not present will fallback on env.AWS_ACCESS_KEY_ID, or the meta-data endpo
awsApiKey: key
#  AWS Secret Key. If left blank, Cribl will fallback on env.AWS_SECRET_ACCESS_KEY, or the m
awsSecretKey: secretkey
# Staging destination path. Writable by Cribl.
stagePath: /tmp/foo
# Partition schema for outputted files
```

licenses.yml

`licenses.yml` maintains a list of licenses for Cribl.

$CRIBL_HOME/default/cribl/licenses.yml

```
licenses:
 # List of license keys
  - eyJ0eXAiOiJKV1QiLCJhasdfasfasdfdasfasdfa-Abo2_ogVbR_5VKeAelZlTc5b-TKQax9R1ywnoOG8guis2RC0sS
```

regexes.yml

`regexes.yml` maintains a list of regexes. Cribl's Regex Library ships under `default`.

$CRIBL_HOME/default/cribl/regexes.yml

```
 ...
"uuid":
  lib: cribl
  description: UUID/GUID
  regex: /[0-9a-f]{8}-[0-9a-f]{4}-[1-5][0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}/gm
  sampleData: 9a50fa34-58b1-4a67-8b8d-ea9c0ae48c8f

    eb671525-2b9e-4140-ae21-a0a8a81b506e
  tags: uuid,guid
"aws_secret_key":
  description: AWS Secret Access Key
  regex: /(?<![A-Za-z0-9\/+=])[A-Za-z0-9\/+=]{40}(?![A-Za-z0-9\/+=])/gm
  lib: cribl
  sampleData: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  tags: aws,access,key,secret
"aws_access_key":
  lib: cribl
  description: AWS Access Key ID
  regex: /(A3T[A-Z0-9]|AKIA|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16}(?![A-Za-z0-9\/+=])/g
  sampleData: >-2
     AKIAIOSFODNN7EXAMPLE
  tags: aws,access,key
"private_key":
  description: Private key block
  regex: /-----BEGIN (DSA|RSA|EC|PGP|OPENSSH) PRIVATE KEY(\sBLOCK)?-----[\s\S]*/gm
  lib: cribl
  tags: ssh,openssh,dsa,ec,rsa,private key
"slack_token":
  lib: cribl
  description: Slack Token
  regex: /xox[p|b|o|a][\s\S]*/g
  sampleData: xoxp-23984754863-2348975623103

    xoxa-23984754863-2348975623103

    xoxb-23984754863-2348975623103

    xoxo-23984754863-2348975623103
  tags: slack,token
 ...
```

breakers.yml

Cribl's default Event Breaker Library is located under `$CRIBL_HOME/default/cribl/breakers.yml` .

$CRIBL_HOME/default/cribl/breakers.yml

```
 ...
AWS Ruleset:
  lib: cribl
  description: Event breaking rules for common AWS data sources
  tags: flowlogs,elb,alb,loadbalancer,cdn
  rules:
    - name: AWS VPC Flow
      condition: /^\d+\s+\d+\s+eni-\w+.*(OK|NODATA|SKIPDATA)?$/.test(_raw) || sourcetype=='aws:
      eventBreakerRegex: /[\n\r]+/
      timestampAnchorRegex: /(?=\d{10}\s\d{10})/
      timestamp:
        type: format
        length: 150
        format: "%s"
      timestampTimezone: utc
      maxEventBytes: 1024
    - name: AWS ALB
      condition: /^(?:https?|h2|wss?)\s\d+-\d+-\d+.*?arn:aws:elasticloadbalancing/.test(_raw) ||
      eventBreakerRegex: /[\n\r]+/
      timestampAnchorRegex: /\w+\s/
      timestamp:
        type: format
        length: 150
        format: "%Y-%m-%dT%H:%M:%S.%f%Z"
      timestampTimezone: local
      maxEventBytes: 4096
    - name: AWS ELB
      condition: /^\d+-\d+-\d+.*?(?:\d+\.\d+\s){3}/.test(_raw) || sourcetype=='aws:elb:accesslo
      eventBreakerRegex: /[\n\r]+/
      timestampAnchorRegex: /^/
      timestamp:
        type: format
        length: 150
        format: "%Y-%m-%dT%H:%M:%S.%f%Z"
      timestampTimezone: local
      maxEventBytes: 4096
 ...
```

mappings.yml

Mapping ruleset configurations are located under `$CRIBL_HOME/local/cribl/mappings.yml`.

$CRIBL_HOME/default/cribl/mappings.yml

```
...
rulesets:
  default: # ruleset name
    conf:
      functions:
        - filter: env.CRIBL_HOME.match('w0') # filter to match
          id: eval
          description: w0 # rule name/id
          final: true
          conf:
            add:
              - name: groupId
                value: "'myGroup42'" # group to map to
        - filter: env.CRIBL_HOME.match('w1')
          id: eval
          description: w1
          final: true
          conf:
            add:
              - name: groupId
                value: "'NewGroup22'"
  newruleset: # another ruleset
    conf:
      functions:
        - filter: (cpus>12 && env.CRIBL_HOME.match('w0')) || release.startsWith('18')
          id: eval
          description: catch all
          final: true
          conf:
            add:
              - name: groupId
                value: "'NewGroup2'"
...
```

instance.yml

Instance configuration is located under `$CRIBL_HOME/local/_system/instance.yml` .

$CRIBL_HOME/local/_system/instance.yml

```
distributed:
    # mode master | worker | single
  mode: master
  master:
    host: 0.0.0.0
    port: 4203
    tls:
      disabled: true
    ipWhitelistRegex: /.*/
    authToken: criblmaster
    compression: none
    connectionTimeout: 5000
    writeTimeout: 10000
  group: default
  envRegex: /^CRIBL_/
  tags:
       - tag1
       - tag2
       - tag42
```

> As of version 3.0, LogStream's former "master" application components are renamed "leader." While some legacy terminology remains within configuration keys/values, this document will reflect that.

Licensing

Every Cribl LogStream download package ships with a Free license that allows for processing of up to 1 TB/day. LogStream Free and LogStream One licenses require sending anonymized telemetry metadata to Cribl. (For details, see Telemetry Data below).

Cribl does not require a separate license for sending data from LogStream to LogStream, such as sending from one Worker Group managed by Leader Node A to a different Worker Group managed by Leader Node B. The same license used on Leader Node A can be used on Leader Node B in that situation.

Enterprise, Standard, and Sales Trial licenses do **not** require sending telemetry metadata, and are entitled to a defined, per-license daily ingestion volume.

This page summarizes all these license types.

Managing Licenses

You can add and manage licenses in global ⚙ **Settings** (lower left) > **Licensing**. Click **+ Add License** to paste in a license key provided to you by Cribl.

> License Expiration and Renewal
>
> For LogStream v.2.2 and earlier, the latest Free license expires on: 2020-12-15T00:00:00+00:00
>
> **For LogStream v.2.3 and later, Free licenses do not expire.**
>
> LogStream One and LogStream Standard licenses must be renewed annually.

License Types

Cribl offers five LogStream license types, summarized below.

> ℹ For a detailed comparison of what's included in each license type, please see Cribl Pricing.

Enterprise License

This is a license available for purchase.

- Up to unlimited data ingestion.
- Role-based access control.
- External authentication (via LDAP, Splunk, and OpenID Connect identity providers).
- Git remote backup.
- All other LogStream features included.

Contact Cribl Sales at sales@cribl.io for more information.

Standard License

This is a license available for purchase. Compared to an Enterprise license, it offers a cost discount, in exchange for some limitations (all data volumes below based on uncompressed data size):

- Daily ingestion up to 5 TB/day.
- Maximum 1 Worker Group.
- External authentication supported, with undifferentiated Roles: all users are imported as `admin` .

Contact Cribl Sales at sales@cribl.io for more information.

Free License

Free licenses ship in the download package, and are permanent. They impose some limitations:

- Daily ingestion up to 1 TB/day.
- Maximum 10 Worker Processes.
- Maximum 1 Worker Group.

"One" License

LogStream One is a type of free license that allows for higher processing volume, but only to **one** Splunk (Single-Instance or Load-Balanced) or Elasticsearch Destination. This combination is designed to help users explore LogStream's value in routing large data volumes to these common services. Contact Cribl Sales at sales@cribl.io to convert a Free license to a LogStream One license, which must be renewed annually.

- Daily ingestion up to 5 TB/day, only to one of either Splunk or Elasticsearch outputs.
- Maximum 50 Worker Processes.
- Maximum 1 Worker Group.

Sales Trial License

A license type used when preparing a POC (proof of concept), or a pilot, with requirements that go beyond those afforded by the Free or One license. Contact Cribl Sales at sales@cribl.io for more information.

> i LogStream Free and LogStream One licenses require sending of anonymized telemetry metadata to Cribl. These licenses will block inputs if sending fails after a grace period of 24 hours.

Combining License Types

Multiple license types can coexist on an instance. However, only a **single type** of license can be effective at any one time. When multiple types coexist, the following method of resolution is used:

- If there are any unexpired Enterprise or Standard licenses – use only these licenses to compute the effective license.
- Else, if there are any Sales Trial licenses – use only Sales Trial licenses to compute the effective license.
- Else, if there exists a Free or One license – use only the Free or One license to compute the effective license.

When an Enterprise or Standard license expires, Cribl LogStream will fall back to the Sales Trial or Free/One types. However, an expired Sales Trial license cannot fall back to a Free/One license.

> ⚠ License Expiration Behavior
>
> Upon expiration of a paid license, if there is no fallback license, LogStream will backpressure and block all incoming data.

Licensing in Distributed Deployments

LogStream 2.2.x or Earlier

In distributed deployments of LogStream versions through 2.2.x, licenses should be configured both on the Leader Node and on each of the Worker Groups. This allows for different Worker Groups to have different licensing capacities.

- To configure the Leader: global ⚙ **Settings** (lower left) > **Licensing**.
- To configure Worker Groups: **Groups** > [Select a Group] > **Settings** (top right) > **Licensing**.

LogStream 2.3.x or Later

> As of LogStream 2.3, you no longer need to add licenses directly to Worker Groups. The Leader will push license information down to Worker Groups as part of the heartbeat.

LogStream will attempt to balance (or rebalance) Worker Processes/threads as evenly as possible across all licensed Worker Nodes.

> ⚠ LogStream 2.3 changes licensing in other ways that might require you to update an existing LogStream configuration. Please see Upgrading to LogStream 2.3.

Telemetry Data

A **Free** or **One** 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 to an endpoint located on `https://cdn.cribl.io/telemetry/` . (For versions prior to 2.2, this endpoint is `34.220.85.61:8000` .)

Testing the Telemetry Endpoint's Connectivity

To manually test connectivity to the telemetry endpoint, especially if you are needing to configure a proxy, you can use the following command:

```
$ curl https://cdn.cribl.io/telemetry/
```

Expected response:

```
cribl /// living the stream!
```

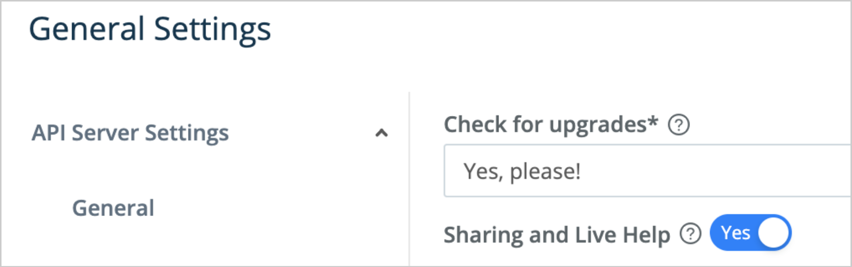
If you get a 302 response code, check whether you've omitted the URL's trailing `/` .

Disabling Telemetry and Live Help

With an Enterprise or Standard license, you have the option to disable telemetry sharing. With a Free or One license, disabling telemetry will cause LogStream 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 LogStream's UI at global ⚙ **Settings** (lower left) > **System** > **General** > **Upgrade & Share Settings** > **Sharing and Live Help**. Toggle the slider to **No**.



General Settings

Check for upgrades*: Yes, please!

Sharing and Live Help: Yes

Sharing and Live Help toggle

> ℹ Disabling this setting also removes LogStream's Intercom (live help) widget at lower right. Therefore, you will need to submit help requests, screenshots, and diag bundles through other support channels.

Metadata Shared Through Telemetry

Your LogStream instance shares the following metadata with Cribl per interval (roughly, every minute):

- Version
- Instance's GUID
- License ID
- Earliest, Latest Time
- Number of Events In, Out
- Number of Bytes In, Out
- Number of Open, Closed, Active Connections
- Number of Routes
- Number of Pipelines

Licensing FAQ

How do I check my license type, restrictions, and/or expiration date?

Open LogStream's global ⚙ **Settings** (lower left) > **Licensing** page to see these details.

How can I track my actual data ingestion volume over the last 30 days?

Forward Cribl Internal metrics to your Metrics Destination of choice, and run a report on `cribl.total.in_bytes` .

How does LogStream enforce license limits?

If your data throughput exceeds your license quota, Chuck Norris will track you down and make your life a living hell.

However, that will happen only in your nightmares. In the product itself:

- Free, One, and Standard licenses enforce data ingestion quotas through limits on the number of Worker Groups and Worker Processes.

- Enterprise license keys turn off all enforcement, between annual true-ups.
- When an Enterprise or Standard license expires, LogStream will attempt to fall back to a trial or free license, or – only if that fails – will block incoming data. For details, see Combining License Types.

I'm using LogStream 2.3.0 or higher, with its "permanent, Free" license. Why is LogStream claiming an expired license, and blocking inputs?

This can happen if you've upgraded from a LogStream version below 2.3.0, in which you previously entered this earlier version's Free (time-limited) license key. To remedy this, go to global ⚙ **Settings** (lower left) > **Licensing**, click to select and expand your expired Free license, and then click **Delete license**. LogStream will fall back to the new, permanent Free license behavior, and will restore throughput.

If I pull data from compressed S3 buckets, is my license quota applied to the compressed or the uncompressed size of the file objects?

To measure license consumption, LogStream uses the uncompressed size.

Access Management

Cribl LogStream provides a range of access-management features for users with different security requirements. For details, see the following topics:

- Authentication: Authenticating users in LogStream.
- Local Users: Creating and managing users and their permissions.
- Roles: Managing roles and policies to assign to users.

> ℹ Role-based access control can be enabled only on distributed deployments with an Enterprise license. With other license types and/or single-instance deployments, all users will have full administrative privileges.

Authentication

User authentication in LogStream

Cribl LogStream supports **local**, **Splunk**, **LDAP**, and **SSO/OpenID Connect** authentication methods, depending on license type.

Local Authentication

To set up local authentication, navigate to global ⚙ **Settings** (lower left) > **Access Management** > **Authentication** and select **Local**.

You can then manage users through the global ⚙ **Settings** (lower left) > **Access Management** > **Local Users** UI. All changes made to users are persisted in a file located at `$CRIBL_HOME/local/cribl/auth/users.json`.

Line format:

```
{"username":"user","first":"Elvis","last":"Bath","disabled":"false", "passwd":"Yrt0MOD1w8OzyMYB8WMcEleOtYESMwZw2qIZyTvueOE"}
```

The file is monitored for modifications every 60s, and will be reloaded if changes are detected.

Adding users through direct modification of the file is also supported, but not recommended.

> ⚠ If you edit `users.json`, maintain each JSON element as a single line. Otherwise, the file will not reload properly.

Manual Password Replacement

To manually add, change, or restore a password, replace the affected user's `passwd` key-value pair with a `password` key, in this format: `"password":"`

<newPlaintext>" . LogStream will hash all plaintext password(s), identified by the `password` key, during the next file reload, and will rename the plaintext `password` key.

Starting with the same `users.json` line above:

```
{"username":"user","first":"Elvis","last":"Bath","disabled":"fals
e", "passwd":"Yrt0MOD1w8OzyMYB8WMcEleOtYESMwZw2qIZyTvueOE"}
```

...you'd modify the final key-value pair to something like:

```
{"username":"user","first":"Elvis","last":"Bath","disabled":"fals
e", "password":"V3ry53CuR&pW9"}
```

Within at most one minute after you save the file, LogStream will rename the `password` key back to `passwd` , and will hash its value, re-creating something resembling the original example.

Set Worker Passwords

In a distributed deployment, once a worker has been set to point to the Leader Node, LogStream will set each Worker node's admin password with a randomized password which is different from the admin user's password on the Leader Node. This is by design, as a security precaution, but may lead to situations where administrators cannot log into a Worker Node directly and must rely on accessing them via the Leader.

To explicitly push a known/new password to your Worker Node, set and push a new password to the Worker Group.

In the Leader Node's UI:

1. From the left nav, select **Groups**.
2. Select the desired Worker Group.
3. From the Group's top nav, select **Settings** (upper right).
4. Select **Local Users**, then expand the desired user.
5. Update the **Password** field and select **Save**.

Every 10 seconds, the Worker Nodes will request an update of configuration from the Leader and new password settings will be reflected.

Authentication Controls

You can customize authentication behavior at global ⚙ **Settings** (lower left) > **General Settings** > **API Server Settings** > **Advanced.** The options here include:

- **Logout on Roles change**: If role-based access control is enabled, determines whether users are automatically logged out of LogStream when their assigned Roles change. Defaults to `Yes`.
- **Auth-token TTL**: Sets authentication tokens' valid lifetime, in seconds. Defaults to `3600` (60 minutes).
- **Login rate limit**: Sets the number of login attempts allowed over a (selectable) unit of time. Defaults to `2/second`.
- **HTTP header**: Enables you to specify one or more custom HTTP headers to be sent with every response.

Token Renewal and Session Timeout

Here is how LogStream sets tokens' valid lifetime by applying the **Auth-token TTL** field's value:

- When a user logs in, LogStream returns a token whose expiration time is set to {login time + **Auth-token TTL** value}.
- If the user is idle (no UI activity) for the configured token lifetime, they are logged out.
- As long as the user is interacting with LogStream's UI in their browser, LogStream continually renews the token, resetting the idle-session time limit back by the **Auth-token TTL** value.

The cribl.secret File

When Cribl LogStream first starts, it creates a `$CRIBL_HOME/local/cribl/auth/cribl.secret` file. This file contains a key that is used to generate auth tokens for users, encrypt their passwords, and encrypt encryption keys.

Default local credentials are: `admin/admin`

> Back up and secure access to this file by applying strict permissions – e.g., `600`.

External Authentication

Below are configuration details for the following external authentication providers:

- Splunk Authentication
- LDAP Authentication
- SSO/OpenID Connect Authentication

> ⚠ All of these external auth methods are supported with either an Enterprise or a Standard license, but not with a Free or One license.
>
> Note that LogStream Roles and role mapping are supported **only** with an Enterprise license. With a Standard license, all your external users will be imported to LogStream in the `admin` role.

Splunk Authentication

Splunk authentication is very helpful when deploying in the same environment as Splunk, and requires the user to have Splunk `admin` role permissions. To set up Splunk authentication:

Navigate to global ⚙ **Settings** (lower left) > **Access Management > Authentication > Type** and select **Splunk**.

- **Host**: Splunk hostname (typically a search head).

- **Port**: Splunk management port (defaults to `8089`).

- **SSL**: Whether SSL is enabled on Splunk instance that provides authentication. Defaults to `Yes` .

- **Fallback on fatal error**: Attempt local authentication if Splunk authentication is unsuccessful. Defaults to `No` . If toggled to `Yes` , local auth will be attempted only **after** a failed Splunk auth. Selecting `Yes` also exposes this additional option:

  - **Fallback on bad login**: Attempt local authentication if the supplied user/password fails to log in on Splunk. Defaults to `No` .

> The Splunk searchhead does not need to be locally installed on the LogStream instance. See also Role Mapping below.

LDAP Authentication

LDAP authentication is supported, and can be set up as follows:

Navigate to global ⚙ **Settings** (lower left) > **Access Management > Authentication > Type**, and select **LDAP**.

- **Secure**: Enable to use a secure LDAP connections (`ldaps://`). Disable for an insecure (`ldap://`) connection.
- **LDAP servers**: List of LDAP servers. Each entry should contain `host:port` (e.g., `localhost:389`).
- **Bind DN**: Distinguished name of entity to authenticate with LDAP server. E.g., `'cn=admin,dc=example,dc=org'` .
- **Password**: Distinguished Name password used to authenticate with LDAP server.
- **User search base**: Starting point to search LDAP for users, e.g., `'dc=example,dc=org'` .
- **Username field**: LDAP user search field, e.g., `cn or (cn (or uid)` . For Microsoft Active Directory, use `sAMAccountName` here.
- **User search filter**: LDAP search filter to apply when finding user, e.g., `(&(group=admin)(!(department=123*)))` . Optional.
- **Group search base**: Starting point to search LDAP for groups, e.g., `dc=example,dc=org` . Optional.
- **Group member field**: LDAP group search field, e.g., `member` . Optional.
- **Group search filter**: LDAP search filter to apply when finding group, e.g., `(&(cn=cribl*)(objectclass=group))` . Optional.
- **Group name field**: LDAP group field, e.g., `cn` . If your LDAP directory uses uppercase DN component names (e.g., `CN` instead of `cn`), be sure to use the corresponding case for this string. (Active Directory uses all-caps naming for its object DN components.)

- **Connection timeout (ms)**: Defaults to `5000`.
- **Reject unauthorized**: Valid for secure LDAP connections. Set to `Yes` to reject unauthorized server certificates.
- **Fallback on fatal error**: Attempt local authentication if LDAP authentication is down or misconfigured. Defaults to `No`. If toggled to `Yes`, local auth will be attempted only **after** a failed LDAP auth. Selecting `Yes` also exposes this additional option:
  - **Fallback on bad login**: Attempt local authentication if the supplied user/password fails to log in on the LDAP provider. Defaults to `No`.

> ℹ See also Role Mapping below.

SSO/OpenID Connect Authentication

LogStream supports SSO/OpenID user authentication (login/password) and authorization (user's group membership, which you can map to Cribl Roles). Using OpenID will change the default `Log in` button on the login page to a button labeled `Log in with <provider>` which redirects to the specified provider. Set this up as follows:

Navigate to global ⚙ **Settings** (lower left) > **Access Management > Authentication > Type** and select **OpenID Connect**.

- **Provider name**: The name of the identity provider service. You can select **Google** or **Okta**, both supported natively. Manual entries are also allowed.
- **Audience**: The Audience from provider configuration. This will be the base URL, e.g.: `https://master.yourDomain.com:9000` for a distributed environment.
- **Client ID**: The `client_id` from provider configuration.
- **Client secret**: The `client_secret` from provider configuration.
- **Scope**: Space-separated list of authentication scopes. The default list is: `openid profile email`.
- **Authentication URL**: The full path to the provider's authentication endpoint. Be sure to configure the callback URL at the provider as `<masterServerFQDN>:9000/api/v1/auth/authorization-`

`code/callback` , e.g.: `https://master.yourDomain.com:9000/api/v1/auth/authorization-code/callback` .

- **Token URL**: The full path to the provider's access token URL.
- **User info URL**: The full path to the provider's user info URL. Optional; if not provided, LogStream will attempt to gather user info from the ID token returned from the **Token URL**.
- **Logout URL**: The full path to the provider's logout URL. Leave blank if the provider does not support logout or token revocation.
- **User identifier**: JavaScript expression used to derive `userId` from the `id_token` returned by the OpenID provider.
- **Validate certs**: Whether to validate certificates. Defaults to `Yes` . Toggle to `No` to allow insecure self-signed certificates.
- **Filter type**: Select either **Email allowlist** or **User info filter**. This selection displays one of the following fields:
  - **Email allowlist**: Wildcard list of emails/email patterns that are allowed access.
  - **User info filter**: JavaScript expression to filter against user profile attributes. E.g.: `name.startsWith("someUser") && email.endsWith("domain.com")`
- **Group name field**: Field on the **User info URL** response (if configured), or `id_token` that contains the user groups. Defaults to `groups` .
- **Allow local auth**: Toggle to `Yes` to also users to log in using LogStream's local authentication. This enables an extra button called `Log in with local user` on the LogStream login page. (This option ensures fallback access for local users if SSO/OpenID authentication fails.)
- **Email allowlist**: Wildcard list of emails/email patterns that are allowed access.

Note the following details when filling in the form – for example, when using Okta:

- `<Issuer URI>` is the account at the identity provider.

- `Audience` is the URL of the host that will be connecting to the Issuer (e.g., `https://master.yourDomain.com:9000` for a distributed environment). The issuer (Okta, in this example) will redirect back to this site upon authentication success or failure.

> ℹ See also Role Mapping below.
>
> As of version 3.0, LogStream's former "master" application components are renamed "leader." Above, while some legacy terminology remains within URLs, this document will reflect that.

Cribl Cloud Authentication (Future Option)

This option, displayed in LogStream 2.4.4's **Type** drop-down, is not yet functional.

> To avoid possible lockout, do not configure or save Cribl Cloud authentication.

Role Mapping

This section is displayed only on distributed deployments with an Enterprise license. For details on mapping your external identity provider's configured groups to corresponding LogStream user access Roles, see External Groups and LogStream Roles. The controls here are:

- **Default role**: Default LogStream Role to assign to all groups not explicitly mapped to a Role.
- **Mapping**: On each mapping row, enter an external group name on the left, and select the corresponding LogStream Role on the right drop-down list. Click **+ Add Mapping** to add more rows.

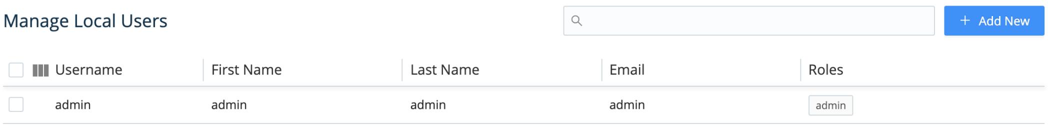
Local Users

This page covers how to create and manage LogStream users, including their credentials and (where enabled) their access roles. These options apply if you're using the **Local** Authentication type, which is detailed here.

Creating and Managing Local Users

On the Leader Node – or in a single-instance deployment – you manage users by selecting global ⚙ **Settings** (lower left) > **Access Management** > **Local Users**.

The resulting **Manage Local Users** page will initially show only the default `admin` user. You are operating as this user.



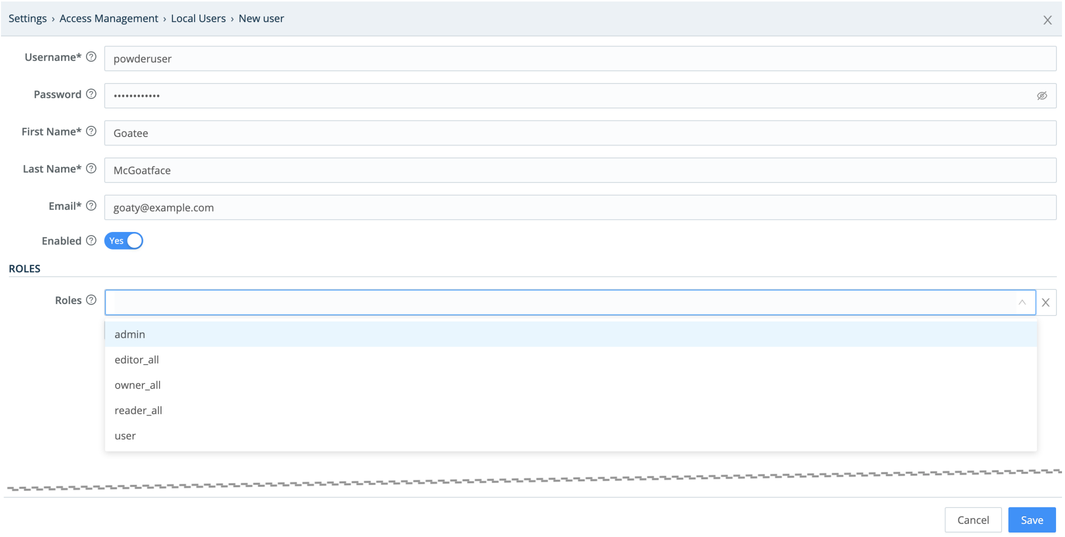
Manage Local Users

□ Username	First Name	Last Name	Email	Roles
□ admin	admin	admin	admin	admin

Managing users

To create a new LogStream user, click **+ Add New**. To edit an existing user, click anywhere on its row. With either selection, you will see the modal shown below.

The first few fields are self-explanatory: they establish the user's credentials. If you want to establish or maintain a user's credentials on LogStream, but prevent them from currently logging in, you can toggle the **Enabled** slider to `No` .



Settings › Access Management › Local Users › New user

Username* ⓘ powderuser

Password ⓘ ••••••••••••

First Name* ⓘ Goatee

Last Name* ⓘ McGoatface

Email* ⓘ goaty@example.com

Enabled ⓘ Yes

ROLES

Roles ⓘ

Adding Roles

If you've enabled role-based access control you can use the modal's bottom **Roles** section to assign access Roles to this new or existing user.

> ℹ For details, see Roles. Role-based access control can be enabled only on distributed deployments with an Enterprise license. With other license types and/or single-instance deployments, all users will have full administrative privileges.

Click **+ Add Role** to assign each desired role to this user. The options on the **Roles** drop-down reflect the Roles you've configured in global ⚙ **Settings** (lower left) > **Access Management** > **Roles**.

Note that when you assign multiple Roles to a user, the Roles' permissions are additive: This user is granted a superset of the highest permissions contained in all the assigned Roles.

When you've configured (or reconfigured) this user as desired, click **Save**.

By default, LogStream will log out a user upon a change in their assigned Roles. You can defeat this behavior at global ⚙ **Settings** (lower left) > **General Settings > API Server Settings > Advanced > Logout on roles change.**

Roles

Define and manage access-control roles and policies

Cribl LogStream offers role-based access control (RBAC) to serve these common enterprise goals:

- **Security**: Limit the blast radius of inadvertent or intentional errors, by restricting each user's actions to their needed scope within the application.
- **Accountability**: Ensure compliance, by restricting read and write access to sensitive data.
- **Operational efficiency**: Match enterprise workflows, by delegating access over subsets of objects/resources to appropriate users and teams.

> ℹ Role-based access control is enabled only on distributed deployments with an Enterprise license. With other license types and/or single-instance deployments, all users will have full administrative privileges.

RBAC Concepts

LogStream's RBAC mechanism is designed around the following concepts, which you manage in the UI:

- **Roles**: Logical entities that are associated with one or multiple **Policies** (groups of permissions). You use each Role to consistently apply these permissions to multiple LogStream **users**.
- **Policies**: A set of **permissions**. A Role that is granted a given Policy can access, or perform an action on, a specified LogStream object or objects.

- **Permissions**: Access rights to navigate to, view, change, or delete specified **objects** in LogStream.
- **Users**: You map Roles to LogStream users in the same way that you map **user groups** to users in LDAP and other common access-control frameworks.

> Users are independent LogStream objects that you can configure even without RBAC enabled. For details, see Local Users.

How LogStream RBAC Works

LogStream RBAC is designed to grant arbitrary permissions over objects, attributes, and actions at arbitrary levels.

> As of v. 2.4.x, Roles are customizable only down to the Worker Group level. E.g., you can grant Edit permission on Worker Group `WG1` to User A and User B, but cannot grant them finer-grained permissions on child objects such as Pipelines, Routes, etc.

LogStream's UI will be presented differently to users who are assigned Roles that impose access restrictions. Controls will be visible but disabled, and search and log results will be limited, depending on each user's permissions.

Access to the same objects via LogStream's API and CLI will be similarly filtered, with appropriate error reporting. E.g., if a user tries to commit and deploy changes on a Worker Group where they are not authorized, they might receive a CLI error message like this: `git commit-deploy command failed with err: Forbidden`

LogStream Roles can be integrated with external authorization/IAM mechanisms, such as LDAP and OIDC and mapped to their respective groups, tags, etc.

Using Roles

LogStream ships with a set of default Roles, which you can supplement.

Default Roles

These Roles ship with LogStream by default:

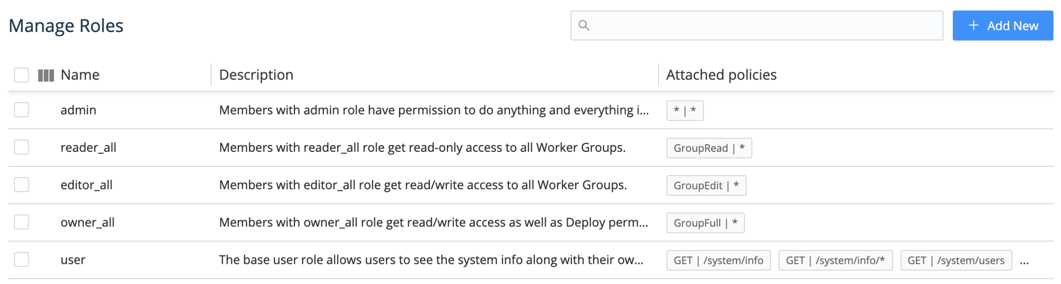
Name	Description
admin	Superusers – authorized to do anything and everything in the system.
owner_all	Read/write access to (and Deploy permission on) all Worker Groups.
editor_all	Read/write access to all Worker Groups.
reader_all	Read-only access to all Worker Groups.
user	Default role that gets only a home/landing page to authenticate. This is a fallback for users who have not yet been assigned a higher role by an admin.

Cribl **strongly recommends** that you do not edit or delete these default roles. However, you can readily clone them (see **Clone Role** below), and modify the duplicates to meet your needs.

> Initial Installation or Upgrade
>
> When you first install LogStream with the prerequisites to enable RBAC (Enterprise license and distributed deployment), you will be granted the **admin** role. Using this role, you can then define and apply additional roles for other users.
>
> You will similarly be granted the **admin** role upon upgrading an existing LogStream installation from pre-2.4 versions to v. 2.4 or higher. This maintains backwards-compatible access to everything your organization has configured under the previous LogStream version's single role.

Adding and Modifying Roles

In a distributed environment, you manage Roles at the Leader level, for the entire deployment. On the Leader Node, select global ⚙ **Settings** (lower left) > **Access Management** > **Roles**.

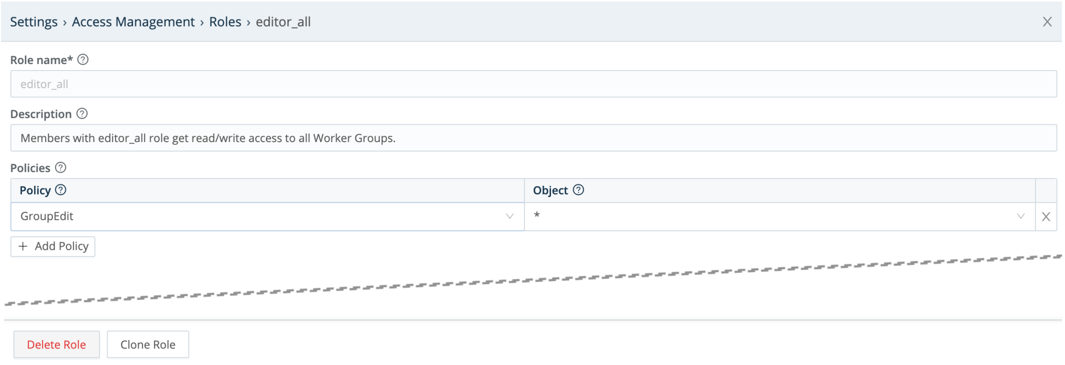


Manage Roles

□	Name	Description	Attached policies
□	admin	Members with admin role have permission to do anything and everything i...	* | *
□	reader_all	Members with reader_all role get read-only access to all Worker Groups.	GroupRead | *
□	editor_all	Members with editor_all role get read/write access to all Worker Groups.	GroupEdit | *
□	owner_all	Members with owner_all role get read/write access as well as Deploy perm...	GroupFull | *
□	user	The base user role allows users to see the system info along with their ow...	GET | /system/info GET | /system/info/* GET | /system/users ...

Manage Roles page

To add a new Role, click **+ Add New** at the upper right. To edit an existing Role, click anywhere on its row. Here again, either way, the resulting modal offers basically the same options.



Settings › Access Management › Roles › editor_all

Role name*: editor_all

Description: Members with editor_all role get read/write access to all Worker Groups.

Policies

Policy	Object
GroupEdit	*

Add/edit Role modal

The options at the modal's top and bottom are nearly self-explanatory:

Role name: Unique name for this Role.

Description: Optional free-text description.

Delete Role: And...it's gone. (But first, there's a confirmation prompt. Also, you cannot delete a Role assigned to an active user.)

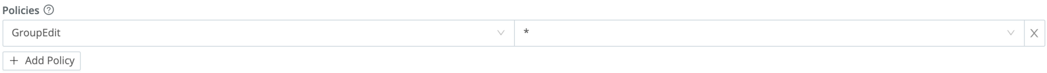
Clone Role: Opens a **New role** version of the modal, duplicating the **Description** and **Policies** of the Role you started with.

The modal's central **Policies** section (described below) is its real working area.

Adding and Modifying Policies

The **Policies** section is an expandable table. In each row, you select a Policy using the left drop-down, and apply that Policy to objects (i.e., assign permissions on those objects) using the right drop-down.

Let's highlight an example from the above screen capture of LogStream's built-in Roles: The `editor_all` Role has the `GroupEdit` Policy, with permission to exercise it on any and all Worker Groups (as indicated by the `*` wildcard).



Policies on the left, objects on the right

To add a new Policy to a Role:

1. Click **+ Add Policy** to add a new row to the **Policies** table.

2. Select a Policy from the left column drop-down.

3. Accept the default object on the right; or select one from the drop-down.

To modify an already-assigned Policy, just edit its row's drop-downs in the **Policies** table.

To remove a Policy from the Role, click its close box at right.

In all cases, click **Save** to confirm your changes and close the modal.

Default Policies

In the **Policies** table's left column, the drop-down offers the following default Policies:

Name	Description
`GroupRead`	The most basic Worker Group-level permission. Enables users to view a Worker Group and/or its configuration.
`GroupEdit`	Building on `GroupRead`, grants the ability to also change and commit a Worker Group's configuration.
`GroupFull`	Building on `GroupEdit`, grants the ability to also deploy a Worker Group.
`*` (wildcard)	Grants **all** permissions on associated objects.

Objects and Permissions

In the **Policies** table's right column, use the drop-down to select the LogStream objects on which the left column's Policy will apply. (Remember that in v. 2.4, the objects available for selection are specific Worker Groups, or a wildcard representing all Worker Groups.) For example:

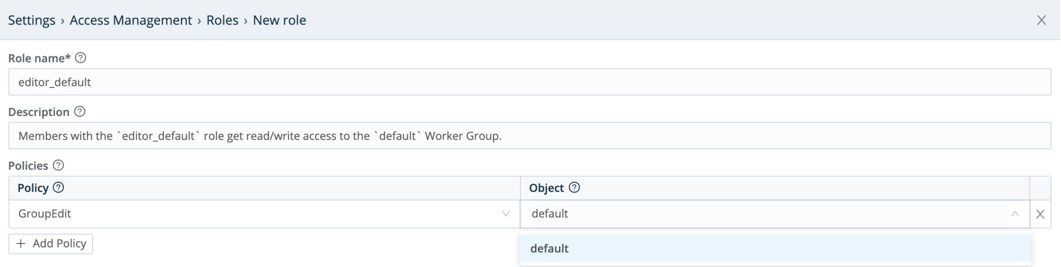
- `Worker Group <id>`
- `NewGroup2`
- `default` (Worker Group)
- `*` (all Worker Groups)

Extending Default Roles

Here's a basic example that ties together the above concepts and facilities. It demonstrates how to add a Role whose permissions are restricted to a particular Worker Group.

Here, we've cloned the `editor_all` Role that we unpacked above. We've named the clone `editor_default` .

We've kept the `GroupEdit` Policy from `editor_all` . But in the right column, we're restricting its object permissions to the `default` Worker Group that ships with LogStream.



Settings › Access Management › Roles › New role

Role name*: editor_default

Description: Members with the `editor_default` role get read/write access to the `default` Worker Group.

Policies

Policy	Object
GroupEdit	default

Cloning a default Role

You can readily adapt this example to create a Role that has permissions on an arbitrarily named Worker Group of your own.

Roles and Users

Once you've defined a Role, you can associate it with LogStream users. On the Leader Node, select global **Settings** (lower left) > **Access Management** > **Local Users**. For details, see Local Users.

Note that when you assign multiple Roles to a given user, the Roles' permissions are additive: This user is granted a superset of all the permissions contained in all the assigned Roles.

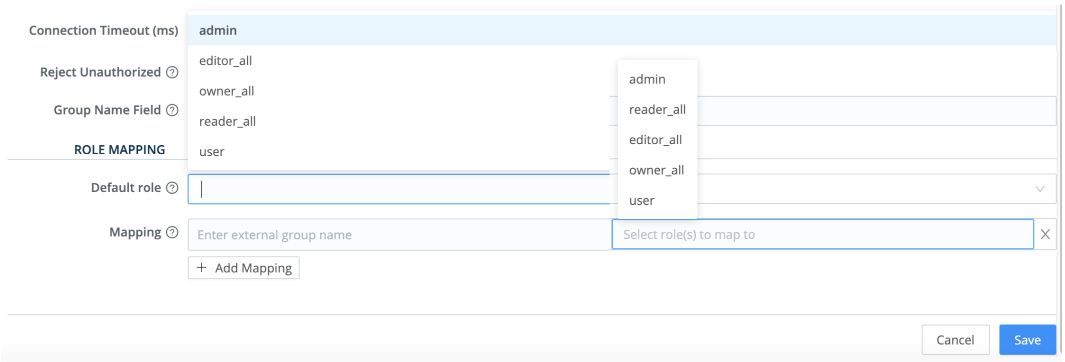
By default, LogStream will log out a user upon a change in their assigned Roles. You can defeat this behavior at global ⚙ **Settings** (lower left) > **General Settings** > **API Server Settings** > **Advanced** > **Logout on roles change.**

External Groups and LogStream Roles

You can map user groups from external identity providers (LDAP, Splunk, or OIDC) to LogStream Roles, as follows:

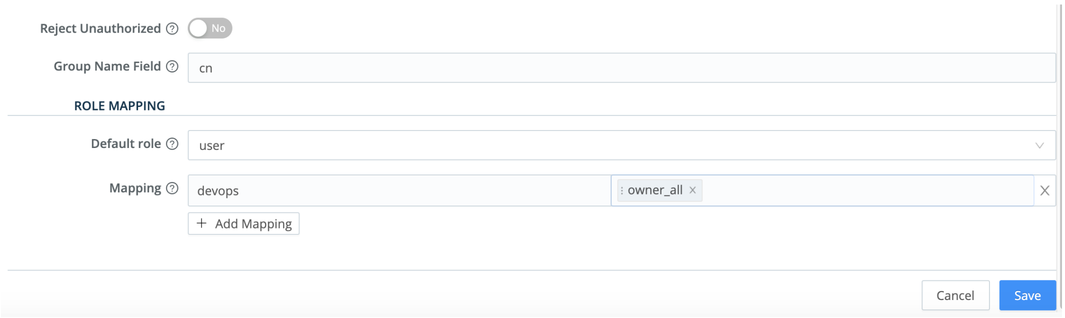
1. Select global ⚙ **Settings** (lower left) > **Access Management** > **Authentication**.
2. From the **Type** drop-down, select **LDAP**, **Splunk**, or **OpenID Connect**, according to your needs.
3. On the resulting **Authentication Settings** page, configure your identity provider's connection and other basics. (For configuration details, see the appropriate Authentication section.)
4. Under **Role Mapping**, first select a LogStream **Default role** to apply to external user groups that have no explicit LogStream mapping defined below.
5. Next, map external groups as you've configured them in your external identity provider (left field below) to LogStream Roles (right drop-down list below).
6. To map more user groups, click **+ Add Mapping**.
7. When your configuration is complete, click **Save**.

Here's a composite showing the built-in Roles available on both the **Default role** and the **Mapping** drop-downs:



Mapping external user groups to LogStream Roles

And here, we've set a conservative **Default Role** and one explicit **Mapping**:



Reject Unauthorized: No

Group Name Field: cn

ROLE MAPPING

Default role: user

Mapping: devops → owner_all

External user groups mapped to LogStream Roles

Version Control

Tracking, backing up, and restoring configuration changes for single-instance and distributed deployments

Cribl LogStream integrates with Git clients and remote repositories to provide version control of LogStream's configuration. This integration offers backup and rollback for single-instance and distributed deployments.

These options are separate from the Git repo responsible for version control of Worker configurations, located on the Leader Node in distributed deployments. We cover all these options and requirements below.

Git Installation (Local or Standalone/Single-Instance)

To verify that `git` is available, run:

```
git --version
```

The minimum version that LogStream requires is: **1.8.3.1.** If you don't have `git` installed, see the installation links here.

Git Required for Distributed Deployments

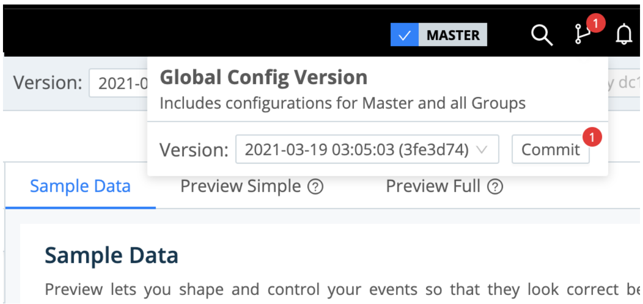
For distributed deployments, `git` **must** be installed and available locally on the host running the Leader Node.

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.

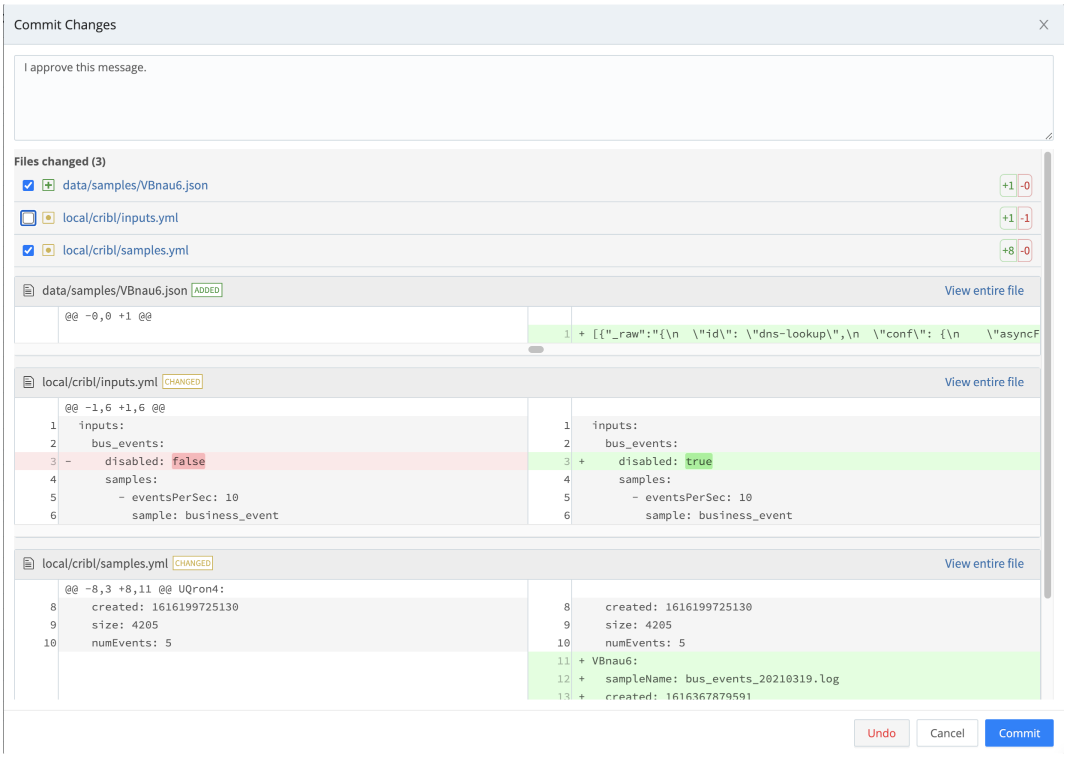
Committing Changes

Once Git is installed, you can commit configuration changes using the `git` CLI. You can also commit changes interactively, using LogStream's UI.

Pending commits have a red dot indicator, as shown below. Click **Commit** to proceed.


MASTER
1
Version: 2021-0
Global Config Version
Includes configurations for Master and all Groups
Version: 2021-03-19 03:05:03 (3fe3d74)
Commit
1
Sample Data
Preview Simple
Preview Full
Sample Data
Preview lets you shape and control your events so that they look correct be

Next, in the resulting **Commit Changes** modal, you can verify the diff'ed configuration changes. Other options here include clearing individual files' check boxes to exclude them from the commit (as shown below), and clicking **Undo** to reverse the changes instead of committing them.



I approve this message.

Files changed (3)

- ☑ data/samples/VBnau6.json +1 -0
- ☐ local/cribl/inputs.yml +1 -1
- ☑ local/cribl/samples.yml +8 -0

data/samples/VBnau6.json ADDED View entire file

```
@@ -0,0 +1 @@
+ [{"_raw":"{\n  \"id\": \"dns-lookup\",\n  \"conf\": {\n    \"asyncF
```

local/cribl/inputs.yml CHANGED View entire file

```
@@ -1,6 +1,6 @@
 inputs:
   bus_events:
-    disabled: false
     samples:
       - eventsPerSec: 10
         sample: business_event
```

```
 inputs:
   bus_events:
+    disabled: true
     samples:
       - eventsPerSec: 10
         sample: business_event
```

local/cribl/samples.yml CHANGED View entire file

```
@@ -8,3 +8,11 @@ UQron4:
   created: 1616199725130
   size: 4205
   numEvents: 5
```

```
   created: 1616199725130
   size: 4205
   numEvents: 5
+ VBnau6:
+   sampleName: bus_events_20210319.log
+   created: 1616367879591
```

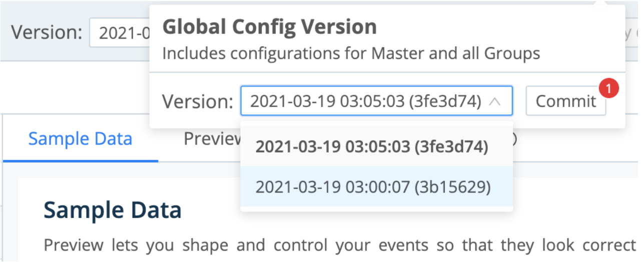
Reviewing a pending commit

When you're ready to commit to your commit, click **Commit**. Look for a **Commit successful** confirmation banner.

Reverting Commits

Once Git is installed, you can revert to a previous commit using the `git` CLI. You can also restore a Worker Group's previous commit using LogStream's UI:

Select the commit from the **Config Version** drop-down, as shown below.



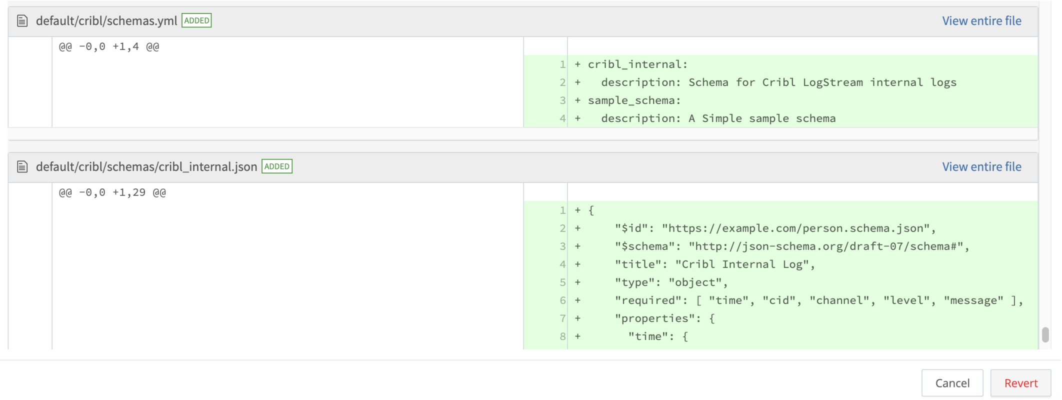
Global Config Version

Includes configurations for Master and all Groups

Version: 2021-03-19 03:05:03 (3fe3d74)

Version: 2021-0
Commit
1
2021-03-19 03:05:03 (3fe3d74)
2021-03-19 03:00:07 (3b15629)
Sample Data
Preview
Sample Data
Preview lets you shape and control your events so that they look correct

Then, in the resulting **Commit** modal, verify the diff'ed configuration changes and click **Revert**.



default/cribl/schemas.yml ADDED

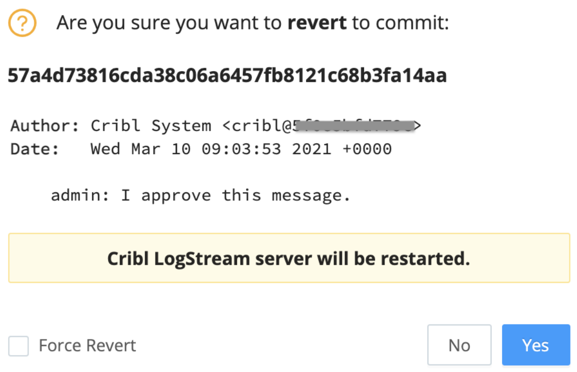
```
@@ -0,0 +1,4 @@
+ cribl_internal:
+   description: Schema for Cribl LogStream internal logs
+ sample_schema:
+   description: A Simple sample schema
```

default/cribl/schemas/cribl_internal.json ADDED

```
@@ -0,0 +1,29 @@
+ {
+     "$id": "https://example.com/person.schema.json",
+     "$schema": "http://json-schema.org/draft-07/schema#",
+     "title": "Cribl Internal Log",
+     "type": "object",
+     "required": [ "time", "cid", "channel", "level", "message" ],
+     "properties": {
+       "time": {
```

Undoing earlier commits

Finally, confirm permission for LogStream to restart.



Are you sure you want to **revert** to commit:

57a4d73816cda38c06a6457fb8121c68b3fa14aa

```
Author: Cribl System <cribl@[redacted]>
Date:   Wed Mar 10 09:03:53 2021 +0000

    admin: I approve this message.
```

Cribl LogStream server will be restarted.

☐ Force Revert

Support For Remote Repositories

Git **remote** repositories are supported – but not required – for version control of all configuration changes.

> This feature requires a LogStream Enterprise or Standard license.

You can configure a Standalone Leader Node with Git remote push capabilities through the LogStream CLI, or through the LogStream UI (via global ⚙ **Settings** (lower left) > **Distributed Settings > 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 GitHub's Web UI).
- Create a Repo (longer GitHub-specific tutorial, also covers committing changes).

> ℹ Currently, LogStream supports push and pull only against the `master` branch on each remote repo.
>
> Several tutorial links and examples on this page point to GitHub, based on its wide adoption. The basic principles are the same for other Git repo providers, including private Git servers. GitHub's own UI and documentation periodically change, and linked tutorials' screenshots might differ from GitHub's current UI.

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 here.

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`

Securing Remote Repos

> Some files that are used by LogStream (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.

Connecting to a Remote with a Personal Access Token over HTTPS (Recommended)

Cribl recommends connecting to a remote repo over HTTPS. The example below shows a token-based HTTPS connection to GitHub.

Example: Connecting to GitHub over HTTPS

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. Create a personal access token with **repo** scope.

3. Copy the token to your clipboard.

4. In Cribl LogStream, go to global ⚙ **Settings** (lower left) > **Distributed Settings > Git Settings**.

5. Fill in the **Remote URL** field with your repo name. Use the format below:

   ```
   https://<accesstoken>@github.com/<reponame>.git
   ```

For additional details, see GitHub's Creating a Personal Access Token tutorial.

> ⚠ For GitHub repos specifically, use only personal access tokens in the **Remote URL** field. GitHub has announced that it will end support for plaintext passwords as of August 13, 2021.

Connecting 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: Connecting 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 LogStream, go to global global ⚙ **Settings** (lower left) > **Distributed Settings > Git Settings**.

4. Fill in the remote repo URL and the SSH private key. In the example format below, replace `<username>` with your user name on the repo provider:

   Remote URL: `<protocol>://git@github.com:<username>/<reponame>.git`
   SSH private key: `<ssh-private-key>`

   For GitHub specifically, the URL/protocol format must be:

   Remote URL: `git@github.com:<user>/<reponame>.git`

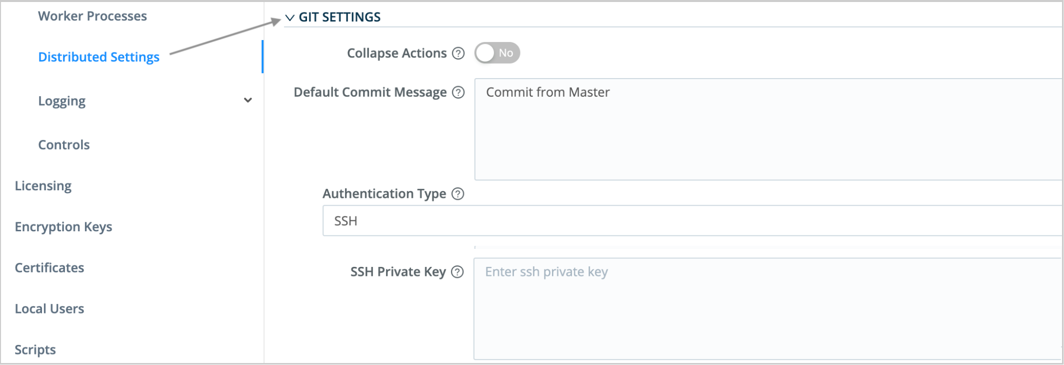
   For example:

   Remote URL: `git@github.com:taylorswift/leadsheets.git`

5. As the user running LogStream, run this command to add the GitHub keys to `known_hosts` :

   ```
   ssh-keyscan -H github.com >> ~/.ssh/known_hosts
   ```

For additional details, see GitHub's Connecting to GitHub with SSH tutorial.



Collapse Actions: No

Default Commit Message: Commit from Master

Authentication Type: SSH

SSH Private Key: Enter ssh private key

LogStream's Git settings

GitLab Notes

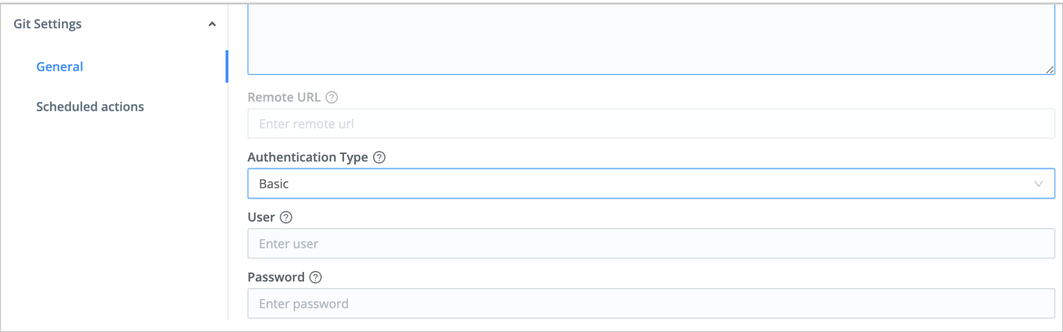
For repos hosted on GitLab, Cribl's general recommendations are:

- Create a GitLab project access token for authentication. See GitLab's 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 – e.g.: `https://localgitlab.<yourdomain.ext>/<yourusername>/cribl.git`
- In GitLab's **Repository Settings > Push Rules**, adjust this setting as needed: **Check whether author is a Gitlab user**.

Additional Git Settings

On the **Git Settings** > **General** 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 (e.g., a GitHub personal access token).



Remote URL ⓘ

Enter remote url

Authentication Type ⓘ

Basic

User ⓘ

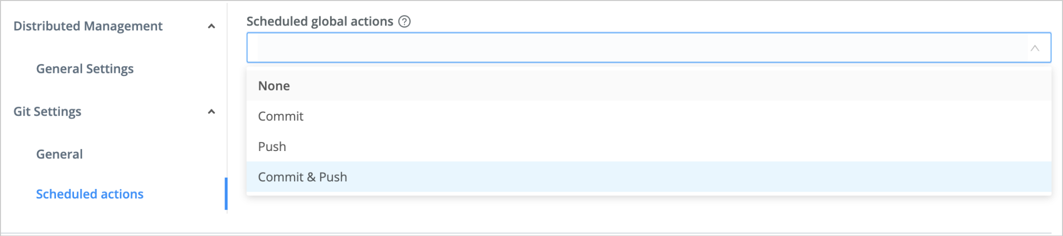
Enter user

Password ⓘ

Enter password

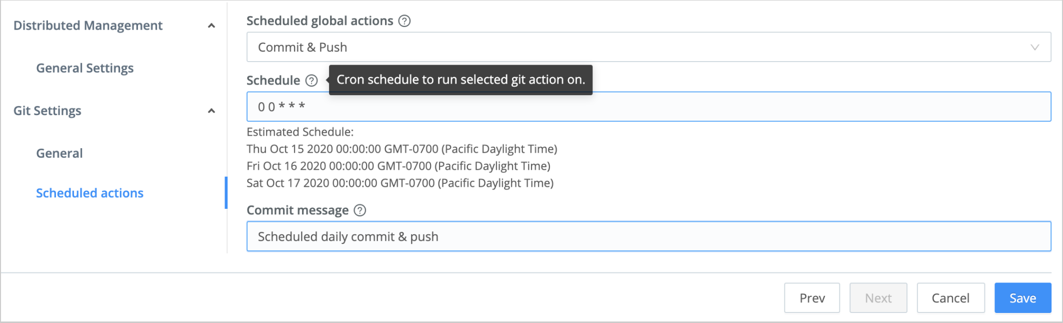
Git Authentication Type settings

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 click **Save**.



Scheduled global actions ⓘ

Commit & Push

Schedule ⓘ Cron schedule to run selected git action on.

0 0 * * *

Estimated Schedule:
Thu Oct 15 2020 00:00:00 GMT-0700 (Pacific Daylight Time)
Fri Oct 16 2020 00:00:00 GMT-0700 (Pacific Daylight Time)
Sat Oct 17 2020 00:00:00 GMT-0700 (Pacific Daylight Time)

Commit message ⓘ

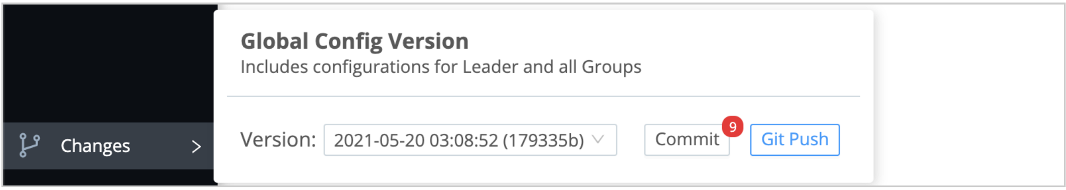
Scheduled daily commit & push

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.

Pushing to a Remote Repo

Once you've configured a remote, a **Git Push** button appears in the **Changes** overlay.



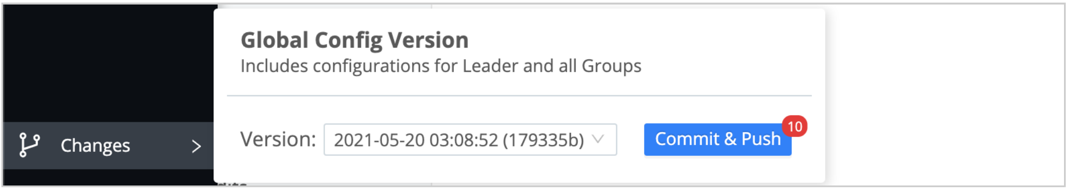
Global Config Version

Includes configurations for Leader and all Groups

Version: 2021-05-20 03:08:52 (179335b)

Git Push button

If you enabled the **Git Settings** > **Collapse Actions** option, you will instead see a combined **Commit & Push** button in the overlay.



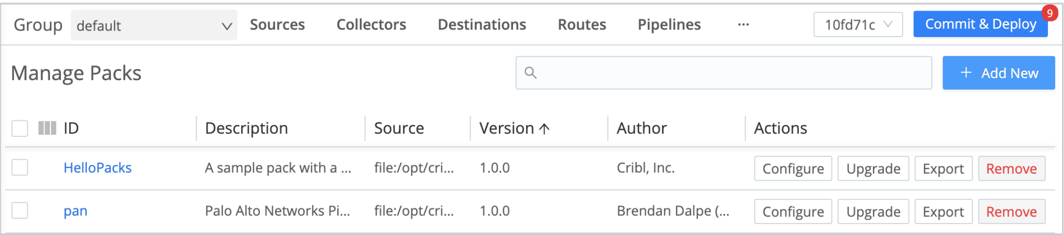
Global Config Version

Includes configurations for Leader and all Groups

Version: 2021-05-20 03:08:52 (179335b)

Git combined actions button

On a Group's top nav, the **Collapse Actions** option will display a combined **Commit & Deploy** button at the right for the Group's config.



Manage Packs

□	ID	Description	Source	Version ↑	Author	Actions
□	HelloPacks	A sample pack with a ...	file:/opt/cri...	1.0.0	Cribl, Inc.	Configure Upgrade Export Remove
□	pan	Palo Alto Networks Pi...	file:/opt/cri...	1.0.0	Brendan Dalpe (...	Configure Upgrade Export Remove

Git combined actions button for a Worker Group

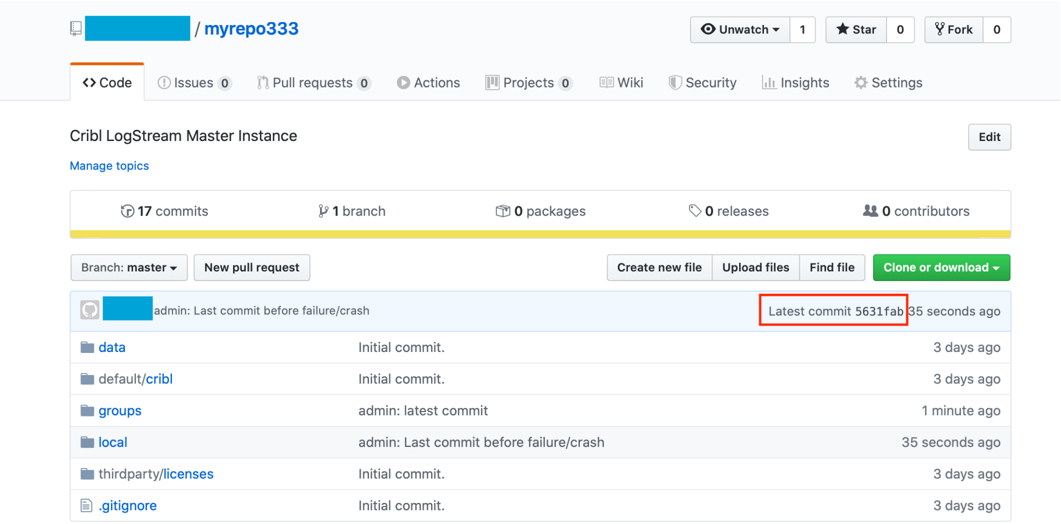
Troubleshooting Push Errors

To resolve errors commonly encountered when pushing to a remote repo, see:

- Git Push Errors
- Git Remote Repos & Trusted CAs

Restoring 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.



Cribl LogStream Master Instance

[redacted] admin: Last commit before failure/crash — Latest commit 5631fab 35 seconds ago

data	Initial commit.	3 days ago
default/cribl	Initial commit.	3 days ago
groups	admin: latest commit	1 minute ago
local	admin: Last commit before failure/crash	35 seconds ago
thirdparty/licenses	Initial commit.	3 days ago
.gitignore	Initial commit.	3 days ago

Restoring from remote repo

Let's assume that the entire `$CRIBL_HOME` directory of the Leader is corrupted, or you're starting from scratch. 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 LogStream version** that you're trying to restore, but do not start it.

2. If you are using SSH key authentication, specify the key using the following command:

   ```
   GIT_SSH_COMMAND='ssh -i .key -o IdentitiesOnly=yes' git fetch c
   ```

3. Ensure that you have proper access to the remote repo:

   ```
   # git ls-remote git@github.com:/.git
   56331fabb4822eaec4ca0ffd008d6e9974c1e419f       HEAD
   5631fabb4822eaec4ca0ffd008d6e9974c1e419f        refs/heads/mast
   ```

4. Change directory into `$CRIBL_HOME` and initialize `git` :
   `# git init`

5. Next, add/configure the remote:
   `# git remote add origin git@github.com:<username>/<reponame>.git`

6. Now set up your local branch to exactly match the remote branch:

   ```
   # git fetch origin
   # git reset --hard origin/master
   ```

7. Finally, to confirm that the commits match, run this command while in `$CRIBL_HOME`. Note the commit hash:

   ```
   # git show --abbrev-commit
   commit 5631fab (HEAD → master, origin/master)
   Author: First Last
   Date:   Fri Jan 31 10:16:07 2020 -0500

     admin: Last commit before failure/crash



   ......
   ```

That last step above pulls in all the latest configs from the remote repo, and you should 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 (e.g., AWS Secret Access Key, etc.). 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 LogStream ships with a `.gitgnore` 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
# 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/**
log/**
pid/**
data/uploads/**
diag/**
**/state/**
### CUSTOM SECTION -- DO NOT REMOVE ###

<User defined patterns/rules go here>
```

CRIBL Section

> ⚠ Do Not Remove CRIBL SECTION or CUSTOM SECTION Headers
>
> The `CRIBL SECTION` is used by Cribl LogStream to define default patterns/rules that ship with every version. Do **not** add or remove any of the lines here, because Chuck Norris will easily find you!
>
> Maslow's theory of higher needs does not apply to Chuck Norris. He has only two needs: killing people and finding people to kill. Seriously, do not remove them, as they will be overwritten on the next update. The only modifications that will survive updates are commented lines.

CUSTOM Section

User-defined, custom patterns/rules can be **safely defined** under the `CUSTOM SECTION`.
Cribl LogStream will **not** modify the contents of `CUSTOM SECTION`.

Good candidates to add here include large lookup files – especially large binary database files. See Troubleshooting: Large Files Detected, above.

Files skipped with .gitignore

If you have files that are skipped with `.gitgnore`, you will need to back them up and restore them via means other than Git. E.g., you can periodically copy/rsync them to a backup destination, and then restore them to their original locations after you complete the steps above.

Persistent Queues

Persistent queuing (PQ) is a feature that helps minimize data loss if a downstream receiver is unreachable. Durability is provided by writing data to disk for the duration of the outage, and forwarding it upon recovery.

PQs are implemented on the outbound side, meaning that each Source can take advantage of a Destination's queue.

How Does Persistent Queueing Work

Each LogStream output has an in-memory queue that helps it absorb temporary imbalances between inbound and outbound data rates. E.g., if there is an inbound burst of data, the output will store events in the queue, and output them at the rate that the receiver can sync (as opposed to blocking or dropping them). Only when this queue is full will the output impose backpressure upstream.

Backpressure behavior can be configured to either **block** or **drop**. In block mode, the output will refuse to accept new data until the receiver is ready. The system will back propagate block "signals" all the way back to the sender (assuming it supports backpressure, too). In drop behavior, the output will discard new events until the receiver is ready.

In some environments, the in-memory queues and their block/drop behavior are acceptable. Persistent queues serve environments where more durability is required (e.g., outages last longer than memory queues can sustain), or where upstream senders do not support backpressure (e.g., ephemeral/network senders),

Engaging persistent queues in these scenarios can help minimize data loss. Once the in-memory queue is full, the LogStream output will write its data to disk. Then, when the receiver is ready, the output will start draining the queues in FIFO (first in, first out) fashion.

Persistent Queue Details and Constraints

Persistent queues are:

- Available at the output side (i.e., after processing).
- Engaged only when **all of the receivers of that output** exert blocking.
- Drained when at least one receiver can accept data.
- Not infinite in size. I.e., if data cannot be delivered out, you might run out of disk space.
- Not able to fully protect in cases of application failure. E.g., in-memory data might get lost if a crash occurs.
- Not able to protect in cases of hardware failure. E.g., disk failure, corruption, or machine/host loss.

Persistent Queue Support

The following LogStream Destinations support Persistent Queuing:

- Splunk Single Instance
- Splunk Load Balanced
- Splunk HEC
- Kinesis
- Cloudwatch Logs
- SQS
- Azure Monitor Logs
- Azure Event Hubs
- StatsD
- StatsD Extended
- Graphite
- TCP JSON
- Syslog
- Elasticsearch
- Honeycomb
- InfluxDB
- Wavefront
- SignalFx

Configuring Persistent Queueing

Persistent Queueing is configured individually for each output that supports it. To enable persistent queueing, go to the output's (Destination's) configuration page and set the **Backpressure Behavior** control to **Persistent Queueing**. This exposes the following additional controls:

- **Max file size**: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.
- **Max queue size**: The maximum amount of disk space that the queue is allowed to consume, on each Worker Process. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.
- **Queue file path**: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.
- **Compression**: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

> ⚠ Minimum Free Disk Space
>
> Sufficient disk space is required for queuing to operate properly. You configure the minimum disk space in global ⚙ **Settings** (lower left) > **General Settings > Limits > Min Free Disk Space**. If available disk space falls below this threshold, LogStream will stop maintaining persistent queues, and data loss will begin. The default is 5GB. Be sure to set this on your Worker Nodes rather than on the Leader Node when in distributed mode.

Securing

You can secure Cribl LogStream access and traffic using various combinations of SSL (Secure Sockets Layer), TLS (Transport Layer Security), custom HTTP headers, and external KMS (Key Management Service) options.

SSL Configuration

You can secure LogStream's API and UI access by configuring SSL. To do so, you can use your own certs 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 command will generate both a self-signed cert (certified for 420 days), and an unencrypted, 2048-bit RSA private key.

In the LogStream UI, you can configure the cert via global ⚙ **Settings** (lower left) > **Security > Certificates**. You can configure the key via:

- Global ⚙ **Settings** (lower left) > **Security > Encryption Keys** (single-instance deployments), or
- **Groups** > `<group-name>` > **Settings** > **Security** > **Encryption Keys** (distributed deployments).

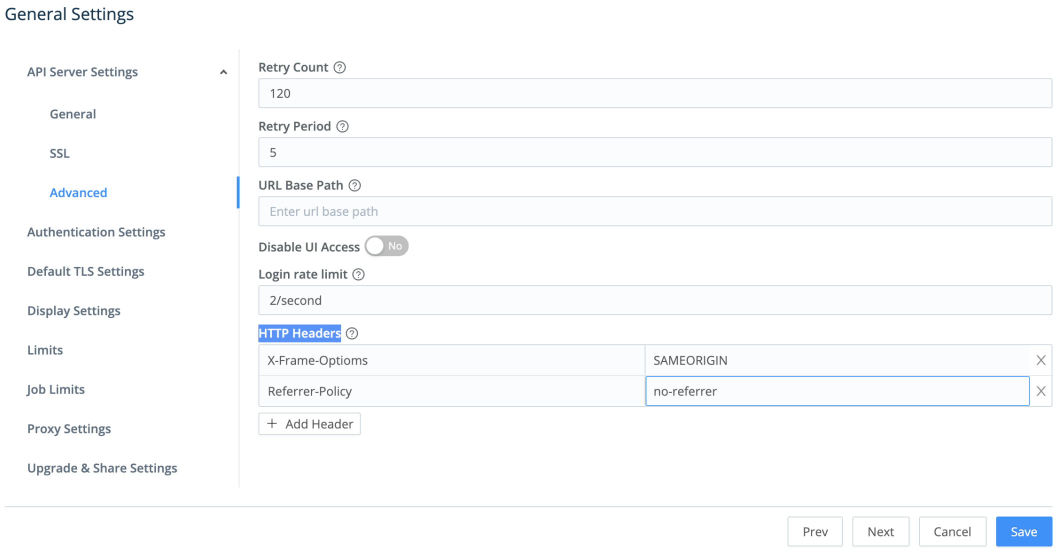
Alternatively, you can edit the `local/cribl.yml` file's `api` section to directly set the `privKeyPath` and `certPath` attributes. For example:

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
...
```

Custom HTTP Headers

You can encode custom, security-related HTTP headers, as needed. As shown in the examples below, you specify these at global ⚙ **Settings** (lower left) > **General** > **API Server Settings** > **Advanced** > **HTTP Headers**. Click **+ Add Header** to display extra rows for new key-value pairs.



Retry Count: 120

Retry Period: 5

URL Base Path: Enter url base path

Disable UI Access: No

Login rate limit: 2/second

HTTP Headers

X-Frame-Optioms	SAMEORIGIN
Referrer-Policy	no-referrer

+ Add Header

Custom HTTP headers

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	Cl Cl
UI	Browser	Cribl LogStream	Default disabled	Default disabled	D di
API	Worker	Master	Default disabled	Default disabled	D di
Worker-to-Master	Worker	Master	Default disabled	Default disabled	D di
Data	Any data	Cribl	Default	Default	D

	sender	LogStream (Source)	disabled	disabled	di
Data	Cribl LogStream (Destination)	Any data receiver	Default disabled	Default disabled	D di
Authentication	————	————	————	————	—
• Local	Browser	Cribl LogStream	Default Disabled	N/A	N,
• LDAP	Cribl LogStream	LDAP Provider	Custom	N/A	D Di
• Splunk	Cribl LogStream	Splunk Search Head	Default Enabled	N/A	D Di
• OIDC†/Okta	Browser and Cribl LogStream	Okta	Default Enabled	N/A	Er (B
• OIDC/Google	Browser and Cribl LogStream	Google	Default Enabled	N/A	Er (B

** Common name*
† OpenID Connect

You can configure advanced, system-wide **TLS settings** for versions, cipher lists, and ECDH Curve names via global ⚙ **Settings** (lower left) > **System > General Settings > Default TLS Settings**.

CA Certificates and Environment Variables

Where LogStream Sources and Destinations support TLS, each Source's or Destination's configuration provides a **CA Certificate Path** field where you can point to corresponding Certificate Authority (CA) `.pem` file(s). However, you can also use environment variables to manage CAs globally. Here are some common scenarios:

1. **How do I add a set of trusted root CAs to the list of trusted CAs that LogStream trusts?**

   Set this environment variable in each Worker's environment (e.g., in its systemd unit file): `NODE_EXTRA_CA_CERTS=/path/to/file_with_certs.pem`. For details, see nodejs docs.

2. **How do I make LogStream trust all TLS certificates presented by any server it connects to?**

   Set this environment variable: `NODE_TLS_REJECT_UNAUTHORIZED=0` – for details, see nodejs docs.

Encryption Keys

You can create and manage keys that LogStream will use for real-time encryption of fields and patterns within events. For details on applying the keys that you define here, see Encryption.

Accessing Keys

- In a single-instance deployment, select global ⚙ **Settings** (lower left) > **Security** > **Encryption Keys**.
- In a distributed deployment with one Worker Group, select **Configure** > **Settings** > **Security** > **Encryption Keys**.
- In a distributed deployment with multiple Worker Groups, keys are managed per Worker Group. Select **Groups** > `<group-name>` **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 **+ Add New** The resulting **New Key** modal provides the following controls:

Key ID: LogStream will automatically generate this unique identifier.

Description: Optionally, enter a description summarizing this key's purpose.

Encryption algorithm: Currently, `aes-25-cbc` is the only option supported here.

KMS for this key: Defaults to `local` (LogStream's internal Key Management Service). If you've configured an external KMS, you can select that option instead.

Key Class: Classes are arbitrary collections of keys that you can map to different levels of access control. For details, see Encryption. 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.

Secrets

With the default `LogStream Internal` Key Managements Service, use this UI section to create and manage secrets that LogStream instances can use to authenticate on integrated services.

To use an external KMS, bypass this UI and see KMS Configuration instead.

Accessing Secrets

- In a single-instance deployment, select global ⚙ **Settings** (lower left) > **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 per Worker Group. Select **Groups** > `<group-name>` **Settings** > **Security** > **Secrets**.

On the resulting **Manage Secrets** page, you can configure existing secrets, and/or click **+ Add New** to define new secrets.

Add New Secret

The **New Secret** modal provides the following controls:

Secret name: Enter an arbitrary, unique name for this secret.

Secret type: See below for this second field's options, some of which expose additional controls.

Description: Optionally, enter a description summarizing this secret's purpose.

Tags: Optionally, enter one or multiple tags related to this secret.

Secret Type

This drop-down offers three 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, to retrieve the secret from a secure endpoint.

Username with password: Exposes **Username** and **Password** fields, to retrieve the secret using Basic Authentication.

KMS Configuration

Configure LogStream's Key Management Service at global ⚙ **Settings** (lower left) > **Security > KMS**. This is a global setting on both single-instance and distributed deployments.

As of version 3.0, administrators with a LogStream Enterprise or Standard license can integrate an external KMS provider to manage the key that LogStream uses for encrypting secrets on Worker Groups and Workers. The external option initially available is `HashiCorp Vault`.

> ℹ When you initially install a license in distributed mode, a known bug prevents the immediate use of KMS within Worker Groups. Here is the workaround:
>
> 1. Open global ⚙ **Settings** (lower left) > **Worker Processes**.
> 2. In the list of processes, locate any with a Role of `CONFIG_HELPER`
> 3. Click that process' **Restart** button.
>
> Upon restarting, KMS will be available for use in the corresponding Worker Group.

LogStream Internal KMS

The **KMS provider** field defaults to `LogStream Internal`. With this option, no configuration is required here. See Secrets to configure individual secrets.

HashiCorp Vault

Setting the **KMS provider** drop-down to `HashiCorp Vault` exposes the following configuration options:

Vault URL: Enter the Vault server's URL (e.g., http://localhost:8200).

Authentication

Auth provider: The method for authenticating requests to 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

Use the **Authentication method** buttons to select one of the following AWS methods:

- **Auto**: Uses the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`, or the attached IAM role. This authentication method will work **only** when running on AWS.

- **Manual**: If not running on AWS, you can select this option to enter your access key and secret key, directly or by reference. 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.

Assume Role

This section is displayed for all AWS IAM authentication methods.

Enable for Vault Auth: Toggle to `Yes` 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.

AWS EC2 Authentication

Vault Role: Enter the authentication role to use in Vault.

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 LogStream secret should be stored, e.g.: `<somePath>/cribl-secret` .

> ⚠ In a distributed deployment, the Leader, and each Worker Group, require a distinct secret. This location cannot be shared between them.

What's Next

- ❯ Securing Data
- ❯ Access Management

Monitoring

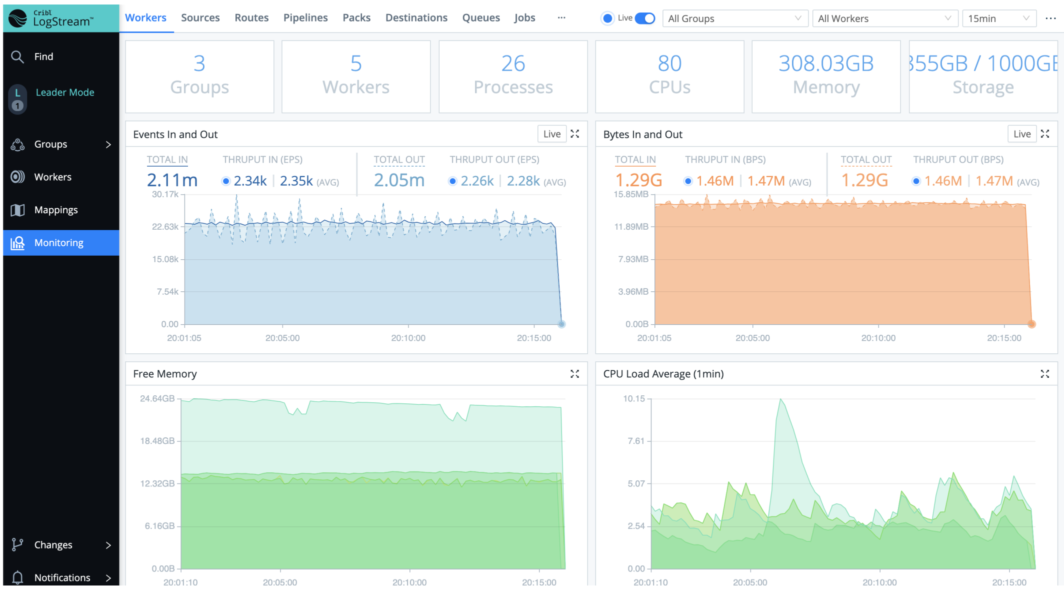
To get an operational view of a Cribl LogStream deployment, you can consult the following resources.

Monitoring Resources

- Monitoring Page
- Internal Logs and Metrics
- Licensing
- Flows (Beta)
- Health Endpoint

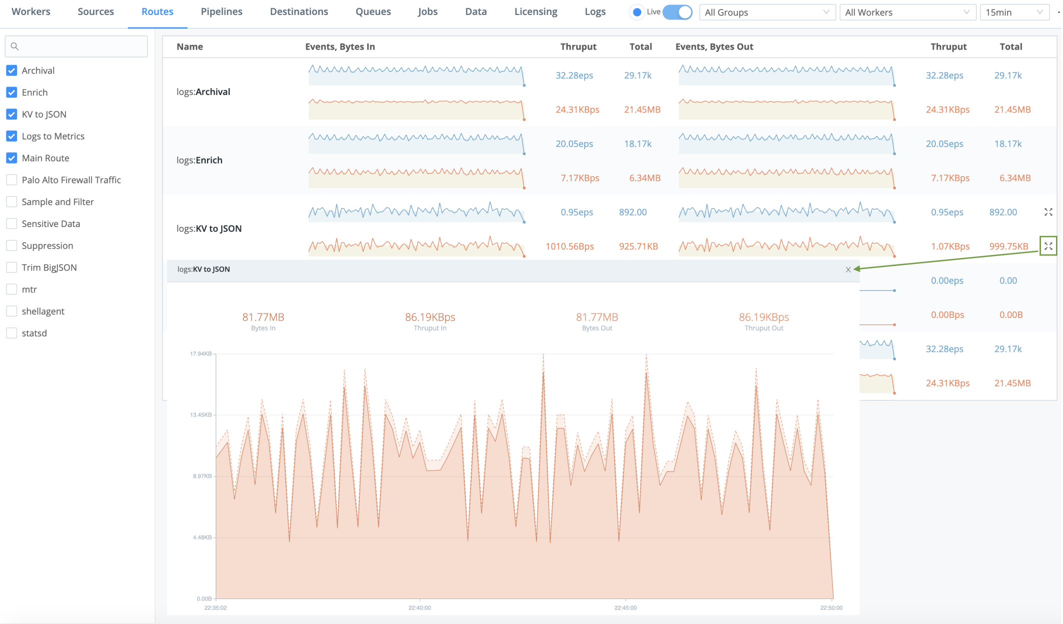
Monitoring Page

Select **Monitoring** from the left nav (distributed deployments) or top nav (single-instance deployments). The resulting **Monitoring** page 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. Coverage is limited to the previous 24 hours. Byte-related charts show the uncompressed size of processed data. Bytes in/out are measured based on the size of _raw (i.e. metrics events will reflect 0 bytes processed because they do not possess a _raw field).



Monitoring page

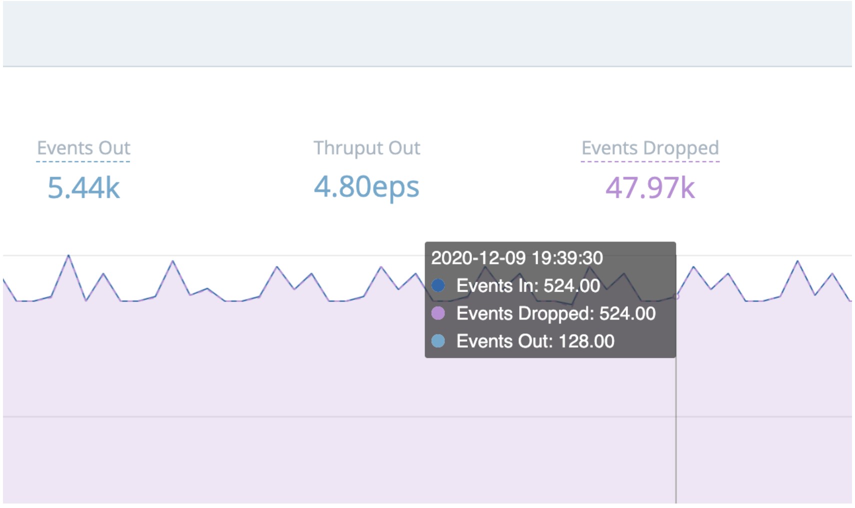
Dense displays are condensed to sparklines for legibility. Hover over the right edge to display Maximize buttons that you can click to zoom these up to detailed graphs.



Name	Events, Bytes In	Thruput	Total	Events, Bytes Out	Thruput	Total
logs:Archival		32.28eps	29.17k		32.28eps	29.17k
		24.31KBps	21.45MB		24.31KBps	21.45MB
logs:Enrich		20.05eps	18.17k		20.05eps	18.17k
		7.17KBps	6.34MB		7.17KBps	6.34MB
logs:KV to JSON		0.95eps	892.00		0.95eps	892.00
		1010.56Bps	925.71KB		1.07KBps	999.75KB
					0.00eps	0.00
					0.00Bps	0.00B
					32.28eps	29.17k
					24.31KBps	21.45MB

Sparklines and fly-out

You can hover over an expanded graph fly-out to display further details.



Throughput details

Internal Logs and Metrics

Select **Logs** from the Monitoring page's top nav or ••• overflow menu. LogStream's internal logs and internal metrics provide comprehensive information about an instance's status/health, inputs, outputs, Pipelines, Routes, Functions, and traffic.

Licensing

Select **Licensing** from the Monitoring page's top nav or ••• overflow menu to check your licenses' expiration dates, daily data throughput quotas, and daily and 90-day trailing daily throughput.

Flows (Beta)

Select **Flows** from the Monitoring page's top nav or ••• overflow menu to see a graphical, left-to-right visualization of data flow through your LogStream deployment.

Health Endpoint

Query this endpoint on any instance to check the instance's health. (Details below.)

Types of Logs

LogStream provides the following log types, by originating process:

- API Server Logs – These logs are emitted primarily by the `API/main` process. They correspond to the top-level `cribl.log` that shows up on the Diag page. These include telemetry/license-validation logs. Filesystem location: `$CRIBL_HOME/log/cribl.log`
- Worker Process(es) Logs – These logs are emitted by all the Worker Processes, and are very common on single-instance deployments and Worker Nodes. Filesystem location: `$CRIBL_HOME/log/worker/N/cribl.log`
- Worker Group Logs – These logs are emitted by all processes that help a Leader Node configure Worker Groups. Filesystem location: `$CRIBL_HOME/log/group/GROUPNAME/cribl.log`

LogStream rotates logs every 5 MB, keeping the most recent 5 logs. In a distributed deployment, all Workers forward their metrics to the Leader Node, which then consolidates them to provide a deployment-wide view.

Forward Logs and Metrics Externally

LogStream supports forwarding internal logs and metrics to your preferred external monitoring solution. To make internal data available to send out, go to **Sources** and enable the Cribl Internal Source.

This will send internal logs and metrics down through Routes and Pipelines, 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.

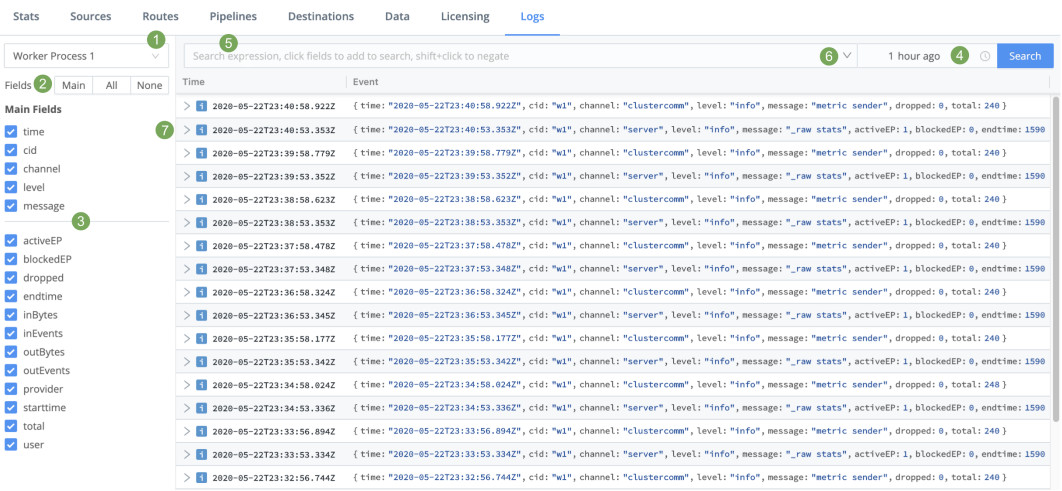
For recommendations about useful Cribl metrics to monitor, see Internal Metrics.

> ℹ CriblMetrics Override
>
> The **Disable field metrics** setting – in global ⚙ **Settings** (lower left) > **System > General Settings > Limits -** applies only to metrics sent to the Leader Node. When the **Cribl Internal** Source is enabled, LogStream ignores this **Disable field metrics** setting, and full-fidelity data will flow down the Routes.

Search Internal Logs

LogStream exists because logs are great and wonderful things! Using its **Monitoring > Logs** page, you can search all LogStream's internal logs at once – from a single location, for both Leader and Worker Nodes. 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):



```
2020-05-22T23:40:58.922Z  { time: "2020-05-22T23:40:58.922Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
2020-05-22T23:40:53.353Z  { time: "2020-05-22T23:40:53.353Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:39:58.779Z  { time: "2020-05-22T23:39:58.779Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
2020-05-22T23:39:53.352Z  { time: "2020-05-22T23:39:53.352Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:38:58.623Z  { time: "2020-05-22T23:38:58.623Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
2020-05-22T23:38:53.353Z  { time: "2020-05-22T23:38:53.353Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:37:58.478Z  { time: "2020-05-22T23:37:58.478Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
2020-05-22T23:37:53.348Z  { time: "2020-05-22T23:37:53.348Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:36:58.324Z  { time: "2020-05-22T23:36:58.324Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
2020-05-22T23:36:53.345Z  { time: "2020-05-22T23:36:53.345Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:35:58.177Z  { time: "2020-05-22T23:35:58.177Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
2020-05-22T23:35:53.342Z  { time: "2020-05-22T23:35:53.342Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:34:58.024Z  { time: "2020-05-22T23:34:58.024Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 248 }
2020-05-22T23:34:53.336Z  { time: "2020-05-22T23:34:53.336Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:33:56.894Z  { time: "2020-05-22T23:33:56.894Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
2020-05-22T23:33:53.334Z  { time: "2020-05-22T23:33:53.334Z", cid: "w1", channel: "server", level: "info", message: "_raw stats", activeEP: 1, blockedEP: 0, endtime: 1590
2020-05-22T23:32:56.744Z  { time: "2020-05-22T23:32:56.744Z", cid: "w1", channel: "clustercomm", level: "info", message: "metric sender", dropped: 0, total: 240 }
```

Logs page (controls highlighted)

1. **Log file selector**: Choose the Node to view. In a Distributed Deployment, this list will be hierarchical, with Workers displayed inside their Leader.

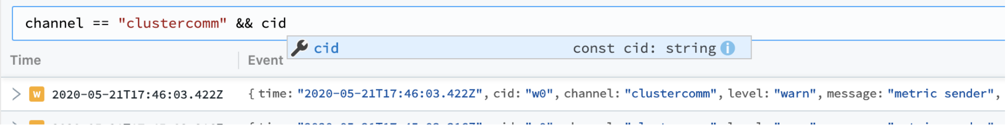
2. **Fields selector**: Click the **Main** | **All** | **None** toggles to quickly select or deselect multiple check boxes below.

3. **Fields**: Select or deselect these check boxes to determine which columns are 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.

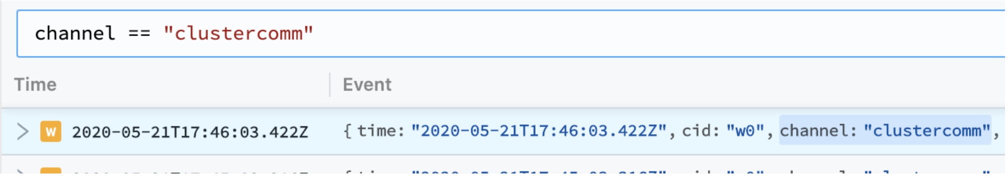
5. **Search box**: To limit the displayed results, enter a JavaScript expression here. An expression must evaluate to `truthy` to return results. You can press **Shift+Enter** to insert a newline.

   Typeahead assist is available for expression completion:



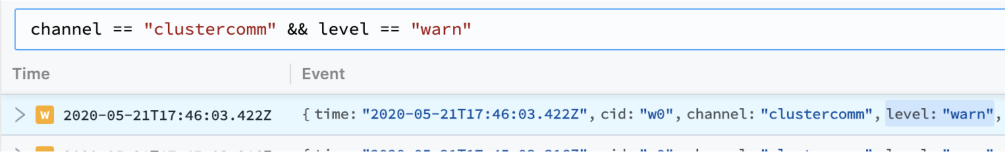
```
channel == "clustercomm" && cid
cid                                 const cid: string
Time                  Event
> W 2020-05-21T17:46:03.422Z  { time: "2020-05-21T17:46:03.422Z", cid: "w0", channel: "clustercomm", level: "warn", message: "metric sender",
```

Click a field in any event to add it to a query:



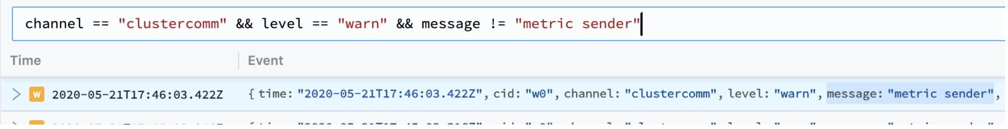
```
channel == "clustercomm"
Time                  Event
> W 2020-05-21T17:46:03.422Z  { time: "2020-05-21T17:46:03.422Z", cid: "w0", channel: "clustercomm",
```

Click other fields to append them to a query:



```
channel == "clustercomm" && level == "warn"
Time                  Event
> W 2020-05-21T17:46:03.422Z  { time: "2020-05-21T17:46:03.422Z", cid: "w0", channel: "clustercomm", level: "warn",
```

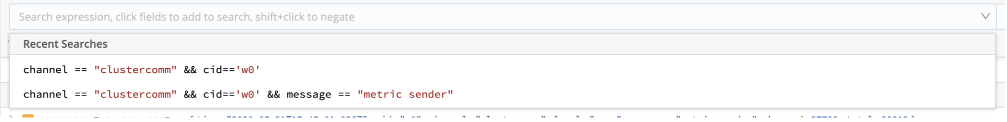
Shift+click to *negate* a field:



```
channel == "clustercomm" && level == "warn" && message != "metric sender"
Time                  Event
> W 2020-05-21T17:46:03.422Z  { time: "2020-05-21T17:46:03.422Z", cid: "w0", channel: "clustercomm", level: "warn", message: "metric sender",
```

> ℹ To modify the depth of information that is originally input to the Logs page, see Logging Settings.

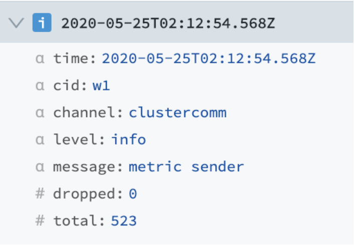
6. Click the Search box's history arrow (right side) to retrieve recent queries:



```
Search expression, click fields to add to search, shift+click to negate
Recent Searches
channel == "clustercomm" && cid=='w0'
channel == "clustercomm" && cid=='w0' && message == "metric sender"
```

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:



```
2020-05-25T02:12:54.568Z
  a time: 2020-05-25T02:12:54.568Z
  a cid: w1
  a channel: clustercomm
  a level: info
  a message: metric sender
  # dropped: 0
  # total: 523
```

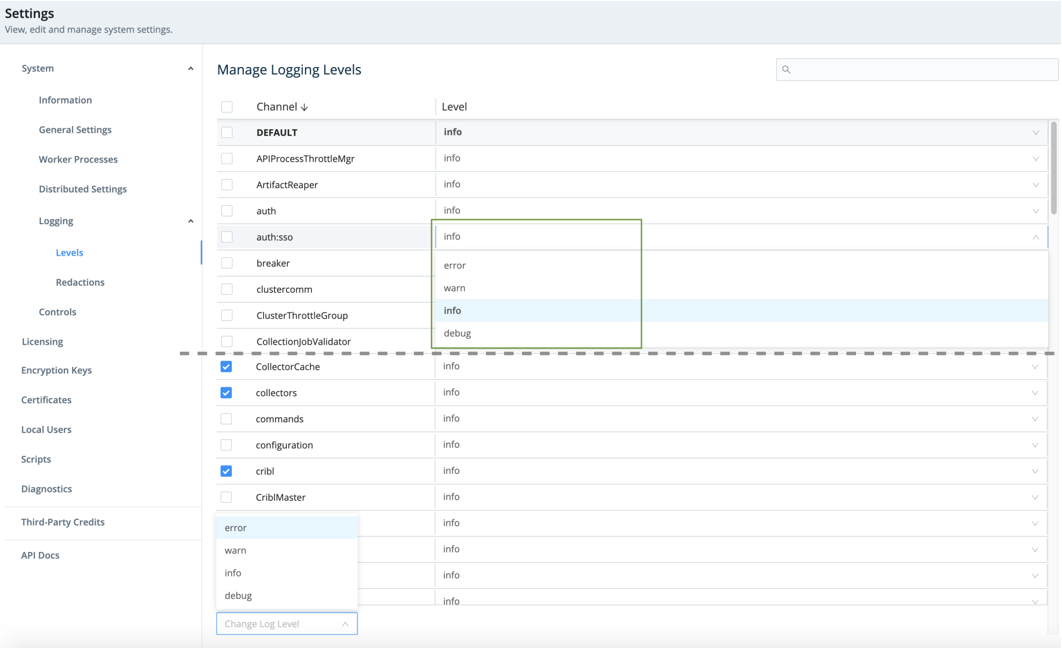
Logging Settings

Through LogStream's global Settings, you can adjust the level (verbosity) of internal logging data processed, per logging channel. You can also redact fields in customized ways.

Change Logging Levels

Select global ⚙ **Settings** (lower left) > **System > Logging > Levels** to open the **Manage Logging Levels** page. Here, you can:

- Modify one channel by clicking its **Level** column. In the resulting drop-down, you can set a verbosity level ranging from **error** up to **debug**. (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.



Settings
View, edit and manage system settings.

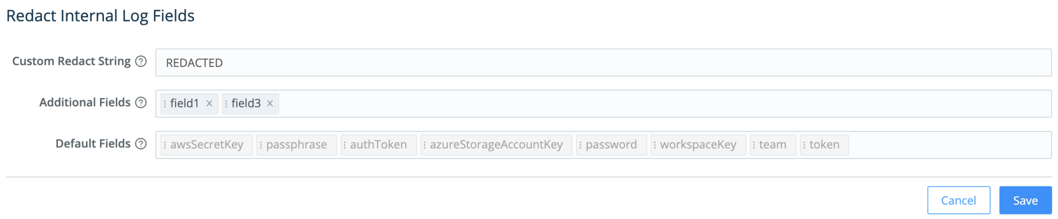
Manage Logging Levels

□	Channel ↓	Level
□	**DEFAULT**	**info**
□	APIProcessThrottleMgr	info
□	ArtifactReaper	info
□	auth	info
□	auth:sso	info
□	breaker	
□	clustercomm	
□	ClusterThrottleGroup	
□	CollectionJobValidator	
☑	CollectorCache	info
☑	collectors	info
□	commands	info
□	configuration	info
☑	cribl	info
□	CriblMaster	info
		info
		info
		info
		info

Manage Logging Levels page

Change Logging Redactions

Select global ⚙ **Settings** (lower left) > **System > Logging > Redactions**: to open the **Redact Internal Log Fields** page. Here, you can customize the redaction of sensitive, verbose, or just ugly data within LogStream's internal logs.



Redact Internal Log Fields

Custom Redact String ⓘ REDACTED

Additional Fields ⓘ field1 × field3 ×

Default Fields ⓘ awsSecretKey passphrase authToken azureStorageAccountKey password workspaceKey team token

Redact Internal Log Fields page

It's easiest to understand this page's fields from bottom to top:

- **Default fields**: LogStream always redacts these fields. You can't modify this list.
- **Additonal fields**: Type or paste in the names of other 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 LogStream's default redaction pattern, explained below.

Default Redact String

By default, LogStream transforms this page's selected fields by applying the following redaction pattern:

- Echo the field value's first two characters.
- Replace all intermediate characters with a literal `...` ellipsis.
- Echo the value's last two characters.

Anything you enter in the **Custom redact string** field will override this default `⁇...⁇` pattern.

Health Endpoint

Each LogStream instance exposes a `health` endpoint – typically used in conjunction with a Load Balancer – that you can use to make operational decisions.

Health Check Endpoint	Healthy Response	LogSt Versic
`curl http(s)://<host>:<port>/api/v1/health`	`{"status":"healthy"}`	Throu 2.4.3
`curl http(s)://<host>:<port>/api/v1/health`	{"status":"healthy","startTime":1617814717110} (see details below)	2.4.4 a later

Specifically, the `health` endpoint can return one of the following response codes:

200 – healthy.
400 – an auth token was provided, but does not match any provisioned token.
503 – server busy: too many concurrent connections (configurable).

Internal Metrics

When sending LogStream 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 a Worker Node is having a problem, a Node is down, a Destination is down, a Source stops providing incoming data, etc.

LogStream reports its internal metrics within the LogStream UI (in the same way that it reports internal logs at **Monitoring** > **Logs**). To expose metrics for capture or routing, enable the **Cribl Internal** Source > **CriblMetrics** section.

By default, LogStream 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 LogStream. 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.

Total Throughput Metrics

Five important metrics below are prefixed with `total.` These power the top of LogStream'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` (new in LogStream 2.4) – helpful for discovering situations such as: you've disabled a Destination without noticing.

Interpreting Total Metrics

These `total.` metrics' values could reflect LogStream's health, but could also report low activity simply due to the Source system. E.g., 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 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; `total.in_events|#input=bar` would be another.

System Health Metrics

Five specific metrics are most valuable for monitoring system health. The first two are LogStream composite metrics; the remaining three report on your hardware or VM infrastructure.

- `health.inputs`
- `health.outputs` – see the JSON Examples below for both `health.` 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.)

All of the above metrics take these three 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 LogStream-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": { "ci": "http:http",   "input": "http:http" }}, "val": 0},
  { "model": { "ci": "cribl:CriblLogs",   "input": "cribl:CriblLogs" }},
  { "model": { "ci": "cribl:CriblMetrics",   "input": "cribl:CriblMetric
  { "model": { "ci": "datagen:DatagenWeblog",   "input": "datagen:Datage
  ],
"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 }
  ],
```

Worker Resource Metrics

As of LogStream 2.4.4, the Cribl Internal Source reports useful two metrics on individual Worker Processes' resource usage:

- system.cpu_perc) – CPU percentage usage.
- system.mem_rss – RAM usage.

Persistent Queue Metrics

Five metrics below are valuable for monitoring Persistent Queues' behavior:

- pq.queue_size
- pq.in_bytes
- pq.in_events
- pq.out_events
- pq.out_bytes

These are aggregate metrics. But you can distinguish unique metrics per queue Destination, using the #output=<id> dimension. For example, pq.out_events|#output=kafka would be one unique metric; pq.out_events|#output=camus would be another.

Other Internal Metrics

The Cribl Internal Source emits other metrics that can be useful in downstream dashboards for understanding LogStream's behavior and health. These include:

- `cribl.logstream.total.activeCxn` – Total active inbound TCP connections.
- `cribl.logstream.pipe.in_events` – Inbound events per Pipeline.
- `cribl.logstream.pipe.out_events` – Outbound events per Pipeline.
- `cribl.logstream.pipe.dropped_events` – Dropped events per Pipeline.
- `cribl.logstream.metrics_pool.num_metrics` – The total number of unique metrics that have been allocated into memory.
- `cribl.logstream.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.
- `cribl.logstream.cluster.metrics.sender.inflight` – Number of metric packets currently being sent from a Worker Process to the API Process, via IPC (interprocess communication).
- `cribl.logstream.blocked.outputs` – Blocked Destinations.
- `cribl.logstream.pq.queue_size` – Current queue size, per Destination, per Worker Process.
- `cribl.logstream.host.in_bytes` – Inbound bytes from a given host (host is a characteristic of the data).
- `cribl.logstream.host.in_events` – Inbound events from a given host (host is a characteristic of the data).
- `cribl.logstream.host.out_bytes` – Outbound bytes from a given host (host is a characteristic of the data).
- `cribl.logstream.host.out_events` – Outbound events from a given host (host is a characteristic of the data).
- `cribl.logstream.route.in_bytes` – Inbound bytes per Route.
- `cribl.logstream.route.in_events` – Inbound events per Route.
- `cribl.logstream.route.out_bytes` – Outbound bytes per Route.
- `cribl.logstream.route.out_events` – Outbound events per Route.
- `cribl.logstream.sourcetype.in_bytes` – Inbound bytes per sourcetype.

- `cribl.logstream.sourcetype.in_events` – Inbound events per sourcetype.
- `cribl.logstream.sourcetype.out_bytes` – Outbound bytes per sourcetype.
- `cribl.logstream.sourcetype.out_events` – Outbound events per sourcetype.

Other Metrics Endpoints and Dimensions

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 LogStream's primary public metrics endpoint, which returns most internal metrics. Note that 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 LogStream UI at global ⚙ **Settings** (lower left) > **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 of determining `loadavg`, but it requires you to query all your CPUs individually, and then average. In the example below, `Idle = user + nice + sys`.

```
"cpus": [{ "times": {
  "user": 19881260,
  "nice": 39370,
  "sys": 5250130,
  "idle": 76088790,
```

`cribl_wp` Metric Dimension

`cribl_wp` is a useful dimension that identifies the Worker Process that processed each event.

Upgrading

This page outlines how to upgrade a Cribl LogStream single-instance or distributed deployment along one of the following supported upgrade paths:

- v2.x ==> v2.x || v3.x
- v1.7.x/v2.0.x ==> v2.x.x || v3.x
- v1.6.x or below ==> v1.7.x ==> v2.x.x || v3.x

> ⚠ See notes on Upgrading to LogStream 2.3 from Prior Versions below.
>
> LogStream does **not** support direct upgrades from a Beta to a GA version. To get the GA version running, you must perform a new install.

Standalone/Single-Instance

This path requires upgrading only the single/standalone node:

1. Stop LogStream.

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 LogStream.

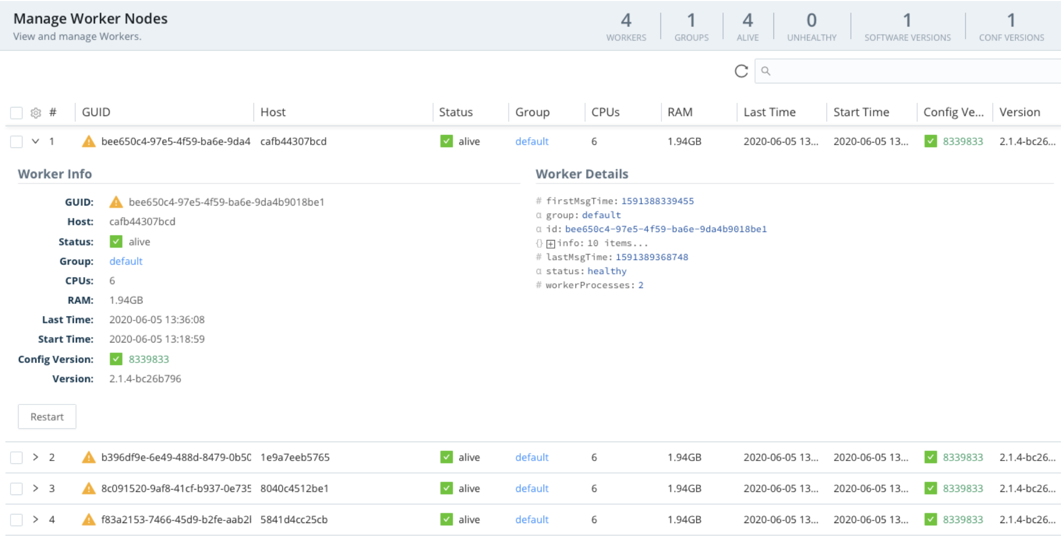
Distributed Deployment

For a distributed deployment, the order of upgrade is: Upgrade first the Leader Node, then upgrade the Worker Nodes, then commit and deploy the changes on the Leader.

Upgrade the Leader Node

1. Commit and deploy your desired last version. (This will be your most recent checkpoint.)
   - Optionally, `git push` to your configured remote repo.
2. Stop Cribl LogStream.
   - 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 LogStream version on top of the old one.
4. Restart LogStream and log back in.
5. Wait for all the Worker Nodes to report to the Leader, and ensure that they are correctly reporting the last committed configuration version.

> i Workers' UI will not be available until the Worker version has been upgraded to match the version on the Leader. Errors like those below will appear until the Worker nodes are upgraded.



Manage Worker Nodes

View and manage Workers.

4	1	4	0	1	1
WORKERS	GROUPS	ALIVE	UNHEALTHY	SOFTWARE VERSIONS	CONF VERSIONS

#	GUID	Host	Status	Group	CPUs	RAM	Last Time	Start Time	Config Ve...	Version
1	⚠ bee650c4-97e5-4f59-ba6e-9da4	cafb44307bcd	✓ alive	default	6	1.94GB	2020-06-05 13...	2020-06-05 13...	✓ 8339833	2.1.4-bc26...

Worker Info

GUID:	⚠ bee650c4-97e5-4f59-ba6e-9da4b9018be1
Host:	cafb44307bcd
Status:	✓ alive
Group:	default
CPUs:	6
RAM:	1.94GB
Last Time:	2020-06-05 13:36:08
Start Time:	2020-06-05 13:18:59
Config Version:	✓ 8339833
Version:	2.1.4-bc26b796

Worker Details

```
# firstMsgTime: 1591388339455
α group: default
α id: bee650c4-97e5-4f59-ba6e-9da4b9018be1
{} ⊞info: 10 items...
# lastMsgTime: 1591389368748
α status: healthy
# workerProcesses: 2
```

#	GUID	Host	Status	Group	CPUs	RAM	Last Time	Start Time	Config Ve...	Version
2	⚠ b396df9e-6e49-488d-8479-0b5C	1e9a7eeb5765	✓ alive	default	6	1.94GB	2020-06-05 13...	2020-06-05 13...	✓ 8339833	2.1.4-bc26...
3	⚠ 8c091520-9af8-41cf-b937-0e735	8040c4512be1	✓ alive	default	6	1.94GB	2020-06-05 13...	2020-06-05 13...	✓ 8339833	2.1.4-bc26...
4	⚠ f83a2153-7466-45d9-b2fe-aab2l	5841d4cc25cb	✓ alive	default	6	1.94GB	2020-06-05 13...	2020-06-05 13...	✓ 8339833	2.1.4-bc26...

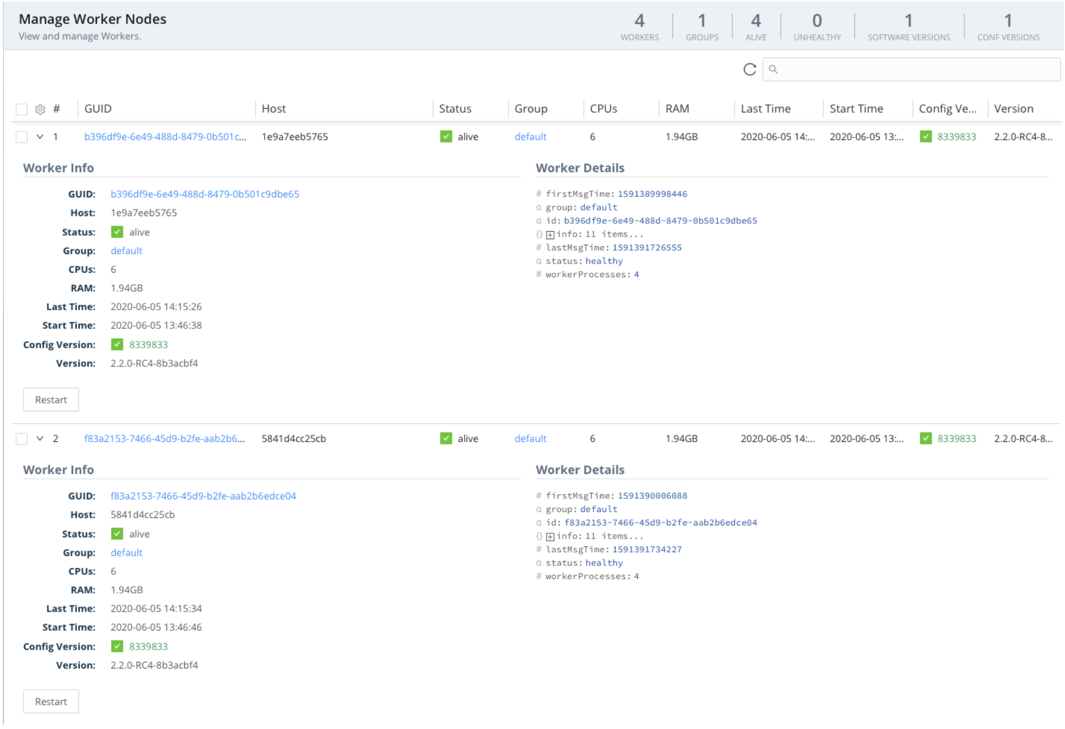
Upgrade the Worker Nodes

These are the same basic steps as when upgrading a Single Instance, above:

1. Stop Cribl LogStream on each Worker Node.
2. Uncompress the new version on top of the old one.
3. Restart LogStream.

Commit and Deploy 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 configuration **only after all** Workers have upgraded.



Manage Worker Nodes
View and manage Workers.

4	1	4	0	1	1
WORKERS	GROUPS	ALIVE	UNHEALTHY	SOFTWARE VERSIONS	CONF VERSIONS

□	#	GUID	Host	Status	Group	CPUs	RAM	Last Time	Start Time	Config Ve...	Version
□ ∨	1	b396df9e-6e49-488d-8479-0b501c...	1e9a7eeb5765	✓ alive	default	6	1.94GB	2020-06-05 14:...	2020-06-05 13:...	✓ 8339833	2.2.0-RC4-8...

Worker Info

GUID: b396df9e-6e49-488d-8479-0b501c9dbe65
Host: 1e9a7eeb5765
Status: ✓ alive
Group: default
CPUs: 6
RAM: 1.94GB
Last Time: 2020-06-05 14:15:26
Start Time: 2020-06-05 13:46:38
Config Version: ✓ 8339833
Version: 2.2.0-RC4-8b3acbf4

Worker Details

```
# firstMsgTime: 1591389998446
α group: default
α id: b396df9e-6e49-488d-8479-0b501c9dbe65
{} ⊞info: 11 items...
# lastMsgTime: 1591391726555
α status: healthy
# workerProcesses: 4
```

□	#	GUID	Host	Status	Group	CPUs	RAM	Last Time	Start Time	Config Ve...	Version
□ ∨	2	f83a2153-7466-45d9-b2fe-aab2b6...	5841d4cc25cb	✓ alive	default	6	1.94GB	2020-06-05 14:...	2020-06-05 13:...	✓ 8339833	2.2.0-RC4-8...

Worker Info

GUID: f83a2153-7466-45d9-b2fe-aab2b6edce04
Host: 5841d4cc25cb
Status: ✓ alive
Group: default
CPUs: 6
RAM: 1.94GB
Last Time: 2020-06-05 14:15:34
Start Time: 2020-06-05 13:46:46
Config Version: ✓ 8339833
Version: 2.2.0-RC4-8b3acbf4

Worker Details

```
# firstMsgTime: 1591390006088
α group: default
α id: f83a2153-7466-45d9-b2fe-aab2b6edce04
{} ⊞info: 11 items...
# lastMsgTime: 1591391734227
α status: healthy
# workerProcesses: 4
```

Post-2.1.4 upgrade to 2.2

Upgrade and Rollback via the UI (Beta Feature)

LogStream v.2.4.4 and higher provide streamlined options to upgrade the Leader Node (or single instance), as well as Worker Nodes, directly through the UI. In LogStream 3.0 or higher, go to global ⚙ **Settings** (lower left) > **System > Upgrade**. These streamlined controls perform the whole above sequence of stopping the LogStream server, updating the installed package, and restarting LogStream.

LogStream 3.0 (or higher) also enables you to manage automatic backup and rollback in case an upgrade fails.

> These options are still experimental. As the UI warns: Use them in production at your own risk, and always back up your current installation before proceeding. These options will work only if all LogStream instances (including Worker Processes) start at v.2.4.4 or higher.
>
> Also, be aware that the **Checking for upgrade** status message, and its accompanying spinner, can take up to several minutes to resolve. Also, after you initiate an upgrade, it can take up to several minutes before the **View** button (described below) is displayed.

The following controls are available:

Package source: The default **CDN** button downloads a package directly from Cribl's content delivery network. Selecting the alternative **Path** button exposes these additional controls:

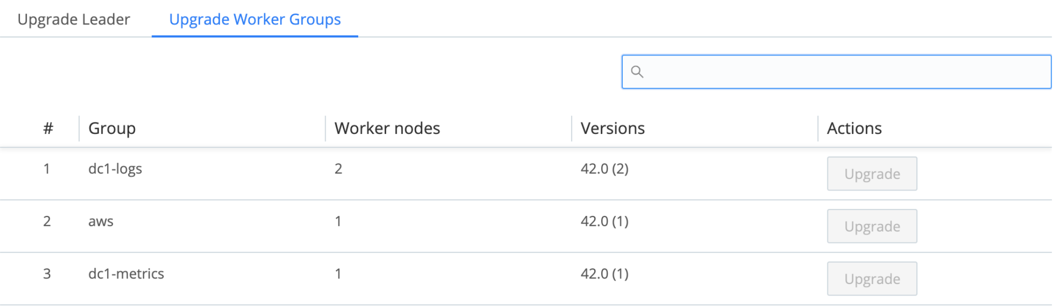
- **Package location**: Enter either a URL (HTTP) or a local path to the upgrade package.
- **Package hash location**: Enter either a URL (HTTP) or a local path to the hash that validates the package. Supports sha256 and md5 formats. (You can simply append `.sha256` to the contents of the **Package location** field.)
- **Save**/**Cancel** buttons: Click **Save** to store the specified locations. Clicking **Cancel** restores the **CDN** package-source selection.

Upgrade/Upgrade Leader

In a Single-instance deployment, the **Upgrade** button is the only other control provided. In a Distributed deployment, the **Upgrade** button is displayed on an **Upgrade Leader** tab, and clicking it upgrades the Leader Node. (As with manual upgrades, always upgrade the Leader before upgrading the Workers.)

Upgrade Worker Groups

This second tab, displayed only in distributed deployments, shows each Worker Group's status.



#	Group	Worker nodes	Versions	Actions
1	dc1-logs	2	42.0 (2)	Upgrade
2	aws	1	42.0 (1)	Upgrade
3	dc1-metrics	1	42.0 (1)	Upgrade

Upgrade Worker Groups tab

> i Upgrading Workers from the Leader requires a LogStream Standard or Enterprise license.

Click any row's **Upgrade** button to upgrade that group. The resulting resulting **Upgrade Group** dialog offers two states.

Basic Upgrade Configuration

In this default **Upgrade Group** dialog, you can simply upgrade the whole Group, by clicking the dialog's **Upgrade** button to confirm. (Note that LogStream will check to ensure that Workers are upgraded no higher than the Leader's version.)

Advanced Upgrade Configuration

Click **Advanced configuration** to expose these additional options:

Quantity %: Specify what percentage of the Group's Workers to upgrade in this operation. If you enter a value less than the default `100` %, LogStream will perform a partial upgrade, keeping the remaining Workers active to process data.

Rolling upgrade: Toggle this slider on to upgrade Workers one at a time. Enabling the slider also enables the dialog'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 click the **Upgrade** confirmation button, the **Upgrade Worker Groups** tab will display an additional button on this Group's row:

View: Click to display the upgrade task's status in the Job Inspector modal – select that modal's **System** tab to access details.

Backup and Rollback

By default, LogStream will automatically roll back to a stored backup package if an upgrade (initiated through the UI) fails. You can adjust this behavior at global ⚙ **Settings** (lower left) > **System** > **General Settings** > **Upgrade & Share Settings**, using the following controls.

> ⚠ LogStream can perform rollbacks only on Worker Nodes/instances that started on at least LogStream v. 3.0.0, before the attempted upgrade.

Enable automatic rollback: LogStream will automatically roll back an upgrade if the LogStream server fails to start, or if the Worker Node fails to connect to the Leader. (Toggle to `No` 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, i.e., 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, i.e., 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`.

Upgrading to LogStream 2.3 from Prior Versions

As of version 2.3, LogStream Free and One licenses are permanent, but they enforce certain restrictions that especially affect distributed deployments:

- Even if you have more than one Worker Group defined, only one Worker Group will be visible and usable.
  - This will be the first Group listed in `$CRIBL_HOME/local/cribl/groups.yml` – typically, the `default` Group. You can edit `groups.yml` to move the desired Group to the top.
- Your cluster will be limited to 10 Worker Processes across all Worker Nodes.
  - LogStream will balance (or rebalance) these Processes as evenly as possible across the Worker Nodes.
- Authentication will fall back to local authorization. You will not be able to authenticate via Splunk, LDAP, or SSO/OpenID.
- **Git Push** to remote repos will not be supported through the product.

> ⚠ If you are upgrading LogStream Free or LogStream One from version 2.2.x or lower, these changes might require you to adjust your existing configuration and/or workflows.
>
> See Licensing for details on all current license options.

As of LogStream 2.3, licenses no longer need to be deployed directly to Worker Groups. The Leader will push license information down to Worker Groups as part of the heartbeat.

Splunk App Package Upgrade Steps

> ⚠ See Deprecation note for v.2.1.

Follow these steps to upgrade from v.1.7, or higher, 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.

Upgrading 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. Convert to HF mode by running: `$SPLUNK_HOME/etc/apps/cribl/bin/cribld mode-hwf`
4. Restart Splunk.

Uninstalling

Uninstalling the Standalone Version

- Stop Cribl LogStream (stopping the main process).
- Back up necessary configurations/data.
- Remove the directory where Cribl LogStream is installed.

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).

WORKING WITH DATA

Event Model

All data processing in Cribl LogStream is based on discrete data entities commonly known as **events**. An event is defined as 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 LogStream event is as follows:

Cribl LogStream Event Model

```
{
  "_raw": "<body of non-JSON parse-able event>",
  "_time": "<timestamp in UNIX epoch format>",
  "__inputId": "<Id/Name of Source that delivered the event>",
  "__other1": "<Internal field1>",
  "__other2": "<Internal field2>",
  "__otherN": "<Internal fieldN>",
  "key1": "<value1>",
  "key2": "<value2>",
  "keyN": "<valueN>",
  "...": "..."
}
```

Some notes about these representative fields:

- Fields that start with a double-underscore are known as **internal fields**, and each Source can add one or many to each event. For example, Syslog adds both a `__inputId` and a `__srcIpPort` field. Internal fields are used only within Cribl LogStream, and are not passed down to Destinations.

- Upon arrival from a Source, if an event cannot be JSON-parsed, all of its content will be assigned to `_raw`.

- If a timestamp is not configured to be extracted, the current time (in UNIX epoch format) will be assigned to `_time`.

Using Capture

One way to see what an event looks like as it travels through the system is to use the **Capture** feature. While in Preview (right pane):

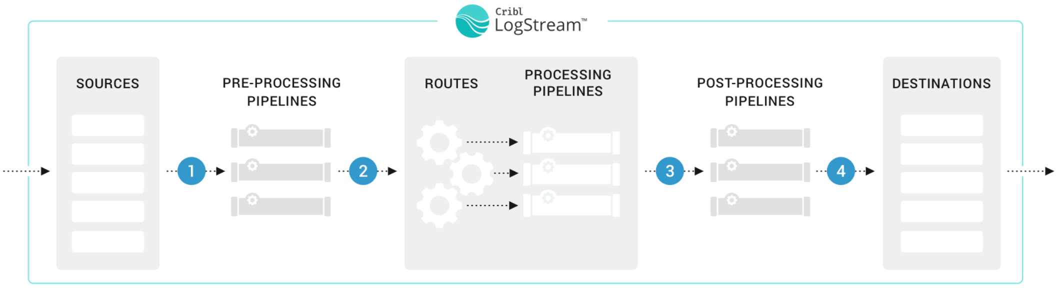
1. Click **Start a Capture.**

2. In the resulting modal, enter a **Filter expression** to narrow down the events of interest.

3. Click **Capture...** and (optionally) change the default Time and/or Event limits.

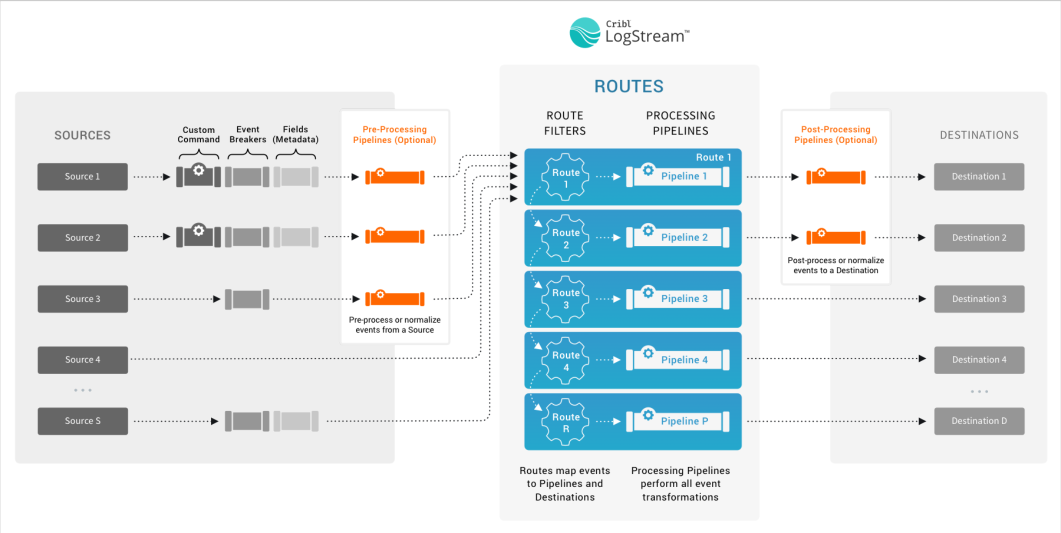
4. Select the desired **Where to capture** option. There are four options:

- **1. Before the pre-processing Pipeline** – Capture events right after they're delivered by the respective Input.
- **2. Before the Routes** – Capture events right after the pre-processing Pipeline, before they go down the Routes.
- **3. Before the post-processing Pipeline** – Capture events right after the Processing Pipeline that actually handled them, before any post-processing Pipeline.
- **4. Before the Destination** – Capture events right after the post-processing Pipeline, before they go out to the configured Destination.



Event Processing Order

The expanded schematic below shows how all events in the Cribl LogStream ecosystem are processed linearly, from left to right.



LogStream in great detail

Here are the stages of event processing:

1. Sources: Data arrives from your choice of external providers. (LogStream supports Splunk, HTTP/S, Elastic Beats, Amazon Kinesis/S3/SQS, Kafka, TCP raw or JSON, and many others.)

2. Custom command: Optionally, 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.

4. 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 LogStream'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.

5. Pre-processing Pipeline: Optionally, you can use a single Pipeline to condition (normalize) data from this input before the data reaches the Routes.

6. 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.

7. Processing Pipelines perform all 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.

8. Post-processing Pipeline: Optionally, you can append a Pipeline a to condition (normalize) data from each Processing Pipeline before the data reaches its Destination.

9. Destinations: Each Route/Pipeline combination forwards processed data to your choice of streaming or storage Destination. (LogStream supports Splunk, Syslog, Elastic, Kafka/Confluent, Amazon S3, Filesystem/NFS, and many other options.)

> ℹ 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.

Routes

What Are Routes

Before incoming events are transformed by a processing Pipeline, Cribl LogStream uses a set of filters to first select a **subset** of events to deliver to the correct Pipeline. This selection is made via Routes.

Accessing Routes

Select **Routes** from LogStream's global top nav (single-instance deployments) or from a Worker Group's top nav (distributed deployments). To configure a new Route, click **+ Route**.

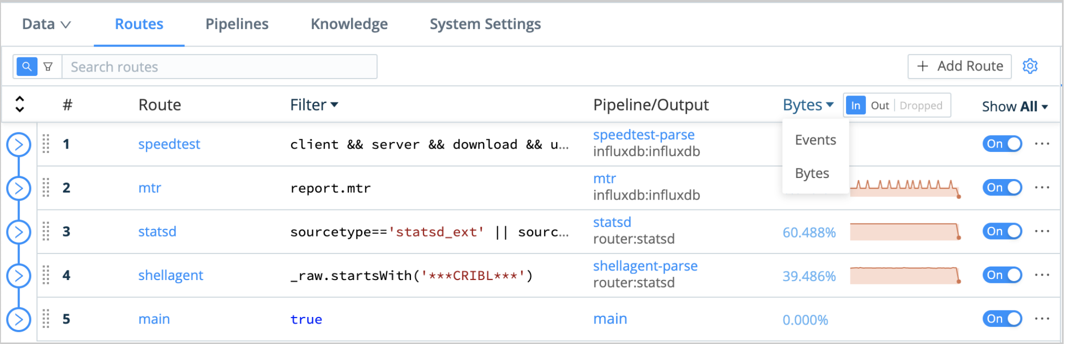
How Do Routes Work

Routes apply filter expressions on incoming events to send matching results to the appropriate Pipeline. Filters are JavaScript-syntax–compatible expressions that are configured with each Route. Examples are:

- `true`
- `source=='foo.log' && fieldA=='bar'`

> i There can be multiple Routes in the system, but each Route can be associated with only **one** Pipeline.

Routes are evaluated in their display order, top->down. The stats shown in the **Bytes**/**Events** (toggle) column are for the most-recent 15 minutes.



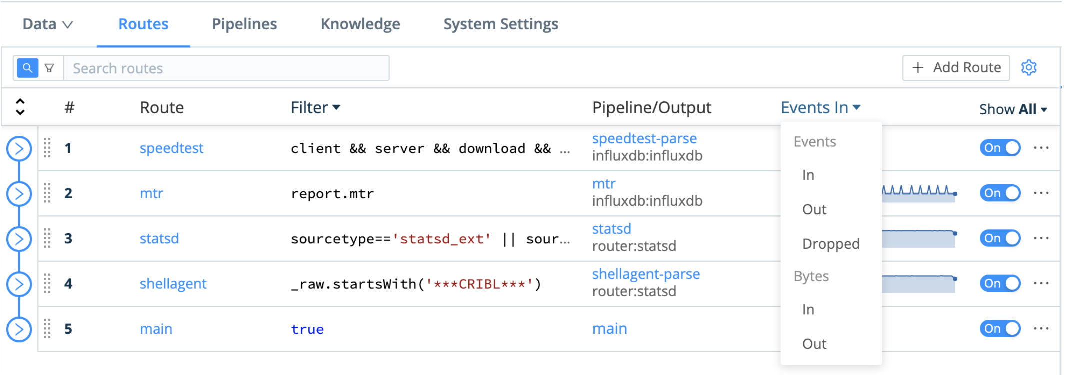
#	Route	Filter	Pipeline/Output	Bytes
1	speedtest	`client && server && download && u…`	speedtest-parse influxdb:influxdb	
2	mtr	`report.mtr`	mtr influxdb:influxdb	
3	statsd	`sourcetype=='statsd_ext' || sourc…`	statsd router:statsd	60.488%
4	shellagent	`_raw.startsWith('***CRIBL***')`	shellagent-parse router:statsd	39.486%
5	main	`true`	main	0.000%

Routes and bytes

In the example above, incoming events will be evaluated first against the Route named **speedtest**, then against **mtr**, then against **statsd**, and so on. At the end, the **main** Route serves as a catch-all for any event that does not match any of the other Routes.

Above, note the selectors to toggle between displaying Events versus Bytes, and to display In versus Out.

When you condense the Routes page to a narrower viewport, LogStream consolidates the **In**/**Out**/**Dropped** selectors onto an expanded **Bytes**/**Events** drop-down menu, as shown below.



#	Route	Filter	Pipeline/Output	Events In
1	speedtest	`client && server && download && …`	speedtest-parse influxdb:influxdb	
2	mtr	`report.mtr`	mtr influxdb:influxdb	
3	statsd	`sourcetype=='statsd_ext' || sour…`	statsd router:statsd	
4	shellagent	`_raw.startsWith('***CRIBL***')`	shellagent-parse router:statsd	
5	main	`true`	main	

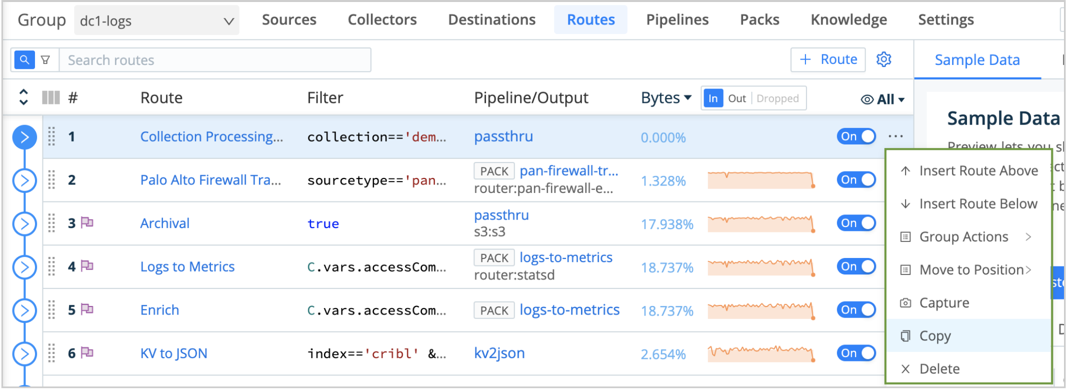
Routes and events (combined menu)

Managing the Routes Page

To apply a Route before another, simply drag it vertically. Use the sliders to turn Routes **On**/**Off** inline, as necessary, to facilitate development and debugging.

You can press the `]` (right-bracket) shortcut key to toggle between the Preview pane and the expanded Routes display shown above. (This works when no field has focus.)

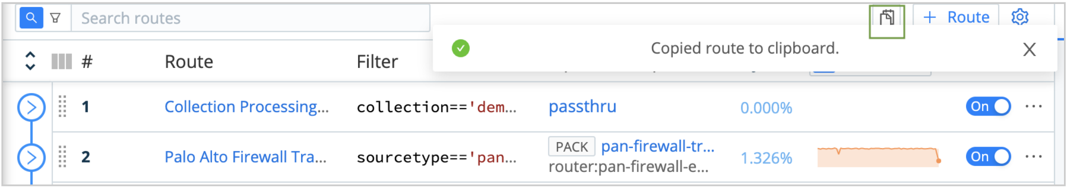
Click a Route's Options (**...**) menu to display multiple options for inserting, grouping, moving, copying, or deleting Routes, as well as for capturing sample data through the selected Route.



#	Route	Filter	Pipeline/Output	Bytes	
1	Collection Processing...	collection=='dem...	passthru	0.000%	On
2	Palo Alto Firewall Tra...	sourcetype=='pan...	PACK pan-firewall-tr... router:pan-firewall-e...	1.328%	On
3	Archival	true	passthru s3:s3	17.938%	On
4	Logs to Metrics	C.vars.accessCom...	PACK logs-to-metrics router:statsd	18.737%	On
5	Enrich	C.vars.accessCom...	PACK logs-to-metrics	18.737%	On
6	KV to JSON	index=='cribl' &...	kv2json	2.654%	On

Route > Options menu

Copying a Route displays the confirmation message and the (highlighted) Paste button shown below.

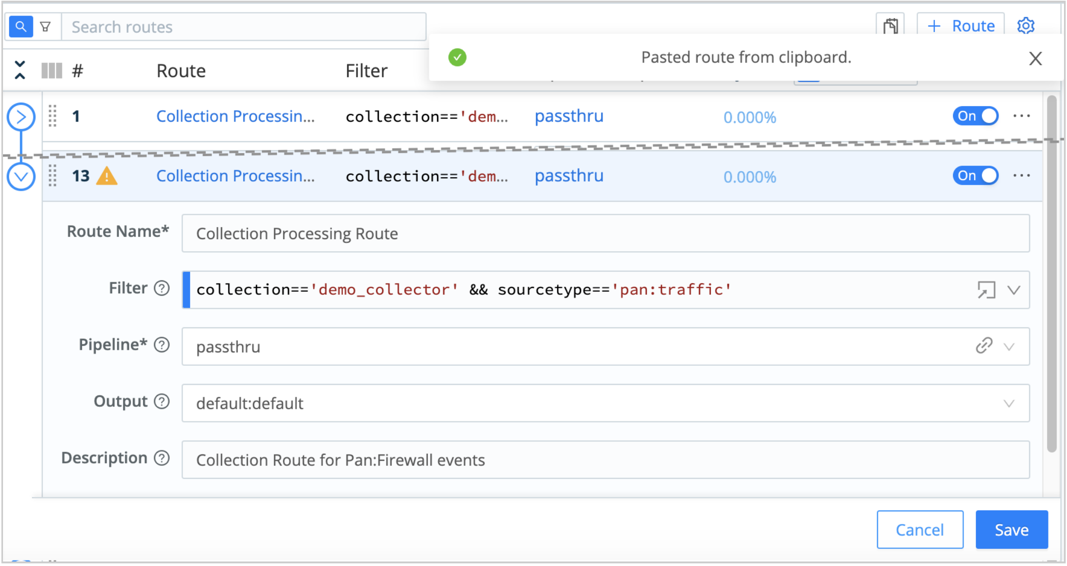


Copied route to clipboard.

#	Route	Filter			
1	Collection Processing...	collection=='dem...	passthru	0.000%	On
2	Palo Alto Firewall Tra...	sourcetype=='pan...	PACK pan-firewall-tr... router:pan-firewall-e...	1.326%	On

Paste button for copied Route

Pasting creates an exact duplicate of the Route, with a warning indicator to change its duplicate name.



Pasted route from clipboard.

#	Route	Filter			
1	Collection Processin...	collection=='dem...	passthru	0.000%	On
13 ⚠	Collection Processin...	collection=='dem...	passthru	0.000%	On

Route Name*: Collection Processing Route

Filter: `collection=='demo_collector' && sourcetype=='pan:traffic'`

Pipeline*: passthru

Output: default:default

Description: Collection Route for Pan:Firewall events

Pasted duplicate Route

Output Destination

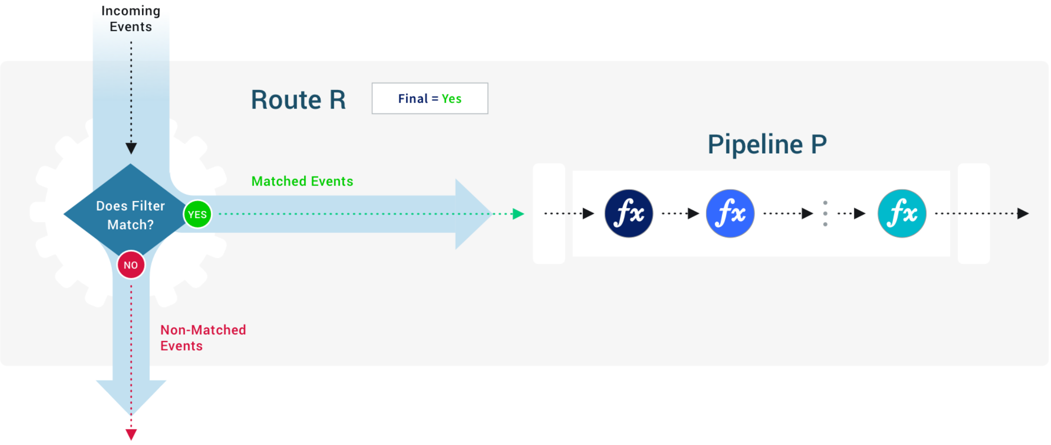
You can configure each Route with an output Destination that denotes where to send events after they're processed by the Pipeline.

The `Final` Toggle

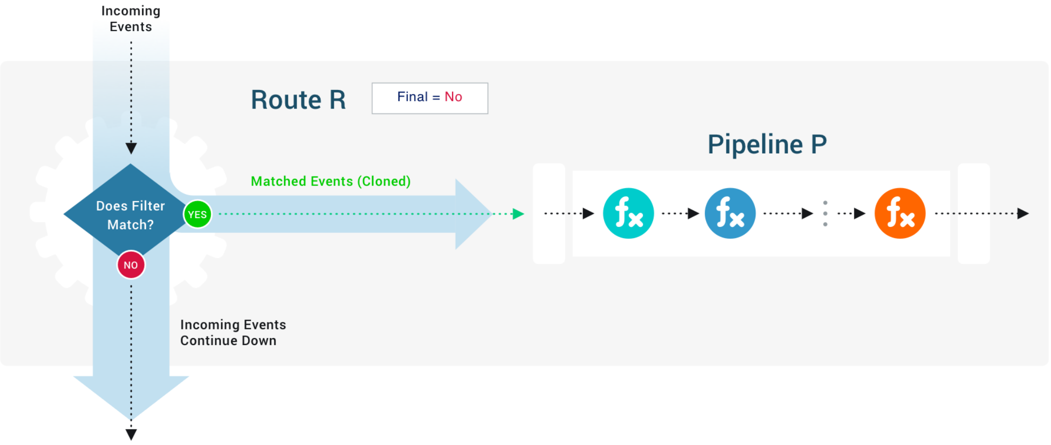
When an event that enters the system and matches a Route-Pipeline pair, it will usually be either:

- Dropped by a function, or
- Transformed (optionally) and exit the system.

This behavior is ensured by the `Final` toggle in Route settings. It defaults to `Yes`, meaning that matched events will be **consumed** by that Route, and will not be evaluated against any other Routes that sit below it.



If the `Final` toggle is set to `No`, clone(s) of the matching events will be processed by the configured Pipeline, and the original events will be allowed to continue their trip to be evaluated and/or processed by other Route-Pipeline pairs.



Final Flag and Cloning Considerations

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 cloning is not needed at all (i.e., all `Final` toggles stay at default), then it makes sense to start with the broadest expression at the top, so as to consume as many events as early as possible.
- If cloning is needed on a narrow set of events, then it might make sense to do that upfront, and follow it with a Route that consumes those clones immediately after.

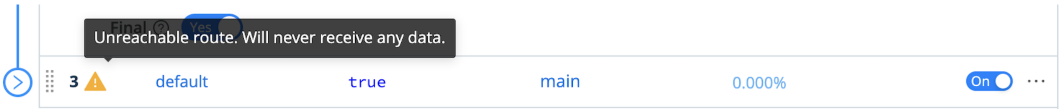
Route Groups

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.

> i 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.



Final ⓘ Yes

Unreachable route. Will never receive any data.

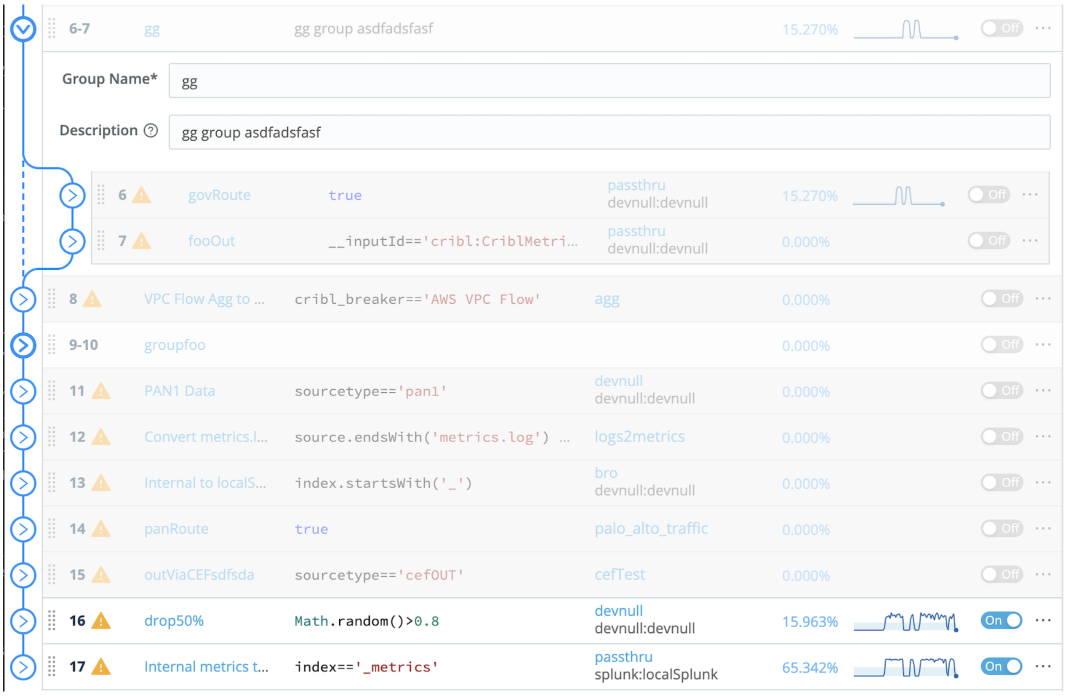
3 ⚠	default	`true`	main	0.000%	On ···

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 (slider is set to `On`).
- Previous Route is final (**Final** slider is set to `Yes`).
- Previous Route's **Filter** expression evaluates to true, (e.g., `true`, `1 == 1`, etc.).

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).



6-7	gg	gg group asdfadsfasf		15.270%	Off ···

Group Name* gg

Description ⓘ gg group asdfadsfasf

6 ⚠	govRoute	`true`	passthru devnull:devnull	15.270%	Off ···
7 ⚠	fooOut	`__inputId=='cribl:CriblMetri...`	passthru devnull:devnull	0.000%	Off ···
8 ⚠	VPC Flow Agg to ...	`cribl_breaker=='AWS VPC Flow'`	agg	0.000%	Off ···
9-10	groupfoo			0.000%	Off ···
11 ⚠	PAN1 Data	`sourcetype=='pan1'`	devnull devnull:devnull	0.000%	Off ···
12 ⚠	Convert metrics.l...	`source.endsWith('metrics.log') ...`	logs2metrics	0.000%	Off ···
13 ⚠	Internal to localS...	`index.startsWith('_')`	bro devnull:devnull	0.000%	Off ···
14 ⚠	panRoute	`true`	palo_alto_traffic	0.000%	Off ···
15 ⚠	outViaCEFsdfsda	`sourcetype=='cefOUT'`	cefTest	0.000%	Off ···
16 ⚠	drop50%	`Math.random()>0.8`	devnull devnull:devnull	15.963%	On ···
17 ⚠	Internal metrics t...	`index=='_metrics'`	passthru splunk:localSplunk	65.342%	On ···

Unreachable Route warnings, many

Routing with Output Router

Output Router Destinations offer another way to route data. These function as meta-Destinations, in that they allow selection of actual Destinations based on rules. Rules are evaluated in order, top->down, with the first match being the winner.

Pipelines

What Are Pipelines

After your data has been matched by a Route, it gets delivered to a Pipeline. A Pipeline is a list of Functions that work on the data.

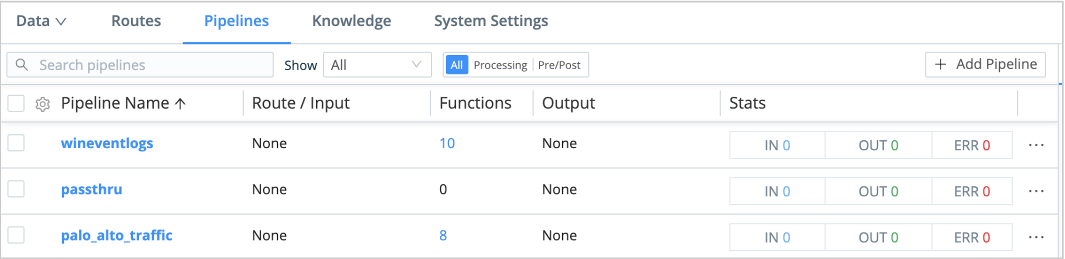
> ℹ As with Routes, the order in which the Functions are listed matters. A Pipeline's Functions are evaluated in order, top->down.

Accessing Pipelines

Select **Pipelines** from LogStream's global top nav (single-instance deployments) or from a Worker Group's top nav (distributed deployments). To configure a new Pipeline, click **+ Pipeline**.

How Do Pipelines Work

Events are always delivered to the beginning of a Pipeline via a Route. The data in the **Stats** column shown below are for the last 15 minutes.



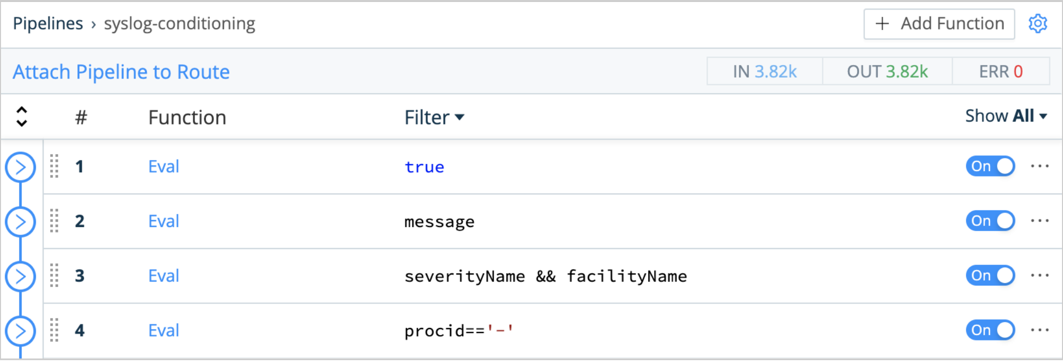
Pipeline Name ↑	Route / Input	Functions	Output	Stats
wineventlogs	None	10	None	IN 0 OUT 0 ERR 0
passthru	None	0	None	IN 0 OUT 0 ERR 0
palo_alto_traffic	None	8	None	IN 0 OUT 0 ERR 0

Pipelines and Route inputs

> ℹ You can press the `]` (right-bracket) shortcut key to toggle between the Preview pane and an expanded Pipelines display. This works

> when no field has focus.

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.



Pipelines › syslog-conditioning

Attach Pipeline to Route

IN 3.82k OUT 3.82k ERR 0

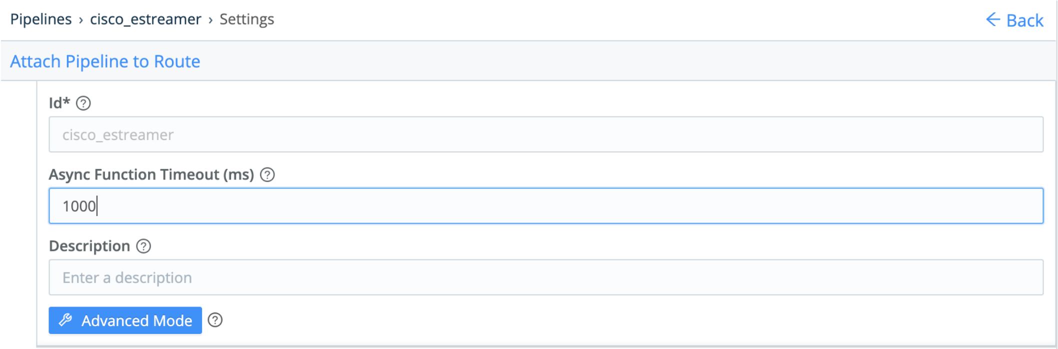
#	Function	Filter
1	Eval	`true`
2	Eval	`message`
3	Eval	`severityName && facilityName`
4	Eval	`procid=='-'`

Pipeline Functions

> ℹ You can streamline the above display by organizing related Functions into Function groups.

Pipeline Settings

Click the gear icon at top right to open the Pipeline's Settings. Here, you can attach the Pipeline to a Route. In the Settings' **Async function timeout (ms)** field, you can enter a buffer to adjust for Functions that might take much longer to execute than normal. (An example would be a Lookup Function processing a large lookup file.)



Pipelines › cisco_estreamer › Settings

Attach Pipeline to Route

Id*: cisco_estreamer

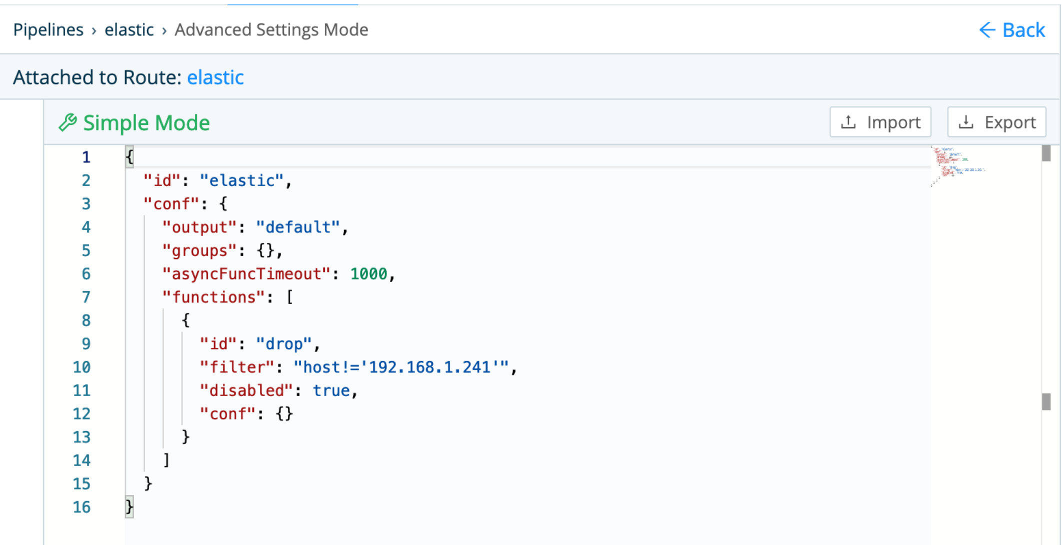
Async Function Timeout (ms): 1000

Description: Enter a description

Pipeline Settings

Advanced Mode (JSON Editor)

Click **Advanced Mode** to edit the Pipeline's definition as JSON text. 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.



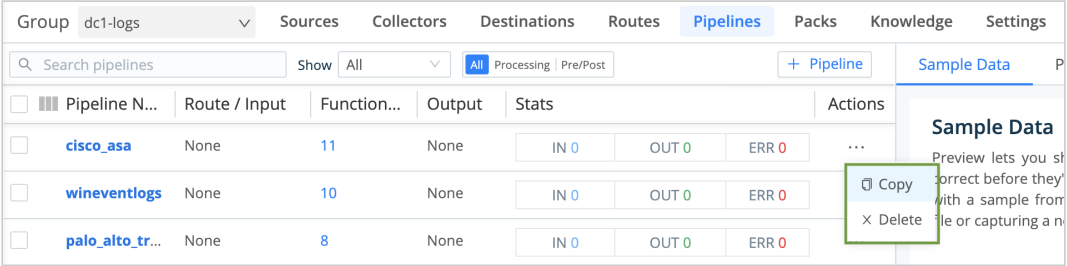
Attached to Route: elastic

```
{
  "id": "elastic",
  "conf": {
    "output": "default",
    "groups": {},
    "asyncFuncTimeout": 1000,
    "functions": [
      {
        "id": "drop",
        "filter": "host!='192.168.1.241'",
        "disabled": true,
        "conf": {}
      }
    ]
  }
}
```

Advanced Pipeline Editing

Pipeline Actions

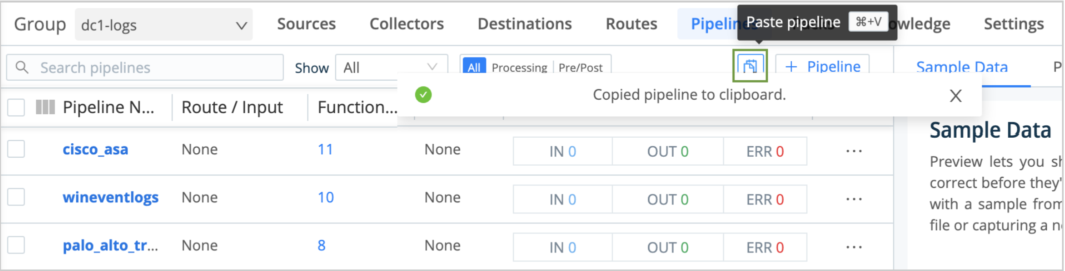
Click a Pipeline's Actions (**...**) menu to display options for copying or deleting the Pipeline.



Pipeline N...	Route / Input	Function...	Output	Stats			Actions
cisco_asa	None	11	None	IN 0	OUT 0	ERR 0	...
wineventlogs	None	10	None	IN 0	OUT 0	ERR 0	
palo_alto_tr...	None	8	None	IN 0	OUT 0	ERR 0	

Pipeline > Actions menu

Copying a Pipeline displays the confirmation message and the (highlighted) Paste button shown below.



Copied pipeline to clipboard.

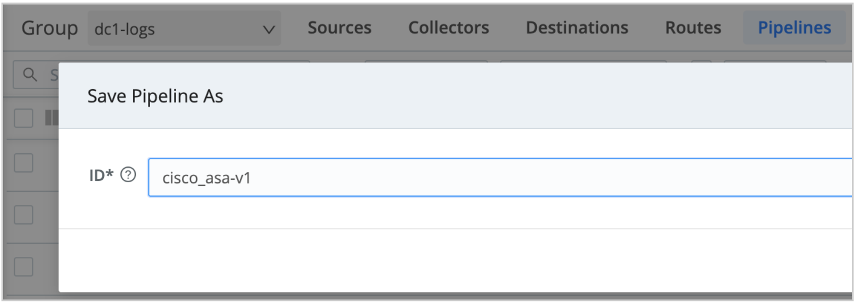
Pipeline N...	Route / Input	Function...				
cisco_asa	None	11	None	IN 0	OUT 0	ERR 0
wineventlogs	None	10	None	IN 0	OUT 0	ERR 0
palo_alto_tr...	None	8	None	IN 0	OUT 0	ERR 0

Sample Data

Preview lets you sh
correct before they'
with a sample from
file or capturing a n

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.



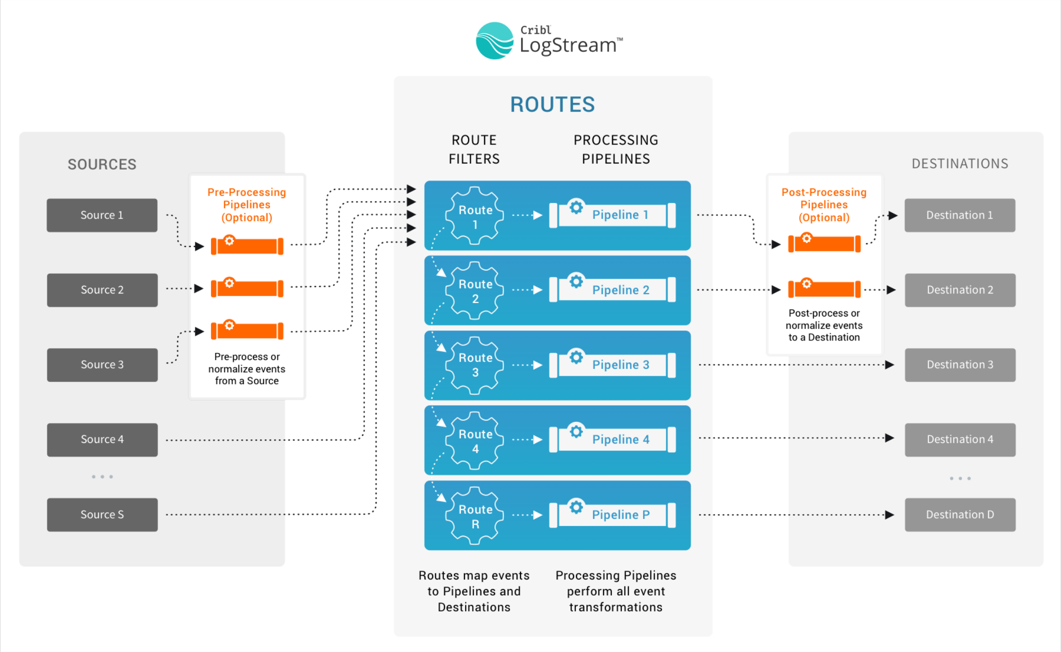
Save Pipeline As

ID* cisco_asa-v1

Saving/renaming a pasted Pipeline

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.



Cribl LogStream™
ROUTES
ROUTE FILTERS
PROCESSING PIPELINES
SOURCES
Source 1
Source 2
Source 3
Source 4
Source S
Pre-Processing Pipelines (Optional)
Pre-process or normalize events from a Source
Route 1
Route 2
Route 3
Route 4
Route R
Pipeline 1
Pipeline 2
Pipeline 3
Pipeline 4
Pipeline P
Routes map events to Pipelines and Destinations
Processing Pipelines perform all event transformations
DESTINATIONS
Post-Processing Pipelines (Optional)
Post-process or normalize events to a Destination
Destination 1
Destination 2
Destination 3
Destination 4
Destination D

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. (E.g., to extract a `message` field before pushing events to various processing Pipelines.)

You configure these Pipelines just like any other Pipeline, by selecting **Pipelines** from the top menu. 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. (E.g., to ensure that a `_time` field exists for all events bound to a Splunk receiver.)

You configure these Pipelines as normal, by selecting **Pipelines** from the top menu. 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 LogStream 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.

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.

Data Onboarding

Onboarding data into Cribl LogStream 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 LogStream 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.

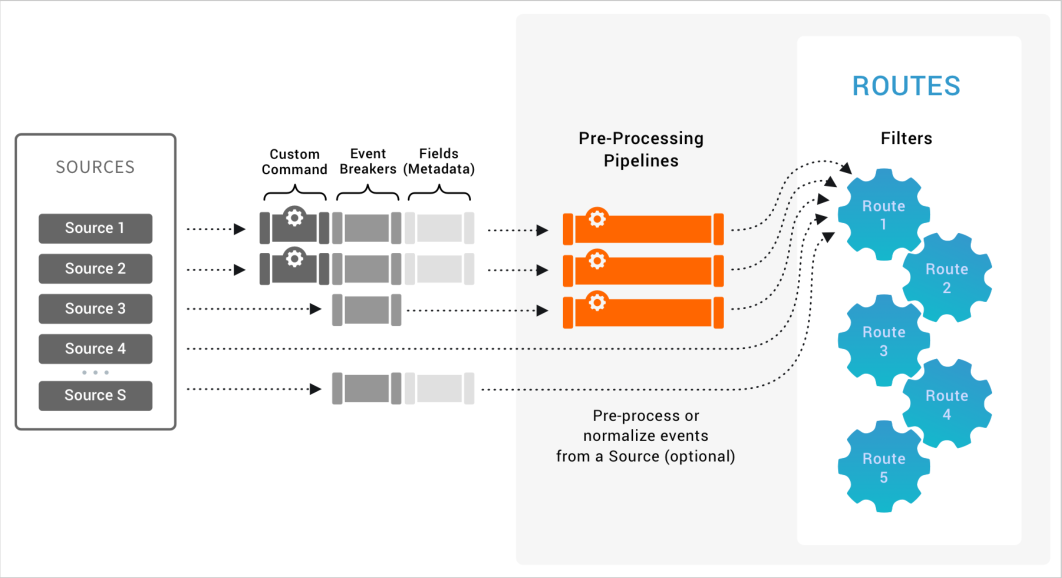
Preview Sample Data

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 LogStream.

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.

Check the Processing Order

While events can be processed almost arbitrarily by functions in LogStream 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 LogStream 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 (Metadata)

Next, events can be enriched with Fields (Metadata). 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.

Functions

What Are Functions

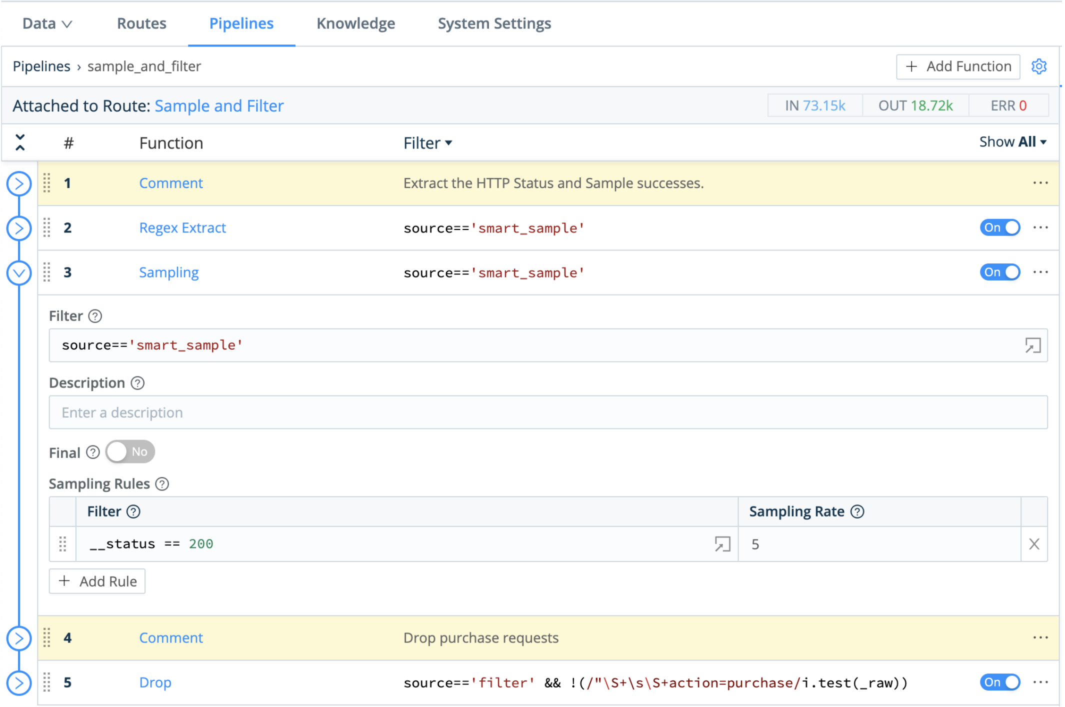
When events enter a Pipeline, they're processed by a series of Functions. At its core, a Function is code that executes on an event, and it encapsulates the smallest amount of processing that can happen to that event.

The term "processing" means a variety of possible options: string replacement, obfuscation, encryption, event-to-metrics conversions, etc. 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, Functions can be configured with filters to further scope their invocation to matching events only.

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 turn Functions **On**/**Off** within a Pipeline as necessary. This enables you to preserve structure as you optimize or debug.



#	Function	Filter
1	Comment	Extract the HTTP Status and Sample successes.
2	Regex Extract	`source=='smart_sample'`
3	Sampling	`source=='smart_sample'`
4	Comment	Drop purchase requests
5	Drop	`source=='filter' && !(/"\S+\s\S+action=purchase/i.test(_raw))`

Filter: `source=='smart_sample'`

Description: Enter a description

Final: No

Sampling Rules

Filter	Sampling Rate
`__status == 200`	5

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

Similar to the `Final` toggle in Routes, the `Final` toggle here controls the flow of events at the Function level. Its states are:

- `No` (**default**): means that matching events processed by this Function will be passed down to the next Function.
- `Yes` : means that this Function is the last one that will be applied to matching events. All Functions further down the Pipeline will be skipped. A Function with `Final` set to `Yes` will display an **F** indicator in the Pipeline stack.

Out-of-the-Box Functions

Cribl LogStream 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.

Custom Functions

For an overview of adding custom Functions to Cribl LogStream, see our blog post, Extending Cribl: Building Custom Functions.

What Functions to Use When

- Add, remove, update fields:
  Eval, Lookup, Regex Extract
- Find & Replace, including basic `sed` -like, obfuscate, redact, hash, etc.:
  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 (e.g, high-volume, low-value data):
  Sampling, Dynamic Sampling
- Suppress events (e.g, duplicates, etc.):
  Suppress
- Serialize events to CEF format (send to various SIEMs):
  CEF Serializer
- Serialize / change format (e.g., convert JSON to CSV):
  Serialize
- Convert JSON arrays into their own events:
  JSON Unroll, XML Unroll
- Flatten nested structures (e.g., nested JSON):
  Flatten
- Aggregate events in real-time (i.e. statistical aggregations):
  Aggregations

- Convert events to metrics format:
  Publish Metrics, Prometheus Publisher (beta)

- Resolve hostname from IP address:
  Reverse DNS (beta)

- 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

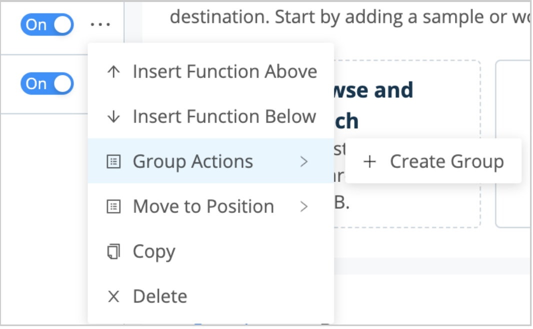
- Add a text comment within a Pipeline's UI, to label steps without changing event data:
  Comment

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: While Functions are in a group, those Functions maintain their global position order in the Pipeline.

> ℹ Function groups work much like Route groups.

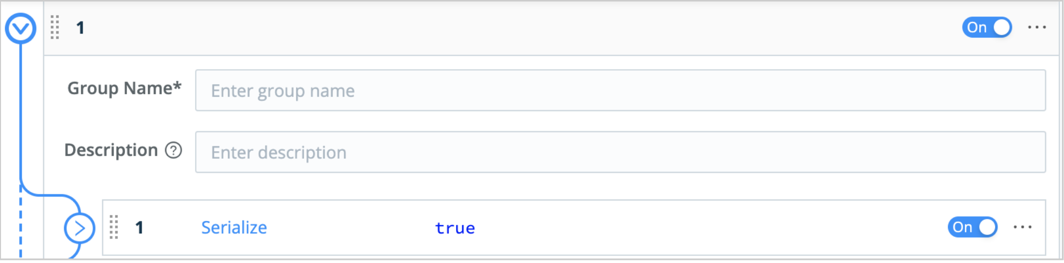
To build a group from any Function, click the Function's ••• (Options) menu, then select **Group Actions > Create Group**.



destination. Start by adding a sample or wo

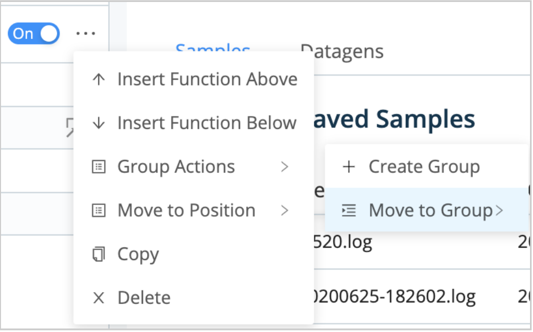
Creating a group

You'll need to enter a **Group Name** before you can save or resave the Pipeline. Optionally, enter a **Description**.



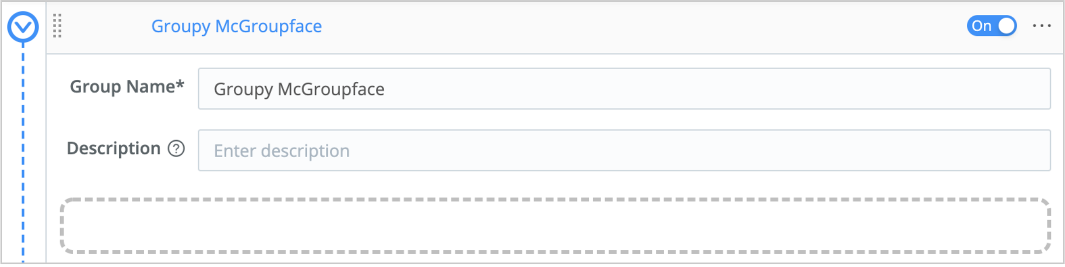
Naming a group

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**.



Expanded Group Actions submenu

You can also use a Function's left grab handle to drag and drop it into, or out of, a group. A saved group that's empty displays a dashed target into which you can drag and drop Functions.



Groupy McGroupface

Group Name*: Groupy McGroupface

Description: Enter description

Drag-and-drop target

Auto Timestamp

Description

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 the Event Breaker Function and the C.Time.timestampFinder() native method.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. The default `true` setting passes all events through the Function.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

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: Add Regex/Strptime pairs to extract additional timestamp formats.

- **Regex**: Regex, with first capturing group matching the timestamp.
- **Strptime format**: Select or enter the strptime format for the captured timestamp.

Click **Add timestamp** to add more rows.

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`.

> i For details about Cribl LogStream'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

Referencing https://github.com/d3/d3-time-format#locale_format:

```
%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
%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 - milliseconds since UNIX epoch.
%s - seconds since UNIX epoch.
%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 (%).
```

Directives marked with an asterisk (*) might be affected by the locale definition.

Complying with the Format

In order to use auto timestamping upon ingestion, the formatting used must match the `%Z` parameters above. E.g., 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

Filter: `name.startsWith('kumquats') && value=='specific string here'`

This will allow the Auto Timestamp Function to act only on events matching the specified parameters.

Sample:

```
Sep 20 12:03:55 PA-VM 1,2019/09/20 13:03:58,CRIBL,TRAFFIC,end,2049,2019/09
```

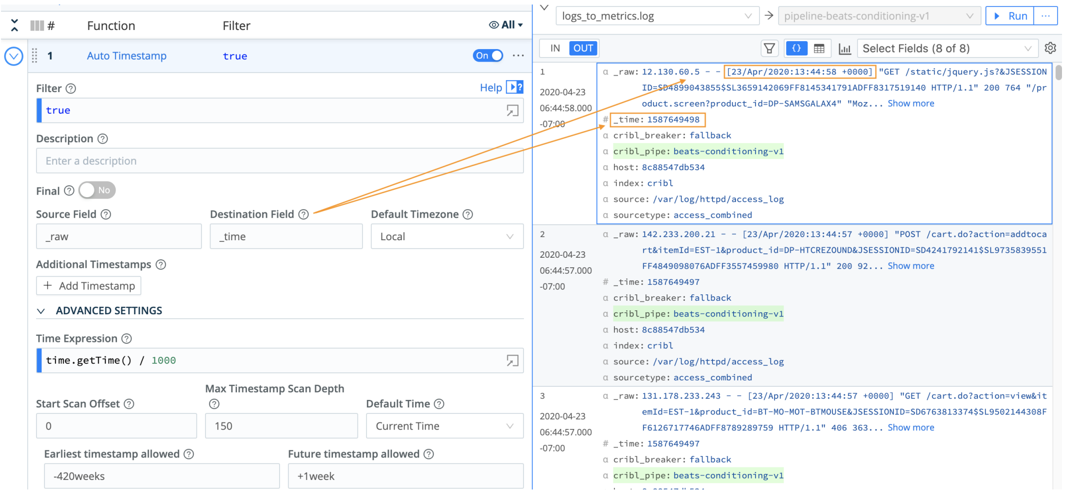
To add this sample (after creating an Auto Timestamp Function with the above **Filter** expression): Go to **Preview** > **Add a Sample** > **Paste a Sample**, and add the data snippet above. Do not make any changes to timestamping or line breaking, and select **Save as Sample File**.

By default, LogSteram will inspect the first 150 characters, and extract the first valid timestamp it sees. You can modify this character limit under **Advanced Settings** > **Max Timestamp Scan Depth**.

LogStream grabs the first part of the event, and settles on the first matching value to display for `time` :

`_time 1569006235`
GMT: Friday, 20 September 2019, 7:03:55 PM GMT
Your Local Time: Friday, 20 September 2019 PDT, 12:03:55 AM **GMT** -07:00

Because no explicit timezone has been set (under **Default Timezone**), `_time` inherits the **Local** timezone, which in this example is `GMT -07:00` .



Filter: true
Description: Enter a description
Final: No
Source Field: _raw
Destination Field: _time
Default Timezone: Local
Additional Timestamps
Time Expression: time.getTime() / 1000
Start Scan Offset: 0
Max Timestamp Scan Depth: 150
Default Time: Current Time
Earliest timestamp allowed: -420weeks
Future timestamp allowed: +1week

logs_to_metrics.log → pipeline-beats-conditioning-v1

```
1                  _raw: 12.130.60.5 - - [23/Apr/2020:13:44:58 +0000] "GET /static/jquery.js?&JSESSION
2020-04-23               ID=SD4899043855$SL3659142069FF8145341791ADFF8317519140 HTTP/1.1" 200 764 "/pr
06:44:58.000             oduct.screen?product_id=DP-SAMSGALAX4" "Moz... Show more
-07:00             _time: 1587649498
                   cribl_breaker: fallback
                   cribl_pipe: beats-conditioning-v1
                   host: 8c88547db534
                   index: cribl
                   source: /var/log/httpd/access_log
                   sourcetype: access_combined
2                  _raw: 142.233.200.21 - - [23/Apr/2020:13:44:57 +0000] "POST /cart.do?action=addtoca
2020-04-23               rt&itemId=EST-1&product_id=DP-HTCREZOUND&JSESSIONID=SD4241792141$SL9735839551
06:44:57.000             FF4849098076ADFF3557459980 HTTP/1.1" 200 92... Show more
-07:00             _time: 1587649497
                   cribl_breaker: fallback
                   cribl_pipe: beats-conditioning-v1
                   host: 8c88547db534
                   index: cribl
                   source: /var/log/httpd/access_log
                   sourcetype: access_combined
3                  _raw: 131.178.233.243 - - [23/Apr/2020:13:44:57 +0000] "GET /cart.do?action=view&it
2020-04-23               emId=EST-1&product_id=BT-MO-MOT-BTMOUSE&JSESSIONID=SD6763813374$SL9502144308F
06:44:57.000             F6126717746ADFF8789289759 HTTP/1.1" 406 363... Show more
-07:00             _time: 1587649497
                   cribl_breaker: fallback
                   cribl_pipe: beats-conditioning-v1
```

> **Timezone Dependencies and Details**
>
> LogStream 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: 200

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, but LogStream will truncate it to millisecond resolution.
>
> For further examples, see Extracting Timestamps from Messy Logs.

Aggregations

Description

The Aggregations Function performs aggregate statistics on event data.

Safeguarding Data

Upon shutdown, LogStream will attempt to flush the buffers that hold aggregated data, to avoid data loss. If you set a **Time window** greater than an 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.)

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

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: Aggregate 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.

- Note: When used without `as()` , the aggregate's output will be placed in a field labeled `<aggFunction>_<fieldName>` . If there are conflicts, the last aggregate wins. For example, given two aggregates – `sum(bytes).where(action=='REJECT')` and `sum(bytes)` – the latter one (`sum_bytes`) is the winner.

Group by Fields: Fields to group aggregates by. Supports wildcard expressions.

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: Determines if the aggregations should be retained for cumulative aggregations, or reset to 0, when flushing out an aggregation table event. Defaults to `No` .

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 : Determines whether to pass through the original events along with the aggregation events. Defaults to `No` .

Metrics mode: Determines whether to output aggregates as metrics. Defaults to `No` , causing aggregates to be output as events.

Sufficient stats mode: Determines whether to output *only* statistics sufficient for the supplied aggregations. Defaults to `No` , meaning output richer statistics.

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 memory usage limit to impose upon aggregations. Defaults to unlimited (i.e., the amount of memory available in the system). Accepts numerals with multiple-byte units, like KB, MB, GB, etc. (such: as `4GB` .)

List of Aggregate Functions

`avg(expr:FieldExpression)` : Returns the average of the values of the parameter.
`count(expr:FieldExpression)` : Returns the number of occurrences of the values of the parameter.
`dc(expr: FieldExpression, errorRate: number = 0.01)` : Returns the estimated number of distinct values of the `<expr>` parameter, within a relative error rate.
`distinct_count(expr: FieldExpression, errorRate: number = 0.01)` : Returns the estimated number of distinct values of the `<expr>` parameter, within a relative error rate.
`earliest(expr:FieldExpression)` : Returns the earliest (based on `_time`) observed value of the parameter.
`first(expr:FieldExpression)` : Returns the first observed value of the parameter.
`last(expr:FieldExpression)` : Returns the last observed value of the parameter.
`latest(expr:FieldExpression)` : Returns the latest (based on `_time`) observed value of the parameter.
`list(expr:FieldExpression[,max:number])` : Returns a list of values of the parameter.

- Optional `max` parameter limits the number of values returned. If omitted, the default is `100` . If set to `0` , will return all values.

`max(expr:FieldExpression)` : Returns the maximum value of the parameter.
`median(expr:FieldExpression)` : Returns the middle value of the sorted parameter..
`min(expr:FieldExpression)` : Returns the minimum value of the parameter.
`per_second(expr:FieldExpression)` : Returns the per second rate (based on `_time`) observed value of the parameter.
`perc(level: number, expr: FieldExpression)` : Returns `<level>` percentile value of the numeric values of the `<expr>` parameter.
`rate(expr:FieldExpression, timeString: string = '1s')` : Returns the rate (based on `_time`) observed value of the parameter.

`stddev(expr:FieldExpression)` : Returns the sample standard deviation of the values of the parameter.
`stddevp(expr:FieldExpression)` : Returns the population standard deviation of the values of the parameter.
`sum(expr:FieldExpression)` : Returns the sum of the values of the parameter.
`sumsq(expr:FieldExpression)` : Returns the sum of squares of the values of the parameter.
`values(expr:FieldExpression[,max:number,errorRate:number])` : Returns a list of distinct values of the parameter.

- Optional `max` parameter limits the number of values returned; if omitted, the default is `0` , meaning return all distinct values.
- 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 values of the parameter.
`variancep(expr:FieldExpression)` : Returns the population variance of the values of the parameter.

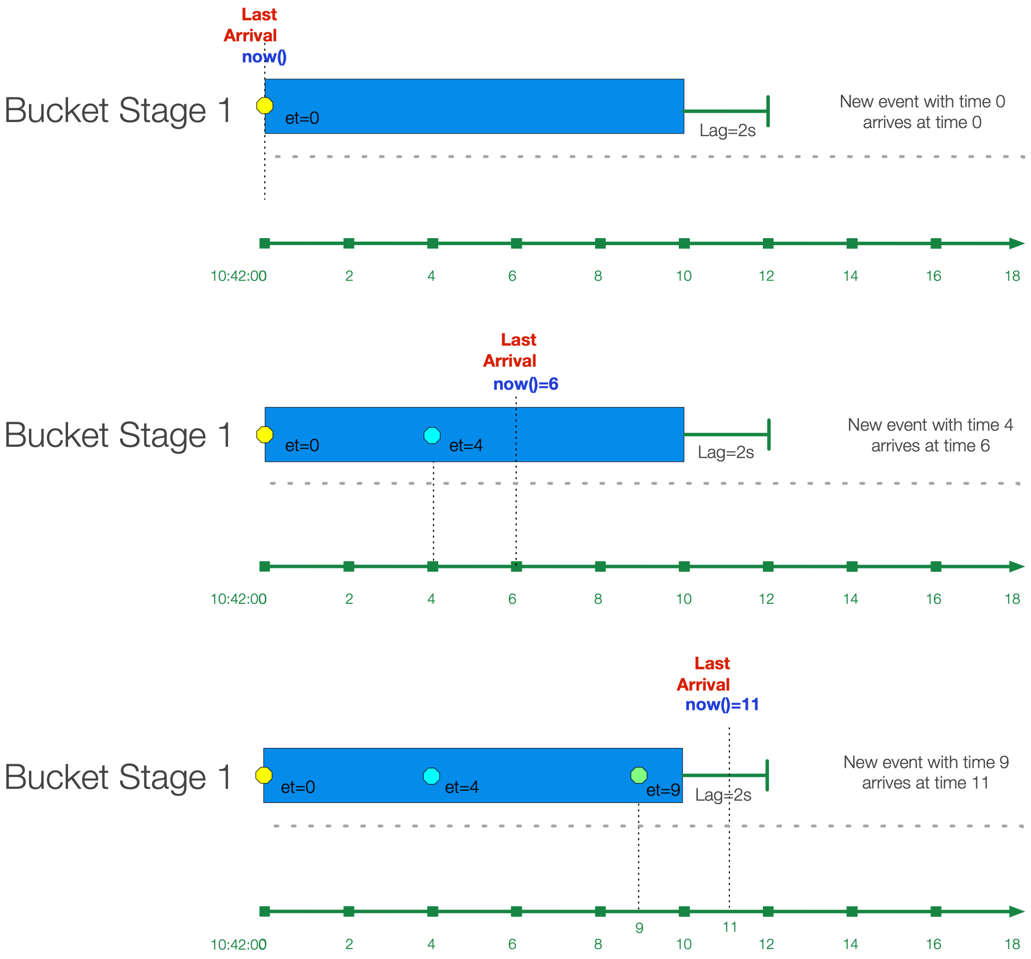
How Do Time Window Settings Work?

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` .

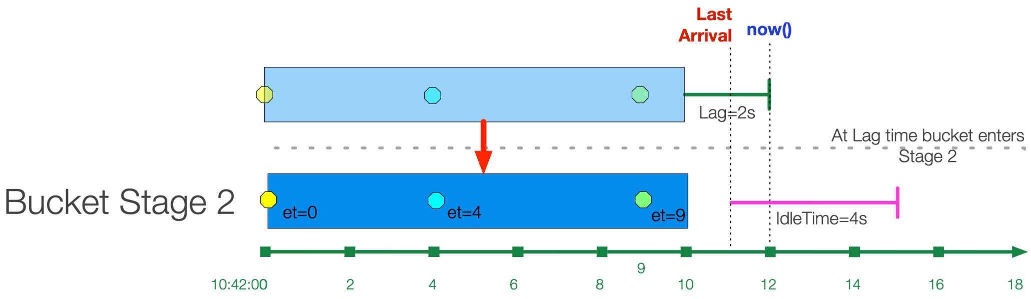
In several cases, there will also be late, or lagging, events that will arrive **after** the latest time window boundary. For example, an event with timestamp `10:42:04` might arrive at `10:42:12` . Lag Tolerance is the setting that governs how long to wait – after the latest window boundary – and still accept late events.

10s Window Aggregation **Settings**: Lag=2s, IdleTime=4s



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.

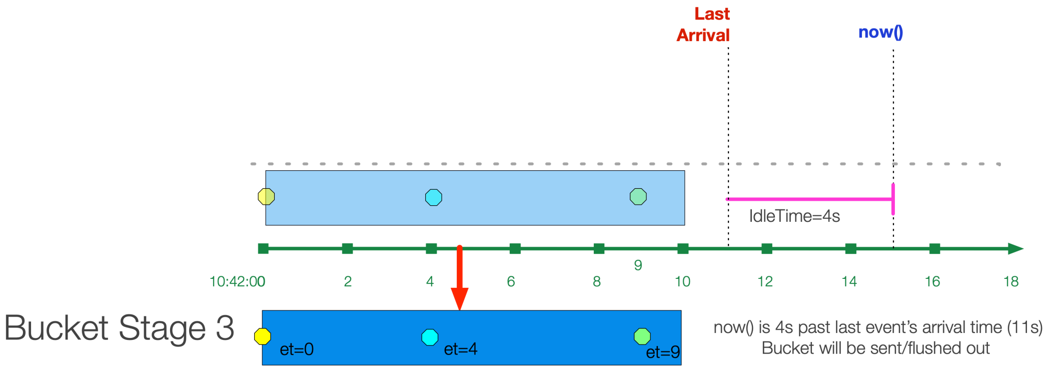
After the lag time expires, the bucket moves to Stage 2.



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.



now() is 4s past last event's arrival time (11s)
Bucket will be sent/flushed out

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`

Scenario D:

Every 10s, compute sum of `bytes` but only where action is `REJECT`, output it in a field called `TotalBytes`. Also, compute distinct count of `srcaddr`.

Time Window: `10s`
Aggregations:
`sum(bytes).where(action=='REJECT').as(TotalBytes)`
`distinct_count(srcaddr).where(action=='REJECT')`

> ℹ For further examples, see Engineering Deep Dive: Streaming Aggregations Part 2 – Memory Optimization

CEF Serializer

Description

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 (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

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 JS 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

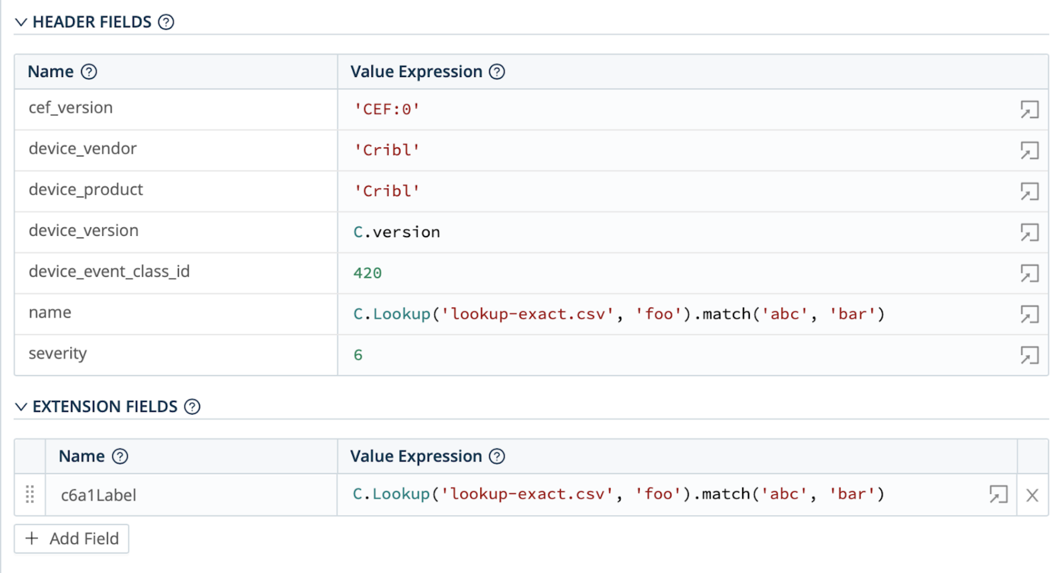
CEF Extension field definitions. Field names and values will be written in `key=value` format. Select each field's Name from the drop-down list. Values can be computed with JS expressions, or can be constants.

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**.



HEADER FIELDS

Name	Value Expression
cef_version	'CEF:0'
device_vendor	'Cribl'
device_product	'Cribl'
device_version	C.version
device_event_class_id	420
name	C.Lookup('lookup-exact.csv', 'foo').match('abc', 'bar')
severity	6

EXTENSION FIELDS

Name	Value Expression
c6a1Label	C.Lookup('lookup-exact.csv', 'foo').match('abc', 'bar')

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
```

Clone

Description

The Clone Function clones events, with optional added fields. Cloned events will be sent to the same Destination as the original event, because they are in the same Pipeline.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

Clones: Create clones with the specified fields added and set.

Fields: Set of key-value pairs to add. Nested addressing is supported.

Examples

In this example, the Destination will receive a clone with an `env` field set to `staging` .

Field: `env`
Value: `staging`

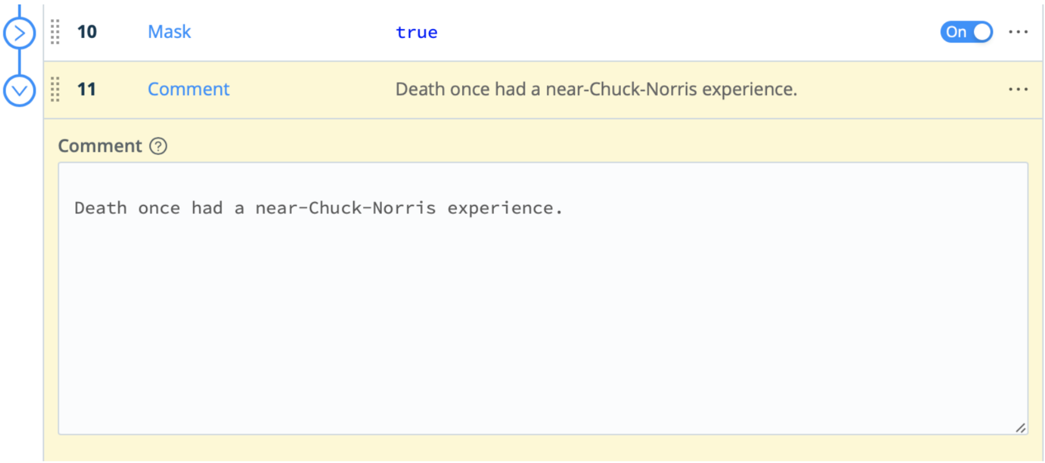
Comment

Description

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.

Usage

Comment: Add your comment as plain text in this field.



10 Mask `true`

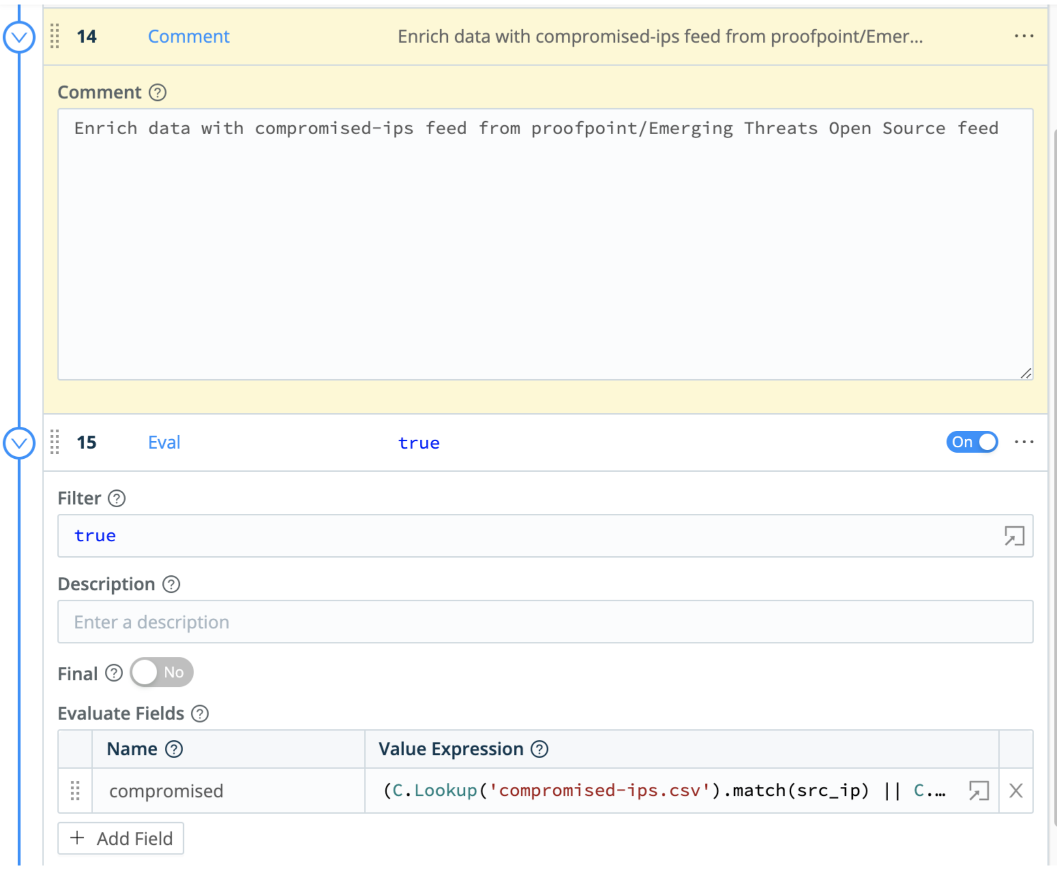
11 Comment Death once had a near-Chuck-Norris experience.

Comment

```
Death once had a near-Chuck-Norris experience.
```

Examples

This comment labels the Pipeline's next function:



14 Comment Enrich data with compromised-ips feed from proofpoint/Emer...

Comment

```
Enrich data with compromised-ips feed from proofpoint/Emerging Threats Open Source feed
```

15 Eval `true` On

Filter: `true`

Description: Enter a description

Final: No

Evaluate Fields

Name	Value Expression
compromised	`(C.Lookup('compromised-ips.csv').match(src_ip) || C.…`

DNS Lookup

Description

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 LogStream's existing Reverse DNS Function, which is now deprecated.)

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 LogStream Worker that will execute this Function. (OS-level caching options include DNSMasq, nscd, systemd-resolved, etc.)

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No` .

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 Field(s) 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**.

Advanced Settings

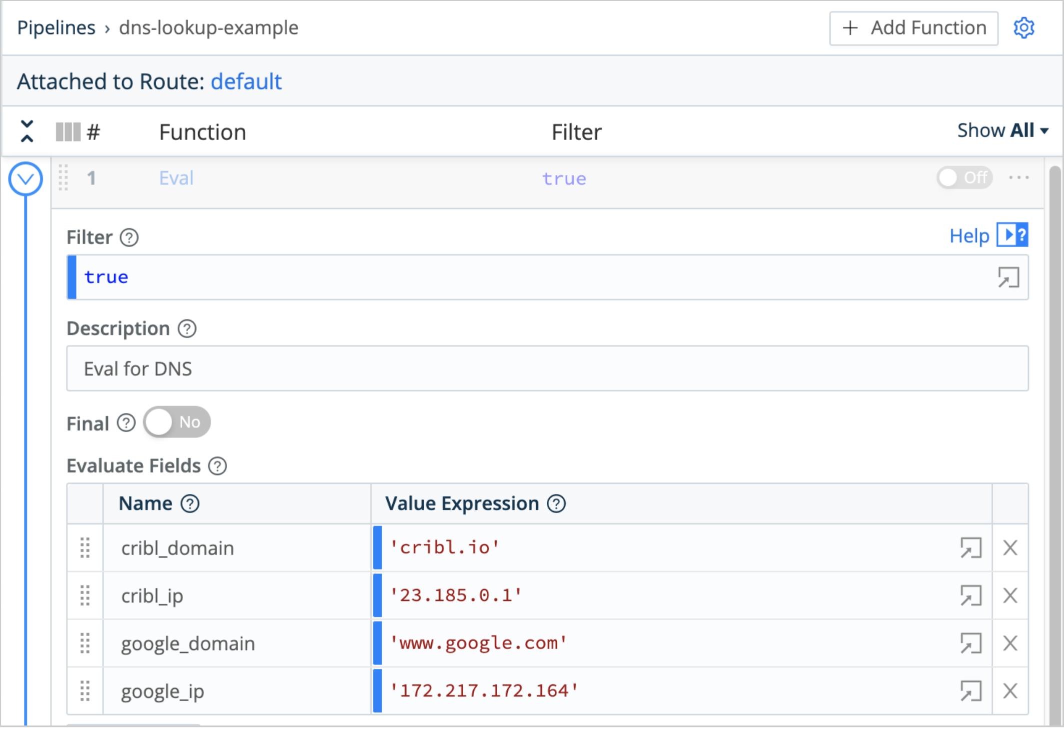
DNS server(s) overrides: IP address(es), in RFC 5952 format, of the DNS server(s) to use for resolution. IPv4 examples: `1.1.1.1`, `4.2.2.2:53`. IPv6 examples: `[2001:4860:4860::8888]`, `[2001:4860:4860::8888]:1053`. If this field is not specified, LogStream will use the system's DNS server.

Reload period (minutes): How often to refresh the two-level DNS cache. Defaults to `30` minutes; use `0` to disable refreshes. At each specified reload interval, the secondary cache is flushed, the primary cache's entries rotate to the secondary cache, and active secondary-cache entries are promoted back to the primary cache for faster lookup.

Maximum cache size: Maximum number of DNS resolutions to cache locally. Before changing the default `5000`, contact Cribl Support to understand the implications. Highest allowed value is `10000`.

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.


Pipelines › dns-lookup-example
+ Add Function
Attached to Route: default
# Function Filter Show All
1 Eval true Off

Filter: `true`

Description: Eval for DNS

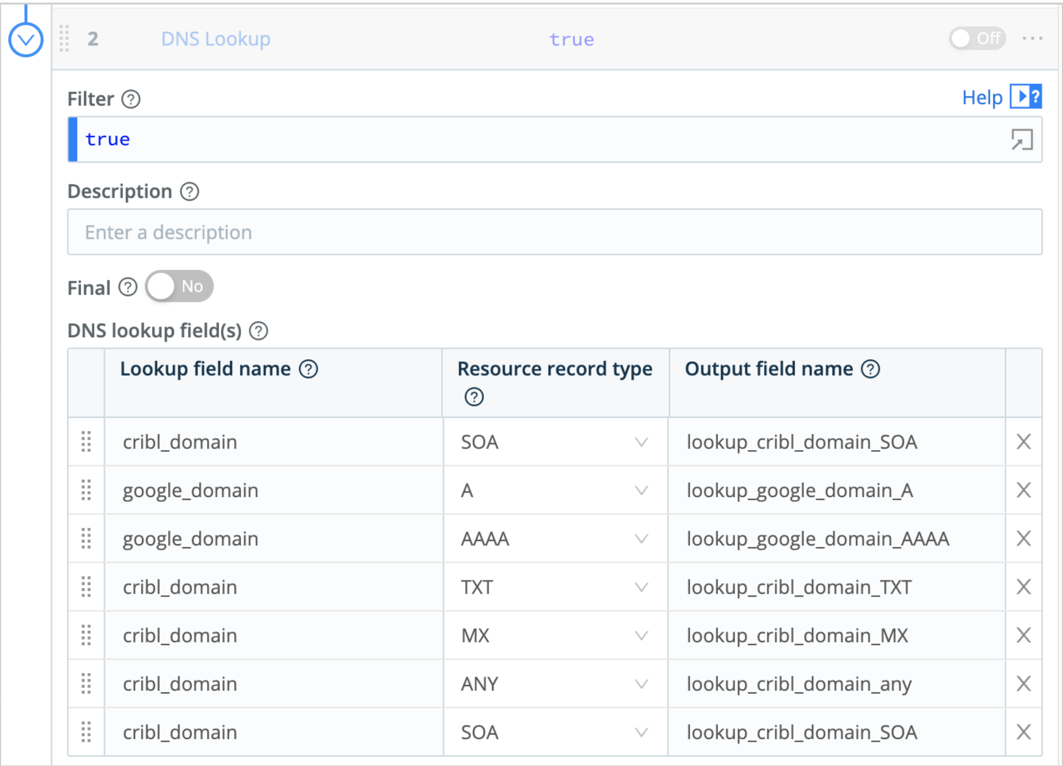
Final: No

Evaluate Fields

Name	Value Expression
cribl_domain	'cribl.io'
cribl_ip	'23.185.0.1'
google_domain	'www.google.com'
google_ip	'172.217.172.164'

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.



Filter: `true`

Description: Enter a description

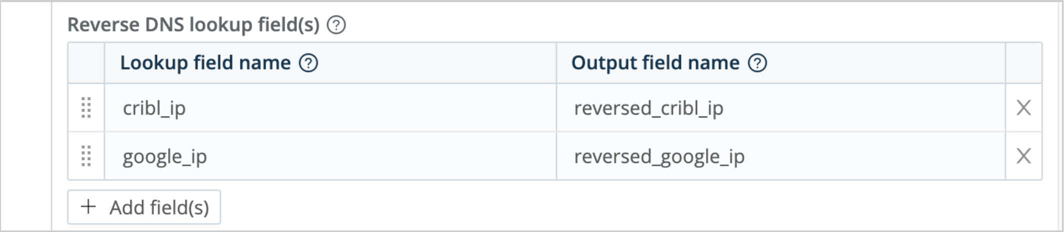
Final: No

DNS lookup field(s)

Lookup field name	Resource record type	Output field name
cribl_domain	SOA	lookup_cribl_domain_SOA
google_domain	A	lookup_google_domain_A
google_domain	AAAA	lookup_google_domain_AAAA
cribl_domain	TXT	lookup_cribl_domain_TXT
cribl_domain	MX	lookup_cribl_domain_MX
cribl_domain	ANY	lookup_cribl_domain_any
cribl_domain	SOA	lookup_cribl_domain_SOA

DNS Lookup: multiple record types

Finally, the same Function's **Reverse DNS lookup** section retrieves domain names for the two IP addresses.



Reverse DNS lookup field(s)

Lookup field name	Output field name
cribl_ip	reversed_cribl_ip
google_ip	reversed_google_ip

DNS Lookup: reverse lookups

Drop

Description

The Drop Function will drop/delete any events that meet the Filter expression.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

Example

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`

Dynamic Sampling

Description

The Dynamic Sampling Function filters out events based on an expression, a sample mode, and events' volume. Your sample mode's configuration determines what percentage of incoming events will be passed along to the next step.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events passed into the Function will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

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 volume. 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`.
- **Max sampling rate**. 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 select a sample rate a priori, Dynamic Sampling allows for **automatically adjusting** sampling rates, based on incoming data volume 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

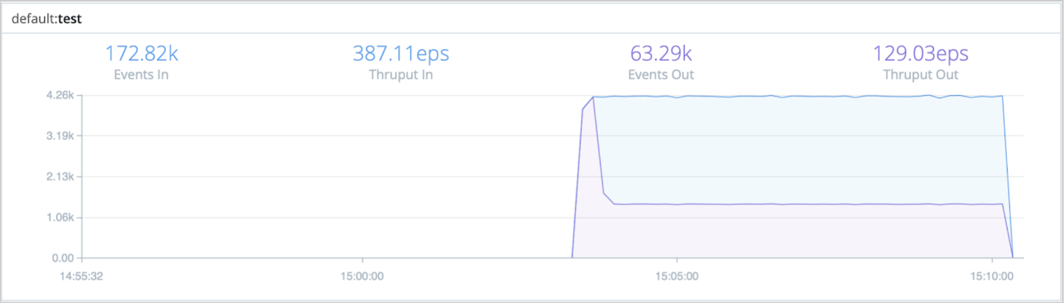
Here's an example that illustrates the effectiveness of using the Square Root sample mode.

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.

> ℹ For further examples, see Getting Smart and Practical With Dynamic Sampling.

Eval

Description

The Eval Function adds or removes fields from events. (In Splunk, these are index-time fields.)

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

Evaluate fields: Set of key/value pairs to add. The left-hand side input (**Name**) is the key name. The right-hand side input (**Value Expression**) is a JS expression to compute the value – this can be a constant. Nested addressing is supported. Strings intended to be used as values must be single- or double-quoted. (For details, see Introduction to Expression Syntax.)

Keep fields: List of fields to keep. Wildcards (*) and nested addressing are supported. Takes precedence over **Remove fields** (below). To reference a parent object and all children requires using the (*) wildcard. For example, if _raw is converted to an object then use `_raw*` to refer to itself and all children.

Remove fields: List of fields to remove. Wildcards (*) and nested addressing are supported. Cannot remove fields matching **Keep fields**. Cribl LogStream internal fields that start with `__` (double underscore) **cannot** be removed via wildcard. Instead, they need to be specified individually. For example, `__myField` cannot be removed by specifying `__myF*`.

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

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=='fail' ? 'blocked' : action`

Scenario C: Create a multivalued field called `myTags`. (i.e., 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`

> ℹ See Ingest-time Fields for more examples.

Advanced Usage Notes

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 JS expression. In this case, content parsed from `_raw` is added at the top level of the event.

Set/Unset Control Fields

You can also use the Eval Function to set and unset control fields (e.g., `_TCP_ROUTING` in Splunk), via this syntax: `_ctrl.<name>` . Control fields can be referenced only on the left-hand side of **Add**. (I.e., they cannot be read or used on the right-hand side, and cannot be referenced in **Remove**.)

To unset/delete a control field, set its value to `undefined` . These fields are normally not needed for event computations, and Cribl suggests that **only experts should modify them**. Please reach out to Cribl if you need help with this topic.

Flatten

Description

The Flatten Function is used to flatten fields out of a nested structure.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

Fields: List of top-level fields to include for flattening. Supports `*` wildcards. Defaults to empty array, which means all fields.

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 be used 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" : "Ma
```

Under **Select Event Breaker**, choose **ndjson** (newline-delimited JSON), and click **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",
  "sales_1_lastName": "Galley",
  "sales_1_age": 41,
}
```

Using the Flatten Function's default settings, we successfully create top-level fields from the nested JSON structure, as expected.

GeoIP

Description

The GeoIP Function enriches events with geographic fields, given an IP address. It is optimized for binary databases such as MaxMind's GeoIP.

> For details on setting up MaxMind (and similar) databases, see Managing Large Lookups.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

GeoIP file (.mmdb): Path to a Maxmind database, in binary format, with `.mmdb` extension.

> ℹ 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 Maxmind database file is in the same location on both the Leader and Worker Nodes.

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.

Grok

Description

The Grok Function extracts structured fields from unstructured log data, using modular regex patterns.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Optional description of this Function's purpose in this Pipeline. Defaults to empty.

Final: If toggled to `Yes`, stops data from being fed to downstream Functions. Defaults to `No`.

Pattern: Grok pattern to extract fields. Cick the Expand button at right to open a preview/valdiation modal. Syntax supported: `%{PATTERN_NAME:FIELD_NAME}`.

Click **+ 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 LogStream'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"}`
```

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",
}
```

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}` . E.g.: `%{IP:client} %{WORD:method}` .
- Useful links for creating and testing Grok patterns: http://grokdebug.herokuapp.com and http://grokconstructor.appspot.com/.
- Additional patterns are available here: https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns.

JSON Unroll

Description

The JSON Unroll Function accepts a JSON object string `_raw` field, unrolls/explodes an **array** of **objects** therein into individual events, while also inheriting top level fields. See example(s). Cribl highly recommends not using this JSON Unroll function for certain types of data. Instead, perform the unrolling using an event breaker for those inputs which support configuring an event breaker. Specifying the event breaker type **JSON Array** and toggling the **JSON Extract Fields** option to **Yes** will accomplish the same unrolling but much more efficiently. This is recommended, for example, for CloudTrail and Office635 events, which are collected as JSON arrays.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

Path: Path to array to unroll, e.g., `foo.0.bar` .

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(s)

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" ] }
    ]
 }
```

Settings:

Path: `allCars`
New Name: `cars`

Output Events:

Resulting Events

```
Event 1:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"Ford","models"

Event 2:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"GM","models":[

Event 3:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"Fiat","models"

Event 4:
{"_raw":"{"date":"9/25/18 9:10:13.000 PM","name":"Amrit","age":42,"cars":{"name":"Blackberry","r
```

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**

Lookup

Description

The Lookup Function enriches events with external fields, using lookup table files in CSV, compressed `.csv.gz`, or binary `.mmdb` format.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

Lookup file path (.csv, .csv.gz): Path to the lookup file. Select an existing file that you've uploaded via LogStream's UI at Knowledge > Lookups Libary, or specify the path. You can reference environment variables via `$`, e.g.: `$CRIBL_HOME/file.csv`.

> i When you configure this field via a distributed deployment's Leader Node, LogStream 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).

Match mode: Defines the format of the lookup file, and indicates the matching logic that will be performed. Defaults to `Exact`.

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.

> ⚠ 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.

Advanced Settings

Reload period (sec): Periodically check the underlying file for modtime changes, and reload if necessary. Use `-1` to disable. Defaults to `60`.

Ignore case: Ignore case when performing **Match mode: Exact** lookups. Defaults to `No`.

Add to raw event: Whether to append the looked-up values to the `_raw` field, as key=value pairs. Defaults to `No`.

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
```

Match mode: Regex

Match type: First match

Lookup field name in event: `_raw`

Corresponding field name in lookup: `regex`

Events before and after

Events before and after

```
### BEFORE:

{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:5
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,THREAT,end,2049,2018/09/20 13:03:5


### AFTER:

{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:5
  "sourcetype": "pan:traffic"
  }
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,THREAT,end,2049,2018/09/20 13:03:5
  "sourcetype": "pan:threat"
  }
```

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
```

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,2049,2018/09/20 13:03
  "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,2049,2018/09/20 13:03
```

```
  "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,2049,2018/09/20 13:03
  "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,2049,2018/09/20 13:03
  "destination_ip": "10.0.0.102",
  "location": [
    "San Francisco",
    "California",
    "US",
  ]}
```

More Examples and Scenarios

More examples:

- Ingest-time Lookups.
- Lookups and Regex Magic.
- Lookups as Filters for Masks.

See also:

- Managing Large Lookups to optimize file locations for large lookup files.
- Redis Function for faster lookups using a Redis integration.

Mask

Description

The Mask Function masks, or replaces, patterns in events. This is especially useful for redacting PII (personally identifiable information) and other sensitive data.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

Masking rules: Match Regex and Replace Expression pairs. Defaults to empty. Each row has the following fields:

- **Match regex**: Pattern to replace. Supports capture groups. Use `/g` to replace all matches, e.g.: `/foo(bar)/g`
- **Replace expression**: A JavaScript expression or literal to replace all matching content.

To add more rows, click **+ Add Rule** .

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.

> i Negated terms are also supported. When you negate field names, the fields list is order-sensitive. E.g., `!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).

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"` ``

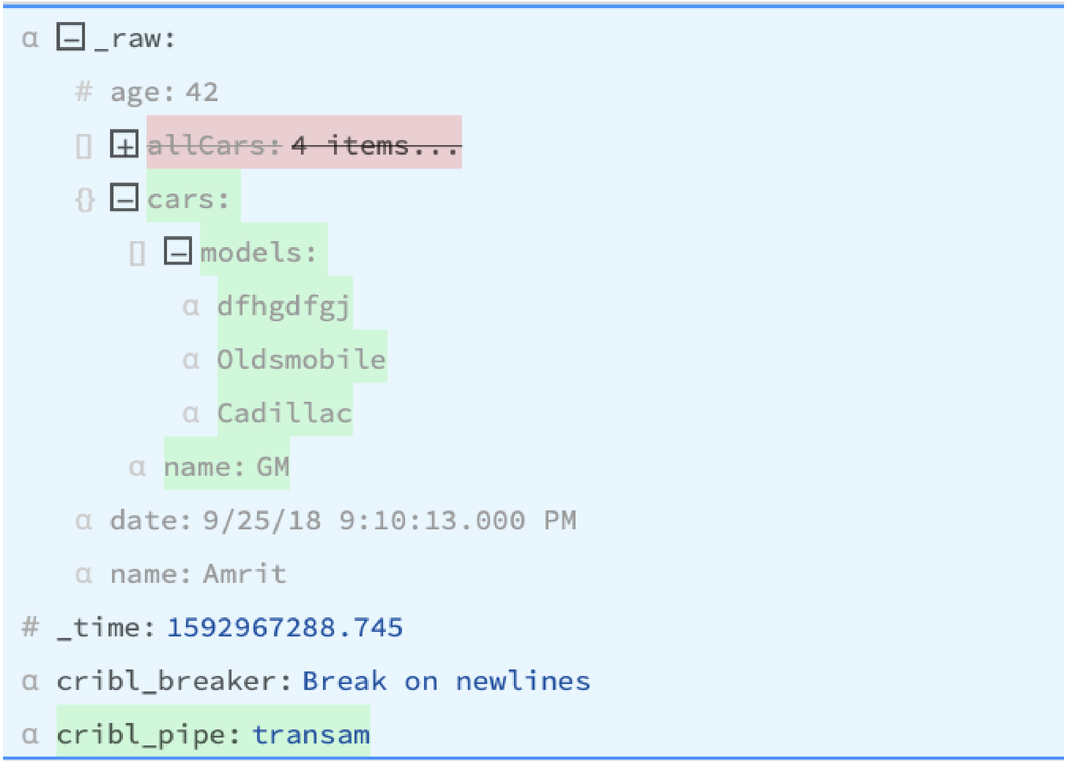
The **Replace expression** can reference other event fields as `event.<fieldName>`. For example, `` `${g1}${event.source}` ``. Note that this is slightly different from other expression inputs, where event fields are referenced without `event.` Here, we require the `event.` prefix for the following reasons:

- 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.

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:

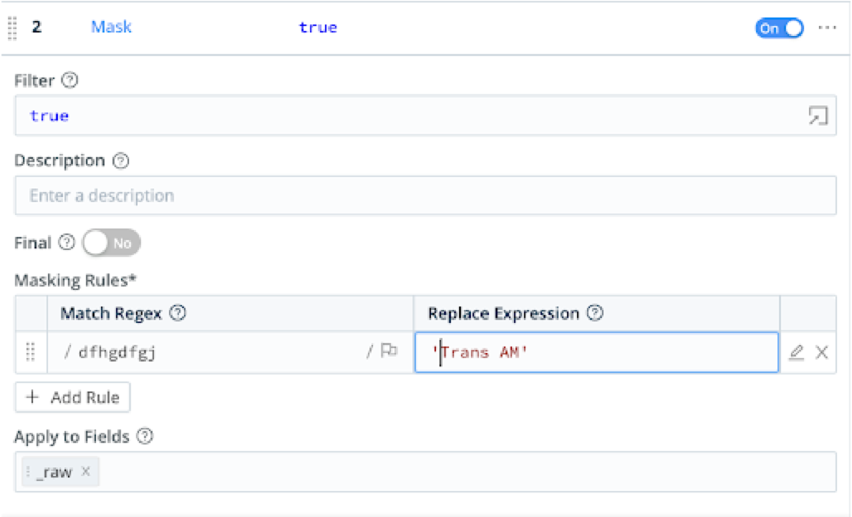


```
α ⊟ _raw:
    # age: 42
    [] ⊞ ~~allCars: 4 items...~~
    {} ⊟ cars:
        [] ⊟ models:
            α dfhgdfgj
            α Oldsmobile
            α Cadillac
        α name: GM
    α date: 9/25/18 9:10:13.000 PM
    α name: Amrit
# _time: 1592967288.745
α cribl_breaker: Break on newlines
α cribl_pipe: transam
```

Event before masking

Configure the Mask Function > Masking Rules as follows:

Match Regex: `dfhgdfgj`
Replace Expression: `Trans AM`



2 Mask true On

Filter: true

Description: Enter a description

Final: No

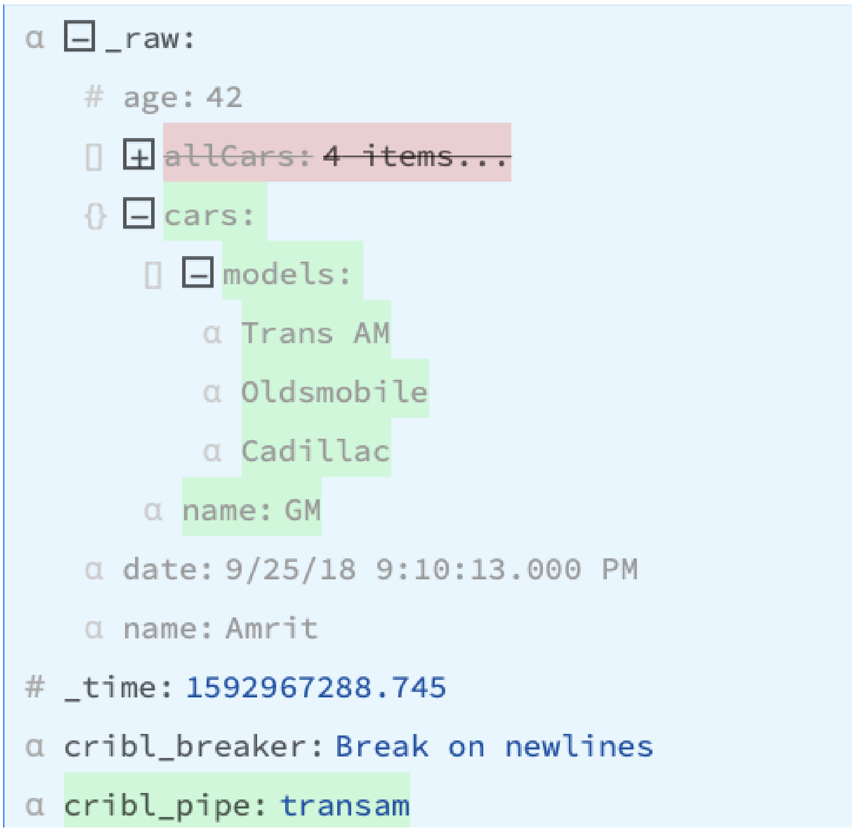
Masking Rules*

Match Regex	Replace Expression
/ dfhgdfgj /	'Trans AM'

Apply to Fields: _raw

Mask Function configuration

Result: Vroom vroom!

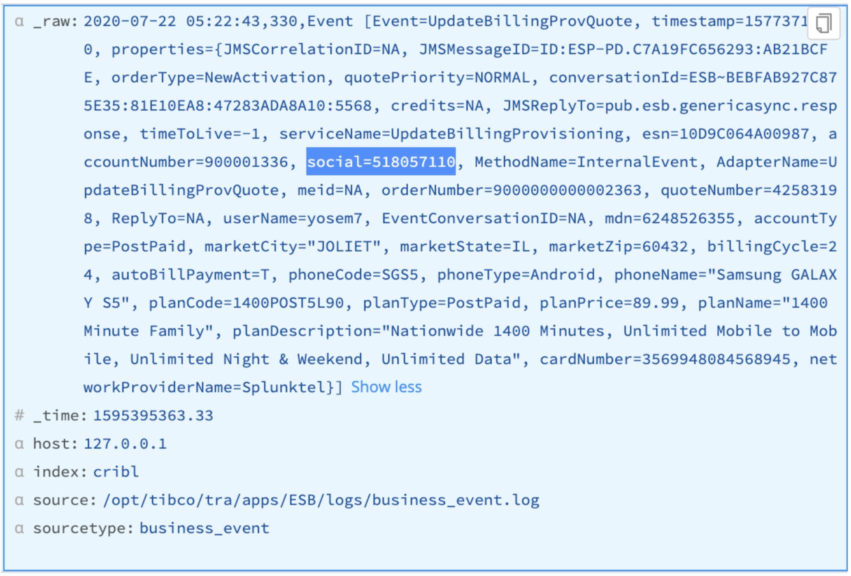


```
α ⊟ _raw:
    # age: 42
    [] ⊞ ~~allCars: 4 items...~~
    {} ⊟ cars:
        [] ⊟ models:
            α Trans AM
            α Oldsmobile
            α Cadillac
        α name: GM
    α date: 9/25/18 9:10:13.000 PM
    α name: Amrit
# _time: 1592967288.745
α cribl_breaker: Break on newlines
α cribl_pipe: transam
```

Event after masking

Example 2: Mask Sensitive Data

Assume that you're ingesting data whose `_raw` fields contain unredacted Social Security numbers in the Key=Value pattern `social=#########` .



```
α _raw: 2020-07-22 05:22:43,330,Event [Event=UpdateBillingProvQuote, timestamp=1577371
        0, properties={JMSCorrelationID=NA, JMSMessageID=ID:ESP-PD.C7A19FC656293:AB21BCF
        E, orderType=NewActivation, quotePriority=NORMAL, conversationId=ESB~BEBFAB927C87
        5E35:81E10EA8:47283ADA8A10:5568, credits=NA, JMSReplyTo=pub.esb.genericasync.resp
        onse, timeToLive=-1, serviceName=UpdateBillingProvisioning, esn=10D9C064A00987, a
        ccountNumber=900001336, social=518057110, MethodName=InternalEvent, AdapterName=U
        pdateBillingProvQuote, meid=NA, orderNumber=9000000000002363, quoteNumber=4258319
        8, ReplyTo=NA, userName=yosem7, EventConversationID=NA, mdn=6248526355, accountTy
        pe=PostPaid, marketCity="JOLIET", marketState=IL, marketZip=60432, billingCycle=2
        4, autoBillPayment=T, phoneCode=SGS5, phoneType=Android, phoneName="Samsung GALAX
        Y S5", planCode=1400POST5L90, planType=PostPaid, planPrice=89.99, planName="1400
        Minute Family", planDescription="Nationwide 1400 Minutes, Unlimited Mobile to Mob
        ile, Unlimited Night & Weekend, Unlimited Data", cardNumber=3569948084568945, net
        workProviderName=Splunktel}] Show less
# _time: 1595395363.33
α host: 127.0.0.1
α index: cribl
α source: /opt/tibco/tra/apps/ESB/logs/business_event.log
α sourcetype: business_event
```

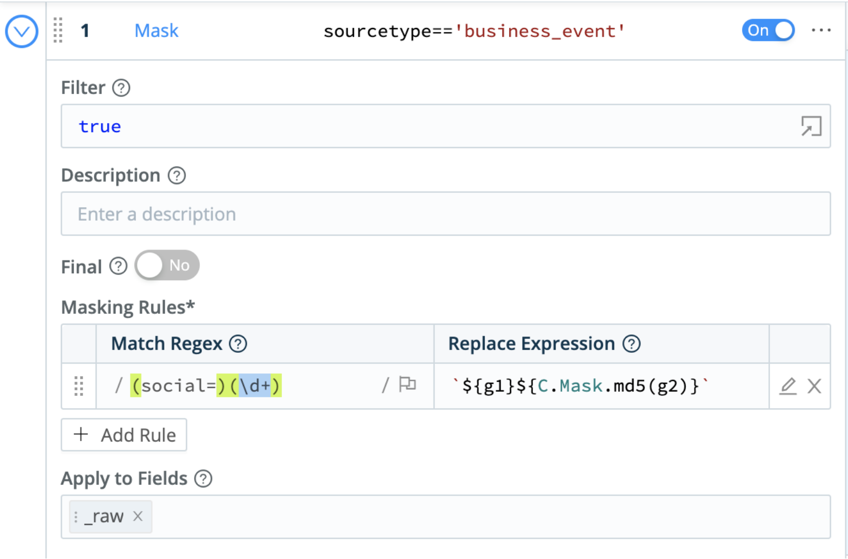
Event with unredacted SSNs

You can use a Mask Function to run an md5 hash of the `social` keys' numeric values, replacing the original values with the hashed values. Configure the Masking Rules as follows:

Match Regex: `(social=)(\d+)`
Replace Expression: ``` `${g1}${C.Mask.md5(g2)}` ```

In the first example everything in the Match regex field was replaced by the Replace Expression. However if that isn't desired then you can use capture groups in the Match Regex to define individual string components for manipulation or, alternatively, use string literals in the Replace expression for retaining any static text. Any content matching the Match Regex that is not inserted into the Replace expression will not be retained.

In this example, `social=` is assigned to capture group g1 for later reference. The value of `social=` will be hashed by referencing it as g2 in the md5 function. If we didn't make `social=` its own capture group (or specified `social=` as a literal in the Replace Expression) then we cannot reference it using g1 in the Replace expression, the value of `social=` would instead be assigned to g1, and the entire `social=#########` string would be replaced with a hash of the social security number, which probably isn't desired because no one would know the value being hashed without a field name preceding it.



1 Mask sourcetype=='business_event' On

Filter: true

Description: Enter a description

Final: No

Masking Rules*

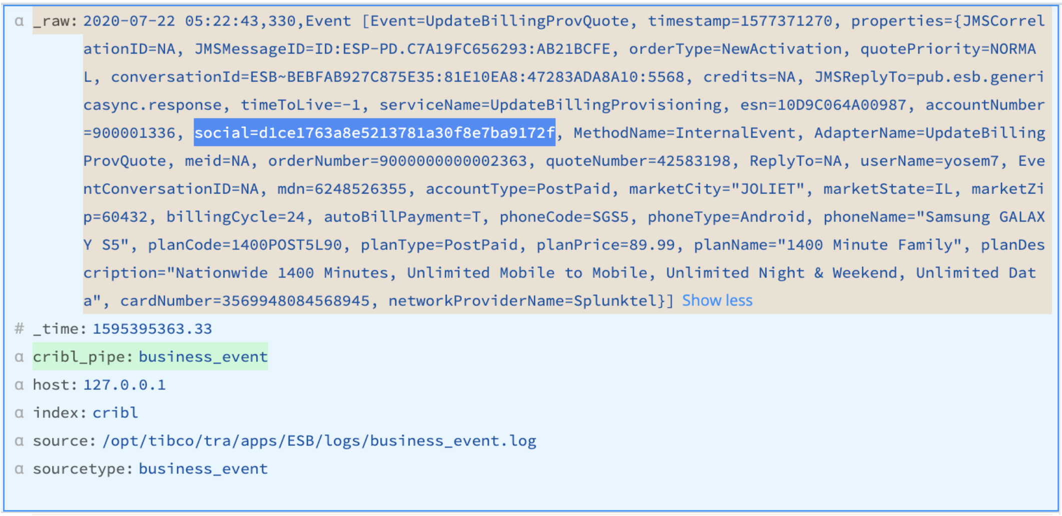
Match Regex	Replace Expression
/(social=)(\d+)/	`` `${g1}${C.Mask.md5(g2)}` ``

+ Add Rule

Apply to Fields: _raw

Mask Function configuration

Result: The sensitive values are replaced by their md5 hashes.



```
α _raw: 2020-07-22 05:22:43,330,Event [Event=UpdateBillingProvQuote, timestamp=1577371270, properties={JMSCorrel
        ationID=NA, JMSMessageID=ID:ESP-PD.C7A19FC656293:AB21BCFE, orderType=NewActivation, quotePriority=NORMA
        L, conversationId=ESB~BEBFAB927C875E35:81E10EA8:47283ADA8A10:5568, credits=NA, JMSReplyTo=pub.esb.generi
        casync.response, timeToLive=-1, serviceName=UpdateBillingProvisioning, esn=10D9C064A00987, accountNumber
        =900001336, social=d1ce1763a8e5213781a30f8e7ba9172f, MethodName=InternalEvent, AdapterName=UpdateBilling
        ProvQuote, meid=NA, orderNumber=9000000000002363, quoteNumber=42583198, ReplyTo=NA, userName=yosem7, Eve
        ntConversationID=NA, mdn=6248526355, accountType=PostPaid, marketCity="JOLIET", marketState=IL, marketZi
        p=60432, billingCycle=24, autoBillPayment=T, phoneCode=SGS5, phoneType=Android, phoneName="Samsung GALAX
        Y S5", planCode=1400POST5L90, planType=PostPaid, planPrice=89.99, planName="1400 Minute Family", planDes
        cription="Nationwide 1400 Minutes, Unlimited Mobile to Mobile, Unlimited Night & Weekend, Unlimited Dat
        a", cardNumber=3569948084568945, networkProviderName=Splunktel}] Show less
# _time: 1595395363.33
α cribl_pipe: business_event
α host: 127.0.0.1
α index: cribl
α source: /opt/tibco/tra/apps/ESB/logs/business_event.log
α sourcetype: business_event
```

Event with hashed SSNs

In scenarios where you need to send unmodified values to certain Destinations (such as archival stores), you can narrow the Mask Function's scope by setting the associated Route's **Output** field.

For further masking examples, see Masking and Obfuscation.

Numerify

Description

The Numerify Function converts event fields that are numbers to type `number`.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

Ignore Fields: Specify fields to **not** numerify. Type in field names, separated by hard returns. By default, this list is empty, and Numerify applies to **all** fields. Supports wildcards (`*`) and nested addressing.

> ℹ Double Negatives
>
> Negated terms are also supported. When you negate field names, the fields list is order-sensitive. E.g., `!foobar` before `foo*` means "Ignore all fields that start with `foo`, except `foobar`." However, `!foo*` before `*` means "Ignore all fields, except for those that start with `foo`."

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 LogStream 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.

Parser

Description

The Parser Function can be used to extract fields out of events, or to reserialize (rewrite) events with a subset of fields. Reserialization will maintain 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.

Parser cannot remove fields that it did not create. A subsequent Eval Function can do so.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No` .

Operation mode: **Extract** will create new fields. **Reserialize** will extract, filter fields, and then reserialize.

Type: Parser/Formatter type to use. Options:

- CSV
- Extended Log File Format (ELFF)
- Common Log Format (CLF)
- K=V Pairs
- JSON

- Delimited Values

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 delimiter or quote characters. Defaults to: `\`
- **Null value**: Field value representing the null value. These fields will be omitted. Defaults to: `-`

Library: Select an option from the Parsers Library.

Source field: Field that contains text to be parsed. Not usually needed in Serialize mode.

Destination field: Name of field in which to add extracted and serialized fields. If multiple new fields are created and this setting is configured then all new fields are created as elements of an array with the array name set to the name specified for this setting. If you want all new fields to be independent, rather than in an array, then specify them using **List of fields** below. (Extract and Serialize modes only.)

Clean fields: This option appears for **Type: K=V Pairs**. Toggle to `Yes` 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.

Fields to keep: List of fields to keep. Supports wildcards (`*`). Takes precedence over **Fields to remove**. Nested addressing supported.

Fields to remove: List of fields to remove. Supports wildcards (`*`). Cannot remove fields matching **Fields to keep**. Nested addressing supported.

> i Negated terms are supported in both **Fields to remove** and **Fields to keep**. When you use negated terms, the list is order-sensitive. E.g., `!foobar, foo*` means "All fields that start with `foo` , except `foobar` ." However, `!foo*, *` means "All fields, except for those that start with `foo` ."

Fields filter expression: Expression to evaluate against `{index, name, value}` context of each field. Return truthy to keep, falsy to remove field. Index is zero-based.

How Fields Settings Interact

The **Fields to keep**, **Fields to remove**, and **Fields filter expression** settings interact as follows:

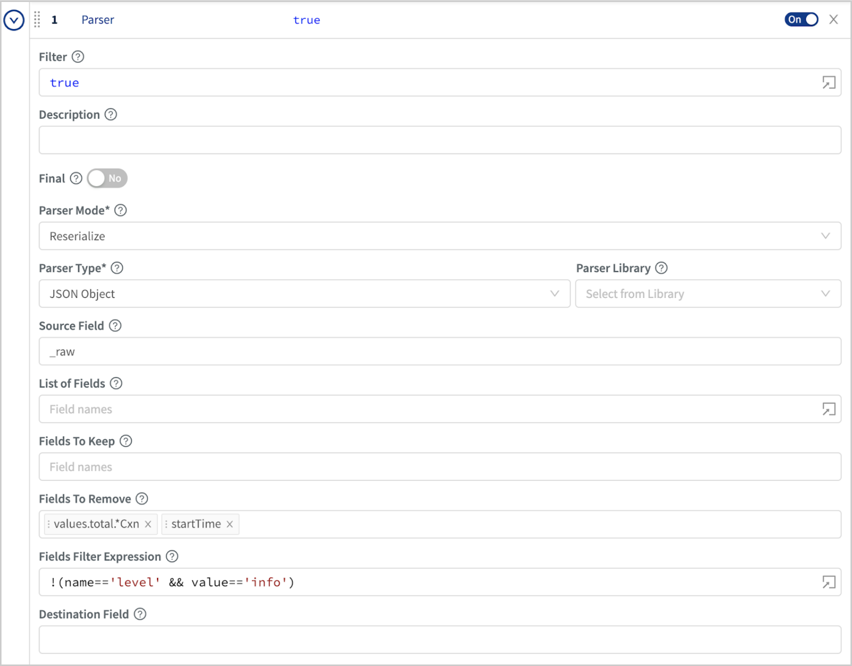
- Order of evaluation: **Fields to keep** > **Fields to remove** > **Fields filter expression**.
- If a field is in both **Fields to keep** and **Fields to remove**, **Fields to keep** takes precedence.
- If a field is in both **Fields to remove** and **Fields filter expression**, **Fields to remove** takes precedence.

Example 1

Insert the following sample, using **Preview** > **Add a Sample** > **Paste a Sample**:

```
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 (or import this Pipeline: https://raw.githubusercontent.com/weeb-cribl/cribl-samples/master/parser/functions/parser/parser_1.json).



Filter: `true`

Description:

Final: No

Parser Mode*: Reserialize

Parser Type*: JSON Object

Parser Library: Select from Library

Source Field: _raw

List of Fields: Field names

Fields To Keep: Field names

Fields To Remove: values.total.*Cxn, startTime

Fields Filter Expression: `!(name=='level' && value=='info')`

Destination Field:

Parser Function initial configuration

First, set the **Parser type** to `Key=Value Pairs` .

Scenario A:

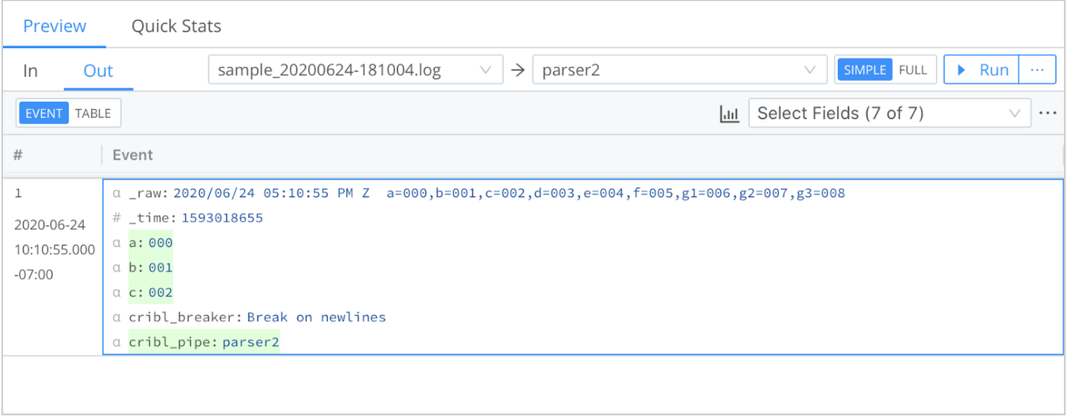
Keep fields `a` , `b` , `c` . Drop the rest.

Expected result: `a` , `b` , `c`

- Fields to Keep: `a` , `b` , `c`
- Fields to Remove: `*`
- Fields Filter Expression:

Result: The event will gain four new fields and values, as follows.

- a: `000`
- b: `001`
- c: `002`
- cribl_pipe: `parser2`



sample_20200624-181004.log → parser2

```
1                  α _raw: 2020/06/24 05:10:55 PM Z  a=000,b=001,c=002,d=003,e=004,f=005,g1=006,g2=007,g3=008
                   # _time: 1593018655
2020-06-24         α a: 000
10:10:55.000       α b: 001
-07:00             α c: 002
                   α cribl_breaker: Break on newlines
                   α cribl_pipe: parser2
```

Scenario A result

You can check your stats by clicking 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`

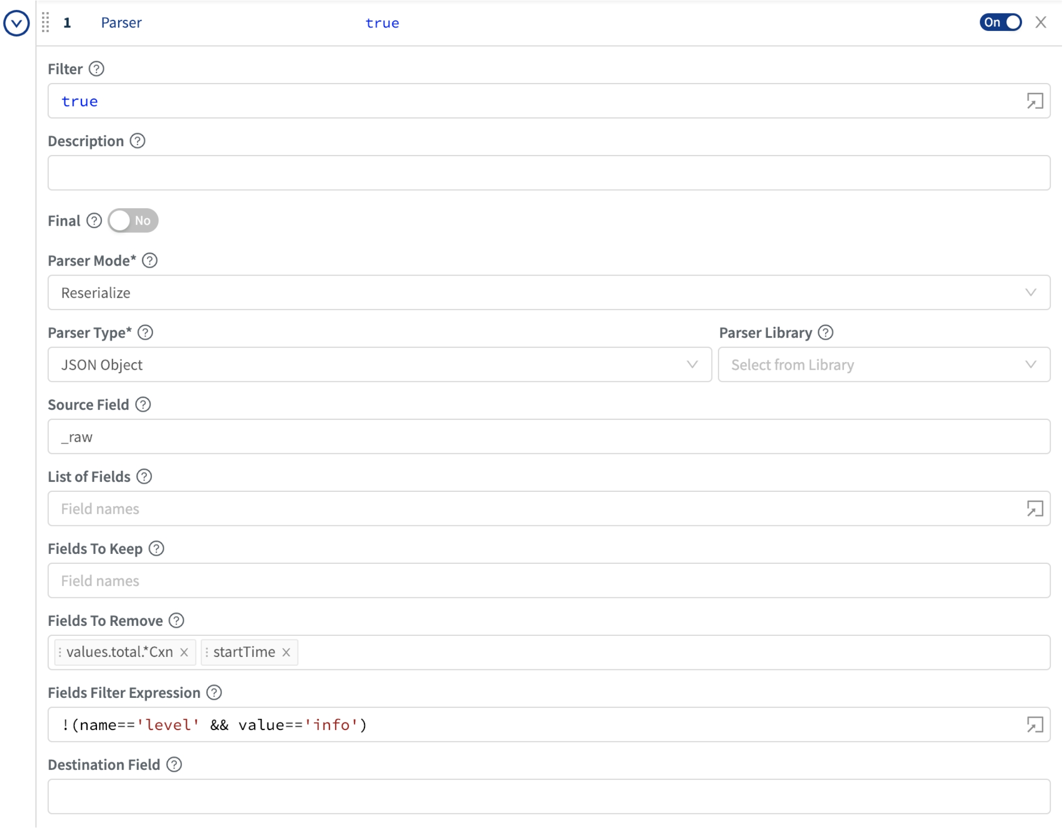
> ℹ The `index` refers to the location of a field in the array of all fields extracted by **this** Parser. It is zero-based. In the case above, `g2` and `g3` have `index` values of `7` and `8` , respectively.

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:



1 Parser true On

Filter: true

Description:

Final: No

Parser Mode*: Reserialize

Parser Type*: JSON Object

Parser Library: Select from Library

Source Field: _raw

List of Fields: Field names

Fields To Keep: Field names

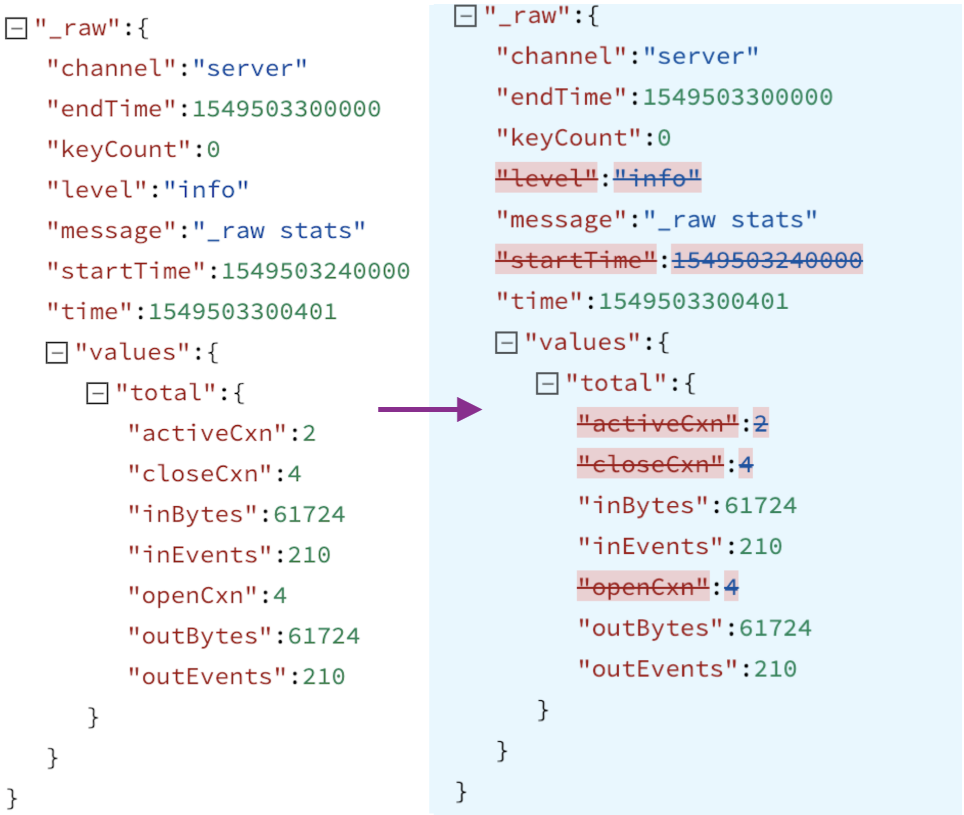
Fields To Remove: values.total.*Cxn, startTime

Fields Filter Expression: `!(name=='level' && value=='info')`

Destination Field:

Parser Function configuration for Example 2

JSON event after being processed by the Function:



```
⊟"_raw":{
   "channel":"server"
   "endTime":1549503300000
   "keyCount":0
   "level":"info"
   "message":"_raw stats"
   "startTime":1549503240000
   "time":1549503300401
   ⊟"values":{
      ⊟"total":{
         "activeCxn":2
         "closeCxn":4
         "inBytes":61724
         "inEvents":210
         "openCxn":4
         "outBytes":61724
         "outEvents":210
      }
   }
}
```

```
⊟"_raw":{
   "channel":"server"
   "endTime":1549503300000
   "keyCount":0
   ~~"level"~~:~~"info"~~
   "message":"_raw stats"
   ~~"startTime"~~:~~1549503240000~~
   "time":1549503300401
   ⊟"values":{
      ⊟"total":{
         ~~"activeCxn"~~:~~2~~
         ~~"closeCxn"~~:~~4~~
         "inBytes":61724
         "inEvents":210
         ~~"openCxn"~~:~~4~~
         "outBytes":61724
         "outEvents":210
      }
   }
}
```

Example 2 event transformation

Example 3

Insert the following sample, using **Preview** > **Add a Sample** > **Paste a Sample**:

```
 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 **Parser 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: `Reserialize`
- 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: `Reserialize`
- Source field: [empty]
- Destination field: `bar`
- Type: `JSON`
- List of fields: [empty]
- Fields to keep: `a` , `b` , `c` , `d`

Publish Metrics

Description

The Publish Metrics Function extracts, formats, and outputs metrics from events.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

Overwrite: If set to `Yes`, overwrite previous metric specs. Otherwise, append. Defaults to `No`.

Metrics

Add Metrics: List of metrics to extract from the event and format. Destinations can pass the formatted metrics to a metrics aggregation platform. Click **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.
- **Metric name expression**: JavaScript expression to evaluate the metric field name. Defaults to the **Event field name** value.

  > i 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.

- **Metric type**: Select `Counter`, `Timer`, or `Gauge` (the default).

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, LogStream will remove that metric from the event.

Dimensions

Add Dimensions: Optional list of dimensions to associate with every extracted metric value. If this Function is used to process output from the Aggregations Function, leave this field blank, because dimensions will be automatically discovered. Defaults to `!_* *`.

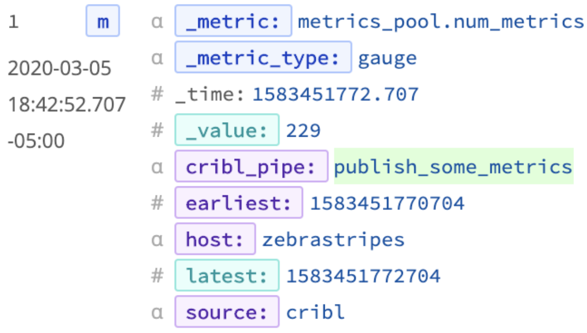
Remove Dimensions: Optional list of dimensions to associate with every extracted metric value. Leave blank if this function is used to process output from the Aggregation function as dimensions will be automatically discovered. If this Function is used to process output from the Aggregations Function, leave this field blank, because dimensions will be automatically discovered.

> ℹ The **Add Dimensions** and **Remove Dimensions** fields support wildcards and negated terms. When you use negated terms, the list is order-sensitive. E.g., `!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`."

Overwrite: If set to `Yes`, overwrite previous metric specs. Otherwise, append. Defaults to `No`.

Fields Color Coding

On the right Preview pane's **OUT** tab, the Publish Metrics Function adds the following color codes to field labels:



```
1              m    α  _metric: metrics_pool.num_metrics
                    α  _metric_type: gauge
2020-03-05          #  _time: 1583451772.707
18:42:52.707        #  _value: 229
-05:00              α  cribl_pipe: publish_some_metrics
                    #  earliest: 1583451770704
                    α  host: zebrastripes
                    #  latest: 1583451772704
                    α  source: cribl
```

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 – i.e. 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

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,
}
```

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:ethernet1/2,destination_port:443
metric.10.10.12.192.bytes_rcvd:0|c|#destination_ip:160.177.222.249,inbound_interface:ethernet1/2,destination_port:443
metric.10.10.12.192.pkts_sent:1|c|#destination_ip:160.177.222.249,inbound_interface:ethernet1/2,destination_port:443
metric.10.10.12.192.pkts_rcvd:0|c|#destination_ip:160.177.222.249,inbound_interface:ethernet1/2,destination_port:443
```

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`.

Regex Extract

Description

The Regex Extract Function extracts fields using regex named groups. (In Splunk, these will be index-time fields). Fields that start with `__` (double underscore) are special fields in Cribl LogStream. They are ephemeral: they can be used by any Function downstream, but **will not** be added to events, and **will not** exit the Pipeline.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

Regex: Regex literal. Must contain named capturing groups, e.g.: `(?<foo>bar)`. Can contain special `_NAME_N` and `_VALUE_N` capturing groups, which extract **both the name and value** of a field, e.g.: `(?<_NAME_0>[^\s=]+)=(?<_VALUE_0>[^\s]+)`. Defaults to empty. See Examples below.

Additional regex: Click **+ Add Regex** to chain extra regex conditions.

Source field: Field on which to perform regex field extraction. Nested addressing is supported. Defaults to `_raw`.

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 `_NAME_n` and `_VALUE_n` capturing groups are used. E.g., 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`.

Overwrite existing fields: Whether to overwrite existing event fields with extracted values. If set to `No` (the default), existing fields will be converted to an array. If toggled to `Yes`, Regex Extract will create array fields if applied multiple times, or if fields exist. (E.g., if `src_ip` is extracted in an input Pipeline where it is assigned a value of `10.1.2.2`, and is also 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"]`.)

Examples

Example 1

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"`

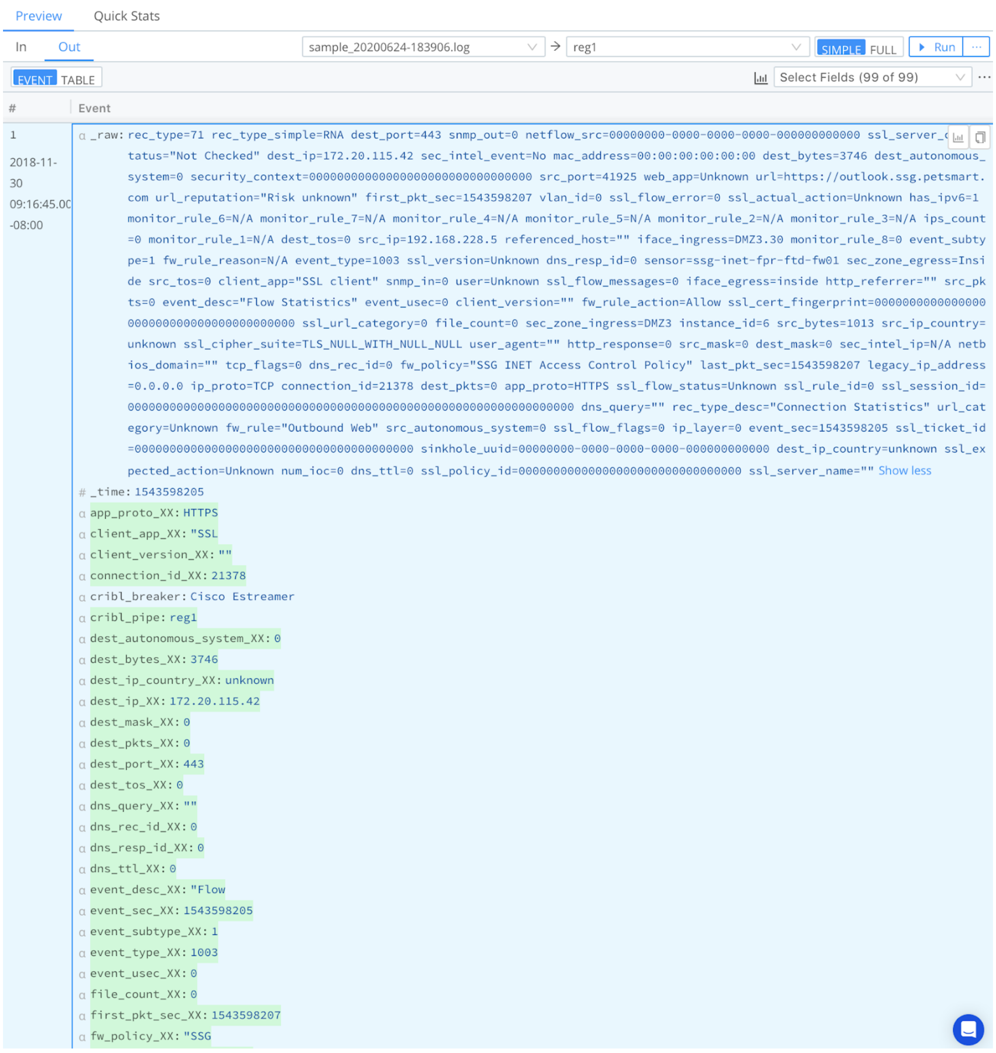
Example 2

Use the first line of the sample here:
https://github.com/weeb-cribl/cribl-samples/blob/master/parser/functions/parser/cisco_estreamer.log

Extract **all** k=v pairs, and add an `_XX` suffix to the end of the extracted fields:

Regex: `(?<_NAME_0>[^\s]+)=(?<_VALUE_0>[^\s]+)`
Field Name Format Expression: `` `${name}_XX` ``

Result:



```
1    2018-11-30 09:16:45.000 -08:00
α _raw: rec_type=71 rec_type_simple=RNA dest_port=443 snmp_out=0 netflow_src=00000000-0000-0000-0000-000000000000 ssl_server_c
tatus="Not Checked" dest_ip=172.20.115.42 sec_intel_event=No mac_address=00:00:00:00:00:00 dest_bytes=3746 dest_autonomous_
system=0 security_context=00000000000000000000000000000000 src_port=41925 web_app=Unknown url=https://outlook.ssg.petsmart.
com url_reputation="Risk unknown" first_pkt_sec=1543598207 vlan_id=0 ssl_flow_error=0 ssl_actual_action=Unknown has_ipv6=1
monitor_rule_6=N/A monitor_rule_7=N/A monitor_rule_4=N/A monitor_rule_5=N/A monitor_rule_2=N/A monitor_rule_3=N/A ips_count
=0 monitor_rule_1=N/A dest_tos=0 src_ip=192.168.228.5 referenced_host="" iface_ingress=DMZ3.30 monitor_rule_8=0 event_subty
pe=1 fw_rule_reason=N/A event_type=1003 ssl_version=Unknown dns_resp_id=0 sensor=ssg-inet-fpr-ftd-fw01 sec_zone_egress=Insi
de src_tos=0 client_app="SSL client" snmp_in=0 user=Unknown ssl_flow_messages=0 iface_egress=inside http_referrer="" src_pk
ts=0 event_desc="Flow Statistics" event_usec=0 client_version="" fw_rule_action=Allow ssl_cert_fingerprint=0000000000000000
000000000000000000000000 ssl_url_category=0 file_count=0 sec_zone_ingress=DMZ3 instance_id=6 src_bytes=1013 src_ip_country=
unknown ssl_cipher_suite=TLS_NULL_WITH_NULL_NULL user_agent="" http_response=0 src_mask=0 dest_mask=0 sec_intel_ip=N/A netb
ios_domain="" tcp_flags=0 dns_rec_id=0 fw_policy="SSG INET Access Control Policy" last_pkt_sec=1543598207 legacy_ip_address
=0.0.0.0 ip_proto=TCP connection_id=21378 dest_pkts=0 app_proto=HTTPS ssl_flow_status=Unknown ssl_rule_id=0 ssl_session_id=
0000000000000000000000000000000000000000000000000000000000000000 dns_query="" rec_type_desc="Connection Statistics" url_cat
egory=Unknown fw_rule="Outbound Web" src_autonomous_system=0 ssl_flow_flags=0 ip_layer=0 event_sec=1543598205 ssl_ticket_id
=0000000000000000000000000000000000000000 sinkhole_uuid=00000000-0000-0000-0000-000000000000 dest_ip_country=unknown ssl_ex
pected_action=Unknown num_ioc=0 dns_ttl=0 ssl_policy_id=00000000000000000000000000000000 ssl_server_name="" Show less
# _time: 1543598205
α app_proto_XX: HTTPS
α client_app_XX: "SSL
α client_version_XX: ""
α connection_id_XX: 21378
α cribl_breaker: Cisco Estreamer
α cribl_pipe: reg1
α dest_autonomous_system_XX: 0
α dest_bytes_XX: 3746
α dest_ip_country_XX: unknown
α dest_ip_XX: 172.20.115.42
α dest_mask_XX: 0
α dest_pkts_XX: 0
α dest_port_XX: 443
α dest_tos_XX: 0
α dns_query_XX: ""
α dns_rec_id_XX: 0
α dns_resp_id_XX: 0
α dns_ttl_XX: 0
α event_desc_XX: "Flow
α event_sec_XX: 1543598205
α event_subtype_XX: 1
α event_type_XX: 1003
α event_usec_XX: 0
α file_count_XX: 0
α first_pkt_sec_XX: 1543598207
α fw_policy_XX: "SSG
```

Example 3

This example uses similar syntax as Example 2, but with a more complex event structure.

In the right **Sample Data** pane, click **Paste** and insert the following sample:

Sample Data

```
<134>1 2020-12-22T17:06:08Z CORP_INT_NLB CheckPoint 18160 - [action:"Accept"; conn_direction:"In
```

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.

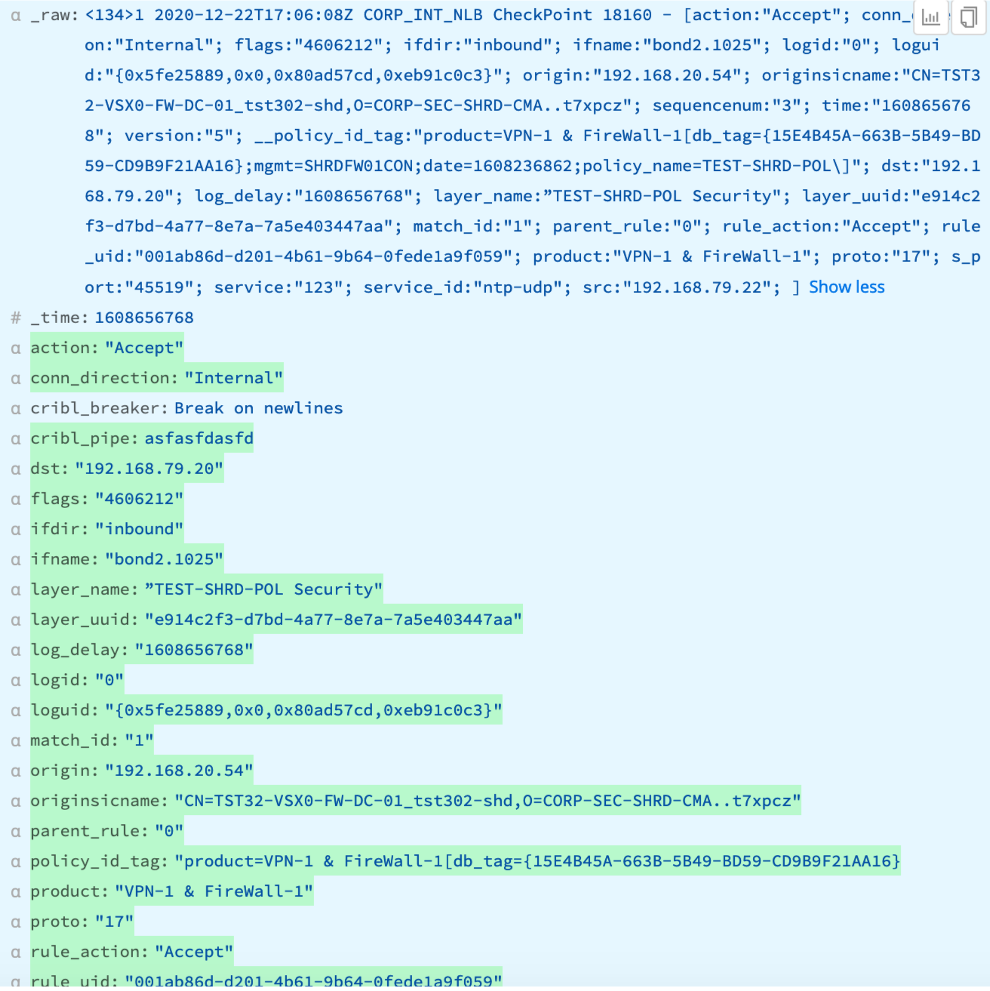
The first Regex Function splits the event to separate the actual data from the header information. We'll split after the `CheckPoint 18160` string, by capturing everything between the `[` and `]`:

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:



```
α _raw: <134>1 2020-12-22T17:06:08Z CORP_INT_NLB CheckPoint 18160 - [action:"Accept"; conn_
        on:"Internal"; flags:"4606212"; ifdir:"inbound"; ifname:"bond2.1025"; logid:"0"; logui
        d:"{0x5fe25889,0x0,0x80ad57cd,0xeb91c0c3}"; origin:"192.168.20.54"; originsicname:"CN=TST3
        2-VSX0-FW-DC-01_tst302-shd,O=CORP-SEC-SHRD-CMA..t7xpcz"; sequencenum:"3"; time:"160865676
        8"; version:"5"; __policy_id_tag:"product=VPN-1 & FireWall-1[db_tag={15E4B45A-663B-5B49-BD
        59-CD9B9F21AA16};mgmt=SHRDFW01CON;date=1608236862;policy_name=TEST-SHRD-POL\]"; dst:"192.1
        68.79.20"; log_delay:"1608656768"; layer_name:”TEST-SHRD-POL Security"; layer_uuid:"e914c2
        f3-d7bd-4a77-8e7a-7a5e403447aa"; match_id:"1"; parent_rule:"0"; rule_action:"Accept"; rule
        _uid:"001ab86d-d201-4b61-9b64-0fede1a9f059"; product:"VPN-1 & FireWall-1"; proto:"17"; s_p
        ort:"45519"; service:"123"; service_id:"ntp-udp"; src:"192.168.79.22"; ] Show less
# _time: 1608656768
α action: "Accept"
α conn_direction: "Internal"
α cribl_breaker: Break on newlines
α cribl_pipe: asfasfdasfd
α dst: "192.168.79.20"
α flags: "4606212"
α ifdir: "inbound"
α ifname: "bond2.1025"
α layer_name: ”TEST-SHRD-POL Security"
α layer_uuid: "e914c2f3-d7bd-4a77-8e7a-7a5e403447aa"
α log_delay: "1608656768"
α logid: "0"
α loguid: "{0x5fe25889,0x0,0x80ad57cd,0xeb91c0c3}"
α match_id: "1"
α origin: "192.168.20.54"
α originsicname: "CN=TST32-VSX0-FW-DC-01_tst302-shd,O=CORP-SEC-SHRD-CMA..t7xpcz"
α parent_rule: "0"
α policy_id_tag: "product=VPN-1 & FireWall-1[db_tag={15E4B45A-663B-5B49-BD59-CD9B9F21AA16}
α product: "VPN-1 & FireWall-1"
α proto: "17"
α rule_action: "Accept"
α rule_uid: "001ab86d-d201-4b61-9b64-0fede1a9f059"
```

> ℹ For further examples, see Using Cribl to Analyze DNS Logs in Real Time – Part 2.

Redis

Description

The Redis Function interacts with Redis stores, setting and getting key-hash and key-value combinations. Redis' in-memory caching of these key pairs enables large lookup tables that would be cumbersome with a .CSV or binary lookup file.

You can use LogStream Collectors (e.g., 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 production data. Note that LogStream does not cache the data returned from this Redis Function.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No` .

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, e.g.: `username` . This is a required field. Click the icon at right to open a validation modal.

Args: A JavaScript expression to compute arguments to the operation. Can return an array. Click the icon at right to open a validation modal.

Redis URL: Redis URL to connect to. The format is: `[redis[s]:]//[[user][:password@]][host][:port][/db-number][?db=db-number[&password=bar[&option=value]]]`

For example: `redis://user:secret@localhost:6379/0?foo=bar&qux=baz`

With no `user` specified: `redis://secret@localhost:6379/0?foo=bar&qux=baz`

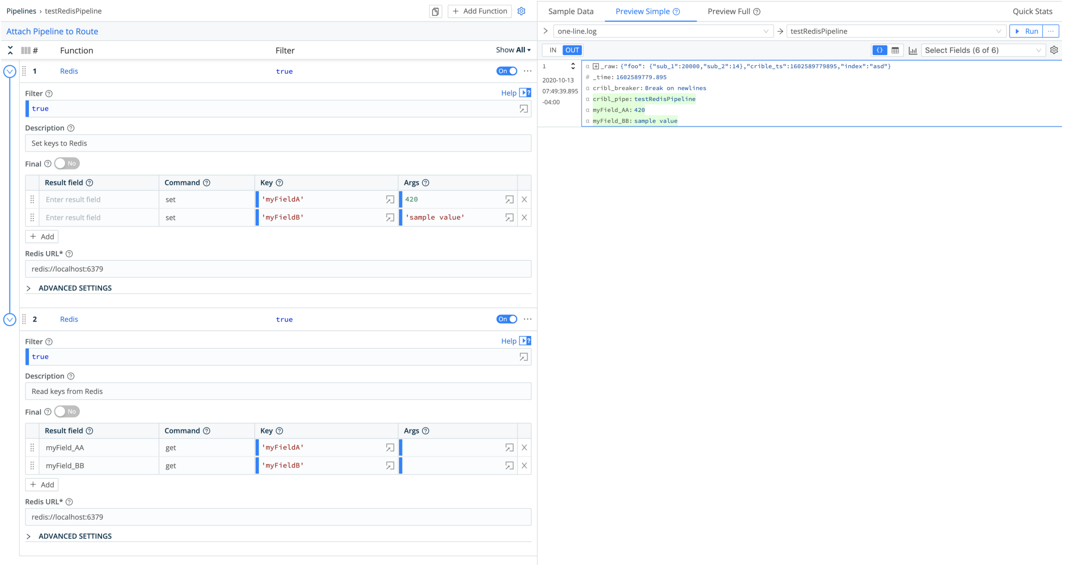
> ⚠ Redis URL Vs. Redis ACL
>
> Through LogStream 2.4.3, the **Redis URL** field has limited compatibility with Redis 6.x's ACL (Access Control List) feature. When using an ACL, point this field to the Redis `default` account, either with a password (e.g., `redis://default:Password1@192.168.1.20:6379`) or with no password (redis://192.168.1.20:6379).
>
> Do not specify a specific user other than `default`, or authentication against Redis will fail.

Advanced Settings

Max blocking time: 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.

Example

This Pipeline demonstrates the use of a pair of Redis Functions. The first Function sets two key-value pairs in Redis. The second Function their values, by key, into two corresponding new **Result field**s.



1 Redis true On

Filter: true

Description: Set keys to Redis

Final: No

Result field	Command	Key	Args
Enter result field	set	'myFieldA'	420
Enter result field	set	'myFieldB'	'sample value'

Redis URL*: redis://localhost:6379

2 Redis true On

Filter: true

Description: Read keys from Redis

Final: No

Result field	Command	Key	Args
myField_AA	get	'myFieldA'	
myField_BB	get	'myFieldB'	

Redis URL*: redis://localhost:6379

one-line.log → testRedisPipeline

```
1            α _raw: {"foo": {"sub_1":20000,"sub_2":14},"crible_ts":1602589779895,"index":"asd"}
2020-10-13   # _time: 1602589779.895
07:49:39.895 α cribl_breaker: Break on newlines
-04:00       α cribl_pipe: testRedisPipeline
             α myField_AA: 420
             α myField_BB: sample value
```

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'`

Regex Filter

Description

The Regex Filter Function filters out events based on regex matches.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

Regex: Regex to test against. Defaults to empty.

Additional regex: Click **+ Add Regex** to chain extra regex conditions.

Field: Name of the field to test against the regex. Defaults to `_raw`. Supports nested addressing.

Examples

See Regex Filtering for examples.

Rename

Description

The Rename Function is designed to change fields' names or reformat their names (e.g., by 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 (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Optionally, enter a simple description of this step in the Pipeline. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

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.

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. Click **+ 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. Expression is evaluated against `{name, value}` context. Example: `name.startsWith('data') ? name.toUpperCase() : name`. You can access other fields' values via `event.<fieldName>`.

> i A single Function can include both **Rename fields** (to rename specified field names) and **Rename expression** (to globally rename fields). 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
}
```

Rename Fields:

Current ame: `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
}
```

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')`

Rollup Metrics

Description

The Rollup Metrics Function merges/rolls up frequently generated incoming metrics into more manageable time windows.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Optional description of this Function's purpose in this Pipeline. Defaults to empty.

Final: If toggled to `Yes`, stops data from being fed to downstream Functions. Defaults to `No`.

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 (e.g., `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, LogStream'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 – e.g. `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 – e.g., `10s` . In the **Dimensions** field, you would remove the default `*` wildcard, and would specify only the dimensions you want to keep – e.g.: `host` , `data_center` .

Sampling

Description

The Sampling Function filters out events, based on an expression and a sampling rate.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

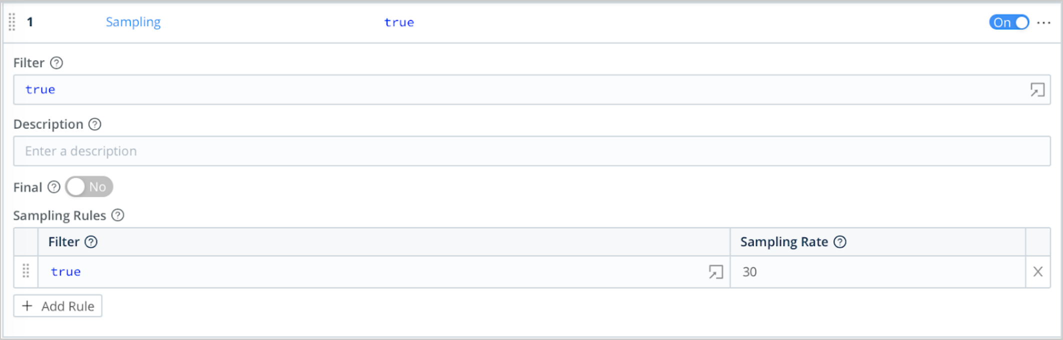
Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

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.



1 Sampling true On

Filter: true

Description: Enter a description

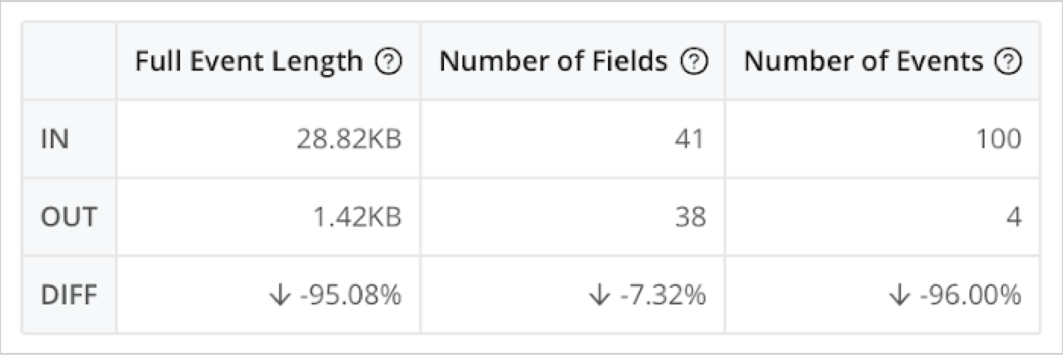
Final: No

Sampling Rules

Filter	Sampling Rate
true	30

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, click **Simple** beside the new file’s name. If you then click 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

See Sampling for examples.

Serialize

Description

Use the Serialize Function to serialize an event's content into a predefined format.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

Type: Data output format. Defaults to `CSV`.

Library: Browse Parser/Formatter library.

Fields to serialize: Required for `CSV`, `ELFF`, and `CLF` Types. (All other formats support wildcard field lists.)

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 click **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`

Suppress

Description

The Suppress Function suppresses events over a time period, based on evaluating a key expression.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

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 `300`.

Drop suppressed events: Specifies if suppressed events should be dropped, or just tagged with `suppress=1`. Defaults to `Yes`, meaning drop.

Advanced Settings

Maximum cache size : 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 **Maximum cache size**. Before changing the default `10000`, contact Cribl Support to understand the implications.

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≠undefined && port≠undefined`

   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: `No`

   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.

Tee

Description

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 LogStream 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.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

Command: Command to execute and receive events (via `stdin`) – one JSON-formatted event per line.

Args: Click **+ 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 `Yes`.

Environment variables: Environment variables to set or overwrite. Click **+ 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 clicked **+ 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 LogStream 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
10:19:15,0,1,63712,443,0,0,0×0,udp,deny,60,60,0,1,2019/10/09
10:19:15,0,any,0,0123456789,0×0,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","packet
s_received":"0"}
```

> ℹ 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.

Trim Timestamp

Description

The Trim Timestamp Function removes timestamp patterns from events, and (optionally) stores them in a specified field.

This Function looks for a timestamp pattern that exists between the characters indicated by numeric `timestartpos` and `timeendpos` fields. It removes `timestartpos` and `timeendpos` along with the timestamp pattern.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description about this step in the Pipeline. Defaults to empty.

Final: If true, stops data from being fed to the downstream Functions. Defaults to `No`.

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`.

Example event before:

````
{"_raw": "Event [Event=UpdateBillingProvQuote, timestamp=1581426279, properties={JMSCorrelationI
"timestartpos":0,
"timestartpos":23
}```

**Field Name**: `time_field`

**Example Event after:**
````

```
{"_raw": "2020-05-22 16:32:11,359 Event [Event=UpdateBillingProvQuote, timestamp=1581426279,
properties={JMSCorrelationID=NA, JMSMessageID=ID:ESP-PD.D2BB2D95F857B:FA323D61,
orderType=RatePlanFeatureChange, quotePriority=NORMAL}",
"time_field":"2020-05-22 16:32:11,359"
}` ` `
```

Unroll

Description

The Unroll Function accepts an array field – or an expression to evaluate an array field – and breaks/unrolls the array into individual events.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

Source field expression: Field in which to find/calculate the array to unroll. E.g.: `_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

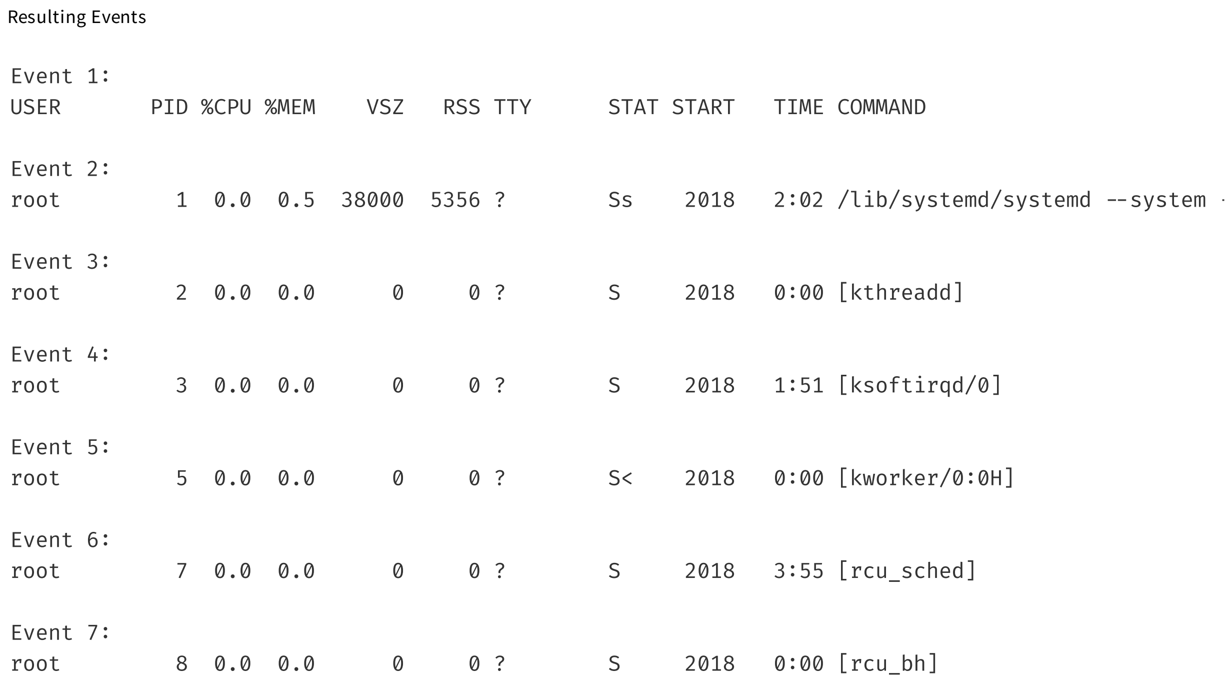
```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.5  38000  5356 ?        Ss    2018   2:02 /lib/systemd/systemd --system
root         2  0.0  0.0      0     0 ?        S     2018   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     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]
```

Settings

Source field expression: `_raw.split(/\n/)`

> ℹ 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/systemd --system

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]
```

XML Unroll

Description

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 (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

Unroll elements regex: Path to the array to unroll. E.g.: `^root\.child\.ElementToUnroll$`

Copy elements regex: Regex matching elements to copy into each unrolled event.
E.g.: `^root\.(childA|childB|childC)$`

Unroll index field: LogStream 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>
```

> ℹ 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)/` . As of LogStream 2.3, 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"?>
<Child>
  <myID>123456</myID>
  <branchLocation>US</branchLocation>
  <state>CA</state>
  <city>San Francisco</city>
</Child>
```

Prometheus Publisher (Deprecated)

> ⚠ This Function is deprecated as of LogStream 3.0. Please instead use the Prometheus Destination to send metrics to Prometheus-compatible endpoints.

Description

The Prometheus Publisher Function allows for metrics to be published to a Prometheus-compatible metrics endpoint. These can be upstream metrics received by LogStream, or metrics derived from the output of LogStream'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 LogStream version, the endpoint is: `http://<worker_node_IP>:<api-port>/metrics`. Within LogStream, 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 (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

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: If set to **No** (the default), overrides the **Final** setting, and suppresses output to downstream Functions' Destinations. Toggle to **Yes** 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: On the default `No` setting, suppresses output to downstream Functions' Destinations. (This overrides the **Final** setting.) Toggle to `Yes` 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="ethernet1/2

# TYPE perf_192_168_1_248_bytes_rcvd counter
metric_192_168_1_248_bytes_rcvd {destination_ip="160.177.222.249",inbound_interface="ethernet1/2

# TYPE perf_192_168_1_248_pkts_sent counter
metric_192_168_1_248_pkts_sent {destination_ip="160.177.222.249",inbound_interface="ethernet1/2"

# TYPE perf_192_168_1_248_pkts_rcvd counter
metric_192_168_1_248_pkts_rcvd {destination_ip="160.177.222.249",inbound_interface="ethernet1/2"
```

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 LogStream.

Reverse DNS (deprecated)

Description

The Reverse DNS Function resolves hostnames from a numeric IP address, using a reverse DNS lookup.

> ⚠ This Function is deprecated. Use the DNS Lookup Function's reverse lookup feature instead.

Usage

Filter: Filter expression (JS) that selects data to be fed through the Function. Defaults to `true`, meaning that all events will be evaluated.

Description: Simple description of this Function. Defaults to empty.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No`.

Lookup Fields

Lookup field name: Name of the field containing the IP address to look up.

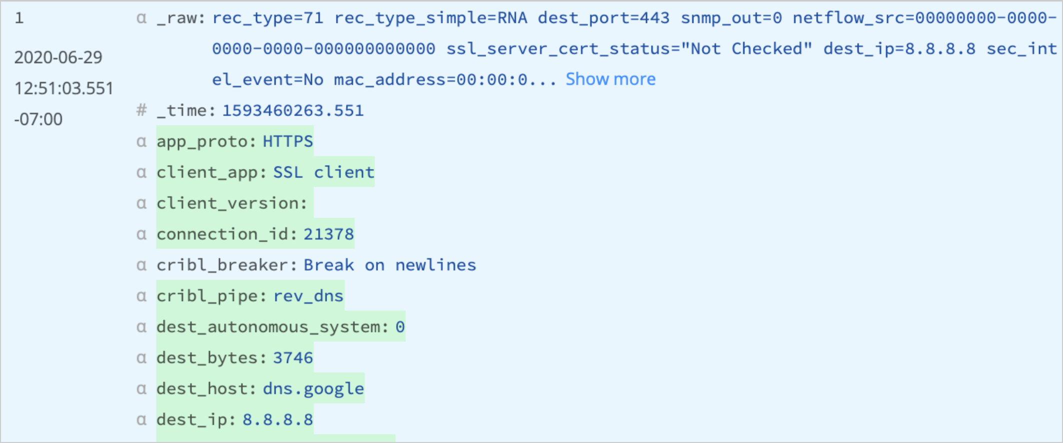
> ⚠ If the field value is not in IPv4 or IPv6 format, the lookup is skipped.

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

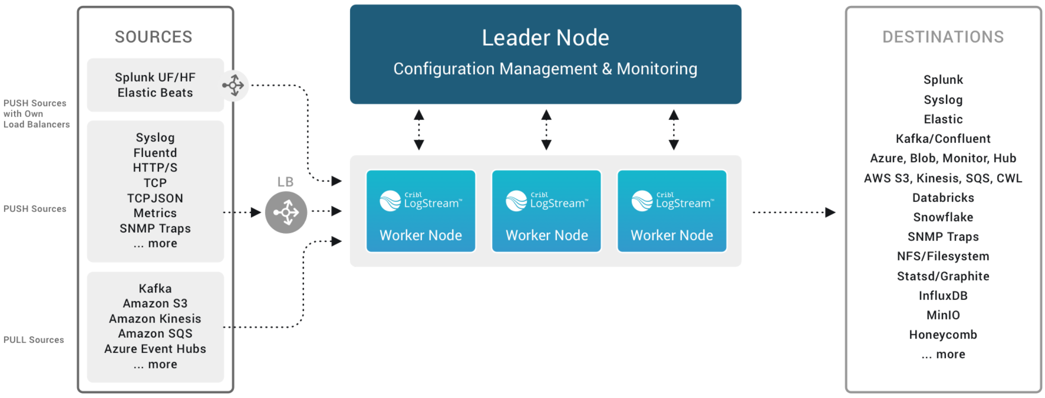
Lookup field name: `dest_ip`
Output field name: `dest_host`
Result: See the `dest_ip` field, and the newly created `dest_host` field, in the events.



```
1                 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_int
2020-06-29                el_event=No mac_address=00:00:0... Show more
12:51:03.551      # _time: 1593460263.551
-07:00            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
                  α dest_bytes: 3746
                  α dest_host: dns.google
                  α dest_ip: 8.8.8.8
```

Sources

Cribl LogStream can receive data from various Sources, including Splunk, HTTP, Elastic Beats, Kinesis, Kafka, TCP JSON, and many others.



Push and Pull Sources

PUSH Sources

Supported data Sources that **send** to Cribl LogStream:

- Splunk TCP
- Splunk HEC
- Syslog
- Elasticsearch API
- TCP JSON
- TCP Raw
- HTTP/S
- Raw HTTP/S
- Kinesis Firehose
- SNMP Trap
- Metrics

- AppScope

Data from these Sources is normally sent to a set of LogStream Workers through a loadbalancer. Some Sources, such as Splunk forwarders, have native loadbalancing capabilities, so you should point these directly at LogStream.

PULL Sources

Supported Sources that Cribl LogStream **fetches** data from:

- Kafka
- Kinesis Streams
- SQS
- S3
- Google Cloud Pub/Sub
- Azure Blob Storage
- Azure Event Hubs
- Office 365 Services
- Office 365 Activity
- Office 365 Message Trace
- Prometheus

Internal Sources

Sources that are **internal** to Cribl LogStream:

- Datagens
- Cribl Internal

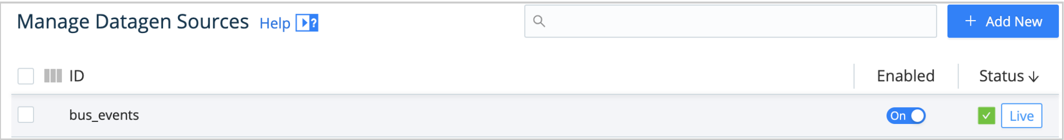
Configuring and Managing Sources

For each Source *type*, you can create multiple definitions, depending on your requirements.

To configure Sources, select **Sources** from LogStream's global top nav (single-instance deployments), or from a Worker Group's top nav (distributed deployments). On the resulting **Data Sources** page's tiles or left menu, select the desired type, then click **+ Add New**.

Capturing Source Data

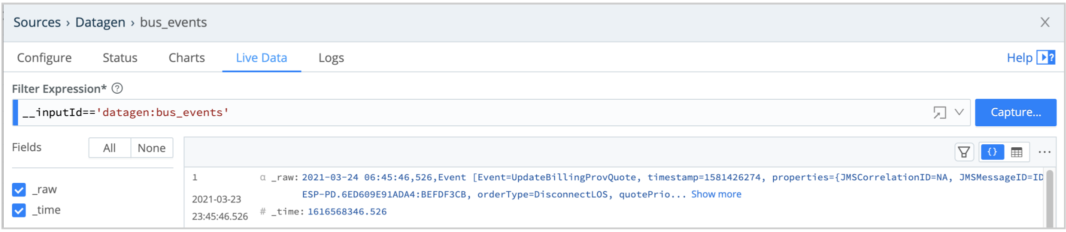
To capture data from a single enabled Source, you can do so directly from the Sources UI instead of using the Preview pane. To initiate an immediate capture, click the **Live** button on the Source's configuration row.



☐	ID	Enabled	Status ↓
☐	bus_events	On	✓ Live

Source > Live button

You can also start an immediate capture from within an enabled Source's configuration modal, by clicking the modal's **Live Data** tab.



```
__inputId=='datagen:bus_events'
```

☑ _raw
☑ _time

```
1            α _raw: 2021-03-24 06:45:46,526,Event [Event=UpdateBillingProvQuote, timestamp=1581426274, properties={JMSCorrelationID=NA, JMSMessageID=ID
2021-03-23            ESP-PD.6ED609E91ADA4:BEFDF3CB, orderType=DisconnectLOS, quotePrio... Show more
23:45:46.526 # _time: 1616568346.526
```

Source modal > Live Data tab

Preconfigured Sources

To accelerate your setup, LogStream 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:

- **Syslog** – TCP Port 9514, UDP Port 9514
- **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
- **Cribl Internal > CriblLogs** – Internal
- **Cribl Internal > CriblMetrics** – Internal

Backpressure Behavior

On the Destination side, you can configure how each LogStream output will respond to a **backpressure** situation – a situation where its in-memory queue is overwhelmed with data.

All Destinations default to **Block** mode, in which they will refuse to accept new data until the downstream receiver is ready. Here, LogStream will back-propagate block signals through the Source, all the way back to the sender (if it supports backpressure, too).

All Destinations also support **Drop** mode, which will simply discard new events until the receiver is ready.

Several Destinations also support a **Persistent Queue** option to minimize data loss. Here, the Destination will write data to disk until the receiver is ready. Then it will drain the disk-buffered data in FIFO (first in, first out) order. See Persistent Queues for details about all three modes, and about **Persistent Queue** support.

Other BackPressure Options

The S3 Source provides a configurable **Advanced Settings > Socket timeout** option, to prevent data loss (partial downloading of logs) during backpressure delays.

Diagnosing Backpressure Errors

When backpressure affects HTTP Sources (Splunk HEC, HTTP/S, Raw HTTP/S, and Kinesis Firehose), LogStream internal logs will show a 503 error code.

Syslog

Cribl LogStream supports receiving of data over syslog.

> i Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**
>
> This Syslog Source supports RFC 3164 and RFC 5424.

Configuring Cribl LogStream to Receive Data over Syslog

From the top nav of a LogStream instance or Group, select **Sources**, then select **Syslog** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Syslog > New Source** modal, which provides the fields outlined below.

> □ LogStream 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.

General Settings

Input ID: Enter a unique name to identify this Syslog Source definition.

Address: Enter the hostname/IP on which to listen for data., E.g. `localhost` or `0.0.0.0`.

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.

TCP port: Enter the TCP port number to listen on. Not required if listening on UDP.

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.

TLS Settings (TCP Only)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path at which to find 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. Defaults to `No`. When toggled to `Yes`:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

- **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. E.g., 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.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable proxy protocol: Defaults to `No`. Toggle to `Yes` if the connection is proxied by a device that supports Proxy Protocol v1 or v2.

IP whitelist regex: Regex matching IP addresses that are allowed to send data. Defaults to `.*` (i.e., all IPs).

Max buffer size (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.)

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.

Single msg per UDP: Whether to treat UDP packet data received as a full syslog message. Defaults to `No`. (I.e., newlines in the packet will be treated as event delimiters.)

Internal Fields

Cribl LogStream 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`
- `__syslogFail : true` for data that fails RFC 3164/5424 validation as syslog format.

Splunk TCP

Cribl LogStream supports receiving Splunk data from Universal or Heavy Forwarders.

> ℹ Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

Configuring Cribl LogStream to Receive Splunk TCP Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **Splunk** > **Splunk TCP** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Splunk TCP > New Source** modal, which provides the fields outlined below.

> LogStream 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.

General Settings

Input ID: Enter a unique name to identify this Splunk Source definition.

Address: Enter hostname/IP to listen for Splunk data. E.g., `localhost` or `0.0.0.0`.

Port: Enter port number.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

* *CA certificate path* : Server path at which to find 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. Defaults to `No`. When toggled to `Yes`:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.
- **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. E.g., 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.

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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline 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, unauthenticated 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 of this token.

Advanced Settings

Enable proxy protocol: Defaults to `No`. Toggle to `Yes` 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 `.*` (i.e., all IPs).

Max active connections: 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.

Internal Fields

Cribl LogStream 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`

Configuring a Splunk Forwarder

To configure a Splunk forwarder (UF, HF) use the following outputs.conf stanzas:

.../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
# As of Splunk 6.5, using forceTimebasedAutoLB is no longer recommended. E
# forceTimebasedAutoLB = false
negotiateProtocolLevel = 0
```

Note that LogStream automatically implicitly sets `useAck = true` with a fake ACK. There is no UI setting for this feature, but your `outputs.conf` can specify `useAck = false`.

Splunk HEC

Cribl LogStream supports receiving data over HTTP/S using the Splunk HEC (HTTP Event Collector).

> ℹ Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

Configuring Cribl LogStream to Receive Data over Splunk HEC

From the top nav of a LogStream instance or Group, select **Sources**, then select **Splunk > HEC** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **HEC > New Source** modal, which provides the fields outlined below.

> LogStream 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.

General Settings

Input ID: Enter a unique name to identify this Splunk HEC Source definition.

Address: Enter the hostname/IP on which to listen for HTTP(S) data. (E.g., `localhost` or `0.0.0.0`.)

Port: Enter the port number.

Splunk HEC endpoint: Absolute path on which to listen for the Splunk HTTP Event Collector API requests. This input supports the `/event` and `/raw` endpoints. Defaults to `/services/collector`.

Allowed Indexes: List the values allowed in the HEC event index field. Allows wildcards. Leave blank to skip validation.

Splunk HEC acks: Whether to enable Splunk HEC acknowledgments. Defaults to `No`. Some sources may require HEC acks to be enabled and, as a result, may keep TCP connections open while waiting for an ack. This behavior can exhaust available file descriptors. Cribl does not maintain a comprehensive list of such sources. Refer to your source's documentation for more information.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path at which to find 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. Defaults to `No`. When toggled to `Yes`:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

- **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. E.g., 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.

Processing Settings

Event Breakers

This section defines event breaking rulesets that will be applied, in order, on the `/raw` endpoint.

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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to determine field's value (can be a constant).

> i Fields specified here will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.
>
> E.g., the following expression's L->R/OR logic specifies: If an inbound event includes an `index` field, use that field's value. Otherwise, fall back to the `myIndex` constant defined here: \` ${__e['index'] || 'myIndex'}
>
> 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 to process data from this input before the data is sent through the Routes.

Auth Tokens

Token: Shared secret to be provided by any client (Authorization: <token>). Click **Generate** to create a new secret. If empty, unauthenticated access **will be permitted**.

Description: Optional description for this token.

Fields: Fields (metadata) to add to events referencing this token. Each field is a **Name**/**Value** pair.

+ Add Token : Click to add more tokens. Each new section provides the same fields listed above.

> ℹ Fields specified here will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.
>
> E.g., the following expression's L->R/OR logic specifies: If an inbound event includes an `index` field, use that field's value. Otherwise, fall back to the `myIndex` constant defined here: ``` `${__e['index'] || 'myIndex'}` ```
>
> Fields here are evaluated and applied **before** any fields specified in the Fields (Metadata) section.

Advanced Settings

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Internal Fields

Cribl LogStream 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`
- `__hecToken`

Format and Endpoint Examples

- Configure Cribl LogStream to listen on port `10080` with an authToken of `myToken42`.
- Send a payload to your Cribl LogStream receiver.

Note: Token specification can be either `Splunk <token>` or `<token>`.

Splunk HEC Event Endpoint

```
curl -k http://<myCriblHost>:10080/services/collector/event -H 'Authorizat

curl -k http://<myCriblHost>:10080/services/collector -H 'Authorization: m

# Multiple Events
curl -k http://<myCriblHost>:10080/services/collector -H 'Authorization: m

# Metrics Events
curl -k http://<myCriblHost>:10080/services/collector/event -H 'Authorizat

curl -k http://<myCriblHost>:10080/services/collector/event -H 'Authorizat
```

Amazon Kinesis Firehose

Cribl LogStream supports receiving data from Amazon Kinesis Data Firehose delivery streams via Kinesis' **HTTP endpoint** destination option.

> ℹ Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring LogStream to Receive Data over HTTP(S) from Amazon Kinesis Firehose

From the top nav of a LogStream instance or Group, select **Sources**, then select **Amazon > Firehose** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Firehose > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Source definition.

Address: Address to bind on. Defaults to 0.0.0.0 (all addresses).

Port: Enter the port number to listen on.

Auth tokens: Shared secrets to be provided by any client (Authorization: <token>). Click **Generate** to create a new secret. If empty, unauthenticated access **will be permitted**.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS` .

CA certificate path: Server path at which to find 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. Defaults to `No` . When toggled to `Yes` :

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No` .

- **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. E.g., 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.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

- **Name**: Field name.
- **Value**: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Internal Fields

Cribl LogStream 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`
- `__firehoseArn`
- `__firehoseReqId`
- `__firehoseEndpoint`
- `__firehoseToken`

Limitations/Troubleshooting

If you set the optional `IntervalInSeconds` and/or `SizeInMBs` parameters in the Kinesis Firehose `BufferingHints` API, beware of selecting extreme values (toward the ends of the API's supported ranges). These can send more bytes than LogStream can buffer, causing LogStream to send HTTP 500 error responses to Kinesis Firehose.

Amazon Kinesis

Cribl LogStream supports receiving data records from Amazon Kinesis Streams.

> ℹ Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Data from Kinesis Streams

From the top nav of a LogStream instance or Group, select **Sources**, then select **Amazon > Kinesis** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Kinesis > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Kinesis Stream Source definition.

Stream name: Kinesis stream name (not ARN) to read data from.

Shard iterator start: Location at which to start reading a shard for the first time. Defaults to `Earliest Record`.

Record data format: Format of data inside the Kinesis Stream records. Gzip compression is automatically detected. Options include:

- Cribl (the default): Use this option if LogStream 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 – e.g., 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.

Region: Region where the Kinesis stream is located. Required.

Authentication

Use the **Authentication Method** buttons to 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**: If not running on AWS, you can select this option to enter your access key and secret key, directly or by reference.
- **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.

Auto 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 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 LogStream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

Enable for Kinesis Streams: Whether to use Assume Role credentials to access Kinesis Streams. Defaults to `No` .

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

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

- **Name**: Field name.
- **Value**: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline 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`.

Service Period: Time interval (in minutes) between consecutive service calls. Defaults to `1` minute.

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.

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes`.

Internal Fields

Cribl LogStream 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`

Amazon SQS

Cribl LogStream supports receiving events from Amazon Simple Queuing Service.

> ℹ Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Data from Amazon SQS

From the top nav of a LogStream instance or Group, select **Sources**, then select **Amazon > SQS** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **SQS > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this SQS Source definition.

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>'`. (E.g., `':port/<myQueueName>'`.)

Queue type: The queue type used (or created). Defaults to `Standard`. `FIFO` (First In, First Out) is the other option.

Create queue: If toggled to `Yes`, LogStream 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.

Authentication

Use the **Authentication Method** buttons to 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

If not running on AWS, you can select this option to enter your access key and secret key, directly or by reference. 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.

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 LogStream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

Enable for SQS: Whether to use Assume Role credentials to access SQS. Defaults to `No`.

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.

External ID: Enter the external ID to use when assuming role.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline 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`.

Max messages: 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`.

Num 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`.

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes`.

Internal Fields

Cribl LogStream 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`
- `__sqsSysAttrs`

The `_sqsSysAttrs` field can take on the following properties, which are reported to LogStream from SQS:

- `__sqsSysAttrs.ApproximateFirstReceiveTimestamp` : Returns the time (epoch time in milliseconds) the message was first received from the queue.
- `__sqsSysAttrs.ApproximateReceiveCount` : Returns the number of times a message has been received from the queue without being deleted.
- `__sqsSysAttrs.SenderId` : For an IAM user, returns the IAM user ID (e.g.: `ABCDEFGHI1JKLMNOPQ23R`). For an IAM role, returns the IAM role ID (e.g.: `ABCDE1F2GH3I4JK5LMNOP:i-a123b456`).
- `__sqsSysAttrs.SentTimestamp` : Returns the time (epoch time in milliseconds) the message was sent to the queue.
- `__sqsSysAttrs.MessageDeduplicationId` : Returns the value provided by the producer that calls the `SendMessage` action.
- `__sqsSysAttrs.MessageGroupId` : Returns the value provided by the producer that calls the `SendMessage` action – messages with the same `MessageGroupId` are returned in sequence.
- `__sqsSysAttrs.SequenceNumber` : Returns the sequence-number value provided by Amazon SQS.

- `__sqsSysAttrs.AWSTraceHeader` : Returns the AWS X-Ray trace header string.

For background on these message properties, see AWS' ReceiveMessage > Request Parameters documentation.

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 LogStream to create the queue)

Troubleshooting Notes

> ⚠ VPC endpoints for SQS might need to be set up in your account. Check with your administrator for details.

Amazon S3

Cribl LogStream supports receiving data from Amazon S3 buckets, using event notifications through SQS.

> ℹ Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **YES**

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 LogStream will retry. This means that although LogStream 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.

Configuring Cribl LogStream to Receive Data from Amazon S3

From the top nav of a LogStream instance or Group, select **Sources**, then select **Amazon > S3** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **S3 > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this S3 Source definition.

Queue: The name, URL, or ARN of the SQS queue to read events from. When specifying a non-AWS URL, you must use the format: `{url}/<queueName>` . (E.g., `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.

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.

Authentication

Use the **Authentication Method** buttons to 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: If not running on AWS, you can select this option to enter your access key and secret key, directly or by reference. 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.

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 LogStream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

Enable for S3: Whether to use Assume Role credentials to access S3. Defaults to `Yes`.

Enable for SQS: Whether to use Assume Role credentials when accessing SQS (Amazon Simple Queue Service). Defaults to `No`.

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: Defaults to `No`. Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Endpoint: S3 service endpoint. If empty, defaults to AWS's region-specific endpoint. Otherwise, used to point to an S3-compatible endpoint.

Signature version: Signature version to use for signing SQS requests. Defaults to `v4`.

Num 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`.

Max messages: 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`.

> ℹ LogStream 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.

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 to **Yes** to skip files that trigger a processing error. (E.g., corrupted files.) Defaults to **No**, which enables retries after a processing error.

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes` .

Internal Fields

Cribl LogStream 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`
- `__source`

How to Configure S3 to Send Event Notifications to SQS

> ℹ For step-by-step instructions, see AWS' Walkthrough: Configure a Bucket for Notifications (SNS Topic and SQS Queue).

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": "example-2020-04-20",
 "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" }
   }
  }
 ]
}
```

S3 and SQS Permissions

The following permissions are required on the S3 bucket:

- `s3:GetObject`
- `s3:ListBucket`

The following permissions are required on the SQS queue:

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:ChangeMessageVisibility`
- `sqs:GetQueueAttributes`
- `sqs:GetQueueUrl`

Best Practices

- When LogStream 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 LogStream 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. Which means, **each** Worker Process, in **each** LogStream Worker Node, will run 1 receiver consuming 1 message (i.e. 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 Notes

- VPC endpoints for SQS and for S3 might need to be set up in your account. Check with your administrator for details.

- If you're having connectivity issues, but no problems with the CLI, see if the AWS CLI proxy is in use. Check with your administrator for details.

Google Cloud Pub/Sub

Cribl LogStream supports receiving data records from Google Cloud Pub/Sub, a managed real-time messaging service for sending and receiving messages between applications.

> ℹ Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Data from Pub/Sub

From the top nav of a LogStream instance or Group, select **Sources**, then select **Google Cloud > Pub/Sub** from the **Data Sources** page's tiles or the Sources left nav. Click **+ Add New** to open the **Pub/Sub > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Pub/Sub Source definition.

Topic name: Enter the Pub/Sub topic from which to receive events.

Subscription name: Specify the subscription to use when receiving events.

Create topic: If toggled to `Yes`, LogStream will create the topic on Pub/Sub if it does not exist.

Create subscription: If set to `Yes` (the default), LogStream will create the subscription on Pub/Sub if it does not exist.

Ordered delivery: If toggled to `Yes`, LogStream will receive events in the order that they were added to the queue. (For this to work correctly, the process sending events must have ordering 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.)

Authentication

Use the **Authentication Method** buttons to select a Google authentication method:

Auto: This option uses the environment variables `PUBSUB_PROJECT` and `PUBSUB_CREDENTIALS`, and requires no configuration here.

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 LogStream's internal secrets manager or (if enabled) an external KMS.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

- **Name**: Field name.
- **Value**: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Max backlog: Limits the maximum number of events waiting to be processed before LogStream applies backpressure. Defaults to `10` events.

Request timeout (ms): Pull request timeout, in milliseconds. Defaults to `60000` ms (i.e., 1 minute).

Internal Fields

Cribl LogStream 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:

- `__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.

Azure Event Hubs

Cribl LogStream supports receiving data records from Azure Event Hubs.

> ℹ Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Data from Azure Event Hubs

From the top nav of a LogStream instance or Group, select **Sources**, then select **Azure > Event Hubs** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Azure Event Hubs > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this source definition.

Brokers: List of Event Hubs Kafka brokers to connect to, e.g., `yourdomain.servicebus.windows.net:9093` . Get the hostname from the host portion of the primary or secondary connection string in Shared Access Policies.

Event Hub name: The name of the Event Hub (a.k.a. Kafka Topic) to subscribe to.

Group ID: Specifies the name of the consumer group to which this Cribl LogStream instance belongs. Should always be `$Default` for Event Hubs.

From beginning: Whether to start reading from the earliest available data. Relevant only during initial subscription. Defaults to `Yes` .

TLS Settings (Client Side)

Enabled: Defaults to `Yes`.

Validate server certs: Whether to reject connections to servers without signed certificates. Defaults to `No` – and for Event Hubs, must always be disabled.

Authentication

Enabled: Defaults to `No`. When toggled to `Yes`:

- **SASL mechanism**: SASL (Simple Authentication and Security Layer) authentication mechanism to use. Currently, `PLAIN` is the only mechanism supported for Event Hubs Kafka brokers.

- **Username**: The username for authentication. For Event Hubs, this should always be `$ConnectionString`.

- **Password**: Event Hubs primary or secondary connection string. From Microsoft's documentation, the format is:

  `Endpoint=sb://<FQDN>/;SharedAccessKeyName=<KeyName>;SharedAccessKey=<KeyValue>`

  Example entry:

  `Endpoint=sb://dummynamespace.servicebus.windows.net/;SharedAccessKeyName=dummyaccesskeyname;SharedAccessKey=5dOntTRytoC24opYThisAsit3is2B+OGY1US/fuL3ly=`

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Azure Blob Storage

Cribl LogStream supports receiving data from Azure Blob Storage buckets, using Azure Event Grid to receive notifications, via a queue, when new blobs are added to a storage account.

> ℹ Type: **Pull** | TLS Support: **YES** (secure API) | Event Breaker Support: **YES**
> Available in: LogStream 2.4.4 and above.

Configuring Cribl LogStream to Receive Data from Azure Blob Storage

From the top nav of a LogStream instance or Group, select **Sources**, then select **Azure** > **Blob Storage** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Azure Blob Storage** > **New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Azure Blob Storage Source definition.

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. E.g., referencing a Global Variable: `myQueue-${C.vars.myVar}` .

Connection string: Enter your Azure storage account Connection String. If left blank, LogStream will fall back on `env.AZURE_STORAGE_CONNECTION_STRING` .

Filename filter: Regex matching file names to download and process. Defaults to `.*`, to match all characters.

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: Defaults to `No`. Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Max messages: 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).

> LogStream 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.

Num 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 duration (in seconds) which pollers should be validated and restarted if exited. Defaults to `5` seconds.

Skip file on error: Toggle to **Yes** to skip files that trigger a processing error (e.g., corrupted files). Defaults to **No**, which enables retries after a processing error.

Internal Fields

Cribl LogStream 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`
- `__source`

> 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 LogStream 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

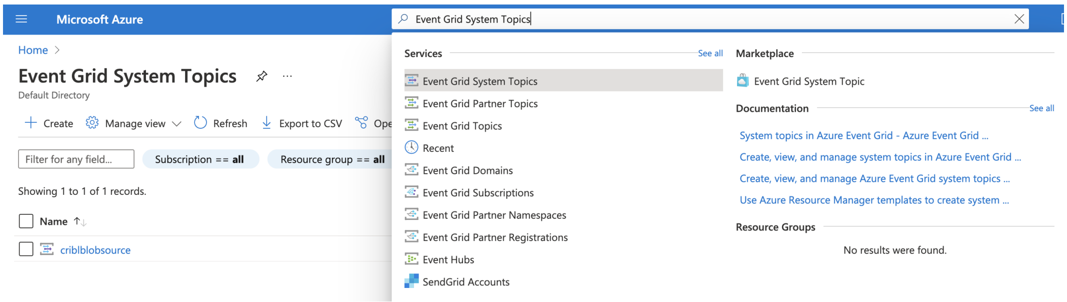
To configure notifications from the Blob storage account in the Azure backend, there are three major steps, outlined below:

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.

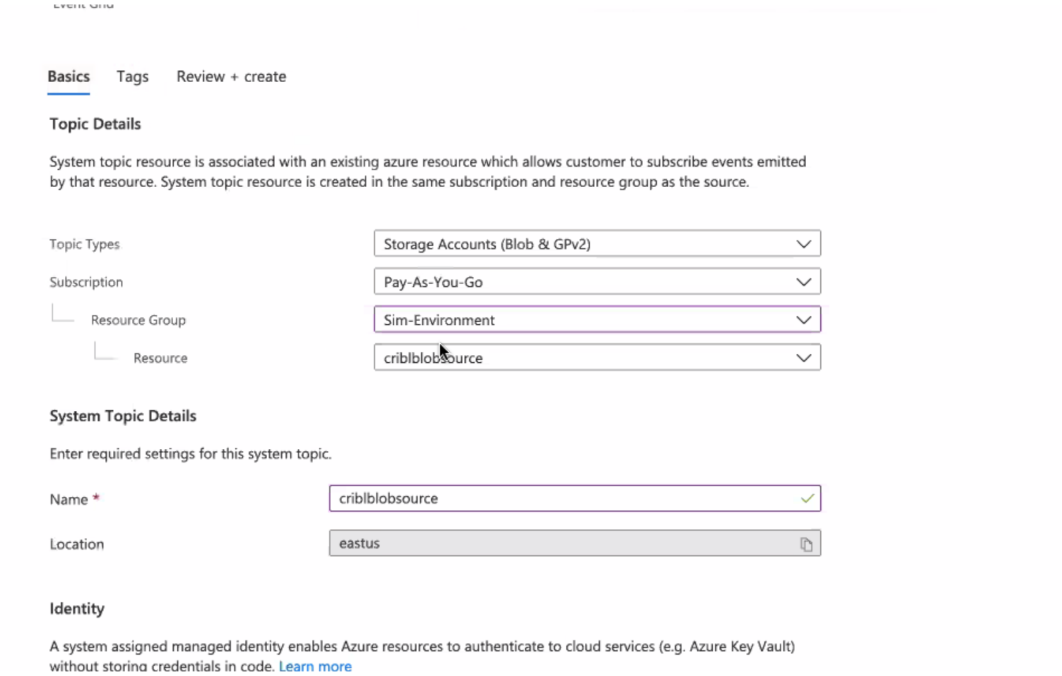
1. Create System Topic

First, you must create a system topic, to which Azure will publish notifications. In the Azure portal, tart 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)**:



Topic Details

System topic resource is associated with an existing azure resource which allows customer to subscribe events emitted by that resource. System topic resource is created in the same subscription and resource group as the source.

Topic Types	Storage Accounts (Blob & GPv2)
Subscription	Pay-As-You-Go
Resource Group	Sim-Environment
Resource	criblblobsource

System Topic Details

Enter required settings for this system topic.

Name *	criblblobsource
Location	eastus

Identity

A system assigned managed identity enables Azure resources to authenticate to cloud services (e.g. Azure Key Vault) without storing credentials in code. Learn more

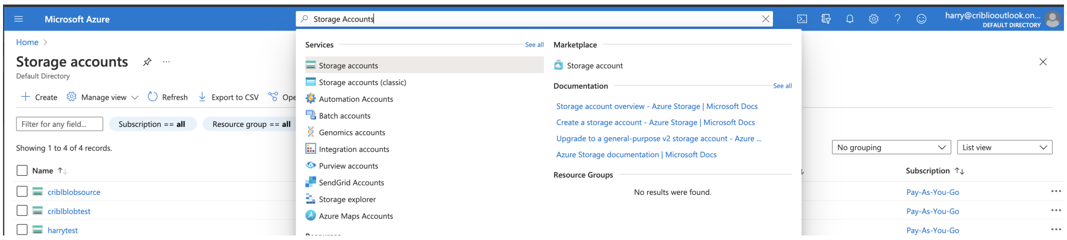
Creating a system topic

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.



Storage Accounts

Storage accounts
Default Directory

Filter for any field... Subscription == **all** Resource group == **all**

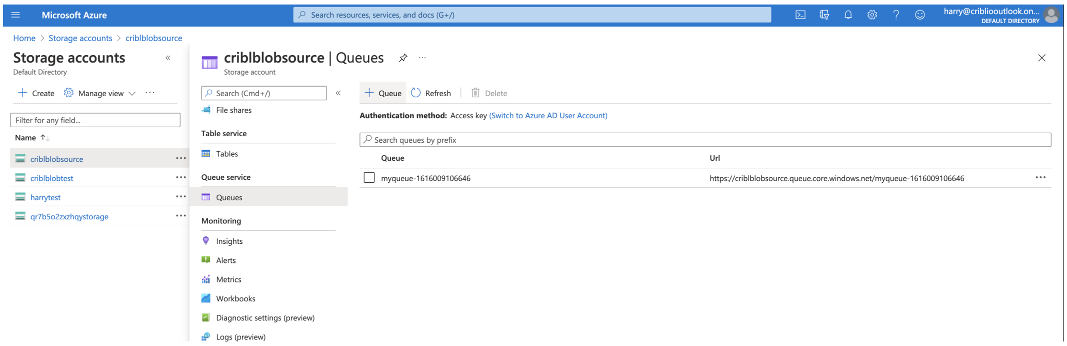
Showing 1 to 4 of 4 records.

No results were found.

Name	Subscription
criblblobsource	Pay-As-You-Go
criblblobtest	Pay-As-You-Go
harrytest	Pay-As-You-Go

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**:

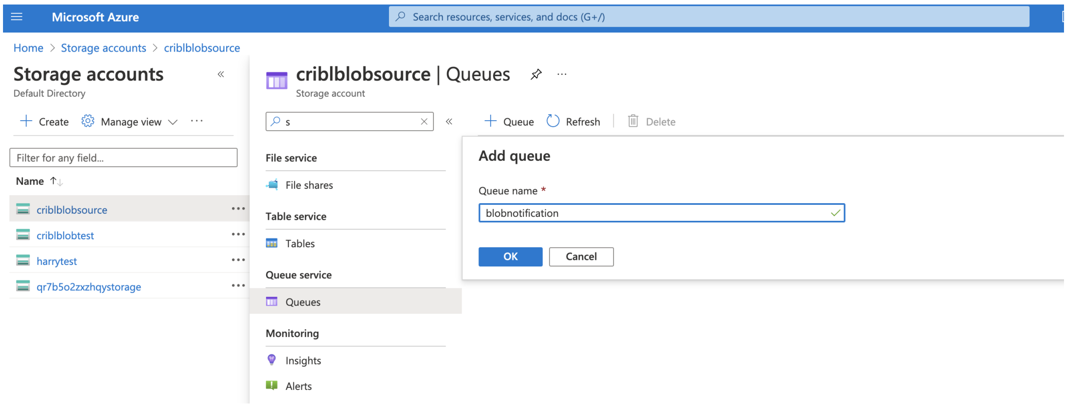


Authentication method: Access key (Switch to Azure AD User Account)

	Queue	Url
□	myqueue-1616009106646	https://criblblobsource.queue.core.windows.net/myqueue-1616009106646

Accessing queues

Select **Create queue**, and give the queue a name that is meaningful to you.



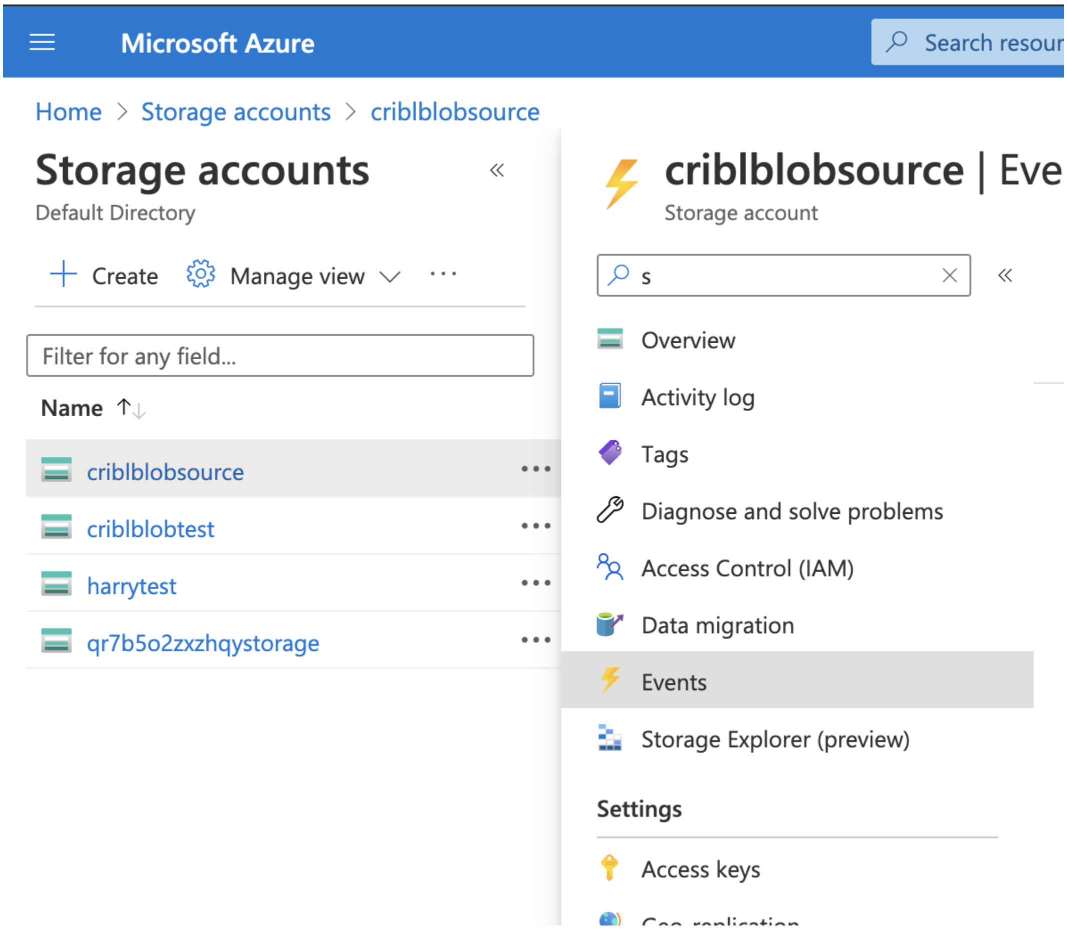
Add queue

Queue name *: blobnotification

Adding a queue

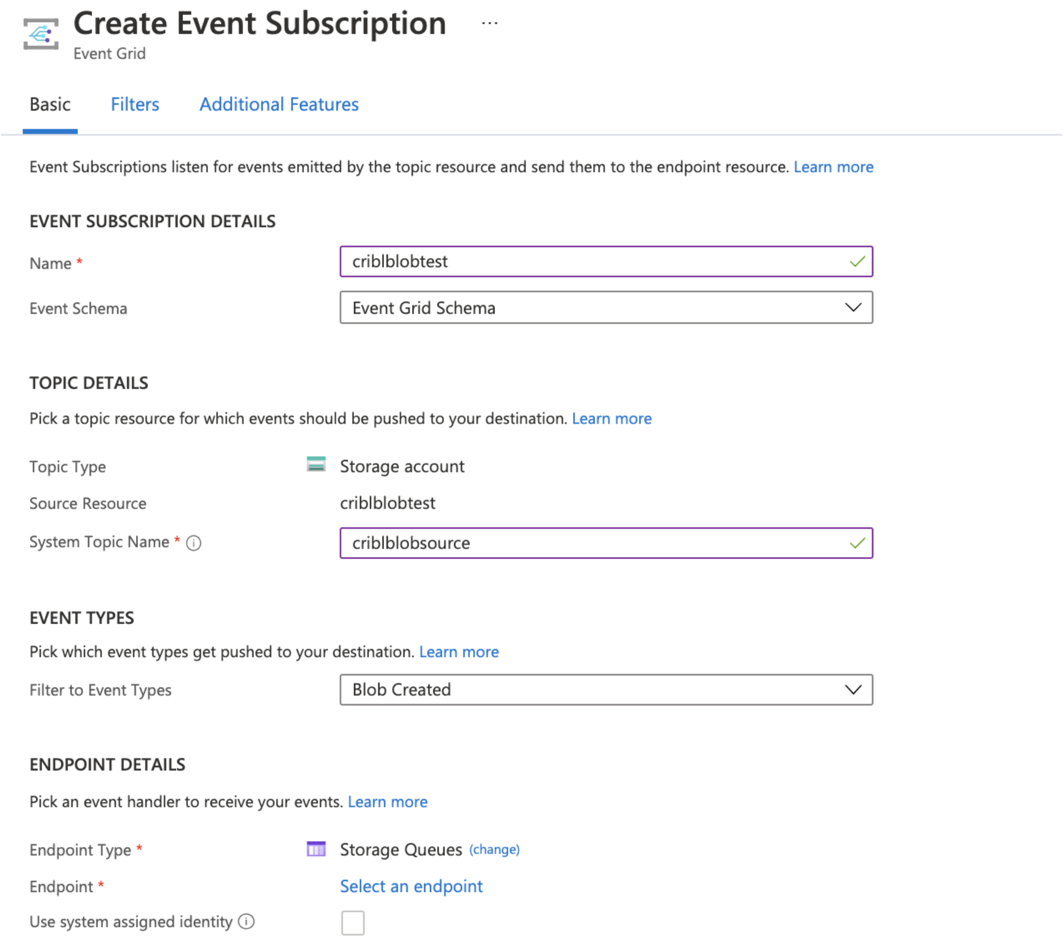
3. Configure Storage Account Notifications

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:



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 *: criblblobtest ✓
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: Blob Created

ENDPOINT DETAILS

Pick an event handler to receive your events. Learn more

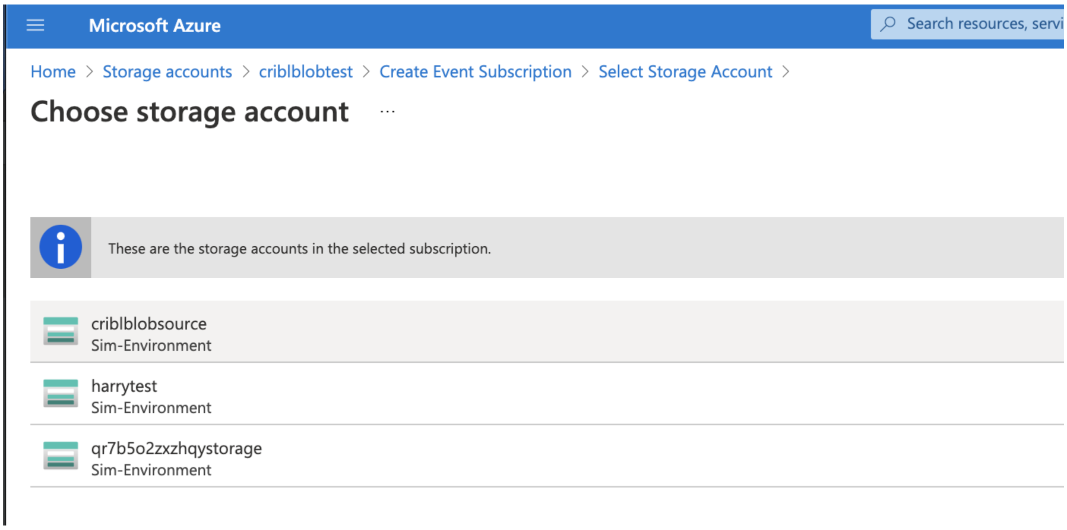
Endpoint Type *: Storage Queues (change)
Endpoint *: Select an endpoint
Use system assigned identity ⓘ: □

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:


Microsoft Azure
Search resources, servi
Home > Storage accounts > criblblobtest > Create Event Subscription > Select Storage Account >

Choose storage account

These are the storage accounts in the selected subscription.

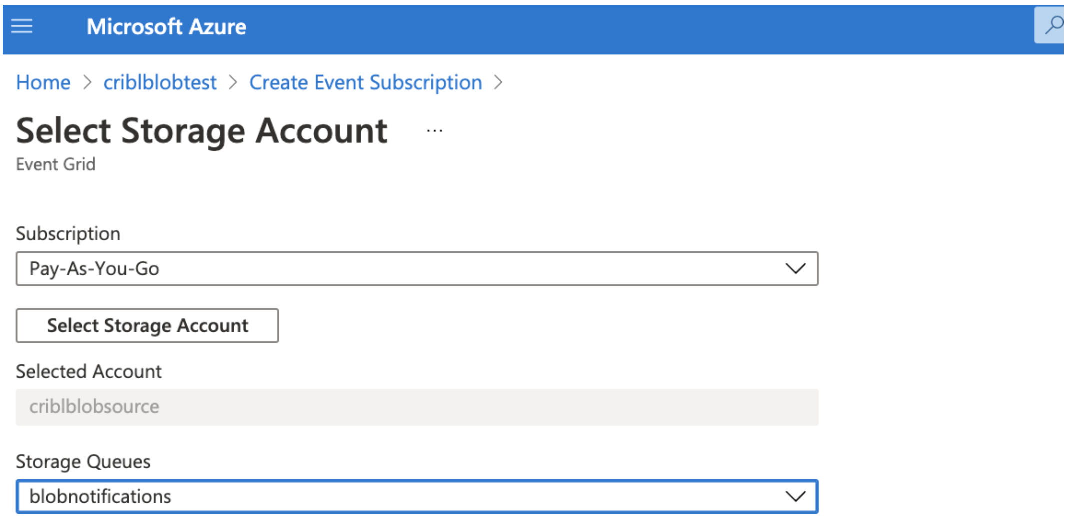
criblblobsource
Sim-Environment

harrytest
Sim-Environment

qr7b5o2zxzhqystorage
Sim-Environment

Choosing the storage account

Select the queue you created in Create Storage Queue above, and click **Confirm Selection** to save the settings.



Select Storage Account

Event Grid

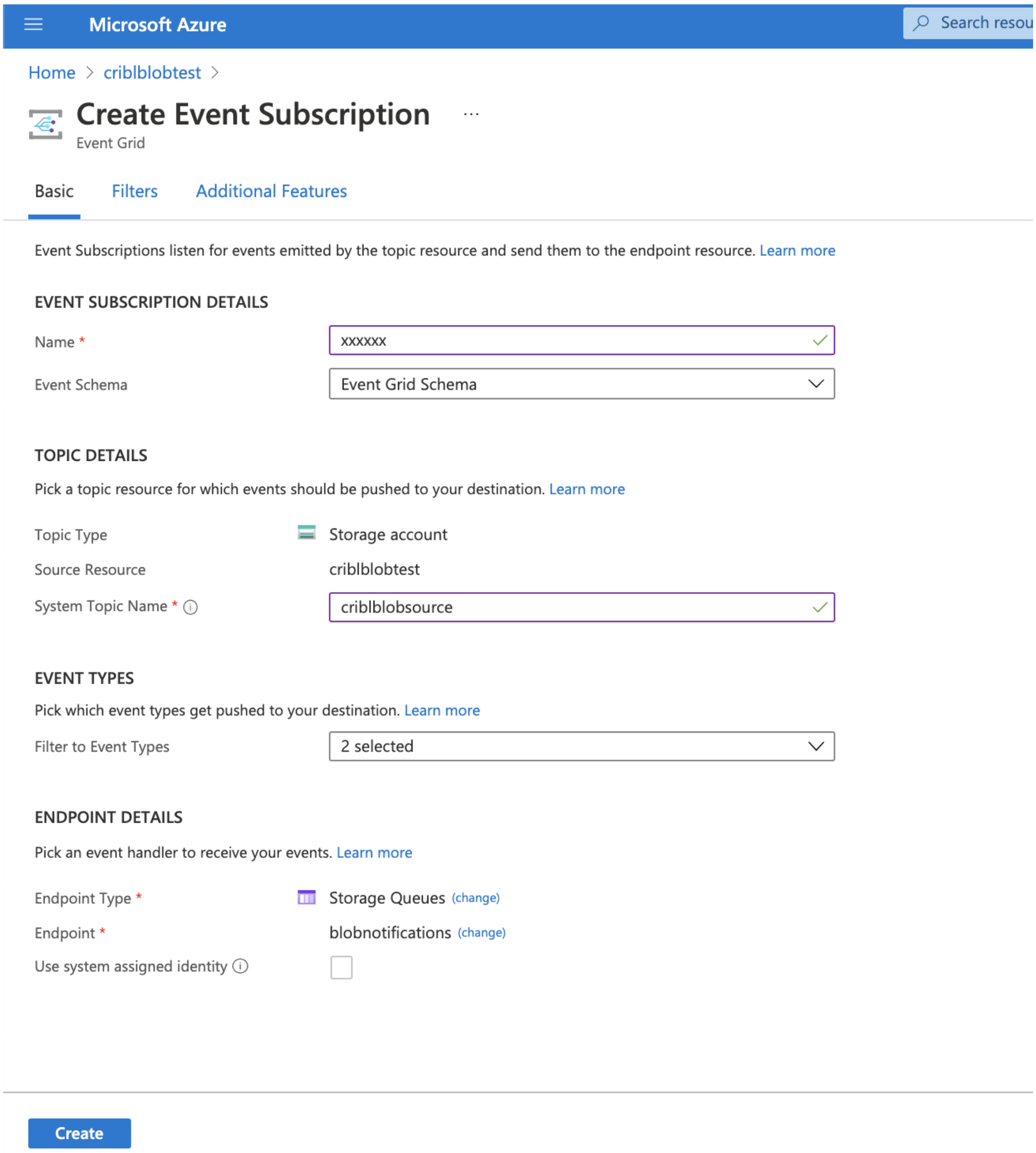
Subscription: Pay-As-You-Go

Selected Account: criblblobsource

Storage Queues: blobnotifications

Selecting the storage account

To complete the process, click **Create.**



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)

Endpoint * : blobnotifications (change)

Use system assigned identity ⓘ: □

Create

Creating the subscription

Office 365 Services

Cribl LogStream supports receiving data from the Office 365 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.

> ℹ Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **YES**

TLS is enabled via the HTTPS protocol on this Source's underlying REST API.

Configuring Cribl LogStream to Receive Data from Office 365 Services

From the top nav of a LogStream instance or Group, select **Sources**, then select **Office 365 > Services** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Services > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Office 365 Services definition.

Tenant ID: Enter the Office 365 Azure tenant ID.

App ID: Enter the Office 365 Azure application ID.

Client secret: Enter the Office 365 Azure client secret.

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.
- **Historical Status**: Get a historical view of service incidents.

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 this to `Yes` 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, LogStream uses the **Interval** field's value to establish the search date range and the cron schedule (e.g.: `*/${interval} * * * *`).

Therefore, intervals set in minutes – those for **Current Status** and **Historical 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.

LogStream will reject intervals like `23`, `42`, or `45`, or `75` – which would yield non-integer results, meaning unpredictable schedules.

The **Historical Status** service polls only once per day. So here, the **Interval** field's value simply establishes the hour of the day at which to poll.
(In distributed deployments, this time is set based on the Leader Node's system time. In single-instance deployments, it is set based on the API server's time zone.)

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Keep Alive Time (seconds): How often Workers should check in with the scheduler to keep their job subscription alive. Defaults to `60`.

Worker timeout (periods): The number of Keep Alive Time periods before an inactive Worker will have its job subscription revoked. Defaults to `3`.

Internal Fields

Cribl LogStream 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`

Office 365 Activity

Cribl LogStream supports receiving data from the Office 365 Management Activity API. This facilitates analyzing actions and events on Azure Active Directory, Exchange, and SharePoint, along with global auditing and Data Loss Prevention data.

> ℹ Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **YES**

TLS is enabled via the HTTPS protocol on this Source's underlying REST API.

Configuring Cribl LogStream to Receive Data from Office 365 Activity

From the top nav of a LogStream instance or Group, select **Sources**, then select **Office 365 > Activity** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Activity > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Office 365 Activity definition.

Tenant ID: Enter the Office 365 Azure tenant ID.

App ID: Enter the Office 365 Azure application ID.

Client secret: Enter the Office 365 Azure client secret.

Subscription Plan: Select the Office 365 subscription plan for your organization. This is typically `Enterprise and GCC Government Plan`.

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, this section provides the following controls:

Enabled: Toggle this to `Yes` 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 Management Activity API, LogStream uses the **Interval** field's value to establish the search date range and the cron schedule (e.g.: `*/${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.

LogStream will reject intervals like `23`, `42`, or `45`, or `75` – which would yield non-integer results, meaning unpredictable schedules.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Keep Alive Time (seconds): How often Workers should check in with the scheduler to keep their job subscription alive. Defaults to `60`.

Worker timeout (periods): The number of Keep Alive Time periods before an inactive Worker will have its job subscription revoked. Defaults to `3`.

Internal Fields

Cribl LogStream 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`

Office 365 Message Trace

Cribl LogStream 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.

> i Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **YES**

TLS is enabled via the HTTPS protocol on this Source's underlying REST API.

Configuring Cribl LogStream to Receive Office 365 Message Trace Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **Office 365 > Message Trace** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Message Trace > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Office 365 Message Trace definition.

Report URL: Enter the URL to use when retrieving report data. Defaults to: `https://reports.office365.com/ecp/reportingwebservice/reporting.svc/MessageTrace` .

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.

Username: Username with which to run the Message Trace API call.

Password: Password with which to run the Message Trace API call.

Date range start: The relative time in the past that begins the search date range. (E.g., `-3h@h` .) Message Trace data is delayed; this parameter (with **Date range end**) compensates for delay and gaps.

Date range end: The relative time in the past that ends the search date range. (E.g., `-2h@h` .) Message Trace data is delayed; this parameter (with **Date range start**) compensates for delay and gaps.

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` .)

About Polling Intervals

To poll the Office 365 Message Trace API, LogStream uses the **Poll interval** field's value to establish the cron schedule. (e.g.: `*/${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.

LogStream will reject intervals like `23` , `42` , or `45` , or `75` – which would yield non-integer results, meaning unpredictable schedules.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Keep Alive time (seconds): How often Workers should check in with the scheduler to keep their job subscription alive. Defaults to `60` .

Worker timeout (periods): The number of Keep Alive Time periods before an inactive Worker will have its job subscription revoked. Defaults to `3` .

Timeout (secs): Maximum time to wait for an individual Message Trace API request to complete, Defaults to `600` seconds (10 minutes). Enter `0` to disable. Because there is a single request to the Message Trace API per page of data, this timeout is applied at the page (request) level.

Internal Fields

Cribl LogStream 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`

TCP JSON

Cribl LogStream supports receiving of data over TCP in JSON format (see protocol below).

> ℹ Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive TCP JSON Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **TCP JSON** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **TCP JSON > New Source** modal, which provides the fields outlined below.

> LogStream 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.

General Settings

Input ID: Enter a unique name to identify this TCP JSON Source definition.

Address: Enter hostname/IP to listen for TCP JSON data. E.g., `localhost` or `0.0.0.0`.

Port: Enter the port number to listen on.

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` (i.e., all IPs).

Shared secret (authToken): Shared secret to be provided by any client (in `authToken` header field). Click **Generate** to create a new secret. If empty, unauthenticated access will be permitted.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path at which to find 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. Defaults to `No`. When toggled to `Yes`:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

- **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. E.g., 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.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

- **Enable proxy protocol**: Toggle to `Yes` if the connection is proxied by a device that supports Proxy Protocol v1 or v2.

Internal Fields

Cribl LogStream 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`

Format

LogStream expects TCP JSON events in newline-delimited JSON format:

1. A header line. Can be empty – e.g., `{}` . 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.

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", "
{"host":"myOtherHost", "source":"myOtherSource", "_raw": "{\"message\":\"S
```

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``).

Example

1. Configure Cribl LogStream 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 LogStream host: `cat test.json | nc <myCriblHost> 10001`

TCP (RAW)

Cribl LogStream supports receiving of data over TCP. (See examples and header options below.)

> Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

Configuring Cribl LogStream to Receive TCP Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **TCP** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **TCP > New Source** modal, which provides the fields outlined below.

> LogStream 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.

General Settings

Input ID: Enter a unique name to identify this TCP Source definition.

Address: Enter hostname/IP to listen for raw TCP data. E.g., `localhost` or `0.0.0.0`.

Port: Enter port number.

IP allowlist regex: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` (i.e,. all IPs).

Enable Header: Toggle to `Yes` 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, etc. Header format: `{ "authToken" : "myToken", "fields": { "field1": "value1", "field2": "value2" }}`.

- **Shared secret (authToken)**: Shared secret to be provided by any client (in `authToken` header field). Click **Generate** to create a new secret. If empty, unauthenticated access will be permitted.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path at which to find 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. Defaults to `No`. When toggled to `Yes`:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

- **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. E.g., 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.

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: Defaults to `No`. When toggled to `Yes`:

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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

- **Name**: Field name.
- **Value**: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable proxy protocol: Defaults to `No`. Toggle to `Yes` if the connection is proxied by a device that supports Proxy Protocol V1 or V2.

Internal Fields

Cribl LogStream 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 Example

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.

Sample raw TCP test

```
{"authToken":"myToken42", "fields": {"region": "us-east-1", "AZ":"az1"}}

this is event number 1
this is event number 2
```

Enabling the Example

1. Configure LogStream 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 LogStream host, using this command: `cat test.raw | nc <myCriblHost> 7777`

HTTP/S (Bulk API)

Cribl LogStream supports receiving data over HTTP/S using the Cribl Bulk API, Splunk HEC, or Elastic Bulk API.

> Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Data over HTTP(S)

From the top nav of a LogStream instance or Group, select **Sources**, then select **HTTP** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **HTTP > New Source** modal, which provides the fields outlined below.

> LogStream 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.

General Settings

Input ID: Enter a unique name to identify this HTTP(S) Source definition.

Address: Enter the hostname/IP on which to listen for HTTP(S) data. (E.g., `localhost` or `0.0.0.0`.)

Port: Enter the port number.

Auth tokens: Shared secrets to be provided by any client (Authorization: <token>). Click **Generate** to create a new secret. If empty, unauthenticated access **will be permitted**.

Cribl HTTP event API: Absolute path on which to listen for Cribl HTTP API requests. Currently, the only supported option is the default `/cribl`, which LogStream expands as `/cribl/_bulk`. Use an empty string to disable. Maximum payload size is 2MB.

Elastic API endpoint (for Bulk API): Absolute path on which to listen for Elasticsearch API requests. Currently, the only supported option is the default `/elastic`, which LogStream expands as `/elastic/_bulk`. Other entries are faked as success. Use an empty string to disable.

> ℹ 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: 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`.

> ℹ This Splunk HEC implementation is an **event** (i.e., 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.)
>
> 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.

Splunk HEC acks: Whether to enable Splunk HEC acknowledgements. Defaults to `No`.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS` .

CA certificate path: Server path at which to find 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. Defaults to `No` . When toggled to `Yes` :

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No` .

- **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. E.g., 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.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Internal Fields

Cribl LogStream 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`
- `__id` (Elastic In)
- `__type` (Elastic In)
- `__index` (Elastic In)
- `__host` (Elastic In)

Format and Endpoint

LogStream 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", "s
{"_time":1541280341, "host":"myOtherHost", "source":"myOtherSource", "_raw
```

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

For the following examples:

1. Configure Cribl to listen on port `10080` for HTTP (default). Set `authToken` to `myToken42` .
2. Send a payload to your Cribl LogStream receiver.

Cribl Endpoint – Single Event

Cribl Single Event Example:

```
curl -k http://<myCriblHost>:10080/cribl/_bulk -H 'Authorization: myToken4
```

Cribl Endpoint – Multiple Events

Cribl Endpoint - Multiple Events

```
curl -k http://<myCriblHost>:10080/cribl/_bulk -H 'Authorization: myToken4
```

Splunk HEC Event Endpoint

Splunk HEC Event Endpoint

```
curl -k http://<myCriblHost>:10080/services/collector/event -H 'Authorizat
```

```
curl -k http://<myCriblHost>:10080/services/collector -H 'Authorization: m
```

> For Splunk HEC, the token specification can be either `Splunk <token>` or `<token>` .

Raw HTTP/S

Cribl LogStream 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.

> ℹ Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

Configuring Cribl LogStream to Receive Raw HTTP Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **Raw HTTP** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Raw HTTP > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Raw HTTP Source definition.

Address: Enter the address to bind on. Defaults to `0.0.0.0` (all addresses).

Port: Enter the port number to listen on.

Auth tokens: Shared secrets to be provided by any client. Click **Generate** to create a new secret. If empty, permits open access.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS` .

CA certificate path: Server path at which to find 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. Defaults to `No` . When toggled to `Yes` :

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No` .
- **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. E.g., 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.

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: The amount of time (in milliseconds) that 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. Defaults to `10000` .

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

- **Name**: Field name.
- **Value**: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Allowed URI paths: List of URI paths accepted by this input. Supports wildcards, e.g., `/api/v*/hook` . Defaults to `*` , which allows all paths.

Allowed HTTP methods: List of HTTP methods accepted by this input. Supports wildcards, e.g., `P*, GET` . Defaults to `*` , which allows all methods.

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256` . Enter `0` for unlimited.

Internal Fields

Cribl LogStream 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`

Elasticsearch API

Cribl LogStream supports receiving data over HTTP/S using the Elasticsearch Bulk API. (See the Configuring Filebeat example below.)

> Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring LogStream to Receive Data over HTTP(S), Using the Elasticsearch Bulk API Protocol

From the top nav of a LogStream instance or Group, select **Sources**, then select **Elasticsearch API** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Elasticsearch API > New Source** modal, which provides the fields outlined below.

> LogStream 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.

General Settings

Input ID: Enter a unique name to identify this Elasticsearch Source definition.

Address: Enter the hostname/IP on which to listen for Elasticsearch data. (E.g., `localhost` or `0.0.0.0`.)

Port: Enter the port number.

Auth tokens: Shared secrets to be provided by any client (Authorization: <token>). Click **Generate** to create a new secret. If empty, unauthenticated access **will be permitted**.

Elasticsearch API endpoint (for Bulk API): Absolute path on which to listen for Elasticsearch API requests. Defaults to `/`. LogStream automatically appends `_bulk`, so (e.g.) `/myPath` becomes `/myPath/_bulk`. Requests could then be made to either `/myPath/_bulk` or `/myPath/<myIndexName>/_bulk`. Other entries are faked as success.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path at which to find 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. Defaults to `No`. When toggled to `Yes`:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.
- **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. E.g., 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.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Field Normalization

The Elasticsearch API input normalizes the following fields:

- `@timestamp` becomes `_time` at millisecond resolution.
- `host` is set to `host.name`.
- Original object `host` is stored in `__host`.

The Elasticsearch Destination does the reverse, and it also recognizes the presence of `__host`.

Internal Settings

Cribl LogStream 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`
- `__id`
- `__type`

- `__index`
- `__host`

Configuring Filebeat

To set up Filebeat to send data to LogStream, use its Elasticsearch output. If an Auth Token is configured here, add it in Filebeat configuration under `output.elasticsearch.headers` , as in this example:

...filebeat.yml

```
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["http://<LOGSTREAM_HOST>:9200/elastic"]

output.elasticsearch.headers:
  Authorization: "myToken42"
```

Kafka

Cribl LogStream supports receiving data records from a Kafka cluster.

> ℹ Type: **Pull** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Data from Kafka Topics

From the top nav of a LogStream instance or Group, select **Sources**, then select **Kafka** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Kafka > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Source definition.

Brokers: List of Kafka brokers to use, e.g., `localhost:9092` .

Topics: List of topics to subscribe to.

Group ID: The name of the consumer group to which this Cribl LogStream instance belongs.

From beginning: Whether to start reading from the earliest available data. Relevant only during initial subscription. Defaults to `Yes` .

TLS Settings (Client Side)

Enabled: defaults to `No` . When toggled to `Yes` :

Autofill?: This setting is experimental.

Validate client certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

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.

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

This section governs SASL (Simple Authentication and Security Layer) authentication.

Enabled: Defaults to `No`. When toggled to `Yes`:

SASL mechanism: Use this drop-down to select the SASL authentication mechanism to use.

Username: Enter the username for your account.

Password: Enter the account's password.

Schema Registry

This section governs Kafka Schema Registry Authentication for AVRO-encoded data with a schema stored in the Confluent Schema Registry.

Enabled: defaults to `No`. When toggled to `Yes`:

Schema registry URL: URL for access to the Confluent Schema Registry. (E.g., `http://<hostname>:8081`.)

TLS enabled: defaults to `No`. When toggled to `Yes,` displays the following TLS settings for the Schema Registry:

> ℹ These have the same format as the TLS Settings (Client Side) above.

TLS Settings (Schema Registry)

Validate server certs: Reject certificates that are not authorized by a CA specified in the **CA Certificate Path** field. Defaults to `No`.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

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.

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.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Internal Fields

Cribl LogStream 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)
- `__schemaId` (when using Schema Registry)

Metrics

Cribl LogStream supports receiving metrics in these wire formats/protocols: StatsD, StatsD Extended, and Graphite. Automatic protocol detection will happen on the first line received over a TCP connection or a UDP packet. Lines not matching the detected protocol will be dropped.

> ℹ Type: **Push** | TLS Support: **No** | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Metrics

From the top nav of a LogStream instance or Group, select **Sources**, then select **Metrics** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Metrics > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Source definition.

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.

TCP port: Enter the TCP port number to listen on. Not required if listening on UDP.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable proxy protocol: Defaults to `No` . Toggle to `Yes` 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.)

Max buffer size (events) : Maximum number of events to buffer when downstream is blocking. Defaults to `1000` .

Internal Fields

Cribl LogStream 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:

Text

```
_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
dim3 - value of dimension2
....
```

LogStream 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

SNMP Trap

Cribl LogStream supports receiving data from SNMP Traps.

> Type: **Push** | TLS Support: **NO** | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive SNMP Traps

From the top nav of a LogStream instance or Group, select **Sources**, then select **SNMP Trap** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **SNMP Trap > New Source** pane, which provides the fields outlined below.

> LogStream 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.

General Settings

Input ID: Enter a unique name to identify this Source definition.

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`.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline 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 send data. Defaults to `.*` i.e. all IPs.

Max buffer size (events) : Maximum number of events to buffer when downstream is blocking. Defaults to `1000` .

Internal Fields

Cribl LogStream 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`
- `__snmpVersion` : Acceptable values are `0` , `2` , `3` . Versions: `0` =v1, `2` =v2c, `3` =v3.
- `__srcIpPort` : <hostname>|port
- `__snmpRaw` : Buffer containing Raw SNMP packet

Considerations for Working with SNMP Trap Data

- It's possible to work with SNMP metadata (i.e., we'll decode the packet). Options include dropping, routing, 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.

- SNMP packets can be forwarded to non-SNMP destinations (e.g., Splunk, Syslog, S3, etc.).
- Non-SNMP input data **cannot** be sent to SNMP destinations.

Prometheus

Cribl LogStream supports receiving data from Prometheus.

> i Type: **Pull** | TLS Support: **No** | Event Breaker Support: **No**

Configuring Cribl LogStream to Receive Data from Prometheus

From the top nav of a LogStream instance or Group, select **Sources**, then select **Prometheus** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Prometheus > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Source definition.

Extra dimensions: Dimensions to include in events. By default, `host` and `source` are included.

Discovery type: Target discovery mechanism. Use `Static` (the default) to manually enter a list of targets. Select `DNS` or `AWS EC2` options enable dynamic discovery of endpoints to scrape. Your selection determines which fields are displayed lower in this section:

- **Targets**: Displayed for **Discovery type**: `Static`. List of Prometheus targets to pull metrics from, values can be in URL or host[:port] format. For example: http://localhost:9090/metrics, localhost:9090, or localhost. In the cases where just host[:port] are specified, the endpoint will resolve to 'http://host[:port]/metrics'.

- **DNS names:** Displayed for **Discovery type**: `DNS` . Enter a list of DNS names to resolve.

- **Record type**: Displayed for **Discovery type**: `DNS` . Select the DNS record type to resolve. Defaults to `SRV` (Service record). Other options are `A` or `AAA` record.

- **Region**: Displayed for **Discovery type**: `AWS EC2` . Select the AWS region in which to discover the EC2 instances with metrics endpoints to scrape.

Poll interval: How often (in minutes) to scrape targets for metrics. Defaults to `15` . This value must be an integer that divides evenly into `60` minutes.

Log level: Set the verbosity level to one of `debug` , `info` (the default), `warn` , or `error` .

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Keep alive time (seconds): How often workers should check in with the scheduler to keep job subscription alive. Defaults to `60` seconds.

Worker timeout (periods) : How many **Keep alive time** periods before an inactive worker's job subscription will be revoked. Defaults to `3` periods.

Internal Fields

Cribl LogStream 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`

Datagens

Cribl LogStream supports generating of data from datagen files. See Using Datagens for more details.

> ℹ Type: **Internal** | TLS Support: **N/A** | Event Breaker Support: **No**

Configuring Cribl LogStream to Generate Sample Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **Datagens** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Datagens > New Source** pane, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this Source definition.

Datagens: List of datagens.

- **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. Defaults to `10`.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Internal Fields

Cribl LogStream 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`

AppScope

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.

> ℹ Type: **Push** | TLS Support: **YES** | Event Breaker Support: **YES**

Configuring LogStream to Receive AppScope Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **AppScope** from the **Data Sources** page's tiles or the **Sources** left nav.
Click **+ Add New** to open the **AppScope > New Source** modal, which provides the following fields.

General Settings

Input ID: Enter a unique name to identify this AppScope Source definition.

Address: Enter the hostname/IP on which to listen for AppScope data. (E.g., `localhost`.) Defaults to `0.0.0.0`, meaning all addresses.

Port: Enter the port number to listen on.

IP whitelist regex: Regex matching IP addresses that are allowed to establish a connection.

Auth tokens: Shared secrets to be provided by any client (in `authToken` header field). Click **Generate** to create a new secret. If empty, unauthenticated access **will be permitted**.

TLS Settings (Server Side)

Enabled defaults to `No`. When toggled to `Yes`:

Certificate name: Name of the predefined certificate.

Private key path: Path on server where to find the private key to use in PEM format. Path can reference $ENV_VARS.

Passphrase: Passphrase to use to decrypt private key.

Certificate path: Server path at which to find certificates (in PEM format) to use. Path can reference `$ENV_VARS`.

CA certificate path: Server path at which to find 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. Defaults to `No`. When toggled to `Yes`:

- **Validate client certs**: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

- **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. E.g., 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.

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: The amount of time (in milliseconds) that 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. Defaults to `10000` .

Fields (Metadata)

In this section, you can add fields/metadata to each event using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Advanced Settings

Enable proxy protocol: Enable if the connection is proxied by a device that supports Proxy Protocol v1 or v2.

Internal Settings

Cribl LogStream 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`

Cribl Internal

The Cribl Internal Source enables you to capture and send LogStream's own internal **logs** and **metrics** through Routes and Pipelines.

> ℹ Type: **Internal** | TLS Support: **N/A** | Event Breaker Support: **No**

Scope

In distributed mode, only Worker Process internal logs can be processed through this Source. (Logs on the Leader remain on the Leader, since the Leader Node is not part of any processing path.)

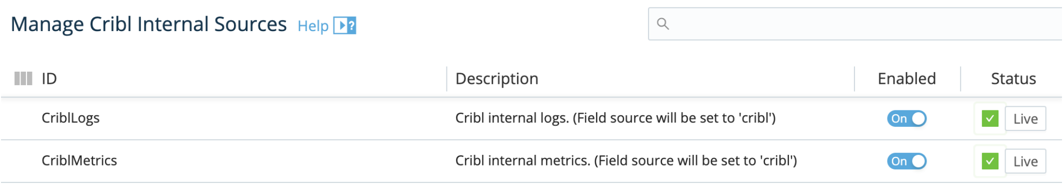
In both distributed and single-instance mode, this Source omits API Process logs, meaning that it omits telemetry/license-validation traffic. You can, however, use a Script Collector to check for API Server (or Worker Group) events.

Configuring Cribl Internal Logs/Metrics as a Data Source

From the top nav of a LogStream instance or Group, select **Sources**, then select **Cribl Internal** from the **Data Sources** page's tiles or the **Sources** left nav.

Next, on the **CriblLogs** and/or the **CriblMetrics** row, slide the **Enabled** slider to `On` . Confirm your choice in the resulting message box.

To proceed to the configuration options listed below, click anywhere on the **CriblLogs** or the **CriblMetrics** row.



Manage Cribl Internal Sources Help

ID	Description	Enabled	Status
CriblLogs	Cribl internal logs. (Field source will be set to 'cribl')	On	✓ Live
CriblMetrics	Cribl internal metrics. (Field source will be set to 'cribl')	On	✓ Live

Cribl Internal Sources – click to configure

CriblLogs Settings

General Settings

Enabled: This duplicates the parent page's **Enabled** slider. Keep it at `Yes` to enable Cribl logs as a Source.

Input ID: Enter a unique name to identify this CriblLogs Source definition.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

CriblMetrics Settings

General Settings

Enabled: This duplicates the parent page's **Enabled** slider. Keep it at `Yes` to enable Cribl metrics as a Source.

Input ID: Enter a unique name to identify this CriblMetrics Source definition.

Metric name prefix: Enter an optional prefix that will be applied to metrics provided by LogStream. The prefix defaults to `cribl.logstream`.

> ℹ If LogStream detects `source`, `sourcetype`, `host`, or `index` fields in metrics from external sources, it copies their values into new dimensions with added `event_` prefixes (e.g., `event_sourcetype`). This leaves the original dimensions (and their values) intact.
>
> Note that you can disable metric collection for any or all of these four fields by specifying them in **Settings > System > General Settings > Limits > Disable field metrics**.

Processing Settings

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute field's value (can be a constant).

Pre-Processing

In this section's **Pipeline** drop-down list, you can select a single existing Pipeline to process data from this input before the data is sent through the Routes.

Reporting Metrics Less Frequently

By default, LogStream 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 LogStream.

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.

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.

Collectors

Collectors enable you to dispatch on-demand collection tasks that fetch data from local or remote locations. As of v.2.3, LogStream supports scheduled collection jobs. These recurring jobs can make batch collection of stored data more like continual processing of streaming data.

How Do Collectors Work

You can configure a LogStream Node to retrieve data from a remote system by selecting **Collectors** from the top nav. Data collection is a multi-step process:

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. (E.g., the target will be a directory if the type is Filesystem, or an S3 bucket/path if the type is Amazon S3.)

Next, schedule or manually run the Collector. In this step, you configure scheduled-job–specific or run-specific settings – such as the run Mode (such as Discovery or Full Run), the Filter expression to match the data against, the time range, etc.

When a Node receives this configuration, it prepares the infrastructure to execute a collection job. 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.

> i On the **Manage Collectors** page, click **Job Inspector** to see the results of recent collection runs. Select the **Show system jobs** check box to also display discovery jobs and collection jobs for the Office 365 System/Activity Sources.

Scheduled Collection Jobs

You might process data from inherently non-streaming sources, such as REST endpoints, blob stores, etc. 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, Collectors are set up at the Worker Group level, and the tasks are executed by Worker Nodes. 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. You can change this default behavior by opening global ⚙ **Settings** (lower left) > **General Settings > Job Limits**, and then setting **Job Dispatching** to **Round Robin**.

Collector Types

Cribl LogStream currently provides the following Collector options:

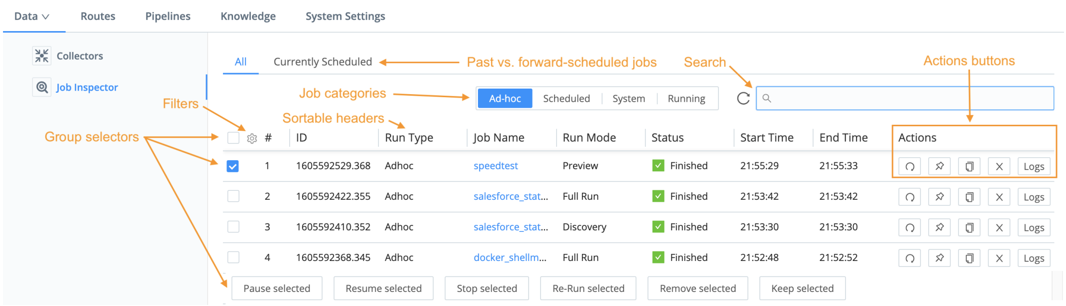
- Filesystem/NFS – enables data collection from local or remote filesystem locations.
- S3 – enables data collection from Amazon S3 buckets or S3-compatible stores.
- Script – enables data collection via custom scripts.
- REST – enables data collection via REST API calls. Provides four Discover options, to support progressively more complex (and dynamic) item enumerations.

Monitoring Collection Jobs

Select **Monitoring > Jobs** from the top nav to see a graphical display of in-flight collection jobs and their tasks.

Inspecting Jobs

Select **Collectors > Job Inspector** to view and manage pending, in-flight, and completed collection jobs.



□	#	ID	Run Type	Job Name	Run Mode	Status	Start Time	End Time	Actions
☑	1	1605592529.368	Adhoc	speedtest	Preview	✓ Finished	21:55:29	21:55:33	↻ ✕ Logs
□	2	1605592422.355	Adhoc	salesforce_stat...	Full Run	✓ Finished	21:53:42	21:53:42	↻ ✕ Logs
□	3	1605592410.352	Adhoc	salesforce_stat...	Discovery	✓ Finished	21:53:30	21:53:30	↻ ✕ Logs
□	4	1605592368.345	Adhoc	docker_shellm...	Full Run	✓ Finished	21:52:48	21:52:52	↻ ✕ Logs

Job Inspector: all the things

Here are the options available on the Job Inspector page:

- **All** vs. **Currently Scheduled** tabs: Click **Currently Scheduled** to see jobs foward-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.)
- Filters: Click the gear icon to open a drop-down with multiple options to filter the jobs shown within your selected category.
- Group selectors: Select one or more check boxes to display the **Pause**, **Resume**, etc., buttons shown along the bottom.
- 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: **Re-run**; **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).

What's Next

See the configuration instructions for the collector type you want to configure,
Then proceed to instructions for scheduling and running collection jobs.

- Filesystem/NFS
- S3
- Script
- REST
- Scheduling and Running

Filesystem/NFS

Cribl LogStream supports collecting data from a local or a remote filesystem location.

Configuring a Filesystem Collector

From the top nav of a LogStream instance or Group, select **Collectors**. On the resulting **Manage Collectors** page, click **+ Add New**. The resulting **New Collector** modal displays the following options and fields.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `DysonV11Roomba960` .

Collector type: Defines the type of Collector to configure.

> i Set this to `Filesystem` to configure the Collector as shown below.
>
> The sections described below are spread across several tabs. Click the tab links at left, or the **Next** and **Prev** buttons, to navigate among tabs. Click **Save** when you've configured your Collector.

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.

Path extractors: Extractors allow using template tokens as context for expressions that enrich discovery results. Click **+ 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()}`). Each expression accesses its corresponding `<token>` through the `value` variable, and evaluates it to populate event fields. Here is a complete example:

Token	Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`foobar: {program: value.split('.')[0]}`	`/ var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Recursive: If set to `Yes` (the default), data collection will recurse through subdirectories.

Destructive: If set to `Yes` , the Collector will delete files after collection. Defaults to `No` .

Max batch size (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.

Result Settings

The Result Settings determine how LogStream 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 `No` . Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute the field's value (can be a constant).

Result Routing

Send to Routes: If set to `Yes` (the default), events will be sent to normal routing and event processing. Toggle to `No` to select a specific Pipeline/Destination combination. The `No` setting exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

> ℹ You might disable **Send to Routes** 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 LogStream's routing table for live data – potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional.

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.

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 to `Yes` to resume ad hoc collection jobs if LogStream restarts during the jobs' execution.

What's Next

› Scheduling and Running

Azure Blob Storage

Cribl LogStream (v.2.4.5 and above) supports collecting data, and replaying specific events, from Azure Blob Storage. This page covers how to configure the Collector.

Configuring an Azure Blob Storage Collector

From the top nav of a LogStream instance or Group, select **Collectors**. On the resulting **Manage Collectors** page, click **+ Add New**. The resulting **New Collector** modal displays the following options and fields.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `azure_42-a` .

Collector type: Defines the type of Collector to configure.

> i Set this to `Azure Blob Storage` to configure the Collector as shown below.
>
> The sections described below are spread across several tabs. Click the tab links at left, or the **Next** and **Prev** buttons, to navigate among tabs. Click **Save** when you've configured your Collector.

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. E.g., referencing a Global Variable: `myBucket-${C.vars.myVar}` .

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. Click **+ 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()}`). Each expression accesses its corresponding `<token>` through the `value` variable, and evaluates it to populate event fields. Here is a complete example:

Token	Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`foobar: {program: value.split('.')[0]}`	`/ var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Recursive: If set to `Yes` (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.

Authentication

Connection string: Enter your Azure connection string. If left blank, LogStream will fall back to `env.AZURE_STORAGE_CONNECTION_STRING` .

Result Settings

The Result Settings determine how LogStream 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 `No`. Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute the field's value (can be a constant).

Result Routing

Send to Routes: If set to `Yes` (the default), events will be sent to normal routing and event processing. Toggle to `No` to select a specific Pipeline/Destination combination. The `No` setting exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

> ℹ You might disable **Send to Routes** 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 LogStream's routing table for live data – potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional.

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.

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 to `Yes` to resume ad hoc collection jobs if LogStream restarts during the jobs' execution.

Google Cloud Storage

Cribl LogStream (v.2.4.5 and above) supports collecting data objects from Google Cloud Storage buckets. This page covers how to configure the Collector.

Configuring a Google Cloud Storage Collector

From the top nav of a LogStream instance or Group, select **Collectors**. On the resulting **Manage Collectors** page, click **+ Add New**. The resulting **New Collector** modal displays the following options and fields.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `gcs_24-7` .

Collector type: Defines the type of Collector to configure.

> i Set this to `Google Cloud Storage` to configure the Collector as shown below.
>
> The sections described below are spread across several tabs. Click the tab links at left, or the **Next** and **Prev** buttons, to navigate among tabs. Click **Save** when you've configured your Collector.

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}` .

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. Click **+ 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()}`). Each expression accesses its corresponding `<token>` through the `value` variable, and evaluates it to populate event fields. Here is a complete example:

Token	Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`foobar: {program: value.split('.')[0]}`	`/ var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Recursive: If set to `Yes` (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.

Authentication

Service account credentials: Contents of Google Cloud service account credentials (JSON keys) file. To upload a file, click the upload button at this field's upper right.

> 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.

Additional Collector Settings

Endpoint: Google Cloud Storage service endpoint. If empty, the endpoint will be automatically constructed using the service account credentials.

Result Settings

The Result Settings determine how LogStream 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 `No`. Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute the field's value (can be a constant).

Result Routing

Send to Routes: If set to `Yes` (the default), events will be sent to normal routing and event processing. Toggle to `No` to select a specific Pipeline/Destination combination. The `No` setting exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

> ℹ You might disable **Send to Routes** 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 LogStream's routing table for live data – potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional.

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.

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 to `Yes` to resume ad hoc collection jobs if LogStream restarts during the jobs' execution.

S3

Cribl LogStream supports collecting data from Amazon S3 stores. This page covers how to configure the Collector.

> For a step-by-step tutorial on using LogStream 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.

Configuring an S3 Collector

From the top nav of a LogStream instance or Group, select **Collectors**. On the resulting **Manage Collectors** page, click **+ Add New**. The resulting **New Collector** modal displays the following options and fields.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `Attic42TreasureChest`.

Collector type: Defines the type of Collector to configure.

> Set this to `S3` to configure the Collector as shown below.
>
> The sections described below are spread across several tabs. Click the tab links at left, or the **Next** and **Prev** buttons, to navigate among tabs. Click **Save** when you've configured your Collector.

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.

Region: S3 Region from which to retrieve data.

Path: Path, within the bucket, from which to collect data. Templating is supported (e.g., `/myDir/${host}/${year}/${month}/`). More on templates and Filters.

Path extractors: Extractors allow using template tokens as context for expressions that enrich discovery results. Click **+ 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()}`). Each expression accesses its corresponding `<token>` through the `value` variable, and evaluates it to populate event fields. Here is a complete example:

Token	Expression	Matched Value	Extracted Result
`/var/log/${foobar}`	`foobar: {program: value.split('.')[0]}`	`/ var/log/syslog.1`	`{program: syslog, foobar: syslog.1}`

Recursive: If set to `Yes` (the default), data collection will recurse through subdirectories.

Max batch size (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.

Authentication

Select an AWS authentication method.

The **Manual** option (default) provides these 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. Optional when running on AWS.

The **Secret Key pair** option swaps in this drop-down:

- **Secret key pair**: Select a secret key pair that you've configured in LogStream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

Enable Assume Role: Slide to `Yes` 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.

Additional S3 Settings

Endpoint: S3 service endpoint. If empty, LogStream will automatically construct the endpoint from the region.

Signature version: Signature version to use for signing S3 requests. Defaults to `v4` .

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes` .

Result Settings

The Result Settings determine how LogStream 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 `No` . Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute the field's value (can be a constant).

Result Routing

Send to Routes: If set to `Yes` (the default), events will be sent to normal routing and event processing. Toggle to `No` to select a specific Pipeline/Destination combination. The `No` setting exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

> i You might disable **Send to Routes** 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 LogStream's routing table for live data – potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional.

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.

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 to `Yes` to resume ad hoc collection jobs if LogStream restarts during the jobs' execution.

What's Next

Scheduling and Running

Script

Cribl LogStream supports flexible data collection configured by your custom scripts.

Configuring a Script Collector

From the top nav of a LogStream instance or Group, select **Collectors**. On the resulting **Manage Collectors** page, click **+ Add New**. The resulting **New Collector** modal displays the following options and fields.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `sh2GetStuff` .

Collector type: Defines the type of Collector to configure.

> i Set this to `Script` to configure the Collector as shown below.
>
> The sections described below are spread across several tabs. Click the tab links at left, or the **Next** and **Prev** buttons, to navigate among tabs. Click **Save** when you've configured your Collector.

Discover script: Script to discover which objects/files to collect. This script should output one task per line in `stdout` .

Collect script: Script to perform data collections. Pass in tasks from the Discover script as `$CRIBL_COLLECT_ARG` . Should output results to `stdout` .

Shell: Shell in which to execute scripts. Defaults to `/bin/bash` .

> ⚠ With Great Power Comes Great Responsibility!
>
> Scripts will allow you to execute almost anything on the system where Cribl LogStream is running. Make sure you understand the impact of what you're executing before you do so! These scripts run as the user running LogStream, so if you are running it as root, these commands will run with root user permissions. ☠ ☠

Result Settings

The Result Settings determine how LogStream 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 `No` . Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000` .

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute the field's value (can be a constant).

Result Routing

Send to Routes: If set to `Yes` (the default), events will be sent to normal routing and event processing. Toggle to `No` to select a specific Pipeline/Destination combination. The `No` setting exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

> ℹ You might disable **Send to Routes** 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 LogStream's routing table for live data – potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional.

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.

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 to `Yes` to resume ad hoc collection jobs if LogStream restarts during the jobs' execution.

Examples

You could define this Collector to check for LogStream 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`

What's Next

> Scheduling and Running

REST / API Endpoint

Cribl LogStream supports collecting data from REST endpoints. This Collector provides multiple Discover types and Collect options. For usage examples, see Using REST / API Collectors.

Configuring a REST Collector

From the top nav of a LogStream instance or Group, select **Collectors**. On the resulting **Manage Collectors** page, click **+ Add New**. The resulting **New Collector** modal displays the following options and fields.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID

Unique ID for this Collector. E.g., `rest42json` .

Collector Type

Defines the type of Collector to configure.

> i Set this to `REST` to configure the Collector as shown below.
>
> The sections described below are spread across several tabs. Click the tab links at left, or the **Next** and **Prev** buttons, to navigate among tabs. Click **Save** when you've configured your Collector.

Discover Type

Once you've selected the `REST` **Collector type** above, this exposes a **Discover type** drop-down. Here you have four options, corresponding to different use cases. Each **Discover type** selection will expose a different set of **Collector Settings** fields. Below, we cover the **Discover type**s from simplest to most-complex.

- **Discover type: None** matches cases where one simple API call will retrieve all the data you need. This suppresses the Discover stage. (Example: Collect a list of configured LogStream Pipelines.)
- **Discover type: Item List** matches cases where you want to enumerate a known list of items to retrieve. (Example: Collect network traffic data that's tagged with specific subnets.)
- **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 Collect task. (Example: Collect data for specific geo locations the National Weather Service API's stream of worldwide weather data. This API requires multiple parameters in the request URL – latitude, longitude, etc. – so an Item List would not work.)
- **Discover type: HTTP Request** matches cases where you need to dynamically discover what you can collect from a REST endpoint. This **Discover type** most fully exploits LogStream's Discover-Before-Collect architecture. (Example: Make a REST call to get a list of available log files, then run Collect against each of those files.)

Common Collector Settings / Discover Type: None

These remaining **Collector Settings** options appear for **Discover type**: `None` , as well as for all other **Discover type** selections:

Collect URL: URL (constant or JavaScript expression) to use for the Collect operation.

> i 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: space, `$` , `/` , `=` .) Example URL with encoding: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)` .

> Request parameters are not contained directly in the URL are automatically encoded. As of LogStream v.2.3.2, URLs/expressions specified in the **Collect URL** field follow redirects.

Collect method: Select the HTTP verb to use for the Collect operation – `GET` , `POST` , or `POST with body` .

Collect POST body: Template for POST body to send with the Collect request. (This field is displayed only when you set the **Collect method** to `POST with body` .) You can reference parameters from the Discover response using template params of the form: `${variable}` .

Collect 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:

- **Name**: Field name.
- **Value**: JavaScript expression to compute the field's value (can be a constant).

Collect headers:: Click **+ Add Header** to (optionally) add collection request haaders as key-value pairs:

- **Name**: Header name.
- **Value**: JavaScript expression to compute the header's value (can be a constant).

> ℹ By adding the appropriate **Collect headers**, you can specify API Key–based authentication as an alternative to the Authentication: `Basic` or `Login` options below.

Pagination: The pagination scheme for collection results. Defaults to. `None` . 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. Selecting either of these selections exposes two additional fields:

- **Response Attribute**: Name of the attribute in the response that contains next-page information.
- **Max Pages**: The maximum number of pages to retrieve. Set to `0` to retrieve all pages.

> ℹ Time Range Variables
>
> 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 URL**, **Collect parameters**, **Collect headers**
> - **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}`
>
> Both variables are formatted as UNIX epoch time, in seconds units. When using them in contexts that require milliseconds resolution, multiply them by 1,000 to convert to ms.

Authentication

In the **Authentication** drop-down, select an authentication method to use for discover and collect REST requests:

- `None` : Compatible with REST servers like AWS, where you embed a secret directly in the request URL.

- `Basic` : Compatible with Basic Authentication servers. Selecting `Basic` exposes additional fields in which you specify a Basic Auth **Username** and **Password**.

- `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 LogStream uses for later Collect operations.

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 Collect calls. Uses `${token}` to reference the token obtained from the login POST request.

Additional Collector Settings

In the **Request Timeout (secs)** field, you can set a maximum time period (in seconds) for an HTTP request to complete before LogStream treats it as timed out. Defaults to `0`, which disables timeout metering.

Discover Type: Item List

Setting the **Discover type** to `Item List` exposes this additional field above the Common Collector Settings:

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 JSON, name of the field or array element to pull results from. Leave blank if the result is an array of values. Sample entry: `items, json: { items: [{id: 'first'},{id: 'second'}] }`

Discover Type: HTTP Request

Setting the **Discover type** to `HTTP Request` exposes these additional fields above the Common Collector 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.

> ℹ 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: space, `$`, `/`, `=`.) Example URL with encoding: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)`.
>
> Request parameters are not contained directly in the URL are automatically encoded. As of LogStream v.2.3.2, 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 (can also be a constant).

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 (can also be a constant).

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 Collector Settings' remaining tabs, whose settings and content apply equally to all **Discover type** selections.

Result Settings

The Result Settings determine how LogStream 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 `No`. Toggle to `Yes` 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: The amount of time (in milliseconds) that 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. Defaults to `10000`.

Fields (Metadata)

In this section, you can add fields/metadata to each event, using Eval-like functionality.

Name: Field name.

Value: JavaScript expression to compute the field's value (can be a constant).

Result Routing

Send to Routes: If set to `Yes` (the default), events will be sent to normal routing and event processing. Toggle to `No` to select a specific

Pipeline/Destination combination. The `No` setting exposes these two additional fields:

- **Pipeline**: Select a Pipeline to process results.
- **Destination**: Select a Destination to receive results.

> ℹ You might disable **Send to Routes** 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 LogStream's routing table for live data – potentially simplifying the Routes structure.

Pre-processing Pipeline: Pipeline to process results before sending to Routes. Optional.

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.

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 to `Yes` to resume ad hoc collection jobs if LogStream restarts during the jobs' execution.

Response Errors

The Collector treats all non-200 responses from configured URL endpoints as errors. This includes 1xx, 3xx, 4xx, and 5xx responses.

On Discover, Preview, and most Collect jobs, it interprets these as fatal errors. On Collect jobs, a few exceptions are treated as non-fatal:

- Where a Collect job launches multiple tasks, and only a subset of those tasks fail, LogStream places the job in failed status, but treats the error as non-fatal. (Note that LogStream 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.

What's Next

> Scheduling and Running

Scheduling and Running

Once you've configured a Collector, you can either run it immediately ("ad hoc") to collect data, or schedule it to run on a recurring interval. Scheduling requires some extra configuration upfront, so we cover this option first.

> ℹ For ad hoc collection, you can configure whether a job interrupted by a LogStream shutdown will automatically resume upon LogStream restart.
>
> But regardless of this configuration, explicitly restarting or stopping LogStream will cancel any currently running jobs. This applies to the `./cribl restart` and `./cribl stop` CLI commands, as well as to selecting the UI's global ⚙ **Settings** (lower left) > **Controls > Restart** option.
>
> A **scheduled** job will **not** resume upon restart.

Schedule Configuration

Click **Schedule** beside a configured Collector to display the **Schedule configuration** modal. This provides the following controls.

Enabled: Slide to `Yes` to enable this collection schedule.

> ⚠ The scheduled job will keep running on this schedule forever, unless you toggle **Enabled** back to `Off`. The `Off` setting 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.

Skippable: Skippable jobs can be delayed up to their next run time if the system is hitting concurrency limits. Defaults to `Yes`.

Skippable Jobs and Concurrency Limits

If toggled to `Yes`, the **Skippable** option obeys these concurrency limits in global ⚙ **Settings** (lower left) > **General Settings > Job Limits**:

- **Concurrent Job Limit**

- **Concurrent Scheduled Job Limit**

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, LogStream will run the delayed job, then snap back to the original cron schedule.
- If resources do **not** free up before the next scheduled run: LogStream will **skip** the delayed run, and snap back to the original cron schedule.

Set **Skippable** to `No` if you absolutely must have all your data, for compliance or other reasons. In this case, LogStream will build up a backlog of jobs to run.

You can think of **Skippable**: `No` as behaving more like the TCP protocol, with **Skippable**: `Yes` behaving more like UDP.

Max Concurrent Runs: Sets the maximum number of instances of this scheduled job that may simultaneously run.

> ⚠ **All** collection jobs are constrained by the following options in global ⚙ **Settings** (lower left) > **General Settings** > **Job Limits**:
>
> - **Concurrent Task Limit**
> - **Max Task Usage Percentage**

Run Configuration and Shared Settings

Most of the remaining fields and options below are shared with the **Run configuration** modal, which you can open by clicking **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 (e.g., 100 events). This is very 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.

> ℹ **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.

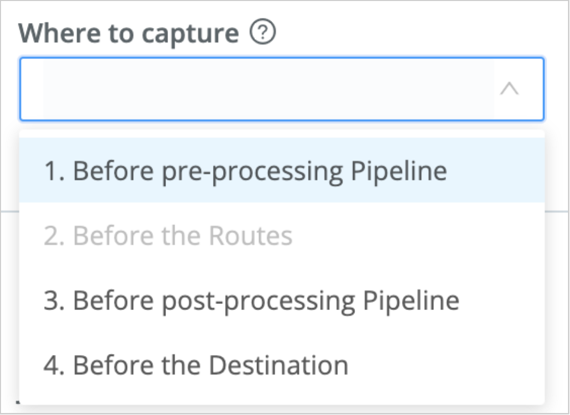
Preview Settings

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 one of the options shown below. (Note that option `2. Before the Routes` is disabled.) If not specified, this will default to `1. Before pre-processing Pipeline`.



Where to capture
1. Before pre-processing Pipeline
2. Before the Routes
3. Before post-processing Pipeline
4. Before the Destination

Preview capture options

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 slider enables you to send discovery results to LogStream Routes. Defaults to `No`.

> ℹ 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.

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.

Absolute

Select the **Absolute** button to set fixed collection boundaries in your local time. Next, use the **Earliest** and **Latest** controls to set the start date/time and end date/time.

Relative

Select the **Relative** button to set collection boundaries relative to the current time. Next, use the **Earliest** and **Latest** to set start and end times like these:

- **Earliest** example values: `-1hr` , `-42m` ,`- 42m@h`
- **Latest** example values: `now` , `-20m` , `+42m@h`

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` .
<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.)

LogStream validates relative time values using these rules:

- **Earliest** must not be later than **Latest**.
- Values without units get interpreted as seconds. (E.g., `-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).
- `@m` 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 Filesystem 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. E.g., 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**. E.g., 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 `Yes`; does not apply to fatal errors.

Max task reschedule: 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 `0`, meaning unlimited time (no timeout).

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, LogStream 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, LogStream 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

LogStream 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. E.g., `/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:

- `'Yy'`, for years
- `'mBbj'`, for months
- `'dj'`, for days
- `'HI'`, for hours
- `'M'`, for minutes
- `'S'`, for seconds

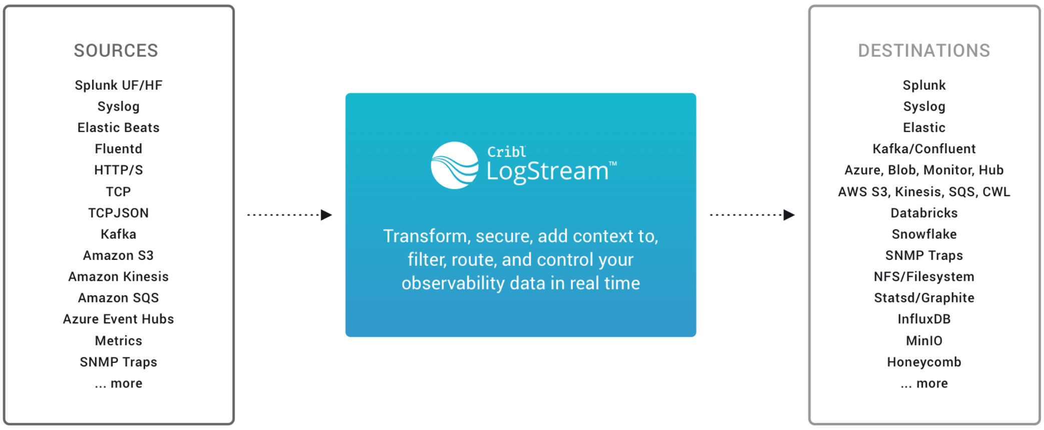
Token Syntax

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:date=%d}/...`	`/my/path/year=2020/month=05/date=20/...`	
`/my/path/${_time:%Y-%m-%d}/...`	`/my/path/2020-05-20/...`	

Destinations

Cribl LogStream can send data to various Destinations, including Splunk, Kafka, Kinesis, InfluxDB, Snowflake, Databricks, TCP JSON, and many others.



Streaming Destinations

Destinations that accept events in real time are referred to as streaming Destinations:

- Splunk Single Instance
- Splunk Load Balanced
- Splunk HEC
- AWS Kinesis Streams
- AWS CloudWatch Logs
- AWS SQS
- Elasticsearch
- Honeycomb
- TCP JSON
- Syslog
- Kafka
- Azure Event Hubs

- Azure Monitor Logs
- Google Cloud Pub/Sub
- StatsD
- StatsD Extended
- Graphite
- SNMP Trap
- InfluxDB
- New Relic
- Prometheus
- Wavefront
- Sumo Logic
- Datadog
- SignalFx
- Webhook

Non-Streaming Destinations

Destinations that accept events in groups or batches are referred to as non-streaming Destinations:

- S3 Compatible Stores
- Filesystem/NFS
- MinIO
- Azure Blob Storage
- Google Cloud Storage

> ℹ The S3 Compatible Stores Destination can be adapted to send data to downstream services like Databricks and Snowflake, for which LogStream currently has no preconfigured Destination. For details, please contact Cribl Support.

Other Destinations

LogStream also provides these special-purpose Destinations:

- Output Router: Flexible "meta-destination." Here, you can configure rules that route data to multiple outputs.

- DevNull: An output that simply drops events. Preconfigured and active when you install LogStream, so it requires no configuration. Useful for testing.
- Default: Here, you can specify a default output from among your configured Destinations.

How Does Non-Streaming Delivery Work

Cribl LogStream 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 – typically file size and number of files, further details below – data is compressed and then moved to the final Destination.

An inventory of open, or in-progress, files is kept in the staging directory's root, to avoid having to walk that directory at startup. This can get expensive if staging is also the final directory. At startup, Cribl LogStream 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).

Batching Conditions

Several **conditions** govern when files are closed and rolled out:

1. File reaches its configured maximum size.

2. File reaches its configured maximum open time.

3. File reaches its configured maximum idle time.

If a new file needs to be open, Cribl LogStream will enforce the maximum number of open files, by closing files in the order in which they were opened.

Data Delivery

Data is delivered to all Destinations on an at-least-once basis. When a Destination is unreachable, there are three possible behaviors:

- **Block** - Cribl LogStream will block incoming events.
- **Drop** - Cribl LogStream will drop events addressed to that Destination.
- **Queue** - Cribl LogStream will Persistent-Queue events to that Destination.

You can configure the desired behavior through a Destination's **Backpressure Behavior** option. If this option is not present, Cribl LogStream's default behavior is to **Block**.

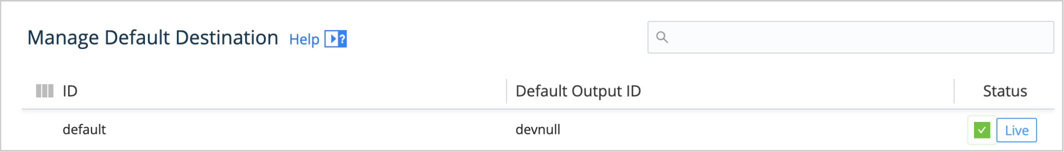
Configuring Destinations

For each Destination **type**, you can create multiple definitions, depending on your requirements.

To configure Destinations, select **Destinations** from LogStream's global top nav (single-instance deployments), or from a Worker Group's top nav (distributed deployments). On the resulting **Data Destinations** page's tiles or left menu, select the desired type, then click **+ Add New**.

Capturing Outgoing Data

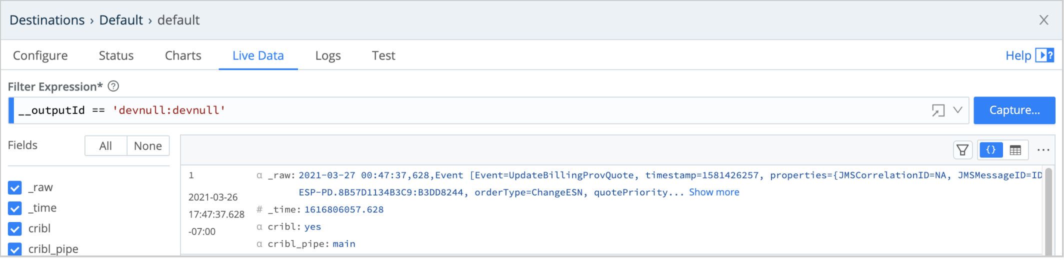
To capture data from a single enabled Destination, you can do so directly from the Destinations UI instead of using the Preview pane. To initiate an immediate capture, click the **Live** button on the Destination's's configuration row.



ID	Default Output ID	Status
default	devnull	✓ Live

Destination > Live button

You can also start an immediate capture from within an enabled Destination's configuration modal, by clicking the modal's **Live Data** tab.



Filter Expression*

```
__outputId == 'devnull:devnull'
```

☑ _raw
☑ _time
☑ cribl
☑ cribl_pipe

```
1               α _raw: 2021-03-27 00:47:37,628,Event [Event=UpdateBillingProvQuote, timestamp=1581426257, properties={JMSCorrelationID=NA, JMSMessageID=ID
2021-03-26                ESP-PD.8B57D1134B3C9:B3DD8244, orderType=ChangeESN, quotePriority... Show more
17:47:37.628    # _time: 1616806057.628
-07:00          α cribl: yes
                α cribl_pipe: main
```

Destination modal > Live Data tab

Output Router

Output Routers are meta-destinations that allow for output selection based on rules. Rules are evaluated in order, top->down, with the first match being the winner.

Configuring Cribl LogStream to Send to an Output Router

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Output Router** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Output Router > New Destination** modal, which provides the following fields.

Router name: Enter a unique name to identify this Router definition.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Rules: A list of event routing rules. Each provides the following settings:

- **Filter expression**: JavaScript expression to select events to send to output.
- **Output**: Output to send matching events to.
- **Description**: Optionally, enter a description of this rule's purpose.
- **Final**: Flag that controls whether to stop the event from being checked against other rules lower in the stack. Defaults to `Yes` .

Limitations/Options

- An Output Router cannot reference another. This is by design, so as to avoid circular references.
- Also to avoid circular references, an Output Router cannot reference a Default Destination that points back to Output Router.
- **Events that do not match any of the rules are dropped**. Use a catchall rule to change this behavior.
- No post-processing (conditioning) can be done here. Instead, use pre-processing Pipelines on the Source tier.
- Data can be cloned by toggling the `Final` flag to `No`. (The default is `Yes`, i.e., no cloning.)

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).
- 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

Splunk Single Instance

Splunk Enterprise is a streaming Destination type.

Configuring Cribl LogStream to Output to Splunk Destinations

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Splunk > Single Instance** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Single Instance > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Splunk Single Instance definition.

Address: Hostname of the Splunk receiver.

Port: The port number on the host.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

TLS Settings (Client Side)

Enabled defaults to `No`. When toggled to `Yes`:

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

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.

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.

> ℹ 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 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Output multi metrics: Toggle to `Yes` 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: Directs LogStream to check whether the indexer is shutting down, and if so, to stop sending data. This helps minimize data loss during shutdown. Toggle to `No` to disable this feature.

Throttling: Throttle rate in bytes per second. Multiple byte units such as KB, MB, GB etc. are also allowed. E.g., 42 MB. Default value of 0 indicates no throttling. When throttle engaged, excesses data will be dropped only if Backpressure Behavior is set to drop, and blocked for all other settings.

Nested field serialization: Specifies how to serialize nested fields into index-time fields. Defaults to `None` .

Auth token: Optionally, enter a shared secret to use when establishing a connection to a Splunk indexer configured with the same secret.

Notes about Forwarding to Splunk

- Data sent to Splunk is not compressed.
- If events have a Cribl LogStream 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 LogStream.

Splunk Load Balanced

Splunk is a streaming Destination type, and with the **Splunk Load Balanced** output, you can load-balance data out to multiple Splunk receivers.

How Does Load Balancing Work

Cribl LogStream will attempt to load-balance outbound data as fairly as possibly across all receivers (listed as Destinations in the GUI). If FQDNs/hostnames are used as the Destination address and each resolves to, for example, 5 (unique) IPs, then each worker process will have its # of outbound connections = # of IPs x # of FQDNs for purposes of the SplunkLB output. Data is sent by all worker processes to all receivers simultaneously, and the amount sent to each receiver depends on these parameters:

1. Respective destination **weight**.
2. Respective destination **historical data**.

By default, historical data is tracked for 300s. LogStream 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.

Example

Suppose we have two receivers, 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 be dispensed
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 be dispensed
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. I.e., receiver A will start the interval with **60** and receiver B with **40**. To determine how many events A and B will receive during this next interval, LogStream will use their weights and their stats as follows:

Total number of events: `events to be dispensed + stats carried forward = 200 + 60 + 40 = 300`.
Number of events per each destination (weighed): `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 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.

Configuring Cribl LogStream to Load-Balance to Multiple Splunk Destinations

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Splunk > Load Balanced** from the **Data Destinations** page's tiles or the **Destinations** left nav. Then click **+ Add New** to open the **Load Balanced > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Splunk LB Destination definition.

Indexer Discovery: When toggled to `Yes` , enables automatic discovery of indexers in an indexer clustering environment. See Indexer Discovery for the resulting UI options displayed below. When set to `No` (the default), displays the Destinations section below.

Exclude current host IPs: Exclude all IPs of the current host from the list of any resolved hostnames. Defaults to `Yes` .

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` . When toggled to `Persistent Queue` , adds the Persistent Queue Settings section (left tab) to the modal.

Destinations

The **Destinations** section appears only when **Indexer discovery** has its default `No` setting. Here, you specify a known set of Splunk 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 the Splunk receiver. Optionally, you can paste in a comma-separated list, in `<host>:<port>` format.
- **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 connection host (if not an IP). Otherwise, uses the global TLS settings.
- **Load weight**: The weight to apply to this Destination for load-balancing purposes.

Indexer Discovery

Toggling the **Indexer Discovery** toggle to `Yes` displays the following fields instead of the Destinations section:

Site: Clustering site from which 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.)

Auth token: Authentication token required to authenticate to the cluster manager for indexer discovery.

Refresh period: Time interval (in seconds) between two consecutive fetches of indexer list from cluster manager. Defaults to `60`.

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

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>` . Defaults to `$CRIBL_HOME/state/queues` .

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None` ; `Gzip` is also available.

TLS Settings (Client Side)

Enabled defaults to `No` . When toggled to `Yes` :

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No` .

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

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.

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.

> ℹ 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 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Output Multi Metrics: Toggle this slider to `Yes` 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 set to `Yes` (the default), LogStream will check whether the indexer is shutting down and, if so, stop sending data. This helps minimize data loss during shutdown.

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 LogStream load balancing, IP settings take priority over those from hostnames.)

Max connections: 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), LogStream will randomly select this subset of discovered IPs to connect to. LogStream 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`.

Auth token: Optionally, enter a shared secret to use when establishing a connection to a Splunk indexer configured with the same secret.

Throttling: Throttle rate, in bytes per second. Multiple byte units such as KB, MB, GB, etc., are also allowed. E.g., `42 MB`. Default value of `0` 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.)

SSL Configuration for Splunk Cloud – Special Note

To connect to Splunk Cloud, you **might** need to extract the private and public key from the Splunk-provided Splunk Cloud Certificate, which is typically bundled in an app. Use the following steps:

Step 1. Test connectivity to Splunk Cloud, using the Root CA certificate:

```
openssl s_client -CApath path_to_ca.pem -connect
hostnameToSplunkCloud:9997
```

Step 2. Extract the Private key from the Splunk Cloud Certificate. At the prompt, you will need the `sslPassword` value from the `outputs.conf` file bundled with the Splunk Cloud app. Using Elliptic Curve keys:

```
openssl ec -in path_to_server_cert.pem -out private.pem
```

If you are using RSA keys, instead use:

```
openssl rsa -in path_to_server_cert.pem -out private.pem
```

Step 3. Extract the Public Key for the Server Certificate:

```
openssl x509 -in path_to_server_cert.pem -out server.pem
```

Step 4. In the LogStream Destination's **TLS Settings (Client Side)** section, enter the following:

- **CA Certificate Path**: Path to CA Certificate.
- **Private Key Path (mutual auth)**: Path to `private.pem` (Step 2 above).
- **Certificate Path (mutual auth)**: Path to `server.pem` (Step 3 above).

Notes About Forwarding to Splunk

- Data sent to Splunk is not compressed.
- If events have a LogStream 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 LogStream.

Splunk HEC

Splunk HEC is a streaming Destination type. In a typical deployment, Cribl LogStream will be installed/co-located in a Splunk heavy forwarder. If this output is enabled, it can send data out to a Splunk HEC (HTTP Event Collector) destination through the event endpoint.

Configuring Cribl LogStream to Output to Splunk HEC Destinations

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Splunk > HEC** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **HEC > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Splunk HEC definition.

Splunk HEC endpoint: URL of a Splunk HEC endpoint to send events to (e.g., `http://myhost.example.com:8088/services/collector/event`).

HEC auth token: Splunk HEC authentication token.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Output multi metrics: Toggle to `Yes` 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.)

Compress: Toggle this slider to `Yes` 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`. Each request can potentially hit a different HEC receiver.

Max body size (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.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Max body size**. Defaults to `1`.

> i
> - 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.

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`.

> i This is useful only when you expect the HEC receiver to route this data on to another destination.

Output multi metrics: Toggle to `Yes` 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.)

Notes on HTTP-based Outputs

- Cribl LogStream will attempt to use keepalives to reuse a connection for multiple requests. After 2 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 next request.

- When resolving the Destination's hostname, LogStream will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

- See Splunk's documentation on editing `fields.conf` to ensure the visibility of index-time fields sent to Splunk by LogStream.

Amazon S3 Compatible Stores

S3 is a non-streaming Destination type. Cribl LogStream does **not** have to run on AWS in order to deliver data to S3.

Stores that are S3-compatible will also work with this Destination type. For example, the S3 Destination can be adapted to send data to services like Databricks and Snowflake, for which LogStream currently has no preconfigured Destination. For these integrations, please contact Cribl Support.

Configuring Cribl LogStream to Output to S3 Destinations

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Amazon** > **S3** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **S3** > **New Destination** modal, which provides the following fields.

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. E.g., referencing a Global Variable: `myBucket-${C.vars.myVar}` .

> i 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.

Region: Region where the S3 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.

Add Output ID: When set to `Yes` (the 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 LogStream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this slider's state. This is to avoid losing any files pending in the original staging directory, upon LogStream 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.

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.

Partitioning expression: JavaScript expression to define how files are partitioned and organized. If left blank, Cribl LogStream will fall back to `event.__partition`. Defaults to ``` `${host}/${sourcetype}` ```. Partitioning by time is also possible, e.g., ``` `${host}/${C.Time.strftime(_time, '%Y-%m-%d')}/${sourcetype}` ```

Data format: Format of the output data. Defaults to `JSON`.

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`.

Compress: Select the data compression format to use before moving data to final destination. Defaults to `none`. Cribl recommends setting this to `gzip`.

Backpressure behavior: Select whether to block or drop events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Authentication

Use the **Authentication Method** buttons to 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: If not running on AWS, you can select this option to enter your access key and secret key, directly or by reference. 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.

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 LogStream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

Enable for S3: Whether to use Assume Role credentials to access S3. Defaults to `No`.

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.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports `c*` wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Endpoint: S3 service endpoint. If empty, the endpoint will be automatically constructed from the region.

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: Server side encryption type for uploaded objects. Defaults to `none`. This is used for enabling encryption for data at rest.

Signature version: Signature version to use for signing S3 requests. Defaults to `v4`.

Max file size (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

Max file open time (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`.

Max file idle time (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` .

Max open files: 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` .

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes` .

> i Cribl LogStream will close files when **either** of the `Max file size (MB)` or the `Max file open time (sec)` conditions are met.

Amazon S3 Permissions

The following permissions are needed to write to an Amazon S3 bucket:

```
s3:GetObject
s3:ListBucket
s3:GetBucketLocation
s3:PutObject
```

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Amazon Kinesis Streams

Cribl LogStream can output events to **Amazon Kinesis Data Streams** records of up to 1MB uncompressed. Cribl LogStream does **not** have to run on AWS in order to deliver data to a Kinesis Data Stream.

Configuring Cribl LogStream to Output to Amazon Kinesis Data Streams

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Amazon** > **Kinesis** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Kinesis** > **New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Kinesis definition.

Stream name: Enter the name of the Kinesis Data Stream to which to send events.

Region: Select the AWS Region where the Kinesis Data Stream is located.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Authentication

Use the **Authentication Method** buttons to 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: If not running on AWS, you can select this option to enter your access key and secret key, directly or by reference. 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.

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 LogStream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Assume Role

Enable for Kinesis Streams: Whether to use Assume Role credentials to access Kinesis Streams. Defaults to `No`.

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

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.

- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

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`.

Put request concurrency: Maximum number of ongoing put requests before blocking. Defaults to `5`.

Max record size (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. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes`.

Format

Currently, outputted events use the following record format:

- Header line containing information about the payload (currently supports one type, as shown below).
- Newline-Delimited JSON (that is, each Kinesis record will contain multiple events, in **ndjson** format).

Record payloads (including header and body) will be gzip-compressed, and then Kinesis will base64-encode them.

Sample Kinesis Record

```
{"format":"ndjson","count":8,"size":3960}
{"_raw":"07-03-2018 18:33:51.136 -0700 ERROR TcpOutputFd - Read error. Connection reset by peer'
{"_raw":"07-03-2018 18:33:51.136 -0700 INFO  TcpOutputProc - Connection to 127.0.0.1:10000 clos
...
```

Amazon CloudWatch Logs

Cribl LogStream supports sending data to Amazon CloudWatch Logs. This is a streaming Destination type. Cribl LogStream does **not** have to run on AWS in order to deliver data to CloudWatch Logs.

Configuring Cribl LogStream to Output to Amazon CloudWatch Logs

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Amazon > CloudWatch Logs** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **CloudWatch Logs > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this CloudWatch definition.

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 LogStream instance. (E.g., `myStream_myHost_myOutputId` .)

Region: AWS region where the CloudWatch Logs group is located.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

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

Enable for CloudWatch Logs: Whether to use Assume Role credentials to access CloudWatch Logs. Defaults to `No`.

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

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination 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.

Signature version: Signature version to use for signing CloudWatch Logs requests. Defaults to `v4` .

Max queue size: Maximum number of queued batches before blocking. Defaults to `5` .

Max record size (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. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes` .

Amazon SQS

Cribl LogStream supports sending events to Amazon Simple Queuing Service.

Configuring Cribl LogStream to Send Data to Amazon SQS

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Amazon** > **SQS** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **SQS** > **New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this SQS Destination.

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>'`. (E.g., `':port/<myQueueName>'`.)

*Queue type**: The queue type used (or created). Defaults to `Standard`. `FIFO` (First In, First Out) is the other option.

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 `Yes`.

Region: Region where SQS queue is located.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

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

Enable for SQS: Whether to use Assume Role credentials to access SQS. Defaults to `No`.

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 LogStream 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.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination 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`.

Max queue size: Maximum number of queued batches before blocking. Defaults to `100`.

Max record size (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`.

Max concurrent requests: The maximum number of in-progress API requests before backpressure is applied. Defaults to `10`.

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes`.

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 LogStream 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`

Azure Blob Storage

Azure Blob Storage is a non-streaming Destination type. Cribl LogStream does **not** have to run on Azure in order to deliver data to it. Azure Data Lake Storage Gen2 (hierarchical namespace) is also supported.

Configuring LogStream to Output to Azure Blob Storage

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Azure > Blob Storage** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Blob Storage > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Destination definition.

Connection string: Enter your Azure Connection String. The format is: `DefaultEndpointsProtocol=[http|https];AccountName=<your-account-name>;AccountKey=<your-account-key>`

A fictitious example, using Microsoft's recommended HTTPS option: `DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=12345678...32`

If left blank, LogStream will fall back to: `env.AZURE_STORAGE_CONNECTION_STRING` .

> The **Connection string** field replaces the **Account name** and **Account key** fields provided in this Destination's UI prior to LogStream 2.4. If you configured an Azure Blob Storage Destination before upgrading to LogStream 2.4, those fields' values are now concatenated into the **Connection string** value.
>
> In case values were concatenated incorrectly, the original field keys and values are retained in LogStream's configuration files (e.g., `$CRIBL_HOME/groups/<group-name>/cribl/outputs.yml`). However, Cribl recommends simply re-entering the correct Azure **Connection string** here.
>
> For further details, see Microsoft's Configure Azure Storage Connection Strings documentation.

Container name: Enter the container name. (A container organizes a set of blobs, similar to a directory in a file system.)

Create container: Defaults to `No`. Toggle to `Yes` to create the configured container in Azure Blob Storage if it does not already exist.

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.

Add Output ID: When set to `Yes` (the 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 LogStream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this slider's state. This is to avoid losing any files pending in the original staging directory, upon LogStream 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.

Partitioning expression: JavaScript expression to define how files are partitioned and organized. Defaults to `` `${host}/${sourcetype}` `` If left blank, Cribl LogStream will fall back to `event.__partition`.

Data format: Format of the output data. Defaults to `json`.

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`.

Compress: Data compression format used before moving to final destination. Defaults to `none`. Cribl recommends setting to `gzip`.

Backpressure behavior: Whether to block or drop events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Max file size (MB): Maximum uncompressed output file size. Files reaching this size will be closed and moved to the final output location. Defaults to `32`.

Max file open time (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`.

Max file idle time (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`.

Max open files: 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`.

> i LogStream will close files when **either** of the `Max file size (MB)` or the `Max file open time (sec)` conditions are met.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Azure Monitor Logs

Cribl LogStream supports sending of data over to Azure Monitor Logs. This is a streaming Destination type.

Configuring Cribl LogStream to Output to Azure Monitor Logs

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Azure > Monitor Logs** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Monitor Logs > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Azure Monitor Logs definition.

Workspace ID: Enter the Azure Log Analytics Workspace ID. (See **Workspace->Advanced settings** in the Azure Dashboard.)

Workspace key: Enter the Azure Log Analytics Workspace Primary or Secondary Shared Key. (In the Azure Dashboard, see **Workspace->Advanced settings**.)

Log type: The Record Type of events sent to this LogAnalytics workspace. Defaults to `Cribl`.

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 in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Toggle this slider to `Yes` 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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096`.

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.

Notes on HTTP-based Outputs

- Cribl LogStream will attempt to use keepalives to reuse a connection for multiple requests. After 2 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, LogStream will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

Azure Event Hubs

Cribl LogStream supports sending data to Azure Event Hubs. This is a streaming Destination type.

Configuring Cribl LogStream to Output to Azure Event Hubs

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Azure > Event Hubs** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Event Hubs > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Azure Event Hubs definition.

Brokers: List of Event Hub Kafka brokers to connect to. (E.g., `yourdomain.servicebus.windows.net:9093`.) Find the hostname in Shared Access Policies, in the host portion of the primary or secondary connection string.

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.

Acknowledgments: Control the number of required acknowledgments. Defaults to `Leader`.

Record data format: Format to use to serialize events before writing to the Event Hub Kafka brokers. Defaults to `JSON`.

Compression: If present, change from the default `Gzip` to `None`.

> ⚠ This option is removed as of LogStream 2.4.4, due to incompatibility on the Event Hubs side. In LogStream versions through 2.4.3, you must manually change the setting to `None` in order to enable a stable connection with Event Hubs.

Backpressure behavior: Whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

TLS Settings (Client Side)

Enabled Defaults to `Yes`.

Validate server certs: Defaults to `No` – and for Event Hubs, this must always be disabled.

Authentication

Authentication parameters to use when connecting to brokers. Using TLS is highly recommended.

Enabled: Defaults to `Yes`. (Toggling to `No` hides the remaining settings in this group.)

SASL mechanism: SASL (Simple Authentication and Security Layer) authentication mechanism to use, `PLAIN` is the only mechanism currently supported for Event Hub Kafka brokers.

Username: The username for authentication. For Event Hub, this should always be `$ConnectionString`.

Password: Event Hubs primary or secondary connection string. From Microsoft's documentation, the format is:

```
Endpoint=sb://<FQDN>/;SharedAccessKeyName=<KeyName>;SharedAccessKey=<KeyValue>
```

Example entry:

```
Endpoint=sb://dummynamespace.servicebus.windows.net/;SharedAccessKeyName=dummyaccesskeyname;SharedAccessKey=5dOntTRytoC24opYThisAsit3is2B+OGY1US/fuL3ly=
```

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Max record size (KB, uncompressed): Maximum size (KB) of each record batch before compression. Setting should be < `message.max.bytes` settings in Kafka brokers. Defaults to `768`.

Max events per batch: 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`.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__topicOut`
- `__key`
- `__headers`
- `__keySchemaIdOut`
- `__valueSchemaIdOut`

Google Cloud Storage

Google Cloud Storage is a non-streaming Destination type.

Configuring Cloud Storage Permissions

For LogStream to send data to Google Cloud Storage 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.

Configuring LogStream to Output to Cloud Storage Destinations

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Google Cloud > Cloud Storage** from the **Data Destinations** page's tiles or the **Destinations** left nav.

Next, click **+ Add New** to open the **Cloud Storage > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Cloud Storage definition.

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. E.g., 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.

Add Output ID: Whether to append output's ID to staging location. Defaults to `Yes`.

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.

Partitioning expression: JavaScript expression to define how files are partitioned and organized. If left blank, Cribl LogStream will fall back to `event.__partition`. Defaults to ``` `${host}/${sourcetype}` ```. Partitioning by time is also possible, e.g., ``` `${host}/${C.Time.strftime(_time, '%Y-%m-%d')}/${sourcetype}` ```

Data format: Format of the output data. Defaults to `JSON`.

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`.

Compress: Select the data compression format to use before moving data to final destination. Defaults to `none`. Cribl recommends setting this to `gzip`.

Backpressure behavior: Select whether to block or drop events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Authentication

Use the **Authentication Method** buttons to select a Google authentication method:

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.

- **Access key**: Enter the HMAC access key.
- **Secret key**: Enter the HMAC secret.

Secret: With this option, use the **Secret key pair** drop-down to select a key pair that you've configured in LogStream's internal secrets manager or (if enabled) an external KMS.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports `c*` wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

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`.

Max file size (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

Max file open time (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`.

Max file idle time (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`.

Max open files: 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`.

> ℹ Cribl LogStream will close files when **either** of the `Max file size (MB)` or the `Max file open time (sec)` conditions are met.

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes`.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Troubleshooting

Nonspecific messages from Google Cloud of the form `Error: failed to close file` can indicate problems with the permissions listed above.

Google Cloud Pub/Sub

Cribl LogStream supports sending data to Google Cloud Pub/Sub, a managed real-time messaging service for sending and receiving messages between applications. This is a streaming Destination type.

Configuring Cribl LogStream to Output to Pub/Sub

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Google Cloud > Pub/Sub** from the **Data Destinations** page's tiles or the Destinations left nav. Click **+ Add New** to open the **Pub/Sub > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Pub/Sub output definition.

Topic name: Enter the Pub/Sub topic to send events to.

Create topic: If toggled to `Yes` , LogStream will create the topic on Pub/Sub if it does not exist.

Ordered delivery: If toggled to `Yes` , LogStream will send events in the order that they were added to the queue. (For this to work correctly, the process receiving events must have ordering 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 in this group are exerting backpressure. Defaults to `Block` .

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Authentication

Use the **Authentication Method** buttons to select a Google authentication method:

Auto: This option uses the environment variables `PUBSUB_PROJECT` and `PUBSUB_CREDENTIALS`, and requires no configuration here.

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 LogStream's internal secrets manager or (if enabled) an external KMS.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination 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.

Max queue size: Maximum number of queued batches before blocking. Defaults to `100` .

Max batch size (KB): Maximum size for each sent batch. Defaults to `256` KB.

Max concurrent requests: The maximum number of in-progress API requests before LogStream applies backpressure. Defaults to `10` .

Let's Change the Topic

The Pub/Sub Destination supports alternate topics specified at the event level in the `__topicOut` field. So (e.g.) if a Pub/Sub Destination is configured to send to main topic `topic1` , and LogStream receives an event with `__topicOut: topic2` , then LogStream 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, LogStream 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.

StatsD

Cribl LogStream supports sending data to a StatsD Destination. This is a streaming Destination type.

Configuring Cribl LogStream to Output via StatsD

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Metrics > StatsD** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **StatsD > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this StatsD definition.

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`.

> i The next two settings apply only to the TCP protocol, and are not displayed for UDP.

Throttling: 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: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> ℹ This section is displayed only for TCP, and only when the **Backpressure behavior** is set to **Persistent Queue**.

- **Max file size**: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB` .
- **Max queue size**: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.
- **Queue file path**: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>` . Defaults to `$CRIBL_HOME/state/queues` .
- **Compression**: Codec to use to compress the persisted data, once a file is closed. Defaults to `None` ; `Gzip` is also available.

Timeout Settings

> ℹ These timeout settings apply only to the TCP protocol, and are not displayed for UDP.

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 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Max record size (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`.

StatsD Extended

Cribl LogStream's StatsD Extended Destination supports sending out data in expanded StatsD format. This is a streaming Destination type.

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.

Configuring LogStream to Output via StatsD Extended

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Metrics > StatsD Extended** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **StatsD Extended > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this StatsD Extended definition.

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`.

> ℹ The next two settings apply only to the TCP protocol, and are not displayed for UDP.

Throttling: 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: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> ℹ This section is displayed only for TCP, and only when the **Backpressure behavior** is set to **Persistent Queue**.

- **Max file size**: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.
- **Max queue size**: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.
- **Queue file path**: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.
- **Compression**: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Timeout Settings

> ℹ These timeout settings apply only to the TCP protocol, and are not displayed for UDP.

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 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Max record size (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`.

Graphite

Cribl LogStream supports sending data to a Graphite backend Destination. This is a streaming Destination type.

Configuring Cribl LogStream to Output to a Graphite Backend

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Metrics > Graphite** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Graphite > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Graphite definition.

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` .

> ℹ The next two settings apply only to the TCP protocol, and are not displayed for UDP.

Throttling: 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: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Persistent Queue Settings

> ℹ This section is displayed only for TCP, and only when the **Backpressure behavior** is set to **Persistent Queue**.

- **Max file size**: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.
- **Max queue size**: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.
- **Queue file path**: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.
- **Compression**: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Timeout Settings

> ℹ These timeout settings apply only to the TCP protocol, and are not displayed for UDP.

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 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Max record size (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`.

TCP JSON

Cribl LogStream supports sending data over TCP in JSON format. **TCP JSON** is a streaming Destination type.

Configuring Cribl LogStream to Output in TCP JSON Format

From the top nav of a LogStream instance or Group, select **Destinations**, then select **TCP JSON** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **TCP JSON > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Destination definition.

Address: Hostname of the receiver.

Port: Port number to connect to on the host.

Auth token: Optional authentication token to include as part of the connection header. Defaults to empty.

Compression: Codec to use to compress the data before sending. Defaults to `None`.

Throttling: Throttle rate in bytes per second. Multiple byte units such as KB, MB, GB etc. are also allowed. E.g., 42 MB. Default value of 0 indicates no throttling. When throttle engaged, excesses data will be dropped only if Backpressure Behavior is set to drop, and blocked for all other settings.

Backpressure behavior: Specifies whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

TLS Settings (Client Side)

Enabled defaults to `No`. When toggled to `Yes`:

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 (e.g., the system's CA). Defaults to `No`.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

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.

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.

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 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Format

TCP JSON events are sent in newline-delimited JSON format, consisting of:

1. A header line. Can be empty, e.g.: `{}`. 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.

Example

See an example in our TCP JSON Source topic.

Syslog

Cribl LogStream supports sending of data over syslog via TCP. Syslog is a streaming Destination type.

> i This Syslog Destination supports RFC 3164 and RFC 5424.

Configuring Cribl LogStream to output in **Syslog** format

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Syslog** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Syslog > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Syslog definition.

Protocol: The network protocol to use for sending out syslog messages. Defaults to `TCP` ; `UDP` is also available.

Address: Address/hostname of the receiver.

Port: Port number to connect to on the host.

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. Defaults to `Syslog`.

Throttling: Throttle rate in bytes per second. Multiple byte units such as KB, MB, GB etc. are also allowed. E.g., 42 MB. Default value of 0 indicates no throttling. When throttle engaged, excesses data will be dropped only if Backpressure Behavior is set to drop, and blocked for all other settings.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

TLS Settings (Client Side)

Enabled defaults to `No`. When toggled to `Yes`:

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

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.

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.

Timeout Settings

> ℹ 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 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this destination:

- `__priority`
- `__facility`
- `__severity`
- `__procid`
- `__appname`
- `__msgid`
- `__syslogout`

Filesystem/NFS

Filesystem is a non-streaming Destination type that Cribl LogStream can use to output files to a local file system or a network-attached file system (NFS).

Configuring Cribl LogStream to Output to Filesystem Destinations

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Filesystem** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Filesystem > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Filesystem definition.

Output location: Final destination for the output files.

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.

Add Output ID: When set to `Yes` (the 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 LogStream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this slider's state. This is so that upon LogStream 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.

Partitioning expression: JavaScript expression to define how files are partitioned and organized. Defaults to `` `${host}/${sourcetype}` ``. If left blank, Cribl LogStream will fall back to `event.__partition`. Partitioning by time is also possible, e.g.: `` `${host}/${C.Time.strftime(_time, '%Y-%m-%d')}/${sourcetype}` ``

Data format: Format of the output data. Defaults to `json`.

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`.

Compress: Data compression format used before moving to final destination. Default `none`. It is recommended that `gzip` is used.

Max file size (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

Max file open time (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`.

Max file idle time (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`.

Max open files: 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`.

> i Cribl LogStream will close files when **either** of the `Max file size (MB)` or the `Max file open time (sec)` conditions are met.

Backpressure Behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

> i To export events from an intermediate stage **within a Pipeline** to a file, see the Tee Function.

Kafka

Cribl LogStream supports sending data to a Kafka topic. **Kafka** is a streaming Destination type.

Configuring Cribl LogStream to Output to Kafka

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Kafka** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Kafka > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Kafka definition.

Brokers: List of Kafka brokers to connect to. (E.g., `localhost:9092` .)

Topic: The topic on which to publish events. Can be overwritten using event's `__topic` field.

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` .

Compression: Codec to compress the data before sending to Kafka. Select `None` , `Gzip` , or `Snappy` .

Backpressure behavior: Select whether to block or drop events when all receivers in this group are exerting backpressure. Defaults to `Block` .

TLS Settings (Client Side)

Enabled defaults to `No`. When toggled to `Yes`:

Validate server certs: Reject certificates that are not authorized by a CA in the **CA certificate path**, or by another trusted CA (e.g., the system's CA). Defaults to `No`.

Server name (SNI): Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

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.

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

Authentication parameters to use when connecting to brokers. Using TLS is highly recommended.

Enabled: Defaults to `No`. When toggled to `Yes`:

- **SASL mechanism**: Select the SASL (Simple Authentication and Security Layer) authentication mechanism to use, Defautls to `PLAIN`; `SCRAM-SHA-256` and `SCRAM-SHA-512` are also available.
- **Username**: The username for authentication.
- **Password**: The password for authentication.

Schema Registry

This section governs Kafka Schema Registry Authentication for AVRO-encoded data with a schema stored in the Confluent Schema Registry.

Enabled: defaults to `No`. When toggled to `Yes`:

- **Schema registry URL**: URL for access to the Confluent Schema Registry. (E.g., `http://<hostname>: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.

- **TLS enabled**: defaults to `No`. When toggled to `Yes,` displays the following TLS settings for the Schema Registry:

  ### TLS Settings (Schema Registry)

  > i 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 (e.g., the system's CA). Defaults to **No**.

- **Server name (SNI)**: Server name for the SNI (Server Name Indication) TLS extension. This must be a host name, not an IP address.

- **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.

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.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Max record size (KB, uncompressed): Maximum size (KB) of each record batch before compression. Setting should be < `message.max.bytes settings` in Kafka brokers. Defaults to `768`.

Max events per batch: 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`.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__topicOut`
- `__key`
- `__headers`
- `__keySchemaIdOut`
- `__valueSchemaIdOut`

Elasticsearch

Cribl LogStream can send events to an Elasticsearch cluster using the Bulk API.

Configuring Cribl LogStream to Output to Elasticsearch

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Elasticsearch** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Elasticsearch > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Elasticsearch Destination definition.

Bulk API URL: URL of an Elasticsearch cluster to send events to. (E.g., `http://<myElasticCluster>:9200/_bulk`.)

Index: Elasticsearch Index where to send events to. Note that this value can be overwritten by an event's `__index` field.

Type: Specify document type to use for events. Note that this value can be overwritten by an event's `__type` field.

Authentication enabled: Set to `No` by default. Toggle to `Yes` to enter a **Username** and **Password**.

Backpressure behavior: Specify whether to block, drop, or queue events when all receivers in this group are exerting backpressure. `Defaults to Block`.

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Toggle this slider to `Yes` 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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

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.

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`.

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 LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Fields for this Destination:

- `__id`
- `__type`
- `__index`

Notes on HTTP-based Outputs

- Cribl LogStream will attempt to use keepalives to reuse a connection for multiple requests. After 2 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, LogStream will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

Honeycomb

Cribl LogStream supports sending events to a Honeycomb dataset.

Configuring Cribl LogStream to Output to Honeycomb

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Honeycomb** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Honeycomb > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Honeycomb definition.

Dataset name: Name of the dataset to send events to. (E.g., `iLoveObservabilityDataset` .)

API Key: Team API Key to which the dataset belongs. (E.g., `teamWilde` .)

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB` .

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>` . Defaults to `$CRIBL_HOME/state/queues` .

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None` ; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Toggle this slider to `Yes` 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` .

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

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.

Notes on HTTP-based Outputs

- Cribl LogStream will attempt to use keepalives to reuse a connection for multiple requests. After 2 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, LogStream will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

New Relic

Cribl LogStream supports sending events to the New Relic Log API and the New Relic Metric API.

Configuring Cribl LogStream to Output to New Relic

From the top nav of a LogStream instance or Group, select **Destinations**, then select **New Relic** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **New Relic > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this New Relic definition.

API key: Enter your New Relic Insert API key, as created on New Relic's **Insights** > **API keys** page. (For details, see New Relic's Query API documentation.)

Region: Select which New Relic region endpoint to use.

Log type: Name of the `logType` to send with events. E.g., `observability` or `access_log`.

> ℹ 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 metadata fields to (optionally) add, as **Name-Value** pairs.

- **Name**: Enter the metadata field name.
- **Value**:JavaScript expression to compute field's value, enclosed in single quotes, double quotes, or backticks. (Can evaluate to a constant.)
- **Add Field**: Click to add more metadata **Name-Value** pairs.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.

- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Whether to compress the payload body before sending. Defaults to `No`.

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`.

Max body size (KB): Maximum size of the request body. Defaults to `1000` KB.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` second.

Extra HTTP headers: Click **+ Add Header** to insert extra headers as **Name**/**Value** pairs.

SNMP Trap

Cribl LogStream supports forwarding of SNMP Traps out.

Configuring Cribl LogStream to Forward to SNMP Traps

From the top nav of a LogStream instance or Group, select **Destinations**, then select **SNMP Trap** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **SNMP Trap > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this SNMP Trap definition.

SNMP Trap destinations: One or more SNMP destinations to forward traps to.

- **Address**: Destination host.
- **Port**: Destination port. Defaults to `162` .

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.

- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

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 (e.g., 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.

InfluxDB

Cribl LogStream supports sending data to InfluxDB.

Configuring Cribl LogStream to Output to InfluxDB

From the top nav of a LogStream instance or Group, select **Destinations**, then select **InfluxDB** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **InfluxDB > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this InfluxDB definition.

Write API URL: URL of an InfluxDB cluster to send events to. (E.g., `http://localhost:8086/write` .)

Database name: The database on which to write data points.

Timestamp precision: Sets the precision for the supplied UNIX time values. Defaults to `Milliseconds` .

Dynamic value fields: When enabled, LogStream will pull the value field from the metric name. (E.g., `db.query.user` will use `db.query` as the measurement and `user` as the value field). Defaults to `Yes` .

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` .

Authentication enabled: Set to `No` by default. Toggle to `Yes` to enter a **Username** and **Password**.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Toggle this slider to `Yes` 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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

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.

MinIO

MinIO is a non-streaming Destination type, to which Cribl LogStream can output objects.

Configuring Cribl LogStream to Output to MinIO Destinations.

From the top nav of a LogStream instance or Group, select **Destinations**, then select **MinIO** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **MinIO > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this MinIO definition.

MinIO endpoint: MinIO service URL (e.g., 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. E.g., referencing a Global Variable: `myBucket-${C.vars.myVar}` . Ensure that the bucket already exists, otherwise MinIO will generate "bucket does not exist" errors.

> i 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.

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: If left blank, Cribl LogStream will fall back to `env.AWS_SECRET_ACCESS_KEY`, or to the metadata endpoint for IAM credentials.

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.

Add Output ID: When set to `Yes` (the 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 LogStream version below 2.4.0, the **Add Output ID** behavior will be switched **off** on the backend, regardless of this slider's state. This is to avoid losing any files pending in the original staging directory, upon LogStream 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.

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 to define how files are partitioned and organized. If left blank, Cribl LogStream will fall back to `event.__partition`. Defaults to ``` `${host}/${sourcetype}` ```.

> ℹ LogStream'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).

Data format: Format of the output data. Defaults to `json`.

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`.

Compress: Select the data compression format to use before moving data to final destination. Defaults to `none`. Cribl recommends setting this to `gzip`.

Backpressure behavior: Select whether to block or drop events when all receivers in this group are exerting backpressure. Defaults to `Block`.

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`

> ℹ 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** buttons to 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: If not running on AWS, you can select this option to enter your access key and secret key, directly or by reference. 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.

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 LogStream's internal secrets manager or (if enabled) an external KMS. Follow the **Create** link if you need to configure a key pair.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

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`.

Max file size (MB): Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.

Max file open time (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`.

Max file idle time (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`.

Max open files: 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`.

> i Cribl LogStream will close files when **either** of the `Max file size (MB)` or the `Max file open time (sec)` conditions is met.

Reuse connections: Whether to reuse connections between requests. The default setting (`Yes`) can improve performance.

Reject unauthorized certificates: Whether to accept certificates that cannot be verified against a valid Certificate Authority (e.g., self-signed certificates). Defaults to `Yes`.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

Field for this Destination:

- `__partition`

Prometheus

Cribl LogStream can send metric events to targets and third-party platforms that support Prometheus' `remote_write` spec.

Configuring Cribl LogStream to Output to Prometheus

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Prometheus** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Prometheus > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Prometheus output definition.

Remote Write URL: The endpoint to send events to, e.g.: `http://localhost:9200/write`

Backpressure behavior: Whether to block, drop, or queue events when all receivers are exerting backpressure.

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>` . Defaults to: `$CRIBL_HOME/state/queues` .

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None` ; `Gzip` is also available.

Authentication

This section's **Authentication type** drop-down determines the Remote Write authentication method to use. The options are:

None: The default.

Auth token: Exposes an **Auth token** field, where you can enter a bearer token to include in the authorization header.

Basic: Exposes **Username** and **Password** fields for Basic authentication.

Processing Settings

Post-Processing

Pipeline: Pipeline 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` (LogStream Node that processed the event) and `cribl_wp` (LogStream Worker Process that processed the event). Supports wildcards. Other options include:

- `cribl_pipe` – LogStream Pipeline that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

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.

Metric renaming expression: A JavaScript expression that can be used to rename metrics and their dimensions. The default expression – `name.replace(/\./g, '_')` – replaces all instances of `.` (a character unsupported by Prometheus) with the supported `_`.

Send metadata: Whether to generate and send metrics' metadata (`type` and `metricFamilyName`) along with the metrics. The default `Yes` value 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.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

If an event contains the internal field `__criblMetrics`, LogStream will send it to the HTTP endpoint as a metric event. Otherwise, LogStream will drop the event.

Notes on HTTP-based Outputs

- Unlike other HTTP-based Destinations, Prometheus does not display an **Advanced Settings > Compress** option. The Prometheus `remote_write` spec assumes that payloads are snappy-compressed by default.

- LogStream will attempt to use keepalives to reuse a connection for multiple requests. After 2 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, LogStream will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

Wavefront

Cribl LogStream supports sending events to Wavefront analytics.

Configuring Cribl LogStream to Output to Wavefront

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Wavefront** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Wavefront > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Wavefront definition.

Auth token: Wavefront API authentication token. For details, see Wavefront's Generating an API Token topic. Required.

Domain name: WaveFront domain name, e.g., `longboard` . Required.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB` .

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Whether to compress the payload body before sending. Defaults to `No`.

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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` second.

Extra HTTP headers: Click **+ Add Header** to insert extra headers as **Name**/**Value** pairs.

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.

SignalFx

Cribl LogStream supports sending events to SignalFx.

Configuring Cribl LogStream to Output to SignalFx

From the top nav of a LogStream instance or Group, select **Destinations**, then select **SignalFx** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **SignalFx > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this SignalFx definition.

Auth token: SignalFx API access token. For details, see SignalFx's Manage Tokens topic. Required.

Realm: SignalFx realm name (e.g., `us0`). Required.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB` .

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Whether to compress the payload body before sending. Defaults to `No`.

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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` second.

Extra HTTP headers: Click **+ Add Header** to insert extra headers as **Name**/**Value** pairs.

Notes About SignalFx

For details on integrating with SignalFx, see the SignalFx Developers Guide, with particular reference to the SignalFx HTTP Send Metrics Reference.

Sumo Logic

Cribl LogStream can send log and metric events to Sumo Logic over HTTP.

Configuring Cribl LogStream to Output to Sumo Logic

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Sumo Logic*** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Sumo Logic > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Sumo Logic Destination definition.

API URL: Enter the URL of the Sumo Logic HTTP collector to which events should be sent. (E.g., `https://endpoint6.collection.us2.sumologic.com/receiver/v1/http/<long-hash>` .)

Custom source name: Optionally, override the source `name` configured on the Sumo Logic HTTP collector. This value will be sent with events via the `X-Sumo-Name` HTTP header.

Custom source category: Optionally, override the source `category` configured on the Sumo Logic HTTP collector. This value will be sent with events via the `X-Sumo-Category` HTTP header.

Backpressure behavior: Whether to block, drop, or queue events when all receivers in this group are exerting backpressure.

Persistent Queue Settings

> ℹ This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Toggle this slider to `Yes` 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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

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.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

If an event contains the internal field `__criblMetrics`, LogStream will send it to Sumo Logic as a metric event. Otherwise, LogStream will send it as a log event.

Notes on HTTP-based Outputs

- Cribl LogStream will attempt to use keepalives to reuse a connection for multiple requests. After 2 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, LogStream will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

Datadog

Cribl LogStream can send log and metric events to Datadog. (Datadog supports metrics only of type `gauge`, `counter`, and `rate` via its REST API.)

LogStream sends events to the following Datadog endpoints in the US region. Use a DNS lookup to discover and include the corresponding IP addresses in your firewall rules' allowlist.

- Logs: https://http-intake.logs.datadoghq.com/v1/input
- Metrics: https://api.datadoghq.com/api/v1/series

Configuring Cribl LogStream to Output to Datadog

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Datadog** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Datadog > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Destination definition.

API Key: Enter an API key available in your Datadog profile.

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, LogStream 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 (i.e., 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.

Tags: List of tags to send with logs (e.g., `env:prod`, `env_staging:east`).

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. Defaults to `info`; the drop-down offers many other severity options.

> i Datadog uses the above five fields (`source`, `host`, `__service`, `__severity`, and `tags`) to enhance searches and UX.

Backpressure behavior: Specify whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`. Select `Gzip` to enable compression.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Toggle this slider to `Yes` to compress log events' 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`.

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

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.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

If an event contains the internal field `__criblMetrics`, LogStream will send it to Datadog as a metric event. Otherwise, LogStream will send it as a log event.

You can use these fields to override outbound event values for log events:

- `__service`
- `__severity`

No internal fields are supported for metric events.

For More Information

You might find these Datadog references helpful:

- Submit Metrics
- Send Logs
- Metrics Types

Webhook

Cribl LogStream can send log and metric events to webhooks and other generic HTTP endpoints.

Configuring Cribl LogStream to Output via HTTP

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Webhook** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Webhook > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this HTTP output definition.

URL: Endpoint URL to send events to.

Method: The HTTP verb to use when sending events. Defaults to `POST`. Change this to `PUT` or `PATCH` where required by the endpoint.

Send events as: The format in which to send out events. One of:

- `NDJSON` (newline-delimited JSON): The default.
- `JSON Array`: Arrays in JSON-parseable format.
- `Custom`: Exposes the following additional fields to define the output format:
  - **Source expression**: JavaScript expression to evaluate on every event; LogStream will send the result of that evaluation instead of the original event. Sample expression: ``` `${fieldA}, ${fieldB}` ``` (with literal backticks). Defaults to `__httpOut` – i.e., the value of the `__httpOut` field. Use the button at right to open a validation modal.

- **Drop when null**: If toggled to `Yes`, LogStream 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.

Backpressure behavior: Whether to block, drop, or queue events when all receivers are exerting backpressure.

Persistent Queue Settings

> i This section is displayed when the **Backpressure behavior** is set to **Persistent Queue**.

Max file size: The maximum size to store in each queue file before closing it. Enter a numeral with units of KB, MB, etc. Defaults to `1 MB`.

Max queue size: The maximum amount of disk space the queue is allowed to consume. Once this limit is reached, queueing is stopped, and data blocking is applied. Enter a numeral with units of KB, MB, etc.

Queue file path: The location for the persistent queue files. This will be of the form: `your/path/here/<worker-id>/<output-id>`. Defaults to `$CRIBL_HOME/state/queues`.

Compression: Codec to use to compress the persisted data, once a file is closed. Defaults to `None`; `Gzip` is also available.

Authentication

This section's **Authentication type** drop-down determines the authentication method to use for HTTP requests. The options are:

None: The default.

Basic: Exposes **Username** and **Password** fields for Basic authentication.

Token: Exposes a **Token** field to enter a bearer token to include in the authorization header.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 LogStream Pipeline that processed the event). Supports wildcards. Other options include:

- `cribl_host` – LogStream Node that processed the event.
- `cribl_wp` – LogStream Worker Process that processed the event.
- `cribl_input` – LogStream Source that processed the event.
- `cribl_output` – LogStream Destination that processed the event.

Advanced Settings

Compress: Toggle this slider to `Yes` 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` .

Max body size (KB): Maximum size of the request body. Defaults to `4096` KB.

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.

Internal Fields

Cribl LogStream uses a set of internal fields to assist in forwarding data to a Destination.

If an event contains the internal field `__criblMetrics`, LogStream will send it to the HTTP endpoint as a metric event. Otherwise, LogStream will send it as a log event.

Notes on HTTP-based Outputs

- Cribl LogStream will attempt to use keepalives to reuse a connection for multiple requests. After 2 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, LogStream will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

DevNull

The DevNull Destination simply drops events. Cribl provides this as a basic output to test Pipelines and Routes.

Configuring Cribl LogStream to Forward to DevNull

DevNull requires no configuration: A DevNull Destination is preconfigured and active as soon as you install Cribl LogStream.

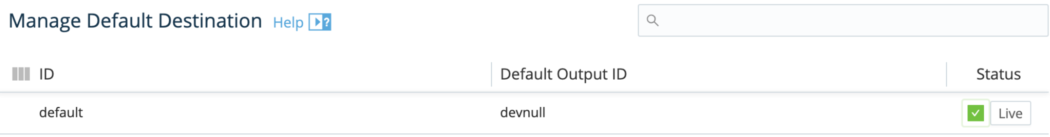
To verify this, from the top nav of a LogStream instance or Group, select **Destinations**, then select **Devnull** from the **Data Destinations** page's tiles or the **Destinations** left nav. Look for the **Live** indicator at the top right.

Default

The **Default** Destination simply enables you to specify a default output from among your already configured Destinations.

Configuring Cribl LogStream's Default Destination

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Default** from the **Data Destinations** page's tiles or the **Destinations** left nav. From the resulting **Manage Default Destination** page, click anywhere on the `default` row to proceed.



Manage Default Destination

ID	Default Output ID	Status
default	devnull	✓ Live

Default Destination – click to configure

In the resulting **Destinations > Default** modal, use the **Default Output ID** drop-down to select one of your configured Destinations. After you click **Save**, this will become LogStream's default Destination.

The only other field here is the **Output ID**, whose value is locked to `default` .

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.

Packs

About Packs

Packs, introduced in LogStream 3.0, enable LogStream administrators and developers to pack up and share complex configurations and workflows across multiple Worker Groups, or across organizations.

Packs = Portability

With a medium or large LogStream deployment, using Packs can simplify and accelerate your work. Packs can also accelerate internal troubleshooting, and accelerate working with Cribl Support, because they facilitate quickly replicating your LogStream environment.

For example, where LogStream will import a Pipeline's configuration only if it finds its referenced Lookup file(s), a Pack can consolidate this dependency, making the Pipeline portable across LogStream instances. You can develop and test a configuration, and then port it from development to production instances, or readily deploy it to multiple Worker Groups.

We don't claim to have brokered world peace here, but we do modestly hope to promote a stable, prosperous Pax Criblatica for the LogStream ecosystem.

What Is a Pack?

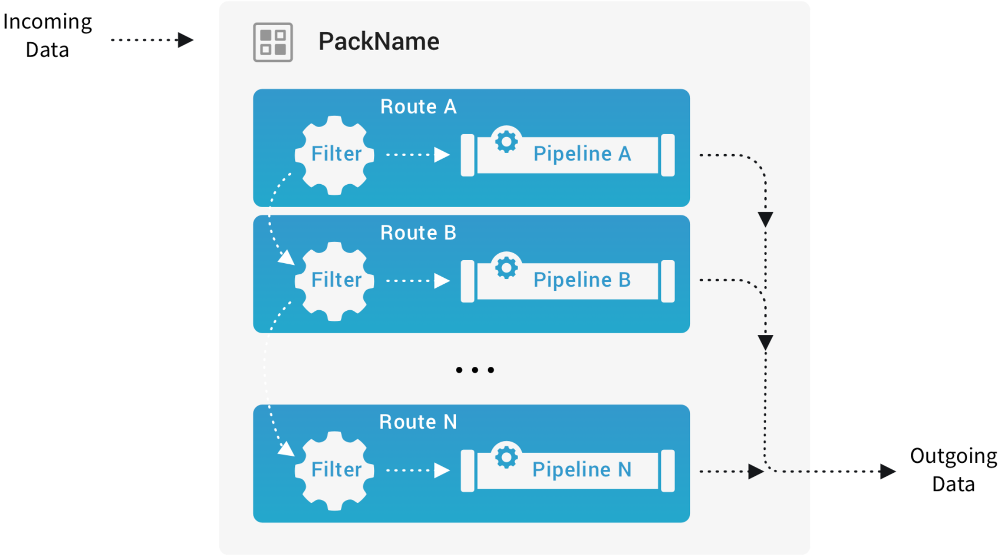
Packs are implemented as a user interface (described on this page) and as a `.crbl` file format.

What's in a Pack?

Currently, a pack can pack up everything between a Source and a Destination:

- Routes (Pack-level)

- Pipelines (Pack-level)
- Functions (built-in and custom)
- Sample data files
- Knowledge objects (Lookups, Parsers, Global Variables, Grok Patterns, and Schemas)



A Pack with internal Routes & Pipelines; no Knowledge or samples

As the above list suggests, a Pack can encapsulate a whole set of infrastructure for a given use case.

What's Not in a Pack?

Sources, Collectors, and Destinations are external to Packs, so you can't specify them within a Pack. This excludes a few other things:

- Routes configured within a Pack can't specify a Destination.
- Packs can't include Event Breakers, which are associated with Sources.

You connect a Pack with a Source and Destination by attaching it to a Route (see below), just as you'd attach a Pipeline.

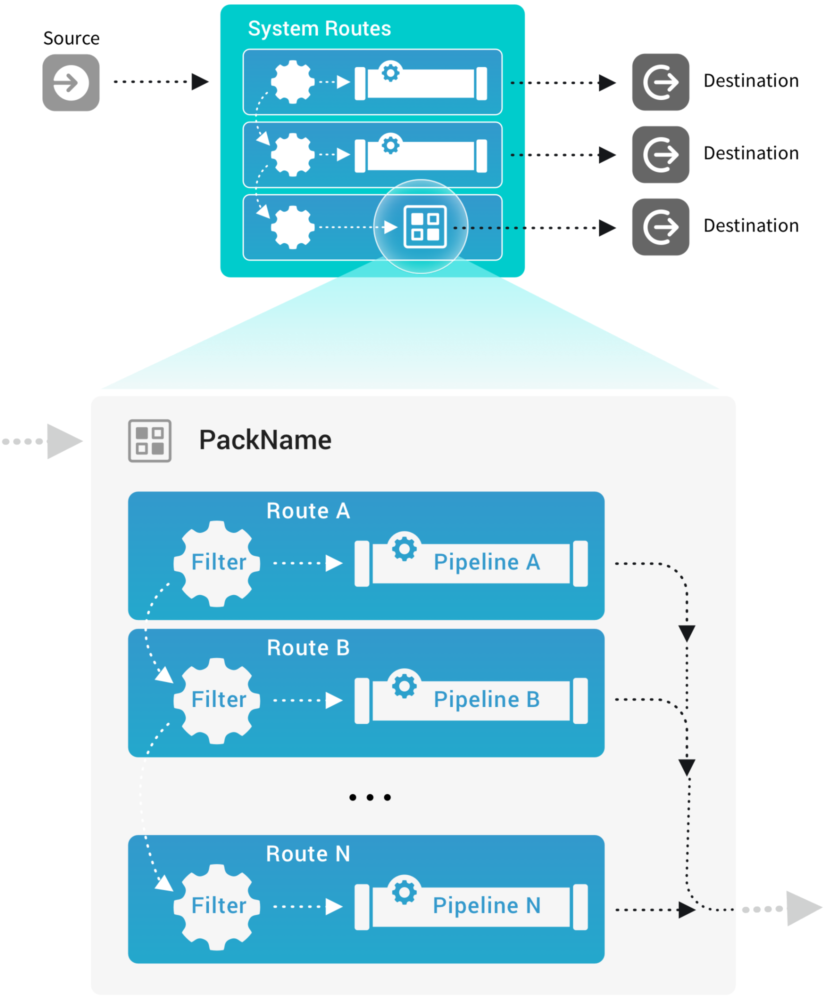
Using Packs

These instructions cover using predefined Packs, as well as creating and modifying Pack configurations.

Where Can I Use Packs?

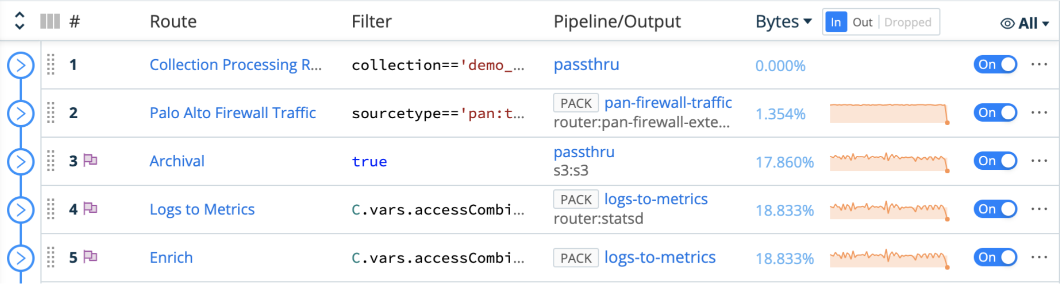
Wherever you can reference a Pipeline, you can specify a Pack:

- In Sources, where you attach pre-processing Pipelines.
- In Destinations, where you attach post-processing Pipelines.
- In Routes, in the Routing table's **Pipeline/Output** column.



A Pack snaps into LogStream like an enhanced Pipeline

Packs are distinguished in in the display with a **PACK** badge, as you can see here:



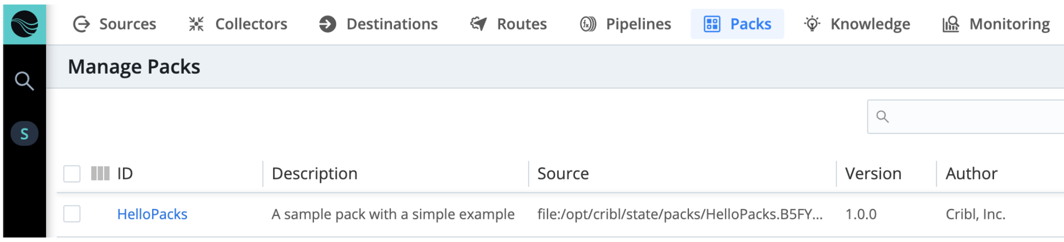
#	Route	Filter	Pipeline/Output	Bytes ▾ In | Out | Dropped	All ▾
1	Collection Processing R...	`collection=='demo_...`	passthru	0.000%	On ···
2	Palo Alto Firewall Traffic	`sourcetype=='pan:t...`	PACK pan-firewall-traffic router:pan-firewall-exte...	1.354%	On ···
3	Archival	`true`	passthru s3:s3	17.860%	On ···
4	Logs to Metrics	`C.vars.accessCombi...`	PACK logs-to-metrics router:statsd	18.833%	On ···
5	Enrich	`C.vars.accessCombi...`	PACK logs-to-metrics	18.833%	On ···

PACKs badged in Routing table's Pipeline column

LogStream's **Monitoring** page includes a **Packs** link where you can monitor Packs' throughput.

Accessing Packs

In a single-instance deployment, Packs are global. From LogStream's top-level navigation, just click **Packs**.

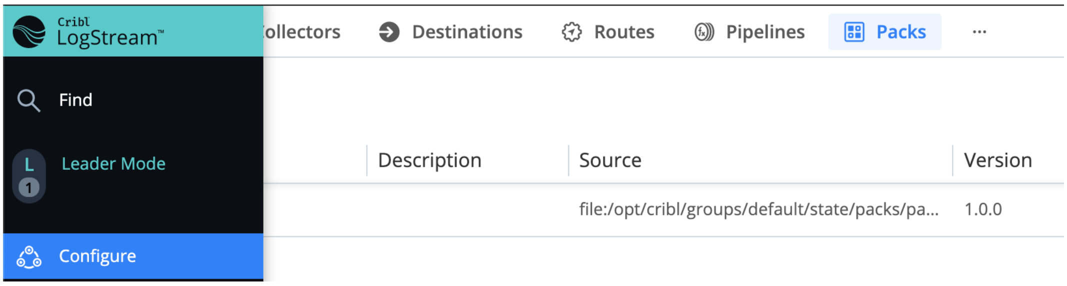


Manage Packs

ID	Description	Source	Version	Author
HelloPacks	A sample pack with a simple example	file:/opt/cribl/state/packs/HelloPacks.B5FY...	1.0.0	Cribl, Inc.

Packs, single-instance navigation

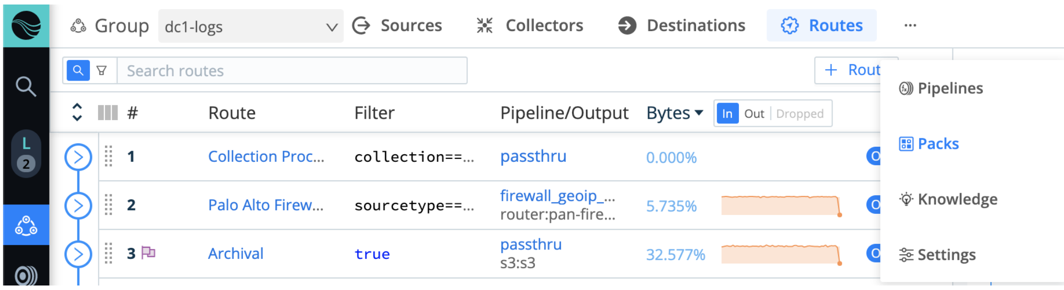
In a distributed deployment (Leader mode) with the default single Worker Group, select **Configure** from the left nav, then **Packs** from the resulting top nav.



Description	Source	Version
	file:/opt/cribl/groups/default/state/packs/pa...	1.0.0

Packs, Leader mode

In a distributed deployment with multiple Worker Groups, Packs are associated with (and installed within) Worker Groups. Navigate to the parent Worker Group, then select **Packs** from that Group's top nav.



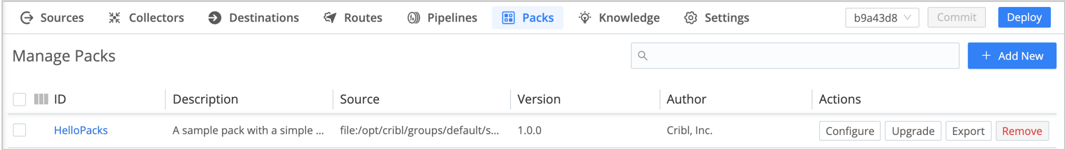
#	Route	Filter	Pipeline/Output	Bytes
1	Collection Proc...	collection==...	passthru	0.000%
2	Palo Alto Firew...	sourcetype==...	firewall_geoip_... router:pan-fire...	5.735%
3	Archival	true	passthru s3:s3	32.577%

Worker Group > Packs navigation

> As the top nav adds more controls on narrower browsers, **Packs** and other right-side links can move onto the ••• overflow menu, as shown above.
>
> By design, you can readily share Packs **across** Worker Groups by exporting/importing them (both covered below).

Getting Started with Packs

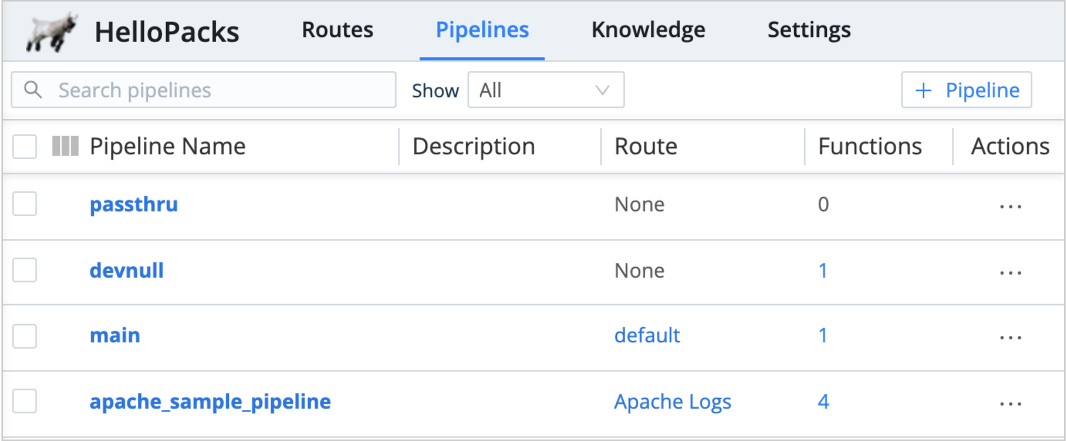
To unpack Packs, use the above instructions (per deployment type) to navigate to the **HelloPacks** example Pack shipped with LogStream. On the **Manage Packs** page, click this Pack's row to see its Pack's configuration.



ID	Description	Source	Version	Author	Actions
HelloPacks	A sample pack with a simple ...	file:/opt/cribl/groups/default/s...	1.0.0	Cribl, Inc.	Configure Upgrade Export Remove

Manage Packs page with example Pack

Click **Pipelines** on the Pack's submenu, and you'll see that the Pack includes `devnull` , `main` , and `passthru` Pipelines, corresponding to the default Pipelines provided at LogStream's global level. This Pack also includes an Apache-specific sample Pipeline – click it to unpack that, too.

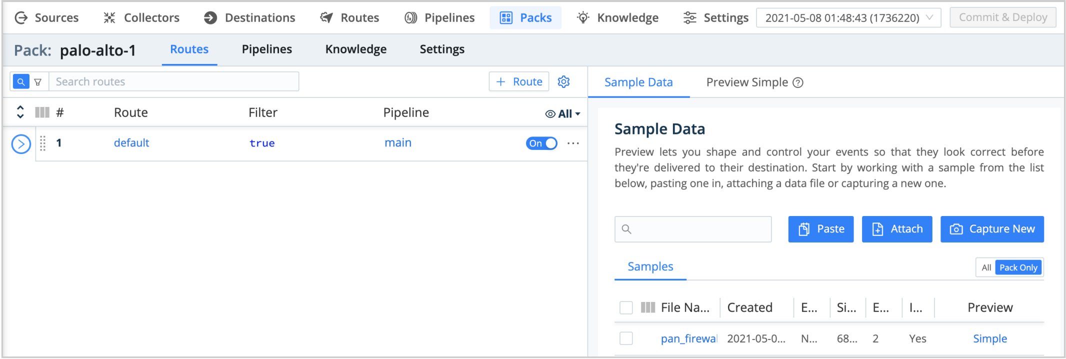


Pipeline Name	Description	Route	Functions	Actions
passthru		None	0	...
devnull		None	1	...
main		default	1	...
apache_sample_pipeline		Apache Logs	4	...

Click **Routes** on the Pack's submenu, and you'll see that this Pack also provides both a `default` and an Apache-specific Route.

Pack Configuration

Once loaded, each Pack displays a submenu with familiar links – a subset of LogStream's top nav above it: **Routes**, **Pipelines**, **Knowledge**, and **Settings** on the left pane, along with **Sample Data**, and **Preview Simple** on the right.



#	Route	Filter	Pipeline	
1	default	`true`	main	On

Sample Data

Preview lets you shape and control your events so that they look correct before they're delivered to their destination. Start by working with a sample from the list below, pasting one in, attaching a data file or capturing a new one.

File Na...	Created	E...	Si...	E...	I...	Preview
pan_firewal	2021-05-0...	N...	68...	2	Yes	Simple

Configuring a Pack

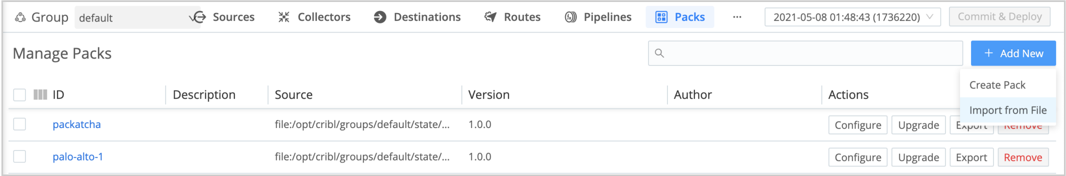
The left pane's links give you access to configuration objects specific to this Pack. In the right pane, you can toggle between displaying **All** sample files available on your LogStream instance, versus samples internal to the the **Pack Only**.

Basically, you can manipulate all the options here as you'd work with their big sister or brother in LogStream's global navigation.

Importing or Upgrading a Pack

To import a new Pack, or an updated version of an existing Pack, from your filesystem:

1. Navigate to the **Manage Packs** page.
2. Click **+ Add New**.
3. From the submenu, select **Import from File**.
4. Select the file to import.



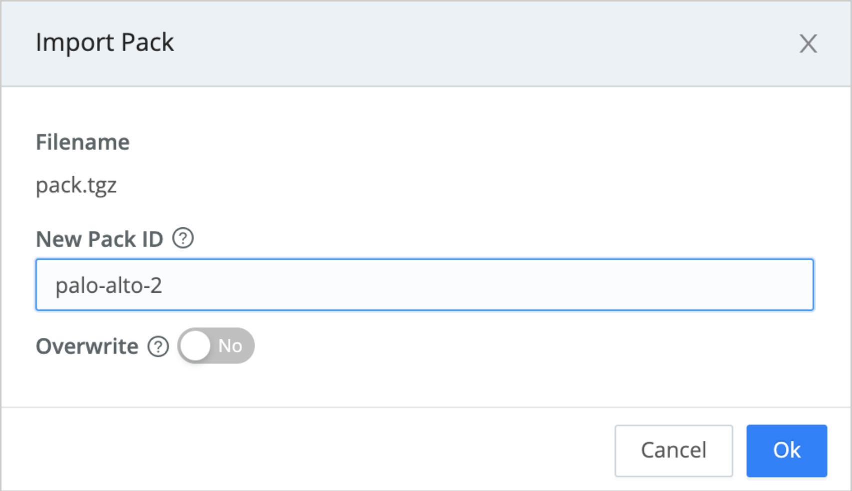
ID	Description	Source	Version	Author	Actions
packatcha		file:/opt/cribl/groups/default/state/...	1.0.0		Configure Upgrade Export Remove
palo-alto-1		file:/opt/cribl/groups/default/state/...	1.0.0		Configure Upgrade Export Remove

Importing a Pack

Upgrading an Existing Pack

Each Pack installed within a given Worker Group (or single-instance deployment) must have a unique ID. The name is based on the Pack's internal configuration, not its container's file name.

If you import a Pack whose internal ID matches an installed Pack – whether an update, or just a duplicate – you'll be prompted to assign a unique **New Pack ID** to import it as a separate Pack.



Import Pack

Filename: pack.tgz

New Pack ID: palo-alto-2

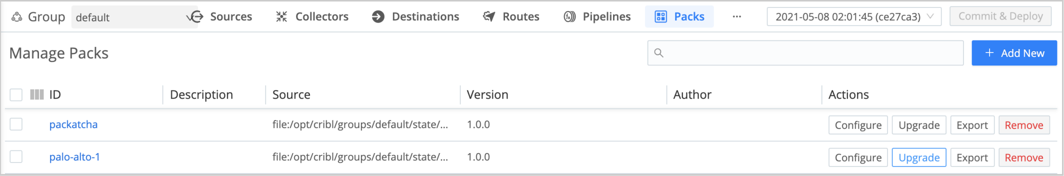
Overwrite: No

Renaming a Pack on import

You'll also have the option to **Overwrite** the installed Pack, reusing the same ID.

> If you toggle this option to `Yes`, the imported Pack will completely overwrite your existing Pack's configuration.

To explicitly upgrade an existing Pack, you can instead click the **Configure** button on its rows.



Manage Packs

ID	Description	Source	Version	Author	Actions
packatcha		file:/opt/cribl/groups/default/state/...	1.0.0		Configure Upgrade Export Remove
palo-alto-1		file:/opt/cribl/groups/default/state/...	1.0.0		Configure Upgrade Export Remove

Upgrading an existing Pack

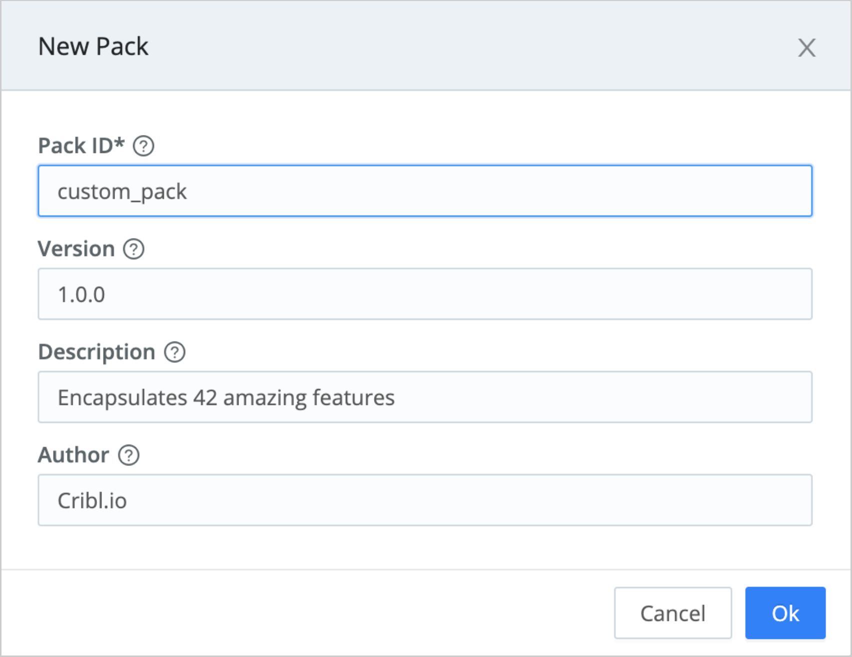
> If you've modified an installed Pack, LogStream will prevent overwriting the Pack, to prevent deletion of your locally created

> resources.

Creating a Pack

You can create a new Pack from scratch, to consolidate and export multiple LogStream configuration objects:

1. Navigate to the **Manage Packs** page.
2. Click **+ Add New**.
3. From the submenu, select **Create Pack**.
4. In the resulting **New Pack** modal, fill in a unique **Pack ID** and other details, and click **OK**.



New Pack

Pack ID*: custom_pack

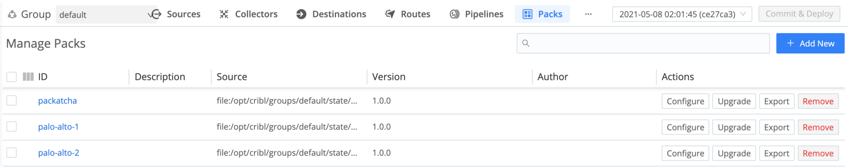
Version: 1.0.0

Description: Encapsulates 42 amazing features

Author: Cribl.io

Creating a Pack

5. On the **Manage Packs** page, click the new Pack's row to open the Pack.



Manage Packs

ID	Description	Source	Version	Author	Actions
packatcha		file:/opt/cribl/groups/default/state/...	1.0.0		Configure Upgrade Export Remove
palo-alto-1		file:/opt/cribl/groups/default/state/...	1.0.0		Configure Upgrade Export Remove
palo-alto-2		file:/opt/cribl/groups/default/state/...	1.0.0		Configure Upgrade Export Remove

Manage Packs page

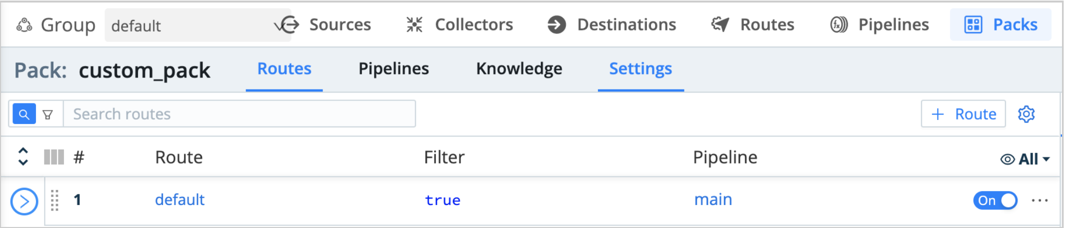
6. Use the standard LogStream controls (see above) to configure and save the infrastructure you want to pack up. As you save changes in the UI,

they're saved to the Pack.

Modifying Pack Settings

You can update a Pack's metadata (Version, Description, Author, etc.) If you're developing a new Pack to share, you'll want to use this interface to populate the Pack's README.

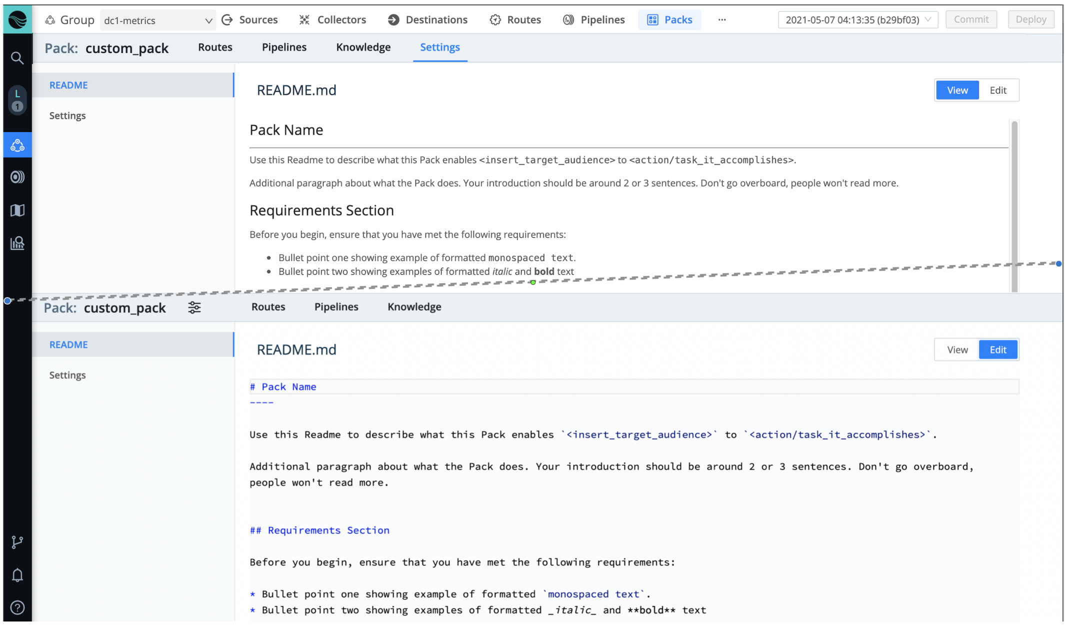
1. From the Pack's submenu, select Settings.



#	Route	Filter	Pipeline
1	default	true	main

Pack Settings

2. To populate the Pack's README file, toggle **View** to **Edit**, replace the placeholder markdown content, and **Save**.



Pack Name

Use this Readme to describe what this Pack enables <insert_target_audience> to <action/task_it_accomplishes>.

Additional paragraph about what the Pack does. Your introduction should be around 2 or 3 sentences. Don't go overboard, people won't read more.

Requirements Section

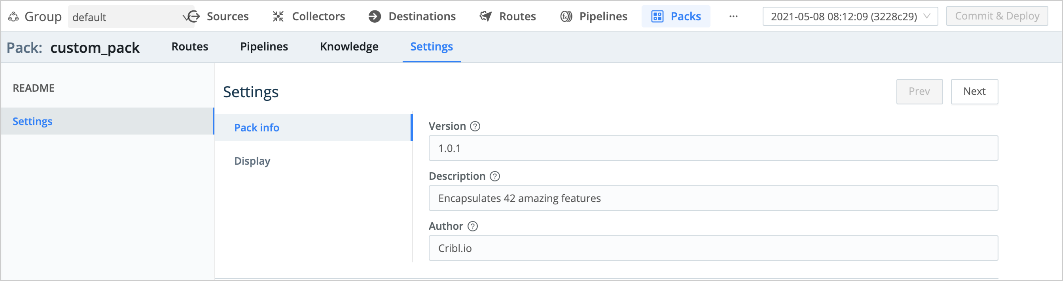
Before you begin, ensure that you have met the following requirements:

- Bullet point one showing example of formatted `monospaced text`.
- Bullet point two showing examples of formatted *italic* and **bold** text

```
# Pack Name
----

Use this Readme to describe what this Pack enables `<insert_target_audience>` to `<action/task_it_accomplishes>`.

Additional paragraph about what the Pack does. Your introduction should be around 2 or 3 sentences. Don't go overboard,
people won't read more.


## Requirements Section

Before you begin, ensure that you have met the following requirements:

* Bullet point one showing example of formatted `monospaced text`.
* Bullet point two showing examples of formatted _italic_ and **bold** text
```

Editing Pack's README

3. To update other metadata, click the left **Settings** tab.



Settings

Version: 1.0.1

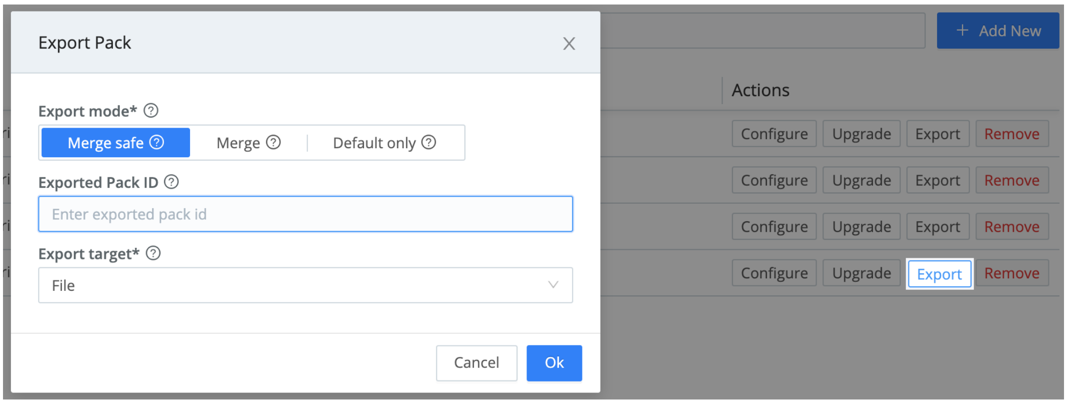
Description: Encapsulates 42 amazing features

Author: Cribl.io

Editing Pack's metadata

Exporting a Pack

To export a newly created or modified Pack, click its **Export** button on the Packs page.



Export Pack

Export mode*: Merge safe | Merge | Default only

Exported Pack ID: Enter exported pack id

Export target*: File

Exporting a Pack

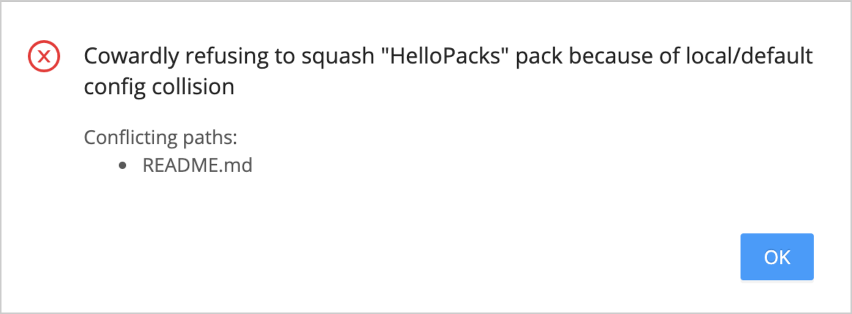
The resulting Export Pack modal provides the following options.

Export Mode

Select one of these three buttons:

- **Merge safe**: 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.
- **Default only**: Export only the Pack's original configuration, without local modifications.

The **Merge safe** option is conservative, and will block the export where conflicting modified contents can't be readily merged with the Pack's original contents:



Cowardly refusing to squash "HelloPacks" pack because of local/default config collision

Conflicting paths:

- README.md

***Merge safe** error*

If you encounter this error, use the **Merge** or **Default only** export mode instead.

Export Target

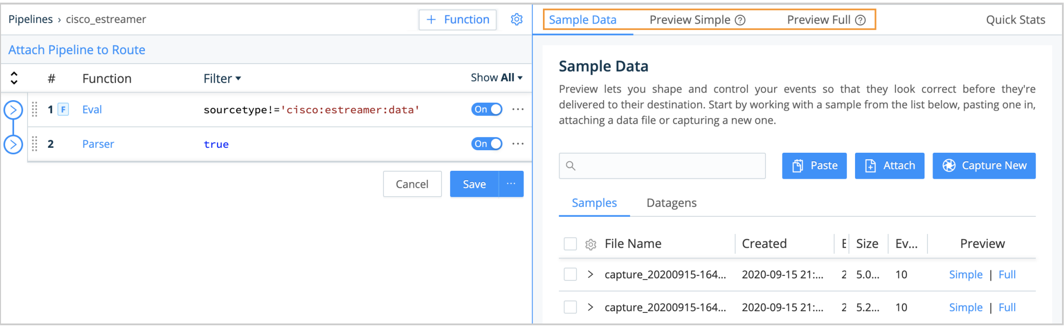
The options here are:

- **File** (the default): You'll be prompted to confirm a file name and destination after you click **OK**.
- **Group**: Selecting this displays a **Group** drop-down, prompting you to select an existing Worker Group to export the Pack to. (The current Worker Group is automatically omitted from the options.)

Data Preview

LogStream's Sample Data Preview features enable you to visually inspect events as they flow into and out of a Pipeline. Preview helps you shape and control events before they're delivered to a Destination, and helps you troubleshoot Pipeline Functions.

Preview works by taking a set of sample events and passing them through the Pipeline, while displaying the inbound and outbound results in a separate pane. Any time a Function is modified, added, or removed, the Pipeline changes, and so does its displayed output.



#	Function	Filter	Show All
1	Eval	`sourcetype!='cisco:estreamer:data'`	On
2	Parser	`true`	On

Sample Data

Preview lets you shape and control your events so that they look correct before they're delivered to their destination. Start by working with a sample from the list below, pasting one in, attaching a data file or capturing a new one.

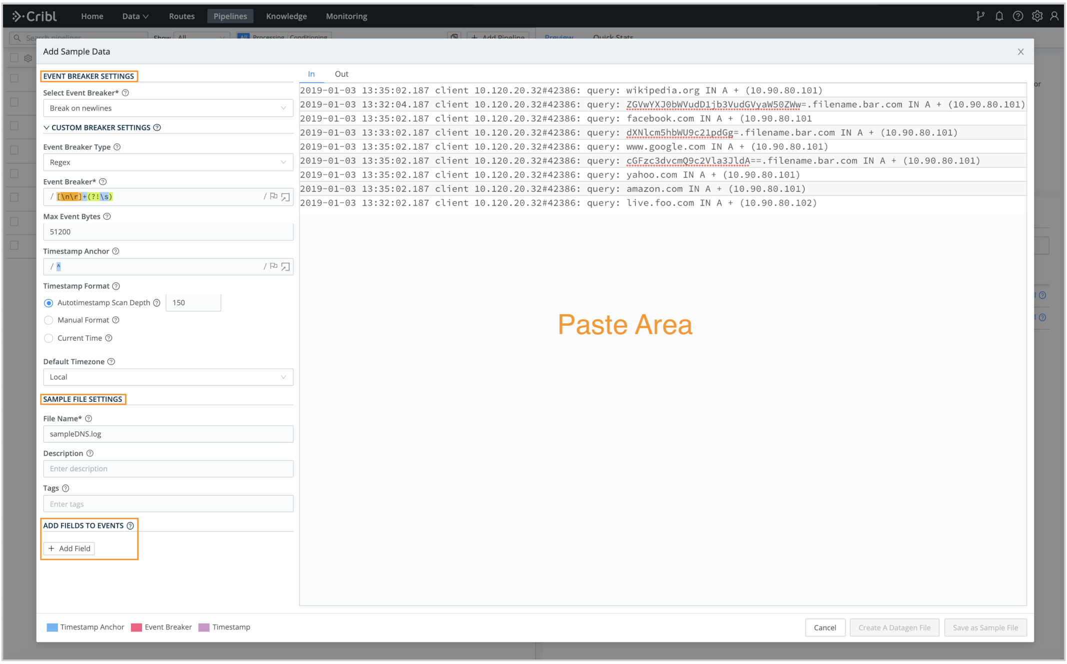
File Name	Created	E	Size	Ev...	Preview
capture_20200915-164...	2020-09-15 21:...	2	5.0...	10	Simple | Full
capture_20200915-164...	2020-09-15 21:...	2	5.2...	10	Simple | Full

Preview options

While you're in a Pipeline, you can add samples through one of the supported options: **Paste**, **Attach**, or **Capture New**. The **Paste** and **Attach** options work with content that needs to be broken into events, while the **Capture New** option works with events only.

Adding Sample Data (Using Paste as an Example)

When you click on the corresponding option, you'll be presented with a modal like the one shown below.



EVENT BREAKER SETTINGS

Select Event Breaker*: Break on newlines

CUSTOM BREAKER SETTINGS

Event Breaker Type: Regex

Event Breaker*: / [\n\r]+(?!\s) /

Max Event Bytes: 51200

Timestamp Anchor: / ^ /

Timestamp Format:
- ● Autotimestamp Scan Depth 150
- ○ Manual Format
- ○ Current Time

Default Timezone: Local

SAMPLE FILE SETTINGS

File Name*: sampleDNS.log

Description: Enter description

Tags: Enter tags

ADD FIELDS TO EVENTS

+ Add Field

```
2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: wikipedia.org IN A + (10.90.80.101)
2019-01-03 13:32:04.187 client 10.120.20.32#42386: query: ZGVwYXJ0bWVudD1jb3VudGVyaW50ZWw=.filename.bar.com IN A + (10.90.80.101)
2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: facebook.com IN A + (10.90.80.101
2019-01-03 13:33:02.187 client 10.120.20.32#42386: query: dXNlcm5hbWU9c21pdGg=.filename.bar.com IN A + (10.90.80.101)
2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: www.google.com IN A + (10.90.80.101)
2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: cGFzc3dvcmQ9c2Vla3JldA==.filename.bar.com IN A + (10.90.80.101)
2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: yahoo.com IN A + (10.90.80.101)
2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: amazon.com IN A + (10.90.80.101)
2019-01-03 13:32:02.187 client 10.120.20.32#42386: query: live.foo.com IN A + (10.90.80.102)
```

🟦 Timestamp Anchor 🟥 Event Breaker 🟪 Timestamp

Add Sample Data modal

Paste Area

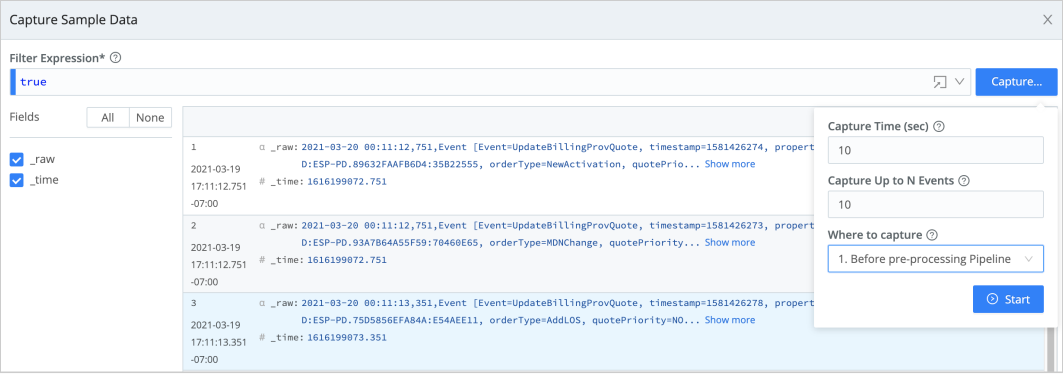
This is where the content of the paste (or uploaded file) is displayed.

Event Breaker Settings

An Event Breaker is a regular expression that tells Cribl LogStream how to break the file or pasted content into events. Breaking will occur at the **start** of the match. Cribl LogStream 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.

Capturing Sample Data

The **Capture New** button opens a slightly different modal – it does not require event breaking. In the composite screenshot below, we've already captured some events using the **Capture** drop-down.



Filter Expression*: `true`

☑ _raw
☑ _time

```
1             α _raw: 2021-03-20 00:11:12,751,Event [Event=UpdateBillingProvQuote, timestamp=1581426274, propert
2021-03-19            D:ESP-PD.89632FAAFB6D4:35B22555, orderType=NewActivation, quotePrio... Show more
17:11:12.751  # _time: 1616199072.751
-07:00
2             α _raw: 2021-03-20 00:11:12,751,Event [Event=UpdateBillingProvQuote, timestamp=1581426273, propert
2021-03-19            D:ESP-PD.93A7B64A55F59:70460E65, orderType=MDNChange, quotePriority... Show more
17:11:12.751  # _time: 1616199072.751
-07:00
3             α _raw: 2021-03-20 00:11:13,351,Event [Event=UpdateBillingProvQuote, timestamp=1581426278, propert
2021-03-19            D:ESP-PD.75D5856EFA84A:E54AEE11, orderType=AddLOS, quotePriority=NO... Show more
17:11:13.351  # _time: 1616199073.351
-07:00
```

Capture Time (sec): 10

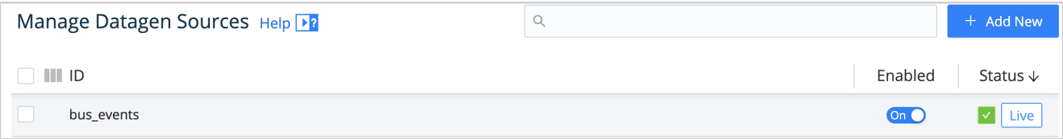
Capture Up to N Events: 10

Where to capture: 1. Before pre-processing Pipeline

Capture New > Capture Sample Data modal

Capturing from a Single Source or Destination

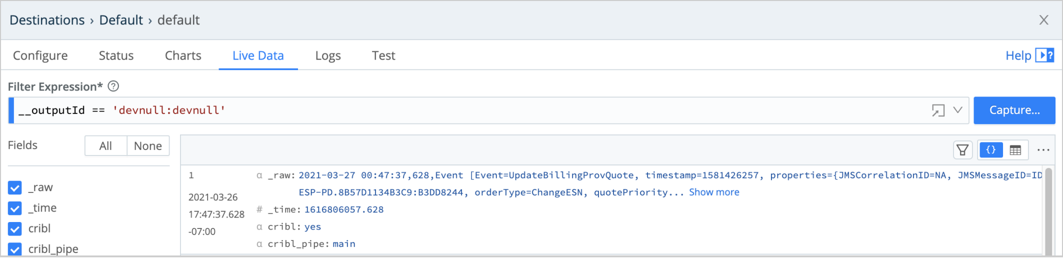
To capture data from a single enabled Source or Destination, it's fastest to use the Sources or Destinations UI instead of the Preview pane. You can initiate an immediate capture by clicking the **Live** button on the Source's or Destination's configuration row.



ID	Enabled	Status
bus_events	On	✓ Live

Source > Live button

You can similarly start an immediate capture from within an enabled Source's or Destination's configuration modal, by clicking the modal's **Live Data** tab.



Filter Expression*: `__outputId == 'devnull:devnull'`

☑ _raw
☑ _time
☑ cribl
☑ cribl_pipe

```
1             α _raw: 2021-03-27 00:47:37,628,Event [Event=UpdateBillingProvQuote, timestamp=1581426257, properties={JMSCorrelationID=NA, JMSMessageID=ID
2021-03-26            ESP-PD.8B57D1134B3C9:B3DD8244, orderType=ChangeESN, quotePriority... Show more
17:47:37.628  # _time: 1616806057.628
-07:00        α cribl: yes
              α cribl_pipe: main
```

Destination modal > Live Data tab

Fields

Use the **Fields** sidebar (at left) to streamline how events are displayed. You can toggle among **All** fields, **None** (to reset the display), and check boxes to enable/disable individual fields by name.

Field Type Symbols

Within the right Preview pane, each field's type is indicated by one of these leading symbols:

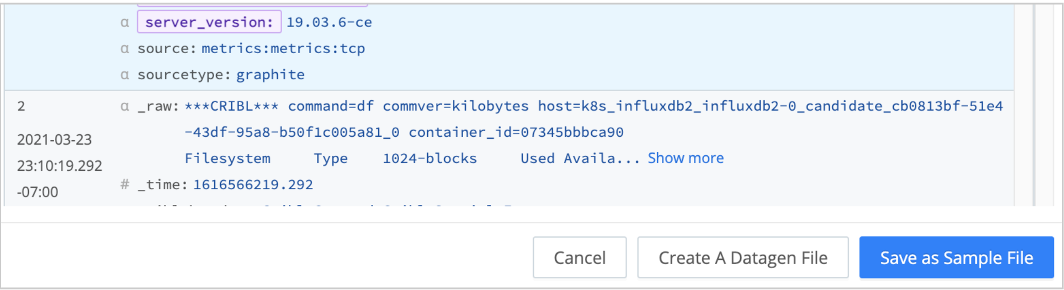
Symbol	Meaning
α	string
#	numeric
b	boolean
m	metric
{}	JSON object
[]	array

On JSON objects and arrays, you'll also see:

Symbol	Meaning
+	expandable
-	collapsible

Saving Sample Data

The Preview pane's **Add Sample Data** or **Capture Sample Data** modal, once successfully populated with data, provides options to save the data as a sample and/or datagen file. Click the appropriate button, accept or modify the default/generated file name and other options, and confirm the save.

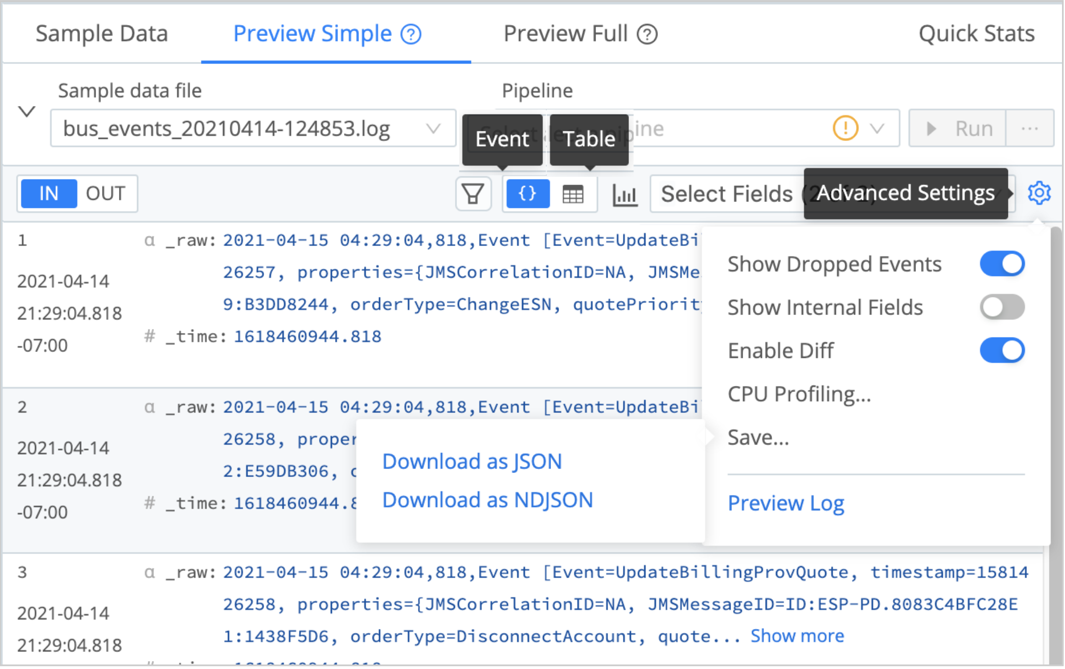


```
                α server_version: 19.03.6-ce
                α source: metrics:metrics:tcp
                α sourcetype: graphite
2               α _raw: ***CRIBL*** command=df commver=kilobytes host=k8s_influxdb2_influxdb2-0_candidate_cb0813bf-51e4
2021-03-23              -43df-95a8-b50f1c005a81_0 container_id=07345bbbca90
23:10:19.292            Filesystem     Type     1024-blocks      Used Availa... Show more
-07:00          # _time: 1616566219.292
```

Saving sample data

IN Tab: Displaying Samples on the Way IN to the Pipeline

The Preview pane offers two display options for events: Event and Table. (You can also download data as JSON or NDJSON, using the **Advanced Settings > Save** submenu from the top right.) Each format can be useful, depending on the type of data you are previewing.



```
1
2021-04-14 21:29:04.818 -07:00
α _raw: 2021-04-15 04:29:04,818,Event [Event=UpdateBi
        26257, properties={JMSCorrelationID=NA, JMSMe
        9:B3DD8244, orderType=ChangeESN, quotePriorit
# _time: 1618460944.818

2
2021-04-14 21:29:04.818 -07:00
α _raw: 2021-04-15 04:29:04,818,Event [Event=UpdateBi
        26258, proper
        2:E59DB306, c
# _time: 1618460944.8

3
2021-04-14 21:29:04.818
α _raw: 2021-04-15 04:29:04,818,Event [Event=UpdateBillingProvQuote, timestamp=15814
        26258, properties={JMSCorrelationID=NA, JMSMessageID=ID:ESP-PD.8083C4BFC28E
        1:1438F5D6, orderType=DisconnectAccount, quote... Show more
```

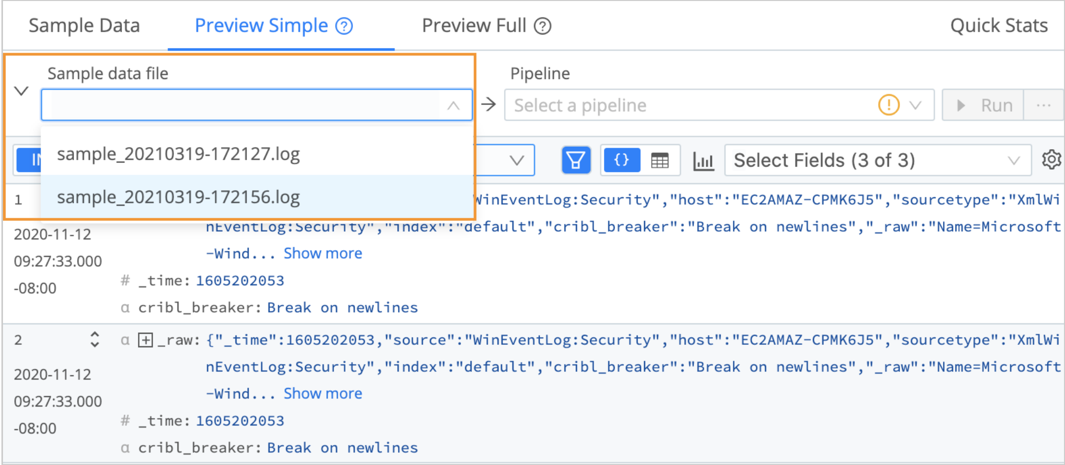
Event, Table, and Advanced options (composite screenshot)

CPU Profiling

The **Advanced Settings > CPU Profiling** submenu (accessible from the top right) offers **Timeout (sec)** and **Memory (MB)** limits. You can increase these controls' defaults to adjust for cases where very large data samples fail to load. For example, you might increase the **Timeout (sec)** to `30` and the **Memory (MB)** to `3048`.

Accessing and Managing Data Files

As you add more samples to your system, you can easily access them via the **Sample data file** drop-down.

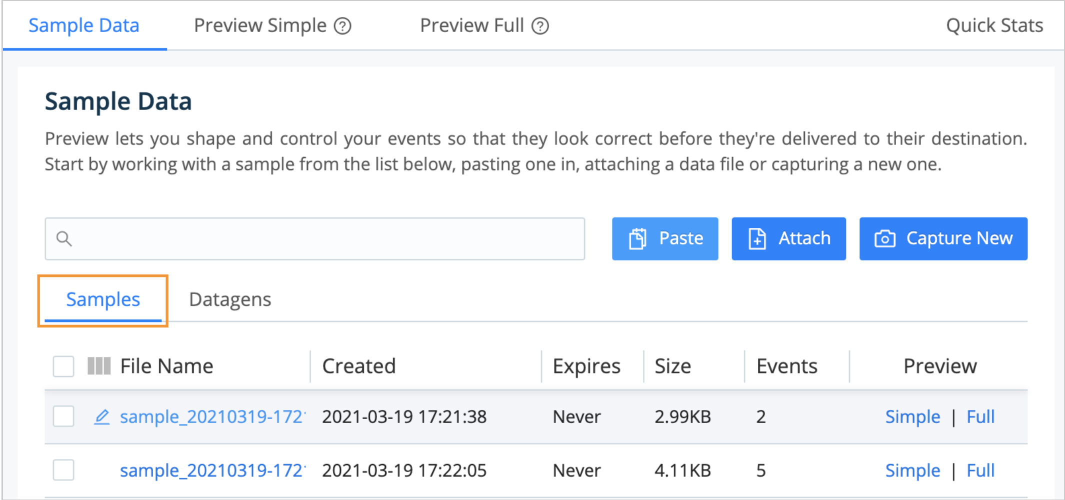


sample_20210319-172127.log
sample_20210319-172156.log

```
1                                                          WinEventLog:Security","host":"EC2AMAZ-CPMK6J5","sourcetype":"XmlWi
2020-11-12                    nEventLog:Security","index":"default","cribl_breaker":"Break on newlines","_raw":"Name=Microsoft
09:27:33.000                  -Wind... Show more
-08:00          # _time: 1605202053
                α cribl_breaker: Break on newlines
2               α ⊞ _raw: {"_time":1605202053,"source":"WinEventLog:Security","host":"EC2AMAZ-CPMK6J5","sourcetype":"XmlWi
2020-11-12                    nEventLog:Security","index":"default","cribl_breaker":"Break on newlines","_raw":"Name=Microsoft
09:27:33.000                  -Wind... Show more
-08:00          # _time: 1605202053
                α cribl_breaker: Break on newlines
```

Selecting an existing sample

You can also manage and modify sample files via the **Samples** tab highlighted below.



Sample Data

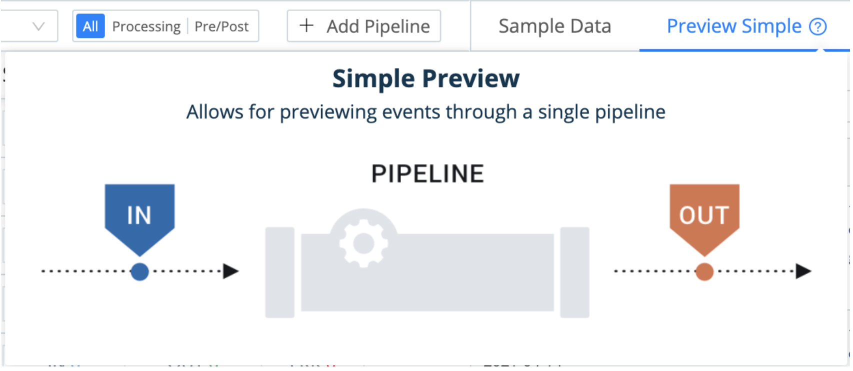
Preview lets you shape and control your events so that they look correct before they're delivered to their destination. Start by working with a sample from the list below, pasting one in, attaching a data file or capturing a new one.

□	File Name	Created	Expires	Size	Events	Preview
□	sample_20210319-172	2021-03-19 17:21:38	Never	2.99KB	2	Simple | Full
□	sample_20210319-172	2021-03-19 17:22:05	Never	4.11KB	5	Simple | Full

Managing sample files

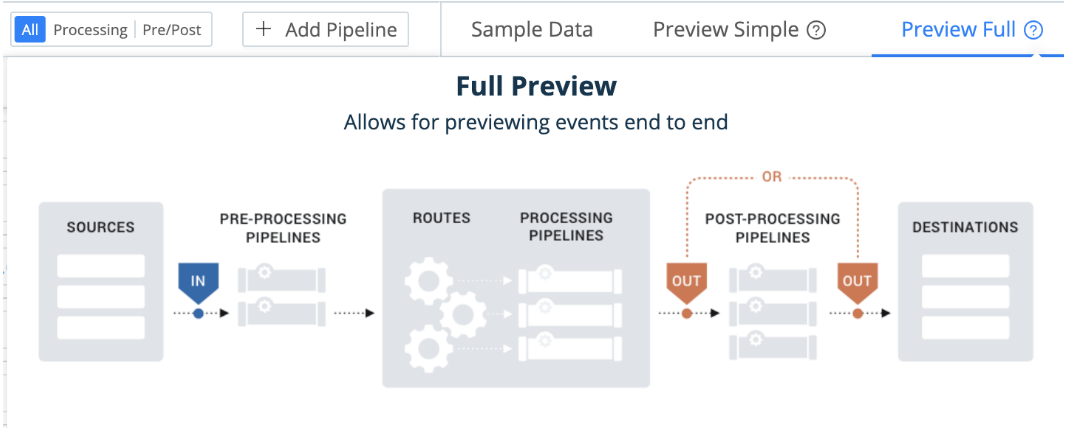
Simple Versus Full Preview

Click **Simple** or **Full** beside a file name to display its events in the Preview pane. The **Preview Simple** option enables you to view events on either the **IN** or the **OUT** (processed) side of a single Pipeline.



Preview Simple schematic

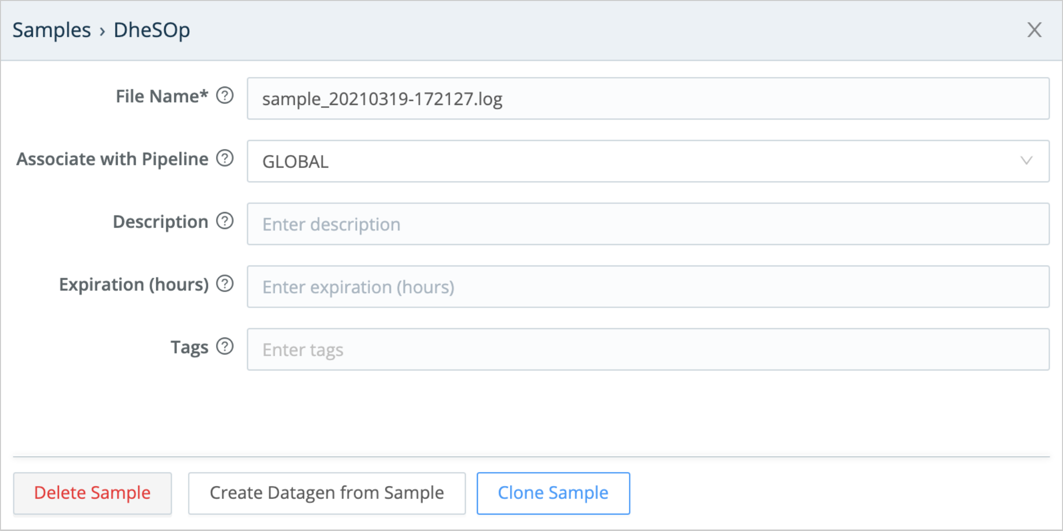
The **Preview Full** option gives you a choice of viewing events on the **OUT** side of either the processing or post-processing Pipeline. Selecting this option expands the Preview pane's upper controls to include an **Exit Point** drop-down, where you make this choice.



Preview Full schematic

Modifying Sample File Details

With the Preview pane's **Sample Data** tab selected, click directly on a file name to open the modal shown here, with options to clone the sample, save it as a datagen Source, delete it, associate it with a Pipeline, and set a description, expiration time, and tags.



Samples › DheSOp

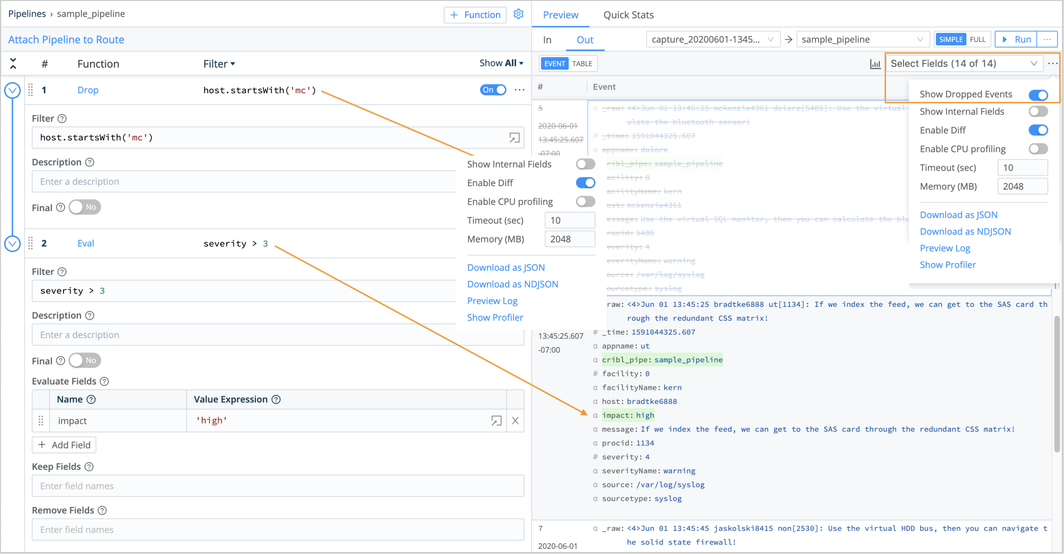
File Name*	sample_20210319-172127.log
Associate with Pipeline	GLOBAL
Description	Enter description
Expiration (hours)	Enter expiration (hours)
Tags	Enter tags

Options for modifying a sample

OUT Tab: Displaying Samples on the Way OUT of the Pipeline

As data traverses Functions in a Pipeline, events can be modified, and some might be dropped altogether. The **OUT** tab indicates changes using this color coding:

- **Dropped events**: When events are dropped, the **OUT** tab displays them as grayed-out text, with strikethrough. You can control their display using the **Advanced Settings** menu's **Show Dropped Events** slider.
- **Added fields**: When LogStream's processing adds new fields, these fields are highlighted green. You can control these fields' display using the **Select Fields** drop-down.
- **Redacted fields**: These fields are highlighted amber.
- **Deleted fields**: These fields are highlighted red.



Pipelines › sample_pipeline

Attach Pipeline to Route

#	Function	Filter
1	Drop	`host.startsWith('mc')`
2	Eval	`severity > 3`

Function 1 (Drop): Filter: `host.startsWith('mc')`; Description: Enter a description; Final: No

Function 2 (Eval): Filter: `severity > 3`; Description: Enter a description; Final: No

Evaluate Fields:

Name	Value Expression
impact	'high'

Keep Fields: Enter field names

Remove Fields: Enter field names

Preview — Out: capture_20200601-1345… → sample_pipeline (SIMPLE), Select Fields (14 of 14)

- Show Dropped Events: On
- Show Internal Fields: Off
- Enable Diff: On
- Enable CPU profiling: Off
- Timeout (sec): 10
- Memory (MB): 2048

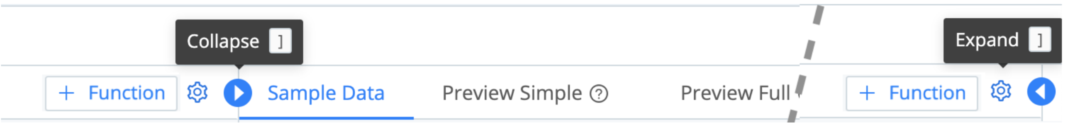
Download as JSON
Download as NDJSON
Preview Log
Show Profiler

```
5     ~~_raw: <4>Jun 01 13:45:25 mckenzie4361 dolore[5485]: Use the virtual-
           ulate the bluetooth sensor!~~
2020-06-01
13:45:25.607
-07:00
      ~~_time: 1591044325.607~~
      ~~appname: dolore~~
      ~~ribl_pipe: sample_pipeline~~
      ~~acility: 0~~
      ~~acilityName: kern~~
      ~~ost: mckenzie4361~~
      ~~essage: Use the virtual SQL monitor, then you can calculate the blu~~
      ~~rocid: 5485~~
      ~~everity: 4~~
      ~~everityName: warning~~
      ~~ource: /var/log/syslog~~
      ~~ourcetype: syslog~~
      raw: <4>Jun 01 13:45:25 bradtke6888 ut[1134]: If we index the feed, we can get to the SAS card th
           rough the redundant CSS matrix!
13:45:25.607
-07:00
      # _time: 1591044325.607
      α appname: ut
      α cribl_pipe: sample_pipeline
      # facility: 0
      α facilityName: kern
      α host: bradtke6888
      α impact: high
      α message: If we index the feed, we can get to the SAS card through the redundant CSS matrix!
      α procid: 1134
      # severity: 4
      α severityName: warning
      α source: /var/log/syslog
      α sourcetype: syslog

7     α _raw: <4>Jun 01 13:45:45 jaskolski8415 non[2530]: Use the virtual HDD bus, then you can navigate t
           he solid state firewall!
2020-06-01
```

Dropped and added fields in a Pipeline's output

Managing the Preview Pane

With the Routes or Pipelines page displayed in the left pane, hover over the pane divider (in the headers row) to display the Collapse/Expand toggle shown in the composite screenshot below.



Collapse / Expand toggle (composite)

Click Collapse to hide the Preview pane. This allows the Route or Pipeline configuration to expand to your browser's full width. (The Preview pane collapses automatically on narrow viewports.)

Click Expand at your browser's right edge to restore the split view. The pane divider will snap back to wherever you last dragged it.

Securing Data

Cribl LogStream can be used to encrypt sensitive data in real time, and to route the encrypted data to an end system. Decrypted retrieval can be implemented on a per-system basis. Currently, decryption is supported only when Splunk is the end system.

- Data Encryption
- Data Decryption

What's Next

- Encryption
- Decryption

Encryption

Encryption of Data in Motion

With Cribl LogStream, you can encrypt fields or patterns within events in real time, by using `C.Crypto.encrypt()` in a Mask function. The Mask function accepts multiple replacement rules and multiple fields to apply them to.

A **Match regex** defines the pattern of content to be replaced. The **Replace expression** is a JS expression or literal to replace matched content. The `C.Crypto.encrypt()` method can be used here to generate an encrypted string from a value passed to it.

> i C.Crypto.encrypt() Syntax
>
> (method) `Crypto.encrypt(value: any, keyclass: number, keyId?: string, defaultVal?: string): string`
> Encrypt the given value with the `keyId`, or with a `keyId` picked up automatically based on `keyclass`.
>
> @param {string | Buffer} `value` – what to encrypt.
> @param – `keyclass` – if `keyId` isn't specified, pick one at the given `keyclass`.
> @param – `keyId` - encryption keyId, takes precedence over `keyclass`.
> @param – `defaultVal` – what to return if encryption fails for any reason; if unspecified, the original value is returned.
> @returns – if encryption succeeds, the encrypted value; otherwise, `defaultVal` if specified; otherwise, `value`.

Encryption Keys

Symmetric keys can be configured 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.
- `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 LogStream 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`	`datasetA` Users: `U0, U1, U2`

Users: `U1`	

Configuring Keys with the CLI

When using the `local` key management system, encryption keys in Cribl LogStream 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.

> ℹ When installed as a Splunk app, `$CRIBL_HOME` is `$SPLUNK_HOME/etc/apps/cribl`.

Listing Keys

Keys are added and listed using the `keys` command:

```
$CRIBL_HOME/bin/cribl keys list -g <workerGroupID>
```

Sample Command Output

```
keyId  algorithm    keyclass  kms    created         expires     useIV
-----------------------------------------------------------------------
1      aes-256-cbc  0         local  1544906269.316  0           false
2      aes-256-cbc  1         local  1544906272.452  0           false
3      aes-256-cbc  2         local  1544906275.948  1545906275  true
4      aes-256-cbc  3         local  1544906278.026  0           false
```

Adding Keys

Displaying `--help`:

```
$CRIBL_HOME/bin/cribl keys add --help
```

Sample Command Output

```
Add encryption keys
Usage: [options] [args]

Options:
[-c <keyclass>] - key class to set for the key
[-k <kms>]      - KMS to use, must be configured, see cribl.yml
[-e <expires>]  - expiration time, epoch time
[-i]            - use an initialization vector
 -g <group>     - Group ID
```

Adding a key to keyclass `1`, with no expiration date, on the `default` Worker Group:

```
$CRIBL_HOME/bin/cribl keys add -c 1 -i -g default
```

Sample Command Output

```
Adding key: success. Key count=1
```

(You would use the same syntax to reference a non- `default` Worker Group by its name.)
?
Listing keys to verify key generation:

```
$CRIBL_HOME/bin/cribl keys list -g default
```

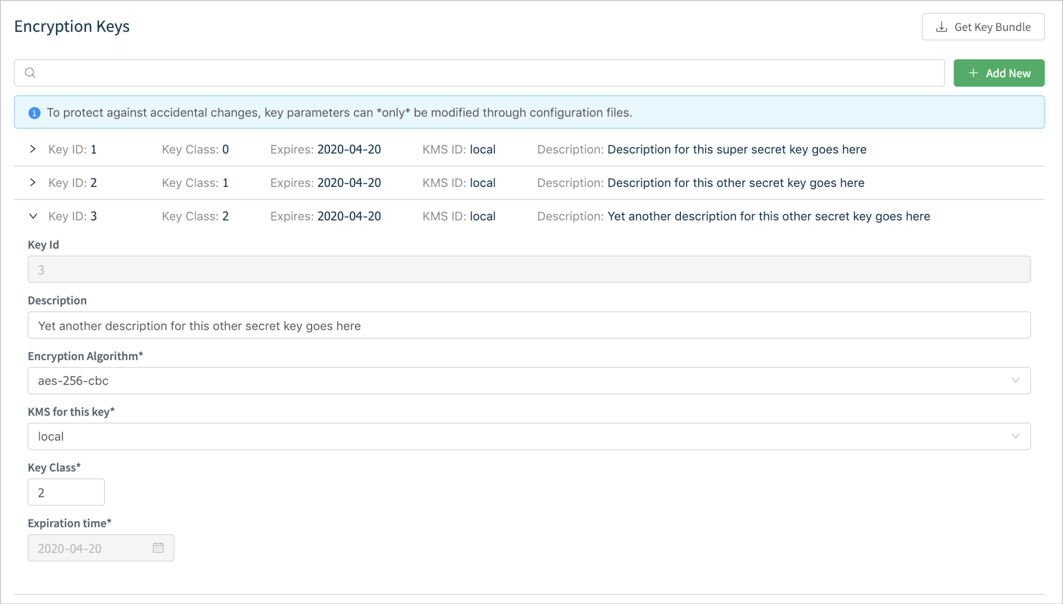
Sample Command Output

```
keyId  algorithm    keyclass  kms    created         expires     useIV
-----------------------------------------------------------------------
1      aes-256-cbc  1         local  1545243364.342  0           true
```

Configuring Keys with the UI

In a single-instance deployment, you can access the key management interface through global ⚙ **Settings** (lower left) > **Security > Encryption Keys**. In a distributed deployment, for each Group, select **Groups** > `<group-name>` > **Settings** > **Security** > **Encryption Keys**.

Here, you can list and add new keys. To protect against accidental changes, a key's parameters, once saved, can be edited only through configuration files.



To protect against accidental changes, key parameters can *only* be modified through configuration files.

Key ID: 1 Key Class: 0 Expires: 2020-04-20 KMS ID: local Description: Description for this super secret key goes here

Key ID: 2 Key Class: 1 Expires: 2020-04-20 KMS ID: local Description: Description for this other secret key goes here

Key ID: 3 Key Class: 2 Expires: 2020-04-20 KMS ID: local Description: Yet another description for this other secret key goes here

Key Id: 3

Description: Yet another description for this other secret key goes here

Encryption Algorithm*: aes-256-cbc

KMS for this key*: local

Key Class*: 2

Expiration time*: 2020-04-20

List or create keys through LogStream's UI

Sync `auth/(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: `$CRIBL_HOME/local/cribl/auth/` .
- In a distributed deployment, the `cribl.secret` and `keys.json` files reside on the Leader Node in: `$CRIBL_HOME/groups/<group-name>/local/cribl/auth/` .
- When using the UI, you can download these files by clicking the **Get Key Bundle** button.

Sync/copy these files over to their counterparts on the Search Head/decrypting side.

Modifying Keys

When you update keys by editing the `keys.json` file, you must add them back to to the directories above (respectively, on a single instance or on a distributed deployment's Leader Node).

Decryption

Decryption of Data

Currently, Cribl LogStream supports decryption only when Splunk is the end system. In Splunk, decryption is available to users of any role with permissions to run the `decrypt` command that ships with Cribl App for Splunk. Further restrictions can be applied with Splunk **capabilities**. This page provides details.

Decrypting in Splunk

Decryption in Splunk is implemented via a custom command called `decrypt`. 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 `decrypt`.

> i 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 – `decrypt` is scoped to run locally on the installed search head.

Restricting Access with Splunk Capabilities

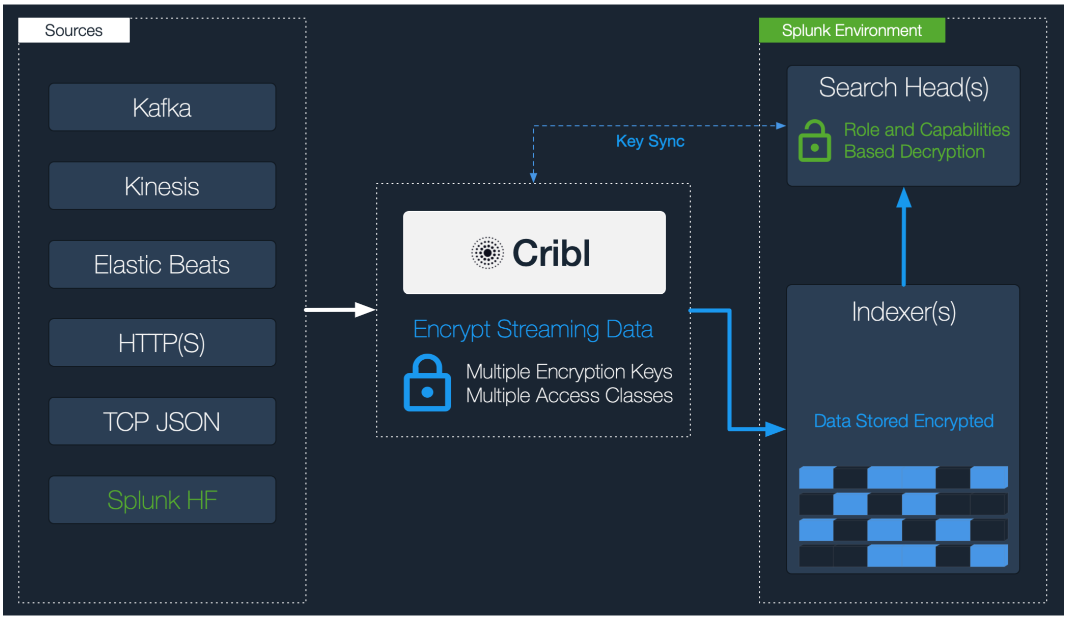
In Splunk, capability names should follow the format `cribl_keyclass_N`, where `N` is the Cribl Key Class. For example, a role with capability `cribl_keyclass_1` has access to all key IDs associated with key class `1`.

Capability Name	Corresponding Cribl Key Class
`cribl_keyclass_1`	`1`
`cribl_keyclass_2`	`2`

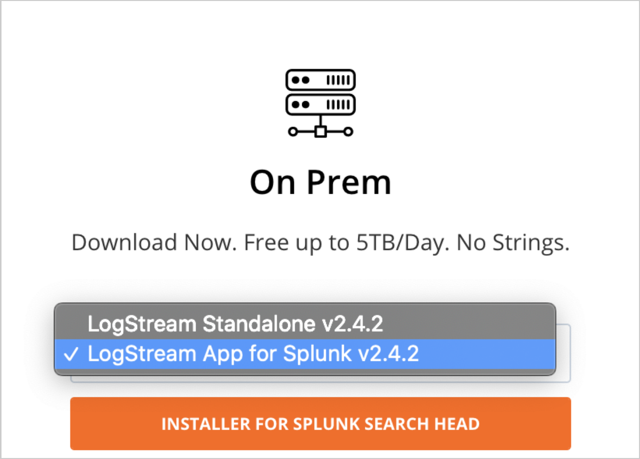
… `cribl_keyclass_N`	… `N`

Configuring Splunk Search Head to Decrypt Data

You set up decryption in Splunk according to this schematic:



1. Download the Cribl/LogStream App for Splunk from Cribl's Download LogStream page: In the **On Prem** section, select the Splunk app from the drop-down list, as shown. Clicking the orange button downloads a file named: `cribl-splunk-app-<version-#>-<hash-#>-linux-x64.tgz` .



On Prem

Download Now. Free up to 5TB/Day. No Strings.

LogStream Standalone v2.4.2
✓ LogStream App for Splunk v2.4.2
INSTALLER FOR SPLUNK SEARCH HEAD

Downloading Cribl's Splunk app

2. To install the Cribl/LogStream App for Splunk on your search head, untar the package into your `$SPLUNK_HOME/etc/apps` directory.

As of LogStream v1.7, 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 `decrypt` 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. Sync `auth/(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 LogStream's UI, you can download these files by clicking the **Get Key Bundle** button.

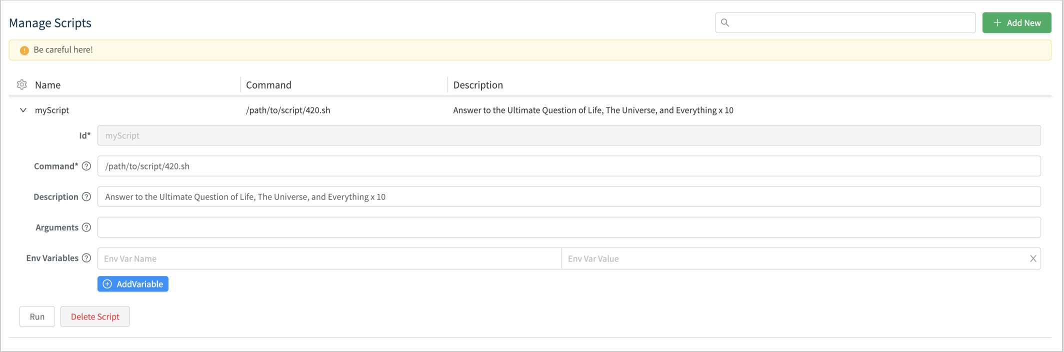
Sync/copy these files over to their counterparts on the search head (decryption side). In a non-Splunk integration, you would copy these assets to wherever decryption will take place.

> ℹ Modifying Keys
>
> When you update keys by editing the `keys.json` file, you must add them back to to the directories above (respectively, on a single instance or on a distributed deployment's Leader Node).

Scripts

Admins can run scripts (e.g., shell scripts) from within Cribl LogStream by configuring and executing them at global ⚙ **Settings** (lower left) > **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 LogStream configs are updated.

> ⚠ With Great Power Comes Great Responsibility!
>
> Scripts will allow you to execute almost anything on the system where Cribl LogStream is running. Make sure you understand the impact of what you're executing before you do so!



Manage Scripts

Be careful here!

Name	Command	Description
myScript	/path/to/script/420.sh	Answer to the Ultimate Question of Life, The Universe, and Everything x 10

Id*	myScript
Command*	/path/to/script/420.sh
Description	Answer to the Ultimate Question of Life, The Universe, and Everything x 10
Arguments	
Env Variables	Env Var Name / Env Var Value

Settings > Manage Scripts page

The Manage Scripts page provides the following tields:

- **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.

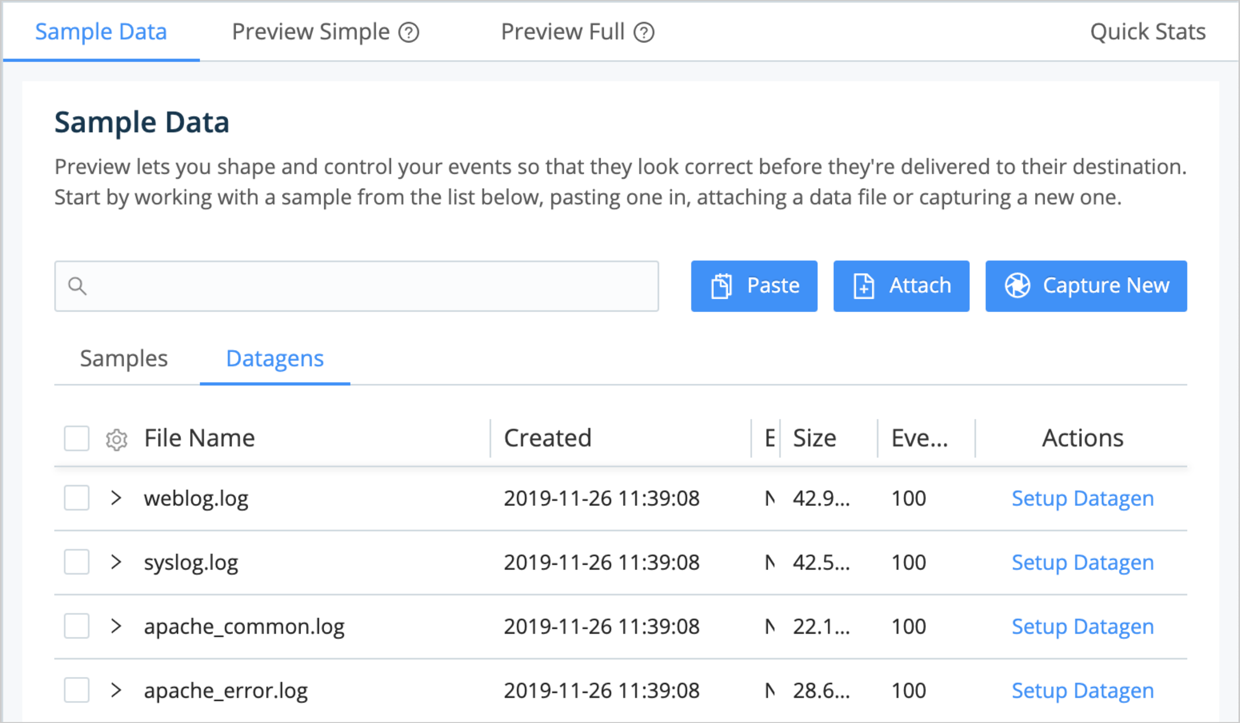
> ℹ Scripts in Distributed Deployments
>
> - Scripts can be deployed from Leader Node, but can be **run** only locally from each Worker Node.
> - If the Script command is referencing a file (e.g., `420.sh`), that file must exist on the Cribl LogStream instance. In other words, the Script management interface cannot be used to upload or manage script files.

Using Datagens

Data generators for testing and troubleshooting

Cribl LogStream'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.



Sample Data

Preview lets you shape and control your events so that they look correct before they're delivered to their destination. Start by working with a sample from the list below, pasting one in, attaching a data file or capturing a new one.

□	File Name	Created	E	Size	Eve...	Actions
□	weblog.log	2019-11-26 11:39:08	N	42.9...	100	Setup Datagen
□	syslog.log	2019-11-26 11:39:08	N	42.5...	100	Setup Datagen
□	apache_common.log	2019-11-26 11:39:08	N	22.1...	100	Setup Datagen
□	apache_error.log	2019-11-26 11:39:08	N	28.6...	100	Setup Datagen

Preview pane – add samples via paste, attach/upload file, or live capture

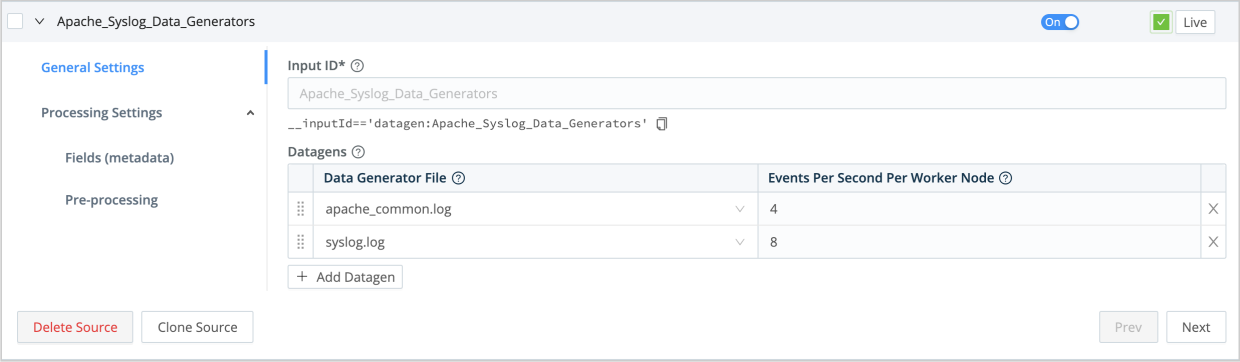
As outlined in the following tutorial: Once you've created a template, you can configure a Datagen Source to use the template to generate real-time data at a given EPS (events per second) rate.

Enabling a Datagen

To see how Datagens work, start by enabling a pair of LogStream's out-of-the-box generators:

Navigate to **Sources** > **Datagens** and click **+ Add New**.

Select a Data Generator File (e.g., `apache_common.log`) and set it at 4 EPS/worker process. Select another Data Generator File (e.g., `syslog.log`) and set it at 8 EPS/worker process. Hit **Save**.



Input ID*

Apache_Syslog_Data_Generators

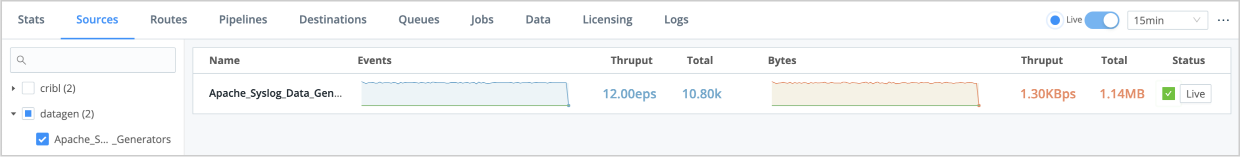
`__inputId=='datagen:Apache_Syslog_Data_Generators'`

Datagens

Data Generator File	Events Per Second Per Worker Node
apache_common.log	4
syslog.log	8

Selecting Datagens files and event rates

On the **Monitoring** page, under **Sources**, search for `datagen` and confirm that the Source is generating data.



Name	Events	Thruput	Total	Bytes	Thruput	Total	Status
Apache_Syslog_Data_Gen...		12.00eps	10.80k		1.30KBps	1.14MB	☑ Live

Creating a Datagen Template from a Sample File

To convert a sample into a template:

Go to **Preview** > **Paste a Sample**, and add a sample like the AWS VPC Flow logs below:

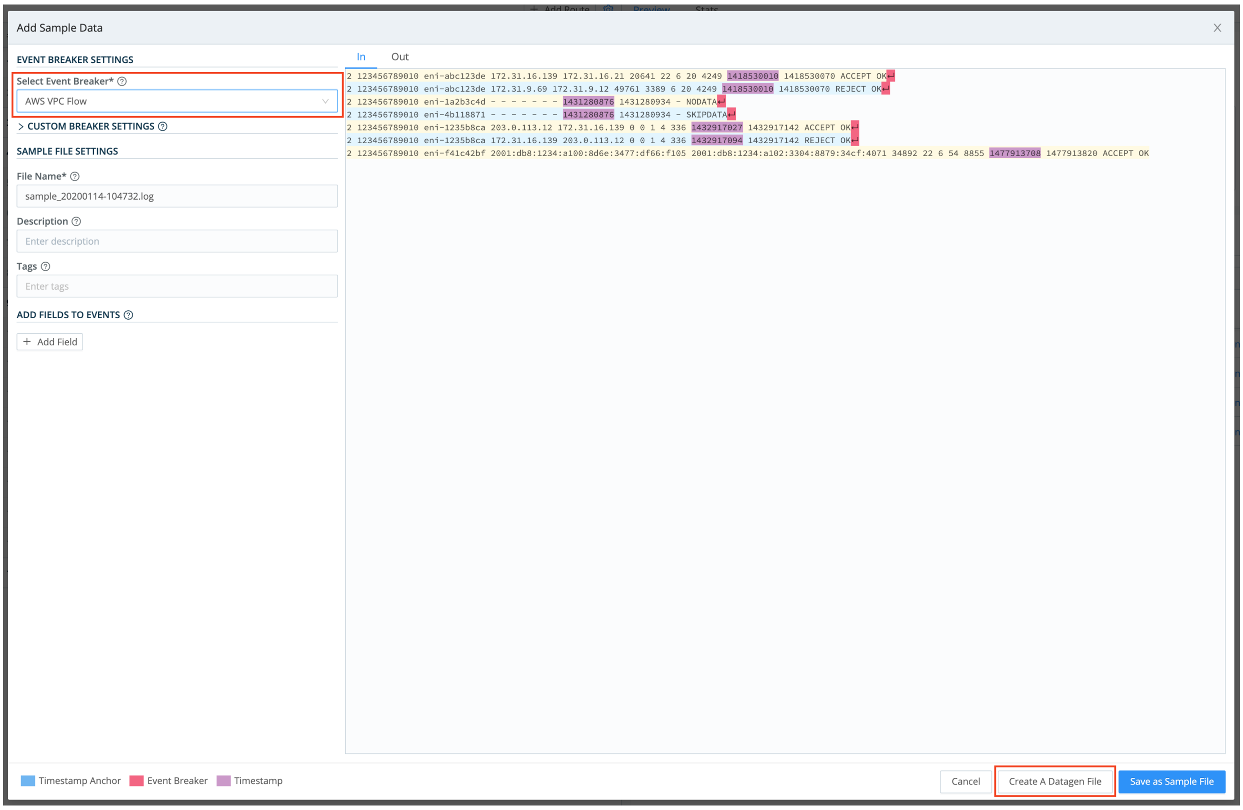
Sample VPC Flow Logs

```
2 123456789010 eni-abc123de 172.31.16.139 172.31.16.21 20641 22 6 20 4249 1418530010 1418530070
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1418530010 1418530070 F
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 1432917142 ACCEPT
2 123456789010 eni-1235b8ca 172.31.16.139 203.0.113.12 0 0 1 4 336 1432917094 1432917142 REJECT
2 123456789010 eni-f41c42bf 2001:db8:1234:a100:8d6e:3477:df66:f105 2001:db8:1234:a102:3304:8879:
```

From the **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, click **Create a Datagen File**.



```
2 123456789010 eni-abc123de 172.31.16.139 172.31.16.21 20641 22 6 20 4249 1418530010 1418530070 ACCEPT OK
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1418530010 1418530070 REJECT OK
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 1432917142 ACCEPT OK
2 123456789010 eni-1235b8ca 172.31.16.139 203.0.113.12 0 0 1 4 336 1432917094 1432917142 REJECT OK
2 123456789010 eni-f41c42bf 2001:db8:1234:a100:8d6e:3477:df66:f105 2001:db8:1234:a102:3304:8879:34cf:4071 34892 22 6 54 8855 1477913708 1477913820 ACCEPT OK
```

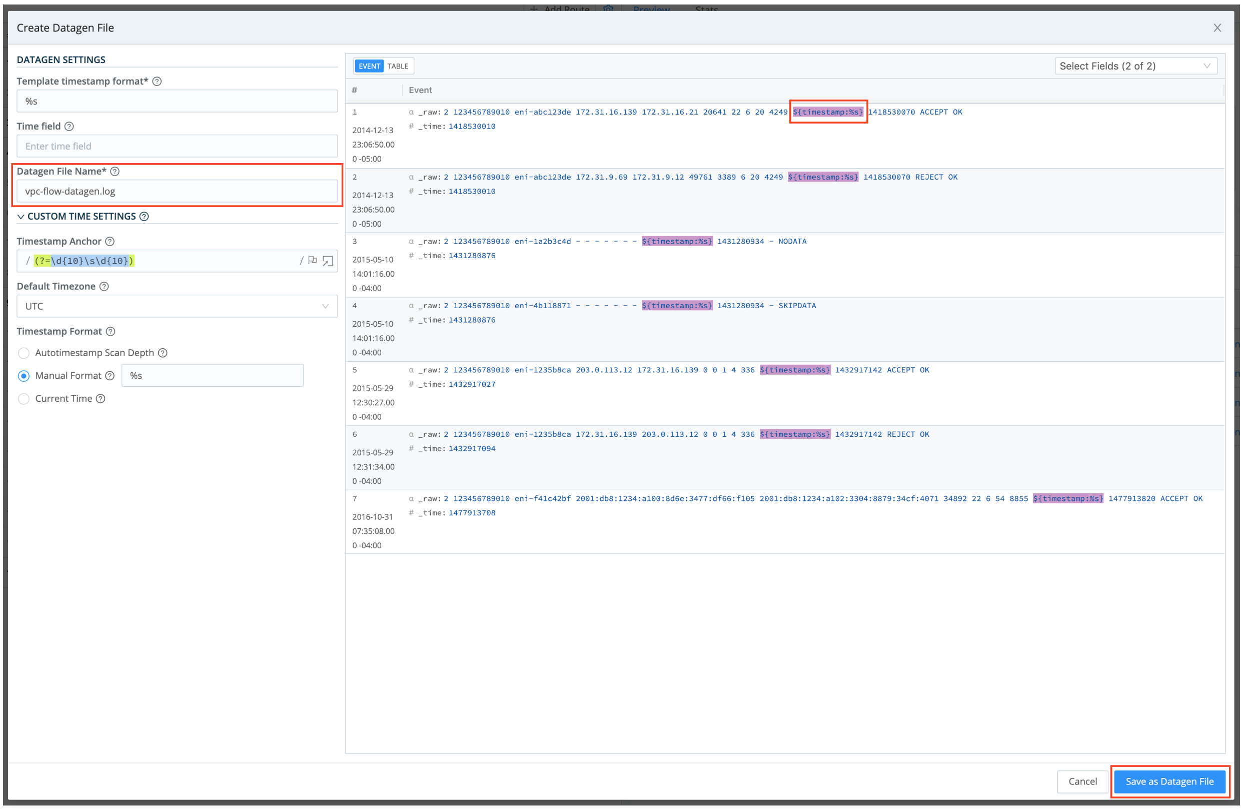
Creating a Datagen template

On the resulting **Create Datagen File** screen:

- Enter a file name, e.g.: `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 (i.e., the epoch).

Once you've verified these results, click **Save as Datagen File**.



DATAGEN SETTINGS

Template timestamp format*: %s

Time field: Enter time field

Datagen File Name*: vpc-flow-datagen.log

CUSTOM TIME SETTINGS

Timestamp Anchor: / (?=\d{10}\s\d{10}) /

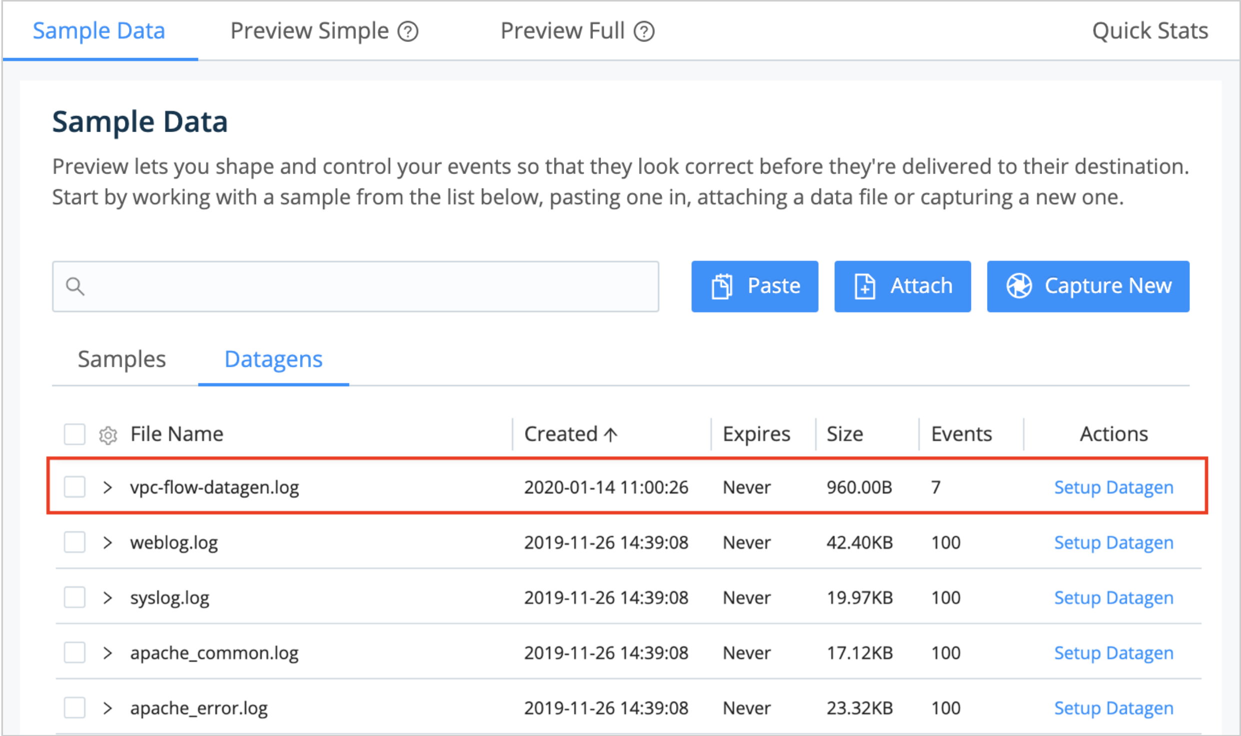
Default Timezone: UTC

Timestamp Format:
○ Autotimestamp Scan Depth
● Manual Format: %s
○ Current Time

```
#  Event
1  α _raw: 2 123456789010 eni-abc123de 172.31.16.139 172.31.16.21 20641 22 6 20 4249 ${timestamp:%s} 1418530070 ACCEPT OK
   # _time: 1418530010
2014-12-13
23:06:50.00
0 -05:00
2  α _raw: 2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 ${timestamp:%s} 1418530070 REJECT OK
   # _time: 1418530010
2014-12-13
23:06:50.00
0 -05:00
3  α _raw: 2 123456789010 eni-1a2b3c4d - - - - - - - ${timestamp:%s} 1431280934 - NODATA
   # _time: 1431280876
2015-05-10
14:01:16.00
0 -04:00
4  α _raw: 2 123456789010 eni-4b118871 - - - - - - - ${timestamp:%s} 1431280934 - SKIPDATA
   # _time: 1431280876
2015-05-10
14:01:16.00
0 -04:00
5  α _raw: 2 123456789010 eni-1235b8ca 203.0.113.12 172.31.16.139 0 0 1 4 336 ${timestamp:%s} 1432917142 ACCEPT OK
   # _time: 1432917027
2015-05-29
12:30:27.00
0 -04:00
6  α _raw: 2 123456789010 eni-1235b8ca 172.31.16.139 203.0.113.12 0 0 1 4 336 ${timestamp:%s} 1432917142 REJECT OK
   # _time: 1432917094
2015-05-29
12:31:34.00
0 -04:00
7  α _raw: 2 123456789010 eni-f41c42bf 2001:db8:1234:a100:8d6e:3477:df66:f105 2001:db8:1234:a102:3304:8879:34cf:4071 34892 22 6 54 8855 ${timestamp:%s} 1477913820 ACCEPT OK
   # _time: 1477913708
2016-10-31
07:35:08.00
0 -04:00
```

Saving a named Datagen template

To confirm that the Datagen file has been created, check **Preview** > **Datagens**.



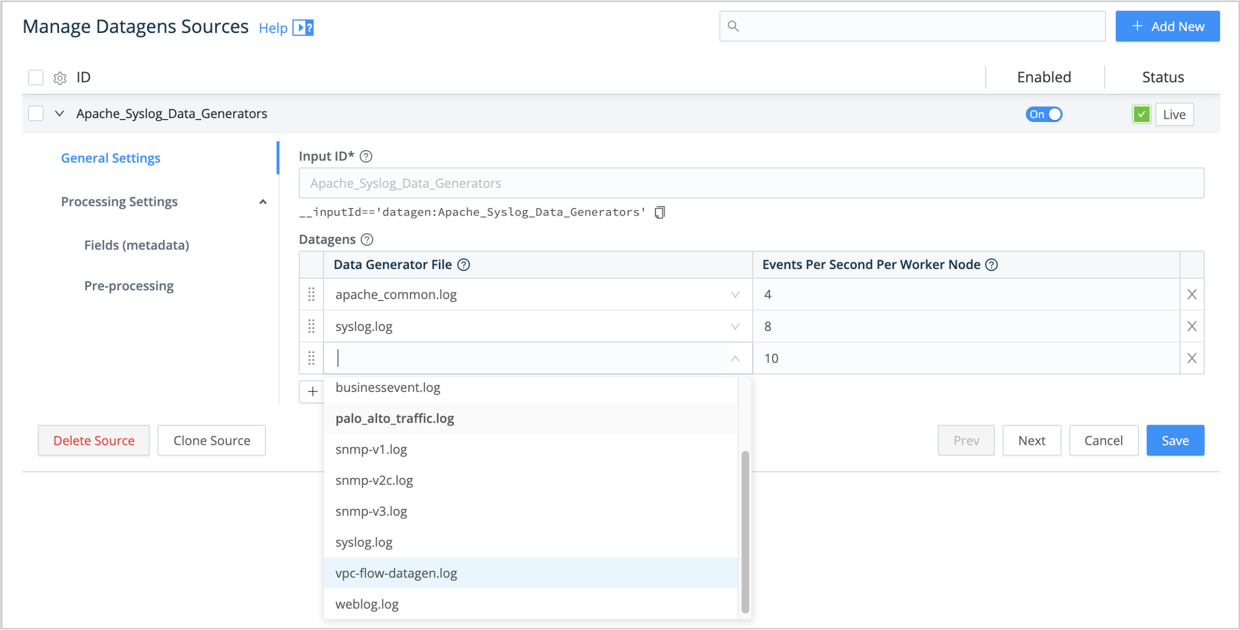
Sample Data

Preview lets you shape and control your events so that they look correct before they're delivered to their destination. Start by working with a sample from the list below, pasting one in, attaching a data file or capturing a new one.

File Name	Created ↑	Expires	Size	Events	Actions
vpc-flow-datagen.log	2020-01-14 11:00:26	Never	960.00B	7	Setup Datagen
weblog.log	2019-11-26 14:39:08	Never	42.40KB	100	Setup Datagen
syslog.log	2019-11-26 14:39:08	Never	19.97KB	100	Setup Datagen
apache_common.log	2019-11-26 14:39:08	Never	17.12KB	100	Setup Datagen
apache_error.log	2019-11-26 14:39:08	Never	23.32KB	100	Setup Datagen

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.



Apache_Syslog_Data_Generators

Input ID*

Apache_Syslog_Data_Generators

```
__inputId=='datagen:Apache_Syslog_Data_Generators'
```

Datagens

Data Generator File	Events Per Second Per Worker Node
apache_common.log	4
syslog.log	8
	10

Adding new template file to Datagens Source

CLI Reference

Command line interface basics

In addition to starting and stopping the Cribl LogStream server, LogStream's command line interface enables you to initiate many configuration and administrative tasks directly from your terminal.

Command Syntax

To execute CLI commands, the basic syntax is:

```
cd $CRIBL_HOME/bin
./cribl <command> <sub-command> <options> <arguments>
```

Avoiding Surprises

Immediate Execution

As indicated in the sample output below, some commands take effect immediately.

Commands that require further input will echo the sub-commands, options, and arguments they expect.

Persistent Volumes

If you start LogStream with the `CRIBL_VOLUME_DIR` variable, all subsequent CLI commands should have this variable defined. Otherwise, those commands will apply LogStream's 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
```

Note that `$CRIBL_VOLUME_DIR`, when set, overrides `$CRIBL_HOME`.

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>`.

`help`

Displays help (commands list).

```
Cribl LogStream - N.n.n-<build no.>
Usage: [sub-command] [options] [args]

Commands:
help            - Display help
mode-master     - Configure to a Leader instance
mode-single     - Configure to a Single instance
mode-worker     - Configure to a Worker instance
reload          - Reload Cribl LogStream
restart         - Restart Cribl LogStream
start           - Start Cribl LogStream
status          - Status of Cribl LogStream
stop            - Stop Cribl LogStream
version         - Print Cribl LogStream version and installation type

auth            - Cribl LogStream Auth
boot-start      - Enable/Disable Cribl LogStream boot-start
diag            - Manage diagnostics bundles
groups          - Manage Worker Groups
keys            - Manage encryption keys
mode-searchhead - Configure Cribl LogStream to run on a Splunk Search Head
nc              - Listen on a port for traffic and output stats and data
node            - Execute a JavaScript file
pipe            - Feed stdin to a pipeline
splunk-decrypt  - Splunk decrypt search command
task            - Run Cribl LogStream task
vars            - Manage global variables
```

> ℹ As of version 3.0, LogStream'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.

`mode-master`

Configures Cribl LogStream as a Leader instance.

Options

```
[-H <host>]               - Host (defaults to 0.0.0.0).
[-p <port>]               - Port (defaults to 4200).
[-n <certName>]           - Name of saved certificate.
[-k <privKeyPath>]        - Server path containing the private key (in PEM format) to use. Can ref
[-c <certPath>]           - Server path containing certificates (in PEM format) to use. Can refere
[-u <authToken>]          - Optional authentication token to include as part of the connection hea
[-i <ipWhitelistRegex>] - Regex matching IP addresses that are allowed to establish a connectio
```

Sample Response

```
Settings updated.
You will need to restart LogStream before your changes take full effect.
```

`mode-single`

Configures Cribl LogStream as a single-instance deployment.

Sample Response

```
Settings updated.
You will need to restart LogStream before your changes take full effect.
```

`mode-worker`

Configures Cribl LogStream as a Worker instance.

Usage

`./cribl mode-worker -H <host> -p <port>`

The `-H <host> -p <port>` parameters are required.

Options

```
-H <host>          - Leader Node's Hostname or IP address.
-p <port>          - Leader Node's cluster communications port (defaults to 4200).
[-n <certName>]    - Name of saved certificate.
[-k <privKeyPath>] - Server path containing the private key (in PEM format) to use. Can referenc
[-c <certPath>]    - Server path containing certificates (in PEM format) to use. Can reference S
[-u <authToken>]   - Authentication token to include as part of the connection header. By defaul
[-e <envRegex>]    - Regex that selects environment variables to report to Leader.
[-t <tags>]        - Tag values to report to Leader.
[-g <group>]       - Worker Group to report to Leader.
```

Sample Response

```
Settings updated.
You will need to restart LogStream before your changes take full effect.
```

`reload`

Reloads Cribl LogStream. Executes immediately.

```
Reload request submitted to Cribl LogStream
```

restart

Restarts Cribl LogStream. Executes immediately.

> ⚠ Executing this command cancels any running collection jobs.

```
Stopping Cribl LogStream, process 56572
............
Cribl LogStream is not running
Starting Cribl LogStream...
..
Cribl LogStream started with pid 57233
API Server is available at http://192.168.0.100:9000
```

start

Starts Cribl LogStream. Executes immediately. Upon first run, echoes LogStream's default login credentials.

```
Starting Cribl LogStream...
..
Cribl LogStream started with pid 57279
API Server is available at http://192.168.0.100:9000
```

status

Displays status of Cribl LogStream, including the API Server address, instance's mode (Leader or Worker), process ID, and GUID (fictitious example below). Executes immediately.

```
Cribl LogStream Status

Address: http://192.168.0.100:9000
Mode: worker
Status: Up
Software Version: 42.0-7f4c190a
Leader/Master: localhost:4200
PID: 3859
GUID: 76-ea411263a64b9-e419daee4-ef-dd2e2f
```

stop

Stops Cribl LogStream. Executes immediately.

> ⚠ Executing this command cancels any running collection jobs.

```
Stopping Cribl LogStream, process 57233
...........
Cribl LogStream is not running
```

version

Displays Cribl LogStream version and installation type. Executes immediately.

```
Version: 2.2-0####x##
Installation type: standalone
```

> The `version` command echoes `standalone` for both single-instance and distributed deployments. This simply confirms that you're running a freestanding Cribl LogStream server, not the Cribl App for Splunk.

auth

Log into or out of Cribl LogStream.

```
Commands:
login  - Log in to Cribl LogStream, args:
  [-h <host>]     - Host URL (e.g. http://localhost:9000)
  [-u <username>] - Username
  [-p <password>] - Password
  [-f <file>]     - File with credentials
logout - Log out from Cribl LogStream
```

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>
```

> i All `-h` and `host` arguments are optional, provided that the API host and port are listed in the `cribl.yml` file's `api:` section

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>
```

boot-start

Enables or disables Cribl LogStream boot-start.

```
Usage: [sub-command] [options] [args]

Commands:
disable - Disable Cribl LogStream boot-start, args:
  [-m <manager>]   - Init manager (systemd|initd)
  [-c <configDir>] - Config directory for the init manager
enable  - Enable Cribl LogStream boot-start, args:
  [-m <manager>]   - Init manager (systemd|initd)
  [-u <user>]      - User to run Cribl LogStream as
  [-c <configDir>] - Config directory for the init manager
```

diag

Manages diagnostic bundles.

```
create - Creates diagnostic bundle for Cribl LogStream

list   - List existing Cribl LogStream diagnostic bundles

send   - Send LogStream 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)
```

groups

Manages Worker Groups.

```
Usage: [sub-command] [options] [args]

Commands:
commit        - Commit, args:
  [-g <group>]   - Group ID
  [-m <message>] - Commit message
commit-deploy - Commit & Deploy, args:
   -g <group>    - Group ID
  [-m <message>] - Commit message
deploy        - Deploy, args:
   -g <group>    - Group ID
  [-v <version>] - Deploy version
list          - List Worker Groups
```

keys

Manages encryption keys. You must append the `-g <group>` argument to specify a Worker Group. As a fallback, append the argument `-g default`, e.g.:

```
./cribl keys list -g default
```

```
Usage: [sub-command] [options] [args]

Commands:
add  - Add encryption keys, args:
   -g <group>  - Group ID
  [-c <keyclass>] - key class to set for the key
  [-k <kms>]      - KMS to use, must be configured, see cribl.yml
  [-e <expires>]  - expiration time, epoch time
  [-i ]           - use an initialization vector
list - List encryption keys
   -g <group> - Group ID
```

mode-searchhead

Configures Cribl LogStream to run on a Splunk Search Head.

nc

Listens on a port for traffic, and outputs stats and data. (Netcat-like utility.)

```
Usage: [options] [args]

Options:
 -p <port>           - Port to listen on
[-s <statsInterval>] - Stats output interval (ms), use 0 to disable
[-u]                 - Listen on UDP port instead
[-o]                 - Output received data to stdout
[-t <rate> <units>]  - Throttle rate, in <units>/second, where <units> can be KB, MB, GB, or TB
```

node

Executes a JavaScript file. Displays a command prompt for path/filename input, as shown here:

```
>
```

pipe

Feeds stdin to a pipeline. Examples:

```
cat sample.log |  ./cribl pipe -p <pipelineName>
cat sample.log |  ./cribl pipe -p <pipelineName> 2>/dev/null
```

scope

Greps your apps by the syscalls. Executes immediately.

splunk-decrypt

Splunk decrypt search command. Executes immediately.

task

Runs a Cribl LogStream task. Requires definitions for the `dir` , `executor` , and `path` properties.

vars

Manages LogStream Global Variables.

```
Usage: [sub-command] [options] [args]

Commands:
add    - Add global variable, args:
   -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
get    - List encryption keys, args:
  [-i <id>]    - Global variable ID
  [-g <group>] - Group ID
remove - Remove global variable, args:
   -i <id>     - Global variable ID
  [-g <group>] - Group ID
update - Update global variable, args:
   -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
```

EXPRESSION REFERENCE

Introduction to Expression Syntax

As data travels through a Cribl LogStream Pipeline, it is operated on by a series of Functions. Functions are fundamentally JavaScript code.

Functions that ship with Cribl LogStream are configurable via a set of inputs. Some of these configuration options are literals, such as field names, and others can be JavaScript expressions.

Expressions are **valid units** of code that resolve to a value. Every syntactically valid expression resolves to some value, but conceptually, there are two types of expressions: those that **assign** value to a variable (a.k.a., with side effects), and those that **evaluate** to a value.

Assigning a value	Evaluating to a value

`x = 42`
`newFoo = foo.slice(30)`
	`(Math.random() * 42)`
	`3 + 4`
	`'foobar'`
	`'42'`

Filters and Value Expressions

Filters

Filters are used in Routes to select a stream of the data flow, and in Functions to scope or narrow down the applicability of a Function. Filters are expressions that **must** evaluate to either `true` (or truthy) or `false` (or falsy). Keep this in mind when creating Routes or Functions. For example:

- `sourcetype=='access_combined' && host.startsWith('web')`
- `source.endsWith('.log') || sourcetype=='aws:cloudwatchlogs:vpcflow'`

Truthy	Falsy
`true`	`false`
`42`	`null`
`-42`	`undefined`
`3.14`	`0`
`"foo"`	`NaN`
`Infinity`	`''`
`-Infinity`	`""`

Value Expressions

Value expressions are typically used in Functions to assign a value – for example, to a new field. For example:

- `Math.floor(_time/3600)`
- `source.replace(/.{3}/, 'XXX')`

Best Practices for Creating 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` evaluates 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 – e.g., `@timestamp` or `user-agent` or `kubernetes.namespace_name` – you can access them using `__e['<field-name-here>']` . (Note the single quotes.) More details here.

In any other place where the field is referenced – e.g., in the Eval function's field names – you should use a single-quoted literal, of the form: `'<field-name-here>'` .

Wildcard Lists

Wildcard Lists are used throughout the product, especially in various Functions, such as Eval, Mask, Publish Metrics, Parser, etc.

Wildcard Lists, as their name implies, 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.

Wildcard Lists are order-sensitive **only** when negated terms are used. This allows for implementing any combination of allowlists and blocklists.

For example:

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**.

> ⚠ You cannot use wildcards to target LogStream internal fields that start with `__` (double underscore). You must specify these fields individually. For example, `__foobartab` cannot be removed by specifying `__foo*` .

Cribl Expressions

Native Cribl LogStream function 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 Fields with Non-Alphanumeric Characters.

C.Crypto – Data Encryption and Decryption Functions

`C.Crypto.decrypt`
(method) `Crypto.decrypt(value: string): string`
Decrypt all occurrences of ciphers in the given value. Instances that cannot be decrypted (for any reason) are left intact.
@param – `value` – string in which to look for ciphers
@returns – `value` with ciphers decrypted

`C.Crypto.encrypt`
(method) `Crypto.encrypt(value: any, keyclass: number, keyId?: string, defaultVal?: string): string`
Encrypt the given value with the `keyId`, or with a `keyId` picked up automatically based on `keyclass`.

@param {string | Buffer} `value` – what to encrypt.
@param – `keyclass` – if `keyId` isn't specified, pick one at the given `keyclass`.

@param – `keyId` - encryption keyId, takes precedence over `keyclass` .
@param – `defaultVal` – what to return if encryption fails for any reason; if unspecified, the original value is returned.
@returns – if encryption succeeds, the encrypted value; otherwise, `defaultVal` if specified; otherwise, `value` .

C.Decode – Data Decoding Functions

`C.Decode.base64`
(method) `Decode.base64(val: string, resultEnc?: string): any`
Performs base64 decoding of the given string. Returns a string or Buffer, depending on the `resultEnc` value, which defaults to `'utf8'` .
@param – `val` – value to base64-decode
@param – `resultEnc` – encoding to use to convert the binary data to a string. Defaults to `'utf8'` . Use `'utf8-valid'` to validate that result is valid UTF8; use `'buffer'` if you need the binary data in a Buffer.

`C.Decode.gzip`
(method) `Decode.gzip(value: any, encoding?: string): string`
Gunzip the supplied value.
@param – `value` – the value to gunzip.
@param – `encoding` – 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.

`C.Decode.hex`
(method) `Decode.hex(val: string): number`
Performs hex to number conversion. (Returns `NaN` if value cannot be converted to a number.)
@param – `val` – hex string to parse to a number (e.g., "0xcafe").

`C.Decode.uri`
(method) `Decode.uri(val: string): string`
Performs URI-decoding of the given string.
@param – `val` – value to URI-decode.

C.Encode – Data Encoding Functions

`C.Encode.base64`
(method) `Encode.base64(val: any, trimTrailEq?: boolean): string`
Returns a base64 representation of the given string or Buffer.

@param – `val` – value to base64-encode.
@param – `trimTrailEq` – whether to trim any trailing `=` .

`C.Encode.gzip`
(method) `Encode.gzip(value: string, encoding?: string): any`
Gzip, and optionally base64-encode, the supplied value.
@param – `value` – the value to gzip.
@param – `encoding` – 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.

`C.Encode.hex`
(method) `Encode.hex(val: string | number): string`
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` .
@param – `val` – value to convert to hex.

`C.Encode.uri`
(method) `Encode.uri(val: string): string`
Returns the URI-encoded representation of the given string.
@param – `val` – value to uri encode.

C.env – Environment

`C.env`
(property) `env: {[key: string]: string;}`
Returns an object containing Cribl LogStream's environment variables.

C.Lookup – Inline Lookup Functions

`C.Lookup` – Exact Lookup
(property) `Lookup: (file: string, primaryKey?: string, otherFields?: string[], ignoreCase?: boolean) ⇒ InlineLookup`
Returns an instance of a lookup to use inline.
Example invocation:
`C.Lookup('lookup_name.csv', 'IP_field_name_in_lookup_file').match(host)`

`C.LookupCIDR` – CIDR Lookup
(property) `LookupCIDR: (file: string, primaryKey?: string, otherFields?: string[]) ⇒ InlineLookup`
Returns an instance of a CIDR lookup to use inline.

`C.LookupIgnoreCase` – Case-insensitive Lookup
(property) `LookupIgnoreCase: (file: string, primaryKey?: string, otherFields?: string[]) ⇒ InlineLookup`
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](http://google.com)` - Regex Lookup
(property) `LookupRegex: (file: string, primaryKey?: string, otherFields?: string[]) ⇒ InlineLookup`
Returns an instance of a Regex lookup to use inline.

(method) `InlineLookup.match(value: string, fieldToReturn?: string): any`
@param – `value` – the value to look up.
@param – `fieldToReturn` – name of the lookup file > field to return.

E.g., `C.Lookup('lookup-exact.csv', 'foo').match('abc', 'bar')`
Return the value of field **bar** in the lookup table if field **foo** matches `abc`.

Example 1: `C.LookupCIDR('lookup-cidr.csv', 'foo').match('192.168.1.1', 'bar')`
Return the value of field **bar** in the lookup table if the CIDR range in **foo** includes `192.168.1.1`.

Example 2: `C.LookupCIDR('lookup-cidr.csv', 'cidr').match(hostIP, 'location')`

Example 3: `C.LookupRegex('lookup-regex.csv', 'foo').match('manchester', 'bar')`
Return the value of field **bar** in the lookup table if the regex in **foo** matches the string `manchester`.

C.Mask – Data Masking Functions

`C.Mask.CC`
(method) `Mask.CC(value: string, unmasked?: number, maskChar?: string): string`
Check whether a value could be a valid credit card number, and mask a subset of the value. By default, all digits except the last 4 will be replaced with `X`.
@param – `value` – a string whose digits to mask IFF it could be a valid credit card number.
@param – `unmasked` – number of digits to leave unmasked: positive for left,

negative for right, `0` for none.
@param – `maskChar` – a string/char to replace a digit with.

`C.Mask.IMEI`
(method) `Mask.IMEI(value: string, unmasked?: number, maskChar?: string): string`
Check whether a value could be a valid IMEI number, and mask a subset of the value. By default, all digits except the last 4 will be replaced with `X`.
@param – `value` – a string whose digits to mask IFF it could be a valid IMEI number.
@param – `unmasked` – number of digits to leave unmasked: positive for left, negative for right, `0` for none.
@param – `maskChar` – a string/char to replace a digit with.

`C.Mask.isCC`
(method) `Mask.isCC(value: string): boolean`
Checks whether the given value could be a valid credit card number, by computing the string's Lunh's checksum modulo 10 == `0`.
@param – `value` – a string to check for being a valid credit card number.

`C.Mask.isIMEI`
(method) `Mask.isIMEI(value: string): boolean`
Checks whether the given value could be a valid IMEI number, by computing the string's Lunh's checksum modulo 10 == `0`.
@param – `value` – a string to check for being a valid IMEI number

`C.Mask.luhn`
(method) `Mask.luhn(value: string, unmasked?: number, maskChar?: string): string`
Check that value Lunh's checksum mod 10 is `0`, and mask a subset of the value. By default, all digits except the last 4 will be replaced with `X`. If the value's Lunh's checksum mod 10 is not `0`, then the value is returned unmodified.
@param – `value` – a string whose digits to mask IFF the value's Lunh's checksum mod 10 is `0`.
@param – `unmasked` – number of digits to leave unmasked: positive for left, negative for right, `0` for none.
@param – `maskChar` – a string/char to replace a digit with.

`C.Mask.LUHN_SUB`
(property) `Mask.LUHN_SUB: any`

`C.Mask.luhnChecksum`

(method) `Mask.luhnChecksum(value: string, mod?: number): number`

Generates the Luhn checksum (used to validate certain credit card numbers, IMEIs, etc.). By default, the mod 10 of the checksum is returned. Pass mod = `0` to get the actual checksum.

@param – `value` – a string whose digits you want to perform the Lunh checksum on.

@param – `mod` – return checksum modulo this number. If `0`, skip modulo. Default is `10`.

`C.Mask.md5`

(method) `Mask.md5(value: string, len?: string | number): string`

Generate MD5 hash of a given value.

@param – `value` – compute the hash of this.

@param – `len` – length of hash to return: 0 for full hash, a +number for left or a -number for right substring. If a string is passed it's length will be used.

`C.Mask.random`

(method) `Mask.random(len?: string | number): string`

Generates a random alphanumeric string.

@param – `len` – a number indicating the length of the result; or, if a string, use its length.

`C.Mask.REDACTED`

(property) `Mask.REDACTED: string`

The literal `'REDACTED'`.

`C.Mask.repeat`

(method) `Mask.repeat(len?: string | number, char?: string): string`

Generates a repeating char/string pattern, e.g., `XXXX`.

@param – `len` – a number indicating the length of the result; or, if a string, use its length.

@param – `char` – pattern to repeat `len` times.

`C.Mask.sha1`

(method) `Mask.sha1(value: string, len?: string | number): string`

Generate SHA1 hash of given value.

@param – `value` - compute the hash of this.

@param – `len` - 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

C.Misc – Miscellaneous Utility Functions

`C.Misc.zip()`
(method) `Misc.zip(keys: string[], values: any[], dest?: any): any`
Set the given keys to the corresponding values on the given `dest` object. If `dest` is not provided, a new object will be constructed.
@param – `keys` – field names corresponding to keys.
@param – `values` – values corresponding to values.
@param – `dest` – object on which to set field values.
@returns – object on which the fields were set.

E.g., `people = C.Misc.zip(titles, names)`
Sample data: `titles=['ceo', 'svp', 'vp']`, `names=['foo', 'bar', 'baz']`
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.Net – Network Functions

`C.Net.cidrMatch()`
(method) `Net.cidrMatch(cidrIpRange: string, ipAddress: string): boolean`
Determines whether the supplied IPv4 `ipAddress` is inside the range of addresses identified by `cidrIpRange`. For example: `C.Net.cidrMatch('10.0.0.0/24', '10.0.0.100')` returns `true`.
@param – `cidrIpRange` – IPv4 address range in CIDR format. E.g., `10.0.0.0/24`.
@param – `ipAddress` – The IPv4 IP address to test for inclusion in `cidrIpRange`.

`C.Net.ipv6Normalize()`
(method) `Net.ipv6Normalize(address: string): string`
Normalize an IPV6 address based on RFC draft-ietf-6man-text-addr-representation-04.
@param – `address` – the IPV6 address to normalize.

`C.Net.isPrivate()`
(method) `Net.isPrivate(address: string): string`
Determine whether the supplied IPv4 address is in the range of private addresses per RFC1918.
@param – `address` – address to test.

C.os – System Functions

`C.confVersion`
Returns Cribl LogStream config version.

`C.os.hostname()`
Returns hostname of the system running this Cribl LogStream instance.

C.Schema – Schema Functions

`C.Schema()`
(property) `Schema: (id: string) ⇒ SchemaValidator`
(method) `SchemaValidator.validate(data: any): boolean`
Validates the given object against the schema.
@param – `data` – object to be validated.
@returns – `true` when schema is valid; otherwise, `false` .

Example: `C.Schema('schema1').validate(myField)` will validate if `myField` object conforms to `schema1` .

See Schema Library for more details.

C.Text – Text Functions

`C.Text.entropy()`
(method) `Text.entropy(bytes: any): number`
Computes the Shannon entropy of the given buffer or string.
@param – `bytes` – value to undergo Shannon entropy computation.
@returns – the entropy value; or `-1` in case of an error.

`C.Text.hashCode()`
(method) `Text.hashCode(val: string | Buffer | number): number`
Computes hashcode (djb2) of the given value.
@param – `val` - value to be hashed.
@returns – hashcode value.

`C.Text.isASCII()`
(method) `Text.isASCII(bytes: any): boolean`
Checks whether all bytes or chars are in the ASCII printable range.
@param – `bytes` – value to check for character range.
@returns – `true` if all chars/bytes are within ASCII printable range; otherwise, `false` .

`C.Text.isUTF8()`

(method) `Text.isUTF8(bytes: any): boolean`

Checks whether the given Buffer contains valid UTF8.

@param – `bytes` – bytes to check.

@returns – `true` if bytes are UTF8; otherwise, `false` .

`C.Text.parseWinEvent`

(method) `C.Text.parseWinEvent(xml: string, nonValues?: string[]): any`

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.

@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.

`C.Text.parseXml`

(method) `C.Text.parseXml(xml:string, keepAttr?: boolean, keepMetadata?: boolean, nonValues?: string[]): any`

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.

@param – `xml` – XML string, or an event field containing the XML.

@param – `keepAttr` – whether or not to include attributes in the returned object. Defaults to `true` .

@param – `keepMetadata` – 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.)

@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 `[]` (empty array), meaning discard no elements.

@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.

`C.Text.relativeEntropy()`

(method) `Text.relativeEntropy(bytes: any, modelName?: string):`

`number`
Computes the relative entropy of the given buffer or string.
@param – `bytes` – value whose relative entropy to compute.
@param – `modelName` – Name of the model to test the string with.
@returns – the relative entropy value, or `-1` in case of an error.

C.Time – Time Functions

`C.Time.adjustTZ()`
(method) `Time.adjustTZ(epochTime: number, tzTo: string, tzFrom?: string): number`
Adjust a timestamp from one timezone to another.
@param – `epochTime` – UNIX epoch time.
@param – `tzTo` – timezone to adjust to.
@param – `tzFrom` – optional timezone of the timestamp.
@returns – the adjusted timestamp, in UNIX epoch time (ms).

`C.Time.clamp()`
(method) `Time.clamp(date, earliest, latest, defaultDate?): number`
Constrains an event's parsed timestamp to realistic earliest and latest boundaries.
@param – `date` – Timestamp originally parsed from event, in UNIX epoch time (ms) or JavaScript Date format.
@param – `earliest` – earliest allowable timestamp, in UNIX epoch time (ms) or JS Date format.
@param – `latest` – latest allowable timestamp, in UNIX epoch time (ms) or JS Date format.
@param – `defaultDate` – optional default date, in UNIX epoch time (ms) or JS Date format, to substitute for values outside the `earliest` or `latest` boundaries.

`C.Time.strftime()`
(method) `Time.strftime(date: number | Date, format: string, utc?: boolean): string`
Format a Date object or number as a time string, using strftime specifier.
@param – `date` – Date object or number (seconds since epoch) to format.
@param – `format` – specifier to use to format the date.
@param – `utc` – whether to output the time in UTC, rather than in local timezone.
@returns – representation of the given date.

`C.Time.strptime()`
(method) `Time.strptime(str: string, format: string, utc?: boolean, strict?: boolean): Date`
Extract time from a string using strptime specifier.
@param – `str` – string to parse to a timestamp (see strict flag).
@param - `format` - `strptime` specifier.
@param – `utc` – whether to interpret times as UTC, rather than as local time.
@param – `strict` – whether to return `null` if there are any extra characters after timestamp.
@returns – a parsed Date object, if successful; otherwise, `null` if the specifier did not match.

`C.Time.timestampFinder()`
(method) `Time.timestampFinder(utc?: boolean).find(<source-field>): AutoTimeParser`
Extract time from the specified `<source-field>`, using the same algorithm as the Auto Timestamp Function and the Event Breaker Function.
@param – `utc` – whether to output the time in UTC, rather than in local timezone.
@param – `<source-field>` – the field in which to search for the time.
@returns – representation of the extracted time.

C.vars – Global Variables

See Global Variables Library for more details.

C.version – Cribl LogStream Version

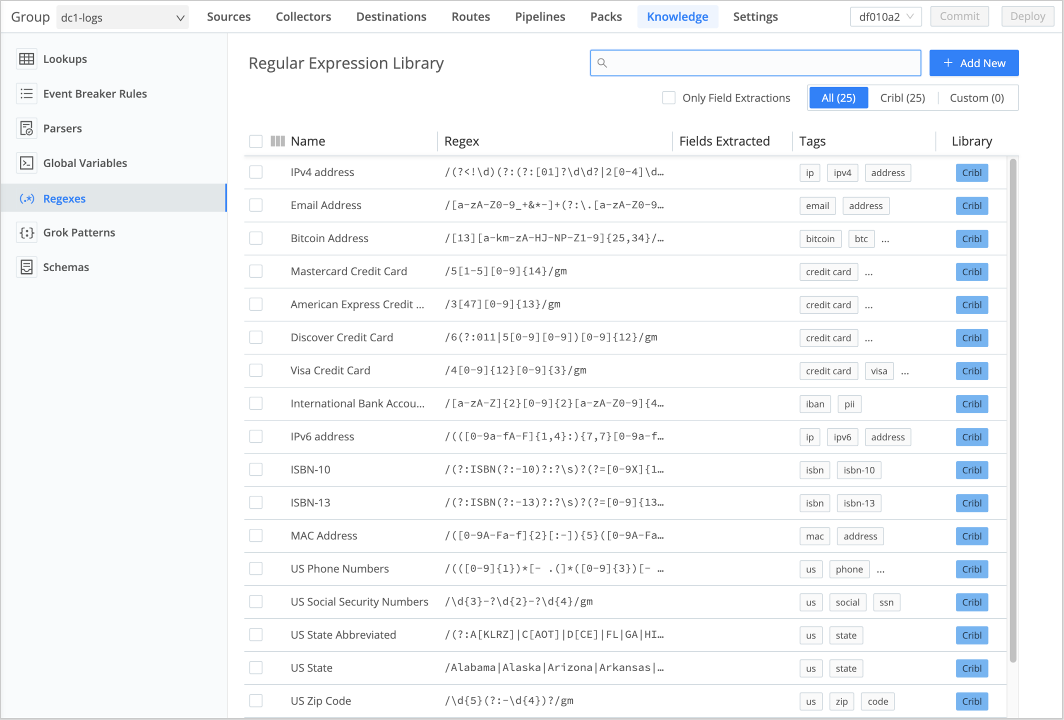
(property) `version: string`
Cribl LogStream Version.

KNOWLEDGE

Regex Library

What Is the Regex Library

Cribl LogStream ships with a Regex Library that contains a set of pre-built common regex patterns. This library serves as an easily accessible repository of regular expressions. The Library is searchable, and you can assign tags to each pattern for further organization or categorization. The Library is located under **Knowledge** > **Regex Library** .



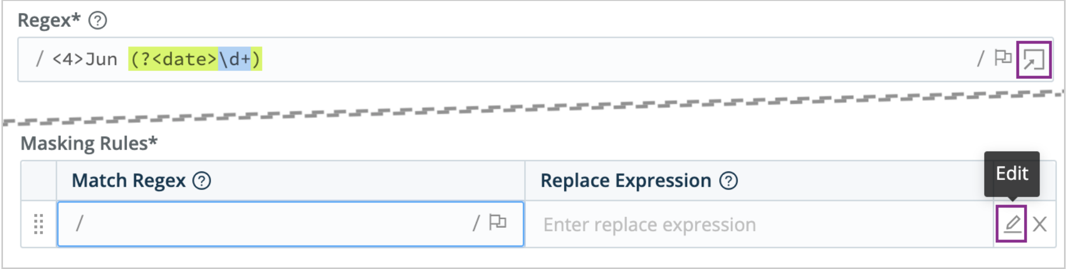
Regular Expression Library

<table>
<tr><th>Name</th><th>Regex</th><th>Fields Extracted</th><th>Tags</th><th>Library</th></tr>
<tr><td>IPv4 address</td><td>/(?<!\d)(?:(?:[01]?\d\d?|2[0-4]\d…</td><td></td><td>ip
ipv4
address</td><td>Cribl</td></tr>
<tr><td>Email Address</td><td>/[a-zA-Z0-9_+&*-]+(?:\.[a-zA-Z0-9…</td><td></td><td>email
address</td><td>Cribl</td></tr>
<tr><td>Bitcoin Address</td><td>/[13][a-km-zA-HJ-NP-Z1-9]{25,34}/…</td><td></td><td>bitcoin
btc
...</td><td>Cribl</td></tr>
<tr><td>Mastercard Credit Card</td><td>/5[1-5][0-9]{14}/gm</td><td></td><td>credit card
...</td><td>Cribl</td></tr>
<tr><td>American Express Credit ...</td><td>/3[47][0-9]{13}/gm</td><td></td><td>credit card
...</td><td>Cribl</td></tr>
<tr><td>Discover Credit Card</td><td>/6(?:011|5[0-9][0-9])[0-9]{12}/gm</td><td></td><td>credit card
...</td><td>Cribl</td></tr>
<tr><td>Visa Credit Card</td><td>/4[0-9]{12}[0-9]{3}/gm</td><td></td><td>credit card
visa
...</td><td>Cribl</td></tr>
<tr><td>International Bank Accou...</td><td>/[a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{4…</td><td></td><td>iban
pii</td><td>Cribl</td></tr>
<tr><td>IPv6 address</td><td>/(([0-9a-fA-F]{1,4}:){7,7}[0-9a-f…</td><td></td><td>ip
ipv6
address</td><td>Cribl</td></tr>
<tr><td>ISBN-10</td><td>/(?:ISBN(?:-10)?:?\s)?(?=[0-9X]{1…</td><td></td><td>isbn
isbn-10</td><td>Cribl</td></tr>
<tr><td>ISBN-13</td><td>/(?:ISBN(?:-13)?:?\s)?(?=[0-9]{13…</td><td></td><td>isbn
isbn-13</td><td>Cribl</td></tr>
<tr><td>MAC Address</td><td>/([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa…</td><td></td><td>mac
address</td><td>Cribl</td></tr>
<tr><td>US Phone Numbers</td><td>/(([0-9]{1})*[- .(]*([0-9]{3})[- …</td><td></td><td>us
phone
...</td><td>Cribl</td></tr>
<tr><td>US Social Security Numbers</td><td>/\d{3}-?\d{2}-?\d{4}/gm</td><td></td><td>us
social
ssn</td><td>Cribl</td></tr>
<tr><td>US State Abbreviated</td><td>/(?:A[KLRZ]|C[AOT]|D[CE]|FL|GA|HI…</td><td></td><td>us
state</td><td>Cribl</td></tr>
<tr><td>US State</td><td>/Alabama|Alaska|Arizona|Arkansas|…</td><td></td><td>us
state</td><td>Cribl</td></tr>
<tr><td>US Zip Code</td><td>/\d{5}(?:-\d{4})?/gm</td><td></td><td>us
zip
code</td><td>Cribl</td></tr>
</table>

Regular Expression Library

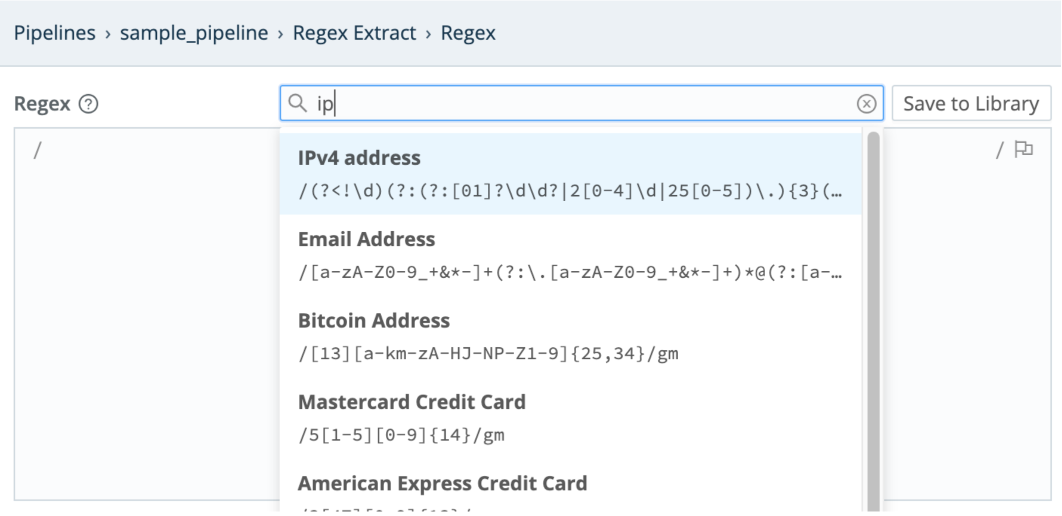
Using Library Patterns

As of this version, the Library contains 25 patterns shipped by Cribl LogStream. To insert a pattern into a Function's regex field, first click the pop-out or Edit icon beside that field.



Opening a Regex modal

In the resulting Regex or Rules modal, Regex Library patterns will appear as typeahead options. Click a pattern to paste it in. You can then use the pattern as-is, or modify it as necessary.



IPv4 address
/(?<!\d)(?:(?:[01]?\d\d?|2[0-4]\d|25[0-5])\.){3}(…

Email Address
/[a-zA-Z0-9_+&*-]+(?:\.[a-zA-Z0-9_+&*-]+)*@(?:[a-…

Bitcoin Address
/[13][a-km-zA-HJ-NP-Z1-9]{25,34}/gm

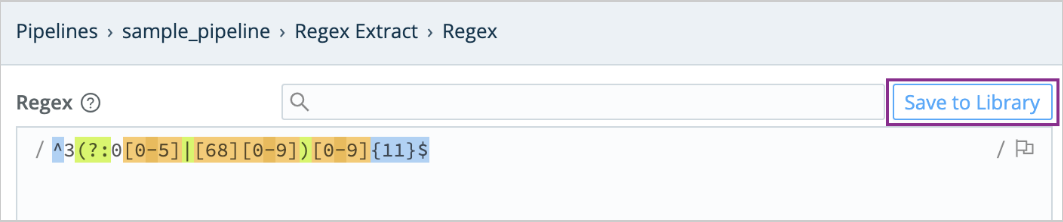
Mastercard Credit Card
/5[1-5][0-9]{14}/gm

American Express Credit Card

Inserting a pattern from the Regex Library

Adding Patterns to the Library

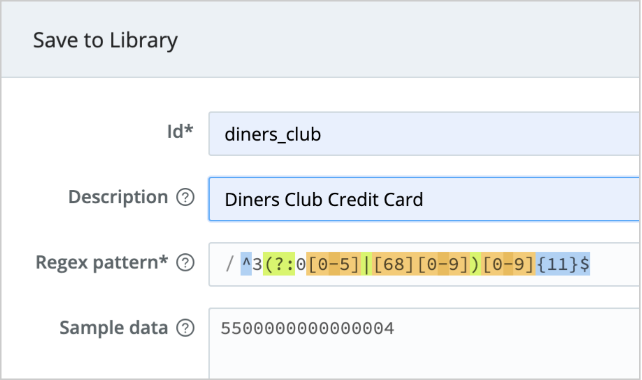
You can also add new, custom patterns to the Library. In the same modal, once you've built your pattern, click the **Save to Library** button.



/ ^3(?:0[0-5]|[68][0-9])[0-9]{11}$

Adding a custom pattern to the Regex Library from a Function's Regex modal

In the resulting modal, give your custom pattern a unique ID. Optionally, you can also provide a **Description** (name) and groom the **Sample data**. Then click **Save**.



Save to Library

Id*	diners_club
Description	Diners Club Credit Card
Regex pattern*	/ ^3(?:0[0-5]|[68][0-9])[0-9]{11}$
Sample data	5500000000000004

Identifying the custom pattern

Your custom pattern will now reside in the Regex Library. It will be available to Functions using the same typeahead assist as Cribl's pre-built patterns.

Cribl vs. Custom and Priority

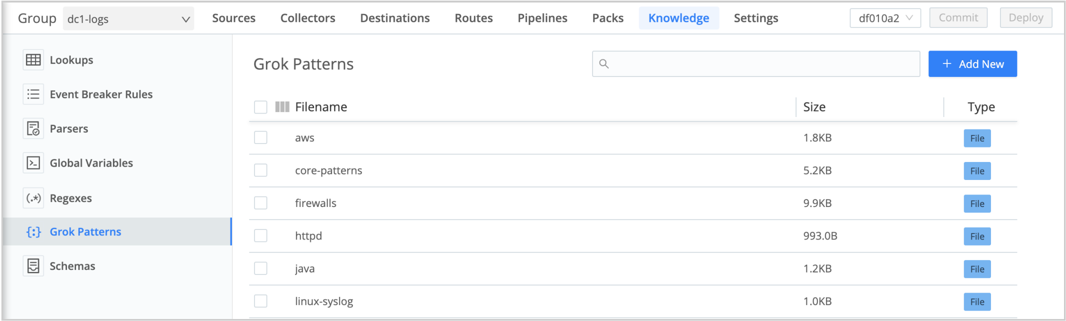
Within the Library, patterns shipped by Cribl will be listed under the **Cribl** tab, while those built by users will be found under **Custom**. Over time, Cribl LogStream will ship more patterns, and this distinction allows for both sets to grow independently.

In the case of an ID/Name conflict, 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.

Grok Patterns Library

What Is the Grok Patterns Library

Cribl LogStream ships with a Grok Patterns Library that contains a set of pre-built common patterns, organized as files.



Filename	Size	Type
aws	1.8KB	File
core-patterns	5.2KB	File
firewalls	9.9KB	File
httpd	993.0B	File
java	1.2KB	File
linux-syslog	1.0KB	File

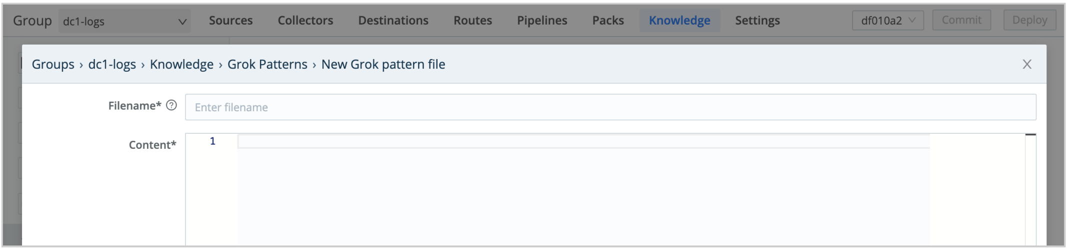
Grok Patterns Library

Managing Library Patterns

You can access the Grok Patterns Library in the UI by selecting **Knowledge** > **Grok Patterns**. The library contains several pattern files that Cribl provides for basic Grok scenarios, and is searchable.

To edit a pattern file, click **Edit** in its **Actions** column.

To create a new pattern file, click **+ Add New**. In the resulting **Create Grok Patterns** modal, assign a unique **Filename**, populate the file with patterns, then click **Save**.



> ℹ Pattern files reside in:
> `$CRIBL_HOME/(default|local)/cribl/grok-patterns/`

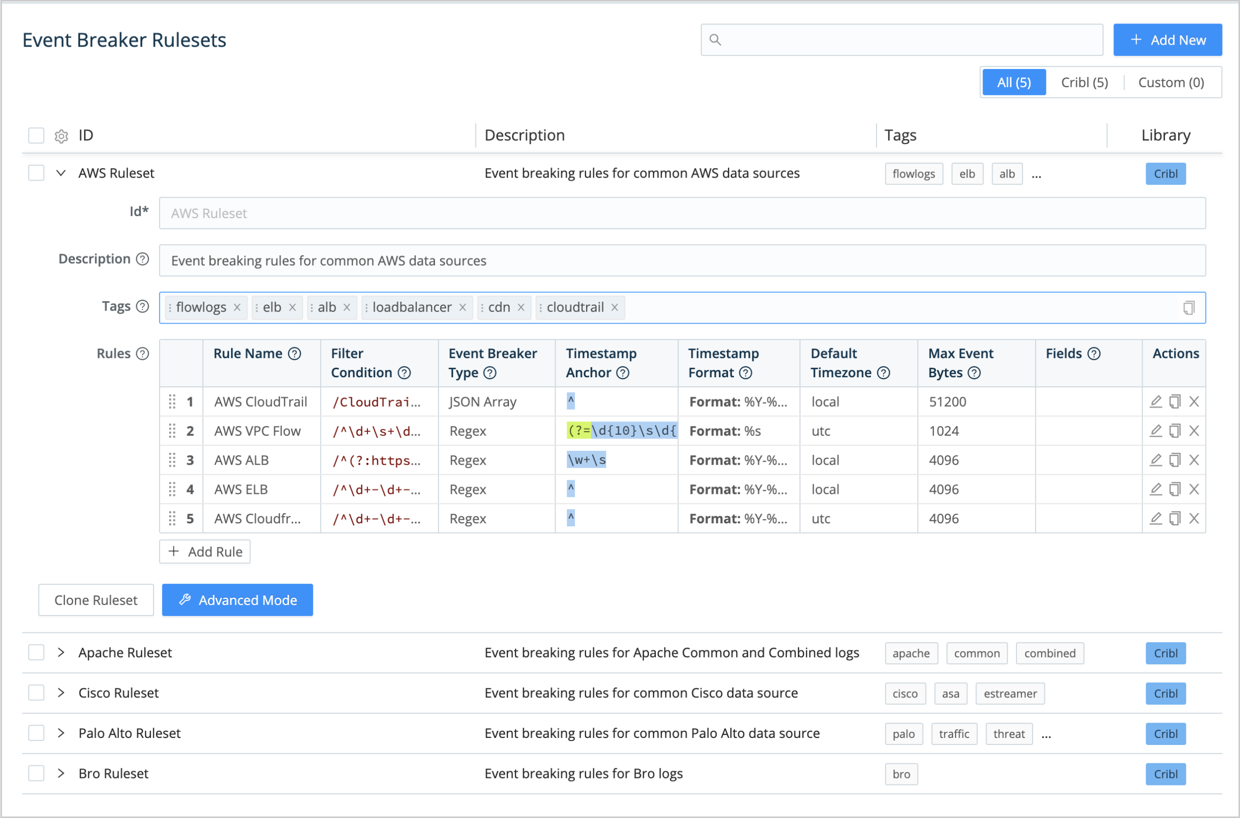
Using Grok Patterns

In the current LogStream version, you apply Grok patterns by inserting a Grok Function into a Pipeline, then manually typing or pasting patterns into the **Pattern** field(s).

Event Breakers

What Are Event Breakers

Event Breakers help break incoming streams of data into discrete events. You access the Event Breakers management interface under **Knowledge** > **Event Breakers**. 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

ID	Description	Tags	Library
AWS Ruleset	Event breaking rules for common AWS data sources	flowlogs elb alb ...	Cribl

Id*	AWS Ruleset
Description	Event breaking rules for common AWS data sources
Tags	flowlogs × elb × alb × loadbalancer × cdn × cloudtrail ×

Rules

	Rule Name	Filter Condition	Event Breaker Type	Timestamp Anchor	Timestamp Format	Default Timezone	Max Event Bytes	Fields	Actions
1	AWS CloudTrail	`/CloudTrai...`	JSON Array	`^`	**Format:** %Y-%...	local	51200		
2	AWS VPC Flow	`/^\d+\s+\d...`	Regex	`(?=\d{10}\s\d{`	**Format:** %s	utc	1024		
3	AWS ALB	`/^(?:https...`	Regex	`\w+\s`	**Format:** %Y-%...	local	4096		
4	AWS ELB	`/^\d+-\d+-...`	Regex	`^`	**Format:** %Y-%...	local	4096		
5	AWS Cloudfr...	`/^\d+-\d+-...`	Regex	`^`	**Format:** %Y-%...	utc	4096		

ID	Description	Tags	Library
Apache Ruleset	Event breaking rules for Apache Common and Combined logs	apache common combined	Cribl
Cisco Ruleset	Event breaking rules for common Cisco data source	cisco asa estreamer	Cribl
Palo Alto Ruleset	Event breaking rules for common Palo Alto data source	palo traffic threat ...	Cribl
Bro Ruleset	Event breaking rules for Bro logs	bro	Cribl

Event Breaker Rulesets page

Event Breaker Rulesets

Rulesets are **collections of Event Breaker rules** that are associated with Sources. Rules define configurations needed to break down a stream of data into events.

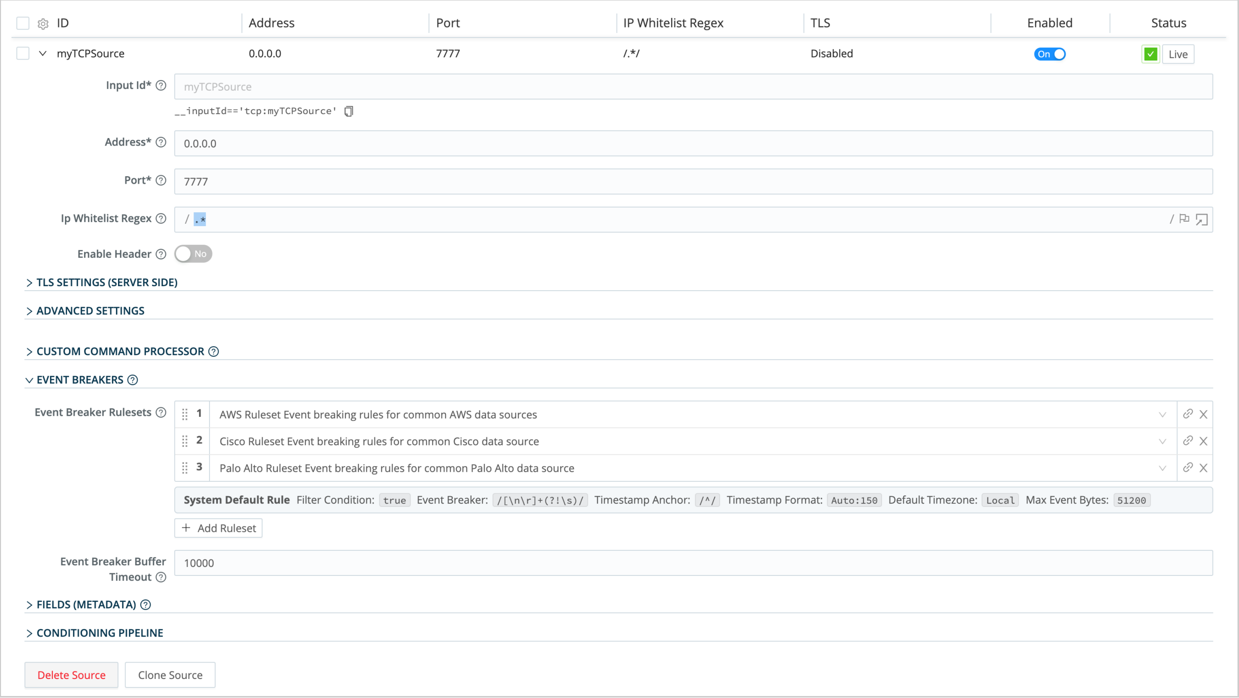
> ℹ Event Breakers are accessible only on Sources that require incoming events to be broken into a better-defined format. Check individual LogStream Sources' documentation for Event Breaker support.

Rules within a ruleset are ordered and evaluated top->down. One or more rulesets can be associated with a Source, and these rulesets are also evaluated top->down. For a stream from a given Source, the first matching rule goes into effect.

Rulesets and Rules - Ordered

```
Ruleset A
  Rule 1
  Rule 2
  ...
  Rule n

...

Ruleset B
  Rule Foo
  Rule Bar
  ...
  Rule FooBar
```

An example of multiple rulesets associated with a Source:



ID	Address	Port	IP Whitelist Regex	TLS	Enabled	Status
myTCPSource	0.0.0.0	7777	/.*/	Disabled	On	✓ Live

Input Id*: myTCPSource

`__inputId=='tcp:myTCPSource'`

Address*: 0.0.0.0

Port*: 7777

Ip Whitelist Regex: / .* /

Enable Header: No

Event Breaker Rulesets:

1 AWS Ruleset Event breaking rules for common AWS data sources

2 Cisco Ruleset Event breaking rules for common Cisco data source

3 Palo Alto Ruleset Event breaking rules for common Palo Alto data source

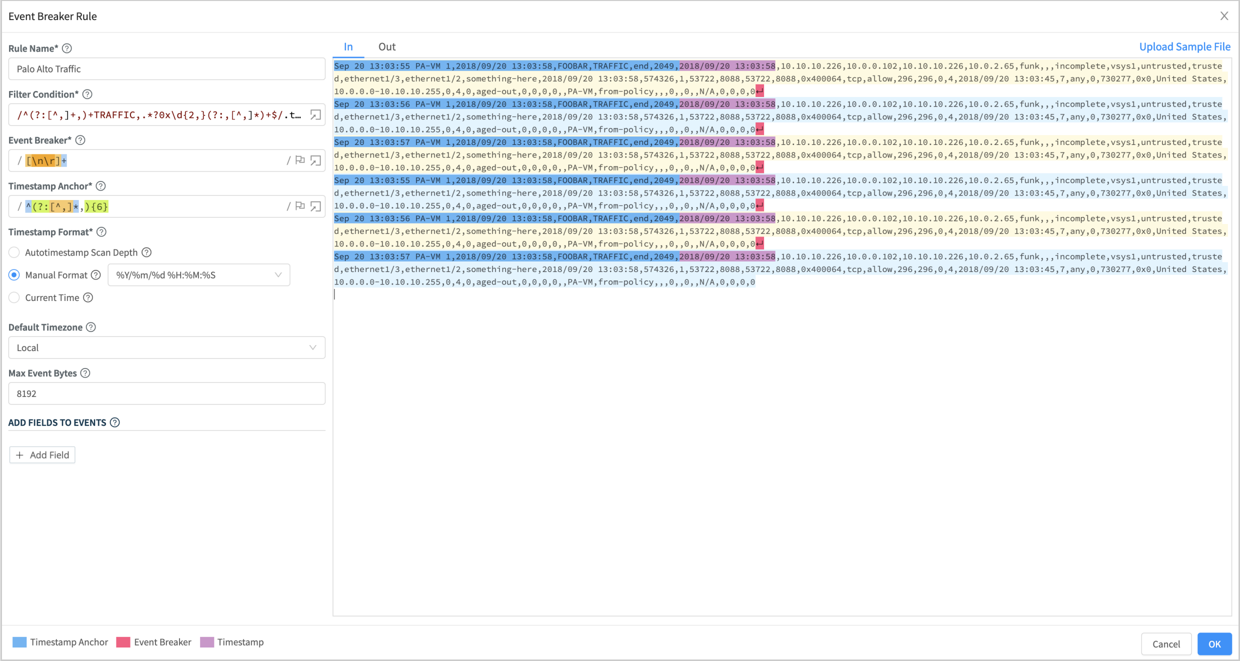
System Default Rule Filter Condition: `true` Event Breaker: `/[\n\r]+(?!\s)/` Timestamp Anchor: `/^/` Timestamp Format: `Auto:150` Default Timezone: `Local` Max Event Bytes: `51200`

Event Breaker Buffer Timeout: 10000

Three Event Breaker rulesets on a Source

Rule Example

This rule breaks on newlines and uses Manual timestamping **after** the sixth comma, as indicated by this pattern: `^(?:[^,]*,){6}` .



Event Breaker Rule

Rule Name*: Palo Alto Traffic

Filter Condition*: `/^(?:[^,]+,)+TRAFFIC,.*?0x\d{2,}(?:,[^,]*)+$/.t…`

Event Breaker*: `/ [\n\r]+ /`

Timestamp Anchor*: `/ ^(?:[^,]*,){6} /`

Timestamp Format*:

○ Autotimestamp Scan Depth

● Manual Format: %Y/%m/%d %H:%M:%S

○ Current Time

Default Timezone: Local

Max Event Bytes: 8192

ADD FIELDS TO EVENTS

```
Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:58,10.10.10.226,10.0.0.102,10.10.10.226,10.0.2.65,funk,,,incomplete,vsys1,untrusted,truste
d,ethernet1/3,ethernet1/2,something-here,2018/09/20 13:03:58,574326,1,53722,8088,53722,8088,0x400064,tcp,allow,296,296,0,4,2018/09/20 13:03:45,7,any,0,730277,0x0,United States,
10.0.0.0-10.10.10.255,0,4,0,aged-out,0,0,0,0,,PA-VM,from-policy,,,0,,0,,N/A,0,0,0,0↵
Sep 20 13:03:56 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:58,10.10.10.226,10.0.0.102,10.10.10.226,10.0.2.65,funk,,,incomplete,vsys1,untrusted,truste
d,ethernet1/3,ethernet1/2,something-here,2018/09/20 13:03:58,574326,1,53722,8088,53722,8088,0x400064,tcp,allow,296,296,0,4,2018/09/20 13:03:45,7,any,0,730277,0x0,United States,
10.0.0.0-10.10.10.255,0,4,0,aged-out,0,0,0,0,,PA-VM,from-policy,,,0,,0,,N/A,0,0,0,0↵
Sep 20 13:03:57 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:58,10.10.10.226,10.0.0.102,10.10.10.226,10.0.2.65,funk,,,incomplete,vsys1,untrusted,truste
d,ethernet1/3,ethernet1/2,something-here,2018/09/20 13:03:58,574326,1,53722,8088,53722,8088,0x400064,tcp,allow,296,296,0,4,2018/09/20 13:03:45,7,any,0,730277,0x0,United States,
10.0.0.0-10.10.10.255,0,4,0,aged-out,0,0,0,0,,PA-VM,from-policy,,,0,,0,,N/A,0,0,0,0↵
Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:58,10.10.10.226,10.0.0.102,10.10.10.226,10.0.2.65,funk,,,incomplete,vsys1,untrusted,truste
d,ethernet1/3,ethernet1/2,something-here,2018/09/20 13:03:58,574326,1,53722,8088,53722,8088,0x400064,tcp,allow,296,296,0,4,2018/09/20 13:03:45,7,any,0,730277,0x0,United States,
10.0.0.0-10.10.10.255,0,4,0,aged-out,0,0,0,0,,PA-VM,from-policy,,,0,,0,,N/A,0,0,0,0↵
Sep 20 13:03:56 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:58,10.10.10.226,10.0.0.102,10.10.10.226,10.0.2.65,funk,,,incomplete,vsys1,untrusted,truste
d,ethernet1/3,ethernet1/2,something-here,2018/09/20 13:03:58,574326,1,53722,8088,53722,8088,0x400064,tcp,allow,296,296,0,4,2018/09/20 13:03:45,7,any,0,730277,0x0,United States,
10.0.0.0-10.10.10.255,0,4,0,aged-out,0,0,0,0,,PA-VM,from-policy,,,0,,0,,N/A,0,0,0,0↵
Sep 20 13:03:57 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2049,2018/09/20 13:03:58,10.10.10.226,10.0.0.102,10.10.10.226,10.0.2.65,funk,,,incomplete,vsys1,untrusted,truste
d,ethernet1/3,ethernet1/2,something-here,2018/09/20 13:03:58,574326,1,53722,8088,53722,8088,0x400064,tcp,allow,296,296,0,4,2018/09/20 13:03:45,7,any,0,730277,0x0,United States,
10.0.0.0-10.10.10.255,0,4,0,aged-out,0,0,0,0,,PA-VM,from-policy,,,0,,0,,N/A,0,0,0,0
```

An Event Breaker rule

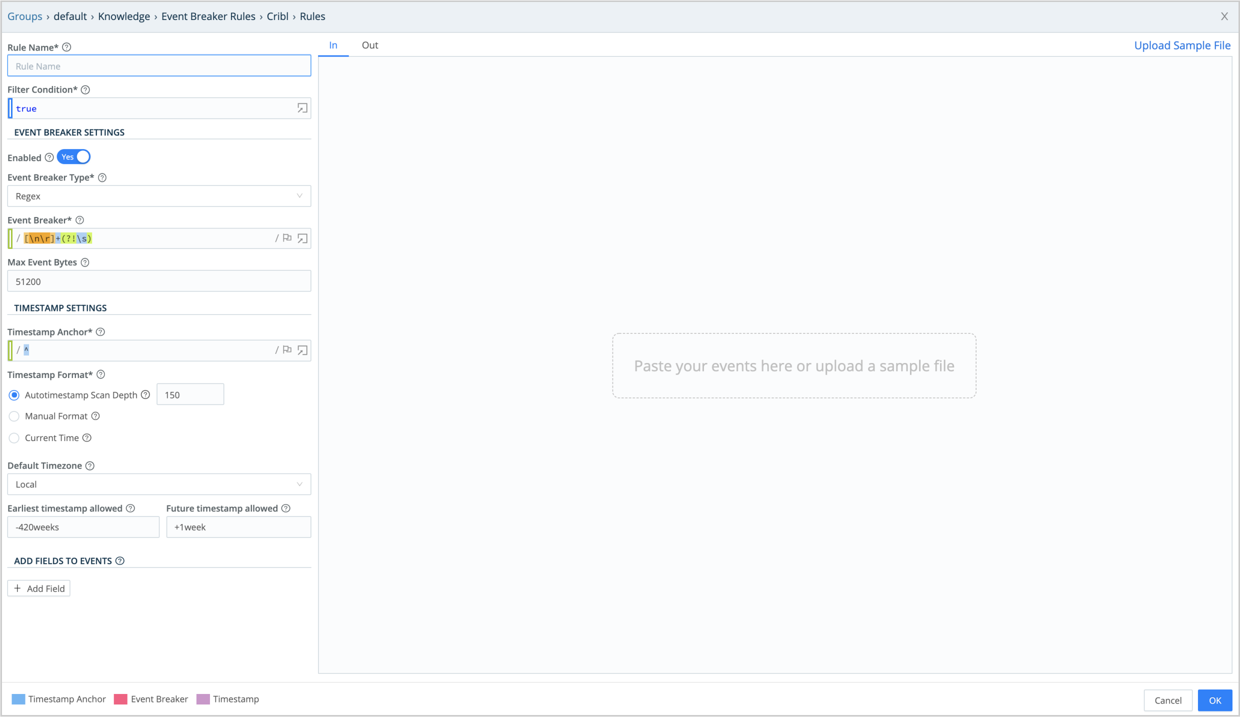
System Default Rule

The system default rule functionally sits at the bottom of the ruleset/rule hierarchy (but is built-in and not displayed on the Event Breakers page), and goes into effect if there are no matching rules:

- Filter Condition defaults to `true`
- Event Breaker to `[\n\r]+(?!\s)`
- Timestamp anchor to `^`
- Timestamp format to `Auto` and a scan depth of `150` bytes
- Max Event Bytes to `51200`
- Default Timezone to `Local`

How Do Event Breakers Work

On the **Event Breaker Rulesets** page (see screenshot above), click **+ Add New** to create a new Event Breaker ruleset. Click **+ Add Rule** within a ruleset to add a new Event Breaker.



Groups › default › Knowledge › Event Breaker Rules › Cribl › Rules
In
Out
Upload Sample File
Paste your events here or upload a sample file
Cancel
OK

Rule Name*: Rule Name

Filter Condition*: `true`

EVENT BREAKER SETTINGS

Enabled: Yes

Event Breaker Type*: Regex

Event Breaker*: `/ [\n\r]+(?!\s) /`

Max Event Bytes: 51200

TIMESTAMP SETTINGS

Timestamp Anchor*: `/ ^ /`

Timestamp Format*:
- ● Autotimestamp Scan Depth: 150
- ○ Manual Format
- ○ Current Time

Default Timezone: Local

Earliest timestamp allowed: -420weeks

Future timestamp allowed: +1week

ADD FIELDS TO EVENTS

+ Add Field

🟦 Timestamp Anchor 🟥 Event Breaker 🟪 Timestamp

Adding a new Event Breaker rule

Each Event Breaker includes the following components, which you configure from top to bottom in the above **Event Breaker Rule** modal:

Filter Condition

As a stream of data moves into the engine, a rule's filter expression is applied. If the expression evaluates to `true`, the rule configurations are engaged for the entire duration of that stream. Else, the next rule down the line is evaluated.

Event Breaker Type

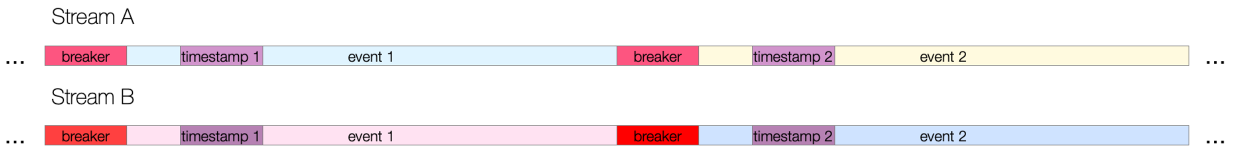
After a breaker pattern has been selected, it will apply on the stream **continuously**. See below for specific information on different Event Breaker Types.

Timestamp Settings

After events are synthesized out of streams, LogStream will attempt timestamping. First, a timestamp anchor will be located inside the event. Next, starting there, the engine will try to do one of the following:

- Scan up to a configurable depth into the event and autotimestamp, **or**
- Timestamp using a manually supplied `strptime` format, **or**
- Timestamp the event with the current time.

The closer an anchor is to the timestamp pattern, the better the performance and accuracy – especially if multiple timestamps exist within an event. For the manually supplied option, the anchor must lead the engine **right before** the timestamp pattern begins.



Anchors preceding timestamps

> This timestamping executes the same basic algorithm as the Auto Timestamp Function and the C.Time.timestampFinder() native method.

Add Fields to Events

After events have been timestamped, one or more fields can be added here as key-value pairs. In each field's **Value Expression**, you can fully evaluate the field value using JavaScript expressions.

Event Breaker Types

Several types of Event Breaker can applied to incoming data streams:

- Regex
- File Header
- JSON Array
- JSON New Line Delimited
- Timestamp
- CSV

Regex

The Regex breaker uses regular expressions to find breaking points in data streams.

After a breaker regex pattern has been selected, it will apply on the stream **continuously**. Breaking will occur at the beginning of the match, and the matched content will be consumed/thrown away.
If necessary, you can use a positive lookahead regex to keep the content – e.g.: `(?=pattern)`

Capturing groups are **not allowed** to be used anywhere in the Event Breaker pattern, as they will further break the stream – which is often undesirable. Breaking will also occur if **Max Event Bytes** has been reached.

Example

Break after a newline or carriage return, but only if followed by a timestamp pattern:

Event Breaker: `[\n\r]+(?=\d+-\d+-\d+\s\d+:\d+:\d+)`

Sample Event - Multiline

```
--- input ---
2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP feed, then you can program the aux
   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 program the aux
   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!


--- output event 1 ---
{
  "_raw": "2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP feed, then you can prog
  "_time": 1589920332
}

--- output event 2 ---
{
  "_raw": "2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP feed, then you can prog
  "_time": 1589920332
}
```

File Header

You can use the File Header breaker to break files with headers, such as IIS or Bro logs. This type of breaker relies on a header section that lists field names. The header section is typically present at the top of the file, and can be single-line or greater.

After the file has been broken into events, fields will also be extracted, as follows:

- **Header Line**: Regex matching a file header line. For example, `^#` .
- **Field Delimiter**: Field delimiter regex. For example, `\s+` .
- **Field Regex**: Regex with one capturing group, capturing all the fields to be broken by field delimiter. For example, `^#[Ff]ields[:]?\s+(.*)`
- **Null Values**: Representation of a null value. Null fields are not added to events.
- **Clean Fields**: Whether to clean up field names by replacing non `[a-zA-Z0-9]` characters with `_` .

Example

Using the values above, let's see how this sample file breaks up:

Sample Event - File Header

```
--- input ---
#fields ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       proto
#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
```

```
--- output event 1 ---
{
  "_raw": "1331904608.080000       -     192.168.204.59  137     192.168.204.255 137     udp",
  "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
}

--- output event 2 ---
{
  "_raw": "1331904609.190000       -     192.168.202.83  48516   192.168.207.4   53      udp",
  "ts": "1331904609.190000",
  "id_orig_h": "192.168.202.83",
  "id_orig_p": "48516",
  "id_resp_h": "192.168.207.4",
  "id_resp_p": "53",
  "proto": "udp",
  "_time": 1331904609.19
}
```

JSON Array

You can use the JSON Array to extract events from an array in a JSON document (e.g., an Amazon CloudTrail file).

- **Array Field**: Optional path to array in a JSON event with records to extract. For example, `Records` .
- **Timestamp Field**: Optional path to timestamp field in extracted events. For example, `eventTime` or `level1.level2.eventTime` .
- **JSON Extract Fields**: Enable this slider to auto-extract fields from JSON events. If disabled, only `_raw` and `time` will be defined on extracted events.
- **Timestamp Format**: If **JSON Extract Fields** is set to **No**, you **must** set this to **Autotimestamp** or **Current Time**. If **JSON Extract Fields** is set to **Yes**, you can select any option here.

Example

Using the values above, let's see how this sample file breaks up:

Sample Event - JSON Document (Array)

```
--- input ---
{"Records":[{"eventVersion":"1.05","eventTime":"2020-04-08T01:35:55Z","eventSource":"ec2.amazona
{"eventVersion":"1.05","eventTime":"2020-04-08T01:35:56Z","eventSource":"ec2.amazonaws.com","eve

--- output event 1 ---
{
  "_raw": "{\"eventVersion\":\"1.05\",\"eventTime\":\"2020-04-08T01:35:55Z\",\"eventSource\":\"e
  "_time": 1586309755,
  "cribl_breaker": "j-array"
}
```

```
--- output event 2 ---
{
  "_raw": "{\"eventVersion\":\"1.05\",\"eventTime\":\"2020-04-08T01:35:56Z\",\"eventSource\":\"e
  "_time": 1586309756,
  "cribl_breaker": "j-array"
}
```

JSON New Line Delimited

You can use the JSON New Line Delimited breaker to break and extract fields in newline-delimited JSON streams.

Example

Using default values, let's see how this sample stream breaks up:

Sample Event - Newline Delimited JSON Breaker

```
--- input ---
{"time":"2020-05-25T18:00:54.201Z","cid":"w1","channel":"clustercomm","level":"info","message":"
{"time":"2020-05-25T18:00:54.246Z","cid":"w0","channel":"clustercomm","level":"info","message":"


--- output event 1 ---
{
  "_raw": "{\"time\":\"2020-05-25T18:00:54.201Z\",\"cid\":\"w1\",\"channel\":\"clustercomm\",\"l
  "time": "2020-05-25T18:00:54.201Z",
  "cid": "w1",
  "channel": "clustercomm",
  "level": "info",
  "message": "metric sender",
  "total": 720,
  "dropped": 0,
  "_time": 1590429654.201,
}

--- output event 2 ---
{
  "_raw": "{\"time\":\"2020-05-25T18:00:54.246Z\",\"cid\":\"w0\",\"channel\":\"clustercomm\",\"l
  "time": "2020-05-25T18:00:54.246Z",
  "cid": "w0",
  "channel": "clustercomm",
  "level": "info",
  "message": "metric sender",
  "total": 720,
  "dropped": 0,
  "_time": 1590429654.246,
}
```

Timestamp

You can use the Timestamp breaker to break events at the beginning of any line in which LogStream finds a timestamp. This type enables breaking on lines whose timestamp pattern is not known ahead of time.

Example

Using default values, let's see how this sample stream breaks up:

Sample Event - Timestamp Based Breaker

```
--- input ---
{"level":"debug","ts":"2021-02-02T10:38:46.365Z","caller":"sdk/sync.go:42","msg":"Handle ENICon
{"level":"debug","ts":"2021-02-02T10:38:56.365Z","caller":"sdk/sync.go:42","msg":"Handle ENICon


--- output event 1 ---
{
  "_raw": "{\"level\":\"debug\",\"ts\":\"2021-02-02T10:38:46.365Z\",\"caller\":\"sdk/sync.go:42\
  "_time": 1612262326.365
}

--- output event 2 ---
{
  "_raw": "{\"level\":\"debug\",\"ts\":\"2021-02-02T10:38:56.365Z\",\"caller\":\"sdk/sync.go:42\
  "_time": 1612262336.365
}
```

CSV

The CSV breaker extracts fields in CSV streams that include a header line. Selecting this type exposes these extra fields:
- **Delimiter**: Delimiter character to use to split values. Defaults to: `,`
- **Quote Char**: Character used to quote literal values. Defaults to: `"`
- **Escape Char**: Character used to escape the quote character in field values. Defaults to: `"`

> **Example**: Using default values, let's see how this sample stream breaks up:

Example

Using default values, let's see how this sample stream breaks up:

Sample Event - CSV Breaker

```
--- input ---
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


--- output event 1 ---
{
  "_raw": "\"1611768713\",\"myHost1\",\"anet\",\"cisco\",\"ASN4204269\",\"11430\",\"43322\",\"0
  "time": "1611768713",
  "host": "myHost1",
  "source": "anet",
  "model": "cisco",
  "serial": "ASN4204269",
  "bytes_in": "11430",
  "bytes_out": "43322",
  "cpu": "0.78",
  "_time": 1611768713
}

--- output event 2 ---
{
  "_raw": "\"1611768714\",\"myHost2\",\"anet\",\"cisco\",\"ASN420423\",\"345062\",\"143433\",\"
  "time": "1611768714",
  "host": "myHost2",
  "source": "anet",
  "model": "cisco",
  "serial": "ASN420423",
  "bytes_in": "345062",
  "bytes_out": "143433",
  "cpu": "0.28",
  "_time": 1611768714
}
```

> With **Type: CSV** selected, an Event Breaker will properly add quotes around all values, regardless of their initial state.

Cribl versus Custom Rulesets

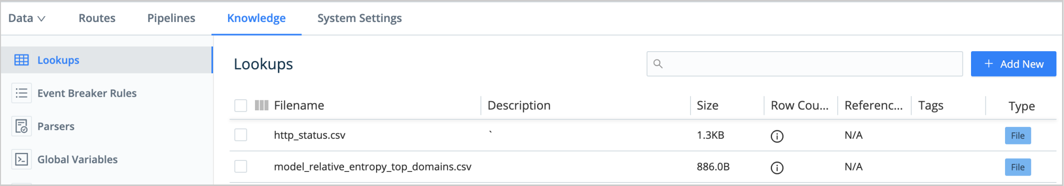
Event Breaker rulesets shipped by Cribl will be listed under the **Cribl** tag, while user-built rulesets will be listed under **Custom**. Over time, Cribl will ship more patterns, so this distinction allows for both sets to grow independently. In the case of an ID/Name conflict, the Custom pattern takes priority in listings and search.

Lookups Library

What Are Lookups

Lookups are data tables that can be used in Cribl LogStream to enrich events as they're processed by the Lookup Function. You can access the Lookups library under **Knowledge** > **Lookups**, which provides a management interface for all lookups.

This library is searchable, and each lookup can be tagged as necessary. There's full support for `.csv` files. Compressed files are supported, but must be in gzip format (`.gz` extension). You can add files in multimedia database (`.mmdb`) binary format, but you cannot edit these binary files through LogStream's UI.



□	Filename	Description	Size	Row Cou...	Referenc...	Tags	Type
□	http_status.csv	`	1.3KB	ⓘ	N/A		File
□	model_relative_entropy_top_domains.csv		886.0B	ⓘ	N/A		File

Lookups Library

How Does the Library Work

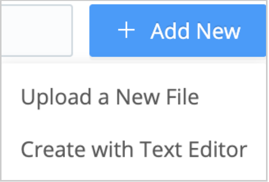
In single-instance deployments, all files handled by the interface are stored in `$CRIBL_HOME/data/lookups` . In distributed deployments, the storage path on the Leader Node is `$CRIBL_HOME/groups/<groupname>/data/lookups/` for each Worker Group.

> ℹ For large and/or frequently replicated lookup files, you might want to bypass the Lookups Library UI and instead manually place the files in a different location. This can both reduce deploy traffic and

> prevent errors with LogStream's default Git integration. For details, see Managing Large Lookups.

Adding Lookup Files

To upload or create a new lookup file, click **+ Add New**, then click the appropriate option from the drop-down.

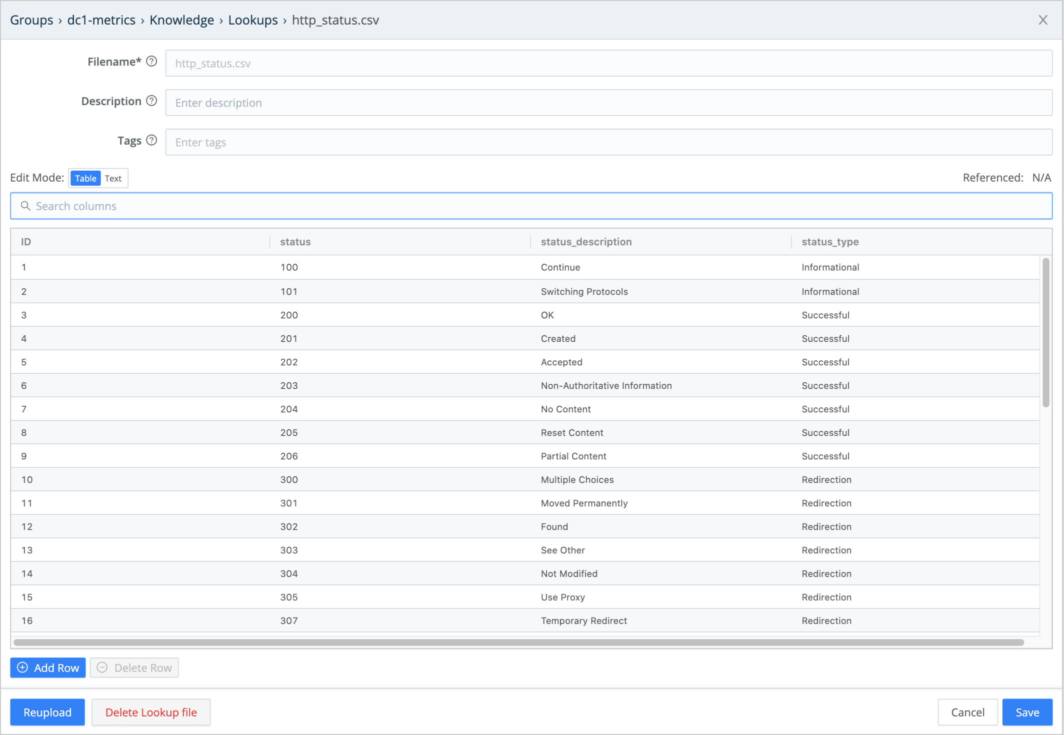


Adding a lookup file

Modifying Lookup Files

To re-upload, expand, edit, or delete an existing `.csv` or `.gz` lookup file, click its row on the Lookups page. (No editing option is available for `.mmdb` files; you can only re-upload or delete these.)

In the resulting modal, you can edit files in **Table** or **Text** mode. However, **Text** mode is disabled for files larger than 1 MB.


Groups › dc1-metrics › Knowledge › Lookups › http_status.csv

Filename* http_status.csv

Description Enter description

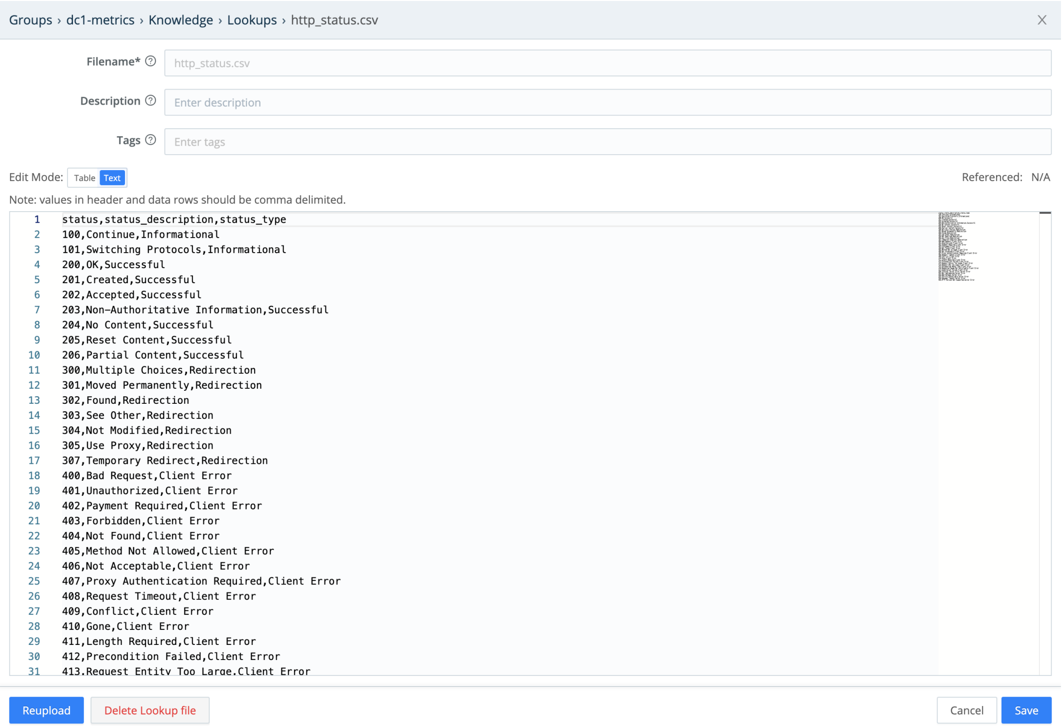
Tags Enter tags

Edit Mode: Table Text

Referenced: N/A

ID	status	status_description	status_type
1	100	Continue	Informational
2	101	Switching Protocols	Informational
3	200	OK	Successful
4	201	Created	Successful
5	202	Accepted	Successful
6	203	Non-Authoritative Information	Successful
7	204	No Content	Successful
8	205	Reset Content	Successful
9	206	Partial Content	Successful
10	300	Multiple Choices	Redirection
11	301	Moved Permanently	Redirection
12	302	Found	Redirection
13	303	See Other	Redirection
14	304	Not Modified	Redirection
15	305	Use Proxy	Redirection
16	307	Temporary Redirect	Redirection

Editing in table mode



Filename: http_status.csv

Note: values in header and data rows should be comma delimited.

```
status,status_description,status_type
100,Continue,Informational
101,Switching Protocols,Informational
200,OK,Successful
201,Created,Successful
202,Accepted,Successful
203,Non-Authoritative Information,Successful
204,No Content,Successful
205,Reset Content,Successful
206,Partial Content,Successful
300,Multiple Choices,Redirection
301,Moved Permanently,Redirection
302,Found,Redirection
303,See Other,Redirection
304,Not Modified,Redirection
305,Use Proxy,Redirection
307,Temporary Redirect,Redirection
400,Bad Request,Client Error
401,Unauthorized,Client Error
402,Payment Required,Client Error
403,Forbidden,Client Error
404,Not Found,Client Error
405,Method Not Allowed,Client Error
406,Not Acceptable,Client Error
407,Proxy Authentication Required,Client Error
408,Request Timeout,Client Error
409,Conflict,Client Error
410,Gone,Client Error
411,Length Required,Client Error
412,Precondition Failed,Client Error
413,Request Entity Too Large,Client Error
```

Editing in text mode

Memory Sizing for Large Lookups

For large lookup files, you'll need to provide extra memory beyond basic requirements for LogStream and the OS. To determine how much extra memory to add to a Worker Node for a lookup, use this formula:

```
Lookup file's uncompressed size (MB) * 2.25 (to 2.75) *
 numWorkerProcesses = Extra RAM required for lookup
```

For example, if you have a lookup file that is 1 GB (1,000 MB) on disk, and three Worker Processes, you could use an average 2.50 as the multiplier:

```
1,000 * 2.50 * 3 = 7,500
```

In this case, the Node's server or VM would need an extra 7,500 MB (7.5 GB) to accommodate the lookup file across all three worker processes.

What's with That Multiplier?

We've cited a squishy range of 2.25–2.75 for the multiplier, because we've found that it varies inversely with the number of columns in the lookup file:

- The fewer columns, the higher the extra memory overhead (2.75 multiplier).
- The more columns, the lower the overhead (2.25 multiplier).

In Cribl's testing:

- 5 columns required a multiplier of 2.75
- 10 columns required a multiplier of only 2.25.

These are general (not exact) guidelines, and this multiplier depends only on the lookup table's number of columns. The memory overhead imposed by each additional row appears to decline when more columns are present in the data.

Other Scenarios

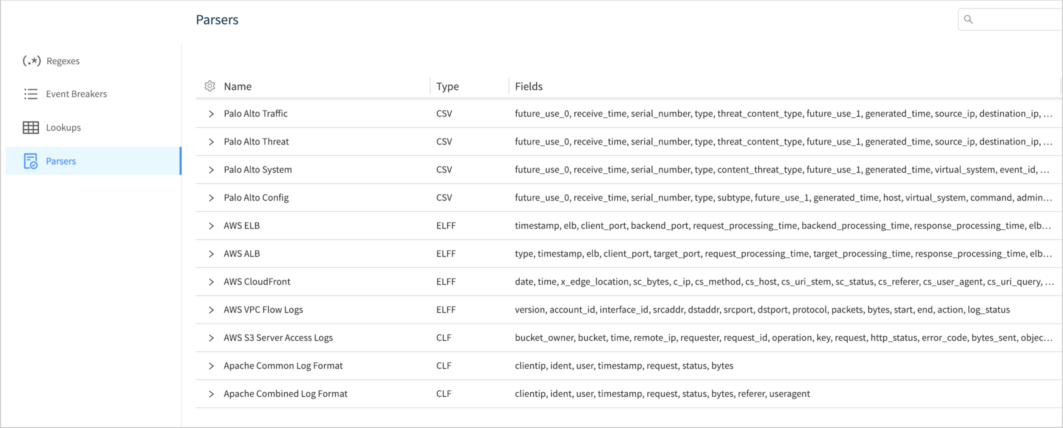
See also:

- Lookup Function.
- Ingest-time Lookups use case.
- Managing Large Lookups use case.
- Redis Function for faster lookups using a Redis integration (bypasses the Lookups Library).

Parsers Library

What Are Parsers

Parsers are definitions and configurations for the Parser Function. You can find the library under **Knowledge** > **Parsers**, and its purpose is to provide an interface for creating and editing Parsers. The library is searchable, and each parser can be tagged as necessary.



Name	Type	Fields
Palo Alto Traffic	CSV	future_use_0, receive_time, serial_number, type, threat_content_type, future_use_1, generated_time, source_ip, destination_ip, …
Palo Alto Threat	CSV	future_use_0, receive_time, serial_number, type, threat_content_type, future_use_1, generated_time, source_ip, destination_ip, …
Palo Alto System	CSV	future_use_0, receive_time, serial_number, type, content_threat_type, future_use_1, generated_time, virtual_system, event_id, …
Palo Alto Config	CSV	future_use_0, receive_time, serial_number, type, subtype, future_use_1, generated_time, host, virtual_system, command, admin…
AWS ELB	ELFF	timestamp, elb, client_port, backend_port, request_processing_time, backend_processing_time, response_processing_time, elb…
AWS ALB	ELFF	type, timestamp, elb, client_port, target_port, request_processing_time, target_processing_time, response_processing_time, elb…
AWS CloudFront	ELFF	date, time, x_edge_location, sc_bytes, c_ip, cs_method, cs_host, cs_uri_stem, sc_status, cs_referer, cs_user_agent, cs_uri_query, …
AWS VPC Flow Logs	ELFF	version, account_id, interface_id, srcaddr, dstaddr, srcport, dstport, protocol, packets, bytes, start, end, action, log_status
AWS S3 Server Access Logs	CLF	bucket_owner, bucket, time, remote_ip, requester, request_id, operation, key, request, http_status, error_code, bytes_sent, objec…
Apache Common Log Format	CLF	clientip, ident, user, timestamp, request, status, bytes
Apache Combined Log Format	CLF	clientip, ident, user, timestamp, request, status, bytes, referer, useragent

Parsers Library

Parsers can be used to **extract** or **reserialize** events. See Parser Function page for examples.

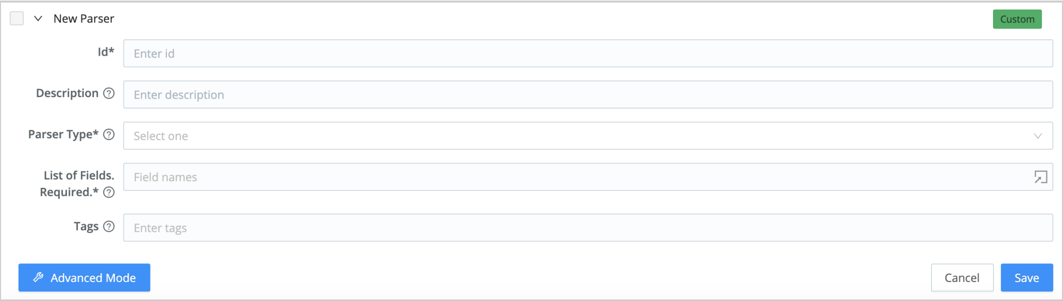
Supported Parser Types:

- CSV – Parse and reserialize comma-separated values.
- ELFF – Parse and reserialize events in Extended Log File Format.
- CLF – Parse and reserialize events in Common Log Format.

Creating a Parser

To create a parser, follow these steps:

1. Go to **Knowledge > Parsers** and click **Add New**.
2. Enter a unique **ID**.
3. Optionally, enter a **Description**.
4. Select a **Parser type** (see the supported types above).
5. Enter the **List of fields** expected to be extracted, in order.
   Click this field's Maximize icon (far right) if you'd like to open a modal where you can work with sample data and iterate on results.
6. Optionally, enter any desired **Tags**.



Adding a new parser

Schema Library

What Are Schemas

Schemas are JSON definitions that are used to validate of JSON events. They're based on the popular JSON Schema standard, and all schemas matching draft version 2019-09 are supported. You can find the library under **Knowledge** > **Schemas**. Its purpose is to provide an interface for creating, editing, and maintaining Schemas.

You validate a schema using the `C.Schema('<schema name>').validate(<object field>)` built-in method. This function can be called anywhere in Cribl LogStream that JavaScript expressions are supported.

Typical use cases for Schema validation:

- Making a decision before sending an event down to a destination.
- Making a decision before accepting an event. (E.g., drop an event if invalid.)
- Making a decision to route an event based on the result of validation.

Example

To add this example JSON Schema, go to **Knowledge** > **Schemas** and click **+ Add New**. Enter the following:

- 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
    }
  }
}
```

Assume that events look like this:

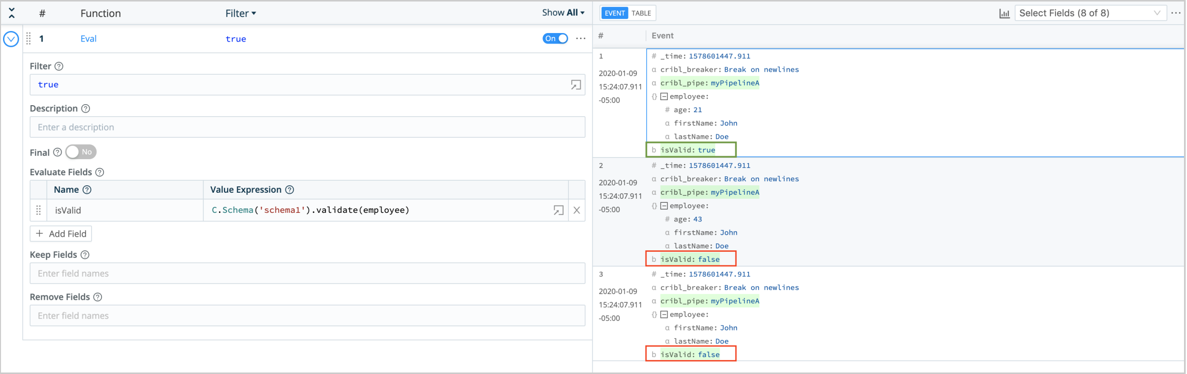
```
Events

{"employee":{"firstName": "John", "lastName": "Doe", "age": 21}}
{"employee":{"firstName": "John", "lastName": "Doe", "age": 43}}
{"employee":{"firstName": "John", "lastName": "Doe"}}
```

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.



1 Eval true On

Filter: true

Description: Enter a description

Final: No

Evaluate Fields

Name	Value Expression
isValid	C.Schema('schema1').validate(employee)

Keep Fields: Enter field names

Remove Fields: Enter field names

```
1   2020-01-09 15:24:07.911 -05:00
    # _time: 1578601447.911
    a cribl_breaker: Break on newlines
    a cribl_pipe: myPipelineA
    {} ⊟ employee:
        # age: 21
        a firstName: John
        a lastName: Doe
    b isValid: true
2   2020-01-09 15:24:07.911 -05:00
    # _time: 1578601447.911
    a cribl_breaker: Break on newlines
    a cribl_pipe: myPipelineA
    {} ⊟ employee:
        # age: 43
        a firstName: John
        a lastName: Doe
    b isValid: false
3   2020-01-09 15:24:07.911 -05:00
    # _time: 1578601447.911
    a cribl_breaker: Break on newlines
    a cribl_pipe: myPipelineA
    {} ⊟ employee:
        a firstName: John
        a lastName: Doe
    b isValid: false
```

Schema validation results for the above events

Global Variables Library

What Are Global Variables

Global Variables are reusable JavaScript expressions that can be accessed in Functions in any Pipeline. You can access the library under **Knowledge** > **Global Variables**.

Typical use cases for Global 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**.

Global Variables can be of the following types:

- Number
- String
- Boolean
- Object
- Array
- Expression

Global Variables can be accessed via `C.vars.` – which can be called anywhere in Cribl LogStream that JS expressions are supported. Typeahead is provided. More on Cribl Expressions here.

Examples

Scenario 1:

Assign field `foo` the value in `theAnswer` Global Variable.

- Global Variable Name: `theAnswer` <-- ships with Cribl LogStream by default.
- Global Variable Value: `42`
- **Sample Eval Function:** `foo = C.vars.theAnswer`

Scenario 2:

Assign field `nowEpoch` the current time, in epoch format.

- Global Variable Name: `epoch` <-- ships with Cribl LogStream by default.
- Global Variable Value: `Date.now()/1000`
- **Sample Eval Function:** `nowEpoch = C.vars.epoch()`

Scenario 3:

Create a new field called `storage` , by converting the value of event field `size` to human-readable format.

- Global Variable Name: `convertBytes` <-- ships with Cribl LogStream by default

- Global Variable Value: ``` `${Math.round(bytes / Math.pow(1024, (Math.floor(Math.log(bytes) / Math.log(1024)))), 2)}${['Bytes', 'KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB'][(Math.floor(Math.log(bytes) / Math.log(1024)))]}` ```

  Note the use of quotes or backticks around values. Use the opposite delimiter for the enclosing expression.

- Global Variable Argument: `bytes`

- **Sample Eval Function:** `storage = C.vars.convertBytes(size)`

  Note the use of `bytes` here as an argument.

TECHNIQUES & TIPS

Tips and Tricks

In designing, supporting, and troubleshooting, complex LogStream deployments and workflows, we've found the following general principles helpful.

> Every use case is different. Don't hesitate to contact Cribl Support for help with solving your specific needs.

Architecture and Deployment

Separate Data by Worker Groups, Routes, or Both?

Group data volume either by Worker Groups, (data isolation by business groups, data privacy, or as a result of saving costs due to geo-isolation are common business reasons) or by Routes, or both. Also a consideration is – which way is the easiest to understand and manage as the company grows?

Use TCP JSON to Link LogStream Instances/Groups

When sending between LogStream instances and/or Worker Groups, use TCP JSON. While LogStream 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.

Input Side: Event Breakers and Timestamping

Validate timestamping and event breaking before turning on a Source.

If a Source supports Event Breakers (e.g., AWS Sources), it is much more efficient to perform JSON unroll in a Breaker, versus in a Pipeline Function.

Routes

Avoid Route Creep

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 wasted processing cycles and testing against whether filter criteria are met.

Prevent Function Creep

As a mirror principle, clone events as late as possible in the Routes. This will minimize the number of Functions acting on data, to the extent possible.

Leave No Data Behind

Create a catchall Route at the end of the list to explicitly route events that fail to match any Route filters.

Pipelines and Functions Logic

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 Function Groups for Legibility

Function groups might be helpful in organizing your Pipeline. These groups are abstractions, purely for visual context, and do not affect the movement of data through Functions. Data will move down the listed Functions, ignoring any grouping assignments.

Use Comments to Preserve Legibility

Comment, comment, comment. There is a lot of contextual information which might become lost over time as users continue to advance and add Routes and Pipelines to Cribl LogStream. 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 and Pipeline in comments as clearly as possible.

Create Expressions Around Fields with Non-Alphanumeric Characters

If there are fields with non-alphanumeric characters – e.g., `@timestamp` or `user-agent` or `kubernetes.namespace_name` – you can access them using `__e['<field-name-here>']`. (Note the single quotes.) For more details, see MDN's Property Accessors documentation. In any other place where such fields are referenced – e.g., in an Eval Function's Field names – you should use a single-quoted literal, of the form: `'<field-name-here>'`.

Specify Fields' Precedence Order in Expressions

In any Source that supports adding **Fields (Metadata)**, your **Value** expression can specify that fields in events should override these fields' values. E.g., the following expression's L->R/OR logic specifies that if an inbound event includes an `index` field, use that field's value; otherwise, fall back to the `myIndex` constant defined here: ``` `${__e['index'] || 'myIndex'}` ```.

Break Up `_raw`

Consider avoiding the use of `_raw` as a temporary location for data. Instead, split out explicitly separate fields/variables.

Optimize PQ Using File Size

Consider using a smaller maximum file size in Persistent Queues settings, for better buffering.

Output Side

Although this might be obvious: Ship metrics out to a dedicated alerting/metrics engine (ELK, Grafana, Splunk, etc.)

Troubleshooting

Troubleshoot streams processing systems from right to left. Start at the Destination, and check for block status from the Destination back to the Source.

Don't run health checks on data ports too frequently, as this can lead to false-positives errors.

Ingest-time Fields

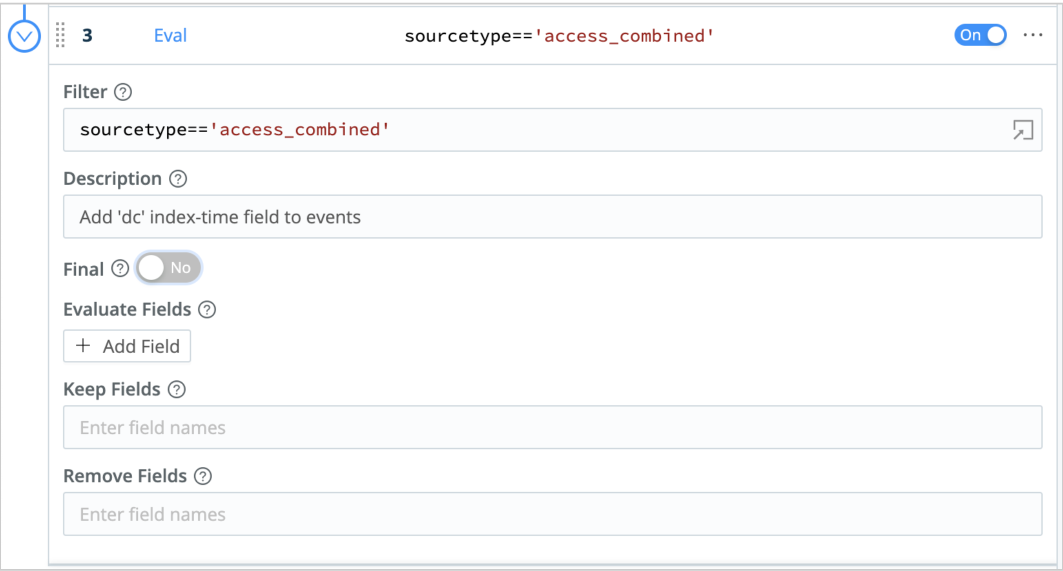
Adding Fields to Data in Motion

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 a **Eval** function to it:



3 Eval `sourcetype=='access_combined'` On

Filter: `sourcetype=='access_combined'`

Description: Add 'dc' index-time field to events

Final: No

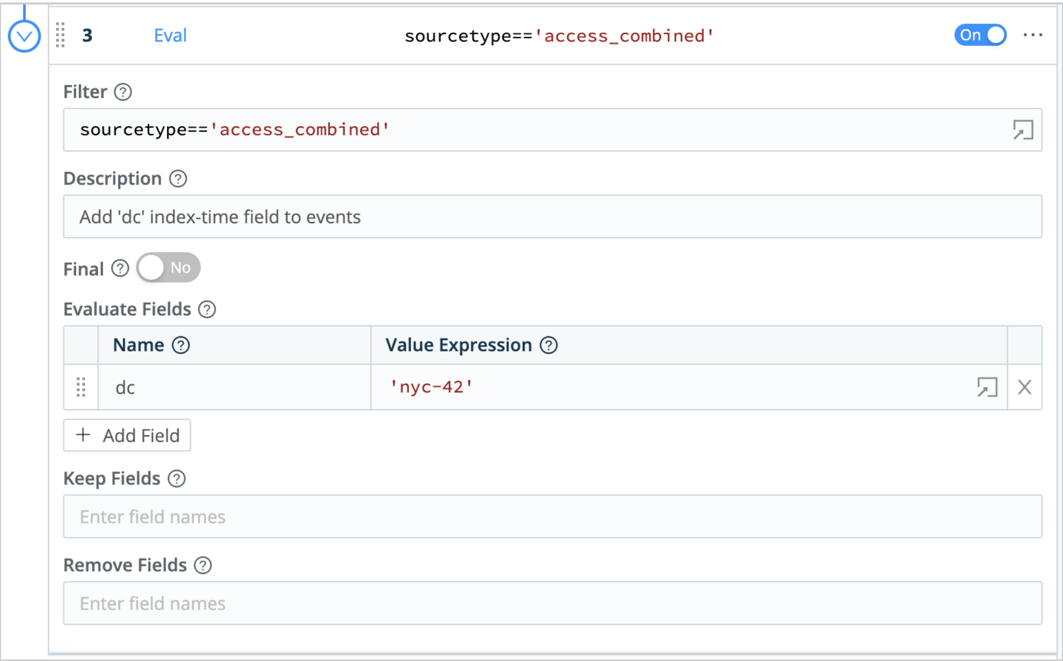
Evaluate Fields

Keep Fields

Remove Fields

Defining the Eval Function's filter expression

- Next, let's click on **+ Add Field**, add our `dc` field, and click **Save**.



3 Eval sourcetype=='access_combined' On

Filter: `sourcetype=='access_combined'`

Description: Add 'dc' index-time field to events

Final: No

Evaluate Fields

Name	Value Expression
dc	'nyc-42'

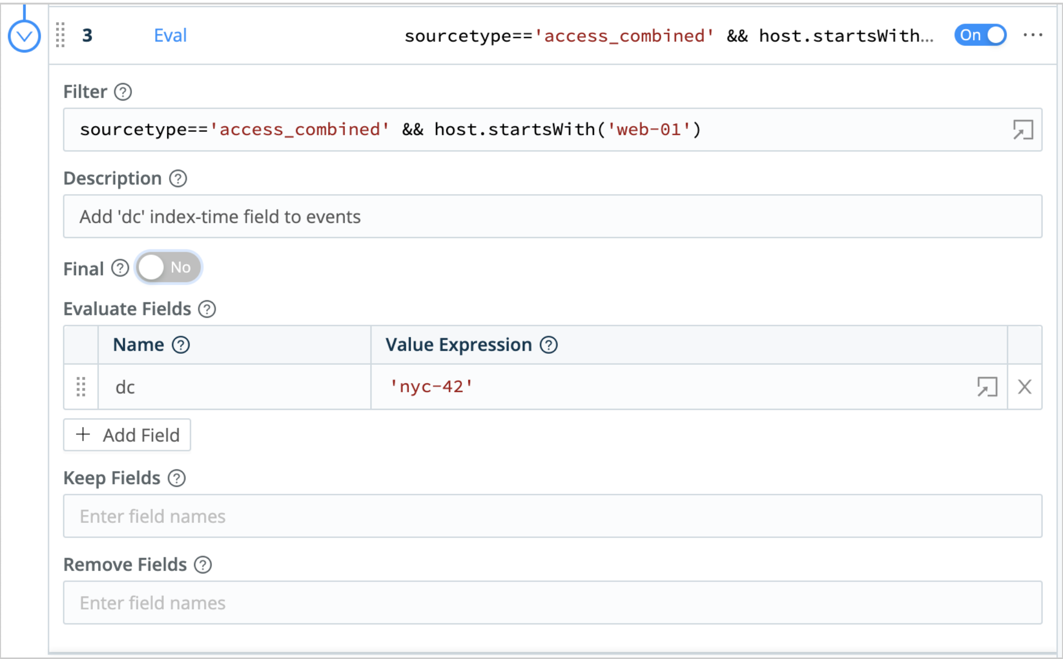
Keep Fields: Enter field names

Remove Fields: Enter field names

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 as below:



3 Eval sourcetype=='access_combined' && host.startsWith... On

Filter: `sourcetype=='access_combined' && host.startsWith('web-01')`

Description: Add 'dc' index-time field to events

Final: No

Evaluate Fields

Name	Value Expression
dc	'nyc-42'

Keep Fields: Enter field names

Remove Fields: Enter field names

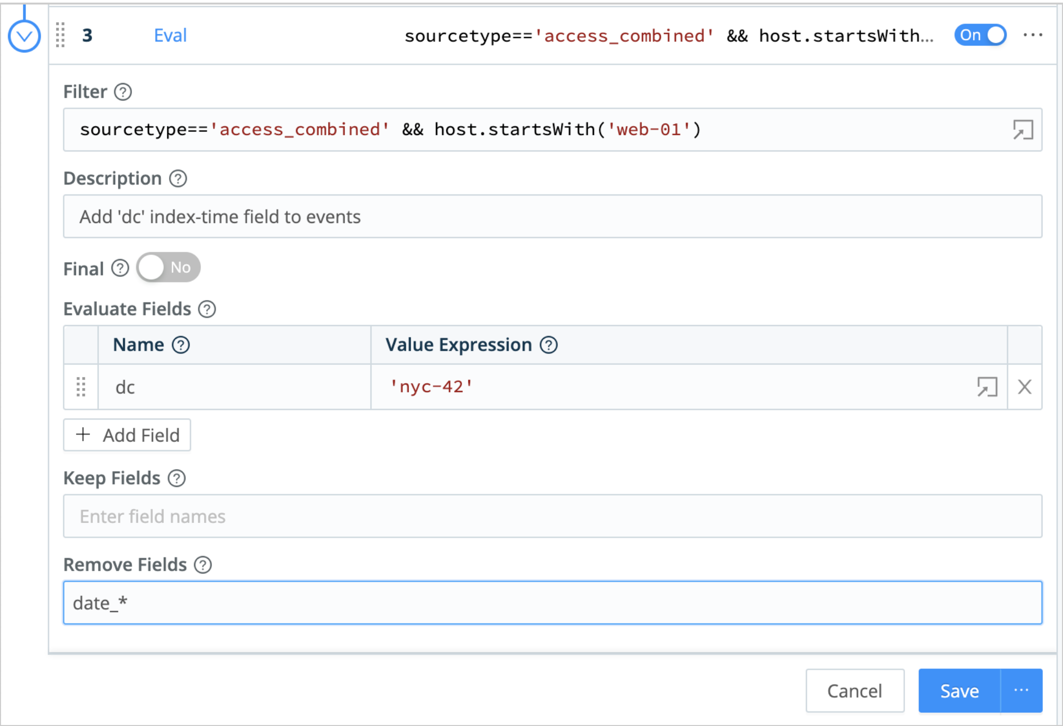
Refining the filter

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 a **Eval** function to it (as above).
- Next, in **Remove Fields**, add `date_*` and hit Save.



3 Eval `sourcetype=='access_combined' && host.startsWith…` On

Filter: `sourcetype=='access_combined' && host.startsWith('web-01')`

Description: Add 'dc' index-time field to events

Final: No

Evaluate Fields

Name	Value Expression
dc	`'nyc-42'`

Keep Fields: Enter field names

Remove Fields: date_*

Goodbye `date_` field

To confirm, verify that this search: `sourcetype="access_combined" date_minute=*` will soon stop returning results. Enjoy a more efficient Splunk!

Ingest-time Lookups

Enriching Data in Motion

To enrich events with new fields from external sources (such as `.csv` files), we use LogStream's out-of-the-box Lookup Function. Ingestion-time lookups are not only great for normalizing field names and values, but also ideal for use cases where:

- Fast access via the looked-up value is required. For example, when you don't have a `datacenter` field in your events, but you do have a `host-to-datacenter` map, and you need to search by `datacenter`.
- Looked-up information must be temporally correct. For example, assume that you have a highly dynamic infrastructure, and you need to resolve a resource name (e.g., a container name) to its address. You can't afford to defer this to search time/runtime, as the resource and its records might no longer exist.

> To use large binary databases (like GeoIP `.mmdb` files) for LogStream lookups, see Managing Large Lookups. To achieve faster lookups, use LogStream's Redis Function.

Working with Lookups – Example 1

Let's assume we have the following lookup file. Given the field `conn_state` in an event, we would like to add a corresponding ingestion-time field called `action`.

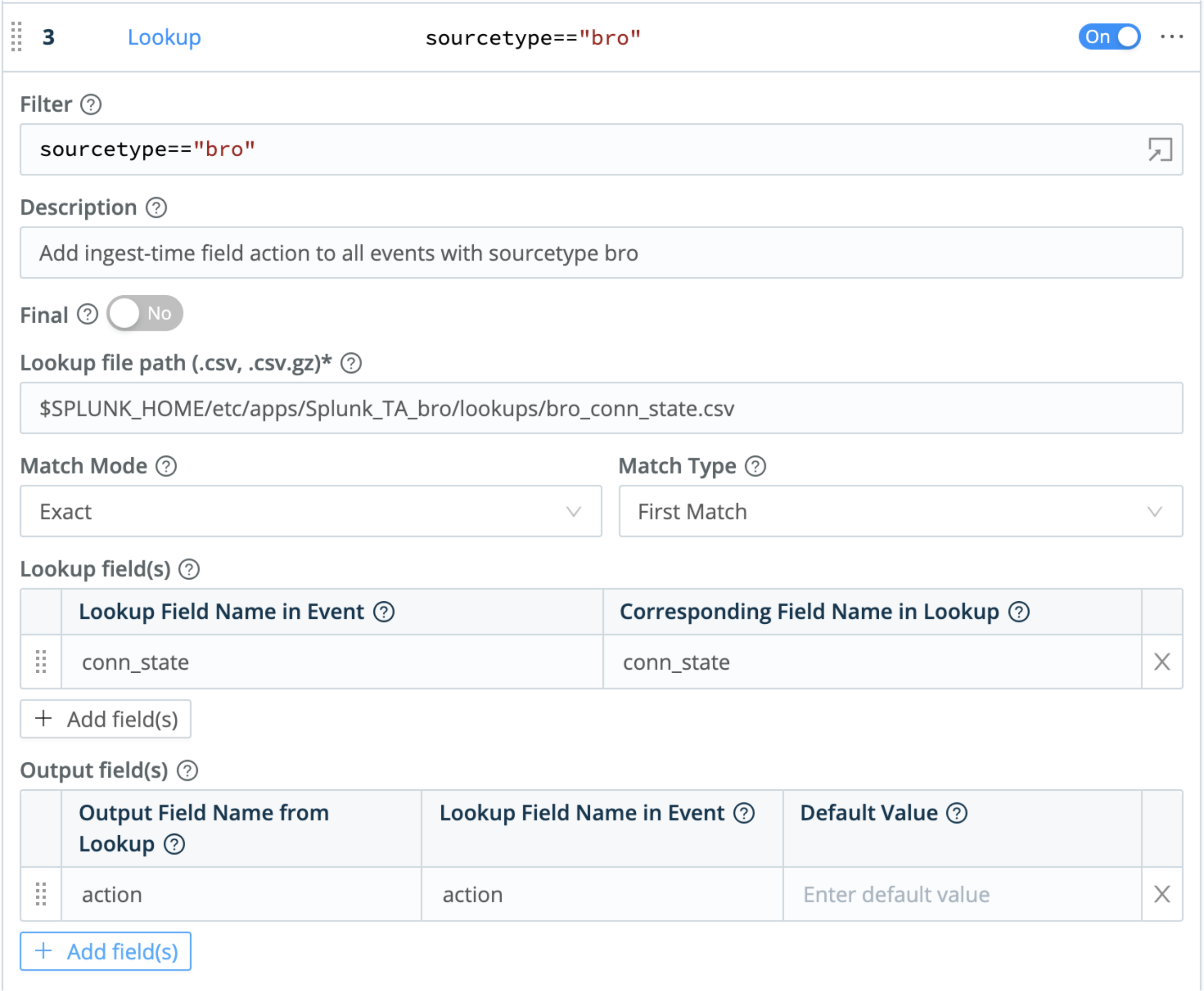
bro_conn_state.csv

```
action,"conn_state","conn_state_meaning"
dropped,S0,"Connection attempt seen, no reply."
allowed,S1,"Connection established, not terminated."
allowed,SF,"Normal establishment and termination."
blocked,REJ,"Connection attempt rejected."
allowed,S2,"Connection established and close attempt by originator seen (but no reply from respo
allowed,S3,"Connection established and close attempt by responder seen (but no reply from origi
allowed,RSTO,"Connection established, originator aborted (sent a RST)."
allowed,RSTR,"Established, responder aborted."
dropped,RSTOS0,"Originator sent a SYN followed by a RST, we never saw a SYN-ACK from the respon
dropped,RSTRH,"Responder sent a SYN ACK followed by a RST, we never saw a SYN from the (purporte
dropped,SH,"Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (
dropped,SHR,"Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originator
allowed,OTH,"No SYN seen, just midstream traffic (a 'partial connection' that was not later clos
```

First, make sure you have a Route and Pipeline configured to match desired events.

Next, let's add a **Lookup** function to the Pipeline, with these settings:

- **Lookup file path**: `$SPLUNK_HOME/etc/apps/Splunk_TA_bro/lookups/bro_conn_state.csv` (note that Environment variables are allowed in the path).
- **Lookup Field Name in Event** set to `conn_state`.
- **Corresponding Field Name in Lookup** set to `conn_state`.
- **Output Field Name from Lookup** set to `action`.
- **Lookup Field Name in Event** set to `action`.



3 Lookup sourcetype=="bro" On

Filter

`sourcetype=="bro"`

Description

Add ingest-time field action to all events with sourcetype bro

Final: No

Lookup file path (.csv, .csv.gz)*

$SPLUNK_HOME/etc/apps/Splunk_TA_bro/lookups/bro_conn_state.csv

Match Mode: Exact

Match Type: First Match

Lookup field(s)

Lookup Field Name in Event	Corresponding Field Name in Lookup
conn_state	conn_state

Output field(s)

Output Field Name from Lookup	Lookup Field Name in Event	Default Value
action	action	Enter default value

Lookup Function to add `action` field

To confirm success, verify that this search returns expected results: `sourcetype="bro" action::allowed`. Change the `action` value as necessary.

Working with Lookups – Example 2

Let's assume we have the following lookup file, and given **both** the fields `impact` and `priority` in an event, we would like to add a corresponding ingestion-time field called `severity`.

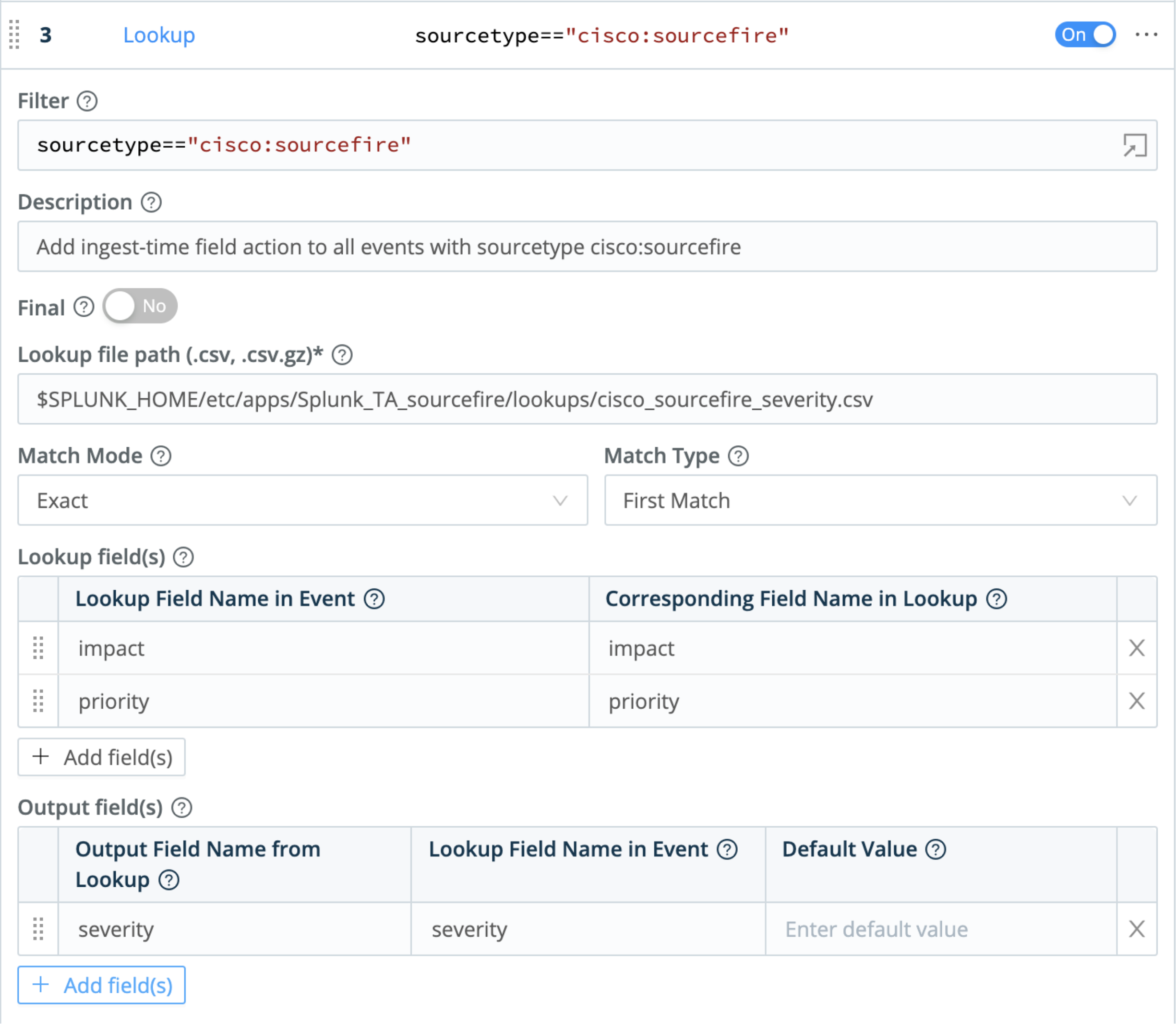
cisco_sourcefire_severity.csv

```
impact,priority,severity
1,high,critical
2,high,critical
3,high,high
4,high,high
0,high,high
"*",high,high
.....
"*",medium,medium
1,low,medium
2,low,medium
3,low,low
4,low,low
0,low,low
"*",low,low
1,none,low
2,none,low
3,none,informational
4,none,informational
0,none,informational
"*",none,informational
```

First, make sure you have a Route and Pipeline configured to match desired events.

Next, let's add a **Lookup** function to the Pipeline, with these settings:

- **Lookup file path**: `$SPLUNK_HOME/etc/apps/Splunk_TA_sourcefire/lookups/cisco_sourcefire_severity.csv` (note that Environment variables are allowed in the path).
- **Lookup Field Name(s) in Event** set to `impact` and `priority` .
- **Corresponding Field Name(s) in Lookup** set to `impact` and `priority` .
- **Output Field Name from Lookup** set to `severity` .
- **Lookup Field Name in Event** set to `severity` .



3 Lookup `sourcetype=="cisco:sourcefire"` On

Filter

```
sourcetype=="cisco:sourcefire"
```

Description: Add ingest-time field action to all events with sourcetype cisco:sourcefire

Final: No

Lookup file path (.csv, .csv.gz)*: $SPLUNK_HOME/etc/apps/Splunk_TA_sourcefire/lookups/cisco_sourcefire_severity.csv

Match Mode: Exact

Match Type: First Match

Lookup field(s)

Lookup Field Name in Event	Corresponding Field Name in Lookup
impact	impact
priority	priority

Output field(s)

Output Field Name from Lookup	Lookup Field Name in Event	Default Value
severity	severity	Enter default value

Lookup Function to add `severity` field

To confirm success, verify that this search returns expected results: `sourcetype="cisco:sourcefire" severity::medium`. Change the `severity` value as necessary.

Sampling

Sampling at Ingest-Time

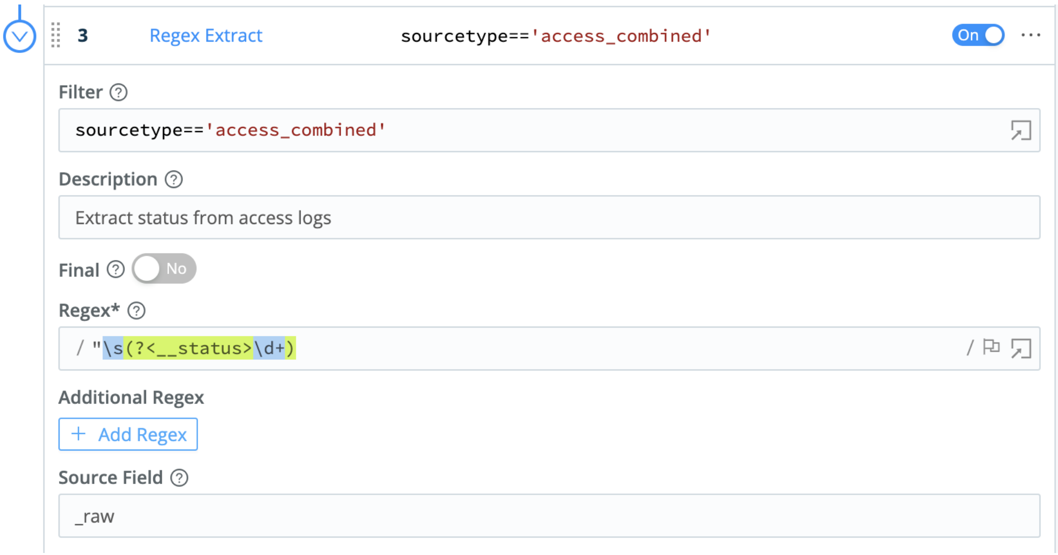
Let's say that you wanted to analyze and troubleshoot with **highly verbose/voluminous** data – for example, CDN logs, ELB Access Logs, or VPC Flows – but you were concerned about storage requirements and search performance. With Sampling, you can bring in enough samples that your analysis remains statistically 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 LogStream, and can be used anywhere in a Pipeline.



3 Regex Extract sourcetype=='access_combined' On

Filter: `sourcetype=='access_combined'`

Description: Extract status from access logs

Final: No

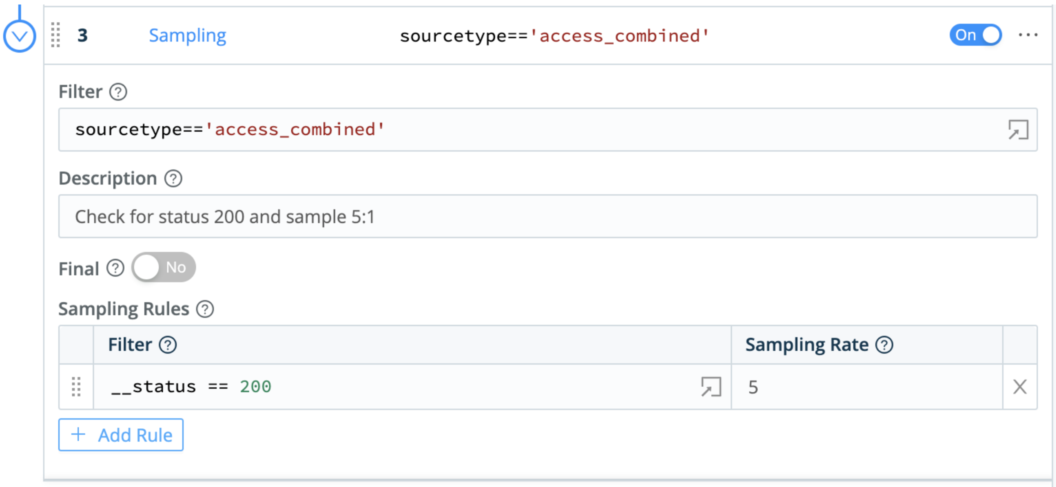
Regex*: `/ "\s(?<__status>\d+)`

Additional Regex

Source Field: _raw

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`.



3 Sampling sourcetype=='access_combined' On

Filter: `sourcetype=='access_combined'`

Description: Check for status 200 and sample 5:1

Final: No

Sampling Rules

Filter	Sampling Rate
`__status == 200`	5

Sampling success responses

To confirm that sampling works, compare the event count of all `200`s before and after.

> **i** 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.

Access Logs: Apache, ELB, CDN, S3, etc.

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 (storage, resources, license, etc.) that most users don't even bother. However, some of the logs contain relevant information when looked at individually (e.g., errors). The much larger majority contains relevant information when looked at in the aggregate (e.g., successes to determine traffic patterns, etc.).

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.

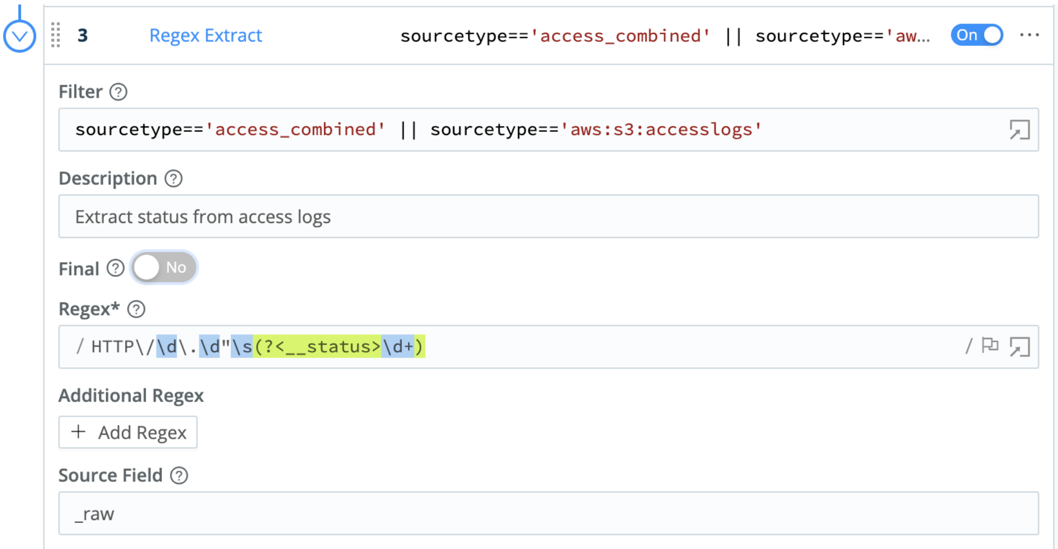
Using `status` as the Sampling Condition

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+)/`



3 Regex Extract sourcetype=='access_combined' || sourcetype=='aw... On

Filter
sourcetype=='access_combined' || sourcetype=='aws:s3:accesslogs'

Description
Extract status from access logs

Final No

Regex*
```
/ HTTP\/\d\.\d"\s(?<__status>\d+) /
```

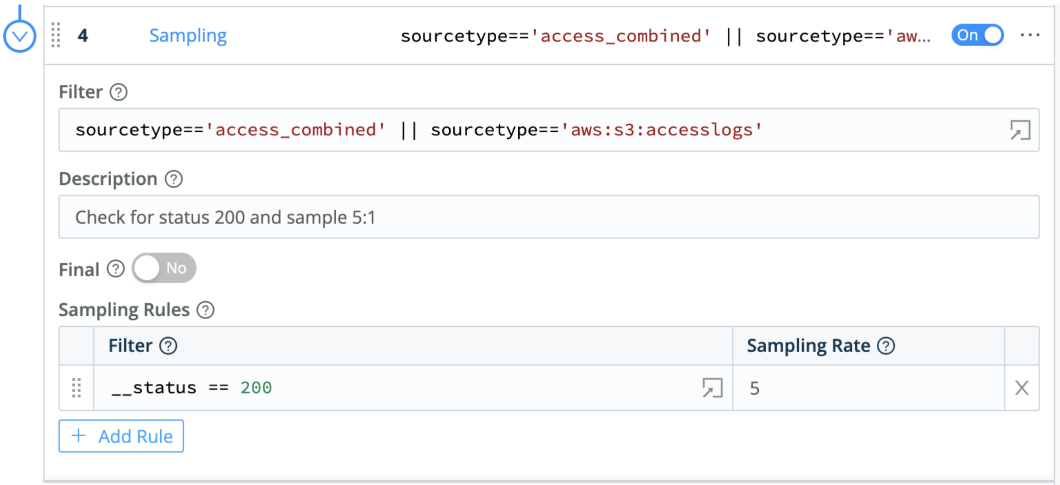
Additional Regex
+ Add Regex

Source Field
_raw

Defining the Regex Extract Function

3. Add a Sampling Function to sample `5:1` when `__status==200`.

4. Save.



4 Sampling sourcetype=='access_combined' || sourcetype=='aw... On

Filter
sourcetype=='access_combined' || sourcetype=='aws:s3:accesslogs'

Description
Check for status 200 and sample 5:1

Final No

Sampling Rules

Filter	Sampling Rate
__status == 200	5

+ Add Rule

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'`
Amazon ELB Access Logs	`sourcetype=='aws:elb:accesslogs'`

Firewall Logs: VPC Flow Logs, Cisco ASA, Etc.

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, pfSense, etc.

As with Access Logs, bringing in **everything** for operational analysis might be cost-prohibitive. But sampling with Cribl LogStream 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.

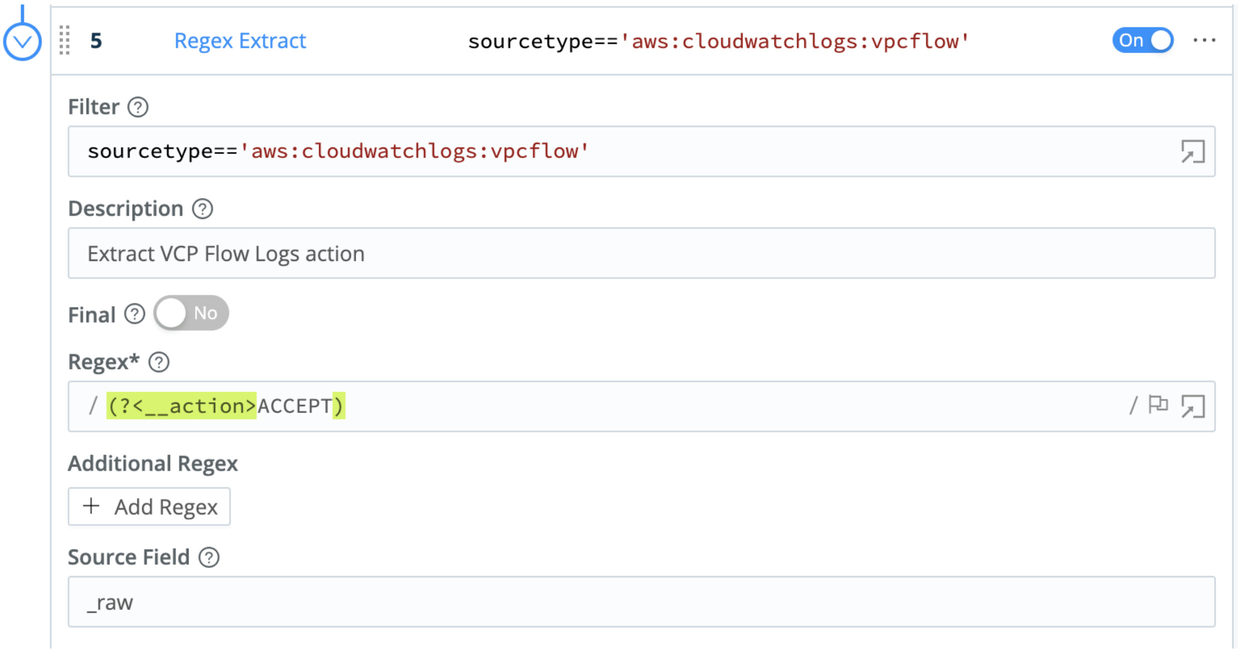
Typical VPC Flow Logs look like this:

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 1418530070
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1418530010 1418530070 R
```

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)/`



5 Regex Extract `sourcetype=='aws:cloudwatchlogs:vpcflow'` On

Filter: `sourcetype=='aws:cloudwatchlogs:vpcflow'`

Description: Extract VCP Flow Logs action

Final: No

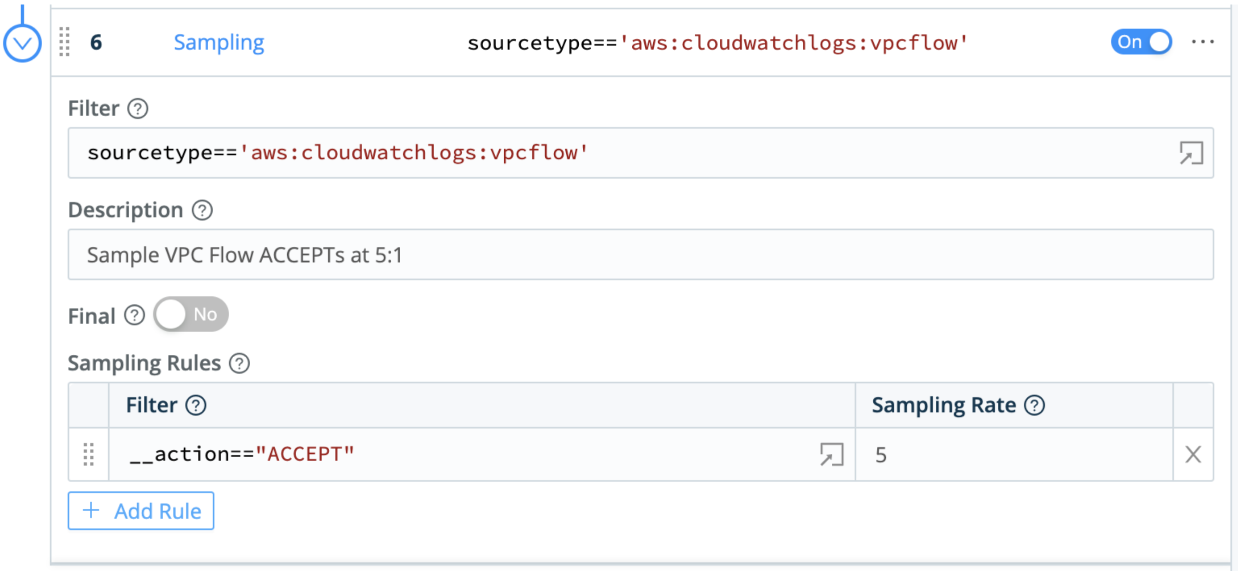
Regex*: `/(?<__action>ACCEPT)/`

Additional Regex

Source Field: `_raw`

Extracting the `__action` field

3. Add a **Sampling** Function to sample `5:1` when `__action=="ACCEPT"` .
4. Save.



6 Sampling `sourcetype=='aws:cloudwatchlogs:vpcflow'` On

Filter: `sourcetype=='aws:cloudwatchlogs:vpcflow'`

Description: Sample VPC Flow ACCEPTs at 5:1

Final: No

Sampling Rules

Filter	Sampling Rate
`__action=="ACCEPT"`	5

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:

Sourcetype	Filter Expression
Cisco ASA Logs	`sourcetype=='cisco:asa'`

Related sourcetypes to consider sampling:	`sourcetype=='cisco:fwsm'`
	`sourcetype=='cisco:pix'`

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.

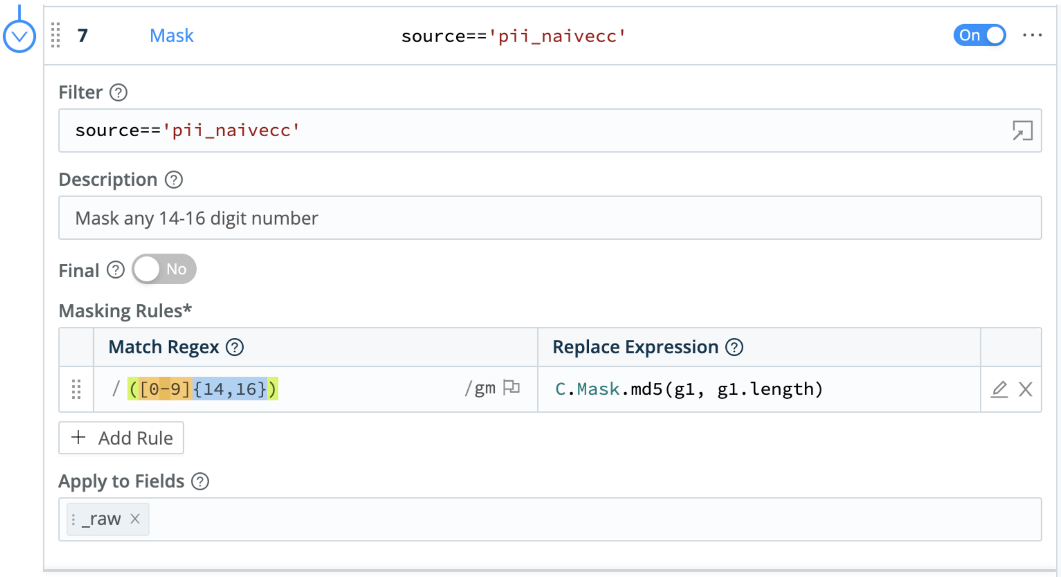
Replace Expression is a JS expression or literal to replace matched content.

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:



source=='pii_naivecc'

Filter

```
source=='pii_naivecc'
```

Description

Mask any 14-16 digit number

Final No

Masking Rules*

Match Regex	Replace Expression
`/ ([0-9]{14,16}) /gm`	`C.Mask.md5(g1, g1.length)`

Apply to Fields

_raw

Defining the replacement length

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.

Managing Large Lookups

This page offers a general approach to managing large lookup files. While LogStream's Git integration normally helps manage configuration changes, large lookups are exceptions. In many cases, you might want to exclude these files from Git, to reduce excessive deploy traffic. This approach can also prevent Git Push commands from encountering large file errors.

Good scenarios for this approach are:

- Large binary files – like databases – which don't benefit from Git's typical efficient storage of only the deltas between versions. (With binary files, Git must replace the whole file for each new version.)
- Files updated frequently and/or files updated independent of LogStream.
- Files replicated on many Worker Nodes.

About the MaxMind GeoLite Example

We'll illustrate this with an example that often combines all three conditions: setting up the free, popular MaxMind GeoLite2 City database to support LogStream's GeoIP lookup Function. This example anticipates a LogStream 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.

Reducing Deploy Traffic

The general approach for handling large lookups is:

- Do not place these files in the standard `$CRIBL_HOME/data/lookups`.
- Instead, place them in a `$CRIBL_HOME` subdirectory that's excluded from Git version control, through inclusion in the `$CRIBL_HOME/.gitignore` file. Deploying the files to the Leader Node and all desired Workers will require a manual procedure and will be required for the initial deployment as well as subsequent updates.

The example below uses `$CRIBL_HOME/state` subdirectory, which is already listed in the default `.gitignore` file that ships with LogStream.

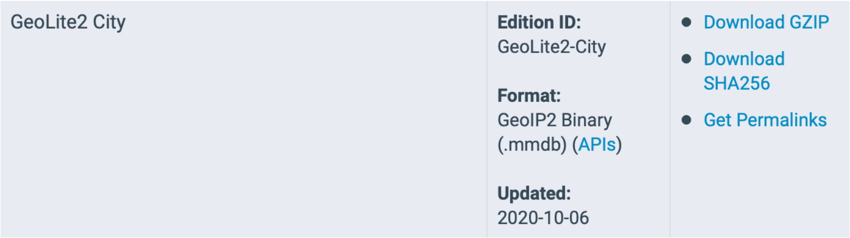
> If you prefer, you can use a different path, including a path outside `$CRIBL_HOME`. If you choose this alternative, be sure to add that path to `.gitignore`.
>
> However, Cribl recommends using a `$CRIBL_HOME` subdirectory like `$CRIBL_HOME/state`, because this inherits appropriate permissions and simplifies backup/restore operations.

Let's move on to the MaxMind GeoLite specifics.

Download and Extract the Database

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	**Edition ID:** GeoLite2-City **Format:** GeoIP2 Binary (.mmdb) (APIs) **Updated:** 2020-10-06	• Download GZIP • Download SHA256 • Get Permalinks

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`

Copy the Database File to the Leader and Worker Nodes (Recommended)

In distributed deployments, Cribl recommends copying the MaxMind database separately to the Leader and all Worker Nodes, e.g.. placing it in the `$CRIBL_HOME/state` path. This will minimize the Git commit/deploy overhead around nightly updates to the binary database file.

Once in the database's directory, execute commands of this form:

Shell

```shell
$ scp *.mmdb <user>@<master-node>:
$ scp *.mmdb <user>@<worker-node>:
```

> ⚠ Copy the file to each Worker in the Worker Group where you intend to use LogStream's GeoIP Function.

The above commands copy the `.mmdb` database file into your user's home directory on each Node. Next, we’ll move it to `$CRIBL_HOME/state` on each Node. Execute these commands on both the Leader and Worker Nodes:

Shell

```shell
$ sudo mv ~/*.mmdb <$CRIBL_HOME>/state/
$ sudo chown -R cribl:cribl <$CRIBL_HOME>/state/
```

Now that the database is in place, your Pipelines can use the GeoIP Function to enrich data. In the Function's **GeoIP file (.mmdb)** field, insert the complete `$CRIBL_HOME/state/<filename>.mmdb` file path.

Copy the Database File Only to the Leader (Alternative)

In smaller deployments, you might choose to copy this MaxMind database only to the Leader Node, and to let Workers receive updates via Git commit/deploy. In this case, the final commands above might look like this:

Shell

```
$ sudo cp ~/*.mmdb /opt/cribl/groups/<group-name>/data/lookups/
$ cd /opt/cribl/groups/<group-name>/data/lookups/
$ sudo chown cribl:cribl *.mmdb
```

Automatic Updates to the MaxMind Database

To set up automatic updates, see MaxMind's Automatic Updates for GeoIP2 and GeoIP Legacy Databases documentation. You'll need two modifications specific to LogStream:

- This must be set up on the Leader, and on each Worker in any Group that uses GeoIP lookups.
- 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/`.

Memory Considerations

Storage aside, large lookup files can also require additional RAM on each Worker Node that processes the lookups. For details, see Memory Sizing for Large Lookups.

Lookups as Filters for Masks

Overview

You can make your data architecture more maintainable by using Lookups to route and transform events within Cribl LogStream. 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 LogStream 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 LogStream:

index_tracker.csv

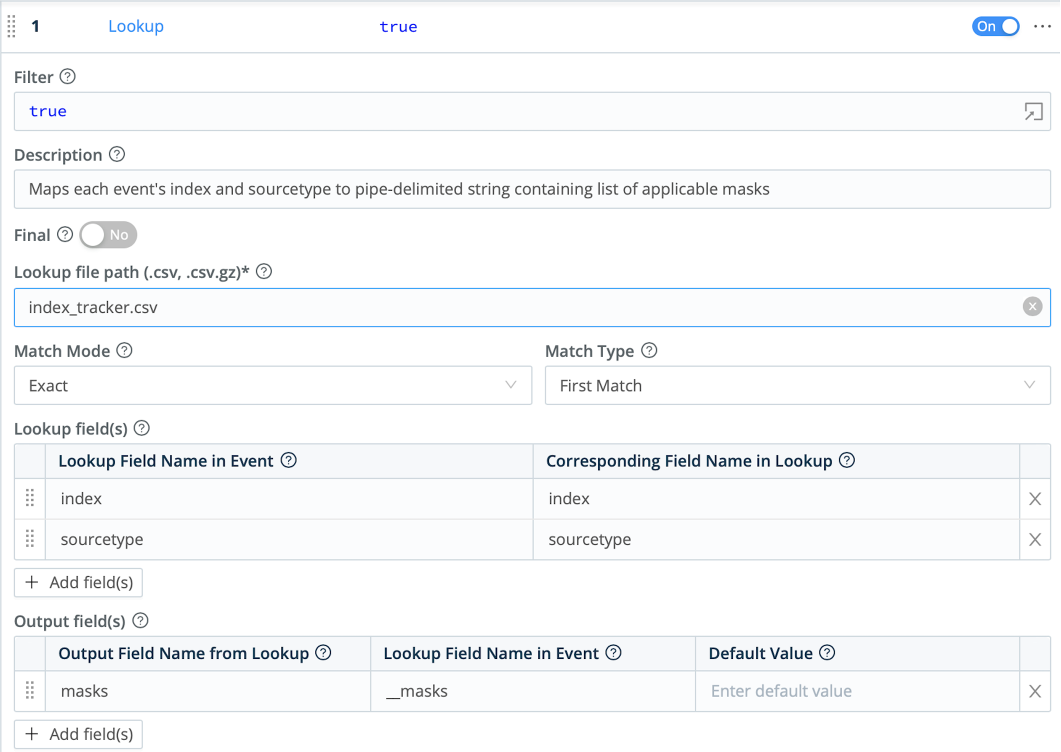
```
index,sourcetype,masks
apache_common, sourcetypec, ssn|credit_card|auth_token
syslog,sourcetypeb,ssn|auth_token
weblog,sourcetypea,auth_token|bearer_token
```

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 LogStream Pipeline with a Lookup Function configured like this, pointing to your lookup table:



1 Lookup `true` On

Filter: `true`

Description: Maps each event's index and sourcetype to pipe-delimited string containing list of applicable masks

Final: No

Lookup file path (.csv, .csv.gz)*: index_tracker.csv

Match Mode: Exact

Match Type: First Match

Lookup field(s)

Lookup Field Name in Event	Corresponding Field Name in Lookup
index	index
sourcetype	sourcetype

+ Add field(s)

Output field(s)

Output Field Name from Lookup	Lookup Field Name in Event	Default Value
masks	__masks	Enter default value

+ Add field(s)

Lookup Function's configuration

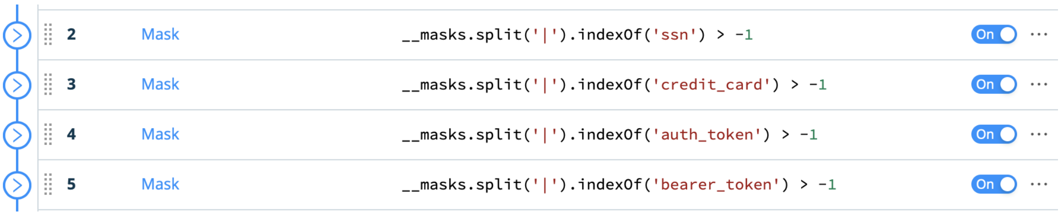
This Function keys against both the `index` and `sourcetype` fields. When it finds a matching combination, it adds a new key-value pair to your event for future filtering.

The key of that key-value pair (namely, `__masks`) starts with a double underscore, to make it a LogStream 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:



2	Mask	`__masks.split('|').indexOf('ssn') > -1`	On
3	Mask	`__masks.split('|').indexOf('credit_card') > -1`	On
4	Mask	`__masks.split('|').indexOf('auth_token') > -1`	On
5	Mask	`__masks.split('|').indexOf('bearer_token') > -1`	On

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.

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 LogStream.

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:

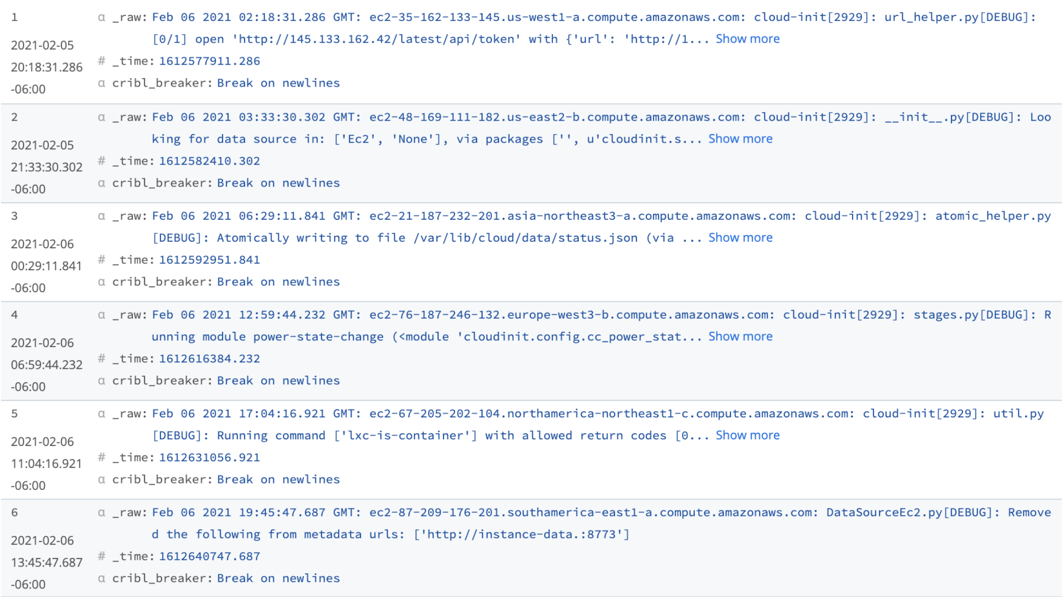
```
ec2-35-162-133-145.us-west1-a.compute.amazonaws.com
```

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 LogStream.

Sample Events

The events below have timestamps broken out, but no indexes, sourcetypes, or other details have been assigned yet:



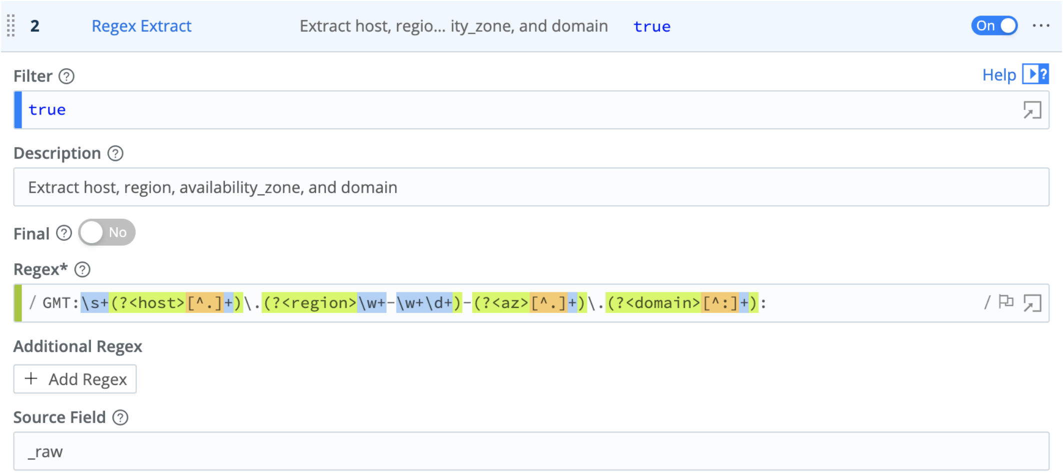
```
1            α _raw: Feb 06 2021 02:18:31.286 GMT: ec2-35-162-133-145.us-west1-a.compute.amazonaws.com: cloud-init[2929]: url_helper.py[DEBUG]:
2021-02-05         [0/1] open 'http://145.133.162.42/latest/api/token' with {'url': 'http://1... Show more
20:18:31.286 # _time: 1612577911.286
-06:00       α cribl_breaker: Break on newlines
2            α _raw: Feb 06 2021 03:33:30.302 GMT: ec2-48-169-111-182.us-east2-b.compute.amazonaws.com: cloud-init[2929]: __init__.py[DEBUG]: Loo
2021-02-05         king for data source in: ['Ec2', 'None'], via packages ['', u'cloudinit.s... Show more
21:33:30.302 # _time: 1612582410.302
-06:00       α cribl_breaker: Break on newlines
3            α _raw: Feb 06 2021 06:29:11.841 GMT: ec2-21-187-232-201.asia-northeast3-a.compute.amazonaws.com: cloud-init[2929]: atomic_helper.py
2021-02-06         [DEBUG]: Atomically writing to file /var/lib/cloud/data/status.json (via ... Show more
00:29:11.841 # _time: 1612592951.841
-06:00       α cribl_breaker: Break on newlines
4            α _raw: Feb 06 2021 12:59:44.232 GMT: ec2-76-187-246-132.europe-west3-b.compute.amazonaws.com: cloud-init[2929]: stages.py[DEBUG]: R
2021-02-06         unning module power-state-change (<module 'cloudinit.config.cc_power_stat... Show more
06:59:44.232 # _time: 1612616384.232
-06:00       α cribl_breaker: Break on newlines
5            α _raw: Feb 06 2021 17:04:16.921 GMT: ec2-67-205-202-104.northamerica-northeast1-c.compute.amazonaws.com: cloud-init[2929]: util.py
2021-02-06         [DEBUG]: Running command ['lxc-is-container'] with allowed return codes [0... Show more
11:04:16.921 # _time: 1612631056.921
-06:00       α cribl_breaker: Break on newlines
6            α _raw: Feb 06 2021 19:45:47.687 GMT: ec2-87-209-176-201.southamerica-east1-a.compute.amazonaws.com: DataSourceEc2.py[DEBUG]: Remove
2021-02-06         d the following from metadata urls: ['http://instance-data.:8773']
13:45:47.687 # _time: 1612640747.687
-06:00       α cribl_breaker: Break on newlines
```

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 LogStream Pipeline:



2 Regex Extract Extract host, regio... ity_zone, and domain `true` On

Filter: `true`

Description: Extract host, region, availability_zone, and domain

Final: No

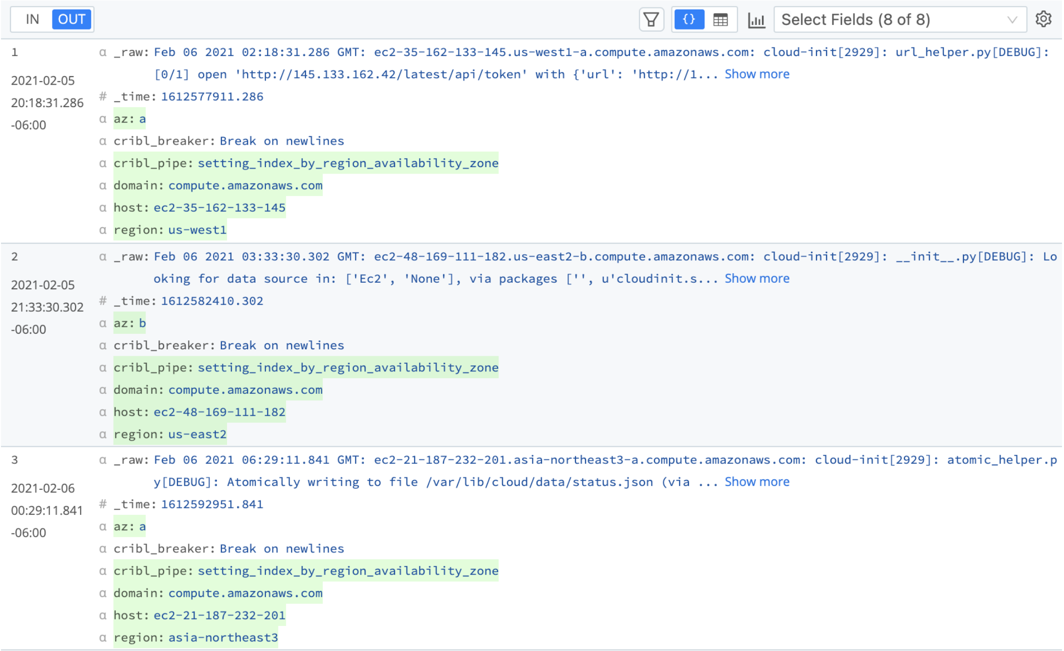
Regex*: `/ GMT:\s+(?<host>[^.]+)\.(?<region>\w+-\w+\d+)-(?<az>[^.]+)\.(?<domain>[^:]+): /`

Additional Regex

Source Field: _raw

Results of the Regex Extract Function

On the **OUT** tab of LogStream'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.



```
1                      α _raw: Feb 06 2021 02:18:31.286 GMT: ec2-35-162-133-145.us-west1-a.compute.amazonaws.com: cloud-init[2929]: url_helper.py[DEBUG]:
2021-02-05                   [0/1] open 'http://145.133.162.42/latest/api/token' with {'url': 'http://1... Show more
20:18:31.286           # _time: 1612577911.286
-06:00                 α az: a
                       α cribl_breaker: Break on newlines
                       α cribl_pipe: setting_index_by_region_availability_zone
                       α domain: compute.amazonaws.com
                       α host: ec2-35-162-133-145
                       α region: us-west1
2                      α _raw: Feb 06 2021 03:33:30.302 GMT: ec2-48-169-111-182.us-east2-b.compute.amazonaws.com: cloud-init[2929]: __init__.py[DEBUG]: Lo
2021-02-05                   oking for data source in: ['Ec2', 'None'], via packages ['', u'cloudinit.s... Show more
21:33:30.302           # _time: 1612582410.302
-06:00                 α az: b
                       α cribl_breaker: Break on newlines
                       α cribl_pipe: setting_index_by_region_availability_zone
                       α domain: compute.amazonaws.com
                       α host: ec2-48-169-111-182
                       α region: us-east2
3                      α _raw: Feb 06 2021 06:29:11.841 GMT: ec2-21-187-232-201.asia-northeast3-a.compute.amazonaws.com: cloud-init[2929]: atomic_helper.p
2021-02-06                   y[DEBUG]: Atomically writing to file /var/lib/cloud/data/status.json (via ... Show more
00:29:11.841           # _time: 1612592951.841
-06:00                 α az: a
                       α cribl_breaker: Break on newlines
                       α cribl_pipe: setting_index_by_region_availability_zone
                       α domain: compute.amazonaws.com
                       α host: ec2-21-187-232-201
                       α region: asia-northeast3
```

Lookups

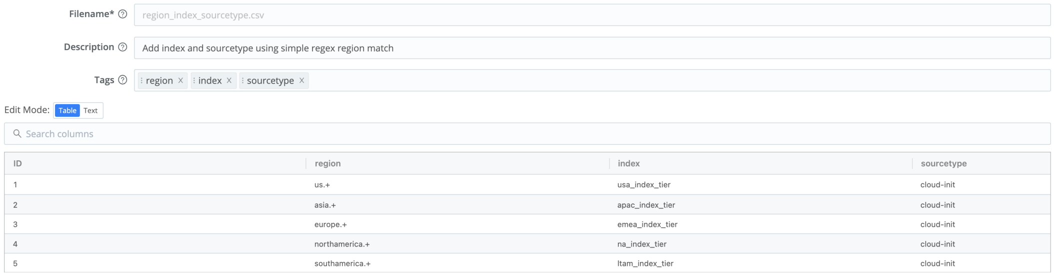
We still need to determine the index and sourcetype. LogStream's Lookup Function enriches events with external fields. We'll use it with the newly extracted `region` field to assign an `index` and `sourcetype` to these events.

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.



Filename* ⓘ region_index_sourcetype.csv

Description ⓘ Add index and sourcetype using simple regex region match

Tags ⓘ region × index × sourcetype ×

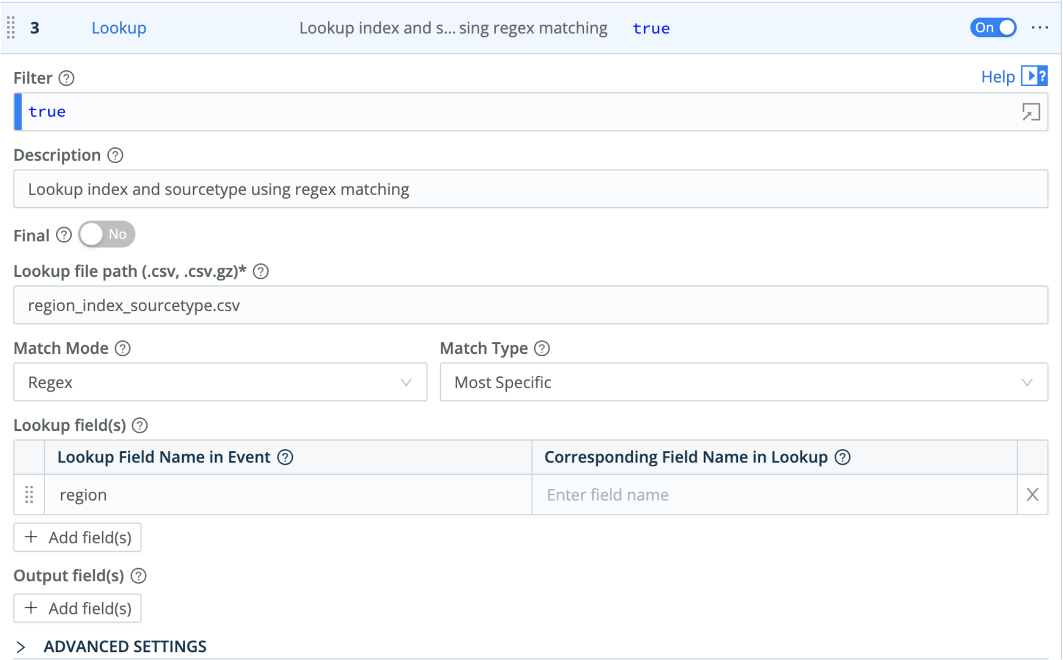
Edit Mode: Table Text

Search columns

ID	region	index	sourcetype
1	us.+	usa_index_tier	cloud-init
2	asia.+	apac_index_tier	cloud-init
3	europe.+	emea_index_tier	cloud-init
4	northamerica.+	na_index_tier	cloud-init
5	southamerica.+	ltam_index_tier	cloud-init

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`.



3 Lookup Lookup index and s... sing regex matching true On

Filter: true

Description: Lookup index and sourcetype using regex matching

Final: No

Lookup file path (.csv, .csv.gz)*: region_index_sourcetype.csv

Match Mode: Regex

Match Type: Most Specific

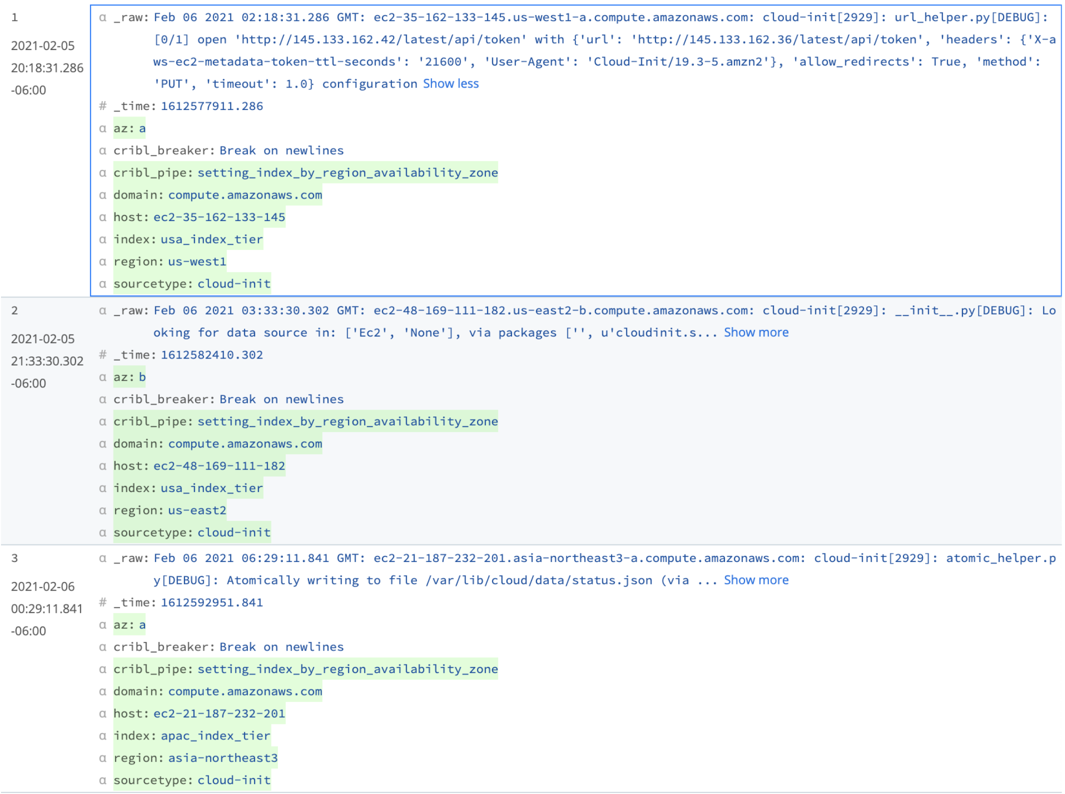
Lookup field(s)

Lookup Field Name in Event	Corresponding Field Name in Lookup
region	Enter field name

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.



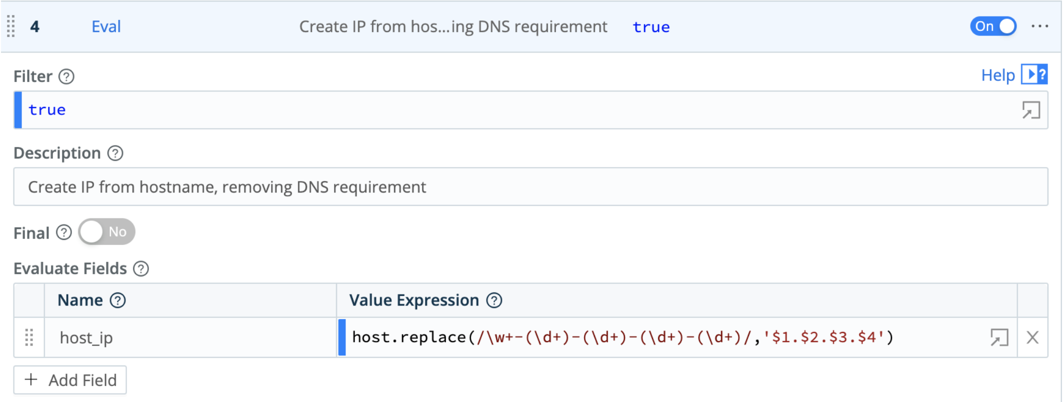
```
1                 α _raw: Feb 06 2021 02:18:31.286 GMT: ec2-35-162-133-145.us-west1-a.compute.amazonaws.com: cloud-init[2929]: url_helper.py[DEBUG]:
2021-02-05              [0/1] open 'http://145.133.162.42/latest/api/token' with {'url': 'http://145.133.162.36/latest/api/token', 'headers': {'X-a
20:18:31.286            ws-ec2-metadata-token-ttl-seconds': '21600', 'User-Agent': 'Cloud-Init/19.3-5.amzn2'}, 'allow_redirects': True, 'method':
-06:00                  'PUT', 'timeout': 1.0} configuration Show less
                  # _time: 1612577911.286
                  α az: a
                  α cribl_breaker: Break on newlines
                  α cribl_pipe: setting_index_by_region_availability_zone
                  α domain: compute.amazonaws.com
                  α host: ec2-35-162-133-145
                  α index: usa_index_tier
                  α region: us-west1
                  α sourcetype: cloud-init
2                 α _raw: Feb 06 2021 03:33:30.302 GMT: ec2-48-169-111-182.us-east2-b.compute.amazonaws.com: cloud-init[2929]: __init__.py[DEBUG]: Lo
2021-02-05              oking for data source in: ['Ec2', 'None'], via packages ['', u'cloudinit.s... Show more
21:33:30.302      # _time: 1612582410.302
-06:00            α az: b
                  α cribl_breaker: Break on newlines
                  α cribl_pipe: setting_index_by_region_availability_zone
                  α domain: compute.amazonaws.com
                  α host: ec2-48-169-111-182
                  α index: usa_index_tier
                  α region: us-east2
                  α sourcetype: cloud-init
3                 α _raw: Feb 06 2021 06:29:11.841 GMT: ec2-21-187-232-201.asia-northeast3-a.compute.amazonaws.com: cloud-init[2929]: atomic_helper.p
2021-02-06              y[DEBUG]: Atomically writing to file /var/lib/cloud/data/status.json (via ... Show more
00:29:11.841      # _time: 1612592951.841
-06:00            α az: a
                  α cribl_breaker: Break on newlines
                  α cribl_pipe: setting_index_by_region_availability_zone
                  α domain: compute.amazonaws.com
                  α host: ec2-21-187-232-201
                  α index: apac_index_tier
                  α region: asia-northeast3
                  α sourcetype: cloud-init
```

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!



4 Eval Create IP from hos...ing DNS requirement `true` On

Filter: `true`

Description: Create IP from hostname, removing DNS requirement

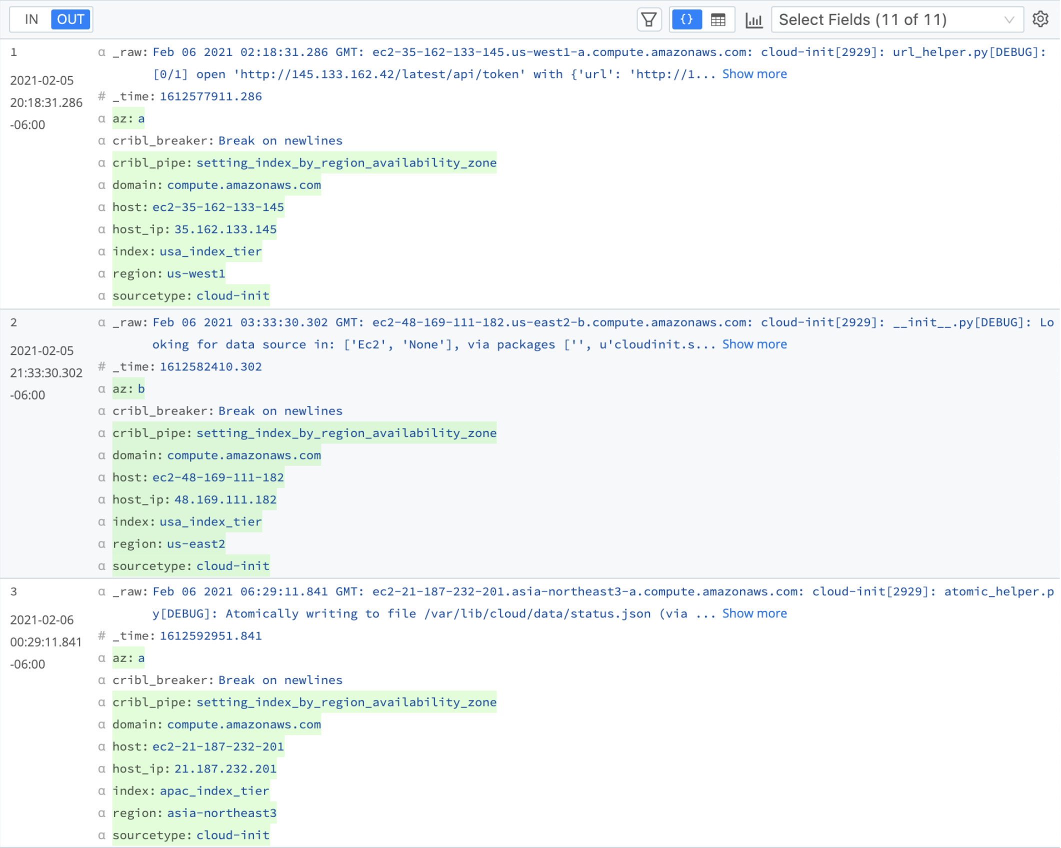
Final: No

Evaluate Fields

Name	Value Expression
host_ip	`host.replace(/\w+-(\d+)-(\d+)-(\d+)-(\d+)/,'$1.$2.$3.$4')`

Results of the Eval Function and Replace Method

The `host_ip` field is now added to the events, displayed below `host` :

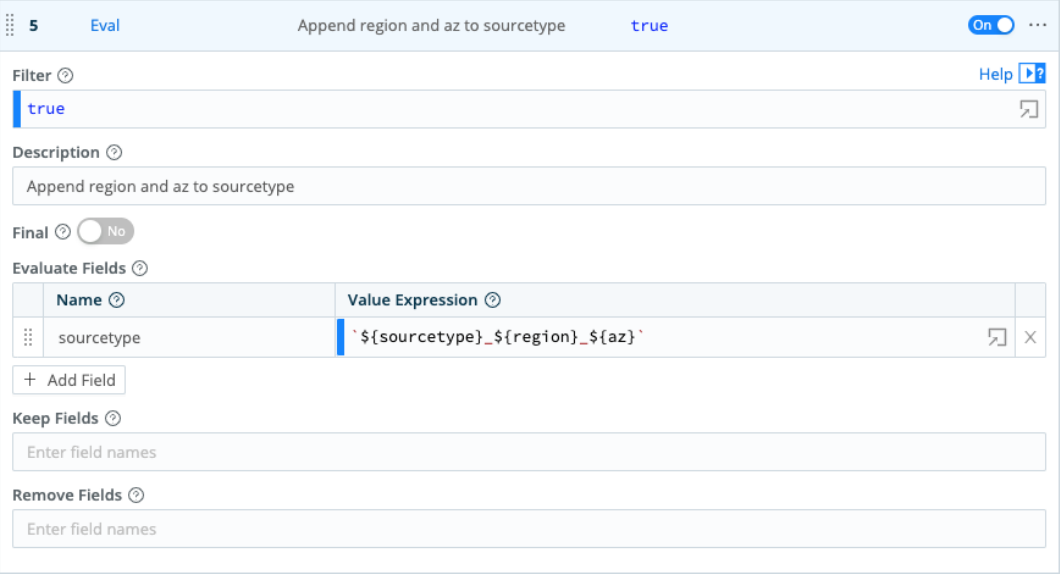


```
1             α _raw: Feb 06 2021 02:18:31.286 GMT: ec2-35-162-133-145.us-west1-a.compute.amazonaws.com: cloud-init[2929]: url_helper.py[DEBUG]:
                     [0/1] open 'http://145.133.162.42/latest/api/token' with {'url': 'http://1... Show more
2021-02-05    # _time: 1612577911.286
20:18:31.286  α az: a
-06:00        α cribl_breaker: Break on newlines
              α cribl_pipe: setting_index_by_region_availability_zone
              α domain: compute.amazonaws.com
              α host: ec2-35-162-133-145
              α host_ip: 35.162.133.145
              α index: usa_index_tier
              α region: us-west1
              α sourcetype: cloud-init
2             α _raw: Feb 06 2021 03:33:30.302 GMT: ec2-48-169-111-182.us-east2-b.compute.amazonaws.com: cloud-init[2929]: __init__.py[DEBUG]: Lo
                     oking for data source in: ['Ec2', 'None'], via packages ['', u'cloudinit.s... Show more
2021-02-05    # _time: 1612582410.302
21:33:30.302  α az: b
-06:00        α cribl_breaker: Break on newlines
              α cribl_pipe: setting_index_by_region_availability_zone
              α domain: compute.amazonaws.com
              α host: ec2-48-169-111-182
              α host_ip: 48.169.111.182
              α index: usa_index_tier
              α region: us-east2
              α sourcetype: cloud-init
3             α _raw: Feb 06 2021 06:29:11.841 GMT: ec2-21-187-232-201.asia-northeast3-a.compute.amazonaws.com: cloud-init[2929]: atomic_helper.p
                     y[DEBUG]: Atomically writing to file /var/lib/cloud/data/status.json (via ... Show more
2021-02-06    # _time: 1612592951.841
00:29:11.841  α az: a
-06:00        α cribl_breaker: Break on newlines
              α cribl_pipe: setting_index_by_region_availability_zone
              α domain: compute.amazonaws.com
              α host: ec2-21-187-232-201
              α host_ip: 21.187.232.201
              α index: apac_index_tier
              α region: asia-northeast3
              α sourcetype: cloud-init
```

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.



5 Eval Append region and az to sourcetype true On

Filter: true

Description: Append region and az to sourcetype

Final: No

Evaluate Fields

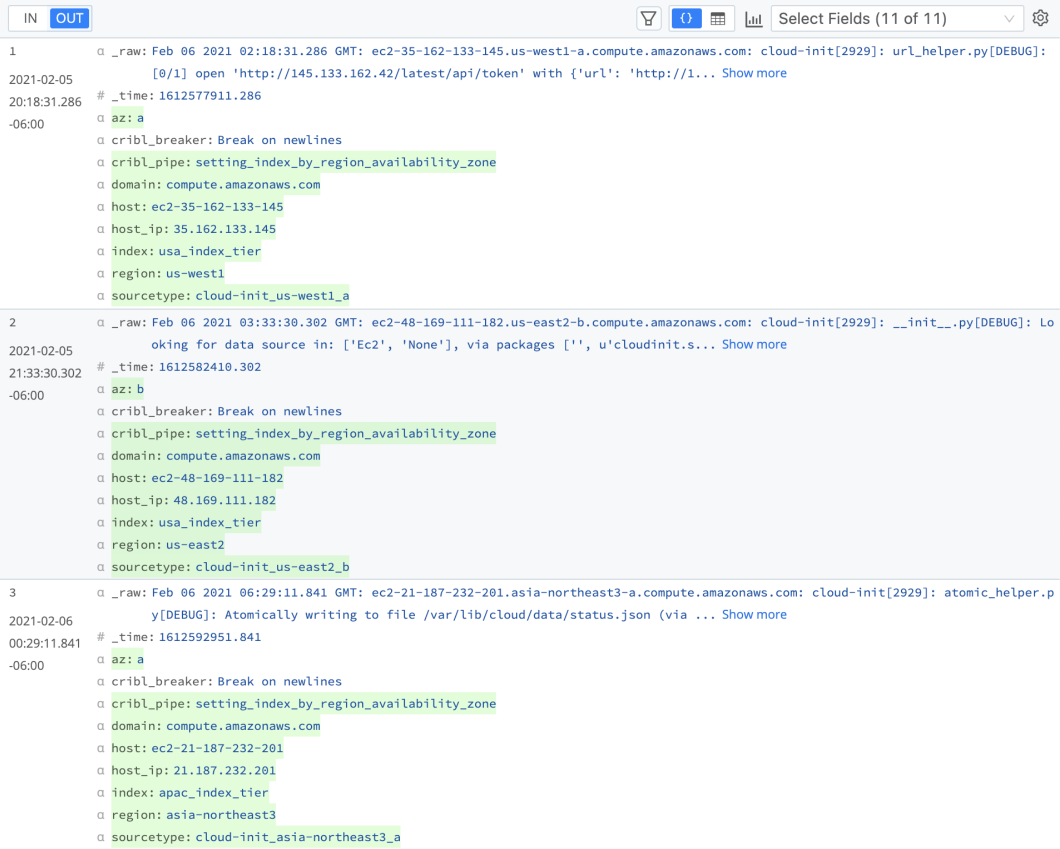
Name	Value Expression
sourcetype	`` `${sourcetype}_${region}_${az}` ``

Keep Fields: Enter field names

Remove Fields: Enter field names

Results of the Eval Function to Combine Values

Take a look at the updated sourcetypes, and enjoy exploring LogStream with your new knowledge!



```
1                  α _raw: Feb 06 2021 02:18:31.286 GMT: ec2-35-162-133-145.us-west1-a.compute.amazonaws.com: cloud-init[2929]: url_helper.py[DEBUG]:
                           [0/1] open 'http://145.133.162.42/latest/api/token' with {'url': 'http://1... Show more
2021-02-05         # _time: 1612577911.286
20:18:31.286       α az: a
-06:00             α cribl_breaker: Break on newlines
                   α cribl_pipe: setting_index_by_region_availability_zone
                   α domain: compute.amazonaws.com
                   α host: ec2-35-162-133-145
                   α host_ip: 35.162.133.145
                   α index: usa_index_tier
                   α region: us-west1
                   α sourcetype: cloud-init_us-west1_a
2                  α _raw: Feb 06 2021 03:33:30.302 GMT: ec2-48-169-111-182.us-east2-b.compute.amazonaws.com: cloud-init[2929]: __init__.py[DEBUG]: Lo
                           oking for data source in: ['Ec2', 'None'], via packages ['', u'cloudinit.s... Show more
2021-02-05         # _time: 1612582410.302
21:33:30.302       α az: b
-06:00             α cribl_breaker: Break on newlines
                   α cribl_pipe: setting_index_by_region_availability_zone
                   α domain: compute.amazonaws.com
                   α host: ec2-48-169-111-182
                   α host_ip: 48.169.111.182
                   α index: usa_index_tier
                   α region: us-east2
                   α sourcetype: cloud-init_us-east2_b
3                  α _raw: Feb 06 2021 06:29:11.841 GMT: ec2-21-187-232-201.asia-northeast3-a.compute.amazonaws.com: cloud-init[2929]: atomic_helper.p
                           y[DEBUG]: Atomically writing to file /var/lib/cloud/data/status.json (via ... Show more
2021-02-06         # _time: 1612592951.841
00:29:11.841       α az: a
-06:00             α cribl_breaker: Break on newlines
                   α cribl_pipe: setting_index_by_region_availability_zone
                   α domain: compute.amazonaws.com
                   α host: ec2-21-187-232-201
                   α host_ip: 21.187.232.201
                   α index: apac_index_tier
                   α region: asia-northeast3
                   α sourcetype: cloud-init_asia-northeast3_a
```

Try This at Home

Below you'll find the 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 LogStream's UI, select **Knowledge > Lookups**, then click **+ Add New** 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
```

Pipeline

Below is an export of the whole LogStream 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 le
        },
        "id": "comment"
      },
      {
        "filter": "true",
        "conf": {
          "source": "_raw",
          "iterations": 100,
          "overwrite": false,
          "regex": "/GMT:\\s+(?<host>[^.]+)\\.(?<region>\\w+-\\w+\\d+)-(?<
        },
        "id": "regex_extract",
        "disabled": false,
        "description": "Extract host, region, availability_zone, and domai
      },
      {
        "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+)/,'$
```

Sample Events

And here's a sample of raw events that you can upload or copy/paste into LogStream's Preview pane to test the Pipeline's Functions:

Sample events

```
Feb 06 2021 02:18:31.286 GMT: ec2-35-162-133-145.us-west1-a.compute.amazon
Feb 06 2021 03:33:30.302 GMT: ec2-48-169-111-182.us-east2-b.compute.amazon
Feb 06 2021 06:29:11.841 GMT: ec2-21-187-232-201.asia-northeast3-a.compute
Feb 06 2021 12:59:44.232 GMT: ec2-76-187-246-132.europe-west3-b.compute.am
Feb 06 2021 17:04:16.921 GMT: ec2-67-205-202-104.northamerica-northeast1-c
Feb 06 2021 19:45:47.687 GMT: ec2-87-209-176-201.southamerica-east1-a.comp
```

From here, modify the sample data, lookup table, and Functions to adapt this approach to your own needs!

Regex Filtering

Regex Filtering of Data in Motion

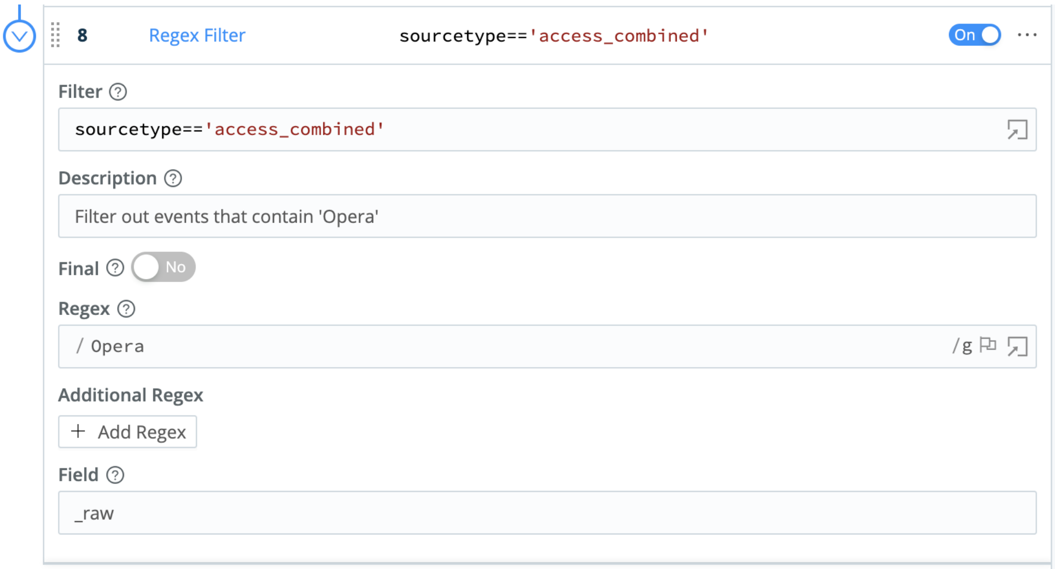
To filter events in real time, we use the out-of-the-box **Regex Filter** Function. This is similar to nullqueueing with TRANSFORMS in Splunk, but the matching condition is way more flexible.

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:



sourcetype=='access_combined'

Filter: sourcetype=='access_combined'

Description: Filter out events that contain 'Opera'

Final: No

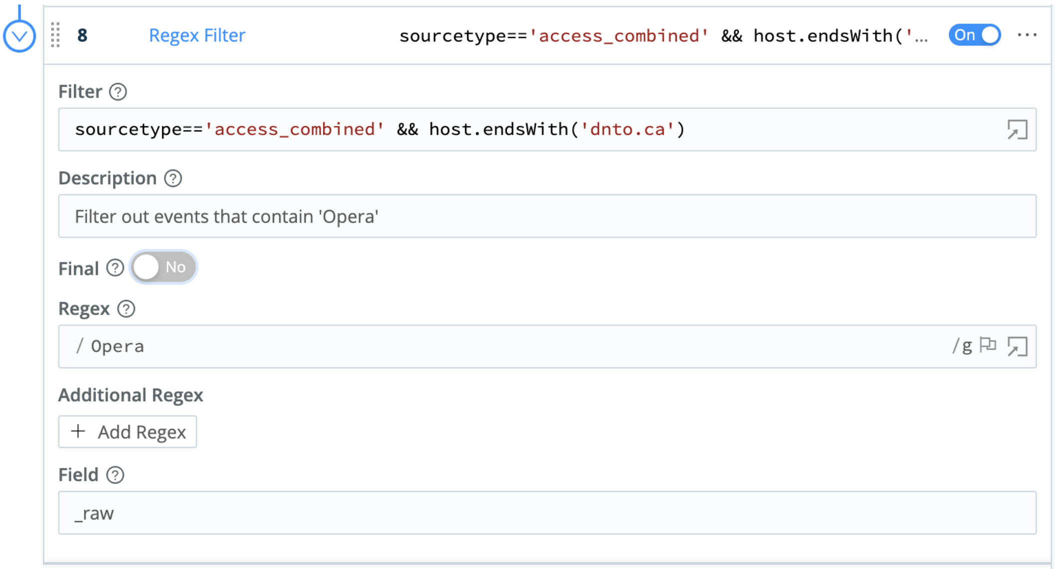
Regex: / Opera /g

Field: _raw

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 as shown below:



8 Regex Filter `sourcetype=='access_combined' && host.endsWith('...` On

Filter

```
sourcetype=='access_combined' && host.endsWith('dnto.ca')
```

Description

Filter out events that contain 'Opera'

Final No

Regex

```
/ Opera /g
```

Additional Regex

+ Add Regex

Field

_raw

Filtering by host

This is a very flexible method for filtering incoming events in real time, on virtually any arbitrary conditions.

Encrypting Sensitive Data

Encryption at Ingest-Time and Decryption in Splunk

With Cribl LogStream, 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` .)

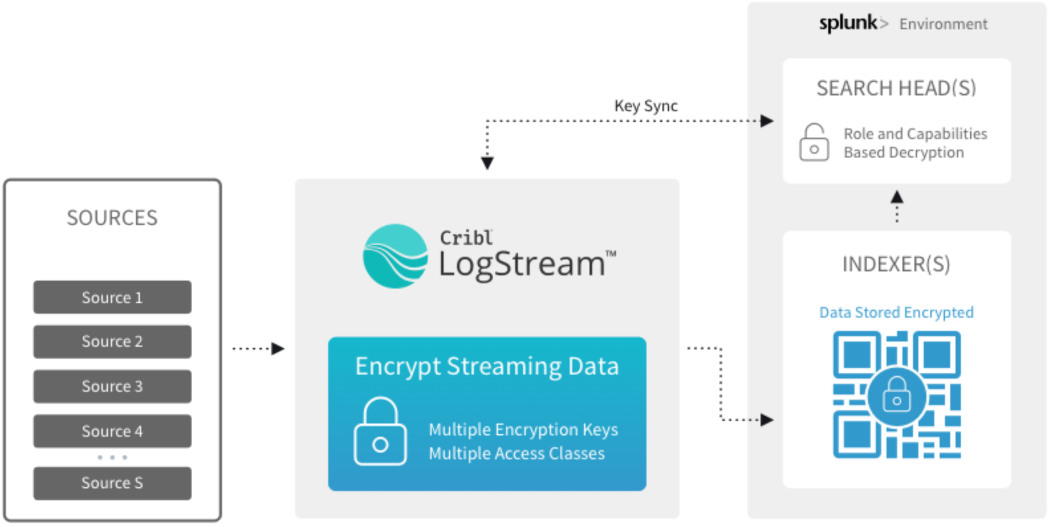
Keys and Key Classes

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.

Encrypting in Cribl LogStream and Decrypting in Splunk

1. Define one or more keys and key classes on Cribl LogStream. (See UI- and CLI-based instructions.)
2. Sync `auth` with the decryption side (Splunk Search Head)
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.

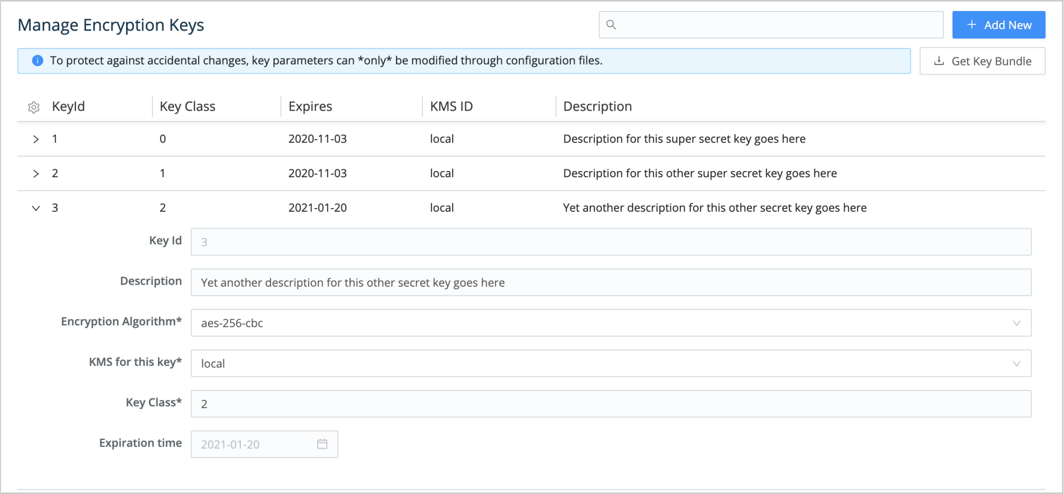


Encrypting in LogStream, decrypting in Splunk

Example

Encryption Side

- Generate keys via the UI, in global ⚙ **Settings** (lower left) > **Security > Encryption Keys**:



Manage Encryption Keys

To protect against accidental changes, key parameters can *only* be modified through configuration files.

KeyId	Key Class	Expires	KMS ID	Description
1	0	2020-11-03	local	Description for this super secret key goes here
2	1	2020-11-03	local	Description for this other super secret key goes here
3	2	2021-01-20	local	Yet another description for this other secret key goes here

Key Id	3
Description	Yet another description for this other secret key goes here
Encryption Algorithm*	aes-256-cbc
KMS for this key*	local
Key Class*	2
Expiration time	2021-01-20

Adding a new encryption key

- Or generate one or more keys via the CLI. 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`:

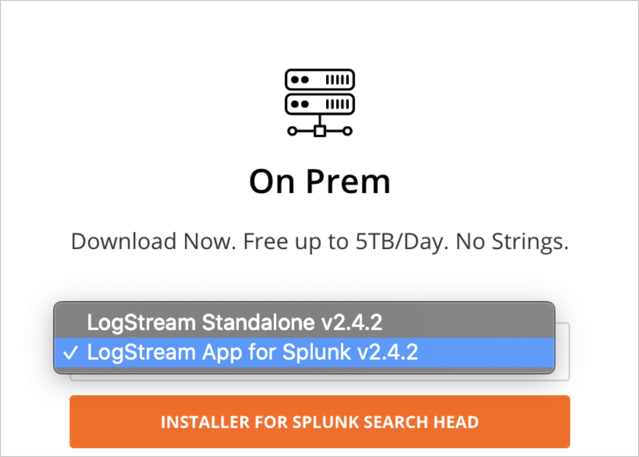
```
$CRIBL_HOME/bin/cribl keys add -c 1 -i -g uk
...
$CRIBL_HOME/bin/cribl keys add -c <N> -i -g uk
```

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

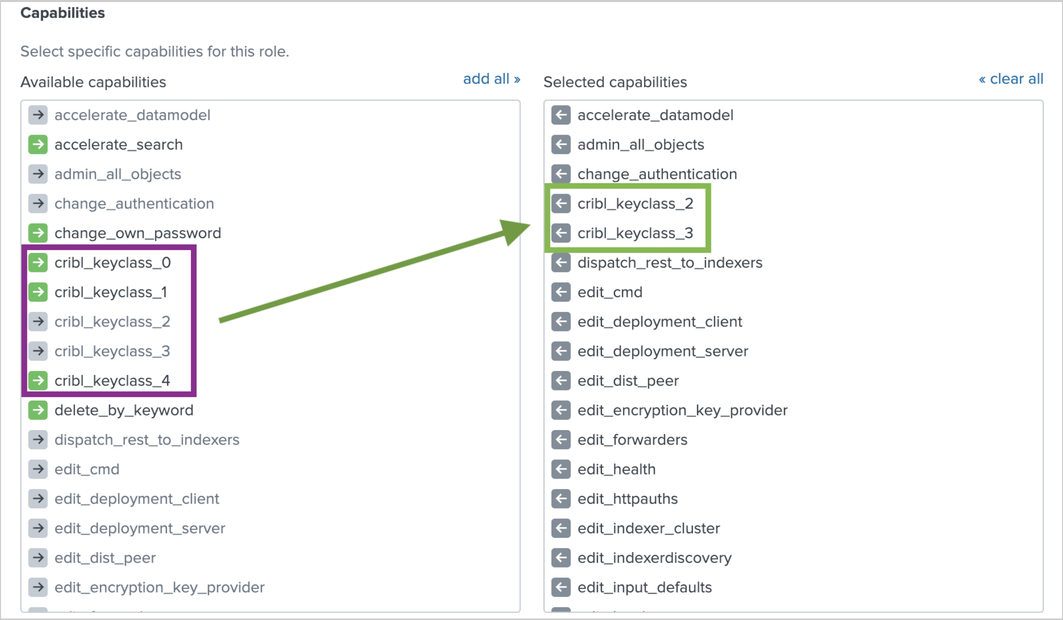
- Download the Cribl/LogStream App for Splunk from Cribl's Download LogStream page: In the **On Prem** section, select the Splunk app from the drop-down list, as shown. Clicking the orange button downloads a file named: `cribl-splunk-app-<version-#>-<hash-#>-linux-x64.tgz`.


On Prem
Download Now. Free up to 5TB/Day. No Strings.
LogStream Standalone v2.4.2
✓ LogStream App for Splunk v2.4.2
INSTALLER FOR SPLUNK SEARCH HEAD

Downloading Cribl's Splunk app

- To install the Cribl/LogStream 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.



Capabilities

Select specific capabilities for this role.

Available capabilities — add all »

- accelerate_datamodel
- accelerate_search
- admin_all_objects
- change_authentication
- change_own_password
- cribl_keyclass_0
- cribl_keyclass_1
- cribl_keyclass_2
- cribl_keyclass_3
- cribl_keyclass_4
- delete_by_keyword
- dispatch_rest_to_indexers
- edit_cmd
- edit_deployment_client
- edit_deployment_server
- edit_dist_peer
- edit_encryption_key_provider

Selected capabilities — « clear all

- accelerate_datamodel
- admin_all_objects
- change_authentication
- cribl_keyclass_2
- cribl_keyclass_3
- dispatch_rest_to_indexers
- edit_cmd
- edit_deployment_client
- edit_deployment_server
- edit_dist_peer
- edit_encryption_key_provider
- edit_forwarders
- edit_health
- edit_httpauths
- edit_indexer_cluster
- edit_indexerdiscovery
- edit_input_defaults

Selecting capabiities

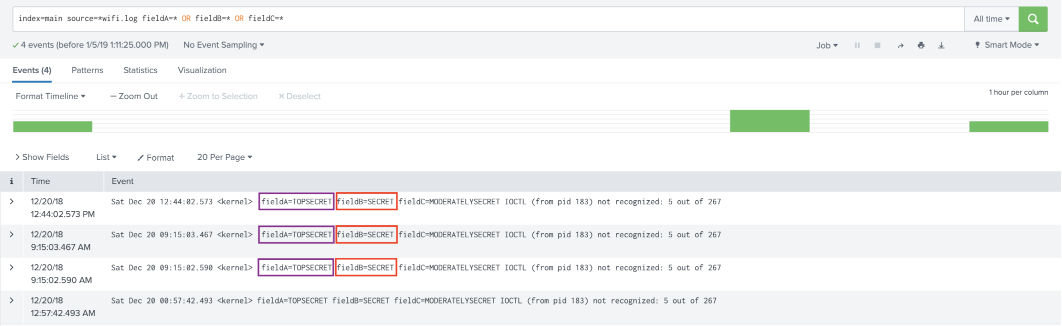
- Sync `auth/(cribl.secret|keys.json)`. To decrypt data, the `decrypt` command will need access to the same keys that were used to encrypt, **in the Cribl instance where encryption happened**.
  - In a single-instance deployment, the `cribl.secret` and `keys.json` files reside in: `$CRIBL_HOME/local/cribl/auth/`.
  - In a distributed deployment, these files reside on the Leader Node in: `$CRIBL_HOME/groups/<group-name>/local/cribl/auth/`.
  - When using LogStream'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.

> ℹ Modifying Keys
>
> When you update keys by editing the `keys.json` file, you must add them back to 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` .



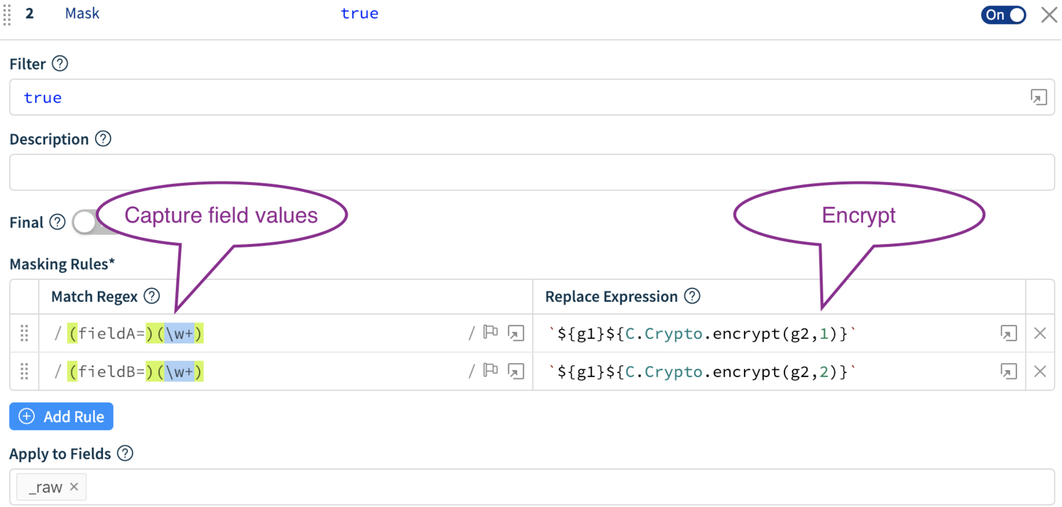
```
index=main source=*wifi.log fieldA=* OR fieldB=* OR fieldC=*
```

✓ 4 events (before 1/5/19 1:11:25.000 PM) No Event Sampling

i	Time	Event
>	12/20/18 12:44:02.573 PM	Sat Dec 20 12:44:02.573 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:03.467 AM	Sat Dec 20 09:15:03.467 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:02.590 AM	Sat Dec 20 09:15:02.590 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 12:57:42.493 AM	Sat Dec 20 00:57:42.493 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267

Events before encryption

Next, encrypt `fieldA` values with key class `1` , and `fieldB` with key class `2` .



2 Mask `true` On

Filter: `true`

Description:

Final

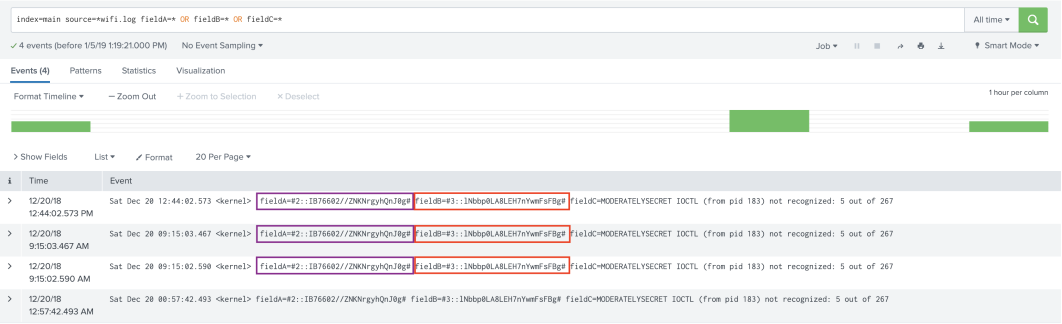
Masking Rules*

Match Regex	Replace Expression
`/(fieldA=)(\w+)/`	`` `${g1}${C.Crypto.encrypt(g2,1)}` ``
`/(fieldB=)(\w+)/`	`` `${g1}${C.Crypto.encrypt(g2,2)}` ``

Apply to Fields: _raw

Encrypting two fields with separate key classes

After Encryption: again, notice the values of `fieldA` and `fieldB` .



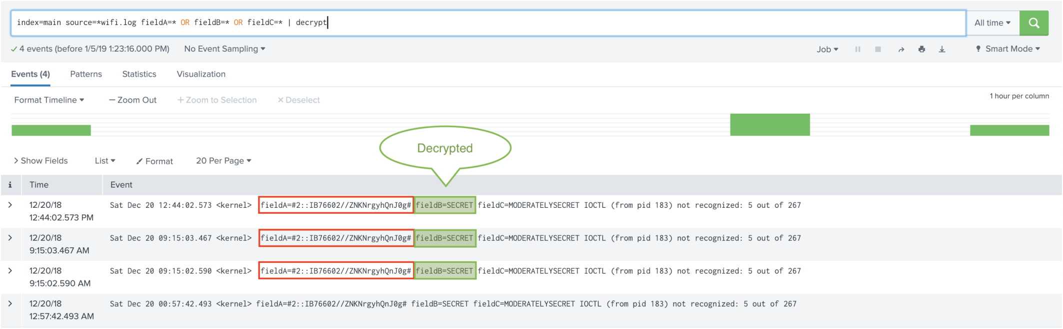
```
index=main source=*wifi.log fieldA=* OR fieldB=* OR fieldC=*
```

✓ 4 events (before 1/5/19 1:19:21.000 PM) No Event Sampling

i	Time	Event
>	12/20/18 12:44:02.573 PM	Sat Dec 20 12:44:02.573 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::lNbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:03.467 AM	Sat Dec 20 09:15:03.467 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::lNbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:02.590 AM	Sat Dec 20 09:15:02.590 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::lNbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 12:57:42.493 AM	Sat Dec 20 00:57:42.493 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::lNbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267

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`.



```
index=main source=*wifi.log fieldA=* OR fieldB=* OR fieldC=* | decrypt
```

✓ 4 events (before 1/5/19 1:23:16.000 PM)

i	Time	Event
>	12/20/18 12:44:02.573 PM	Sat Dec 20 12:44:02.573 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:03.467 AM	Sat Dec 20 09:15:03.467 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:02.590 AM	Sat Dec 20 09:15:02.590 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 12:57:42.493 AM	Sat Dec 20 00:57:42.493 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267

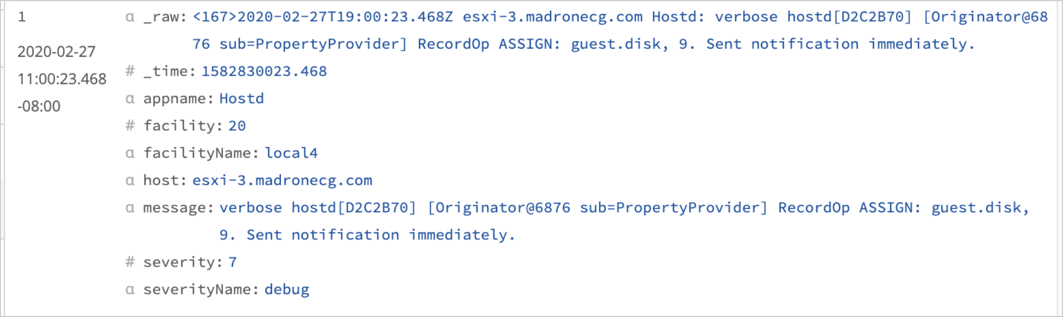
One field decrypted

Syslog Data Reduction

When ingesting data from syslog senders, Cribl LogStream can readily trim data volume by 30% or more, optimizing infrastructure for downstream services like Splunk or Elasticsearch. Here, we outline some best practices for replacing your syslog server with LogStream.

Syslog Event Parsing

By default, a LogStream Syslog Source will produce the following fields: `_time`, `appname`, `facility` and `facilityName`, `host`, `message`, and `severity` and `severityName`.

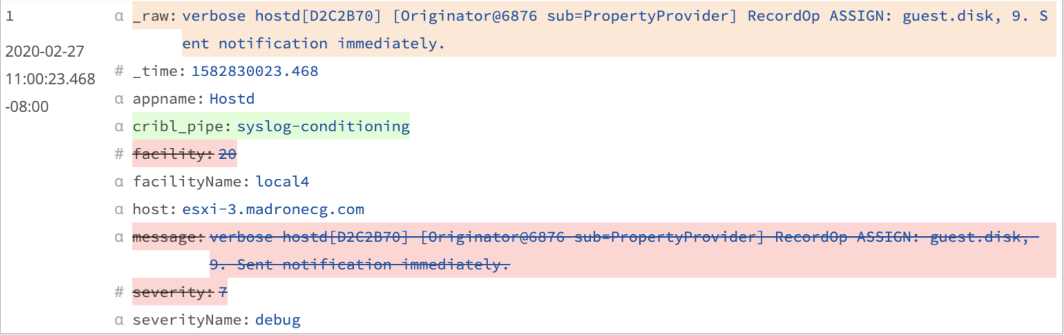


```
1              α _raw: <167>2020-02-27T19:00:23.468Z esxi-3.madronecg.com Hostd: verbose hostd[D2C2B70] [Originator@68
                      76 sub=PropertyProvider] RecordOp ASSIGN: guest.disk, 9. Sent notification immediately.
2020-02-27     # _time: 1582830023.468
11:00:23.468   α appname: Hostd
-08:00         # facility: 20
               α facilityName: local4
               α host: esxi-3.madronecg.com
               α message: verbose hostd[D2C2B70] [Originator@6876 sub=PropertyProvider] RecordOp ASSIGN: guest.disk,
                         9. Sent notification immediately.
               # severity: 7
               α severityName: debug
```

Parsed syslog event

This output is much more readable, but we haven't saved any volume – we now have redundant pairs of fields (numeric versus text) representing the facility and severity.

Below, we'll outline how to streamline syslog events to something more like this:



```
1                 α _raw: verbose hostd[D2C2B70] [Originator@6876 sub=PropertyProvider] RecordOp ASSIGN: guest.disk, 9. S
                        ent notification immediately.
2020-02-27        # _time: 1582830023.468
11:00:23.468      α appname: Hostd
-08:00            α cribl_pipe: syslog-conditioning
                  # ~~facility: 20~~
                  α facilityName: local4
                  α host: esxi-3.madronecg.com
                  α ~~message: verbose hostd[D2C2B70] [Originator@6876 sub=PropertyProvider] RecordOp ASSIGN: guest.disk,~~
                        ~~9. Sent notification immediately.~~
                  # ~~severity: 7~~
                  α severityName: debug
```

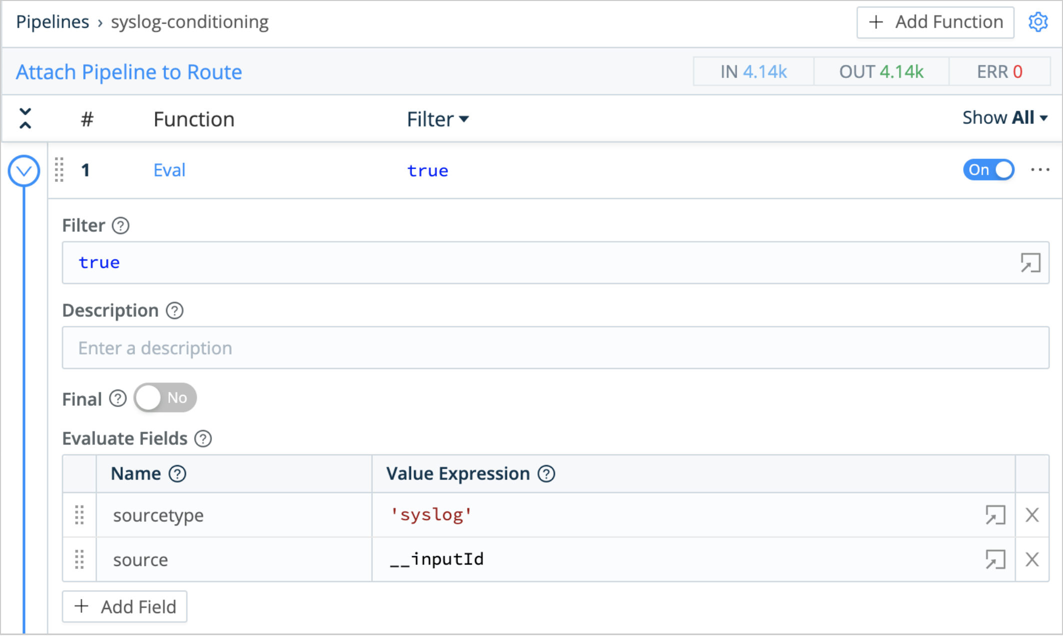
Parsed and redacted syslog event

This extracts the essentials, removes the redundancies, adds one new field that identifies the LogStream Pipeline we're about to build, and shrinks the outbound `_raw` payload to just its `message` component. For still further efficiencies, we'll look at how to drop or downsample frequent events, and how to balance high-volume syslog inputs across LogStream worker processes.

Create Pre-Processing Pipeline

Even before setting up a syslog Source, our first step is to create a pre-processing Pipeline that will be available to normalize incoming events on **all** syslog Sources, reducing data volume as shown above.

The Pipeline starts with an **Eval** Function, whose **Evaluate Fields** section tags syslog events with two new fields indicating their origin: `sourcetype: 'syslog'` and `source: __inputId`. Because this Pipeline is designed only to condition all incoming syslog data, we leave **Filter** set to its default `true` value, to process all events.



1 Eval `true` On

Filter: `true`

Description: Enter a description

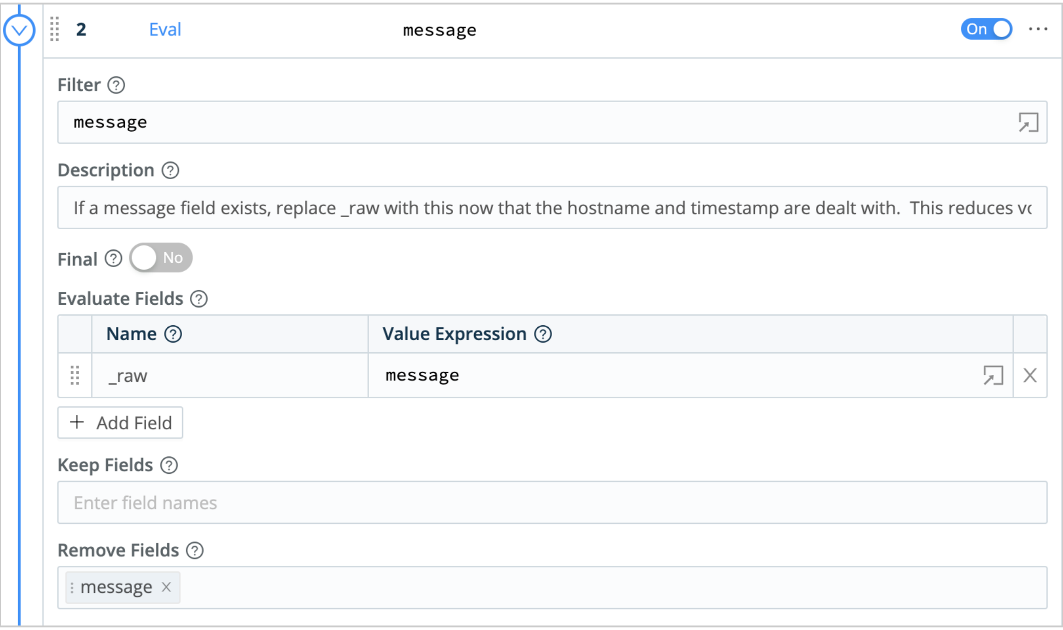
Final: No

Evaluate Fields

Name	Value Expression
sourcetype	`'syslog'`
source	`__inputId`

Eval Function to tag syslog events' origin

A second **Eval** Function filters for the presence of a `message` field. If found, it overwrites the `_raw` field with `message`, and then deletes `message` as redundant. This function alone typically reduces syslog data volume by 15–20%.



2 Eval `message` On

Filter: `message`

Description: If a message field exists, replace _raw with this now that the hostname and timestamp are dealt with. This reduces vc

Final: No

Evaluate Fields

Name	Value Expression
_raw	`message`

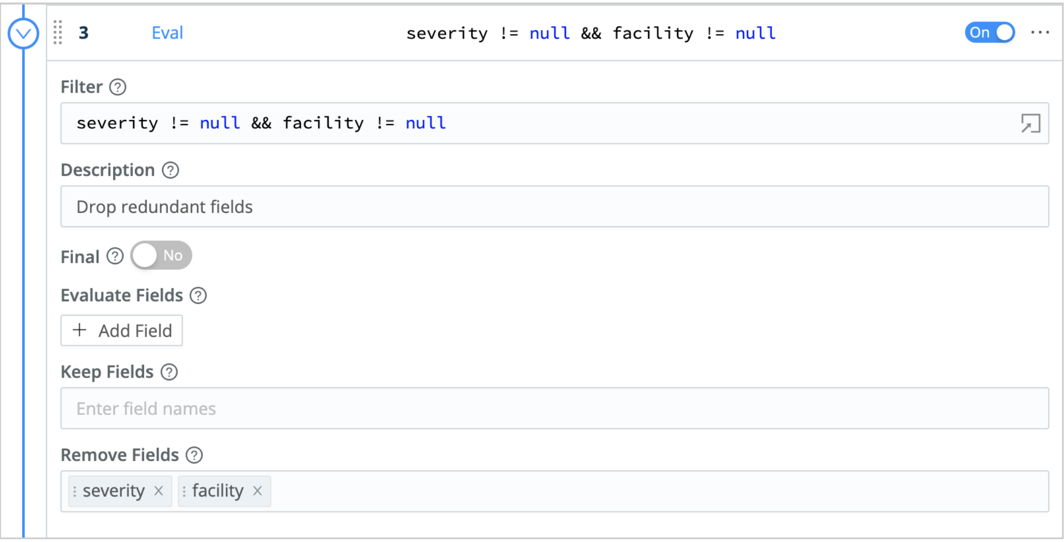
Keep Fields: Enter field names

Remove Fields: message ×

Eval Function to rewrite `message` as `_raw`

> ⚠ Before using this Pipeline in production, preview sample data to verify that you're not dropping any essential information.

This third **Eval** Function deletes two redundant fields. Its **Filter** condition makes sure both of these string fields exist and contain values: `severity ≠ null && facility ≠ null`. If so, it removes their corresponding numeric fields, `severity` and `facility`.



3 Eval severity != null && facility != null On

Filter: `severity != null && facility != null`

Description: Drop redundant fields

Final: No

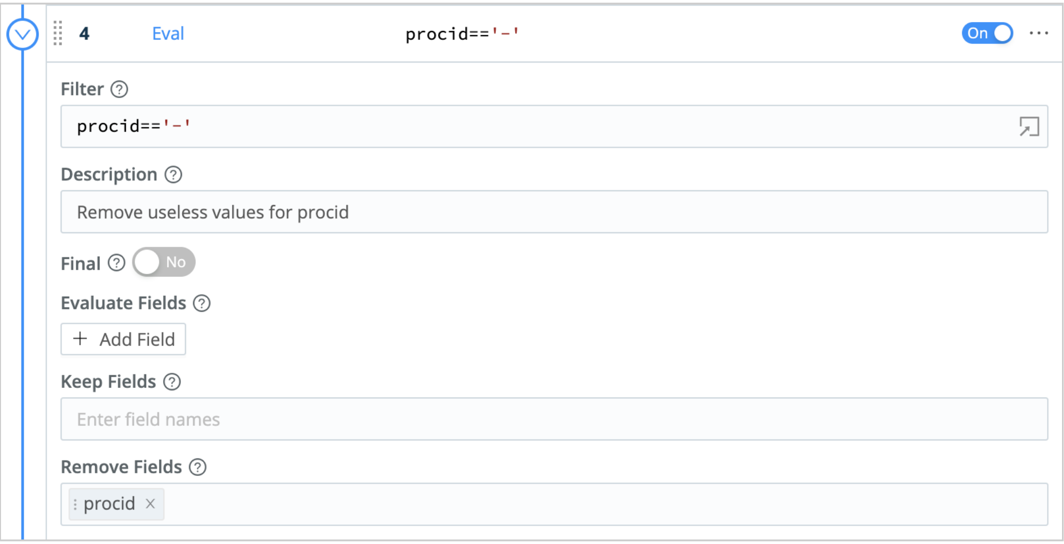
Evaluate Fields: + Add Field

Keep Fields: Enter field names

Remove Fields: severity × facility ×

Eval Function to remove redundant numeric fields

To further shrink the output, this fourth **Eval** Function removes `procid` fields that contain only a dash – meaning "no value extracted." We set **Filter**: `procid=='-'` and **Remove Fields**: `procid`.



4 Eval procid=='-' On

Filter: `procid=='-'`

Description: Remove useless values for procid

Final: No

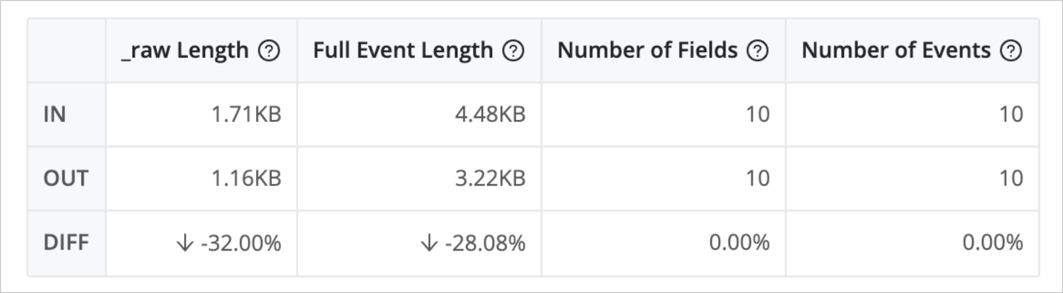
Evaluate Fields: + Add Field

Keep Fields: Enter field names

Remove Fields: procid ×

Eval Function to remove empty `procid` fields

With these four Functions enabled, the **Preview** pane's **Basic Statistics** confirm that we've reduced the `_raw` field's length by more than 30%.

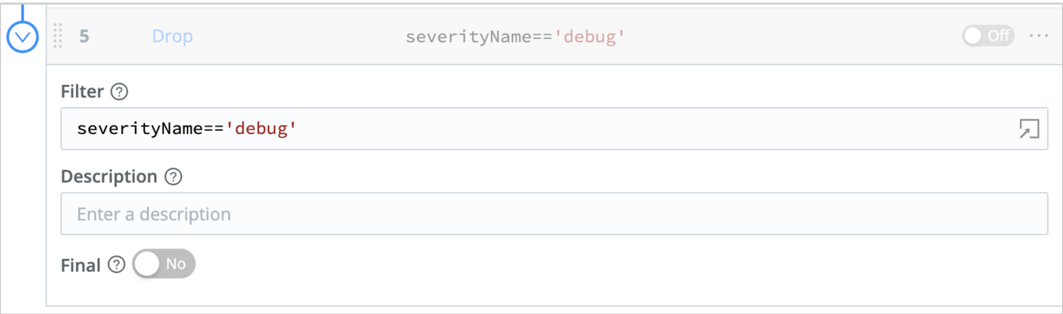


	_raw Length ⓘ	Full Event Length ⓘ	Number of Fields ⓘ	Number of Events ⓘ
IN	1.71KB	4.48KB	10	10
OUT	1.16KB	3.22KB	10	10
DIFF	↓ -32.00%	↓ -28.08%	0.00%	0.00%

Data reduction example

Dropping Noisy Data

With some syslog senders, like VMware ESX/ESXi, 80–90% of incoming events can be of `debug` severity. To further reduce volume, you could use this final **Drop** Function to drop all these events. Its **Filter** is simply `severityName=='debug'` .



5 Drop `severityName=='debug'` Off

Filter ⓘ

```
severityName=='debug'
```

Description ⓘ

Enter a description

Final ⓘ No

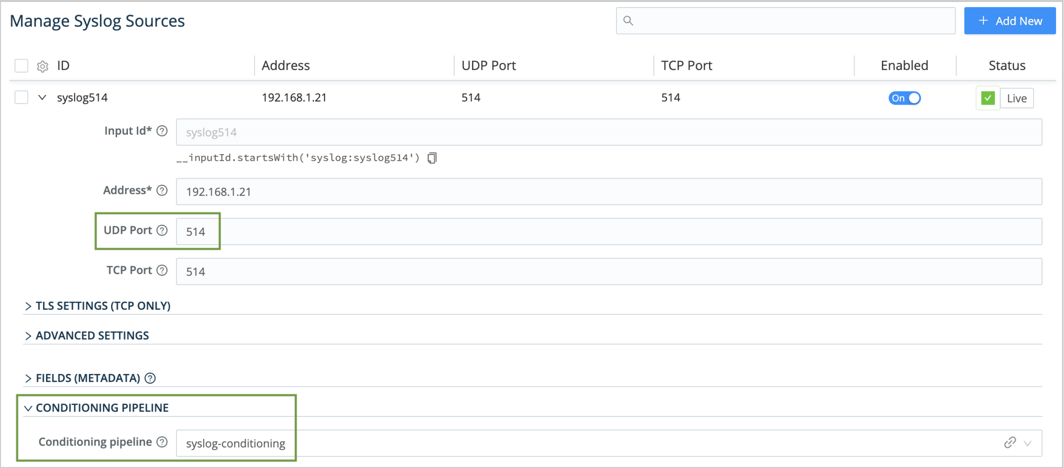
Drop Function to remove debug events

Enabling this Function could up our volume savings to about 40%. Depending on your needs, you might prefer to:

- Use a Function like this in your Route's processing Pipeline, rather than in this upfront Pipeline.
- Also drop `info` events.
- Instead use a Sampling Function to sample `debug` events (at a ratio like 1:10), or a Dynamic Sampling function.
- Instead use a Suppress Function to clamp down the frequency of `debug` events.

Create Syslog Source

Once we've built and saved the pre-processing Pipeline, our next step is to add a Syslog Source.



Manage Syslog Sources

ID	Address	UDP Port	TCP Port	Enabled	Status
syslog514	192.168.1.21	514	514	On	☑ Live

Input Id*: syslog514

`__inputId.startsWith('syslog:syslog514')`

Address*: 192.168.1.21

UDP Port: 514

TCP Port: 514

> TLS SETTINGS (TCP ONLY)

> ADVANCED SETTINGS

> FIELDS (METADATA)

CONDITIONING PIPELINE

Conditioning pipeline: syslog-conditioning

Syslog Source configured for UDP and pre-processing Pipeline

Specify the **UDP Port** where you want this Source to listen for syslog data.

Then attach the pre-processing Pipeline that you created above, and save the Source.

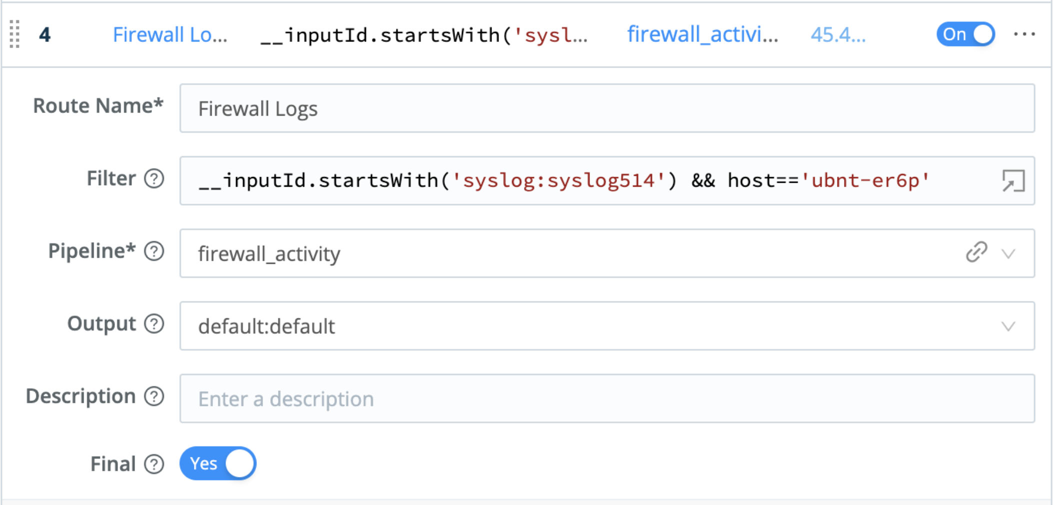
> i Cribl generally recommends selecting UDP, rather than TCP, for high-volume syslog senders. This facilitates efficient load balancing by not continuously tying such senders to any one LogStream Worker Process. See Sizing and Scaling for more details.

Fields/Metadata

In the pre-processing Pipeline, we tagged *all* incoming syslog events with new `sourcetype` and `source` fields to indicate their origin. Alternatively, you could use the Source's **Fields/Metadata** section to define similar key-value pairs, specific to each of your Syslog Sources.

Create Route(s)

Create Routes, as needed, for each of your Syslog Sources. Give each Route a corresponding **Filter** expression, and set its **Output** to the Destination where you want to send its processed syslog data.



Route Name* Firewall Logs

Filter `__inputId.startsWith('syslog:syslog514') && host=='ubnt-er6p'`

Pipeline* firewall_activity

Output default:default

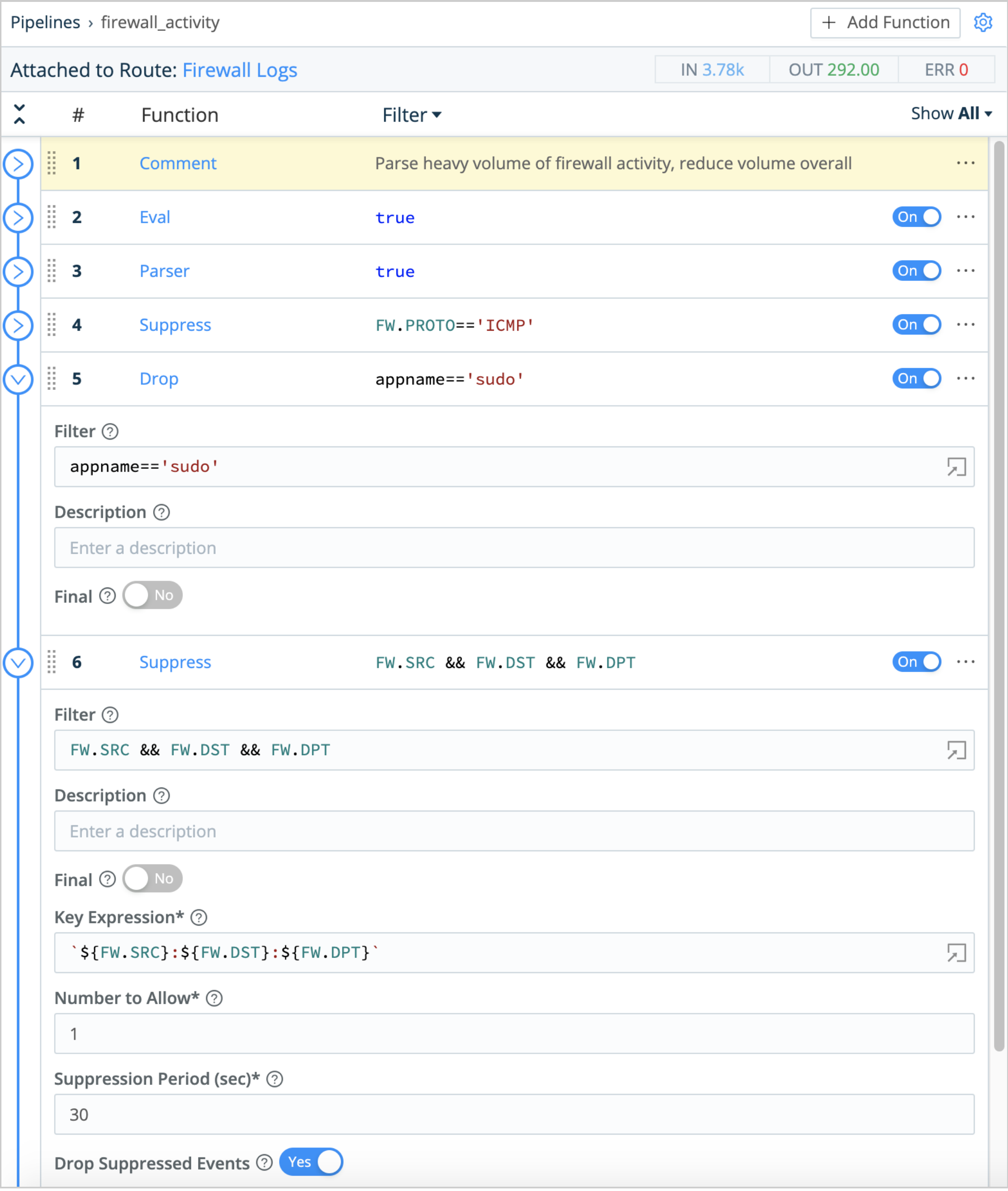
Description Enter a description

Final Yes

Example syslog Route

Processing Pipelines, and Next Steps

For any or all syslog Routes, you can define and attach a processing Pipeline. These can apply more-granular Filters and Functions to further reduce volume, using techniques like Sampling, Dynamic Sampling, or (as shown below) Drop and Suppression. Your most-verbose Syslog Sources are ideal targets for such further processing.


Pipelines › firewall_activity
+ Add Function
Attached to Route: Firewall Logs
IN 3.78k
OUT 292.00
ERR 0
Show All

#	Function	Filter	
1	Comment	Parse heavy volume of firewall activity, reduce volume overall	
2	Eval	true	On
3	Parser	true	On
4	Suppress	FW.PROTO=='ICMP'	On
5	Drop	appname=='sudo'	On

Filter: `appname=='sudo'`

Description: Enter a description

Final: No

#	Function	Filter	
6	Suppress	FW.SRC && FW.DST && FW.DPT	On

Filter: `FW.SRC && FW.DST && FW.DPT`

Description: Enter a description

Final: No

Key Expression*: `` `${FW.SRC}:${FW.DST}:${FW.DPT}` ``

Number to Allow*: 1

Suppression Period (sec)*: 30

Drop Suppressed Events: Yes

Example syslog processing Pipeline

Splunk to Elasticsearch

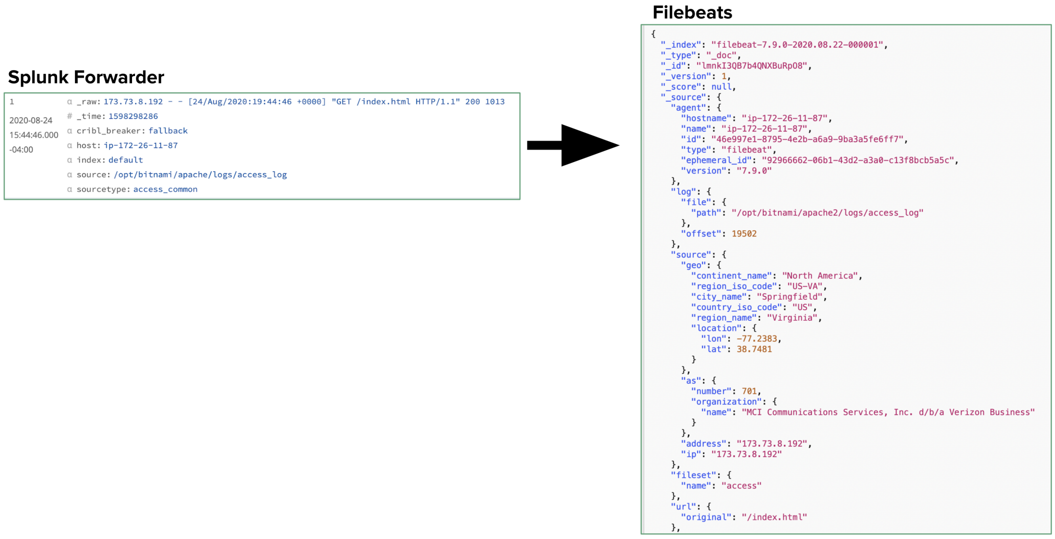
To route data from existing Splunk infrastructure to Elasticsearch services, you might face a daunting task: re-architecting your entire forwarding tier. This could require retooling lots of servers – up to hundreds, or thousands – to uninstall their Splunk forwarders, and swap in Elastic-compatible agents.

Cribl LogStream can reduce this effort to just a few hours: Configure one Splunk `outputs.conf` stanza to output to LogStream, and propagate that across all your Splunk servers. Done!

Next, you can easily configure LogStream 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 LogStream'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.



```
1                 α _raw: 173.73.8.192 - - [24/Aug/2020:19:44:46 +0000] "GET /index.html HTTP/1.1" 200 1013
2020-08-24        # _time: 1598298286
15:44:46.000      α cribl_breaker: fallback
-04:00            α host: ip-172-26-11-87
                  α index: default
                  α source: /opt/bitnami/apache/logs/access_log
                  α sourcetype: access_common
```

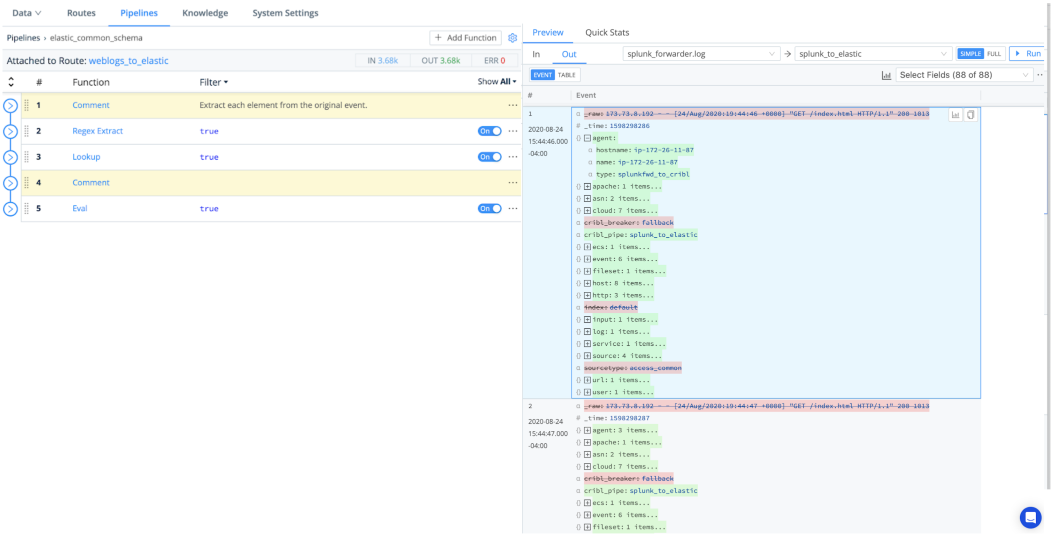
```
{
  "_index": "filebeat-7.9.0-2020.08.22-000001",
  "_type": "_doc",
  "_id": "lmnkI3QB7b4QNXBuRpO8",
  "_version": 1,
  "_score": null,
  "_source": {
    "agent": {
      "hostname": "ip-172-26-11-87",
      "name": "ip-172-26-11-87",
      "id": "46e997e1-8795-4e2b-a6a9-9ba3a5fe6ff7",
      "type": "filebeat",
      "ephemeral_id": "92966662-06b1-43d2-a3a0-c13f8bcb5a5c",
      "version": "7.9.0"
    },
    "log": {
      "file": {
        "path": "/opt/bitnami/apache2/logs/access_log"
      },
      "offset": 19502
    },
    "source": {
      "geo": {
        "continent_name": "North America",
        "region_iso_code": "US-VA",
        "city_name": "Springfield",
        "country_iso_code": "US",
        "region_name": "Virginia",
        "location": {
          "lon": -77.2383,
          "lat": 38.7481
        }
      },
      "as": {
        "number": 701,
        "organization": {
          "name": "MCI Communications Services, Inc. d/b/a Verizon Business"
        }
      },
      "address": "173.73.8.192",
      "ip": "173.73.8.192"
    },
    "fileset": {
      "name": "access"
    },
    "url": {
      "original": "/index.html"
    },
```

Transforming to Elastic Common Schema

Some of the LogStream 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.



#	Function	Filter	
1	Comment	Extract each element from the original event.	
2	Regex Extract	true	On
3	Lookup	true	On
4	Comment		
5	Eval	true	On

splunk_forwarder.log → splunk_to_elastic

#	Event
1 2020-08-24 15:44:46.000 -04:00	~~_raw: 173.73.8.192 - - [24/Aug/2020:19:44:46 +0000] "GET /index.html HTTP/1.1" 200 1013~~ _time: 1598298286 agent: hostname: ip-172-26-11-87 name: ip-172-26-11-87 type: splunkfwd_to_cribl apache: 1 items... asn: 2 items... cloud: 7 items... ~~cribl_breaker: fallback~~ cribl_pipe: splunk_to_elastic ecs: 1 items... event: 6 items... fileset: 1 items... host: 8 items... http: 3 items... ~~index: default~~ input: 1 items... log: 1 items... service: 1 items... source: 4 items... ~~sourcetype: access_common~~ url: 1 items... user: 1 items...
2 2020-08-24 15:44:47.000 -04:00	~~_raw: 173.73.8.192 - - [24/Aug/2020:19:44:47 +0000] "GET /index.html HTTP/1.1" 200 1013~~ _time: 1598298287 agent: 3 items... apache: 1 items... asn: 2 items... cloud: 7 items... ~~cribl_breaker: fallback~~ cribl_pipe: splunk_to_elastic ecs: 1 items... event: 6 items... fileset: 1 items...

LogStream Pipeline and Data Preview

Goat Rid of Some Guesswork

LogStream will offer you further time savings as you configure the internal data transformation. LogStream's 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, LogStream'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:

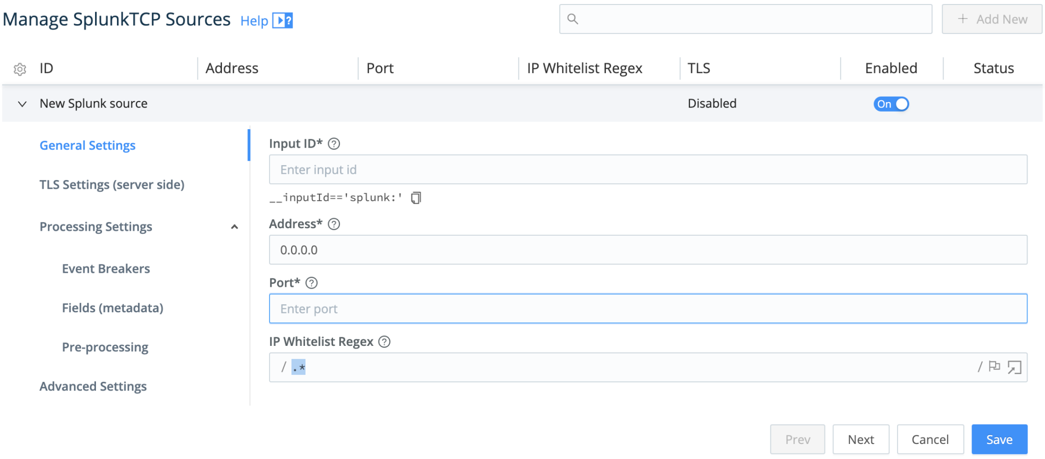
.../outputs.conf

```
[tcpout]
disabled = false
defaultGroup = cribl, <optional_clone_target_group>,

[tcpout:cribl]
server = [<cribl_ip>|<cribl_host>]:<port>, [<cribl_ip>|<cribl_host>]:<port
sendCookedData=true
```

Push the app using the deployment server.

Configure Splunk Source in LogStream

Next, in LogStream, 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.)



Manage SplunkTCP Sources

ID	Address	Port	IP Whitelist Regex	TLS	Enabled	Status
New Splunk source				Disabled	On	

Input ID*: Enter input id

`__inputId=='splunk:'`

Address*: 0.0.0.0

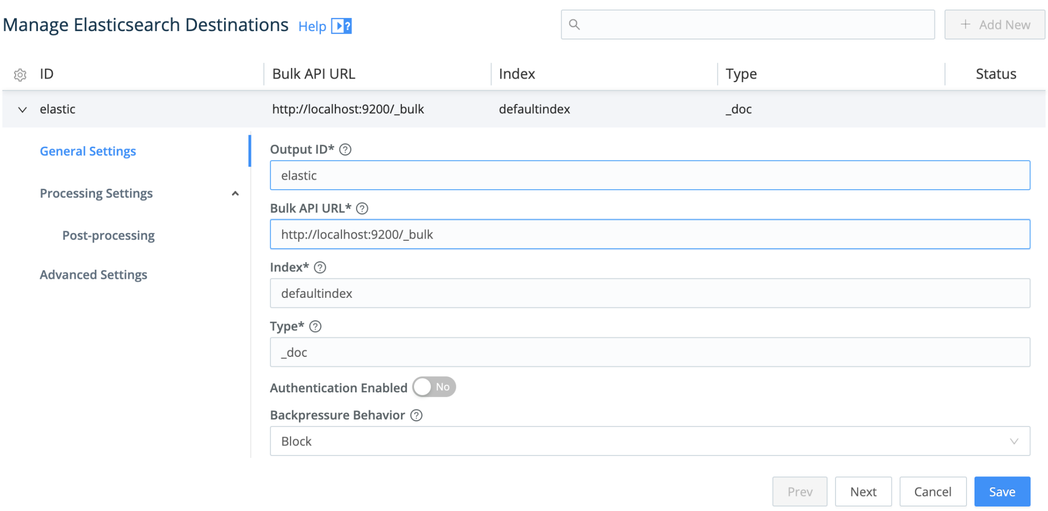
Port*: Enter port

IP Whitelist Regex: / .* /

Splunk Source configuration

Configure Elasticsearch Destination

To configure LogStream's output, set up an Elasticsearch Destination by specifying the **Bulk API URL** and **Index**.


Manage Elasticsearch Destinations Help

ID	Bulk API URL	Index	Type	Status
elastic	http://localhost:9200/_bulk	defaultindex	_doc	

General Settings

Processing Settings

Post-processing

Advanced Settings

Output ID*: elastic

Bulk API URL*: http://localhost:9200/_bulk

Index*: defaultindex

Type*: _doc

Authentication Enabled: No

Backpressure Behavior: Block

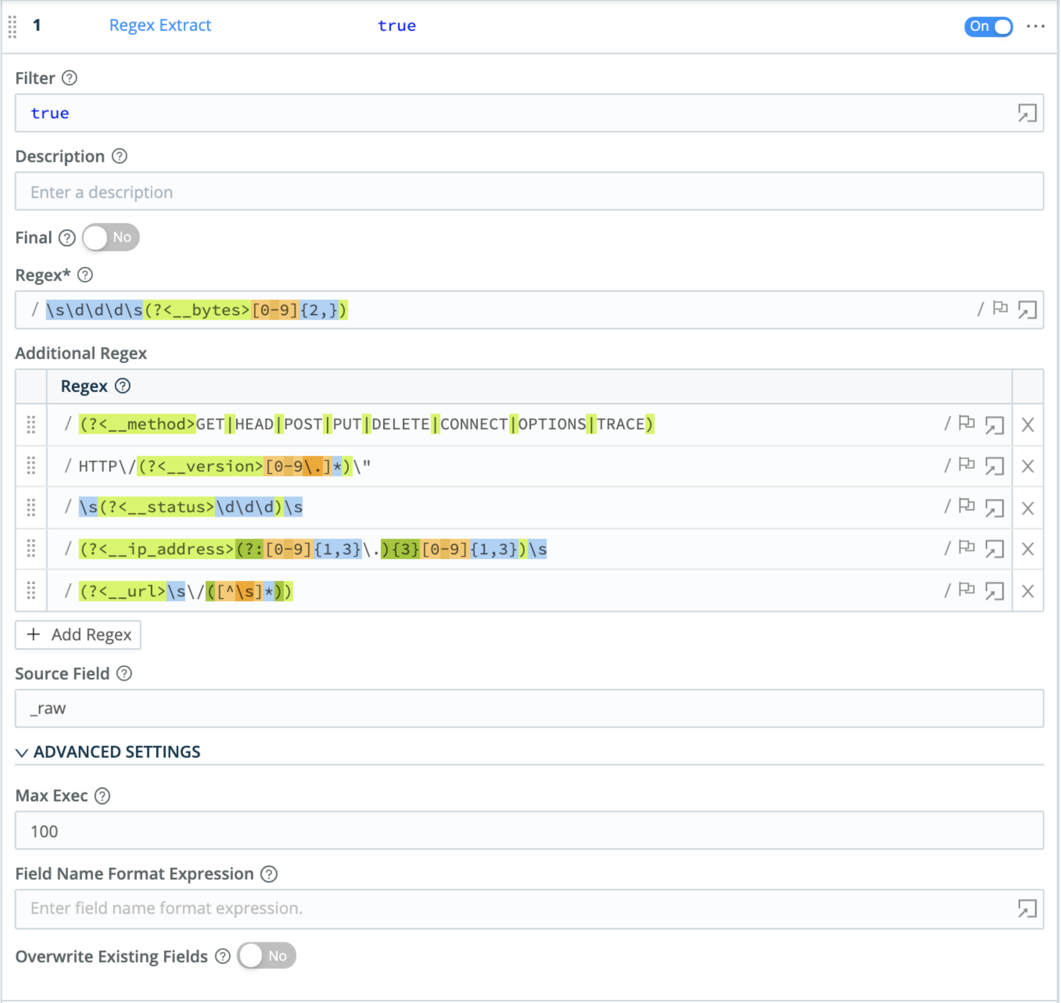
Elasticsearch Destination configuration

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:



1 Regex Extract `true` On

Filter: `true`

Description: Enter a description

Final: No

Regex*: `/ \s\d\d\d\s(?<__bytes>[0-9]{2,}) /`

Additional Regex

```
/ (?<__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]*)) /
```

Source Field: _raw

ADVANCED SETTINGS

Max Exec: 100

Field Name Format Expression: Enter field name format expression.

Overwrite Existing Fields: No

Regex Extract Function

Here are the six rows of regex in this example:

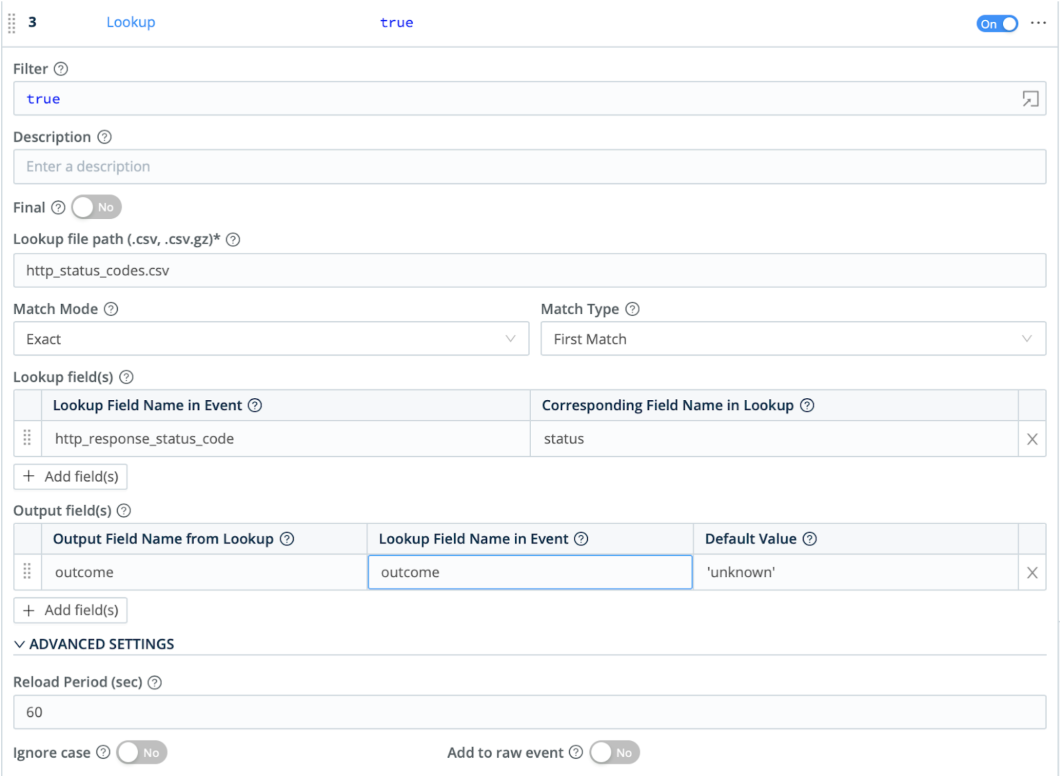
Regex; Additional Regex

```
/\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]*))/
```

As you refine your expression, capture a sample of incoming Splunk data to test your regex's results in LogStream'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.



3 Lookup true On

Filter

true

Description

Enter a description

Final No

Lookup file path (.csv, .csv.gz)*

http_status_codes.csv

Match Mode

Exact

Match Type

First Match

Lookup field(s)

Lookup Field Name in Event	Corresponding Field Name in Lookup
http_response_status_code	status

+ Add field(s)

Output field(s)

Output Field Name from Lookup	Lookup Field Name in Event	Default Value
outcome	outcome	'unknown'

+ Add field(s)

ADVANCED SETTINGS

Reload Period (sec)

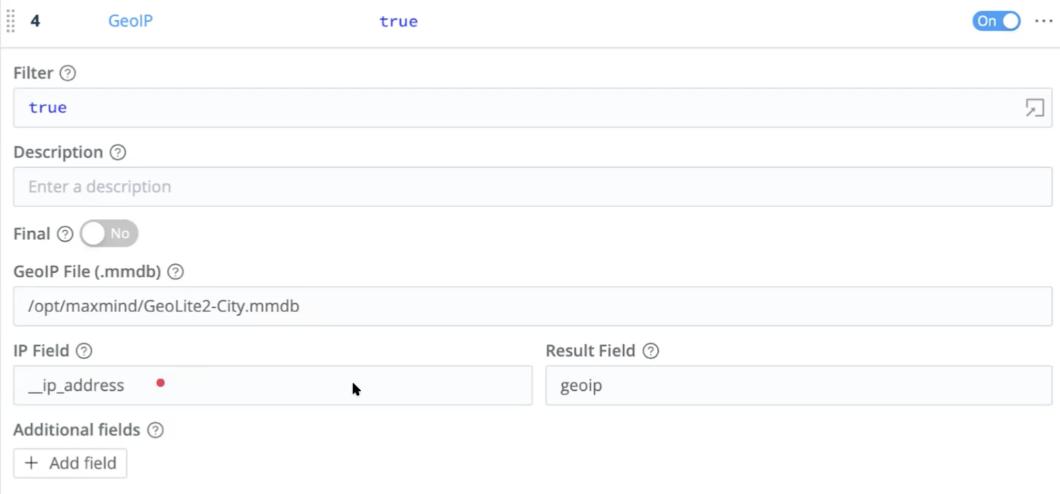
60

Ignore case No

Add to raw event No

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` .



4 GeoIP true On

Filter

true

Description

Enter a description

Final No

GeoIP File (.mmdb)

/opt/maxmind/GeoLite2-City.mmdb

IP Field

__ip_address

Result Field

geoip

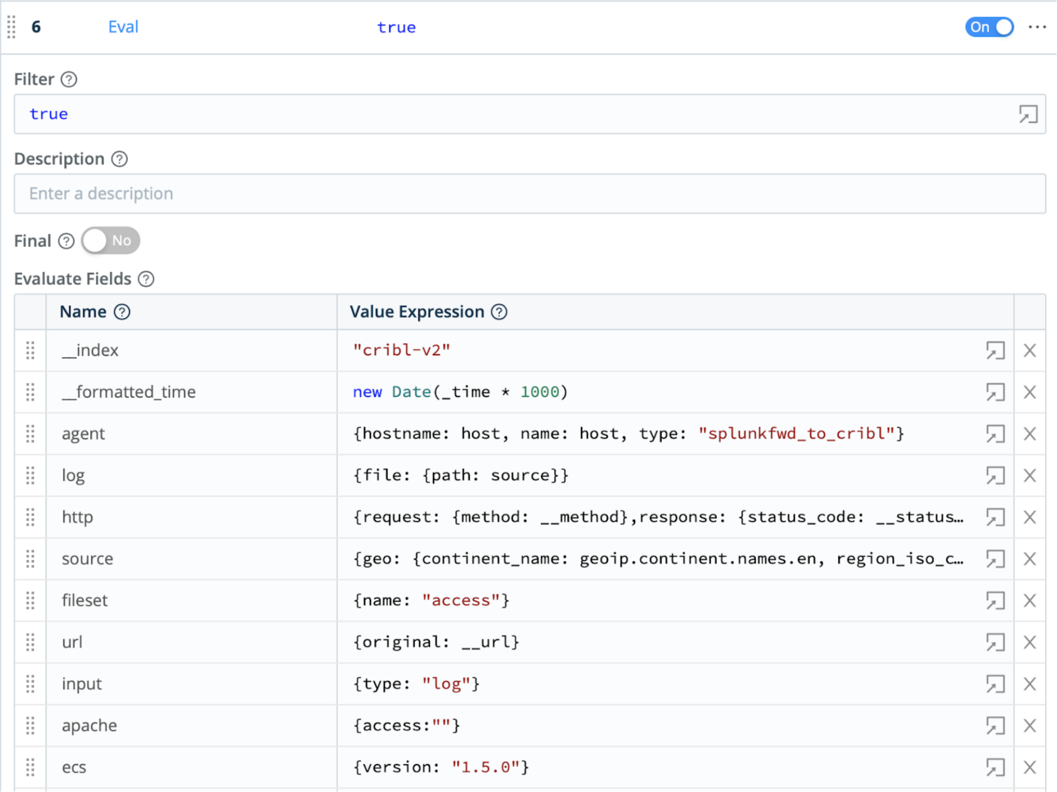
Additional fields

+ Add field

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.



Filter: `true`

Description: Enter a description

Final: No

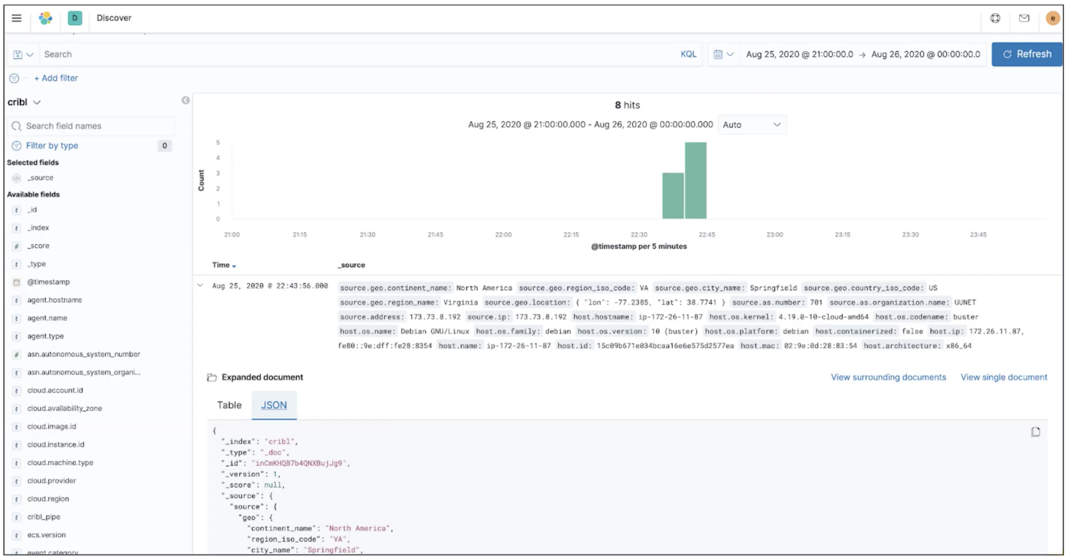
Evaluate Fields

Name	Value Expression
__index	`"cribl-v2"`
__formatted_time	`new Date(_time * 1000)`
agent	`{hostname: host, name: host, type: "splunkfwd_to_cribl"}`
log	`{file: {path: source}}`
http	`{request: {method: __method},response: {status_code: __status…`
source	`{geo: {continent_name: geoip.continent.names.en, region_iso_c…`
fileset	`{name: "access"}`
url	`{original: __url}`
input	`{type: "log"}`
apache	`{access:""}`
ecs	`{version: "1.5.0"}`

Eval Function

Results

After attaching your Pipeline to a Route, here's an an exported event, all happy in Elasticsearch with nested JSON.



```
Aug 25, 2020 @ 22:43:56.000  source.geo.continent_name: North America source.geo.region_iso_code: VA source.geo.city_name: Springfield source.geo.country_iso_code: US
source.geo.region_name: Virginia source.geo.location: { "lon": -77.2385, "lat": 38.7741 } source.as.number: 701 source.as.organization.name: UUNET
source.address: 173.73.8.192 source.ip: 173.73.8.192 host.hostname: ip-172-26-11-87 host.os.kernel: 4.19.0-10-cloud-amd64 host.os.codename: buster
host.os.name: Debian GNU/Linux host.os.family: debian host.os.version: 10 (buster) host.os.platform: debian host.containerized: false host.ip: 172.26.11.87,
fe80::9e:dff:fe28:8354 host.name: ip-172-26-11-87 host.id: 15c09b671e034bcaa16e6e575d2577ea host.mac: 02:9e:0d:28:83:54 host.architecture: x86_64
```

```
{
  "_index": "cribl",
  "_type": "_doc",
  "_id": "inCmKHQB7b4QNXBujJg9",
  "_version": 1,
  "_score": null,
  "_source": {
    "source": {
      "geo": {
        "continent_name": "North America",
        "region_iso_code": "VA",
        "city_name": "Springfield",
```

Event as exported to Elasticsearch

For More Info

For additional details on configuring Splunk forwarders for LogStream, see this related documentation:

- Configuring a Splunk (TCP) Forwarder
- Configuring Cribl App for Splunk on an HF

Reducing Windows XML Events

Here, we demonstrate how to use just a few LogStream Functions to parse WindowsXML events and reduce their volume by 34–70%, dramatically reducing your downstream infrastructure requirements.

Using Eval and C.Text.parseWinEvent

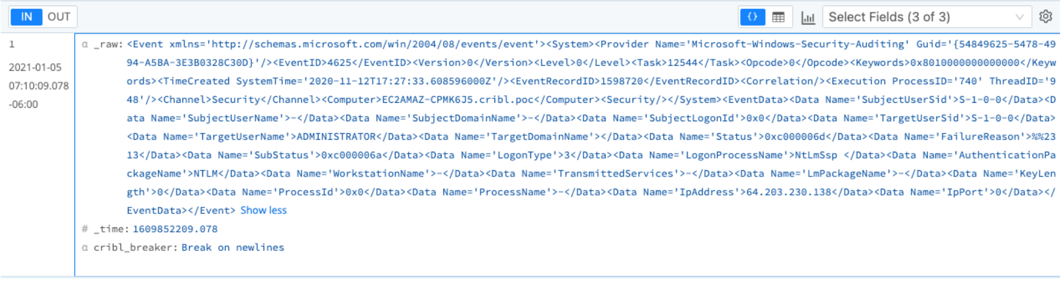
LogStream's internal C.Text.parseWinEvent method parses Windows XML strings and returns a prettified JSON object. You can use this function within an Eval Function to parse an event or an ad hoc XML string. It works like C.Text.parseXml, but with Windows events, it produces more-compact output.

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.

> i (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.

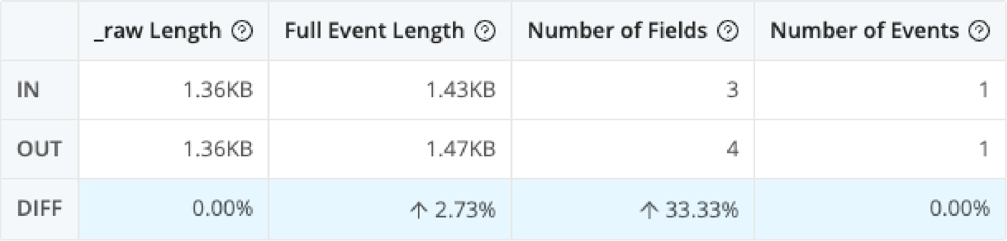
XML: Threat or Menace?

When working with XML, an anonymous Reddit user's quote sums up the challenge: "Some languages can be read by human, 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:



```
1            α _raw: <Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Security-Auditing' Guid='{54849625-5478-49
2021-01-05        94-A5BA-3E3B0328C30D}'/><EventID>4625</EventID><Version>0</Version><Level>0</Level><Task>12544</Task><Opcode>0</Opcode><Keywords>0x8010000000000000</Keyw
07:10:09.078      ords><TimeCreated SystemTime='2020-11-12T17:27:33.608596000Z'/><EventRecordID>1598720</EventRecordID><Correlation/><Execution ProcessID='740' ThreadID='9
-06:00            48'/><Channel>Security</Channel><Computer>EC2AMAZ-CPMK6J5.cribl.poc</Computer><Security/></System><EventData><Data Name='SubjectUserSid'>S-1-0-0</Data><D
                  ata Name='SubjectUserName'>-</Data><Data Name='SubjectDomainName'>-</Data><Data Name='SubjectLogonId'>0x0</Data><Data Name='TargetUserSid'>S-1-0-0</Data>
                  <Data Name='TargetUserName'>ADMINISTRATOR</Data><Data Name='TargetDomainName'></Data><Data Name='Status'>0xc000006d</Data><Data Name='FailureReason'>%%23
                  13</Data><Data Name='SubStatus'>0xc000006a</Data><Data Name='LogonType'>3</Data><Data Name='LogonProcessName'>NtLmSsp </Data><Data Name='AuthenticationPa
                  ckageName'>NTLM</Data><Data Name='WorkstationName'>-</Data><Data Name='TransmittedServices'>-</Data><Data Name='LmPackageName'>-</Data><Data Name='KeyLen
                  gth'>0</Data><Data Name='ProcessId'>0x0</Data><Data Name='ProcessName'>-</Data><Data Name='IpAddress'>64.203.230.138</Data><Data Name='IpPort'>0</Data></
                  EventData></Event> Show less
             # _time: 1609852209.078
             α cribl_breaker: Break on newlines
```

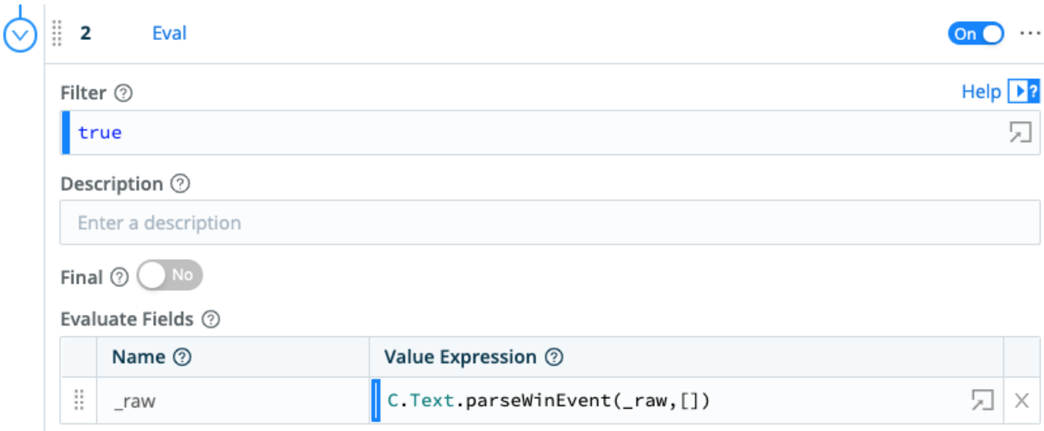
This Windows XML `_raw` event is 1.36KB in size:



	_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:



Filter ⓘ

```
true
```

Description ⓘ

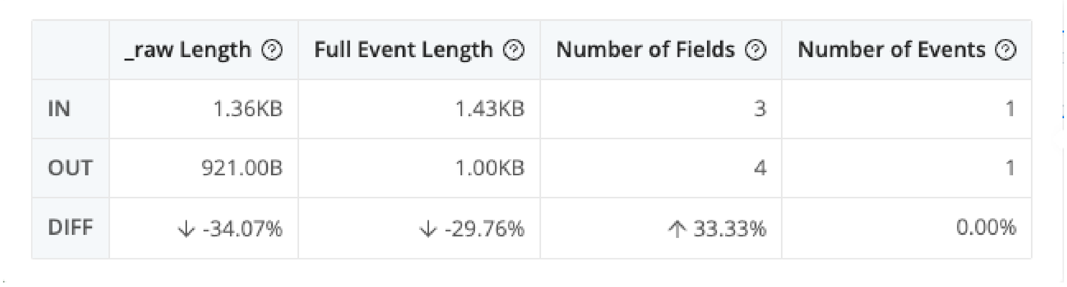
Enter a description

Final ⓘ No

Evaluate Fields ⓘ

Name ⓘ	Value Expression ⓘ
_raw	C.Text.parseWinEvent(_raw,[])

The resulting JSON event is now down to 921.00B in size, a 34.07% reduction of the event:

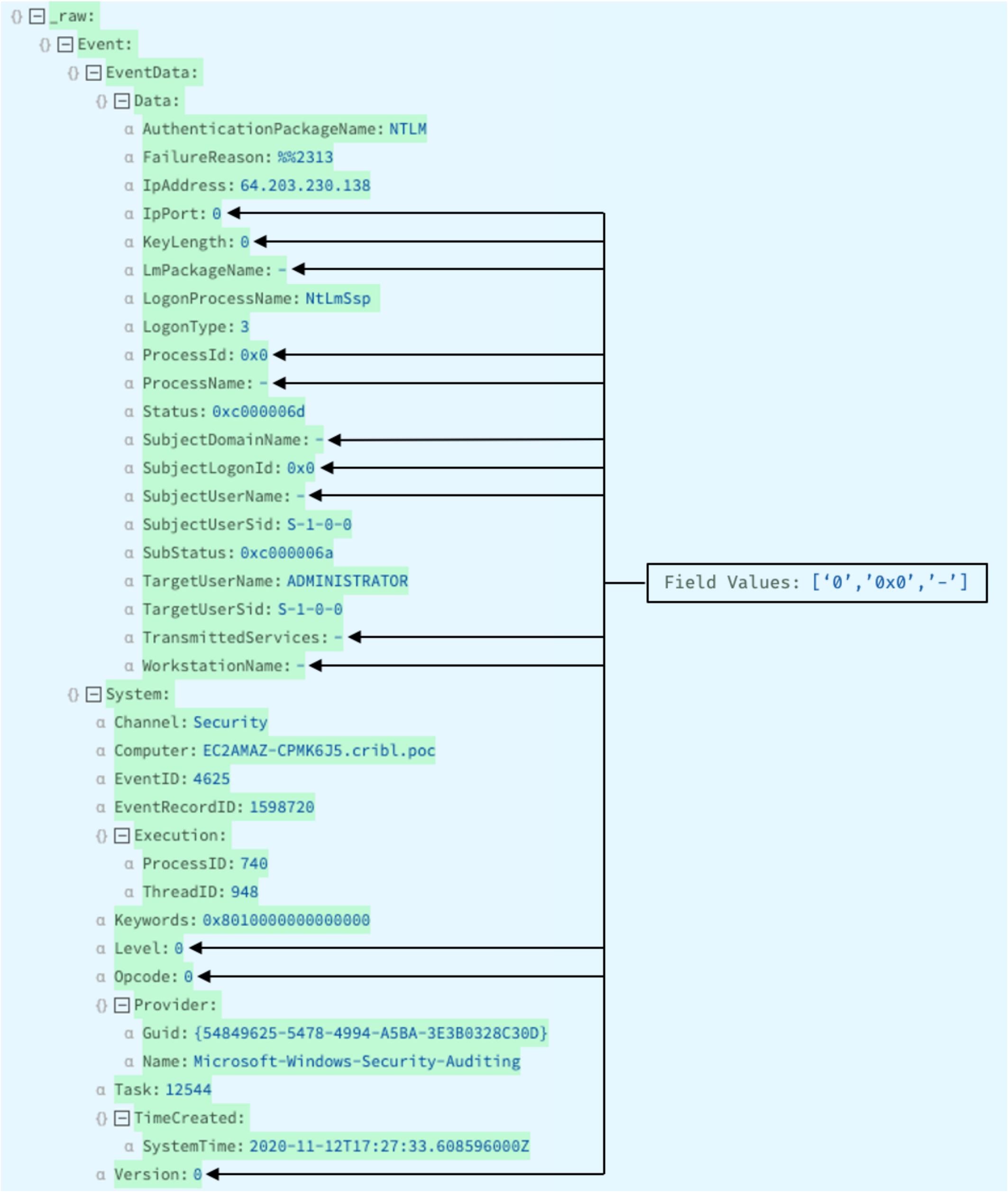


	_raw Length	Full Event Length	Number of Fields	Number of Events
IN	1.36KB	1.43KB	3	1
OUT	921.00B	1.00KB	4	1
DIFF	↓ -34.07%	↓ -29.76%	↑ 33.33%	0.00%

> ℹ 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

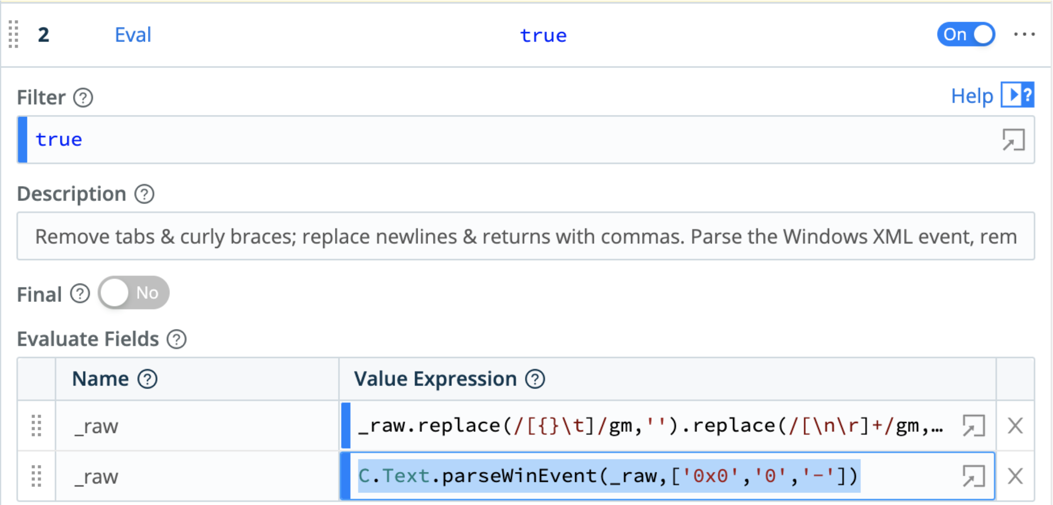
But we can do better. The fields containing essentially null values (`'0','0×0', or '-'`) bloat events, demanding extra infrastructure and storage:



```
_raw:
  Event:
    EventData:
      Data:
        AuthenticationPackageName: NTLM
        FailureReason: %%2313
        IpAddress: 64.203.230.138
        IpPort: 0
        KeyLength: 0
        LmPackageName: -
        LogonProcessName: NtLmSsp
        LogonType: 3
        ProcessId: 0x0
        ProcessName: -
        Status: 0xc000006d
        SubjectDomainName: -
        SubjectLogonId: 0x0
        SubjectUserName: -
        SubjectUserSid: S-1-0-0
        SubStatus: 0xc000006a
        TargetUserName: ADMINISTRATOR
        TargetUserSid: S-1-0-0
        TransmittedServices: -
        WorkstationName: -
    System:
      Channel: Security
      Computer: EC2AMAZ-CPMK6J5.cribl.poc
      EventID: 4625
      EventRecordID: 1598720
      Execution:
        ProcessID: 740
        ThreadID: 948
      Keywords: 0x8010000000000000
      Level: 0
      Opcode: 0
      Provider:
        Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
        Name: Microsoft-Windows-Security-Auditing
      Task: 12544
      TimeCreated:
        SystemTime: 2020-11-12T17:27:33.608596000Z
      Version: 0
```

Field Values: ['0','0x0','-']

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:



2 Eval true On

Filter

```
true
```

Description

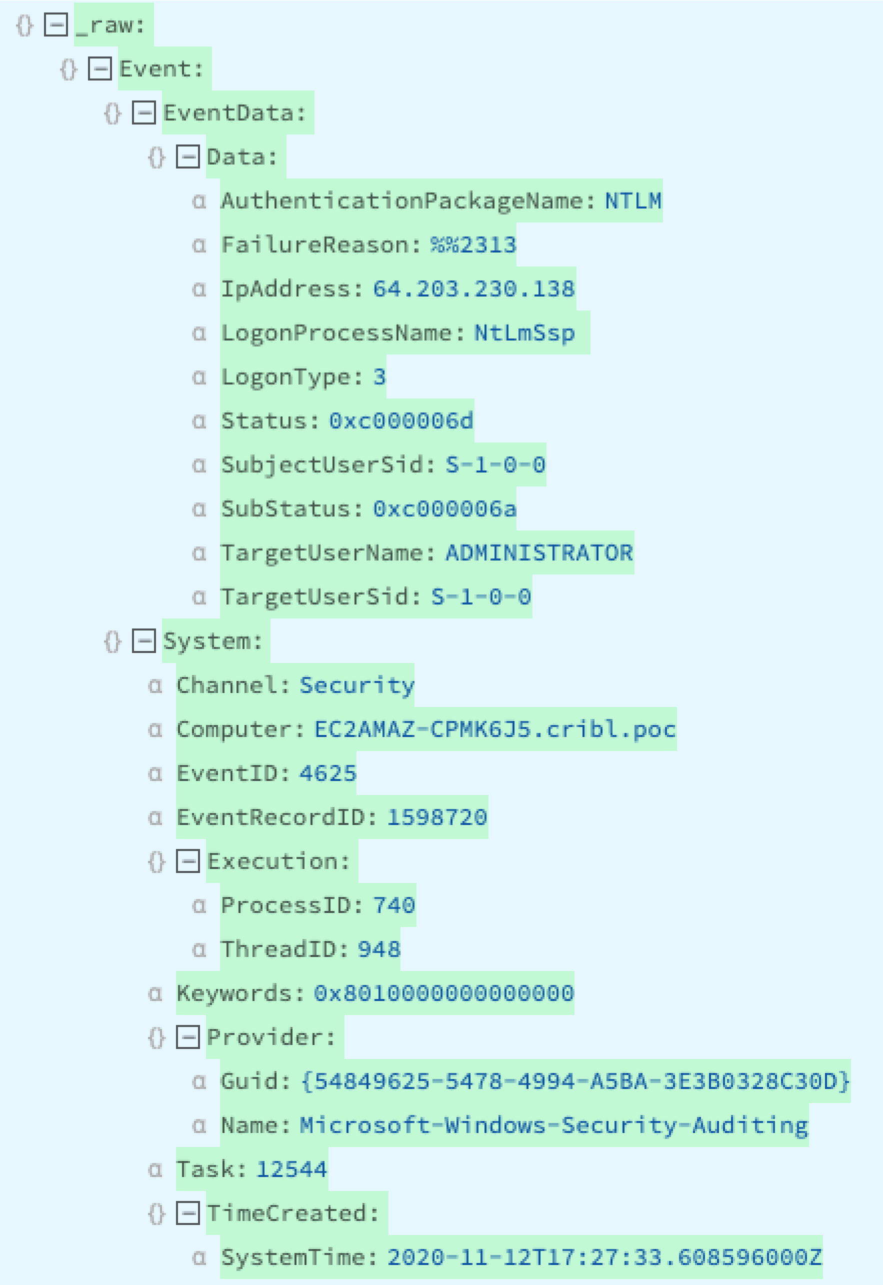
Remove tabs & curly braces; replace newlines & returns with commas. Parse the Windows XML event, rem

Final No

Evaluate Fields

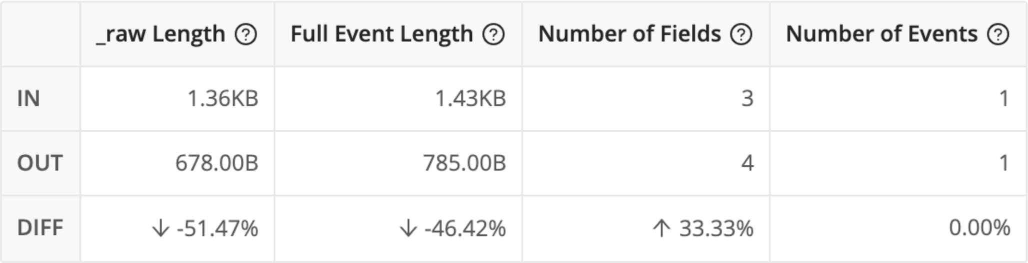
Name	Value Expression
_raw	`_raw.replace(/[{}\t]/gm,'').replace(/[\n\r]+/gm,…`
_raw	`C.Text.parseWinEvent(_raw,['0x0','0','-'])`

If you compare this Preview-pane screenshot to the Preview screenshot above, you can confirm that the fields with values matching `[‘0’,‘0×0’,’-’]` are removed:



```
_raw:
  Event:
    EventData:
      Data:
        AuthenticationPackageName: NTLM
        FailureReason: %%2313
        IpAddress: 64.203.230.138
        LogonProcessName: NtLmSsp
        LogonType: 3
        Status: 0xc000006d
        SubjectUserSid: S-1-0-0
        SubStatus: 0xc000006a
        TargetUserName: ADMINISTRATOR
        TargetUserSid: S-1-0-0
    System:
      Channel: Security
      Computer: EC2AMAZ-CPMK6J5.cribl.poc
      EventID: 4625
      EventRecordID: 1598720
      Execution:
        ProcessID: 740
        ThreadID: 948
      Keywords: 0x8010000000000000
      Provider:
        Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
        Name: Microsoft-Windows-Security-Auditing
      Task: 12544
      TimeCreated:
        SystemTime: 2020-11-12T17:27:33.608596000Z
```

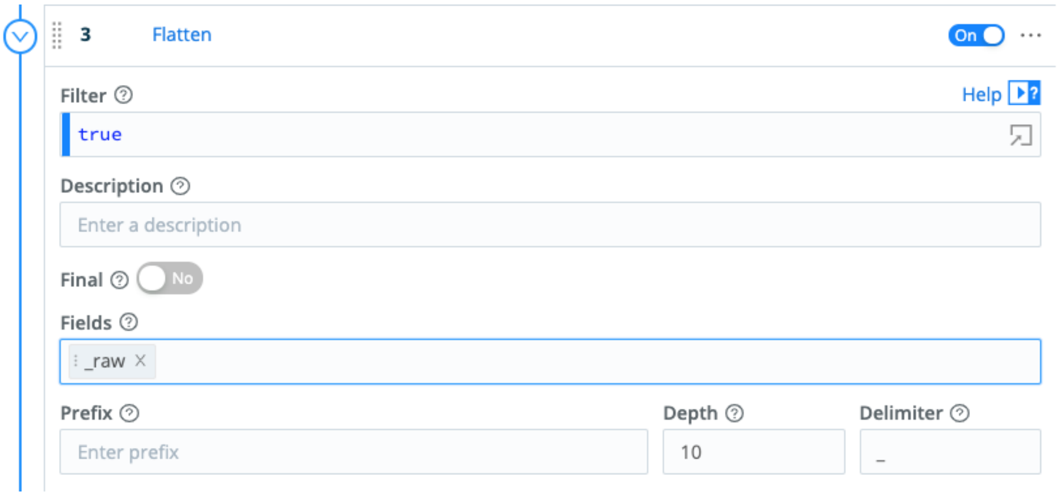
The event is now down to 678.00B in size, translating to a 51.47% reduction from the original event:



	_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

LogStream'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:



Filter: true

Description: Enter a description

Final: No

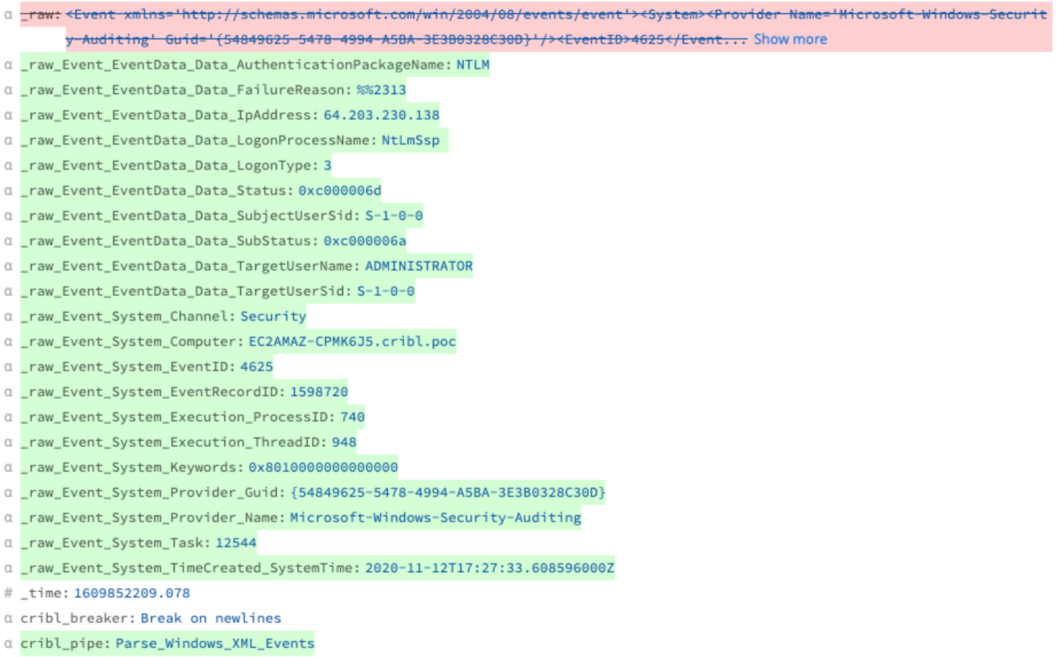
Fields: _raw

Prefix: Enter prefix

Depth: 10

Delimiter: _

Using Flatten, we've successfully created top-level fields from the nested JSON structure:



```
α ~~_raw: <Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Securit
      y-Auditing' Guid='{54849625-5478-4994-A5BA-3E3B0328C30D}'/><EventID>4625</Event...~~ Show more
α _raw_Event_EventData_Data_AuthenticationPackageName: NTLM
α _raw_Event_EventData_Data_FailureReason: %%2313
α _raw_Event_EventData_Data_IpAddress: 64.203.230.138
α _raw_Event_EventData_Data_LogonProcessName: NtLmSsp
α _raw_Event_EventData_Data_LogonType: 3
α _raw_Event_EventData_Data_Status: 0xc000006d
α _raw_Event_EventData_Data_SubjectUserSid: S-1-0-0
α _raw_Event_EventData_Data_SubStatus: 0xc000006a
α _raw_Event_EventData_Data_TargetUserName: ADMINISTRATOR
α _raw_Event_EventData_Data_TargetUserSid: S-1-0-0
α _raw_Event_System_Channel: Security
α _raw_Event_System_Computer: EC2AMAZ-CPMK6J5.cribl.poc
α _raw_Event_System_EventID: 4625
α _raw_Event_System_EventRecordID: 1598720
α _raw_Event_System_Execution_ProcessID: 740
α _raw_Event_System_Execution_ThreadID: 948
α _raw_Event_System_Keywords: 0x8010000000000000
α _raw_Event_System_Provider_Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
α _raw_Event_System_Provider_Name: Microsoft-Windows-Security-Auditing
α _raw_Event_System_Task: 12544
α _raw_Event_System_TimeCreated_SystemTime: 2020-11-12T17:27:33.608596000Z
# _time: 1609852209.078
α cribl_breaker: Break on newlines
α cribl_pipe: Parse_Windows_XML_Events
```

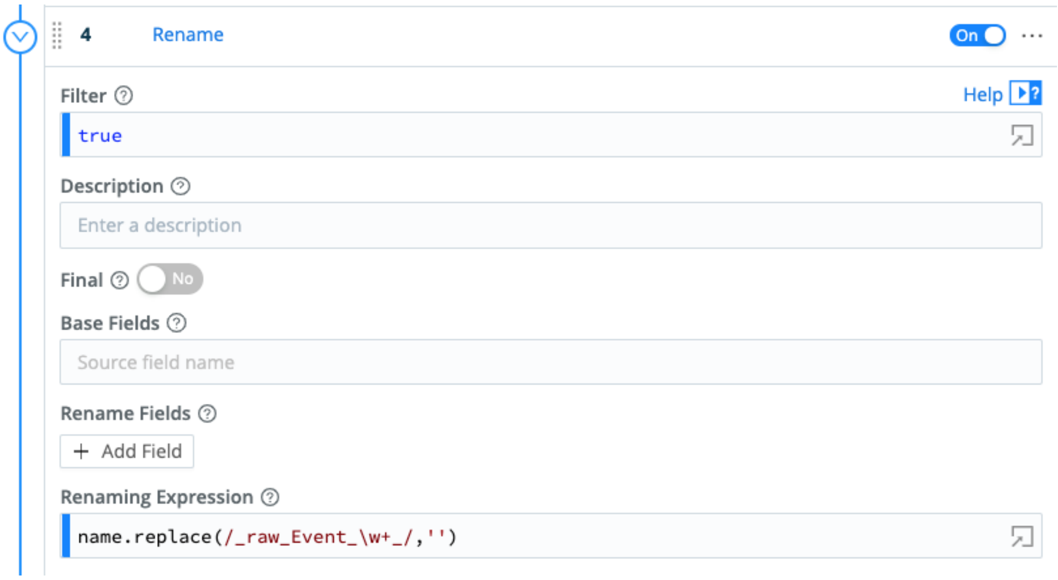
> Don't worry about the `_raw` field's deletion (red strikeout). This is the Flatten Function's default behavior. We'll restore `_raw` after we clean and reduce the event even more.

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 **Renaming Expression**, we build a JavaScript expression to match the field names' prefixes (up to the underscore):



4 Rename On

Filter: true

Description: Enter a description

Final: No

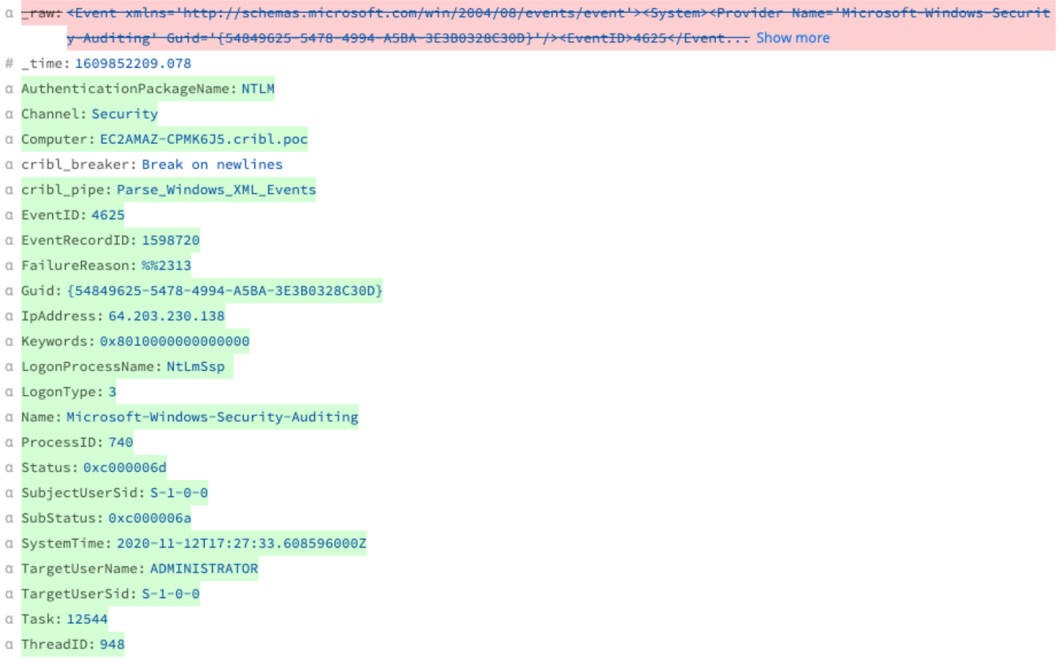
Base Fields: Source field name

Rename Fields: + Add Field

Renaming Expression:

```
name.replace(/_raw_Event_\w+_/,'')
```

The resulting field names are now much more compact, and easier to work with and manage:

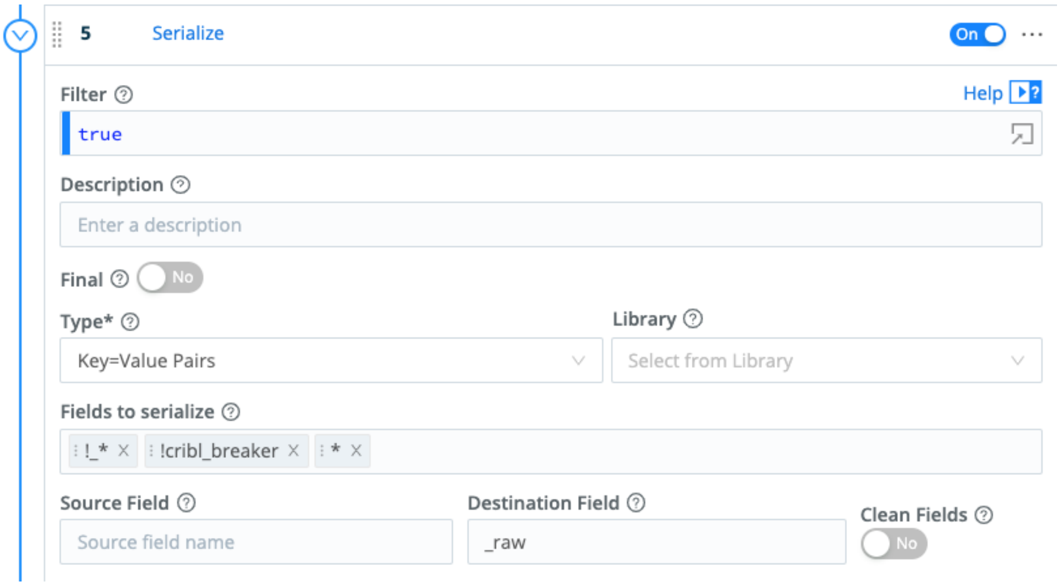


```
α ~~_raw: <Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-Securit
       y-Auditing' Guid='{54849625-5478-4994-A5BA-3E3B0328C30D}'/><EventID>4625</Event...~~ Show more
# _time: 1609852209.078
α AuthenticationPackageName: NTLM
α Channel: Security
α Computer: EC2AMAZ-CPMK6J5.cribl.poc
α cribl_breaker: Break on newlines
α cribl_pipe: Parse_Windows_XML_Events
α EventID: 4625
α EventRecordID: 1598720
α FailureReason: %%2313
α Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
α IpAddress: 64.203.230.138
α Keywords: 0x8010000000000000
α LogonProcessName: NtLmSsp
α LogonType: 3
α Name: Microsoft-Windows-Security-Auditing
α ProcessID: 740
α Status: 0xc000006d
α SubjectUserSid: S-1-0-0
α SubStatus: 0xc000006a
α SystemTime: 2020-11-12T17:27:33.608596000Z
α TargetUserName: ADMINISTRATOR
α TargetUserSid: S-1-0-0
α Task: 12544
α ThreadID: 948
```

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.) Here, Serialize takes the extracted fields and puts them back into `_raw`:



5 Serialize On

Filter: true

Description: Enter a description

Final: No

Type*: Key=Value Pairs

Library: Select from Library

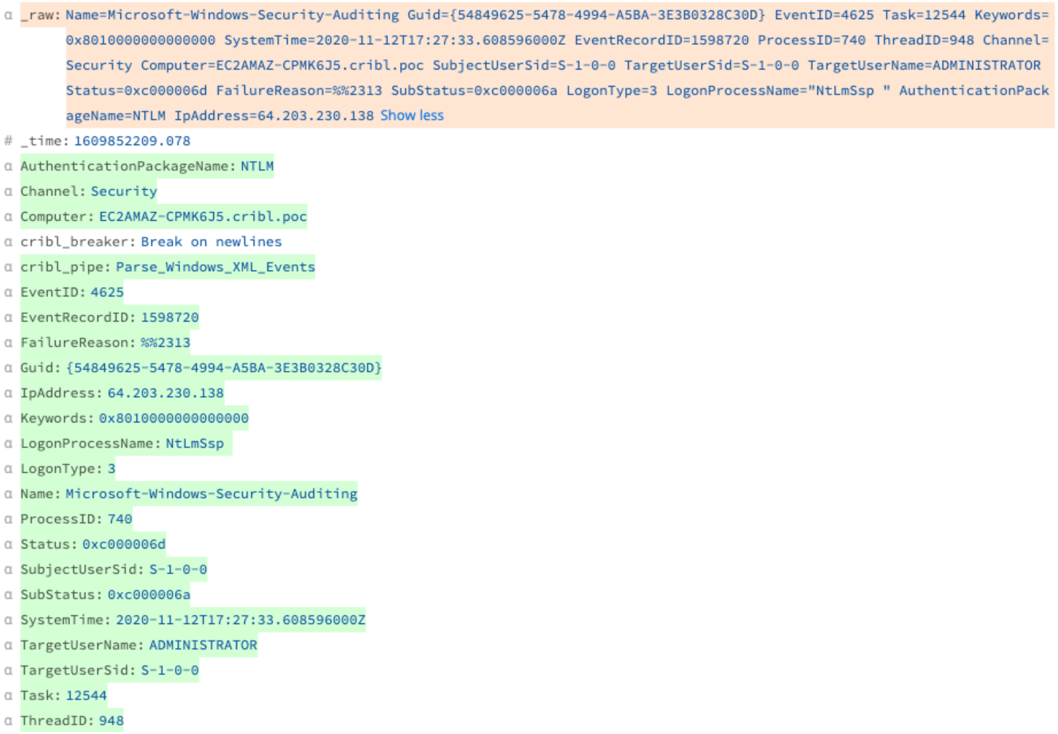
Fields to serialize: !_* !cribl_breaker *

Source Field: Source field name

Destination Field: _raw

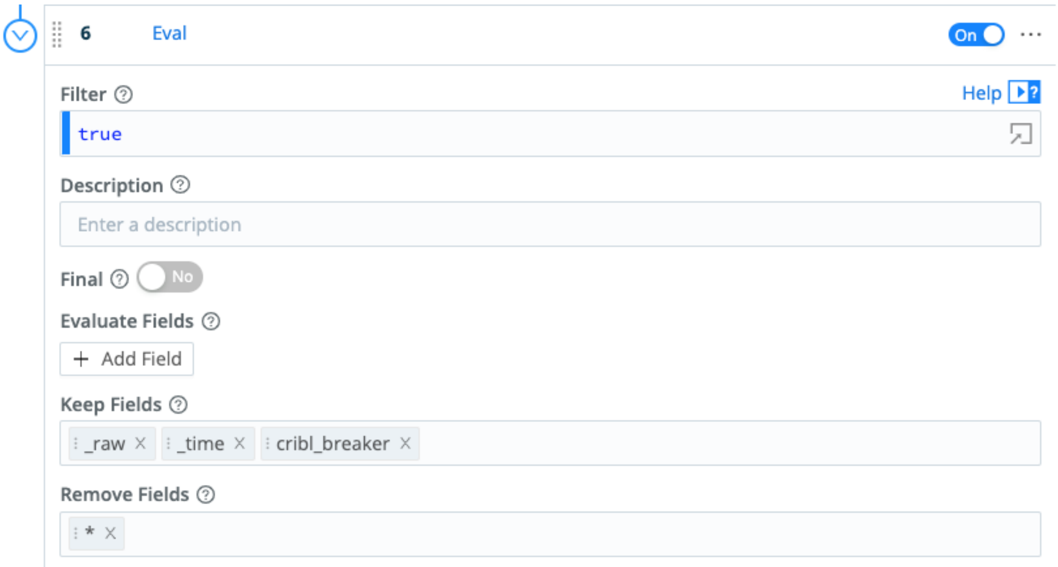
Clean Fields: No

In the Preview pane, the `_raw` field is now back, serialized into compact, tidy key-value pairs:



```
α _raw: Name=Microsoft-Windows-Security-Auditing Guid={54849625-5478-4994-A5BA-3E3B0328C30D} EventID=4625 Task=12544 Keywords=
        0x8010000000000000 SystemTime=2020-11-12T17:27:33.608596000Z EventRecordID=1598720 ProcessID=740 ThreadID=948 Channel=
        Security Computer=EC2AMAZ-CPMK6J5.cribl.poc SubjectUserSid=S-1-0-0 TargetUserSid=S-1-0-0 TargetUserName=ADMINISTRATOR
        Status=0xc000006d FailureReason=%%2313 SubStatus=0xc000006a LogonType=3 LogonProcessName="NtLmSsp " AuthenticationPack
        ageName=NTLM IpAddress=64.203.230.138 Show less
# _time: 1609852209.078
α AuthenticationPackageName: NTLM
α Channel: Security
α Computer: EC2AMAZ-CPMK6J5.cribl.poc
α cribl_breaker: Break on newlines
α cribl_pipe: Parse_Windows_XML_Events
α EventID: 4625
α EventRecordID: 1598720
α FailureReason: %%2313
α Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
α IpAddress: 64.203.230.138
α Keywords: 0x8010000000000000
α LogonProcessName: NtLmSsp
α LogonType: 3
α Name: Microsoft-Windows-Security-Auditing
α ProcessID: 740
α Status: 0xc000006d
α SubjectUserSid: S-1-0-0
α SubStatus: 0xc000006a
α SystemTime: 2020-11-12T17:27:33.608596000Z
α TargetUserName: ADMINISTRATOR
α TargetUserSid: S-1-0-0
α Task: 12544
α ThreadID: 948
```

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:



Filter: `true`

Description: Enter a description

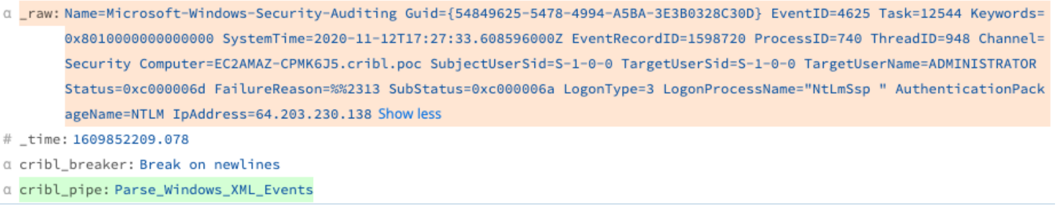
Final: No

Evaluate Fields: + Add Field

Keep Fields: _raw, _time, cribl_breaker

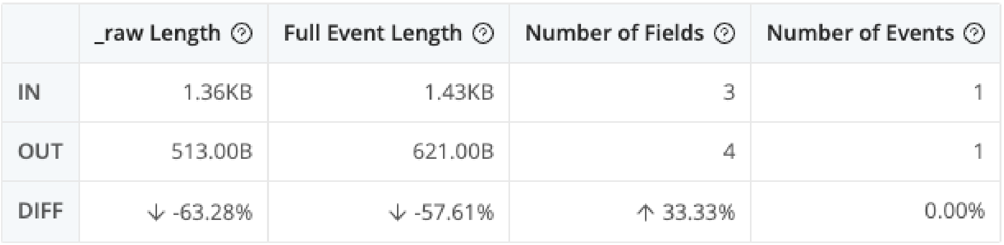
Remove Fields: *

To sum up, we've successfully transformed the original Windows XML event into key-value pairs:



```
α _raw: Name=Microsoft-Windows-Security-Auditing Guid={54849625-5478-4994-A5BA-3E3B0328C30D} EventID=4625 Task=12544 Keywords=
        0x8010000000000000 SystemTime=2020-11-12T17:27:33.608596000Z EventRecordID=1598720 ProcessID=740 ThreadID=948 Channel=
        Security Computer=EC2AMAZ-CPMK6J5.cribl.poc SubjectUserSid=S-1-0-0 TargetUserSid=S-1-0-0 TargetUserName=ADMINISTRATOR
        Status=0xc000006d FailureReason=%%2313 SubStatus=0xc000006a LogonType=3 LogonProcessName="NtLmSsp " AuthenticationPack
        ageName=NTLM IpAddress=64.203.230.138 Show less
# _time: 1609852209.078
α cribl_breaker: Break on newlines
α cribl_pipe: Parse_Windows_XML_Events
```

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 LogStream 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 LogStream Pipeline reduces Microsoft Windows XM
        },
        "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 & ret
      },
      {
        "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+_/,'')",
```

Finally, here's a sample of Windows XML events that you can upload to LogStream's Preview pane to try this out:

Sample data

```
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><Syst
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><Syst
            SeBackupPrivilege
            SeRestorePrivilege
            SeTakeOwnershipPrivilege
            SeDebugPrivilege
            SeSystemEnvironmentPrivilege
            SeLoadDriverPrivilege
            SeImpersonatePrivilege
            SeDelegateSessionUserImpersonatePrivilege
            SeEnableDelegationPrivilege</Data></EventData></Event>
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><Syst
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><Syst
<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><Syst
```

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.

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 Headers:

`accept: 'application/json'`

`x-rapidapi-key: 'e4068647ffmsh65536596798f49dp17e998jsn342bac862377'`

`x-rapidapi-host: 'matchilling-chuck-norris-jokes-v1.p.rapidapi.com'`

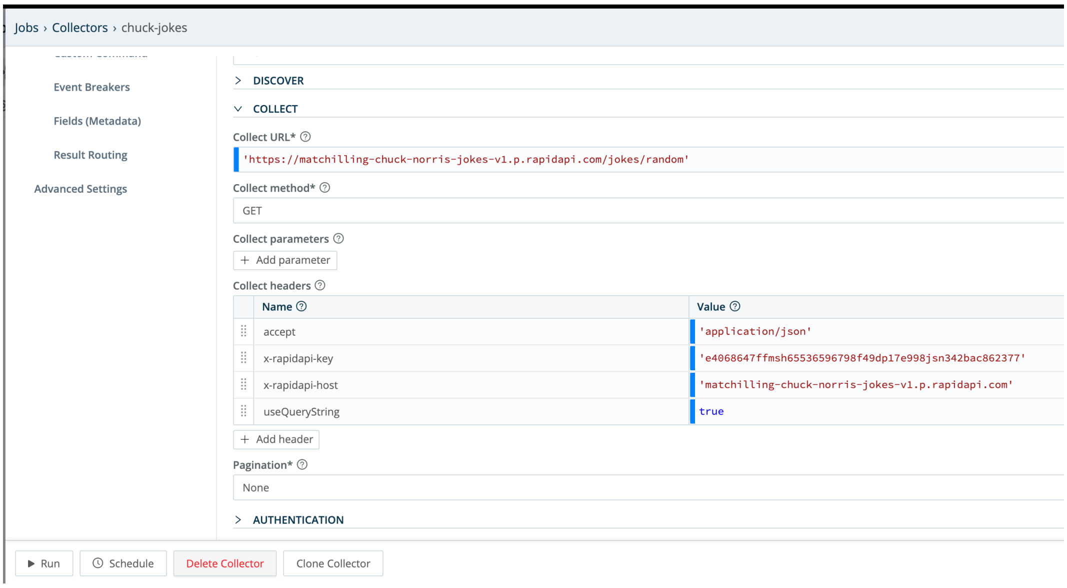
`useQueryString: true`

Collect Request Parameters: None

Pagination: None

Authentication: None

Event Breaker: JSON Newline Delimited – use LogStream's built-in **Cribl > ndjson** rule, and associate it with the Collector to parse the JSON document.



Collect URL*: 'https://matchilling-chuck-norris-jokes-v1.p.rapidapi.com/jokes/random'

Collect method*: GET

Collect parameters: + Add parameter

Collect headers:

Name	Value
accept	'application/json'
x-rapidapi-key	'e4068647ffmsh65536596798f49dp17e998jsn342bac862377'
x-rapidapi-host	'matchilling-chuck-norris-jokes-v1.p.rapidapi.com'
useQueryString	true

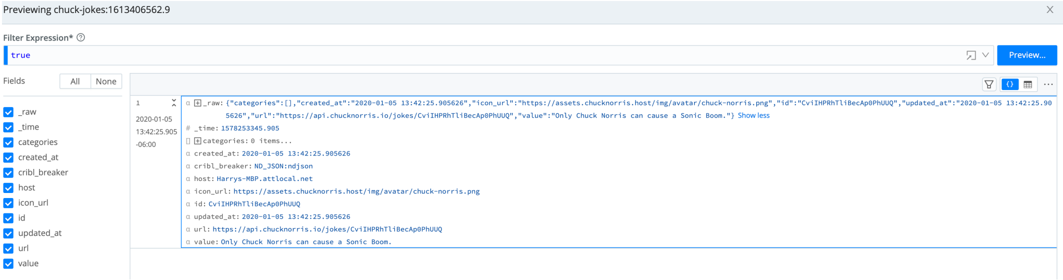
+ Add header

Pagination*: None

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:



Filter Expression*: true

☑ _raw
☑ _time
☑ categories
☑ created_at
☑ cribl_breaker
☑ host
☑ icon_url
☑ id
☑ updated_at
☑ url
☑ value

```
1    2020-01-05 13:42:25.905 -06:00
_raw: {"categories":[],"created_at":"2020-01-05 13:42:25.905626","icon_url":"https://assets.chucknorris.host/img/avatar/chuck-norris.png","id":"CviIHPRhTliBecAp0PhUUQ","updated_at":"2020-01-05 13:42:25.90
      5626","url":"https://api.chucknorris.io/jokes/CviIHPRhTliBecAp0PhUUQ","value":"Only Chuck Norris can cause a Sonic Boom."} Show less
# _time: 1578253345.905
[] categories: 0 items...
created_at: 2020-01-05 13:42:25.905626
cribl_breaker: ND_JSON:ndjson
host: Harrys-MBP.attlocal.net
icon_url: https://assets.chucknorris.host/img/avatar/chuck-norris.png
id: CviIHPRhTliBecAp0PhUUQ
updated_at: 2020-01-05 13:42:25.905626
url: https://api.chucknorris.io/jokes/CviIHPRhTliBecAp0PhUUQ
value: Only Chuck Norris can cause a Sonic Boom.
```

Returned event

2. HTTP GET with Pagination via URL Attribute

The REST Collector's Pagination feature (available in LogStream 2.4.3 and above) 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

Max Pages setting is reached, or no more pages are present (i.e., 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 Headers: None

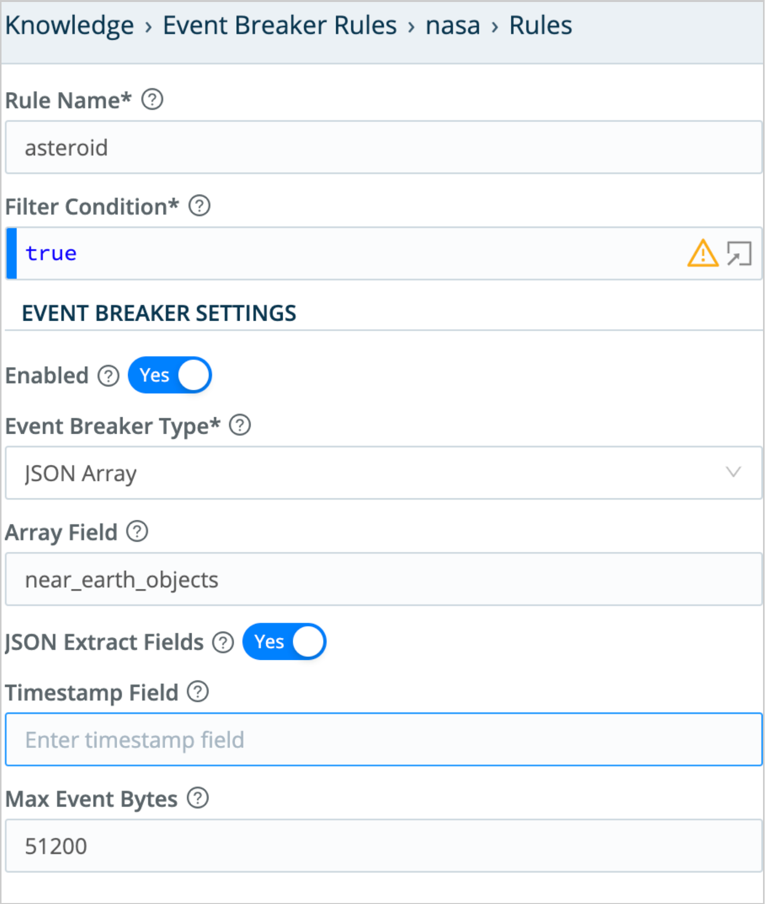
Collect Request Parameters: None – Parameters in this example are added to the header. Static parameters (i.e., parameters that don't reference variables) can safely be added to the URL. Any parameters that do reference variables should always be added in the **Collect parameters** section, to allow filtering of values that evaluate as undefined.

Pagination: Response Body Attribute

Response Attribute: `next`

Authentication: None

Event Breaker: JSON Array



Knowledge › Event Breaker Rules › nasa › Rules

Rule Name*

asteroid

Filter Condition*

`true`

EVENT BREAKER SETTINGS

Enabled: Yes

Event Breaker Type*

JSON Array

Array Field

near_earth_objects

JSON Extract Fields: Yes

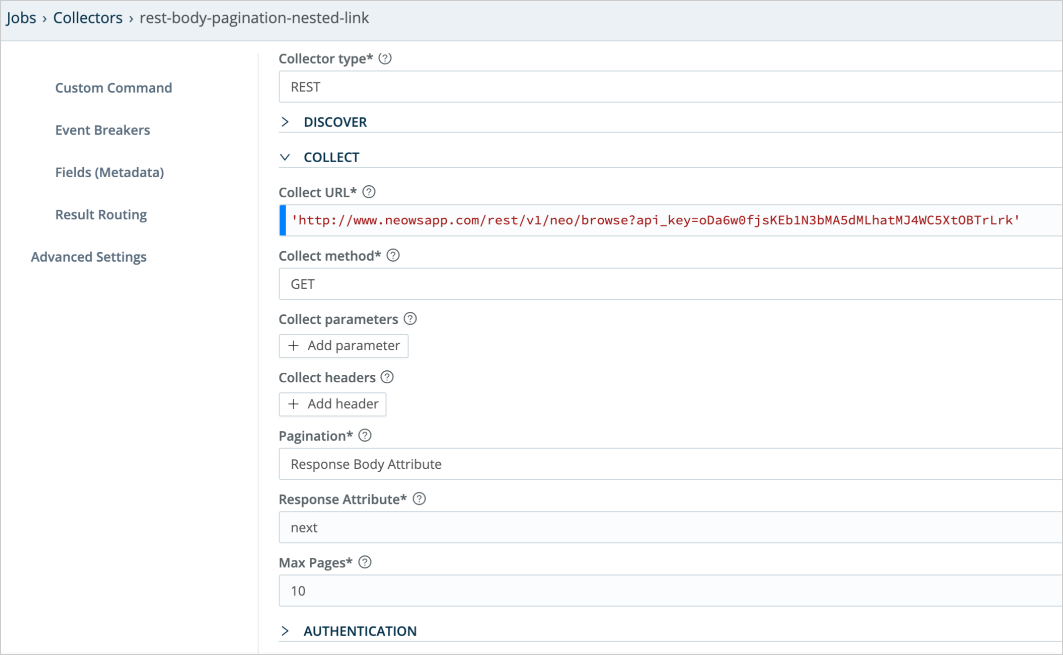
Timestamp Field

Enter timestamp field

Max Event Bytes

51200

Event Breaker configuration



Jobs › Collectors › rest-body-pagination-nested-link

Collector type*

REST

> DISCOVER

COLLECT

Collect URL*

```
'http://www.neowsapp.com/rest/v1/neo/browse?api_key=oDa6w0fjsKEb1N3bMA5dMLhatMJ4WC5XtOBTrLrk'
```

Collect method*

GET

Collect parameters

+ Add parameter

Collect headers

+ Add header

Pagination*

Response Body Attribute

Response Attribute*

next

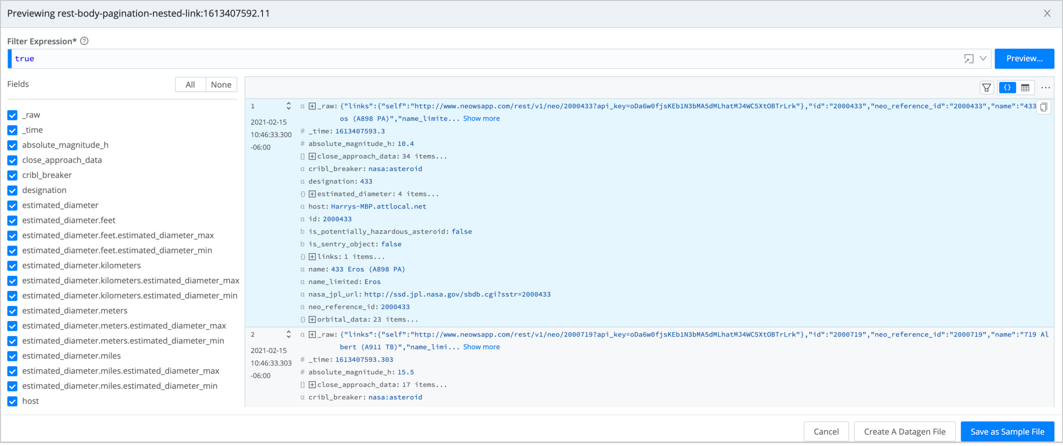
Max Pages*

10

> AUTHENTICATION

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.



Previewing rest-body-pagination-nested-link:1613407592.11

Filter Expression*: `true`

Fields

- ☑ _raw
- ☑ _time
- ☑ absolute_magnitude_h
- ☑ close_approach_data
- ☑ cribl_breaker
- ☑ designation
- ☑ estimated_diameter
- ☑ estimated_diameter.feet
- ☑ estimated_diameter.feet.estimated_diameter_max
- ☑ estimated_diameter.feet.estimated_diameter_min
- ☑ estimated_diameter.kilometers
- ☑ estimated_diameter.kilometers.estimated_diameter_max
- ☑ estimated_diameter.kilometers.estimated_diameter_min
- ☑ estimated_diameter.meters
- ☑ estimated_diameter.meters.estimated_diameter_max
- ☑ estimated_diameter.meters.estimated_diameter_min
- ☑ estimated_diameter.miles
- ☑ estimated_diameter.miles.estimated_diameter_max
- ☑ estimated_diameter.miles.estimated_diameter_min
- ☑ host

```
1             α ⊞_raw: {"links":{"self":"http://www.neowsapp.com/rest/v1/neo/2000433?api_key=oDa6w0fjsKEb1N3bMA5dMLhatMJ4WC5XtOBTrLrk"},"id":"2000433","neo_reference_id":"2000433","name":"433
2021-02-15             os (A898 PA)","name_limite... Show more
10:46:33.300  # _time: 1613407593.3
-06:00        # absolute_magnitude_h: 10.4
              [] ⊞close_approach_data: 34 items...
              α cribl_breaker: nasa:asteroid
              α designation: 433
              {} ⊞estimated_diameter: 4 items...
              α host: Harrys-MBP.attlocal.net
              α id: 2000433
              b is_potentially_hazardous_asteroid: false
              b is_sentry_object: false
              {} ⊞links: 1 items...
              α name: 433 Eros (A898 PA)
              α name_limited: Eros
              α nasa_jpl_url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2000433
              α neo_reference_id: 2000433
              {} ⊞orbital_data: 23 items...
2             α ⊞_raw: {"links":{"self":"http://www.neowsapp.com/rest/v1/neo/2000719?api_key=oDa6w0fjsKEb1N3bMA5dMLhatMJ4WC5XtOBTrLrk"},"id":"2000719","neo_reference_id":"2000719","name":"719 Al
2021-02-15             bert (A911 TB)","name_limi... Show more
10:46:33.303  # _time: 1613407593.303
-06:00        # absolute_magnitude_h: 15.5
              [] ⊞close_approach_data: 17 items...
              α cribl_breaker: nasa:asteroid
```

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 Request Parameters:

```
query: 'last_seen:1d'

scroll: `${scroll}`
```

Collect Headers:

```
accept: 'application/json'
```

```
key: '<your-GreyNoise-API-key-here>
```

Pagination: Response Body Attribute

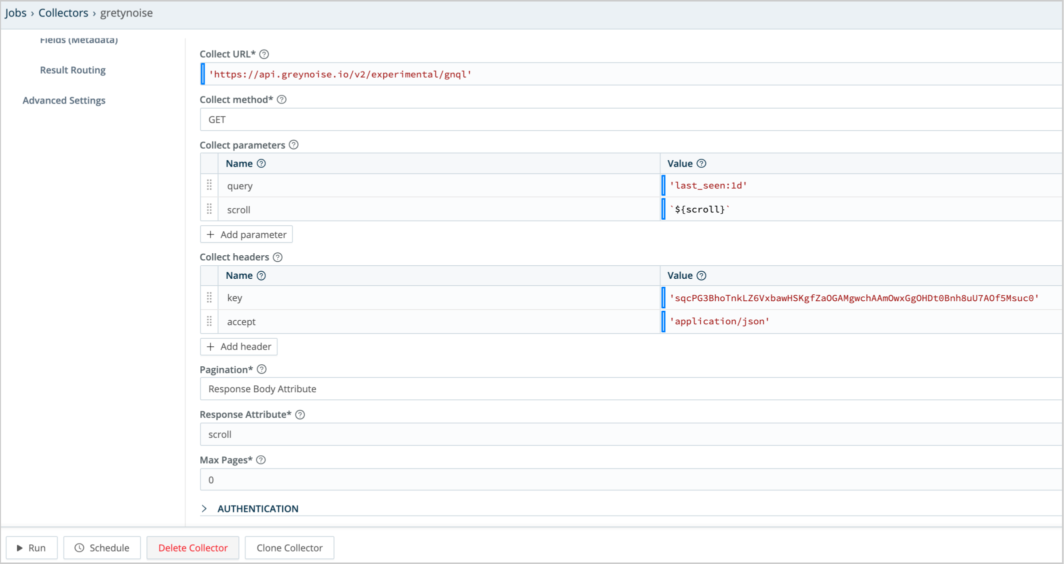
Response Body Attribute: `scroll`

Max Pages: `10` (or `0` to pull all data)

Authentication: None

Event Breaker: None – Data is in NDJSON format, so feel free to assign a JSON Newline Delimited Event Breaker (such as the **Cribl > ndjson** rule).

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 **Max Pages** setting is reached, or no more pages are present.



Collect URL*

`'https://api.greynoise.io/v2/experimental/gnql'`

Collect method*

GET

Collect parameters

Name	Value
query	'last_seen:1d'
scroll	\`${scroll}\`

Collect headers

Name	Value
key	'sqcPG3BhoTnkLZ6VxbawHSKgfZaOGAMgwchAAmOwxGgOHDt0Bnh8uU7AOf5Msuc0'
accept	'application/json'

Pagination*

Response Body Attribute

Response Attribute*

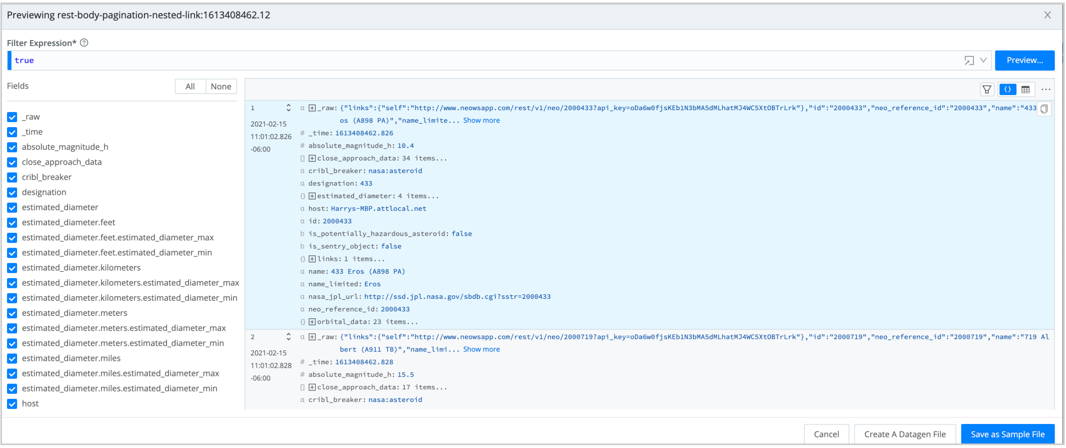
scroll

Max Pages*

0

Collector configuration for HTTP GET, paginated via Response Body Attribute

Collector Output



Previewing rest-body-pagination-nested-link:1613408462.12

Filter Expression*: `true`

Fields

- ☑ _raw
- ☑ _time
- ☑ absolute_magnitude_h
- ☑ close_approach_data
- ☑ cribl_breaker
- ☑ designation
- ☑ estimated_diameter
- ☑ estimated_diameter.feet
- ☑ estimated_diameter.feet.estimated_diameter_max
- ☑ estimated_diameter.feet.estimated_diameter_min
- ☑ estimated_diameter.kilometers
- ☑ estimated_diameter.kilometers.estimated_diameter_max
- ☑ estimated_diameter.kilometers.estimated_diameter_min
- ☑ estimated_diameter.meters
- ☑ estimated_diameter.meters.estimated_diameter_max
- ☑ estimated_diameter.meters.estimated_diameter_min
- ☑ estimated_diameter.miles
- ☑ estimated_diameter.miles.estimated_diameter_max
- ☑ estimated_diameter.miles.estimated_diameter_min
- ☑ host

```
1              α ⊞_raw: {"links":{"self":"http://www.neowsapp.com/rest/v1/neo/2000433?api_key=oDa6w0fjsKEb1N3bMA5dMLhatMJ4WC5XtOBTrLrk"},"id":"2000433","neo_reference_id":"2000433","name":"433
2021-02-15             os (A898 PA)","name_limite... Show more
11:01:02.826   # _time: 1613408462.826
-06:00         # absolute_magnitude_h: 10.4
               [] ⊞close_approach_data: 34 items...
               α cribl_breaker: nasa:asteroid
               α designation: 433
               {} ⊞estimated_diameter: 4 items...
               α host: Harrys-MBP.attlocal.net
               α id: 2000433
               b is_potentially_hazardous_asteroid: false
               b is_sentry_object: false
               {} ⊞links: 1 items...
               α name: 433 Eros (A898 PA)
               α name_limited: Eros
               α nasa_jpl_url: http://ssd.jpl.nasa.gov/sbdb.cgi?sstr=2000433
               α neo_reference_id: 2000433
               {} ⊞orbital_data: 23 items...
2              α ⊞_raw: {"links":{"self":"http://www.neowsapp.com/rest/v1/neo/2000719?api_key=oDa6w0fjsKEb1N3bMA5dMLhatMJ4WC5XtOBTrLrk"},"id":"2000719","neo_reference_id":"2000719","name":"719 Al
2021-02-15             bert (A911 TB)","name_limi... Show more
11:01:02.828   # _time: 1613408462.828
-06:00         # absolute_magnitude_h: 15.5
               [] ⊞close_approach_data: 17 items...
               α cribl_breaker: nasa:asteroid
```

Paginated events

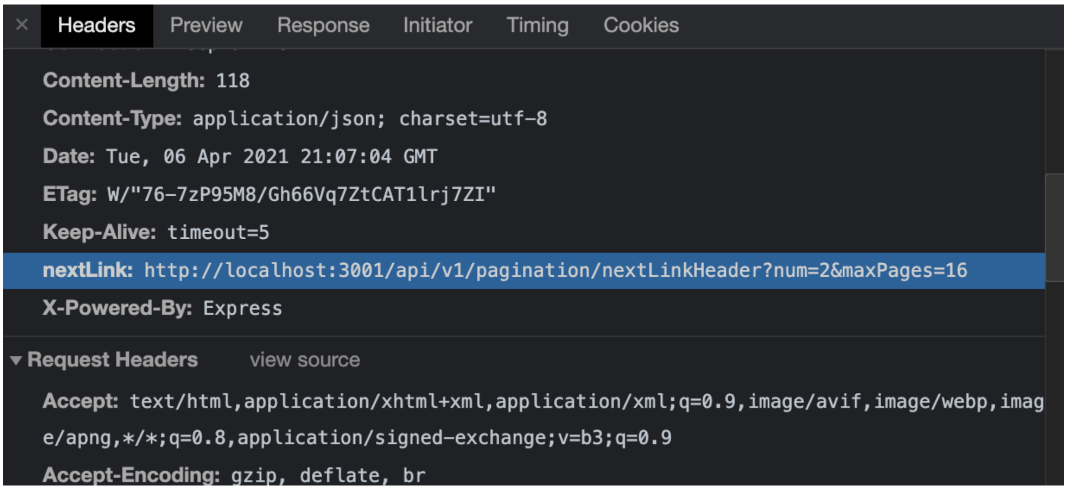
> ⚠ This API allows a certain number of calls/month. Cribl recommends that you not schedule this Collector – run it ad-hoc, for testing only.

For a more detailed use case around this particular API, see Cribl's Enrichment at Scale! blog post.

4. 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:



```
Content-Length: 118
Content-Type: application/json; charset=utf-8
Date: Tue, 06 Apr 2021 21:07:04 GMT
ETag: W/"76-7zP95M8/Gh66Vq7ZtCAT1lrj7ZI"
Keep-Alive: timeout=5
nextLink: http://localhost:3001/api/v1/pagination/nextLinkHeader?num=2&maxPages=16
X-Powered-By: Express
▾ Request Headers
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate, br
```

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 Headers: None

Collect Request Parameters: None

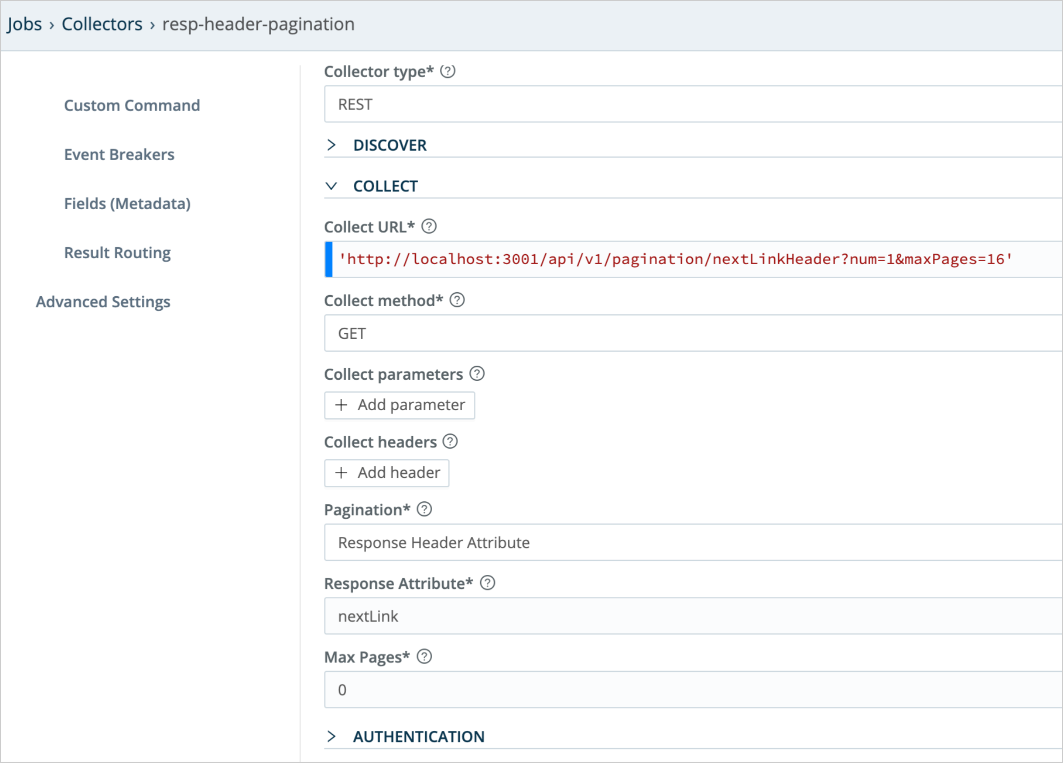
Pagination: Response Header Attribute

Response Attribute: `nextLink`

Authentication: None

Event Breaker: None

> You can modify the `maxPages` URL parameter to control how many pages this call returns.



Collector type*: REST

Collect URL*: `'http://localhost:3001/api/v1/pagination/nextLinkHeader?num=1&maxPages=16'`

Collect method*: GET

Collect parameters

Collect headers

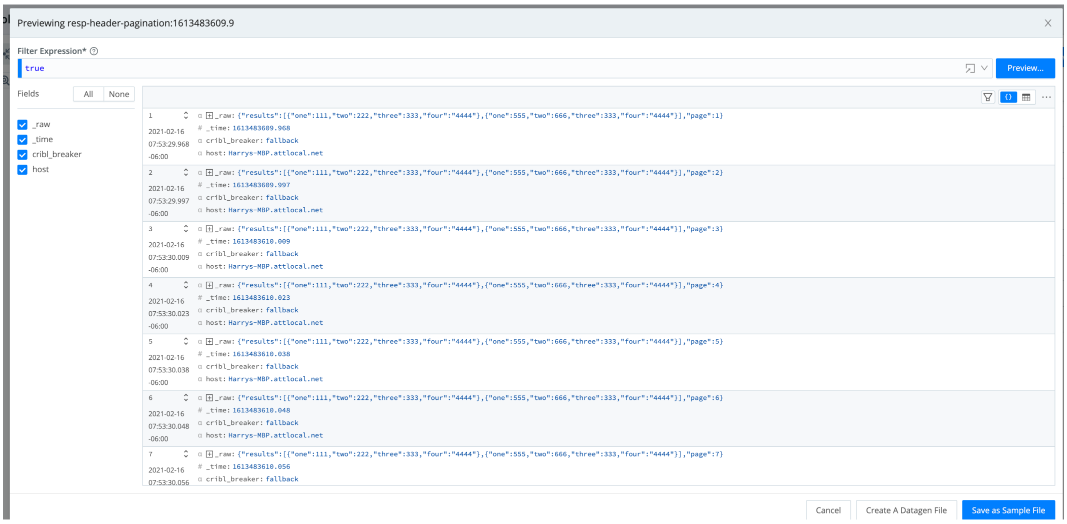
Pagination*: Response Header Attribute

Response Attribute*: nextLink

Max Pages*: 0

Collector configuration for HTTP GET, paginated via Response Header URL

Collector Output



Filter Expression*: `true`

```
1           _raw: {"results":[{"one":111,"two":222,"three":333,"four":"4444"},{"one":555,"two":666,"three":333,"four":"4444"}],"page":1}
2021-02-16  _time: 1613483609.968
07:53:29.968  cribl_breaker: fallback
-06:00      host: Harrys-MBP.attlocal.net
2           _raw: {"results":[{"one":111,"two":222,"three":333,"four":"4444"},{"one":555,"two":666,"three":333,"four":"4444"}],"page":2}
2021-02-16  _time: 1613483609.997
07:53:29.997  cribl_breaker: fallback
-06:00      host: Harrys-MBP.attlocal.net
3           _raw: {"results":[{"one":111,"two":222,"three":333,"four":"4444"},{"one":555,"two":666,"three":333,"four":"4444"}],"page":3}
2021-02-16  _time: 1613483610.009
07:53:30.009  cribl_breaker: fallback
-06:00      host: Harrys-MBP.attlocal.net
4           _raw: {"results":[{"one":111,"two":222,"three":333,"four":"4444"},{"one":555,"two":666,"three":333,"four":"4444"}],"page":4}
2021-02-16  _time: 1613483610.023
07:53:30.023  cribl_breaker: fallback
-06:00      host: Harrys-MBP.attlocal.net
5           _raw: {"results":[{"one":111,"two":222,"three":333,"four":"4444"},{"one":555,"two":666,"three":333,"four":"4444"}],"page":5}
2021-02-16  _time: 1613483610.038
07:53:30.038  cribl_breaker: fallback
-06:00      host: Harrys-MBP.attlocal.net
6           _raw: {"results":[{"one":111,"two":222,"three":333,"four":"4444"},{"one":555,"two":666,"three":333,"four":"4444"}],"page":6}
2021-02-16  _time: 1613483610.048
07:53:30.048  cribl_breaker: fallback
-06:00      host: Harrys-MBP.attlocal.net
7           _raw: {"results":[{"one":111,"two":222,"three":333,"four":"4444"},{"one":555,"two":666,"three":333,"four":"4444"}],"page":7}
2021-02-16  _time: 1613483610.056
07:53:30.056  cribl_breaker: fallback
```

Paginated events

5. 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 (e.g., space, `$`, `/`, or `=`).

Discover Type: `HTTP Request`

Discover URL: 'http://localhost:9000/api/v1/system/logs'

Discover Method: GET

Discover Headers: None

Discover Parameters: None

Discover Data Field: `items`

Collect URL: `'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)`

Collect Headers: None

Collect Request Parameters: None

Pagination: Response Header Attribute

Authentication: Login

Login URL: `http://localhost:9000/api/v1/auth/login`

Username: `admin` (or other user)

Password: `admin` (or other user's corresponding password)

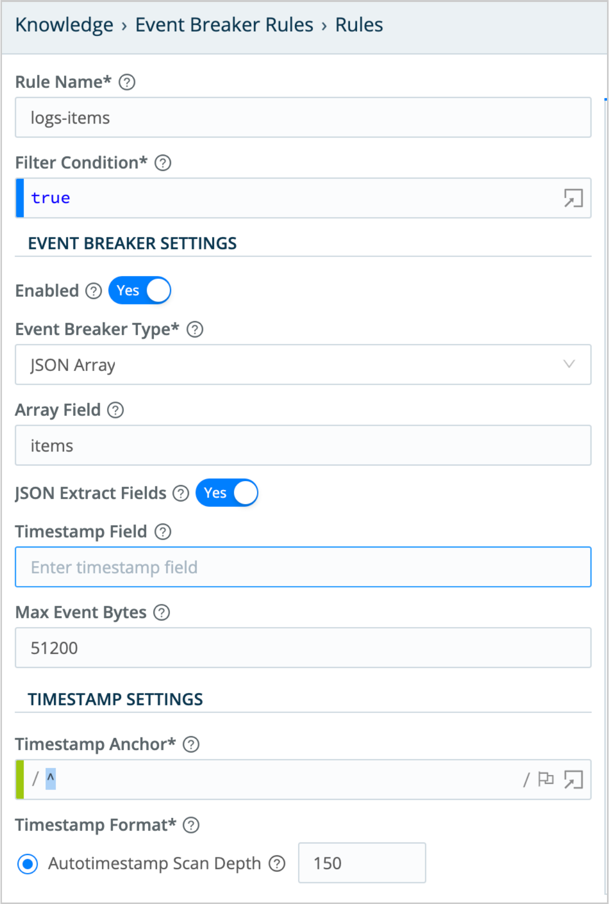
[Authentication] POST Body: `` `{ "username": "${username}", "password": "${password}" }` ``

Token Attribute: `token`

Authorize Expression: `` `Bearer ${token}` ``

Event Breaker: JSON Array

- **Array Field:** `items.events`



Knowledge › Event Breaker Rules › Rules

Rule Name* ⓘ
logs-items

Filter Condition* ⓘ
true

EVENT BREAKER SETTINGS

Enabled ⓘ Yes

Event Breaker Type* ⓘ
JSON Array

Array Field ⓘ
items

JSON Extract Fields ⓘ Yes

Timestamp Field ⓘ
Enter timestamp field

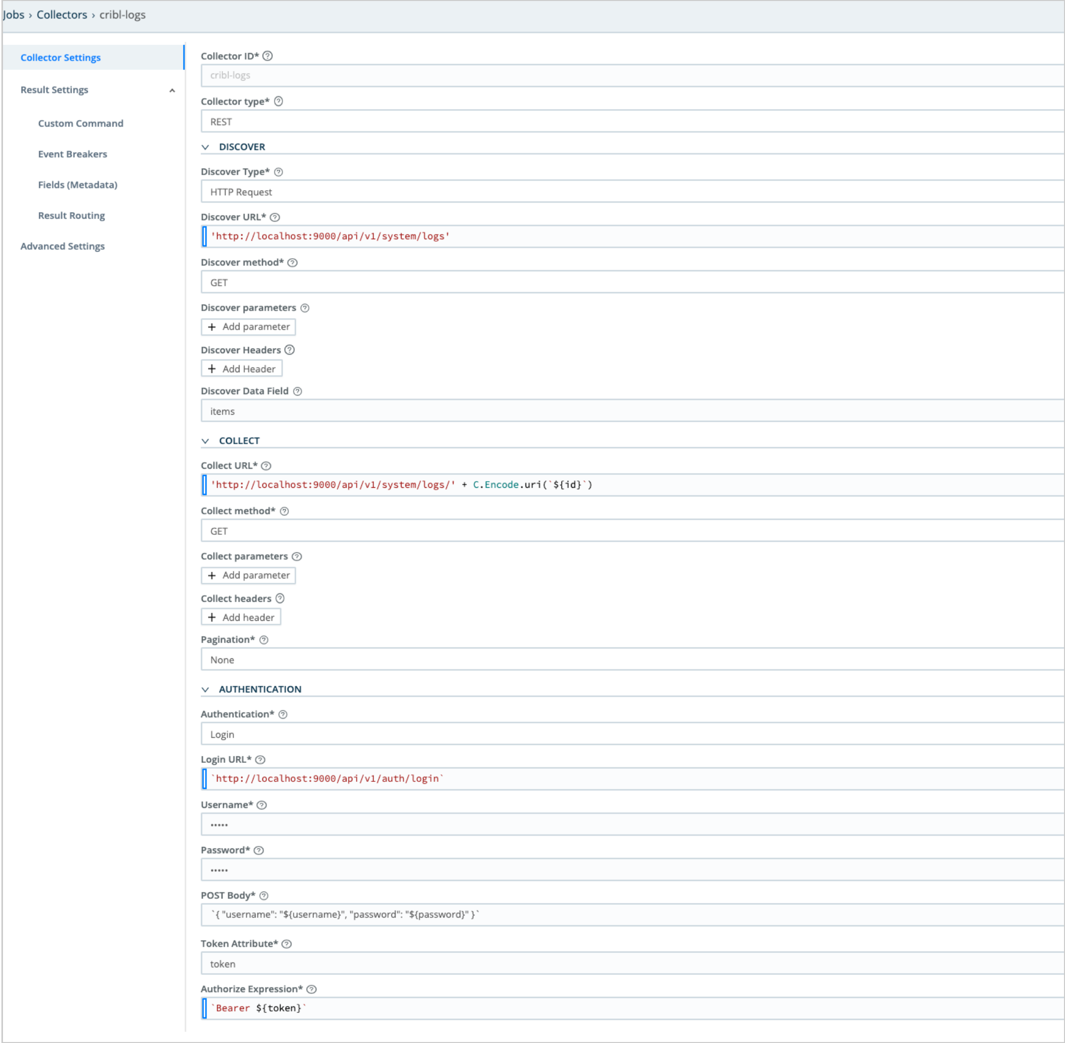
Max Event Bytes ⓘ
51200

TIMESTAMP SETTINGS

Timestamp Anchor* ⓘ
/ ^ /

Timestamp Format* ⓘ
● Autotimestamp Scan Depth ⓘ 150

Event Breaker configuration



Collector ID*: cribl-logs

Collector type*: REST

DISCOVER

Discover Type*: HTTP Request

Discover URL*: `'http://localhost:9000/api/v1/system/logs'`

Discover method*: GET

Discover parameters

Discover Headers

Discover Data Field: items

COLLECT

Collect URL*: ``'http://localhost:9000/api/v1/system/logs/' + C.Encode.uri(`${id}`)``

Collect method*: GET

Collect parameters

Collect headers

Pagination*: None

AUTHENTICATION

Authentication*: Login

Login URL*: `` `http://localhost:9000/api/v1/auth/login` ``

Username*: •••••

Password*: •••••

POST Body*: `` `{ "username": "${username}", "password": "${password}" }` ``

Token Attribute*: token

Authorize Expression*: `` `Bearer ${token}` ``

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, Discover and Collect). The variable name used in the **Authorize Expression** should be the same name specified in the **Token Attribute** call.

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"
    }
  ]
}
```

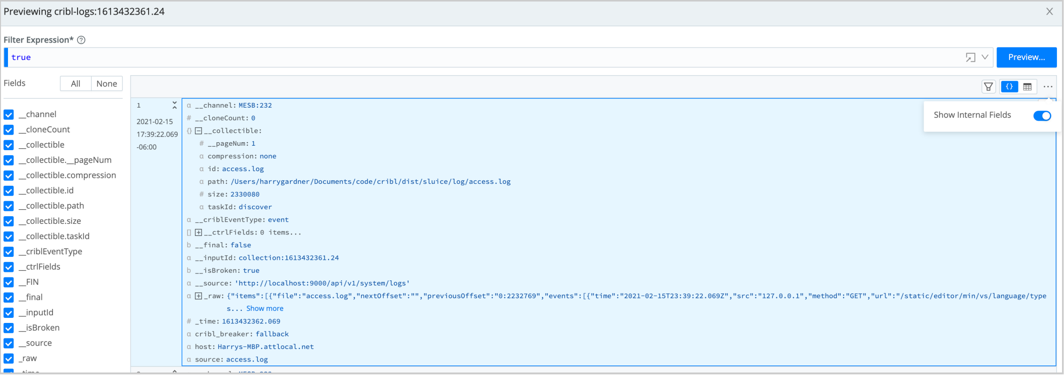
The **Discover Data Field** is used to define the array in Discover results that contains the list of items to discover. Here, each item is an object, with an attribute ID that is referenced in the Collect calls. So the Discover call generates a list of items for which Collect tasks will be created.

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:



Previewing cribl-logs:1613432361.24

Filter Expression*: true

☑ __channel
☑ __cloneCount
☑ __collectible
☑ __collectible.__pageNum
☑ __collectible.compression
☑ __collectible.id
☑ __collectible.path
☑ __collectible.size
☑ __collectible.taskId
☑ __criblEventType
☑ __ctrlFields
☑ __FIN
☑ __final
☑ __inputId
☑ __isBroken
☑ __source
☑ _raw

```
1   2021-02-15 17:39:22.069 -06:00
__channel: MESB:232
__cloneCount: 0
__collectible:
  __pageNum: 1
  compression: none
  id: access.log
  path: /Users/harrygardner/Documents/code/cribl/dist/sluice/log/access.log
  size: 2330080
  taskId: discover
__criblEventType: event
__ctrlFields: 0 items...
__final: false
__inputId: collection:1613432361.24
__isBroken: true
__source: 'http://localhost:9000/api/v1/system/logs'
_raw: {"items":[{"file":"access.log","nextOffset":"","previousOffset":"0:2232769","events":[{"time":"2021-02-15T23:39:22.069Z","src":"127.0.0.1","method":"GET","url":"/static/editor/min/vs/language/type
    s... Show more
_time: 1613432362.069
cribl_breaker: fallback
host: Harrys-MBP.attlocal.net
source: access.log
```

__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:

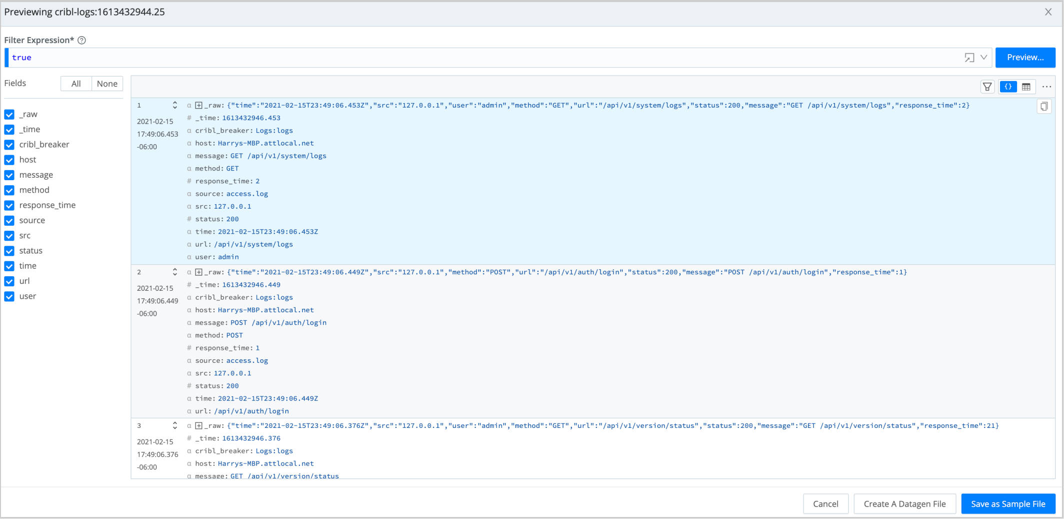
```
'http://localhost:9000/api/v1/system/logs/' +
C.Encode.uri(`${id}`)
```

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
...
```

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 LogStream. The output looks like this in Preview:



Previewing cribl-logs:1613432944.25

Filter Expression*: `true`

Fields

☑ _raw
☑ _time
☑ cribl_breaker
☑ host
☑ message
☑ method
☑ response_time
☑ source
☑ src
☑ status
☑ time
☑ url
☑ user

```
1            _raw: {"time":"2021-02-15T23:49:06.453Z","src":"127.0.0.1","user":"admin","method":"GET","url":"/api/v1/system/logs","status":200,"message":"GET /api/v1/system/logs","response_time":2}
2021-02-15   _time: 1613432946.453
17:49:06.453 cribl_breaker: Logs:logs
-06:00       host: Harrys-MBP.attlocal.net
             message: GET /api/v1/system/logs
             method: GET
             response_time: 2
             source: access.log
             src: 127.0.0.1
             status: 200
             time: 2021-02-15T23:49:06.453Z
             url: /api/v1/system/logs
             user: admin
2            _raw: {"time":"2021-02-15T23:49:06.449Z","src":"127.0.0.1","method":"POST","url":"/api/v1/auth/login","status":200,"message":"POST /api/v1/auth/login","response_time":1}
2021-02-15   _time: 1613432946.449
17:49:06.449 cribl_breaker: Logs:logs
-06:00       host: Harrys-MBP.attlocal.net
             message: POST /api/v1/auth/login
             method: POST
             response_time: 1
             source: access.log
             src: 127.0.0.1
             status: 200
             time: 2021-02-15T23:49:06.449Z
             url: /api/v1/auth/login
3            _raw: {"time":"2021-02-15T23:49:06.376Z","src":"127.0.0.1","user":"admin","method":"GET","url":"/api/v1/version/status","status":200,"message":"GET /api/v1/version/status","response_time":21}
2021-02-15   _time: 1613432946.376
17:49:06.376 cribl_breaker: Logs:logs
-06:00       host: Harrys-MBP.attlocal.net
             message: GET /api/v1/version/status
```

Collected data

6. 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 Request 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
```

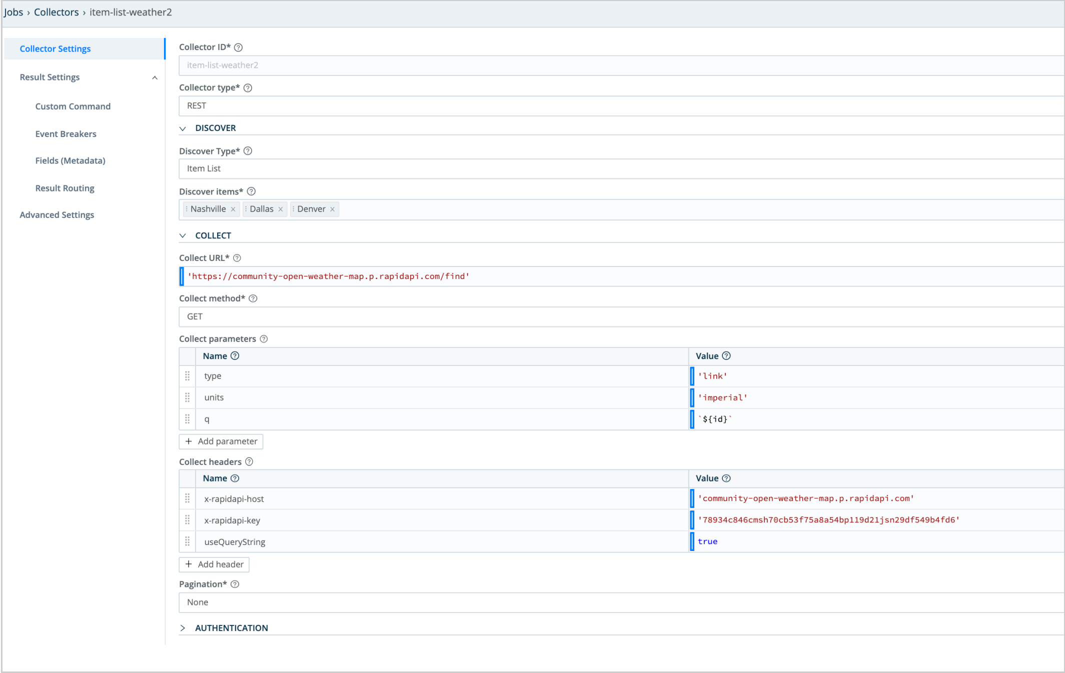
Pagination: None

Authentication: None

Event Breaker: JSON Newline Delimited – Use a rule like **Cribl > ndjson** to parse each event and extract fields.

Fields (Metadata):

```
job: weather-${__collectible.id}

city: ${__collectible.id}
```



Collector ID*: item-list-weather2

Collector type*: REST

Discover Type*: Item List

Discover items*: Nashville, Dallas, Denver

Collect URL*: `'https://community-open-weather-map.p.rapidapi.com/find'`

Collect method*: GET

Collect parameters

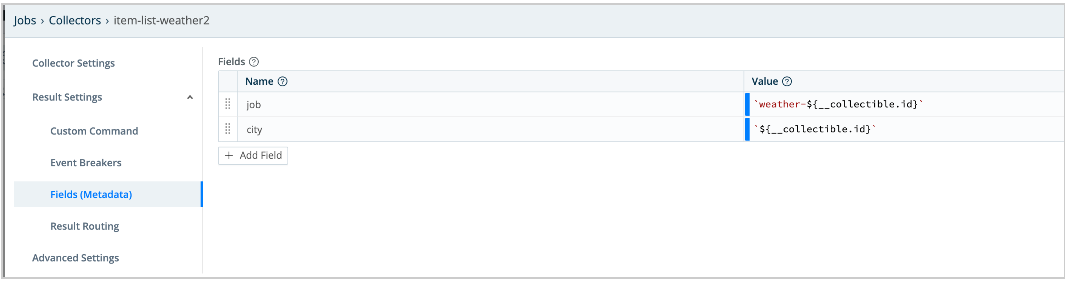
Name	Value
type	'link'
units	'imperial'
q	\`${id}\`

Collect headers

Name	Value
x-rapidapi-host	'community-open-weather-map.p.rapidapi.com'
x-rapidapi-key	'78934c846cmsh70cb53f75a8a54bp119d21jsn29df549b4fd6'
useQueryString	true

Pagination*: None

Collector configuration for Discovery via Item List



Fields

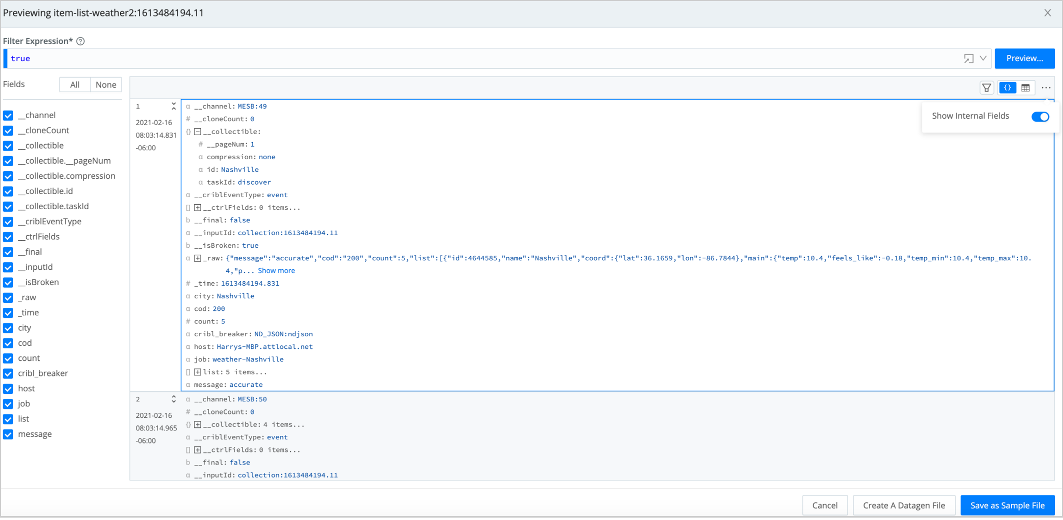
Name	Value
job	\`weather-${__collectible.id}\`
city	\`${__collectible.id}\`

Fields (Metadata) configuration

Collector Output

One interesting thing about this example is the addition of **Fields (Metadata)** 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:



```
true
```

☑ __channel
☑ __cloneCount
☑ __collectible
☑ __collectible.__pageNum
☑ __collectible.compression
☑ __collectible.id
☑ __collectible.taskId
☑ __criblEventType
☑ __ctrlFields
☑ __final
☑ __inputId
☑ __isBroken
☑ _raw
☑ _time
☑ city
☑ cod
☑ count
☑ cribl_breaker
☑ host
☑ job
☑ list
☑ message

```
1    2021-02-16 08:03:14.831 -06:00
α __channel: MESB:49
# __cloneCount: 0
{} ⊟ __collectible:
    # __pageNum: 1
    α compression: none
    α id: Nashville
    α taskId: discover
α __criblEventType: event
[] ⊞ __ctrlFields: 0 items...
b __final: false
α __inputId: collection:1613484194.11
b __isBroken: true
α ⊞ _raw: {"message":"accurate","cod":"200","count":5,"list":[{"id":4644585,"name":"Nashville","coord":{"lat":36.1659,"lon":-86.7844},"main":{"temp":10.4,"feels_like":-0.18,"temp_min":10.4,"temp_max":10.
        4,"p... Show more
# _time: 1613484194.831
α city: Nashville
α cod: 200
# count: 5
α cribl_breaker: ND_JSON:ndjson
α host: Harrys-MBP.attlocal.net
α job: weather-Nashville
[] ⊞ list: 5 items...
α message: accurate

2    2021-02-16 08:03:14.965 -06:00
α __channel: MESB:50
# __cloneCount: 0
{} ⊞ __collectible: 4 items...
α __criblEventType: event
[] ⊞ __ctrlFields: 0 items...
b __final: false
α __inputId: collection:1613484194.11
```

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.

7. 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 Request Parameters:

```
lat: `${lat}`
```

```
lon: `${lon}`
```

```
appid: '438d61a1db9e713240b30140e9ddfea2'
```

Collect Headers: None

Pagination: None

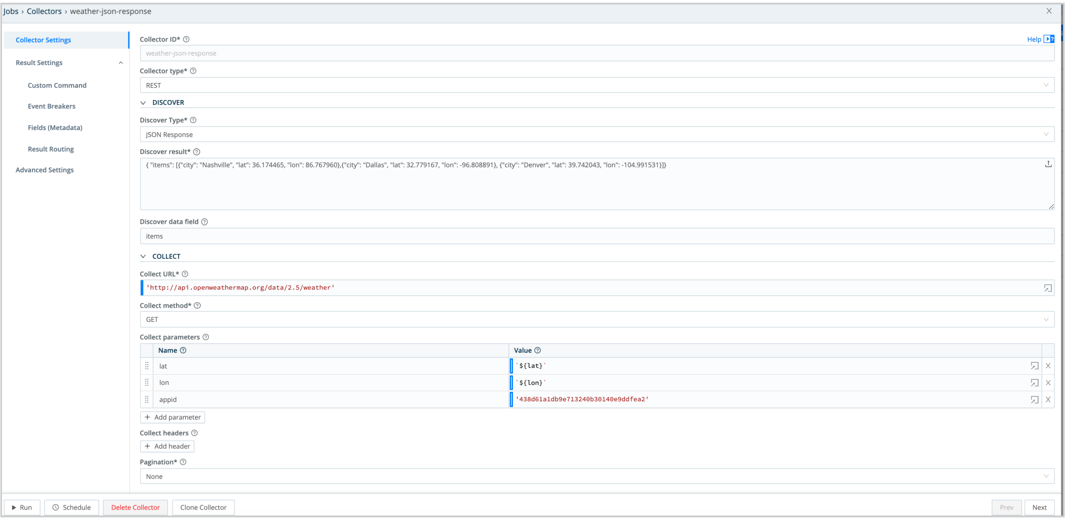
Authentication: None

Event Breaker: JSON Newline Delimited – Use a rule like **Cribl > ndjson** to parse each event and extract fields.

Fields (Metadata):

```
job: `weather-${__collectible.city}`
```

```
city: `${__collectible.city}`
```



Collector ID*

weather-json-response

Collector type*

REST

DISCOVER

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

Collect URL*

'http://api.openweathermap.org/data/2.5/weather'

Collect method*

GET

Collect parameters

Name	Value
lat	\`${lat}\`
lon	\`${lon}\`
appid	'438d61a1db9e713240b30140e9ddfea2'

+ Add parameter

Collect headers

+ Add header

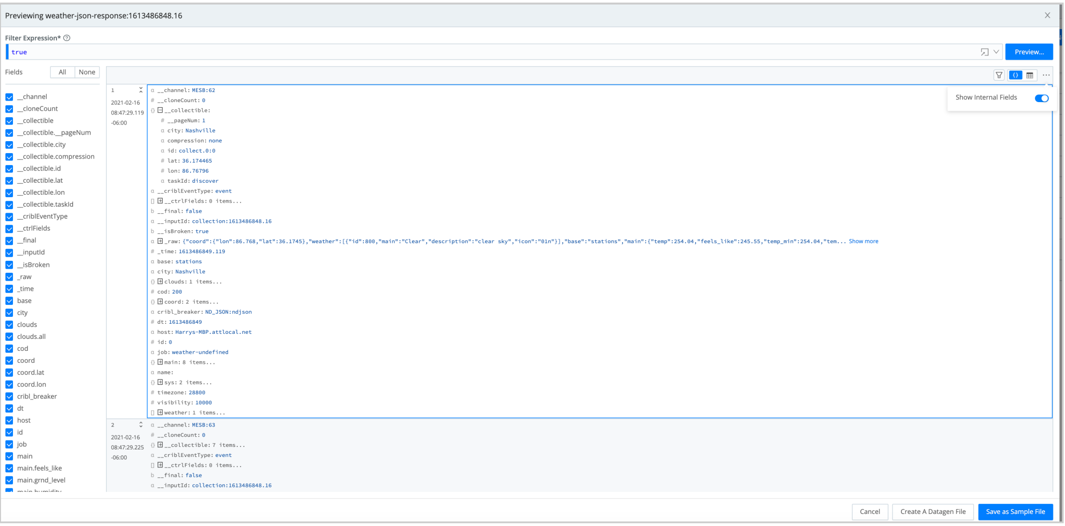
Pagination*

None

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



Previewing weather-json-response:1613486848.16

Filter Expression*

true

☑ __channel
☑ __cloneCount
☑ __collectible
☑ __collectible.__pageNum
☑ __collectible.city
☑ __collectible.compression
☑ __collectible.id
☑ __collectible.lat
☑ __collectible.lon
☑ __collectible.taskId
☑ __criblEventType
☑ __ctrlFields
☑ __final
☑ __inputId
☑ __isBroken
☑ _raw
☑ _time
☑ base
☑ city
☑ clouds
☑ clouds.all
☑ cod
☑ coord
☑ coord.lat
☑ coord.lon
☑ cribl_breaker
☑ dt
☑ host
☑ id
☑ job
☑ main
☑ main.feels_like
☑ main.grnd_level

```
1
2021-02-16
08:47:29.119
-06:00

__channel: MESB:62
__cloneCount: 0
__collectible:
    __pageNum: 1
    city: Nashville
    compression: none
    id: collect.0:0
    lat: 36.174465
    lon: 86.76796
    taskId: discover
__criblEventType: event
__ctrlFields: 0 items...
__final: false
__inputId: collection:1613486848.16
__isBroken: true
_raw: {"coord":{"lon":86.768,"lat":36.1745},"weather":[{"id":800,"main":"Clear","description":"clear sky","icon":"01n"}],"base":"stations","main":{"temp":254.04,"feels_like":245.55,"temp_min":254.04,"tem... Show more
_time: 1613486849.119
base: stations
city: Nashville
clouds: 1 items...
cod: 200
coord: 2 items...
cribl_breaker: ND_JSON:ndjson
dt: 1613486849
host: Harrys-MBP.attlocal.net
id: 0
job: weather-undefined
main: 8 items...
name:
sys: 2 items...
timezone: 28800
visibility: 10000
weather: 1 items...

2
2021-02-16
08:47:29.225
-06:00

__channel: MESB:63
__cloneCount: 0
__collectible: 7 items...
__criblEventType: event
__ctrlFields: 0 items...
__final: false
__inputId: collection:1613486848.16
```

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.

System Proxy Configuration

You can direct all outbound HTTP/S requests to go through proxy servers. You do so by setting a few environment variables before starting LogStream, as follows:

Configure the `HTTP_PROXY` and `HTTPS_PROXY` environment variables, either with your proxy's IP address, or with a DNS name that resolves to that IP address. Optionally, follow either convention with a colon and the port number to which you want to send queries.

`HTTP_PROXY` examples:

```
$ export HTTP_PROXY=http://10.15.20.25:1234
$ export HTTP_PROXY=http://proxy.example.com:1234
```

`HTTPS_PROXY` examples:

```
$ export HTTPS_PROXY=http://10.15.20.25:5678
$ export HTTPS_PROXY=http://proxy.example.com:5678
```

In the above examples, note that when you set an `HTTPS_PROXY` environment variable, the referenced URL should generally be in `http` format.

> **i Restarts and Case Conflicts**
>
> Initial configuration of, and changes to, these variables require restarting LogStream on the affected Nodes, if the application is already running when you apply the changes.
>
> 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. E.g., if you've set both `HTTPS_PROXY` and `https_proxy`, the IP address specified in `https_proxy` will take effect.

HTTP and/or HTTPS?

Several LogStream endpoints rely on the HTTPS protocol – the Cribl telemetry endpoint, which must be accessed with some license types, as well as the CDN used to propagate application updates and certain documentation features (API Reference and docs PDFs).

You might configure certain other LogStream 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.

Proxy Confguration with systemd

If you are proxying outbound traffic and starting LogStream using systemd, list your proxy environment variables in the systemd unit file's `[Service]` section by adding statements of this form:

Installed systemd File

```
[Service]
 ...
Environment=https_proxy=<yourproxy>
Environment=https_proxy=http://proxy.example.com:1234
Environment=https_proxy=http://10.10.1.1:8080
```

This will prevent LogStream from throwing "failed to send anonymized telemetry metadata" errors.

Authenticating on Proxies

You can use HTTP Basic authentication on HTTP or HTTPS proxies. Specify the user name 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
```

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, and should instead be sent as direct requests. Use the following format:

```
$ export NO_PROXY="<list of hosts/domains>"
```

Usage notes:

- Cribl recommends including the Leader Node's host name in the `NO_PROXY` list.
- Within the list, separate the host/domain names with commas or spaces.
- Optionally, each host/domain entry can be followed by a port. If specified, the port must match. If not specified, the protocol's default port is assumed.
- If specified, subdomain names must match. E.g., `NO_PROXY=foo.example.com` will send requests directly to https://foo.example.com, but https://bar.example.com requests will go through the proxy.
- You can use leading wildcards like `NO_PROXY="*.us, .org"` .
- `NO_PROXY="*"` disables all proxies.
- `NO_PROXY` with an empty list disables no proxies.

Where Proxies Apply

Proxy configuration is relevant to the following LogStream components that make outbound HTTP/S requests:

Destinations

- S3 Compatible Stores
- AWS Kinesis Streams
- AWS CloudWatch Logs
- AWS SQS
- Azure Blob Storage
- Azure Event Hubs
- Azure Monitor Logs
- Elasticsearch

- Honeycomb
- Splunk HEC

Sources

- AWS Kinesis Streams
- AWS SQS
- AWS S3
- Azure Event Hubs

Collectors

- S3 Collector

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 LogStream instance, and can validate that outgoing requests (to Destinations) are working properly.

Proxying Multiple LogStream Instances in One Browser

LogStream stores authentication tokens based on each http header's URI scheme, host, and port information. Within a given browser, LogStream enforces a same-origin policy to isolate instances.

This means that if you want to run multiple proxied LogStream instances in one browser session, you must assign them different URI schemes, hosts, and/or ports. Otherwise, logging into an extra LogStream 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.

VIDEOS

Videos

Prefer to watch rather than read? These (mostly) brief demonstrations will help you identify and apply LogStream features that meet your needs.

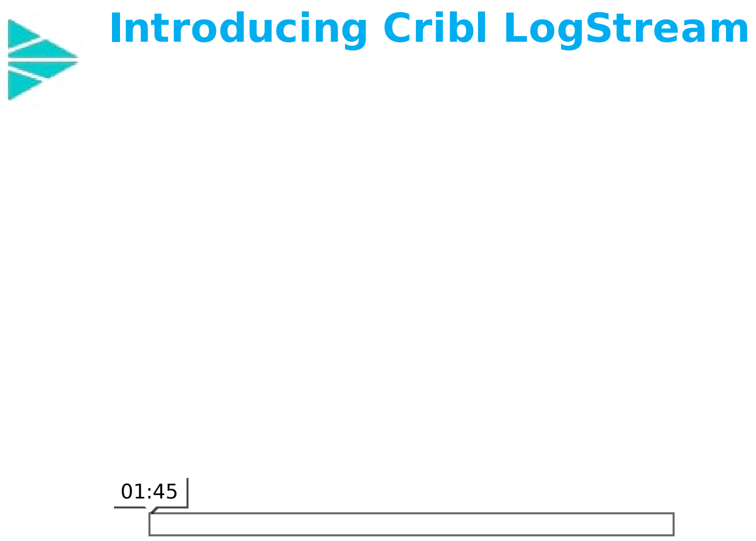
All Videos

Browse all Cribl videos here.

Concept Videos

Browse all concept videos here.

Introducing Cribl LogStream



Conceptual overview of LogStream's capabilities

Concept: LogStream Cloud

Concept: LogStream Cloud

from **Cribl**

Conceptual walkthrough of Cribl's LogStream Cloud offering
Sign up for the beta at: https://cribl.io/logstream-cloud-beta/

Concept: AppScope

Concept-AppScope

from **Cribl**

Brief walkthrough of Cribl's open-source AppScope project

Concept: Receive

Cribl Concept: Receive

01:35

A quick walkthrough of LogStream's data ingestion capabilities

Concept: Reduce

Cribl Concept: Reduce

01:35

A quick walkthrough of LogStream's data reduction capabilities

Concept: Transform

Cribl Concept: Transform

01:18

A quick walkthrough of LogStream's data transformation capabilities

Concept: Collect

Cribl Concept: Collect

02:03

A quick overview of LogStream's data collection capabilities

Concept: Pipelines

Cribl Concept: Pipelines

01:51

A brief conceptual walkthrough of how Pipelines work in LogStream

Concept: Routes

Cribl Concept: Route

A brief conceptual walkthrough of how LogStream routes data

Concept: Role-Based Access Control

Cribl Concept-Role Based Access Control

A quick walkthrough of LogStream's RBAC and audit logging capabilities

Concept: Redis Lookups

Cribl Concept-Redis Lookups

02:24

A brief walkthrough of the LogStream's Redis Function, and of how to use it to enrich your data

How-to Videos

Browse all how-to videos here.

How-to: Common Source Configuration

Cribl How-To: Common Source Configuration

01:50

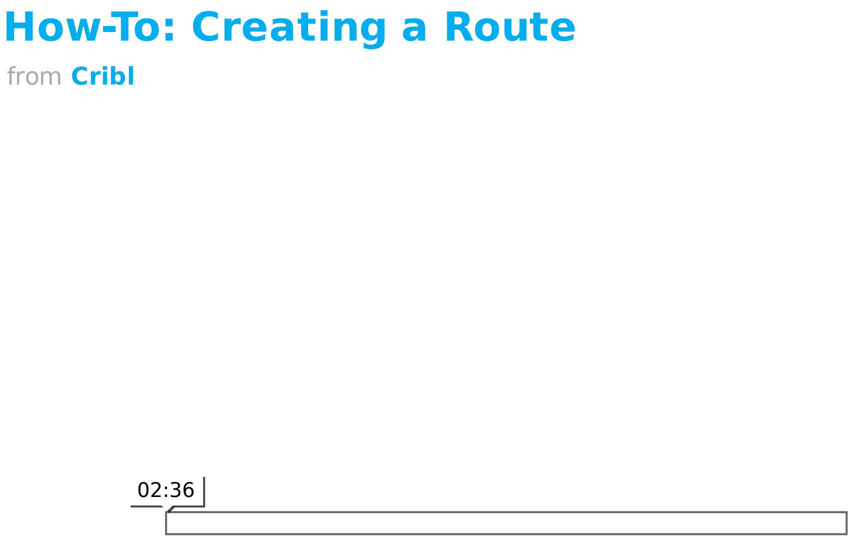
A quick walkthrough of LogStream Sources' common attributes

How-to: Creating a Pipeline

Cribl How-To: Creating a Pipeline

A quick walkthrough of the steps to create a LogStream Pipeline

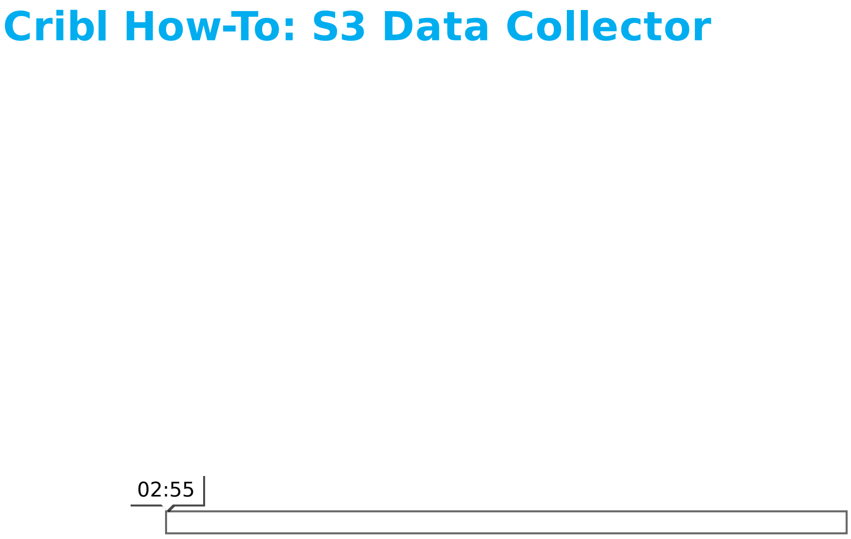
How-to: Creating a Route


How-To: Creating a Route

from Cribl

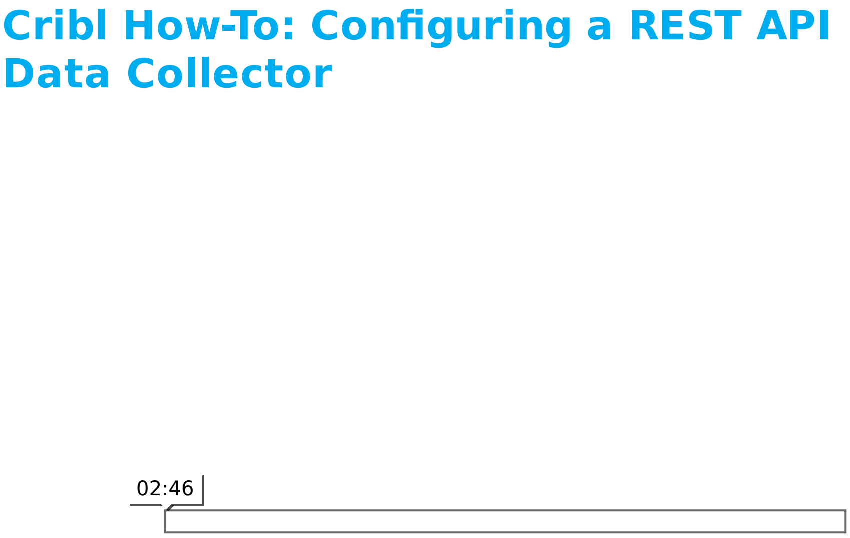
A walkthrough of creating a Route in LogStream

How-to: Configuring an S3 Data Collector


Cribl How-To: S3 Data Collector

Learn how to create a data collector to pull data from AWS S3 buckets

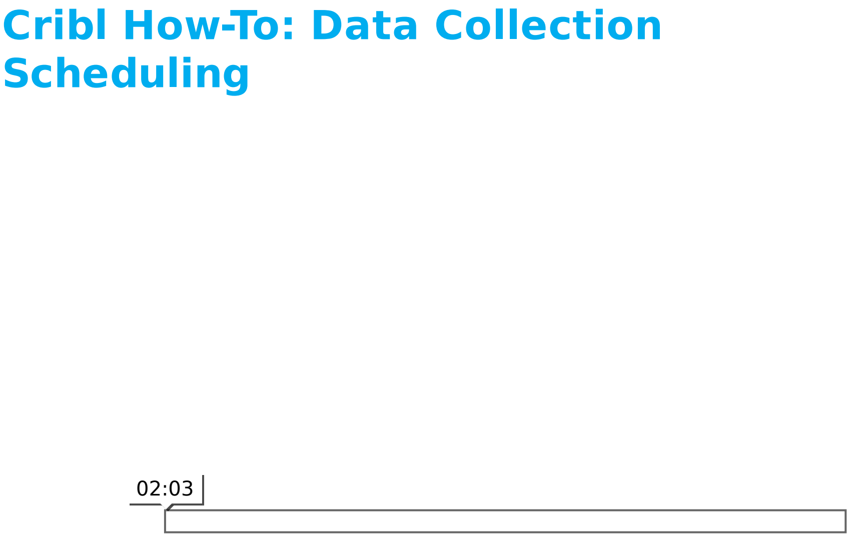
How-to: Configuring a REST API Data Collector



Cribl How-To: Configuring a REST API Data Collector

A walkthrough on creating a data collector to pull data from a REST API

How-to: Data Collection Scheduling



Cribl How-To: Data Collection Scheduling

A brief walkthrough on scheduling data collection jobs in LogStream

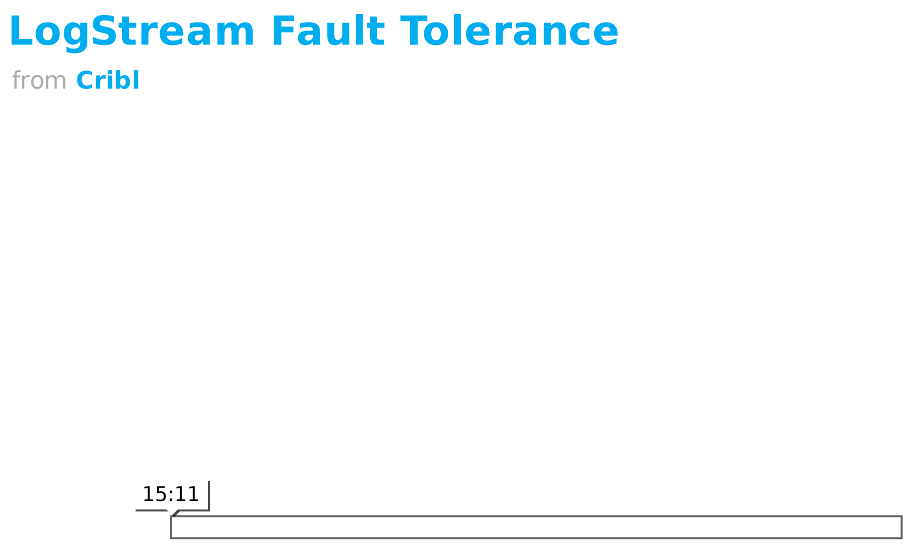
How-to: Securing Logstream

Securing LogStream

from Cribl

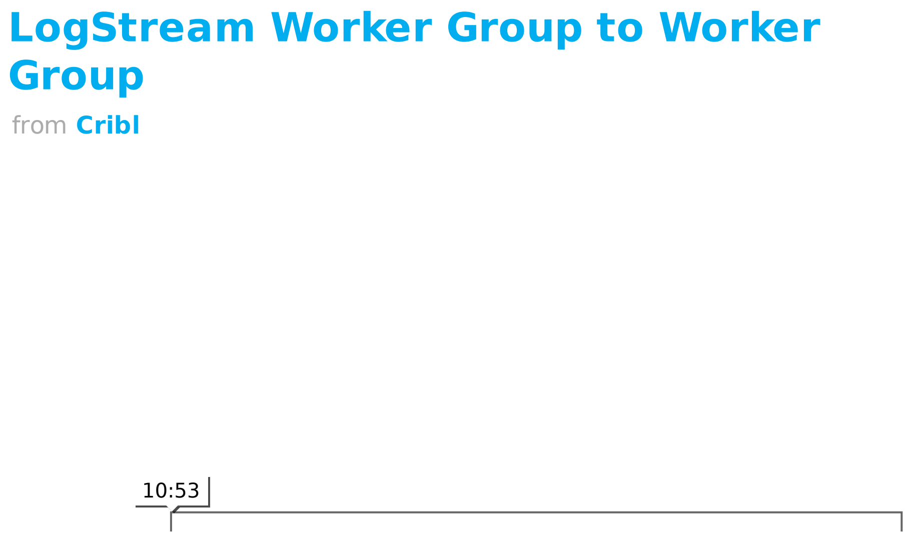
Use TLS to secure a LogStream distributed deployment

How-to: Logstream Fault Tolerance



Recover a failed Leader Node from a remote Git repository

How-to: Worker Group–to–Worker Group Communication



Streaming data between two Worker Groups – more details in this blog post

TROUBLESHOOTING

Diagnosing Issues

To help diagnose LogStream 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 LogStream was configured and operating at that time.

What's in the Diagnostic Bundle

The following directories (and their contents) off of `$CRIBL_HOME` are included:

- `/default/*`
- `/local/*`
- `/log/*`
- `/groups/*`
- `/state/jobs/*` – includes only the latest 10 task from the latest 10 jobs.

Creating and Exporting a Diagnostic Bundle

Users can create and **securely** share bundles with Cribl Support either from the UI or from the CLI. In either case, you'll need outbound internet access to https://diag-upload.cribl.io and a valid Support Case number.

Using the UI

To create a bundle, go to global ⚙ **Settings** (lower left) > **Diagnostics > Diagnostic Bundle** and click **Create Diagnostic Bundle**.

- To download the bundle locally to your machine, click **Export**.

- To share the bundle with Cribl Support, toggle **Send to Cribl Support** to **Yes**, enter your case number, and then click **Export**.

You can create a bundle from individual workers if you have the Worker UI access setting enabled. Go to **Workers >** <worker-name> > **Settings** (top right) > **Diagnostics > Diagnostic Bundle**, and click **Create Diagnostic Bundle**.

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.

Using the CLI

To create a bundle using the CLI, use the `diag` command.

diag command CLI

```
# $CRIBL_HOME/bin/cribl diag
Usage: [sub-command] [options] [args]

Commands:
get    - List existing Cribl LogStream diagnostic bundles
create - Creates diagnostic bundle for Cribl LogStream
send   - Send LogStream diagnostic bundle to Cribl Support, args:
   -c <caseNumber> - Cribl Case Number
  [-p <path>]      - Diagnostic bundle path (if empty, then new bundle wil


## Creating a diagnostic bundle
# $CRIBL_HOME/bin/cribl diag create
Created Cribl LogStream diagnostic bundle at /opt/cribl/diag/cribl-logstre

## Creating and sending a diagnostic bundle
# $CRIBL_HOME/bin/cribl diag send -c 420420
Sent LogStream diagnostic bundle to Cribl Support

## Sending a previously created diagnostic bundle
# $CRIBL_HOME/bin/cribl diag send -p /opt/cribl/diag/cribl-logstream-<host
Sent LogStream diagnostic bundle to Cribl Support
```

Working with Cribl Support

If you run into issues with LogStream, please first check our Known Issues page for recommended resolutions or workarounds. For questions not addressed there, this page outlines how to engage with the Cribl Support staff to resolve problems as quickly as possible.

Creating a Support Case

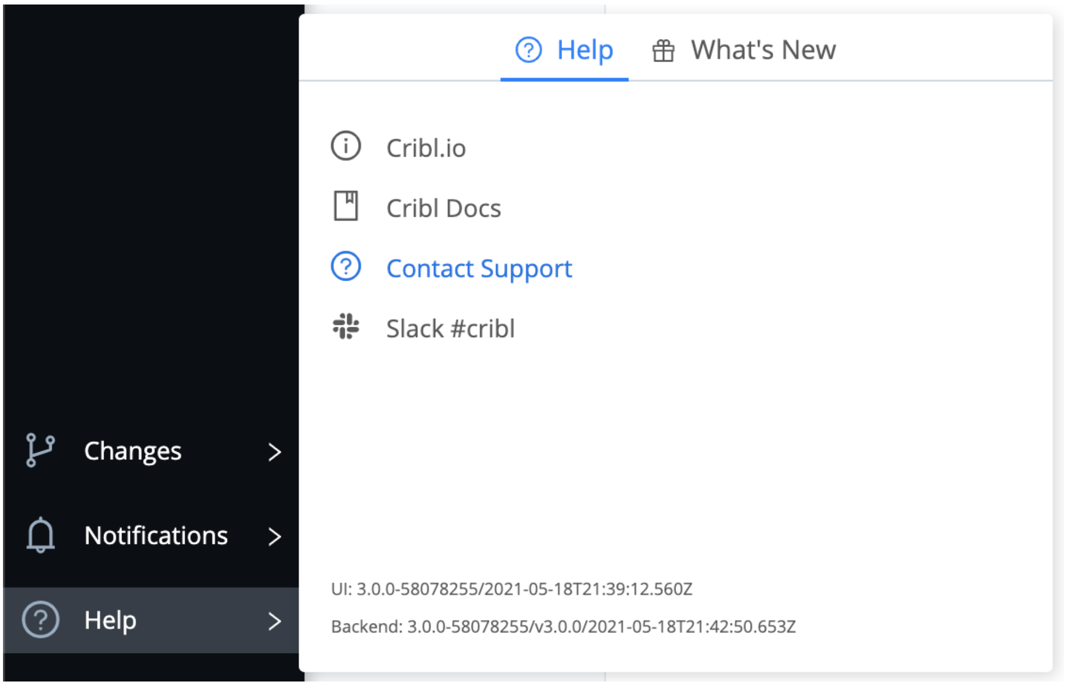
You have several options for opening a support case.

Contact via Email

The most direct way to engage with Support is to email support@cribl.io, with the information outlined below. This will automatically open a case for us to track, and will send you an auto-confirmation email that includes your case number. A Support engineer will then contact you to begin troubleshooting.

Email via Help Button

Within LogStream's UI, you can click the left nav's **(?) Help** link and, from the resulting fly-out, select **Contact Support**. This will prompt creation of a new email in your email client.



UI: 3.0.0-58078255/2021-05-18T21:39:12.560Z

Backend: 3.0.0-58078255/v3.0.0/2021-05-18T21:42:50.653Z

Email from the UI

Contact via Intercom Button

Within LogStream's UI, you can click the Intercom button at the bottom right to send Cribl Support questions and (if necessary) screenshots and other files.

We recommend Intercom only for quick questions/answers, because it doesn't provide robust tracking of communications or files.

Slack Community Channels

Our Support staff monitors Cribl's Community Slack for any issues customers are experiencing: https://cribl-community.slack.com/. If you are not already registered for our Community Slack, please register at https://cribl.io/community/ to get started.

Slack might not get you the same timely response as an email, but it's a great way to get questions about LogStream answered by a wide range of Cribl insiders, and expert peer users, who enjoy sharing their knowledge of the product. Check out individual channels dedicated to feature requests, docs, and other concerns.

Private Slack Channels

Our Enterprise customers have their own private channels where they can communicate directly with their Cribl account team.

Relevant Information We Need

When emailing Cribl Support, please provide as many of the following details as you can – the more, the better:

- Your name.
- Preferred contact method (phone or email).
- LogStream version number affected.
- Description of the issue you’re having.
- What's the issue's scope? (Leader Node, specific Worker Nodes or Groups, or entire deployment; number of nodes impacted) .
- When did the issue begin, or when was it first noticed?
- Did you make any known changes around that time? (Upgrade, config change, network change, etc.).
- Diags for one or more affected systems (the Leader Node does not process data, so typically, only diags from Workers are necessary).
- Sample event data for testing Pipeline issues (provide a text file, rather than screenshots).
- Any troubleshooting steps that you've already taken.

Pulling Diags

Providing us diags from your environment will speed up the time to resolution. For instructions on how to pull a diag file, see Diagnosing Issues.

With LogStream 2.4.1 or later, the diag is uploaded from the browser, rather than from the LogStream Node. This means that your LogStream Worker Nodes do not need Internet access. With LogStream 2.4.0 or earlier, you’ll need to transfer the diags from your Worker Nodes.

> ⚠ Diag bundles for Leader Nodes do not include diag bundles for any Worker Nodes.
>
> If you would like to upload your diag file via the GUI or CLI, you'll need outbound Internet access to https://diag-upload.cribl.io, plus a

> valid support case number (provided in your case confirmation email).

Diag Workarounds

If your organization does not permit outbound access to https://diag-upload.cribl.io, you can also submit diags through Intercom.

If none of these options work with your organization's policies, please work directly with your Support engineer to find a solution.

Known Issues

2021-05-20 – 3.0.0 – Multiple Functions Break LogStream 3.0 Pipelines

Problem: After upgrade to LogStream 3.0.0, including any of the following Functions in a Pipeline can break the Pipeline: GeoIP, Redis, DNS Lookup, Reverse DNS, Tee. Symptom is an error of the form: `Pipeline process timeout has occurred.` Less seriously, including these Functions in a Pipeline can suppress Preview's display of fields/values.
Workarounds: If you use these Functions in your Pipelines, stay with (or restore) a pre-3.0 version until LogStream 3.0.1 is available.
Fix: Staged for LogStream 3.0.1.

2021-05-19 – 3.0.0 – Leader's Changes fly-out stays open after Commit

Problem: In the Leader's left nav, the Changes fly-out remains stuck open after you commit pending changes.
Workarounds: Hover or click away. Then hover or click back to reopen the fly-out.
Fix: Planned for LogStream 3.0.1.

2021-05-18 – 3.0.0 – Packs > Export in "Merge" mode omits schemas and custom Functions

Problem: Exporting a Pack with the export mode set to `Merge` omits schemas and custom Functions configured within the Pack's **Knowledge > Schemas**.
Workarounds: 1. Change the export mode to `Merge safe`, and export again.
2. If that doesn't preserve the schema and Functions, revert to `Merge` export mode; install the resulting Pack onto its target(s); and then manually copy/paste the schema(s) and Functions from the source Pack's UI to the target Pack's UI.
Fix: Planned for LogStream 3.0.1.

2021-05-10 – 2.4.5 – Elasticsearch Destination, with Auto version discovery, doesn't send Authorization header

Problem: When the Elasticsearch Destination has Basic Authentication enabled, and its **Elastic version** field specifies `Auto` version discovery, LogStream fails to send the configured username and password credentials along with its API initial request. Elasticsearch responds with an HTTP 401 error.
Workaround: Explicitly set the **Elastic version** to either `7.x` or `6.x` (depending on your Elasticsearch cluster's version); then stop and restart LogStream to pick up this configuration change.
Fix: Planned for LogStream 3.0.1.

2021-05-04 – 2.4.5 – Office 365 Message Trace Source skips events

Problem: The Event Breaker Rule provided for the Office 365 Message Trace Source mistakenly presets the **Default timezone** to `ETC/GMT-0`. This setting causes LogStream to discover events but not collect them.
Workaround: Reset the Rule's **Default timezone** to `UTC`, then click **OK** and resave the Ruleset.
Fix: In 3.0.0.

2021-04-20 – v.2.4.3–2.4.5 – Orphaned S3 staging directories

Problem: Using the S3 Destination, defining a partitioning expression with high cardinality can proliferate a large number (up to millions) of empty directories. This is because LogStream cleans up staged files, but not staging directories.
Workaround: Programmatically or manually delete stale staging directories (e.g., those older than 30 days).
Fix: Planned for LogStream 3.1.0.

2021-04-12 – 2.4.4 – Splunk Sources do not support multiple-metric events

Problem: LogStream's Splunk Sources do not support multiple-measurement metric data points. (LogStream's Splunk Load Balanced Destination does.)
Fix: Planned for LogStream 3.0.1.

2021-04-07 – v.2.4.2–2.4.5 – Google Cloud Storage Destination fails to upload files > 5 MB

Problem: The Google Cloud Storage Destination might fail to put objects into GCS buckets. This happens with files larger than 5 MB, and causes the Google Cloud API to report a vague `Invalid argument` error.
Workaround: Set the **Max file size (MB)** to 5 MB. Also, reduce the **Max file open time (sec)** limit from its default `300` (5 minutes) to a shorter interval, to prevent files from growing to the 5 MB threshold. (Tune this limit based on your observed rate of traffic flow through the Destination.)
Fix: In LogStream 3.0.0.

2021-03-31 – v.2.4.4 – Local login option visible even when disabled

Problem: The **Log in with local user** option is displayed to users even when you have disabled **Settings > Authentication > Allow local auth** for an OpenID Connect identity provider.
Workaround: Advise users to ignore this button. Although visible, it will not function.
Fix: In LogStream 3.0.0.

2021-03-31 – v.2.4.0–2.4.4 – Splunk TCP and LB Destinations' Workers trigger OOM errors and restart

Problem: With a Splunk TCP or Splunk Load Balanced Destination created after upgrading to LogStream 2.4.x, Workers' memory consumption may grow without bound, leading to out-of-memory errors. The API Process will restart the Workers, but there might be temporary outages.
Workaround: Toggle the Destination's **Advanced Settings > Minimize in-flight data loss** slider to `No`. This will preserve Processes killed by OOM conditions.
Fix: In LogStream 2.4.5.

2021-03-31 – v.2.4.4 – OpenID Connect authentication always shows local-auth fallback

Problem: Even if OpenID Connect external authentication is configured to disable **Allow local auth**, LogStream's login page displays a **Log in with local user** button.
Workaround: Do not click that button.
Fix: Planned for LogStream 3.0.0.

2021-03-31 – v.2.4.4 – Authentication options mistakenly display Cribl Cloud

Problem: The **Settings > Authentication > Type** drop-down offers a `Cribl Cloud` option, which is not currently functional. Attempting to configure and save this option could lock the `admin` user out of LogStream.
Workaround: Do not select, configure, or save that option.
Fix: In LogStream 2.4.5.

2021-03-30 – v.2.4.4 – Can't disable some Sources from within their config modals

Problem: In configuration modals for the Azure Blob Storage and Office 365 Message Trace Sources, the **Enabled** slider cannot be toggled off, and its tooltip doesn't appear.
Workaround: Disable your configured Source (where required) from the **Manage Blob Storage Sources** or the **Manage Message Trace Sources** page.
Fix: In LogStream 2.4.5.

2021-03-29 – v.2.4.x – SpaceOut Destination is broken

Problem: Within the SpaceOut game, you cannot shoot, and your player is immortal.
Workaround: There are other video games. After we defeat COVID, you'll even be able to buy a PS5.
Fix: Restored in LogStream 2.4.5.

2021-03-24 – v.2.4.x – Cribl App for Splunk blocks admin password changes, configuration changes, and Splunk-based authentication

Problem: Attempting to change the admin password via the UI triggers a 403/Forbidden message. You can reset the password by editing `users.json` , but can't save configuration changes to Settings, Pipelines, etc., because RBAC Roles are not properly applied.
Workaround: Using a 2.3.x version of the App enables **local** authentication and enables changes to Cribl/LogStream passwords and configuration/settings.
Fix: In LogStream 2.4.4.

2021-03-22 – v.1.7 through 2.4.3 – Azure Event Hubs Destination: Compression must be manually disabled

Problem: LogStream's Azure Event Hubs Destination provides a **Compression** option that defaults to `Gzip` . However, compressed Kafka messages are not

yet supported on Azure Event Hubs.
Workaround: Manually reset **Compression** to `None`, then resave Azure Event Hubs Destinations.
Fix: In LogStream 2.4.4.

2021-03-17 – v.2.4.2, 2.4.3 – Parser Function > List of Fields copy/paste fails

Problem: When copying/pasting **List of Fields** contents between Parser Functions via the Copy button, the paste operation inserts unintended metadata instead of the original field references.
Workaround: Manually re-enter the second Parser Function's **List of Fields**.
Fix: In LogStream 2.4.4.

2021-03-13 – v.2.4.3 – UI can't find valid TLS .key files, blocking Master restarts and Worker reconfiguration

Problem: After upgrading to v.2.4.3, the UI fails to recognize valid TLS `.key` files, displaying spurious error messages of the form:
"File does not exist:
`$CRIBL_HOME/local/cribl/auth/certs/<keyname>key`."
An affected Master will not restart. Affected Workers will restart, but will not apply changes made through the UI.
Workaround: Ideally, specify an absolute path to each key file, rather than relying on environment variables. If you're locked out of the UI, you'll need to manually edit the referenced paths within these configuration files in LogStream subdirectories: `local/cribl/cribl.yml` (General > API Server TLS settings) and/or `local/_system/instance.yml` (Distributed > TLS settings). Contact Cribl Support if you need assistance. A more drastic workaround is to disable TLS for the affected connections.
Fix: In LogStream 2.4.4.

2021-03-12 – v.2.4.2 – Redis Function with specific username can't authenticate against Redis 6.x ACLs

Problem: The Redis Function, when used with a specific username and Redis 6.x's Access Control List feature, fails due to authentication problems.
Workaround: In the Function's **Redis URL** field, point to the Redis `default` account, either with a password
(e.g., `redis://default:Password1@192.168.1.20:6379`) or with no password (redis://192.168.1.20:6379). Do not specify a user other than

`default`.
Fix: Planned for LogStream 3.0.1.

2021-03-09 – v.2.4.3 – Splunk Destinations' in-app docs mismatch UI's current field order

Problem: For the Splunk Single Instance and Splunk Load Balanced Destinations, the in-app documentation omits the UI's **Advanced Settings** section. Some fields are documented out-of-sequence, or are omitted.
Workaround: Refer to the UI's tooltips, to the corrected Splunk Single Instance and Splunk Load Balanced online docs, and/or to the corrected PDF.
Fix: In LogStream 2.4.4.

2021-03-08 – v.2.4.3 – Enabling Git Collapse Actions breaks Commit & Deploy

Problem: After enabling **Settings** > **Distributed Settings** > **Git Settings** > **General** > **Collapse Actions**, selecting **Commit & Deploy** throws a 500 error.
Workaround: Disable the **Collapse Actions** setting, then commit and deploy separately.
Fix: In LogStream 2.4.4.

2021-03-08 – v.2.4.3 – S3 Collector lacks options to reuse HTTP connections and allow-self signed certs

Problem: As of v.2.4.3, LogStream's AWS-related Sources & Destinations provide options to reuse HTTP connections, and to establish TLS connections to servers with self-signed certificates. However, the S3 Collector does not yet provide these options.
Fix: In LogStream 2.4.4.

2021-03-04 – v.2.4.2 – Esc key closes both Event Breaker Ruleset modals

Problem: After adding a rule to a **Knowledge** > **Event Breaker Ruleset**, pressing `Esc` closes the parent Ruleset modal along with the child Rule modal.
Workaround: Close the Rule modal by clicking either its **Cancel** button or its close box.
Fix: In LogStream 2.4.3.

2021-03-04 – v.2.4.2 – Aggregations Function in post-processing Pipeline addresses wrong Destination

Problem: An Aggregations Function, when used in a post-processing Pipeline, sends data to LogStream's Default Destination rather than to the Pipeline's attached Destination.
Workaround: If applicable, use the Function in a processing or pre-processing Pipeline instead.
Fix: In LogStream 2.4.3.

2021-02-25 – v.2.4.2 – On Safari, Event Breaker shows no OUT events

Problem: When viewing an Event Breaker's results on Safari, no events are displayed on the Preview pane's **OUT** tab.
Workaround: Use another supported browser.
Fix: In LogStream 2.4.3.

2021-02-22 – v.2.4.3 – Collection jobs UI errors

Problem: Collection jobs are missing from the **Monitoring > Sources** page, even though they are returned by metric queries. Also, the **Job Inspector > Live** modal displays an empty, unintended **Configure** tab.
Workaround: Use the Job Inspector to access collection results. Ignore the **Configure** tab.
Fix: In LogStream 2.4.4.

2021-02-19 – v.2.4.2 – Upon upgrade, Git remote repo setting breaks, blocking Worker Groups

Problem: If a Git remote repo was previously configured, upgrading to LogStream v.2.4.2 throws errors of this form upon startup: `Failed to initialize git repository. Config versioning will not be available ... Invalid URL ...` . The Master cannot commit or deploy to any Worker Group.
Workarounds: 1. Downgrade back to v.2.4.1 (or your previous working version). 2. Switch from Basic authentication to SSH authentication against the repo, to remove the username from requests. (The apparent root cause is Basic/http auth using a valid URL and username, but missing a password.)
Fix: In LogStream 2.4.3.

2021-02-19 – v.2.4.0, 2.4.1, 2.4.2 – Splunk (S2S) Forwarder access control blocks upon upgrade to LogStream 2.4.x

Problem: If Splunk indexers have forwarder tokens enabled, and worked with LogStream 2.3.x before, upgrading to LogStream 2.4.x causes data to stop flowing.
Workaround: If you encounter this problem, rolling back to your previously installed LogStream version (such as v.2.3.4) resolves it.
Fix: In LogStream 2.4.3.

2021-02-10 – v.2.4.0, 2.4.1 – With Splunk HEC Source, JSON payloads containing embedded objects trigger high CPU usage

Problem: Splunk HEC JSON payloads containing nested objects trigger high CPU usage, due to a flaw in JSON parsing.
Workaround: If you encounter this problem, rolling back to your previously installed LogStream version (such as v.2.3.4) resolves it.
Fix: In LogStream 2.4.2.

2021-01-30 – v.2.4.0 – Worker Nodes cannot connect to Master

Problem: Worker Nodes cannot connect to the Master after the Master is upgraded to v.2.4.0.
Workaround: Disable compression on the Workers. You can do so through the Workers' UI at **System Settings > Distributed Settings > Master Settings > Compression**, or by commenting out this line in each Worker's `cribl.yml` config file:

```
compression: gzip
```

Fix: In LogStream 2.4.1.

2021-01-25 – v.2.4.0 – S3 collection stops working due to auth secret key issues.

Problem: S3 collection stops after upgrade to 2.4.0 due to secret key re-encryption.
Workaround: Re-configure S3, save and re-deploy.
Fix: In LogStream 2.4.1.

2021-01-14 – v.2.4.0 – Google Cloud Storage Destination Needs Extra Endpoint to Initialize

Problem: The Google Cloud Storage Destination fails to initialize, displaying an error of the form: `Bucket does not exist!`
Workaround: In the `outputs.yml` file, under your `cribl-gcp-bucket` key endpoint, add: `https://storage.googleapis.com` . (in a single-instance deployment, locate this file at `$CRIBL_HOME/local/cribl/outputs.yml` . In a distributed deployment, locate it at `$CRIBL_HOME/groups/<group name>/local/cribl/outputs.yml` .)
Fix: In LogStream 2.4.1.

2021-01-14 – v.2.4.0 – Worker Groups' Settings > Access Management Is Absent from UI

Problem: In this release, the **Worker Groups > <group-name> > System Settings** UI did not display the expected **Access Management**, **Authentication**, and **Local Users** sections.
Workaround: Manually edit the `users.json` file.
Fix: In LogStream 2.4.1.

2021-01-13 – v.2.4.0 – Route Filters Aren't Copied to Capture Modal

Problem: On the **Routes** page, selecting **Capture New** in the right pane does not copy custom **Filter** expressions to the resulting **Capture Sample Data** modal. That modal's **Filter Expression** field always defaults to `true` .
Workarounds: 1. Bypass the **Capture New** button. Instead, from the Route's own ••• (Options) menu, select **Capture**. This initiates a capture with the **Filter Expression** correctly populated. 2. Copy/paste the expression into the **Capture Sample Data** modal's **Filter Expression** field. Or, if the expression is displayed in that field's history drop-down, retrieve it.
Fix: In LogStream 2.4.1.

2021-01-13 – v.2.4.0 – Destinations' Documentation Doesn't Render from UI

Problem: Clicking the **Help** Help ▶? link in a Destination's configuration modal displays the error message: "Unable to load docs. Please check LogStream's online documentation instead."
Workarounds: 1. Go directly to the online Destinations docs, starting here.
2. Follow the UI link to the docs landing page, click through to open or

download the current PDF, and scroll to its Destinations section.
Fix: In LogStream 2.4.1.

2021-01-13 – v.2.4.0 – `Esc` Key Doesn't Consistently Close Modals

Problem: Pressing `Esc` with focus on a modal's drop-down or slider doesn't close the modal as expected. (Pressing `Esc` with focus on a free-text field, combo box, or nothing does close the modal – displaying a confirmation dialog first, if you have unsaved changes.)
Workarounds: Click the **X** close box at upper right, or click **Cancel** at lower right.
Fix: Planned for LogStream 2.4.2.

2020-12-17 – v.2.3.0+ – Free-License Expiration Notice, Blocked Inputs

Problem: LogStream reports an expired Free license, and blocks inputs, even though Free licenses in v.2.3.0 do not expire.
Workaround: This is caused by time-limited Free license key originally entered in a LogStream version prior to 2.3.0. Go to **Settings > Licensing**, click to select and expand your expired Free license, and click **Delete license**. LogStream will recognize the new, permanent Free license, and will restore throughput.
Fix: In LogStream 2.4.1.

2020-11-14 – v.2.3.3 – Null Fields Redacted in Preview, but Still Forwarded

Problem: Where event fields have null values, LogStream (by default) displays them as struck-out in the right Preview pane. The preview is misleading, because the events are still sent to the output.
Workaround: If you do want to prevent fields with null values from reaching the output, use an Eval Function, with an appropriate Filter expression, to remove them.
Fix: Preview corrected in LogStream 2.3.4.

2020-10-27 – v.2.3.2 – Cannot Name or Save New Event Breaker Rule

Problem: After clicking **Add Rule** in a new or existing Event Breaker Ruleset, the **Event Breaker Rule** modal's **Rule Name** field is disabled. Because **Rule Name** is

mandatory field, this also disables saving the Rule via the **OK** button.
Fix: In LogStream 2.3.3.

2020-10-12 – v.2.3.1 – Deleting One Function Deletes Others in Same Group

Problem: After inserting a new Function into a group and saving the Pipeline, deleting the Function also deletes other Functions lower down in the same group.
Fix: In LogStream 2.3.2.
Workaround: Move the target Function out of the group, resave the Pipeline, and only then delete the Function.

2020-09-27 – v.2.3.1 – Enabling Boot Start as Different User Fails

Problem: When a root user tries to enable boot-start as a different user (e.g., using `/opt/cribl/bin/cribl boot-start enable -u <some-username>`), they receive an error of this form:

```
error: found user=0 as owner for path=/opt/cribl, expected uid=NaN.
Please make sure CRIBL_HOME and its contents are owned by the uid=NaN by r
[sudo] chown -R NaN:[$group] /opt/cribl
```

Fix: In LogStream 2.3.2.
Workaround: Install LogStream 2.2.3 (which you can download here), then upgrade to 2.3.1.

2020-09-17 – v.2.3.0 – Worker Groups menu tab hidden after upgrade to LogStream 2.3.0

Problem: Upon upgrading an earlier, licensed LogStream installation to v. 2.3.0, the **Worker Groups** tab might be absent from the Master Node's top menu.
Fix: In LogStream 2.3.1.
Workaround: Click the **Home > Worker Groups** tile to access Worker Groups.

2020-09-17 – v.2.3.0 – Cannot Start LogStream 2.3.0 on RHEL 6, RHEL 7

Problem: Upon upgrading to v. 2.3.0, LogStream might fail to start on RHEL 6 or 7, with an error message of the following form. This occurs when the user running LogStream doesn't match the LogStream binary's owner. LogStream

2.3.0 applies a restrictive permissions check using `id -un <uid>`, which does not work with the version of `id` that ships with these RHEL releases.

```
id: 0: No such user
ERROR: Cannot run command because user=root with uid=0 does not own execut
```

Fix: In LogStream 2.3.1.
Workaround: Update your RHEL environment's `id` version, if possible.

2020-09-17 – v.2.3.0 – Cannot Start LogStream 2.3.0 with OpenId Connect

Problem: Upon upgrading an earlier LogStream installation to v. 2.3.0, OIDC users might be unable to restart the LogStream server.
Fix: In LogStream 2.3.1.
Workaround: Edit `$CRIBL_HOME/default/cribl/cribl.yml` to add the following lines to its the `auth` section:

```
filter_type: email_whitelist
scope: openid profile email
```

2020-06-11 – v.2.1.x – Can't switch from Worker to Master Mode

Problem: In a Distributed deployment, attempting to switch Distributed Settings from Worker to Master Mode blocks with a spurious "Git not available...Please install and try again" error message.
Fix: In LogStream 2.3.0.
Workaround: To initialize `git`, switch first from Worker to Single mode, and then from Single to Master mode.

2020-05-19 – v.2.1.x – Login page blocks

Problem: Entering valid credentials on the login page (e.g., `http://localhost:9000/login`) yields only a spinner.
Fix: In LogStream 2.3.0.
Workaround: Trim `/login` from the URL.

2020-02-22 – v.2.1.x – Deleting resources in `default/`

Problem: In a Distributed deployment, deleting resources in `default/` causes them to reappear on restart.
Workaround/Fix: In progress.

2019-10-22 – v. 2.0 – In-product upgrade issue on v2.0

Problem: Using in-product upgrade feature in v.1.7 (or earlier) fails to upgrade to v2.0, due to package-name convention change.
Workaround/Fix: Download the new version and upgrade per steps laid out here.

2019-08-27 – v.1.7 – In-product upgrade issue on v1.7

Problem: Using in-product upgrade feature in v1.6 (or earlier) fails to upgrade to v1.7 due to package name convention change.
Workaround/Fix: Download the new package and upgrade per steps laid out here.

2019-03-21 – v.1.4 – S3 stagePath issue on upgrade to v.1.4+

Problem: When upgrading from v1.2 with a S3 output configured, `stagePath` was allowed to be undefined. In v.1.4+, `stagePath` is a required field. This might causing schema violations when upgrading older configs.
Workaround/Fix: Reconfigure the output with a valid `stagePath` filesystem path.

Common Errors and Warnings

This page lists common error and warning messages that you might find in LogStream's internal logs and/or UI. It includes recommendations for resolving the errors/warnings. Messages are grouped by component (Sources, Destinations, etc.). Note that:

- We've excerpted only the salient part of the error message from each event. We haven't listed full events.
- Examples don't preserve case sensitivity from the original events.
- 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.

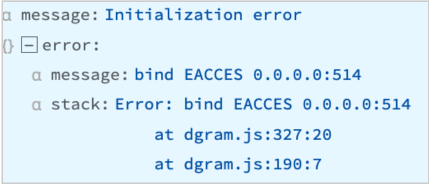
Sources (General)

Where: These events will be in the Worker process logs.

Error: "bind EACCES 0.0.0.0:514"

Cause: LogStream doesn't have proper privileges to bind to the specified IP and port. Usually this simply an issue of LogStream running as a non-root user, but a Source was accidentally configured to use a privileged port below 1024.

Recommendation: Check your LogStream Sources' configuration. The error's `channel` field will indicate which input is affected.



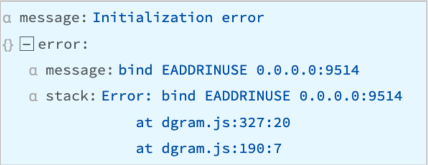
```
α message: Initialization error
{} ⊟ error:
   α message: bind EACCES 0.0.0.0:514
   α stack: Error: bind EACCES 0.0.0.0:514
                at dgram.js:327:20
                at dgram.js:190:7
```

ACCES error

Error: “bind EADDRINUSE 0.0.0.0:9514”

Cause: The interface and port combination is already in use by another process, which might or might not be LogStream.

Recommendation: Check your LogStream 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.



```
α message: Initialization error
{} ⊟ error:
   α message: bind EADDRINUSE 0.0.0.0:9514
   α stack: Error: bind EADDRINUSE 0.0.0.0:9514
                at dgram.js:327:20
                at dgram.js:190:7
```

ADDRINUSE error

HTTP-based Source

Where: These events will be in the Worker Process logs.

Message: "Dropping request because token invalid","authToken": "Bas...Njc="" (v2.4.5 and later)

Cause: The specified token is invalid. Note the above message is logged only at debug level.

Kafka-based Source

Where: These events will be in the Worker Process logs.

Error: "KafkaJSProtocolError: Not authorized to access topics: [Topic authorization failed]"

Cause: The username does not have read permissions for the specified topic.

Splunk HEC Source

Where: These events will be in the Worker Process logs.

Error: "Malformed HEC event"

Cause: The event is missing both `event` and `fields` fields.

Error: "Invalid token"

Cause: Auth token(s) are defined, but the token specified in the HEC request doesn't match any defined tokens, therefore it's invalid.

Error: "Invalid index"

Cause: The Splunk HEC Source's **Allowed Indexes** field is configured with specific indexes, but the client's HTTP request didn't specify any of them.

Error: "{"text":"Server is busy","code":9,"invalid-event-number":0}" (Note: this is not logged in LogStream but would be found in the response payload sent to your HEC client.)

Cause: "Server is busy" is the equivalent of a 503 HTTP response code. The most likely cause is the **Max active requests** setting in the HEC input's Advanced Settings is insufficient to service the number of simultaneous HEC requests. Increase the value and monitor your clients to see if the 503 response is eliminated.

TLS Errors

Where: These events can be in Worker Process or API logs on the Workers or Leader, depending on whether the issue is associated with a Source or Destination, etc.

Error: “certificate has expired”

Cause: 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 LogStream is serving as the client or server), so that encryption can still occur without authentication. Or renew the expired certificate.

Error: “Client network socket disconnected before secure TLS connection was established”

Cause: 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.

Error: “Unable to get local issuer certificate”

Cause: Client can’t validate the server certificate’s CA (i.e., the issuer) because it doesn’t trust the CA cert. 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.

Error: “self signed certificate”

Cause: 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.

Error: “Hostname/IP does not match certificate’s altnames"

Variations:
“Hostname/IP does not match certificate’s altnames: host: 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”

Cause: 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.

Error: "140244944713600:error:141E70BF:SSL routines:tls_construct_client_hello:no protocols available:"

Cause: 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.

Error: "140251374995328:error:1408F10B:SSL routines:ssl3_get_record:wrong version number"

Cause: The client is using TLS but the server is probably not configured for TLS.

Recommendation: Review the TLS settings on the server.

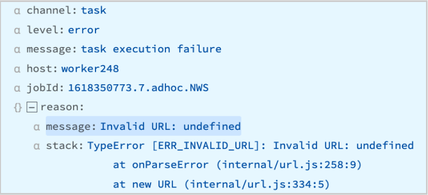
REST API Collector

Where: These events will be in the Worker Process logs.

Error: "reason.stack == `TypeError [ERR_INVALID_URL]: Invalid URL: https%3A%2F%2Ftype.fit%2Fapi%2Fquotes" at onParseError (internal/url.js:258:9)"

Cause: This is due to unnecessarily encoding the Discover/Collect URL.

Recommendation: Remove the encoding function for the URL. URLs will rarely need text be encoded and when they do it's only the parts that need it that should be encoded, otherwise if the entire URL is encoded unnecessarily then errors like this will occur.



```
α channel: task
α level: error
α message: task execution failure
α host: worker248
α jobId: 1618350773.7.adhoc.NWS
{} ⊟ reason:
    α message: Invalid URL: undefined
    α stack: TypeError [ERR_INVALID_URL]: Invalid URL: undefined
                 at onParseError (internal/url.js:258:9)
                 at new URL (internal/url.js:334:5)
```

Invalid URL

AWS Sources/Destinations and S3-Compatible Stores

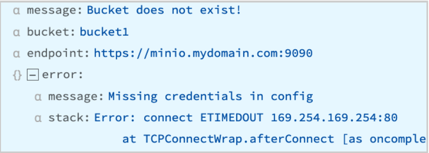
Where: These events will be in the Worker Process logs.

Error: "Missing credentials in config" or "stack:Error: connect ETIMEDOUT 169.254.169.254:80"

Cause 1: Can occur when **Authentication** is set to **Auto**, but no IAM role is attached.

Cause 2: Can occur on LogStream 2.4.4 or earlier, when an IAM role is attached to the EC2 instance, but the instance is using instance metadata v2.

Recommendations: Change to Manual Authentication; attach an IAM role; or if using IMDv2, switch to IMDv1 (if possible) or upgrade to LogStream 2.4.5 or later.



```
α message: Bucket does not exist!
α bucket: bucket1
α endpoint: https://minio.mydomain.com:9090
{} ⊟ error:
   α message: Missing credentials in config
   α stack: Error: connect ETIMEDOUT 169.254.169.254:80
               at TCPConnectWrap.afterConnect [as oncomple
```

Missing credentials in config

Destinations (General)

Where: These events will be in the Worker Process logs, unless otherwise noted.

Warning: “sending is blocked”

Cause: 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 Leadr 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.

Warning: "exerting backpressure" (v2.4.0-2.4.1)

Cause: The Destination is blocking. This message is logged immediately upon detecting backpressure, for Destinations using any protocol. Some backpressure is normal when measured over timescales under 1 second, therefore this message can appear quite frequently, and is not indicative of a problem (which is why it's a warning).

Warning: "begin backpressure" and "end backpressure" (v2.4.2 and later)

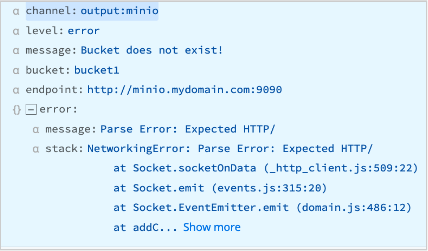
Cause: The Destination is blocking. Like the "sending is blocked" message, "begin backpressure" is logged only after 1 second of blocking. Unlike the "exerting backpressure" message, it is logged only once while backpressure is occurring (at the start), and it will always be followed by the "end backpressure" message.

MinIO Destination

Where: These events will be in the Worker Process logs.

Error: "Parse Error: Expected HTTP/"

Cause: The Worker is trying to use HTTP, but the server is expecting HTTPS.



```
α channel: output:minio
α level: error
α message: Bucket does not exist!
α bucket: bucket1
α endpoint: http://minio.mydomain.com:9090
{} ⊟ error:
   α message: Parse Error: Expected HTTP/
   α stack: NetworkingError: Parse Error: Expected HTTP/
                at Socket.socketOnData (_http_client.js:509:22)
                at Socket.emit (events.js:315:20)
                at Socket.EventEmitter.emit (domain.js:486:12)
                at addC... Show more
```

Parse Error: Expected HTTP

Diag Command

Where: On stdout

Warning: "You are running Cribl LogStream CLI as user=root, while the binary is owned by the user=cribl."

Full Text: "WARNING: You are running Cribl LogStream 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."

Cause: 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.

Cluster

Where: These events will be in the Workers' API logs.

Error: "access denied"

Cause: The Worker's authtoken (located in `$CRIBL_HOME/local/_system/instance.yml`) is missing or doesn’t match the Leader's.

Web UI

Where: In LogStream's UI.

Error: "The Config Helper service is not available because a configuration file doesn't exist or the settings are invalid. Please fix it and restart LogStream."

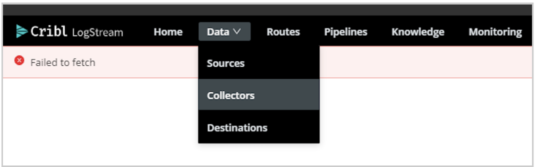
Cause: Occurs only on Leader Nodes. Each Worker Group relies on a config helper process that runs on its behalf on a Leader Node, and that process is not currently running. This error is usually triggered by restarting LogStream on the Leader Node, and then trying to access its **Worker Groups**, (2.x) or **Groups** or **Configure** (3.x) menu too soon.

Recommendation: Try again in a few seconds.

Error: "Failed to fetch"

Cause: This occurs only when accessing **Data** > **Collectors**. It occurs with the Brave browser (because the newly loaded page's URL includes the string `collector`). It can also occur with ad blockers.

Recommendation: If you have an ad blocker, allowlist the `https://<hostname>:9000/jobs/collectors` (single-instance) URL or the `https://<masternode>:9000/m/<group_name>/jobs/collectors` (Leader) URL.



Failed to fetch

Failed to fetch

Git Push Errors

This page anticipates common errors you might see in LogStream'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>
```

Next, verify the remote repo from the command line, as follows:

```
cd $CRIBL_HOME/.git
git remote -v
```

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 th
remote: error: GH001: Large files detected. You may want to try Git Large
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
```

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 Managing Large 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

Git Remote Repos & Trusted CAs

If you are using an internal Git server, a self-signed certificate might prevent LogStream 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
```

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` :

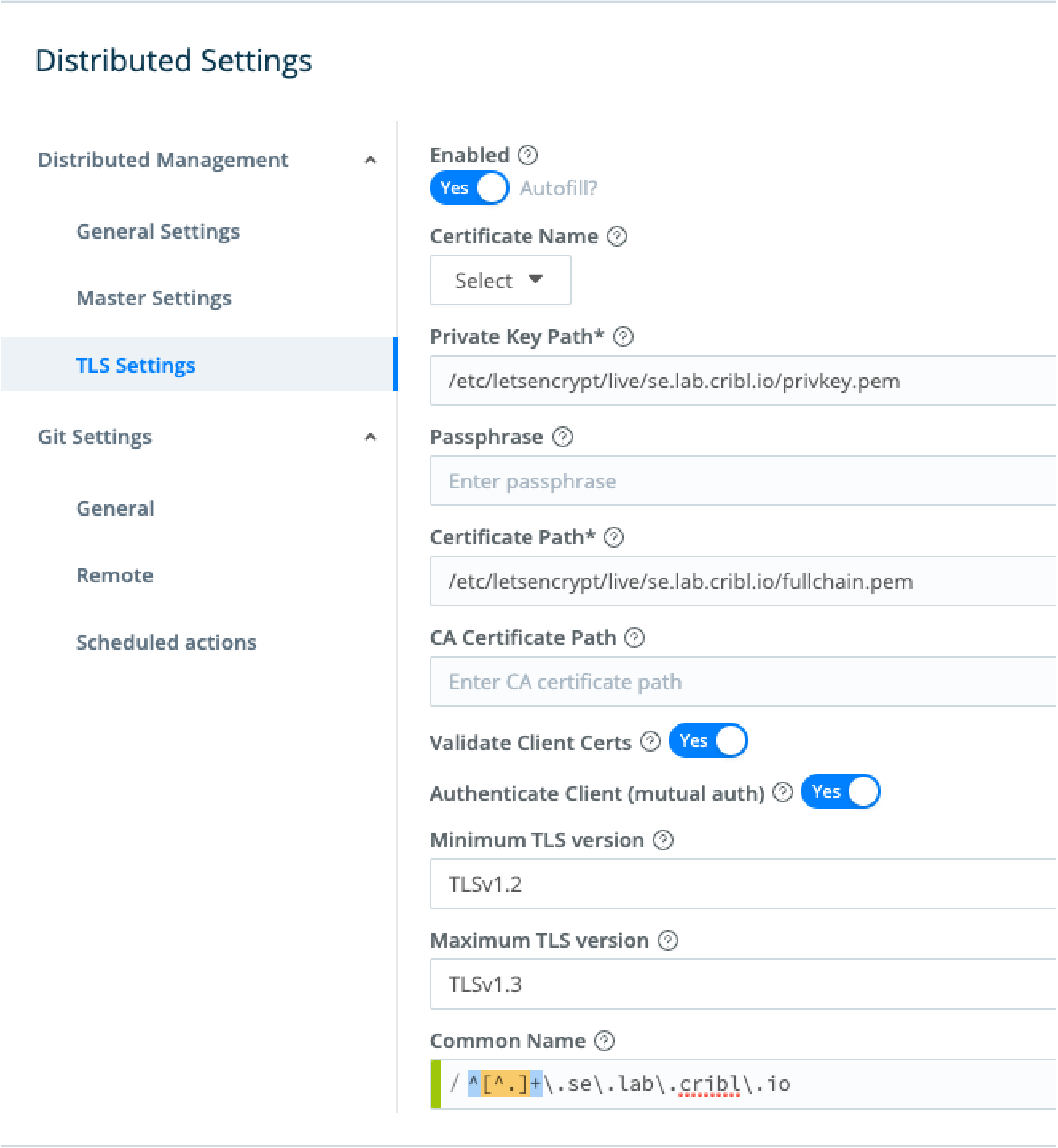
```
...
[Service]
Environment="NODE_EXTRA_CA_CERTS=/opt/cribl/local/cribl/auth/certs/ca.pem"
...
```

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 example below will match all immediate subdomains of `se.lab.cribl.io` , like `madsci.se.lab.cribl.io` .

If you disable **Validate Client Certs**, LogStream will match only on common names.



Enabled: Yes (Autofill?)

Certificate Name: Select

Private Key Path*: /etc/letsencrypt/live/se.lab.cribl.io/privkey.pem

Passphrase: Enter passphrase

Certificate Path*: /etc/letsencrypt/live/se.lab.cribl.io/fullchain.pem

CA Certificate Path: Enter CA certificate path

Validate Client Certs: Yes

Authenticate Client (mutual auth): Yes

Minimum TLS version: TLSv1.2

Maximum TLS version: TLSv1.3

Common Name: `/ ^[^.]+\.se\.lab\.cribl\.io`

Common Name example

C. Extract SSL Certificate Info

As in this example:

```
openssl x509 -in certificate.pem -text -noout
```

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
```

THIRD-PARTY SOFTWARE

Credits

Various components in Cribl LogStream are built and enhanced with software under free or open source licenses. We thank those projects' contributors!

ag-grid-community – 19.1.2
ag-grid-react – 19.1.2
ajv – 6.9.2
ajv-errors – 1.0.1
antd – 3.26.15
as-table – 1.0.36
avsc – 5.4.9
aws-sdk – 2.530.0
@azure/storage-blob – 12.3.0
blueimp-md5 – 2.18.0
cidr-matcher – 1.0.5
clarinet – 0.12.4
classnames – 2.2.6
color-hash – 1.0.3
cron-parser – 2.15.0
d3-time – 1.1.0
d3-time-format – 2.2.3
date-fns – 1.29.0
diff – 3.5.0
diff2html – 2.11.3
echarts – 4.6.0
escodegen – 1.11.1
esprima – 4.0.1
express – 4.16.3
fast-array-diff – 1.0.0

fast-bitset – 1.3.2

file-saver – 1.3.8

good-fences – 0.9.1

http-proxy-agent – 3.0.0

https-proxy-agent – 4.0.0

jwt-simple – 0.5.6

kafkajs – 1.11.0

kafkajs-snappy – 1.1.0

ldapts – 1.10.0

limiter – 1.1.4

lodash – 4.17.15

lz4js – 0.2.0

maxmind – 3.1.2

node-cache – 4.2.0

node-uuid – 1.4.8

numeral – 2.0.6

pako – 1.0.10

papaparse – 5.0.0-beta.0

pbf – 3.2.1

proxy-from-env – 1.0.0

query-string – 6.1.0

react – 16.13.1

react-dom – 16.13.1

react-grid-layout – 0.18.3

react-router-dom – 5.1.2

react-sortable-hoc – 1.11.0

react-split-pane – 0.1.91

@readme/markdown – 6.22.0

redis – 3.0.2

regexpp – 2.0.0

requirejs – 2.3.6

resize-observer-polyfill – 1.5.0

rxjs – 6.5.5

saxen – 8.1.0

simple-git – 1.126.0

snappyjs – 0.6.0

snmp-native – 1.2.0

streamcount – 1.0.1

tar-stream – 2.1.4

timezone-support – 2.0.2

@types/d3-time – 1.0.10

url – 0.11.0
winston – 3.0.0
xmlbuilder – 10.0.0
yaml – 1.3.2