

Cribl LogStream Documentation Manual

Version: v3.1.2

INTRODUCTION	9
About Cribl LogStream	9
Basic Concepts	11
Getting Started Guide	16
DEPLOYMENT	35
Deployment Types	35
Single-Instance Deployment	38
Distributed Deployment	46
Splunk App Deployment *	64
Bootstrap Workers from Leader	69
Kubernetes/Helm Deployment	77
K8s Leader Deployment	80
K8s Worker Deployment	99
(Deprecated:) K8s Master Deployment	112
Docker Deployment	132
Cribl.Cloud Deployment	134
Sizing and Scaling	147
Config Files	155
cribl.yml	158
breakers.yml	160
certificates.yml	161
groups.yml	162
inputs.yml	163
instance.yml	165
jobs.yml	166
job-limits.yml	168
licenses.yml	169
limits.yml	170
logger.yml	171
mappings.yml	172
messages.yml	173
outputs.yml	174
parsers.yml	176
policies.yml	177
regexes.yml	179
roles.yml	180

samples.yml	181
schemas.yml	182
scripts.yml	183
vars.yml	184
Licensing	185
Access Management	192
Authentication	193
Local Users	201
Roles	204
Securing	212
Version Control	221
Persistent Queues	235
Monitoring	239
Internal Metrics	250
Notifications	256
Configuring Targets	264
Upgrading	267
Uninstalling	275
WORKING WITH DATA	276
Event Model	276
Event Processing Order	278
Routes	280
Pipelines	287
Data Onboarding	294
Functions	298
Auto Timestamp	304
Aggregations	309
CEF Serializer	316
Clone	318
Code	319
Comment	322
DNS Lookup	324
Drop	328
Dynamic Sampling	329
Eval	333
Flatten	336
GeoIP	338
Grok	341

JSON Unroll	343
Lookup	345
Mask	350
Numerify	356
Parser	359
Publish Metrics	367
Regex Extract	374
Redis	378
Regex Filter	384
Rename	385
Rollup Metrics	388
Sampling	390
Serialize	392
Suppress	395
Tee	398
Trim Timestamp	401
Unroll	404
XML Unroll	406
Prometheus Publisher (Deprecated)	409
Reverse DNS (deprecated)	412
Collector Sources	414
Filesystem/NFS	419
Azure Blob Storage	423
Google Cloud Storage	428
S3	433
Script	439
REST / API Endpoint	444
Scheduling and Running	454
Job Limits	463
Sources	467
Syslog	471
Splunk TCP	475
Splunk HEC	479
Amazon Kinesis Firehose	485
Amazon Kinesis Streams	489
Amazon SQS	494
Amazon S3	499
Google Cloud Pub/Sub	507

Azure Event Hubs	510
Azure Blob Storage	514
Office 365 Services	525
Office 365 Activity	529
Office 365 Message Trace	536
TCP JSON	540
TCP (Raw)	545
HTTP/S (Bulk API)	550
Raw HTTP/S	556
Elasticsearch API	560
Kafka	564
Metrics	569
SNMP Trap	573
Prometheus Remote Write	576
Prometheus Scraper	580
Grafana	586
Loki	591
AppScope	595
Datagen	599
Cribl Internal	601
Destinations	605
Output Router	609
Splunk Single Instance	611
Splunk Load Balanced	615
Splunk HEC	625
Amazon S3 Compatible Stores	629
Amazon Kinesis Streams	635
Amazon CloudWatch Logs	639
Amazon SQS	643
Azure Blob Storage	648
Azure Monitor Logs	652
Azure Event Hubs	656
Google Chronicle	660
Google Cloud Storage	663
Google Cloud Pub/Sub	667
StatsD	671
StatsD Extended	674
Graphite	677

TCP JSON	680
Syslog	684
Filesystem/NFS	691
Kafka	694
Elasticsearch	699
Honeycomb	703
New Relic Logs & Metrics	706
New Relic Events	711
SNMP Trap	715
InfluxDB	717
MinIO	720
Wavefront	726
SignalFx	729
Sumo Logic	732
Datadog	736
Prometheus	740
Grafana Cloud	744
Loki	751
Webhook	755
DevNull	760
Default	761
Packs	762
Data Preview	778
Securing Data	787
Encryption	788
Decryption	793
Scripts	796
Using Datagens	798
CLI Reference	803
EXPRESSION REFERENCE	817
Introduction to Expression Syntax	817
Cribl Expressions	820
KNOWLEDGE	833
Regex Library	833
Grok Patterns Library	836
Event Breakers	838
Lookups Library	849
Parsers Library	853

Schema Library	855
Global Variables Library	858
TECHNIQUES & TIPS	860
Tips and Tricks	860
Ingest-time Fields	864
Ingest-time Lookups	867
Sampling	872
Access Logs: Apache, ELB, CDN, S3, etc.	874
Firewall Logs: VPC Flow Logs, Cisco ASA, Etc.	877
Masking and Obfuscation	880
Managing Large Lookups	884
Lookups as Filters for Masks	888
Lookups and Regex Magic	892
Regex Filtering	901
Encrypting Sensitive Data	903
Syslog Data Reduction	909
Splunk to Elasticsearch	917
Reducing Windows XML Events	925
Using REST/API Collectors	937
Using S3 Storage and Replay	957
System Proxy Configuration	965
AWS Cross-Account Data Collection	970
OpenID + Azure AD Configuration	975
SSO/Okta Configuration	983
Code Function Examples	992
Setup Guides	1004
Azure Event Hubs Integrations	1005
Splunk Cloud and BYOL Integrations	1010
Webhook/BigPanda Integration	1023
Webhook/Sumo Logic Integration	1028
VIDEOS	1033
Videos	1033
TROUBLESHOOTING	1044
Diagnosing Issues	1044
Working with Cribl Support	1047
Known Issues	1051
Common Errors and Warnings	1069

Git Push Errors	1080
Git Remote Repos & Trusted CAs	1082
THIRD-PARTY SOFTWARE	1085
Credits	1085

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

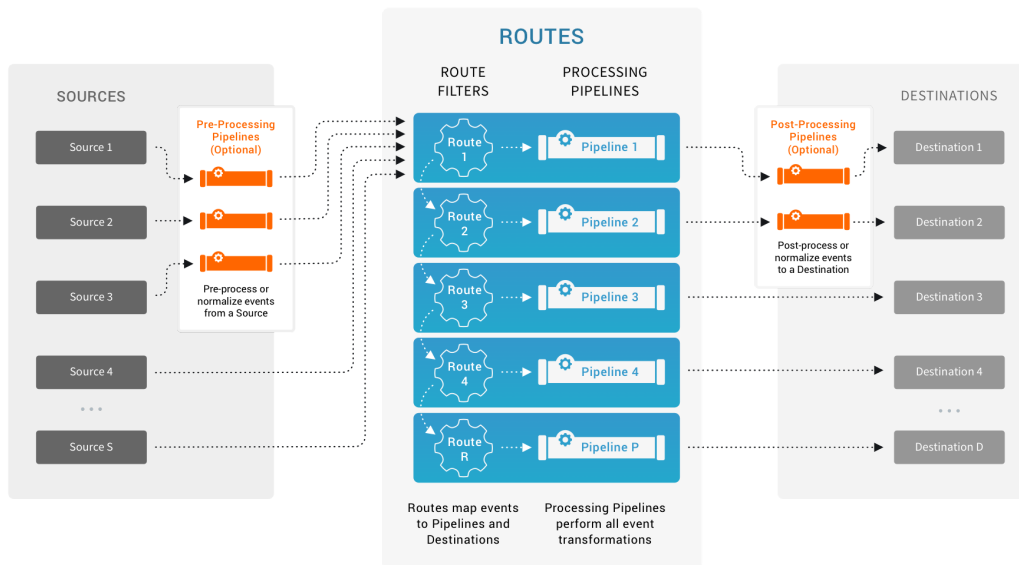
Notable features and concepts to get a fundamental understanding of Cribl LogStream

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](#).



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.

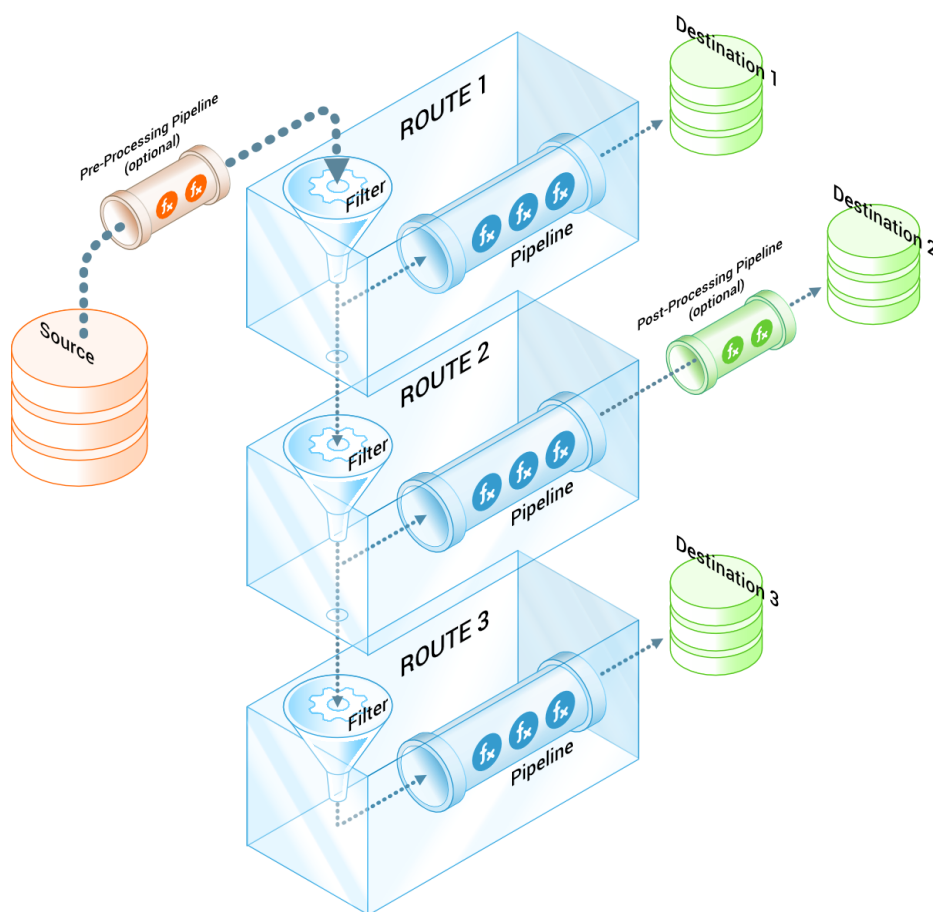
However, if you disable the Route's `Final` flag, one or more event **clones** will be sent down the associated Pipeline, while the original event continues down LogStream's Routing table to be evaluated against other configured Routes. This is very useful in cases where the same set of events needs 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`.

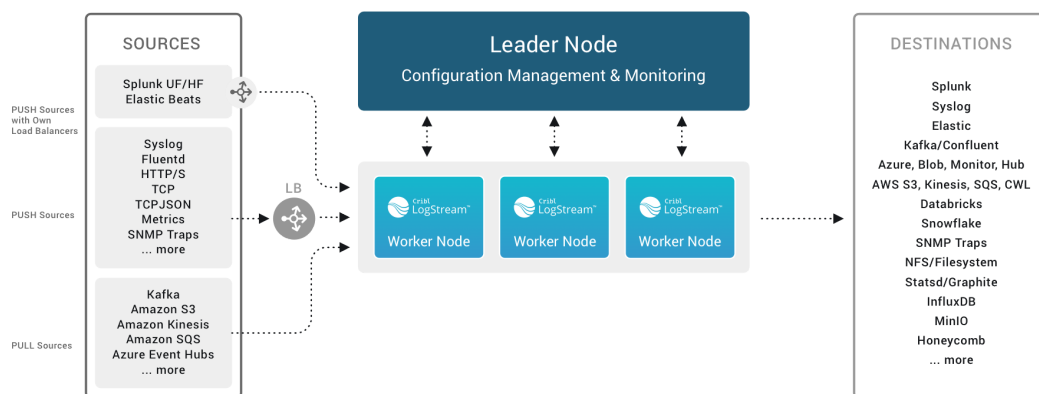
⌵	☐☐☐ #	Function	Filter	👁 All ▾
➤	...	1 Comment	Replace 'foo' with 'bar' (very important)	...
➤	...	2 Eval	"_raw includes('foo')"	On <input type="checkbox"/> ...
➤	...	3 Comment	Mask sensitive information: MD5 hash of a social security number; MD...	...
➤	...	4 Mask	sourcetype='business_event'	On <input type="checkbox"/> ...

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](#)

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 (Intel Processors)

- 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+

OS (ARM64 Processors)

- Linux 64-bit
- Tested so far on Ubuntu (14.04, 16.04, 18.04, and 20.04) and Amazon Linux 2

System

- +4 physical cores, +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) on Intel/Xeon or AMD processors; and to 1 (higher-throughput) vCPU on Graviton2/ARM64 processors.

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 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

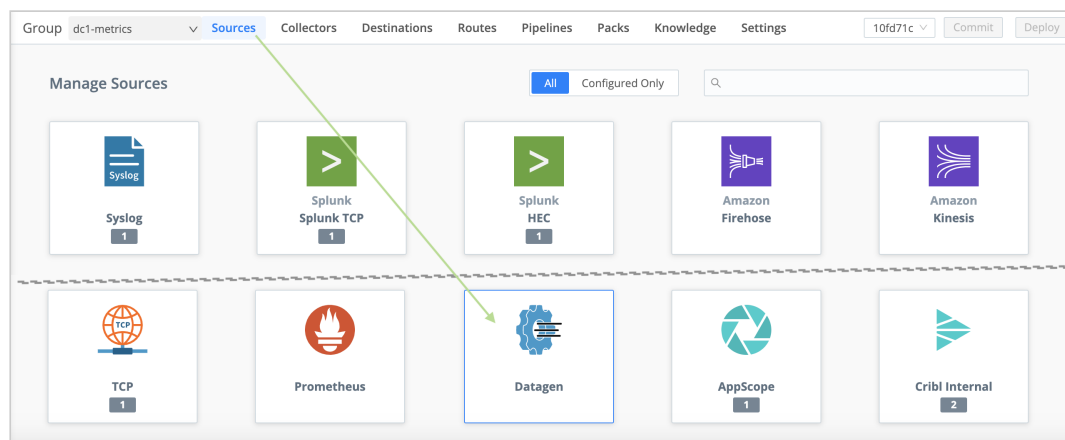
[Functions](#) to refine sample log data.

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 **Sources**.
2. From the **Data Sources** page's tiles or left menu, select **Datagen**.

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

businessesvent On

General Settings

Processing Settings

Fields (metadata)

Pre-processing

Input ID* ?

businessesvent

__inputId== 'datagen:businessesvent' ?

Datagens ?

Data Generator File ?	Events Per Second Per Worker Node ?
businessesvent.log	10

Prev Next Cancel Save

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:

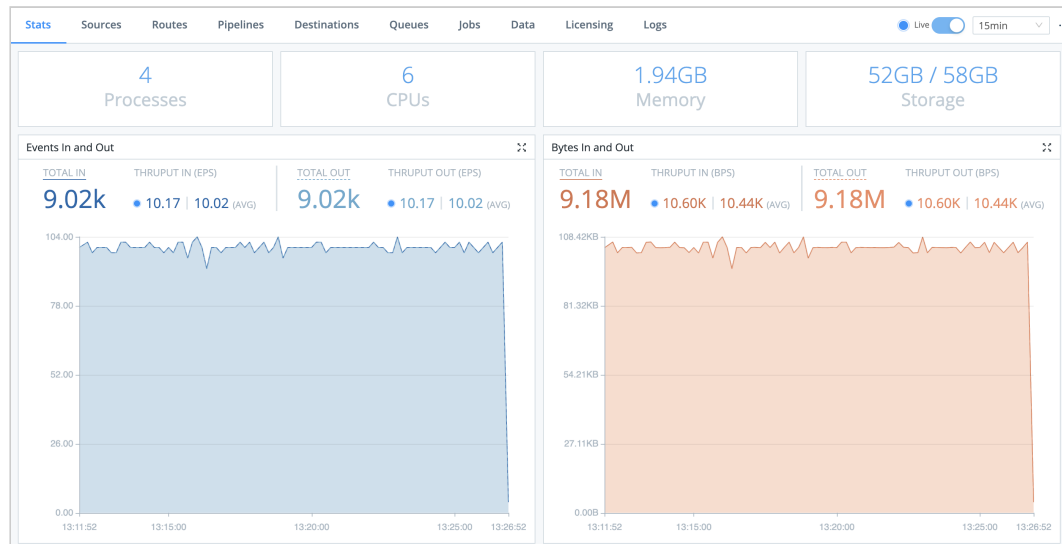
1. From LogStream's top menu, select **Destinations**.
2. Select **DevNull** from the **Data Destinations** page's tiles or left menu.

(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.

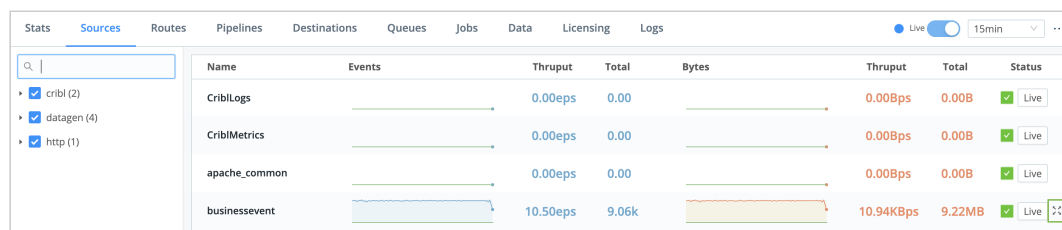
Monitor Data Throughput

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 internal logs. Click **Sources** on the lower submenu to switch to this view:



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:

Stats

Sources

Routes

Pipelines

Destinations

Queues

Jobs

Data

Licensing

Logs

Live

15min

devnull (1)

Name	Events	Thruput	Total	Bytes	Thruput	Total	Status
devnull		10.38eps	9.05k		10.82KBps	9.21MB	<div><div>Live</div><div></div></div>

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.

Pipelines > slicendice + Add Function ⚙️

Attach Pipeline to Route

⬆	#	Function	Filter	Show All
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

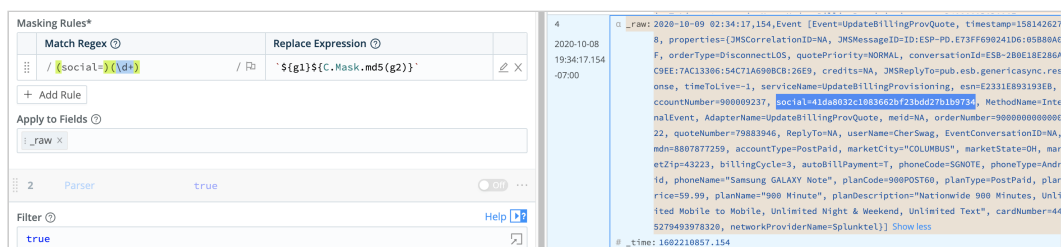
In the right **Preview** pane, notice each that event includes a **social** key, whose value is a (fictitious) raw Social Security number. Before this data goes any further through our Pipeline, let's use LogStream's **Mask** Function to swap in an md5 hash of each SSN.

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.



The screenshot shows the 'Masking Rules' configuration on the left and the 'Preview' pane on the right. The configuration includes a 'Match Regex' of `(social=)(\d+)` and a 'Replace Expression' of ``${g1}${C.Mask.md5(g2)}``. The 'Apply to Fields' section shows `_raw` selected. The 'Preview' pane shows the output of the pipeline, with the `_raw` field highlighted in blue. The `social` field value has been replaced with its MD5 hash.

```

Masking Rules*
Match Regex (social=)(\d+)
Replace Expression `${g1}${C.Mask.md5(g2)}`
Apply to Fields _raw
Filter true

Preview
4
2020-10-08 19:34:17.154 -07:00
{
  "social": "1602210857.154",
  "timestamp": "2020-10-08 19:34:17.154",
  "event": "Event [EventUpdateBillingProvQuote, timestamp=1581426278, properties={JMSCorrelationID=NA, JMSMessageID=ID:ESP-PD.ET3FF698241D6:05888A6F, orderType=DisconnectLOS, quotePriority=NORMAL, conversationId=ESS-288E18E286A9C9EE:7AC13396:54C71A698BCB:26F9, credits=NA, JMSReplyTo=pub.esb.genericasync.resonse, timeToLive=-1, serviceName=UpdateBillingProvisioning, esn=E2331E893193EB, accountNumber=9900099237, social=1602210857.154, methodName=InternalEvent, AdapterName=UpdateBillingProvQuote, messageId=NA, orderNumber=9900099999922, quoteNumber=79883946, ReplyTo=NA, userName=CherSwag, EventConversationID=NA, mdn=8807877259, accountType=PostPaid, marketCity='COLUMBUS', marketState=OH, marketZip=43223, billingCycle=3, autoBillPayment=T, phoneCode=SGNOTE, phoneType=Android, phoneName='Samsung GALAXY Note', planCode=9900POST60, planType=PostPaid, planPrice=59.99, planName='900 Minute', planDescription='Nationwide 900 Minutes, Unlimited Mobile to Mobile, Unlimited Night & Weekend, Unlimited Text', cardNumber=445279493978320, networkProviderName=Spunktel]"] Show less
  }
}
# _time: 1602210857.154

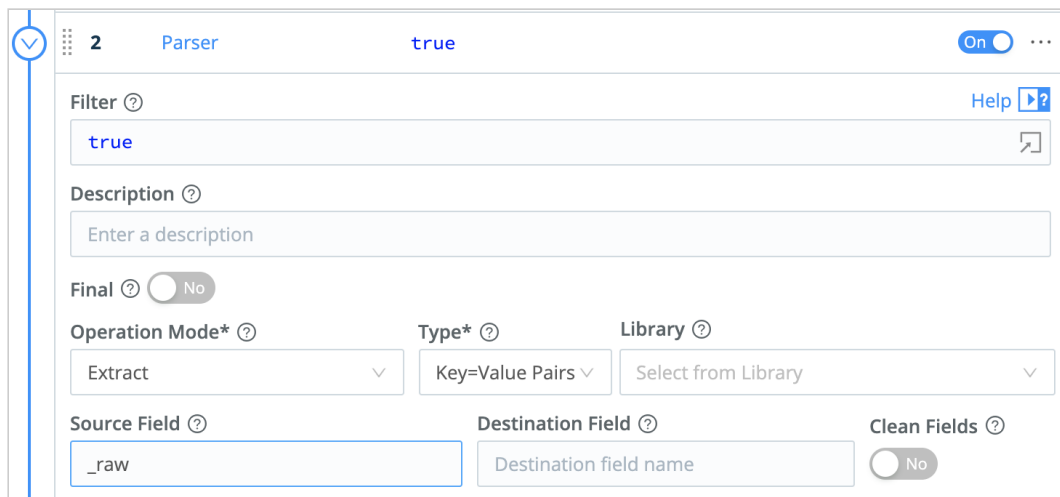
```

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**.



The screenshot shows the configuration for the Parser function. The 'Filter' field is set to 'true'. The 'Description' field is empty with a placeholder 'Enter a description'. The 'Final' toggle is set to 'No'. The 'Operation Mode*' dropdown is set to 'Extract'. The 'Type*' dropdown is set to 'Key=Value Pairs'. The 'Library' dropdown is set to 'Select from Library'. The 'Source Field' dropdown is set to '_raw'. The 'Destination Field' input field is empty with a placeholder 'Destination field name'. The 'Clean Fields' toggle is set to '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 ②

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` .

3 Eval true On ...

Filter ? Help ?

true

Description ?

Add phone co., conditionally based on phone type

Final ? No

Evaluate Fields ?

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

+ Add Field

Adding a field to enrich data

Remove Field (Shrink Data)

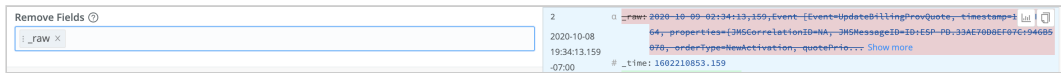
Now that we've parsed out all of the `_raw` field's data – it can go. Deleting a (large) redundant field will give us cleaner events, and reduced load on downstream resources.

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.



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):

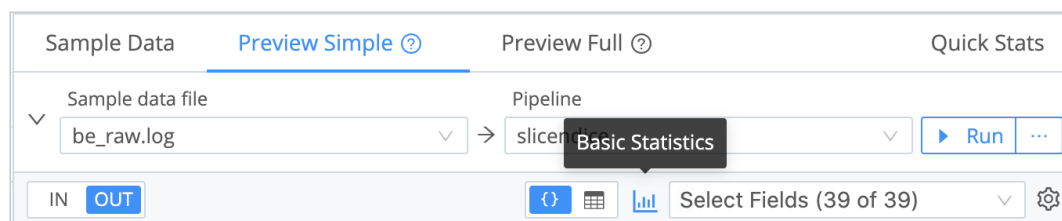
1. In the left **Pipelines** pane, click `+ Function`.
2. Search for `Drop`, then click it.
3. Click into the new Function's **Filter** field.
4. Replace the default `true` value with this JS negation expression:
`accountType != 'PostPaid'`
5. Click **Save**.

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%

Basic Statistics

Select Fields (39 of 39)

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.

The screenshot shows a configuration window for routes. At the top, there's a summary bar for the selected route 'demo', showing it's filtered by 'true', uses the 'slicendice' pipeline, outputs to 'devnull:devnull', and has a 0.000% event rate. Below this is a form with fields for 'Route Name*' (demo), 'Filter' (true), 'Pipeline*' (slicendice), 'Output' (devnull:devnull), and 'Description' (placeholder: Enter a description). A 'Final' toggle is set to 'Yes'. At the bottom, a table lists two routes: 'demo' (rank 1, 0.000% events) and 'default' (rank 2, 100.000% events). A green arrow points to the 'demo' route's left handle, indicating it's being dragged. 'Cancel' and 'Save' buttons are at the bottom right.

Configuring and adding a Route

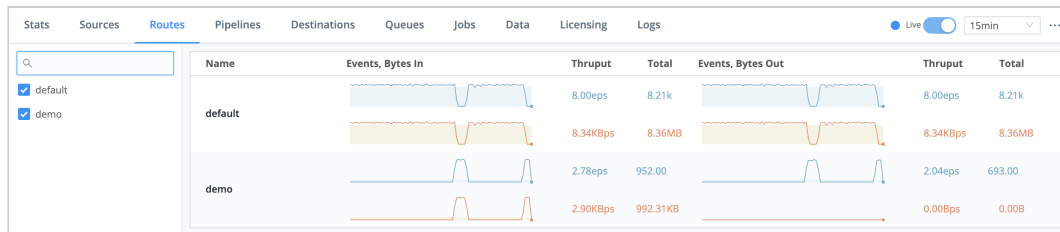
The sparklines should immediately confirm that data is flowing through your new Route:

The screenshot shows a 'Live Routes' table with columns: #, Route, Filter, Pipeline/Output, Events, In/Out status, and Show All. The 'demo' route (rank 1) shows 6.821% events with a small sparkline. The 'default' route (rank 2) shows 93.179% events with a larger sparkline. Both routes have their 'In' status checked. A search bar and '+ Add Route' button are at the top. 'Cancel' and 'Save' buttons are at the bottom right.

#	Route	Filter	Pipeline/Output	Events	In	Out	Show All
1	demo	true	slicendice devnull:devnull	6.821%	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	default	true	main	93.179%	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

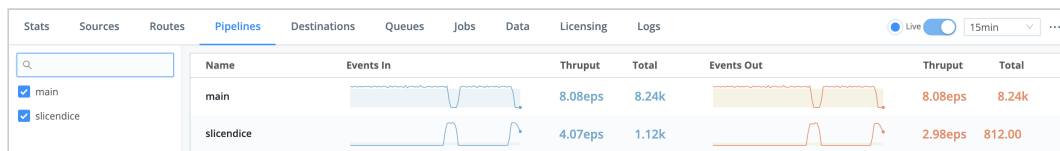
Live Routes

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



Monitoring data flow through Routes

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



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 `busnessevent.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 `busnessevent` 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](#)

Please see details [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.

Shared-Nothing Architecture

All LogStream deployments are based on a shared-nothing architecture pattern, where instances/Nodes and their **Worker Processes operate separately, serving all inputs, outputs, and event processing independently of each other.**

This allows the overall system to continue to operate even if individual Processes or Nodes fail. It also allows individual Nodes to [upgrade](#) without system-wide downtime.

What's Next

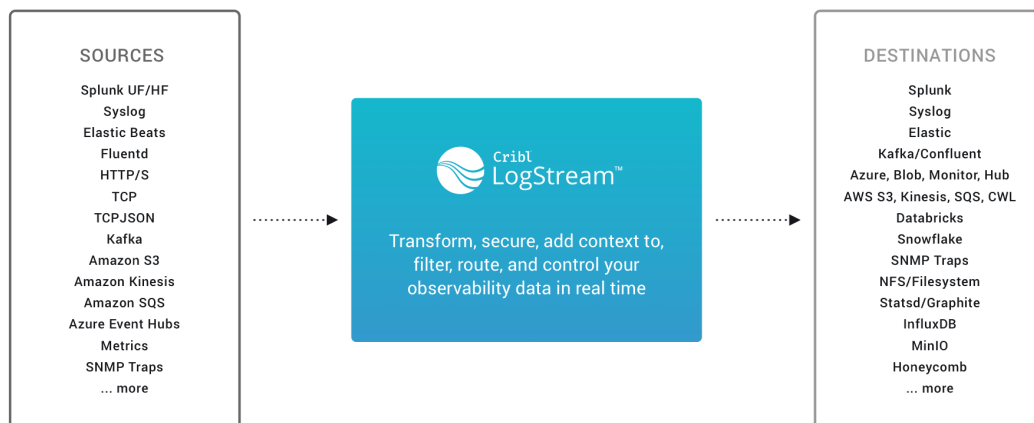
- | |
|---------------------------------|
| ➤ Single-Instance Deployment |
| ➤ Distributed Deployment |
| ➤ Splunk App Deployment * |
| ➤ Bootstrap Workers from Leader |
| ➤ Kubernetes/Helm Deployment |
| ➤ Docker Deployment |

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 (Intel Processors):**
 - 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+
- **OS (ARM64 Processors):**
 - Linux 64-bit
 - Tested so far on Ubuntu (14.04, 16.04, 18.04, and 20.04), CentOS 7.9, and Amazon Linux 2

- **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) on Intel/Xeon or AMD processors; and to 1 (higher-throughput) vCPU on Graviton2/ARM64 processors.

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

All quantities listed above are minimum requirements. To fulfill these requirements using cloud-based virtual machines, see [Recommended AWS, Azure, and GCP Instance Types](#).

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.

Shutdown and Restart Sequence

When a Worker Process receives an explicit shutdown command, it follows this sequence:

1. Shuts down internal system communications: stops receiving any commands from the API Process or [distributed](#) Leader.
2. Shuts down the input Sources.
3. When the input stream ends, receives a signal event from LogStream's event processor to flush out any stateful Pipeline Functions (such as [Aggregations](#), [Sampling](#), [Dynamic Sampling](#), and [Suppress](#)).
4. Waits for 10 seconds, to allow data to finish flowing through the streams processing engine. This wait is designed to allow all Destinations to flush out remaining data. However, any data not flushed within this interval – e.g., because of an error on downstream receivers – will be lost.
5. Exits.

Shutdown/Restart with PQ

Enabling [Persistent Queues](#), on Destinations that support it, generally helps ensure data delivery to your downstream systems. However, note that when a Worker Process restarts, there is a potential for duplicate events to be sent through such Destinations.

This is because PQ doesn't mark events as safe to discard until they've been handed them off to the host OS to send out. So if the Worker Process exits at the final step above **before** all events have flushed, the final handful of events will not have been marked as committed and removed. Upon restart, LogStream will still see them, and will resend them.

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]
```

i 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.

Anti-Virus Exceptions

If you are running anti-virus software on a LogStream instance's host OS, here are general guidelines for minimizing accidental blockage of LogStream's normal operation.

Your overall goals are to prevent the anti-virus software from locking any files while LogStream needs to write to them, and from triggering any changes that LogStream would detect as needing to be committed.

First, if [Persistent Queues](#) are enabled on any Destinations, exclude any directories that these Destinations write to. This is especially relevant if you're writing queues to any custom locations outside of `$CRIBL_HOME`.

Next, for any non-streaming Destinations that you've configured, exclude their staging paths.

Next, exclude these subdirectories of `$CRIBL_HOME`:

- `state/`
- ``log/`
- `.git/` (usually only exists on Leader Nodes)
- `groups/` (on Leader Nodes)
- `local/` (on Workers or Leader)

Finally, avoid scanning any processes. Except for the queueing/staging directories already listed above, LogStream runs everything in memory, so scanning process memory will slow down LogStream's processing and reduce throughput.

Distributed Deployment

Getting started with Cribl LogStream on a 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. 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

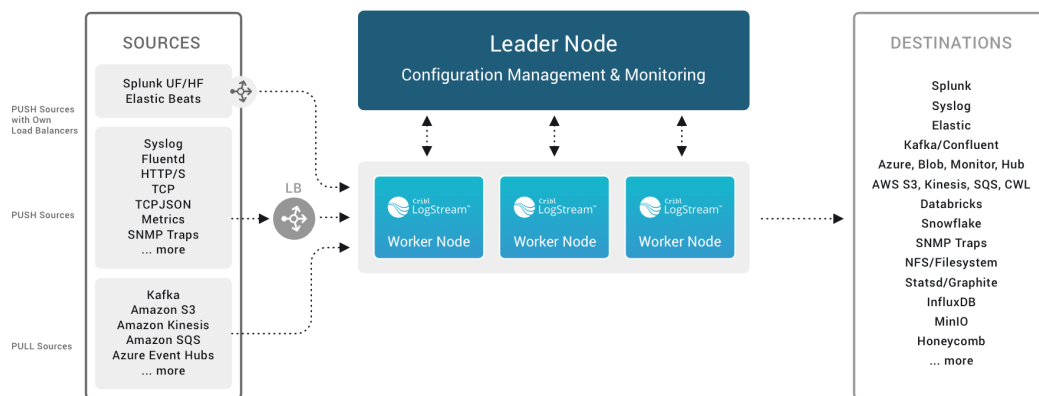
To clarify how the above concepts add up hierarchically, let's use a military metaphor involving toy soldiers:

- 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.
- NConfig 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.
- NWorker 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.
- NWorker 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.

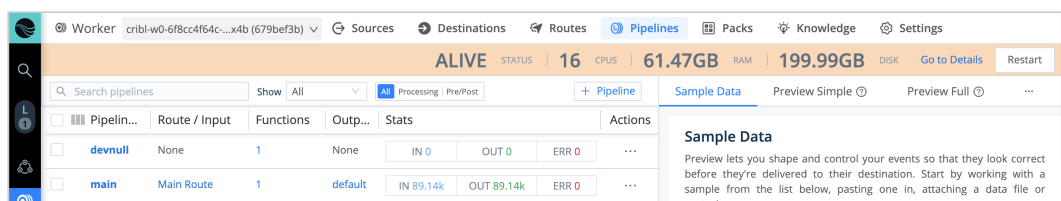
Using the UI

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.



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]
```

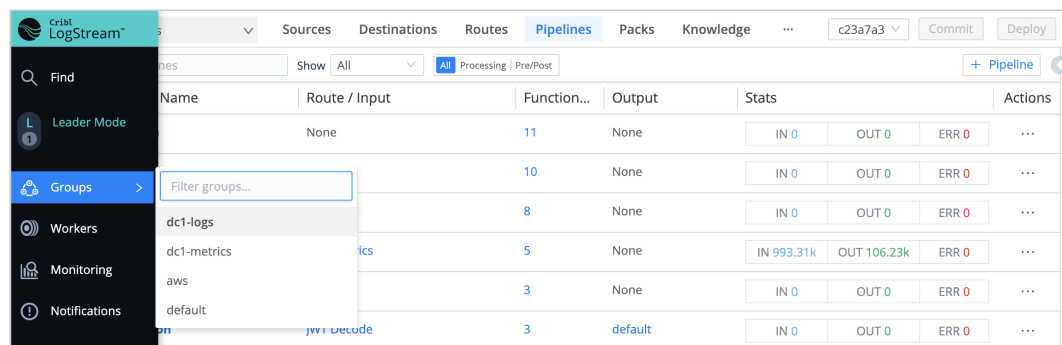
The `-H` and `-p` parameters are required. For other options, see the [CLI Reference](#). Here is an example command:

```
./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**, and **Workers** tabs – all to manage Workers and their assignments. Also, the global **Monitoring** link moves from the top to the left nav.

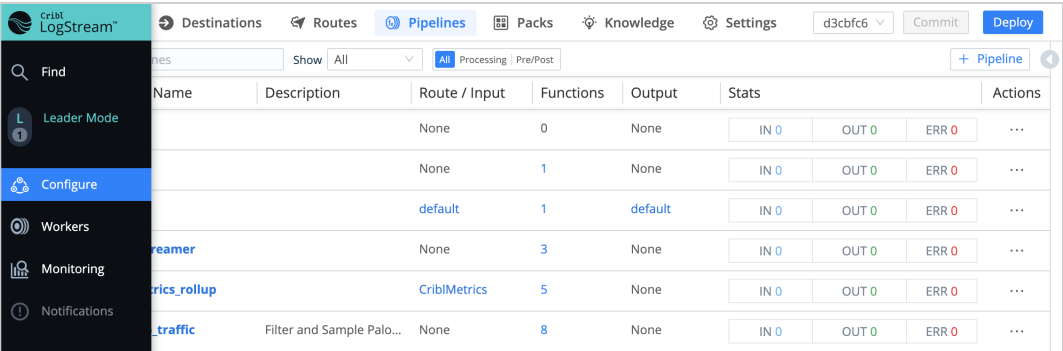


Distributed deployment: menu structure

To access the Group-specific top nav shown above, click **Groups**, then click into your desired Worker Group. This contextual top nav 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.

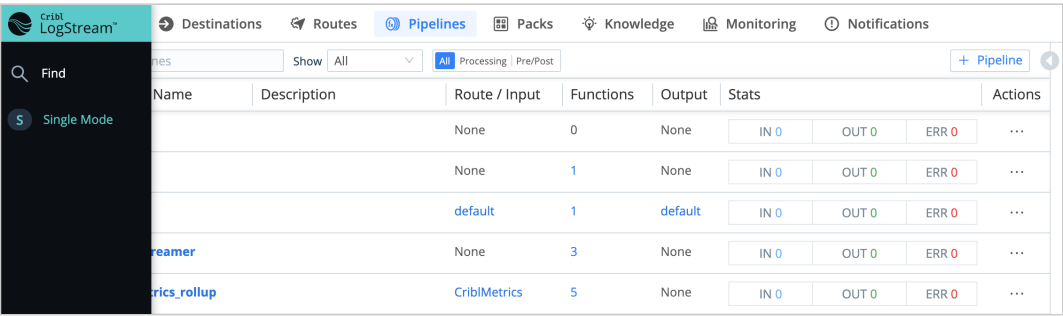


The screenshot shows the Cribl LogStream interface in distributed deployment mode. The left sidebar has a 'Configure' button highlighted. The top navigation bar includes 'Destinations', 'Routes', 'Pipelines', 'Packs', 'Knowledge', and 'Settings'. The 'Pipelines' tab is active, showing a table of pipelines. The table has columns for Name, Description, Route / Input, Functions, Output, Stats, and Actions. The stats column shows IN 0, OUT 0, and ERR 0 for each pipeline.

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	5	None	IN 0 OUT 0 ERR 0	...
traffic	Filter and Sample Palo...	None	8	None	IN 0 OUT 0 ERR 0	...

Distributed deployment with LogStream Free/One license

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



The screenshot shows the Cribl LogStream interface in single-instance deployment mode. The left sidebar has a 'Single Mode' button highlighted. The top navigation bar includes 'Destinations', 'Routes', 'Pipelines', 'Packs', 'Knowledge', 'Monitoring', and 'Notifications'. The 'Pipelines' tab is active, showing a table of pipelines. The table has columns for Name, Description, Route / Input, Functions, Output, Stats, and Actions. The stats column shows IN 0, OUT 0, and ERR 0 for each pipeline.

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	5	None	IN 0 OUT 0 ERR 0	...

Single-instance deployment: anchored top menu

Managing Worker Nodes

If you have an Enterprise or Standard [license](#), clicking the left nav's **Workers** tab opens a **Manage Worker Nodes** page with two upper tabs. The **Workers** tab provides status information for each Worker Node in the selected Worker Group. You can expand each Node's row to display additional details and controls.

Manage Worker Nodes

3

WORKERS

3

GROUPS

3

ALIVE

0

UNHEALTHY

2

SOFTWARE VERSIONS

2

CONF VERSIONS

Workers

Mappings

#

GUID

Host

Status

Group

CPUs

RAM

Last Time

Start Ti...

Config Version

Version

Messages

>

1

c5d35a50-740b-4cf9-aae-

cribl-w0-758d8cd6bc...

alive

dc1-logs

16

61.47GB

2021-08-...

2021-08-...

dc15e8a

42.0-81eb27b9

>

2

0c241493-af82-424d-a00f

cribl-w2-7c894c4d7c-p...

alive

aws

16

61.47GB

2021-08-...

2021-08-...

ae81fb4

42.0-81eb27b9

▼

3

2f38b8d6-a668-49d6-9a2

cribl-w1-79c8f884df-t9...

alive

dc1-m...

16

62.13GB

2021-08-...

2021-08-...

ae81fb4

42.0-20f35f4d

Worker Info

Worker Details

GUID:

2f38b8d6-a668-49d6-9a27-e1798dfc4ff6

Host:

cribl-w1-79c8f884df-t92vt

Status:

alive

Group:

dc1-metrics

CPUs:

16

RAM:

62.13GB

Last Time:

2021-08-05 16:52:55

Start Time:

2021-08-04 23:56:24

Config Version:

ae81fb4

Version:

42.0-20f35f4d

Messages:

Restart

firstMsgTime: 1628146586058

group: dc1-metrics

id: 2f38b8d6-a668-49d6-9a27-e1798dfc4ff6

info: 12 items...

lastMsgTime: 1628207575185

status: healthy

workerProcesses: 14

Workers > Worker Nodes status/controls

Click the **Mappings** tab to display status and controls for the active **Mapping Ruleset**:

Manage Worker Nodes

0

WORKERS

0

GROUPS

0

ALIVE

0

UNHEALTHY

0

SOFTWARE VERSIONS

0

CONF VERSIONS

Workers

Mappings

ID

Rules

Actions

default

4

Active

Configure

Clone

Delete

Workers > Mappings status/controls

Click into a Ruleset to manage and preview its contained Rules:

Managing Ruleset: default

Editing Active Ruleset

+ Rule

Preview Mappings

#

Rule Name

Filter

Group

⌵

All

▼

>

1

On Prem Log Workl...

cribl.tags.incl...

dc1-logs

On

...

>

2

On Prem Metrics W...

cribl.tags.incl...

dc1-metrics

On

...

>

3

AWS Workloads

cribl.tags.incl...

aws

On

...

▼

4

Default Mappings

true

default

On

...

Rule Name

Default Mappings

Filter

true

Group*

default

Cancel

Save

Worker Group

Worker Info

dc1-logs

conn_ip: 10.7.47.6

cpus: 16

cribl: 8 items...

env: 128 items...

freeDiskSpace: 165455745024

groupId: dc1-logs

hostname: cribl-w0-758d8cd6bc-4vgtl

localTime: 1628207616

node: v14.15.1

platform: linux

release: 4.14.209-160.339.amzn2.x86_64

totalDiskSpace: 214735761408

totalMem: 66088072192

aws

conn_ip: 10.7.89.103

cpus: 16

cribl: 8 items...

env: 127 items...

freeDiskSpace: 166814851072

groupId: aws

hostname: cribl-w2-7c894c4d7c-p26cl

localTime: 1628207624

node: v14.15.1

platform: linux

release: 4.14.209-160.339.amzn2.x86_64

totalDiskSpace: 214735761408

totalMem: 66088072192

Managing Ruleset page

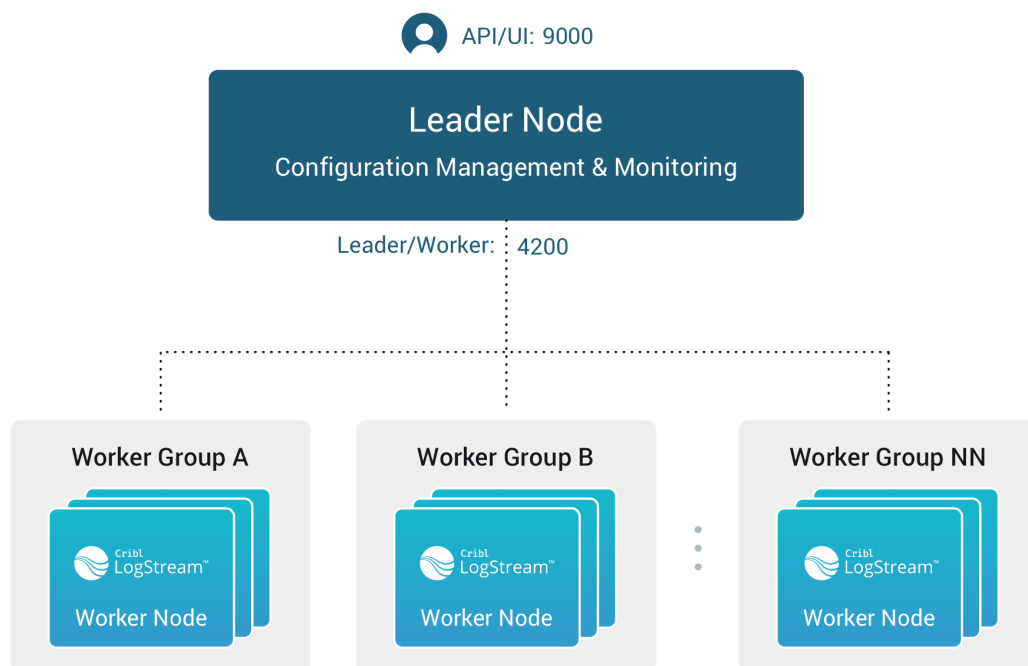
⚠ Distributed mode's repositioning of navigation/menu links also applies to several instructions and screenshots that you'll see throughout this documentation.

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
<code>curl http://<host>:<port>/api/v1/health</code>	<code>{"status": "healthy"}</code>

Environment Variables

- `CRIBL_DIST_MASTER_URL` – URL of the Leader Node.
Format: `<tls|tcp>://<authToken>@host:port?group=defaultGroup&tag=tag1&tag=tag2&tls.<tls-settings below>`.
Example: `CRIBL_DIST_MASTER_URL=tls://<authToken>@leader:4200`
 - `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 were 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

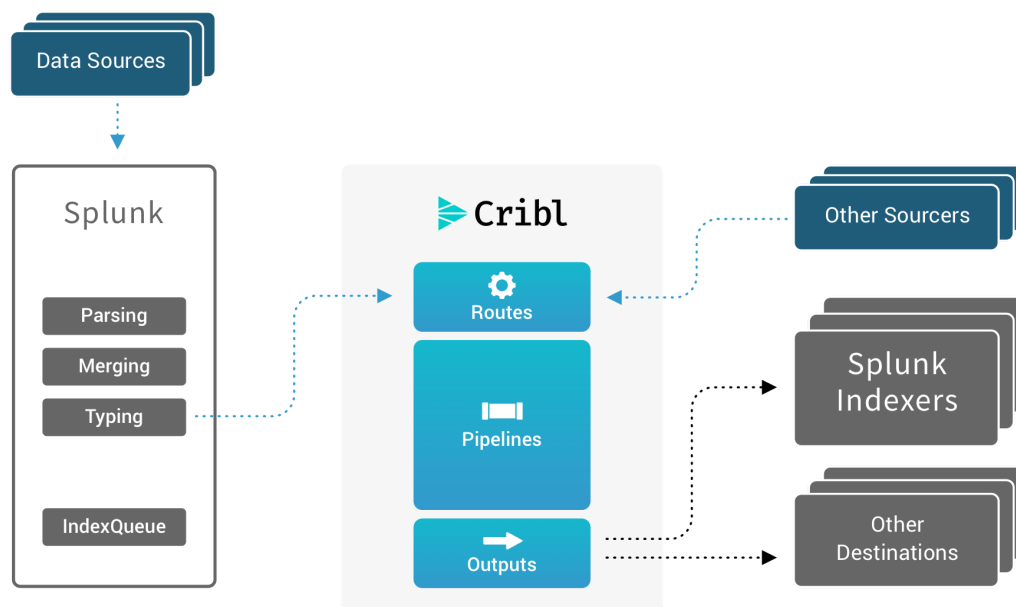
- Select an instance on which to install.
- Ensure that ports 10000 , 10420 , and 9000 are available. See the [Requirements](#) section for more info.
- Get the bits [here](#), and install as a regular Splunk app.
- Restart the Splunk instance.
- Go to `https://<instance>/en-US/app/cribl` or `https://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

Typical Use Cases for Search Head Mode

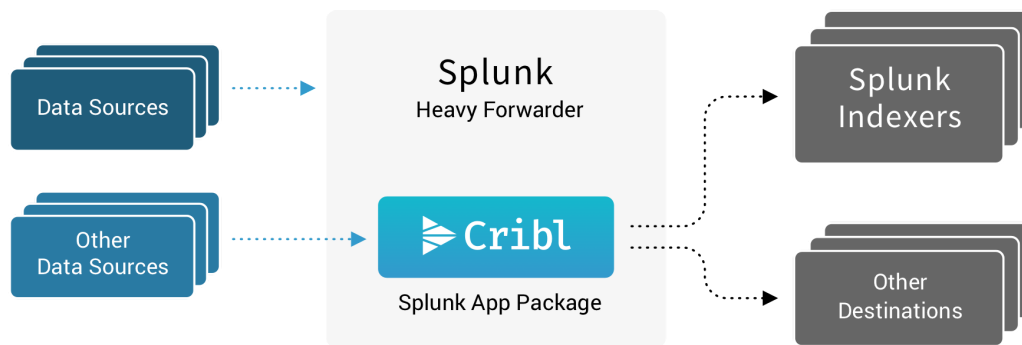
- Working with search results in a Cribl LogStream pipeline.
- Sending search results to any [Destination](#) supported by Cribl LogStream.

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:typin
```

apps/cribl/default/transforms.conf

```
[route2cribl]
SOURCE_KEY = _MetaData:Index
REGEX = ^[^_]
DEST_KEY = _TCP_ROUTING
FORMAT = cribl

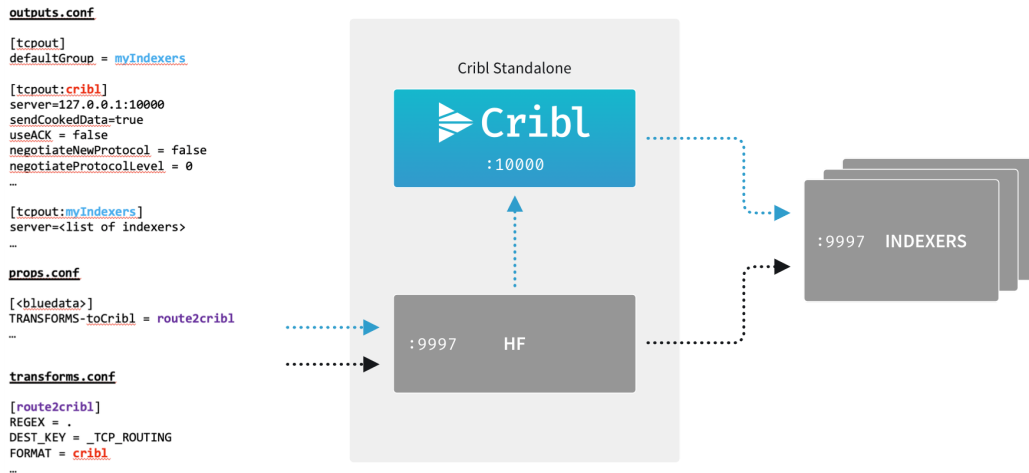
[route2criblQueue]
SOURCE_KEY = _MetaData:Index
REGEX = ^[^_]
DEST_KEY = queue
FORMAT = criblQueue
```

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.



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

This LogStream 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. This fully provisions 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.

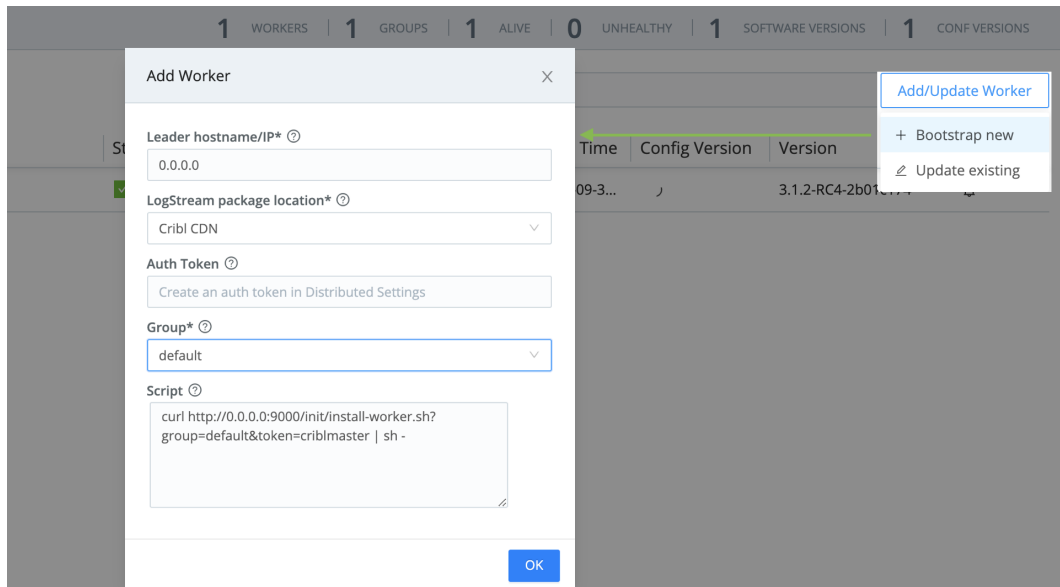
UI Access

In v.3.1.2 and higher, LogStream admins can use the UI to concatenate and copy/paste the bootstrap script, automating several steps below. You can use an adjacent option to grab a script that updates a Worker's Group assignment.

- To use these options, you must have the `admin` [Role](#); you must create a custom Auth token on the Leader; and you must preconfigure the Worker Groups into which you want to add or reassign Workers.

Add/Bootstrap New Worker

1. If necessary, start at global ⚙ **Settings** (lower left) > **Distributed Settings**. Set the Distributed **Mode** to `Leader`. On the **Leader Settings** tab, generate a custom **Auth token**. Click **Save** to restart in Leader mode.
2. From the left nav, select **Workers**.
3. On the resulting **Manage Worker Nodes** page, click + **Add/Update Worker** at the upper right.
4. Select **Bootstrap new** from the menu, as shown in the composite screenshot below.
5. In the resulting **Add Worker** modal, the **LogStream package location** defaults to `Cribl CDN`. If desired, change this to `Download URL`. (For details about this option, see [Adding Download URL](#).)
6. As needed, correct the target **Group**, as well as the **Leader hostname/IP**.
7. As needed (see the note [above](#)), prepend `sudo` to the **Script** field's generated contents.
8. Copy the resulting script to your clipboard.
9. Click either **OK** or **Cancel** to close the modal.
10. Paste the script onto your Leader Node's command line and execute it, to add the Worker.

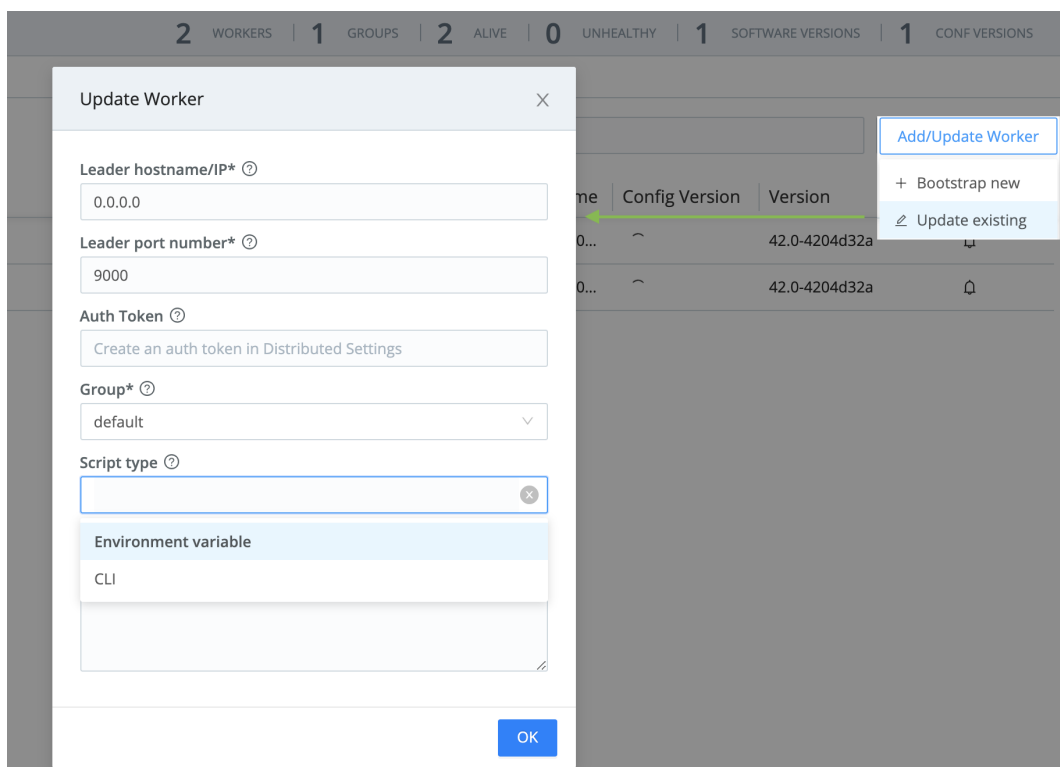


Add/Bootstrap Worker UI option (composite)

Update Existing Worker

You can also auto-generate a script that will update an existing Worker's Group assignment:

1. Follow steps 1–3 in [Add/Bootstrap New Worker](#) above.
2. From the **+ Add/Update Worker** menu, select **Update existing**.
3. In the resulting **Update Worker** modal, select the new target **Group** for this Worker.
4. As needed, correct the **Leader hostname/IP** and **Leader port number**.
5. The **Script type** drop-down defaults to the `Environment variable` command generation option. If you're not relying on environment variables, change this to `CLI`.
6. As needed (see the note [above](#)), prepend `sudo` to the generated **Script** field's contents.
7. Copy the resulting script to your clipboard.
8. Click either **OK** or **Cancel** to close the modal.
9. Paste the script onto your Worker Node's command line and execute it, to update the Worker's assignment.



Update Worker UI option (composite)

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 Worker 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=79364

□ 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
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} |
```

```
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 88 [ $? -eq 127 ] 88 BOOTSTART=0

echo "Checking OS version"
lsb_release -d 2>/dev/null | grep -i ubuntu 88 [ $? -eq 0 ] 88 UBUNTU=1
cat /etc/system-release 2>/dev/null | grep -i amazon 88 [ $? -eq 0 ] 88 AM

echo "Creating cribl user"
if [ $UBUNTU -eq 1 ]; then
    adduser cribl --home /home/cribl --gecos "Cribl LogStream User" --disa
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=7936
```

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

bash

```
curl http://<leader hostname or IP>:9000/init/install-worker.sh?token=7936
```

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=793"
```

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?toke
```

Kubernetes/Helm Deployment

Cribl's leader and workergroup [Helm charts](#) provide a fast way to deploy a distributed LogStream environment to a Kubernetes cluster.

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://cribl.io/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-leader cribl/logstream-leader \
  --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-leader-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-leader-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 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 **Settings > Distributed** page to select **Mode: Leader**.)

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-leader cribl/logstream-leader -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](#)

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.

Deprecation Notice

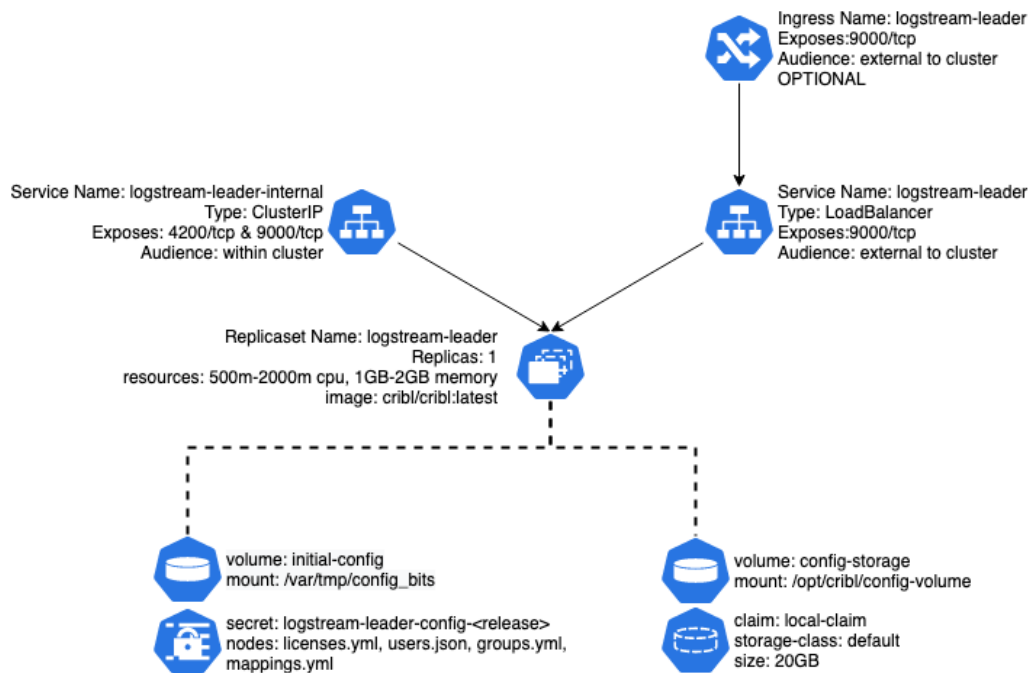
This chart replaces the [logstream-master](#) chart, which was deprecated as of v.2.9.9. See [Migration](#) below for instructions on migrating to this new chart to access features newly provided in this and future versions.

New Capabilities

- Supports LogStream v.3.1.0 (default version).
- Supports the `nodeSelector` configuration option for managing pod scheduling.
- Supports using a fixed IP address for `LoadBalancer`s in both created services, via the `service.internalLoadBalancerIP` and `service.externalLoadBalancerIP` options. (A fixed IP address is not universally supported across K8s implementations; check your implementation before configuring this option.)

Deployment

As built, Cribl's chart will deploy a LogStream Leader 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-leader 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://cribl.io/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/cribl/helm-charts/blob/master/helm-chart-sources/logstream-leader/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-leader`

Key	Type	Default Value	Description
<code>config.adminPassword</code>	String	[No default]	The password you to assign to the admin user.
<code>config.token</code>	String	[No default]	The auth key you to set up for Work access. If you set t

			value, the LogStream instance will be configured only a Leader server for distributed deployment (You can also configure this later via the LogStream UI, after launching the instance single-instance mode).
<code>config.license</code>	String	[No default]	The license for your LogStream instance. If you do not set the license, the default is the Free license. You can change this in the LogStream console as well.
<code>config.groups</code>	List	[No default]	Array of Worker Group names to configure the Leader instance. LogStream will create a mapping for each Group, which is used for the tag <code><groupname></code> , and create the basic structure of each Group's configuration.
<code>config.scName</code>	String	<default StorageClass name>	The StorageClass for all of the persistent volumes.
<code>config.rejectSelfSignedCerts</code>	Number	0	Either 0 (allow self-signed certificates) or 1 (deny self-signed certs).
<code>config.healthPort</code>	Number	9000	The port to use for health checks (readiness/live).
<code>config.healthScheme</code>	String	HTTP	The scheme to use for health checks. Supported values are HTTP or HTTPS.
<code>service.internalType</code>	ClusterIP	[No default]	The type to use for the service. Supported values are <release>-leader or <release>-worker.

			<p>internal service 2.4.5+, this is set to ClusterIP by default. If you have any WorkGroups outside of Kubernetes cluster where the Leader you'll need to change this to NodePort. LoadBalancer to expose it outside cluster.</p>
<p>service . internal LoadBalancerIP</p>	IP address	[No default]	<p>If the service.internal is set to LoadBalancer specifies the IP address to use for the load balancer service interface. Before configuring, check whether your K8s implementation supports fixed IP addresses.</p>
<p>service . externalType</p>	Load Balancer	[No default]	<p>The type to use for user-facing <release>-load service. If ingress.enabled set, this will be forced to NodePort , together with the ingress.</p>
<p>service . external LoadBalancerIP</p>	IP address	[No default]	<p>If the service.external is set to LoadBalancer specifies the IP address to use for the load balancer service interface. Before configuring, check whether your K8s implementation supports fixed IP addresses.</p>

service.ports	Array of Maps	<ul style="list-style-type: none"> - name: api port: 9000 protocol: TCP external: true - name: leadercomm port: 4200 protocol: TCP external: false 	<p>The ports to make available, both in deployment and service. Each "map" in this list needs the following values:</p> <p>name A descriptive name identifying what the port is being used for.</p> <p>port The container port to be made available.</p> <p>protocol The protocol in use for this port (UI or TCP).</p> <p>external Set to <code>true</code> to expose the port on the external service, or <code>false</code> to not expose it.</p>
service.annotations	Object	[No default]	Annotations for the service component. This is where you'll want to put load-balancer specific configuration directives.
criblImage.tag	String	latest	The container image to pull from. Cribl increments this tag with each LogStream version. By default, this will use the version equivalent to the chart's <code>appVersion</code> value. You can override this with <code>latest</code> to get the latest LogStream version, or with a

			LogStream version number (like "3..0
consolidate_volumes	boolean	[No default]	If this value exists the helm command upgrade , this will the split volumes we created in chart before 2.4 and consolidate them to one config volume. This is a one-time
nodeSelector	Object	[No default]	Add nodeSelector values to define the nodes on which pod scheduled. For details and allowed values see K8s' Assigning Pods to Nodes topic.

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.
<code>envValueFrom</code>	Object	[No default]	Environment variables to be exposed from the Downward API.
<code>env</code>	Array	[No default]	Additional static environment variables.
<code>ingress.enable</code>	boolean	false	Enable Ingress in front of the external service. Setting this to <code>true</code> changes the external service to type <code>NodePort</code> , and creates an ingress that connects to it.
<code>ingress.annotations</code>	Object	[No default]	If <code>ingress.enable</code> is set to <code>true</code> , 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:
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-pref
```

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-leader` :

```
helm install logstream-leader cribl/logstream-leader
```

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

```
helm install logstream-leader cribl/logstream-leader --set  
config.scName='ebs-sc'
```

Post-Install/Post-Upgrade

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-leader cribl/logstream-leader --set  
config.license="<long encoded license string redacted>"
```

Run Distributed on a Free License

If you do not specify a license with `config.license`, and you want to run [distributed](#), you'll need to go to 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-leader cribl/logstream-leader --set  
config.adminPassword="<new password>"
```

Set Up Worker Groups/Mappings

As mentioned above, the chart's default is to install a vanilla deployment of 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-leader cribl/logstream-leader --set  
config.groups={group1,group2,group3}
```

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

Migrating from the logstream-master Chart

Here is how to migrate from the deprecated [logstream-master](#) chart to `logstream-leader` .

Exporting your Configuration

You'll need to "export" your data from the existing `logstream-master` pod. And first, you'll need to get the current pod's name, as well as its namespace. The easiest way to do this is to run `kubectl get pods -A` and then look pods that start with the release name you used when you ran `helm install` . For example, if you installed with the following command:

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

...you'd look for a pod name that started with `ls-master` .

Once you've identified your pod and namespace, you can export your configuration using a combination of `kubectl` and `tar` :

```
kubectl exec <pod name> -n <namespace> -- bash -c "cd /opt/cribl/config-vo
```

This command executes the tar based back up of the config-volume, and outputs it to a local tar file (`cribl_backup.tar`).

"Re-Hydrating" the Backup on the logstream-leader Chart

Exploding the tarball onto the new persistent volume is a one-time event. Once the config volume is restored, you'll make changes to the config via the LogStream UI or API. Either approach will change the config on disk, which you wouldn't want to overwrite the next time the pod restarts. You can manually re-hydrate the backup by installing the logstream-leader chart, and then running the following command:

```
cat cribl_backup.tar| kubectl -n <namespace> exec --stdin <pod name> -- ba
```

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

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

After this, you want to **delete** the active pod, allowing the new one to come up with the restored configuration. To do this, you'd run the following `kubectl` command:

```
kubectl -n <namespace> delete <pod name>
```

This will cause the pod to exit, but the deployment will replace it with a new pod which will use the same config persistent volume.

Reconfiguring the Worker Groups

Now that you've got a new working leader chart, you need to tell the workers to connect to the new leader instead of to the old `logstream-master` instance. This is a simple `helm upgrade` operation. You'll need to use the same command string that you used to install, changing the word `install` to `upgrade`. But change the value of `config.host` to the new service that was created for the `logstream-leader` install. (You can change `config.host` either via the `--set` option or in the `values.yml` file.) For example, if you ran the `logstream-leader` install with the release name `ls-lead`, like this:

```
helm install ls-lead -f <values file> cribl/logstream-leader
```

...you'd run `kubectl get service -n <namespace> | grep ls-lead` to get the two services that it created, and you'll want the name of the one that ends in `-internal`. In this case, that name would be `ls-lead-leader-internal`.

Assume that for your workergroup install, you used a release name of `ls-wg1`, and a values file named `my-values.yml` with the following contents:

```
config:
  host: logstream-master-internal
  group: kubernetes
  token: criblmaster
  rejectSelfSignedCerts: 0
```

...then you'd replace the `host` value in this file with `ls-lead-leader-internal`, and then run:

```
helm upgrade ls-wg1 -f my-values.yml -n <namespace>
```

The upgrade **should** replace all the existing workergroup pods with newly reconfigured ones. However, if you notice any workergroup pods with an `AGE` value indicating that it was started before the upgrade command, simply kill those pods, and they will re-spawn with the new configuration.

Preloading Configuration

The `extraConfigmapMounts` and `extraSecretMounts` options enable you to preload configuration files into the leader chart, via ConfigMaps and Secrets that you've created in your Kubernetes environment. However, because ConfigMaps and Secret mounts are read-only, you can't simply mount them into the configuration tree.

Therefore, you must mount them to a location outside of the `/opt/cribl` tree, and then copy the files into the tree at startup. This copying can be accomplished using environment variables, as we'll see [below](#).

- 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_2` 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 ../leader-values.yaml -n logstream-ht cribl/logstre
```

You can now get the pod's name:

```
% kubectl get pods -n logstream-ht
```

NAME	READY	STATUS	RESTARTS
lsms-leader-659bfccdd6-xsz67	1/1	Running	0

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

```
% kubectl describe pod/lsms-leader-659bfccdd6-xsz67 -n logstream-ht | eg
CRIBL_BEFORE_START_CMD_1:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/lic
CRIBL_BEFORE_START_CMD_2:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/map
CRIBL_AFTER_START_CMD_1:       [ ! -f $CRIBL_VOLUME_DIR/users_imported ] &
```

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.yaml` 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}
    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 leader chart into the `logstream` namespace, we'd create it like this:

```
kubectl create configmap job-config --from-file ./config-dir -n
logstream
```

extraConfigmapMounts Config

In our `values.yml` 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-volu
```

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
  cp /var/tmp/job-config/jobs.yml /opt/cribl/config-volume/groups/group1/l
  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;
```

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](#).

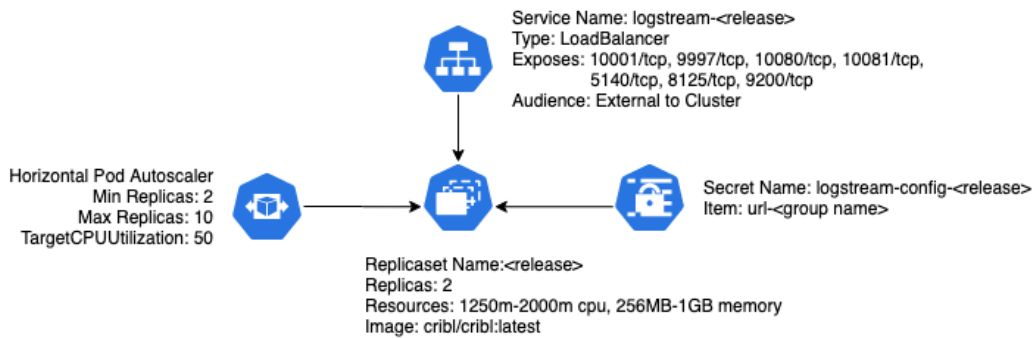
- i**
- This chart will deploy only a LogStream Worker Group, which depends on the presence of a LogStream Leader Node. To deploy the Leader, see [Kubernetes Leader Deployment](#).
-

New Capabilities

- Supports LogStream v.3.1.0 (default version).
 - Supports the `nodeSelector` configuration option for managing pod scheduling.
 - Supports using a fixed IP address for `LoadBalancer`s in the created service, via the `service.loadBalancerIP` option. (A fixed IP address is not universally supported across K8s implementations; check your implementation before configuring this option.)
-

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.



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://cribl.io.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/cribl.io/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 From version 3.0 onward, LogStream's former "master" application components are renamed "leader."

Key	Type	Default Value	Description
<code>config.group</code>	String	<code>criblleader</code>	Tag/group to include in the URL (includec

			as both a group value and a tag value).
<code>config.tag</code>	[Deprecated]	[Deprecated]	The tag/group to include in the URL. (This option is deprecated, but is still supported for backward compatibility)
<code>config.token</code>	String	<code>cribbleader</code>	The authentication token for your LogStream Leader.
<code>config.host</code>	String	<code>logstream-leader</code>	The resolvable hostname of your LogStream Leader.
<code>config.rejectSelfSignedCerts</code>	Number	0	One of: 0 – allow self-signed certs, 1 – deny self-signed certs.
<code>service.type</code>	String	<code>LoadBalancer</code>	The type of service to create for the worker group.
<code>service.loadBalancerIP</code>	IP address	[No default]	If <code>service.type</code> is set to <code>LoadBalancer</code> specifies the IP address to use for the load-balancer service interface. Before configuring, check whether your K8s implementation

			supports fixed IP addresses.
<code>service.ports</code>	Array of Maps	<ul style="list-style-type: none"> - 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 	<p>The ports to make available both in the deployment and in the service. Each "map" in this list needs the following values set:</p> <p>name A descriptive name, identifying what the port is being used for.</p> <p>port The container port to make available.</p> <p>protocol The protocol in use for this port (UDP or TCP).</p>
<code>service.annotations</code>	Object	[No default]	Annotations for the service component – this is where you'll want to put load-balancer-specific configuration directives.

<code>criblImage.tag</code>	String	3.1.0	The container image tag to pull from. Cribl will increment this tag per LogStream version. By default, this value uses a version equivalent to the chart's <code>appVersion</code> value. You can override this with <code>latest</code> to get the latest LogStream version, or with a specific LogStream version number such as <code>3.0</code> .
<code>autoscaling.minReplicas</code>	Number	2	The minimum number of LogStream pods to run.
<code>autoscaling.maxReplicas</code>	Number	10	The maximum number of LogStream pods to scale up to.
<code>autoscaling.targetCPUUtilizationPercentage</code>	Number	50	The CPU utilization percentage that triggers scaling action.
<code>rbac.create</code>	Boolean	false	Enable Service Account, Cluster Role, and Role Binding creation.
<code>rbac.resources</code>	List	<code>["pods"]</code>	Set the resources

			boundary for the role being created (K8s resources).
<code>rbac.verbs</code>	List	<code>["get", "list"]</code>	Set the API verbs allowed the role (default: read ops).
<code>nodeSelector</code>	Object	[No default]	Add nodeSelector values to define the nodes on which pods are scheduled. For details and allowed values see K8s' Assigning Pods to Nodes topic

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.

<code>securityContext.runAsUser</code>	Number	0	User ID to run the container processes under.
<code>securityContext.runAsGroup</code>	Number	0	Group ID to run the container processes under.
<code>envValueFrom</code>	Object	[No default]	Environment variables to be exposed from the Downward API.
<code>env</code>	Array	[No default]	Additional static environment variables.
<code>deployment</code>	String	<code>deployment</code>	One of: <code>deployment</code> to deploy as a Deployment Set; or <code>daemonset</code> to deploy as a DaemonSet.
<code>rbac.extraRules</code>	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 them.
-

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.

(Deprecated:) K8s Master Deployment

Boot a fully provisioned Leader Node via Helm

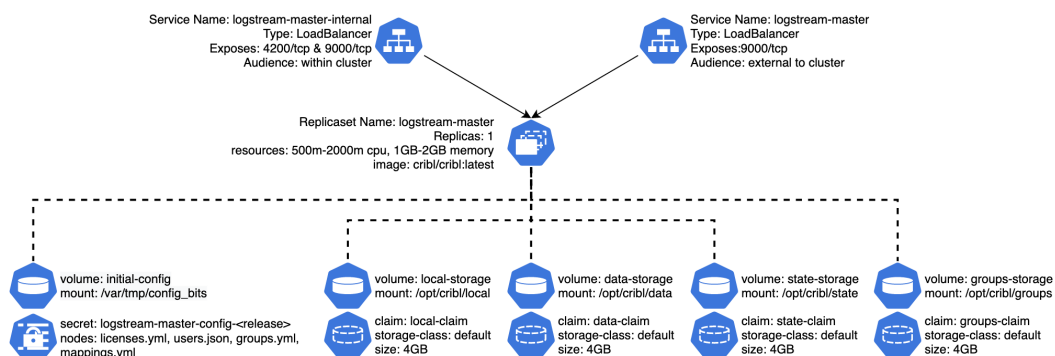
⚠ As of LogStream version 3.0.2, this chart is deprecated. Please instead see the successor [K8s Leader Deployment](#) documentation, which includes instructions for migrating an existing `logstream-master` chart to the new `logstream-leader` configuration.

This document preserves legacy naming, in order to match the legacy chart's configuration.

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

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://cribl.io/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/cribl-io/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
<code>config.adminPassword</code>	String	[No default]	The password you want to assign to the admin user.
<code>config.token</code>	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 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.)
<code>config.license</code>	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.
<code>config.groups</code>	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 a tag <groupname> and will create the basic structure of each Group's configuration.
<code>config.scName</code>	String	<default StorageClass name>	The StorageClass name for all of the persistent volumes.
<code>config.rejectSelfSignedCerts</code>	Number	0	Either 0 (allow self-signed certificates) or 1 (deny self-signed certs).
<code>config.healthPort</code>	Number	9000	The port to use for health checks (readiness/live).
<code>config.healthScheme</code>	String	HTTP	The scheme to use for health checks. Supports HTTP and HTTPS.
<code>service.internalType</code>	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 type to expose it outside the cluster.
<code>service.externalType</code>	Load Balancer	[No default]	The type to use for the user-facing service. If <code>ingress.enabled</code> is set, this will be force-set to NodePort, to work with the ingress.
<code>service.ports</code>	Array of Maps	<ul style="list-style-type: none"> - name: api port: 9000 protocol: TCP external: true - name: mastercomm port: 4200 protocol: TCP external: false 	<p>The ports to make available, both in the deployment and in the service. Each "map" in the list needs the following values set:</p> <p>name A descriptive name, identifying what the port is being used for.</p> <p>port The container port to be made available.</p> <p>protocol The protocol in use for this port (UDP or TCP).</p> <p>external Set to true to expose the port on the external</p>

			service, or false to not expose it.
service.annotations	String	[No default]	Annotations for the service component – th 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 were created in charts before 2.4 and consolidate them down to one container 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 default]	Additional static environment variables.
<code>ingress.enable</code>	boolean	false	Enable Ingress in front of the external service. Setting this to <code>true</code> changes the external service to type <code>NodePort</code> , and creates an ingress that connects to it.
<code>ingress.annotations</code>	Object	[No default]	If <code>ingress.enable</code> is set to <code>true</code> , 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:
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-name
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-pref
```

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. This variable simplifies the chart's persistent-storage requirement, by specifying a path where LogStream should store the persistent data.

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
```

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
```

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/l
```

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 remove 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> -- ba
```

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/conf
```

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_2` 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/logstre
```

You can now get the pod's name:

```
% kubectl get pods -n logstream-ht
```

NAME	READY	STATUS	RESTARTS
lsms-master-659bfccdd6-xsz67	1/1	Running	0

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

```
% kubectl describe pod/lsms-master-659bfccdd6-xsz67 -n logstream-ht | eg
CRIBL_BEFORE_START_CMD_1:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/lic
CRIBL_BEFORE_START_CMD_2:      if [ ! -e $CRIBL_VOLUME_DIR/local/cribl/map
CRIBL_AFTER_START_CMD_1:       [ ! -f $CRIBL_VOLUME_DIR/users_imported ] &
```

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}
      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.yml` 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-volu
```

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
  cp /var/tmp/job-config/jobs.yml /opt/cribl/config-volume/groups/group1/l
  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;
```

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.

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:
      - 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>: <automatically-assigned-host-ports>

With virtual machines, replace `localhost` with the VMs' IP addresses. The automatic assignment of available host-OS ports to the Workers prevents port collisions. **Within** the Docker container, these ports will forward over TCP to port 9000. To see the ports assigned on the OS, enter:

```
docker ps
```

You should see results like these:

CONTAINER ID	IMAGE	COMMAND	CREATED
a3de9ea8f46f	cribl/cribl:latest	"/sbin/entrypoint.sh..."	12 seconds ag
40aa687baefc	cribl/cribl:latest	"/sbin/entrypoint.sh..."	12 seconds ag
df362a65f7d1	cribl/cribl:latest	"/sbin/entrypoint.sh..."	13 seconds ag

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

- Worker1 URL: <http://localhost:63411>
- Worker2 URL: <http://localhost:63410>

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

Cribl.Cloud Deployment

As an alternative to downloading and deploying the LogStream binary to a physical or virtual machine, you can register a hosted LogStream instance on our [Cribl.Cloud portal](#). This launches a SaaS version of LogStream.

About LogStream (and This Page)

If you're new to LogStream, please see our [Basic Concepts](#) page and [Getting Started Guide](#) for orientation. The current page focuses on a Cloud deployment's differences from other deployment options – referred to below as "LogStream binaries" or "customer-managed deployments."

Why Use Cloud Deployment?

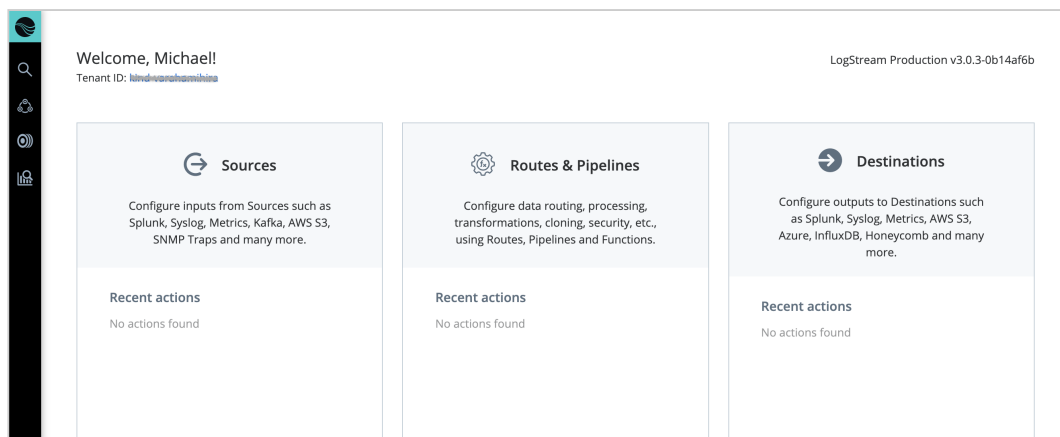
LogStream Cloud is designed to simplify deployment, and to provide certain advantages over using your own infrastructure, in exchange for some current restrictions.

LogStream Cloud Advantages

- Tap LogStream's power, with no responsibility to install or manage software. LogStream Cloud is fully hosted and managed by Cribl, so you can launch a configured instance within minutes.
 - Continuous delivery of upgrades and new features.
 - Free, up to 1 TB/day of data throughput (data ingress + egress) for all new accounts through Jan. 1, 2022.
 - Quickly expand your LogStream Cloud deployment by purchasing metered billing. Pay only for what you use.
-

LogStream Cloud Restrictions

LogStream Cloud provides (by design) a simplified deployment and a simplified user interface.



LogStream Cloud's simplified left nav

These are the current differences – detailed [below](#) – from the LogStream binary deployments that are described in the remainder of this documentation:

- No Worker Groups, and no Worker Mappings controls.
- No Filesystem Source, Collector, or Destination. A Cloud deployment has no local filesystem to read from or write to.
- No Scripts or Script Collector.
- Simplified administration. LogStream Cloud's System Settings exclude Distributed Settings, Git remote repos, LogStream restarts and upgrades (which Cribl handles automatically on your behalf), access management and RBAC Roles (beyond a single, shared admin login), diags uploads, and license uploads.
- No KMS secrets stores (available in LogStream binaries, v.3.0 and above).
- TLS is provided on certain Sources for encryption only, with predefined certificates. LogStream Cloud does not currently support importing your own certificates for TLS mutual authentication.
- Hosted initially on AWS' US West Region, with more options to follow.

Cloud Deployment Quick Start

Ready to take the red pill? This section explains how to register and manage a LogStream Cloud instance.

Registering a Cribl.Cloud Portal

To get started:

1. Start at: <https://cribl.cloud/signup/>
2. Select the **New User? Free signup** option, and register.
3. Follow Cribl's email link to confirm your registration and sign in.
4. Bookmark your Cribl.Cloud portal page, for all that follows.

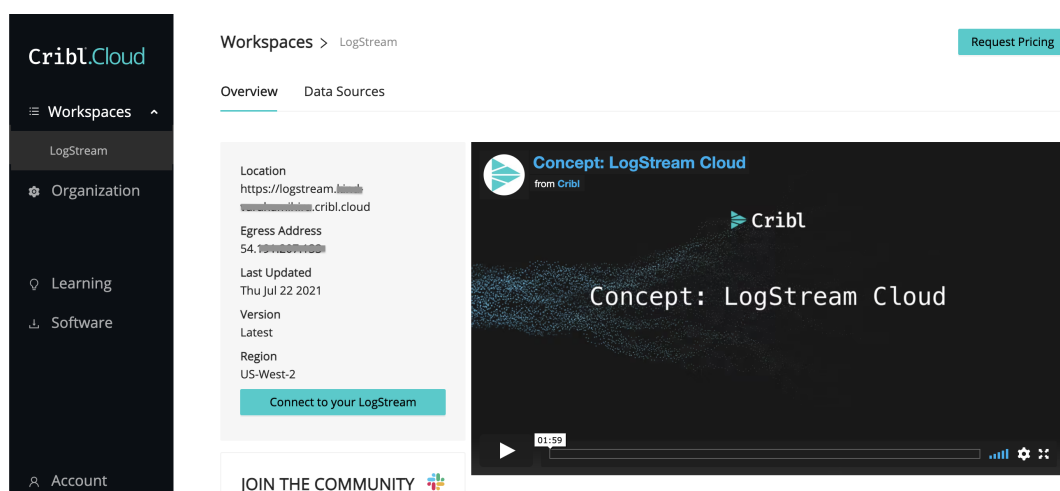
Exploring the Cribl.Cloud Portal

Now that you're here – explore the furniture. The Cribl.Cloud portal's left sidebar allows you to navigate among the following pages/links:

- [Workspaces > LogStream](#)
- [Organization](#)
- [Learning](#)
- [Software](#)
- [Account](#)

Workspaces > LogStream

When you log into the Cribl.Cloud portal, you'll land on this page's **Overview** tab. This is where you'll launch your LogStream instance, and where you'll connect to it on subsequent logins.



Workspaces > LogStream – Overview tab

Overview Tab

The big **Connect to your LogStream** button is the main event here – click it to launch your LogStream instance. However, the surrounding pane displays the following useful information:

Location: Fully-qualified URL at which you access the associated LogStream instance.

Egress Address: The instance's current public IP address. This address is dynamic; Cribl will occasionally update it when we need to rescale core infrastructure.

Last Updated: Date on which Cribl last pushed an infrastructure change (notably including changes to the above **Egress Address**).

Version: Your deployed LogStream version. `Latest` indicates that you're in sync with the most-recent [downloadable](#) LogStream binary.

Region: The AWS Region where your LogStream instance is running.

Data Sources Tab

The same page's **Data Sources** tab lists ports, and data ingestion inputs, that are open and available to use. Return to this tab to copy Ingest Addresses (endpoints) as needed.

Organization

This tab is displayed only to an Organization's owner. Its **Members** tab provides access to [inviting and managing other users](#).

Learning

The **Learning** page links to everything you need to learn about LogStream, to go forth and do great things:

- Sandboxes (free, interactive tutorials on fully hosted integrations).
- Documentation.
- LogStream versions comparison.
- Concept/demo videos.

Software

If you prefer to take the blue pill, this page offers download links for Cribl's LogStream and [AppScope](#) software. You can download either binaries or

Docker containers (hosting Ubuntu 20.04) to install and manage on your own hardware or virtual machines.

Account

This tab offers a **Sign Out** link.

Managing LogStream Cloud

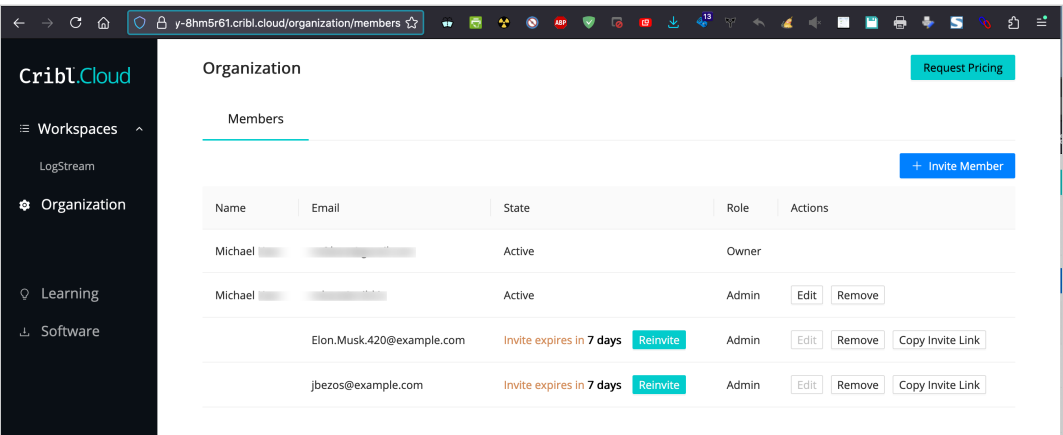
Once you've registered on the portal, here's how to access LogStream Cloud:

1. Sign in to your Cribl.Cloud portal page.
2. From the **Overview** tab, select **Connect to your LogStream**.
3. The LogStream Cloud UI will open in a new tab or window – ready to go!

Note the **Tenant ID** link at the LogStream Cloud home page's upper left, under the **Welcome!** message. You can click this link to reopen the Cribl.Cloud portal page, to access **Data Sources** configurations. To return to this home page from anywhere else in LogStream Cloud's UI, click the LogStream logo in the upper-left corner.

Inviting and Managing Other Users

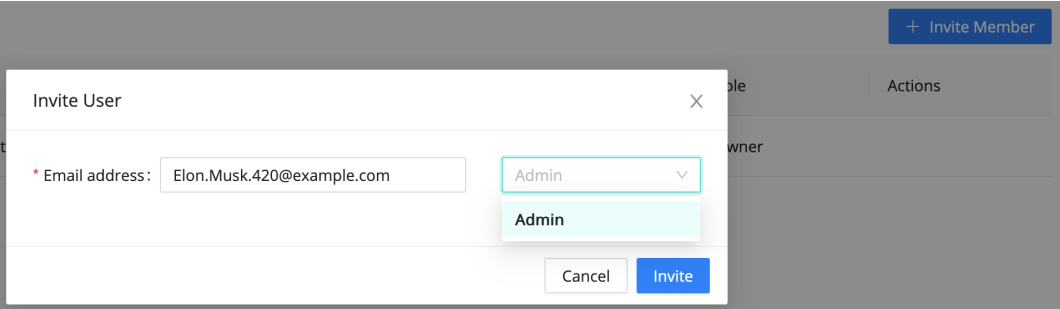
From the **Organization > Members** tab, an Organization's owner can invite new users to join the Organization, assign access roles to new and existing members, and remove pending invites and/or existing members.



Organization > Members tab: Managing Invites and Members

Inviting Members

Click **+ Invite Member** to open the modal shown below. Enter the **Email address** of the new user you want to invite, assign them a **Role** (explained just below), and then click **Invite** to send the invitation.



Invite User modal

Member Roles

The Role that you assign to a member confers permissions on your Cribl.Cloud Organization, as well as a [default Role](#) within the LogStream instance. Currently, you can assign only the **Admin** role.

Member Role	Organization Permissions	LogStream Role
Admin	User	admin
Owner	Owner	admin

You cannot assign or transfer your own **Owner** Role. A user can acquire this superuser Role only by signing up as the owner of their own Cribl.Cloud Organization.

Members whose LogStream Role you assign through the portal's **Organization** tab will **not** be visible as [local users](#) within the LogStream UI.

Responding to Invites

At the address you entered, the new member receives an email with an **Accept Invitation** link to either sign into their existing Cribl.Cloud account, or else sign up to create an account and its credentials.

After signing in, they'll have access to your Organization and LogStream instance at the Role level you've specified.

Managing Invites

While an invite is pending, the **Organization > Members** tab offers you these options to deal with commonly encountered issues:

- **Reinvite:** If your invited member didn't receive your invitation email, you can click this button to resend it.
- **Copy Link:** If emails aren't getting through at all, click this button to copy and share a URL that will take the invitee directly to the signup page. This target page encapsulates the same identity, Organization, and Role you specified in the original email invite.
- **Remove:** This is for scenarios where you need to revoke a pending invite. (You sent someone a duplicate invite, your invitee is spending too much time in space to be a productive collaborator, etc.) After clicking this button, you'll see a confirmation dialog.

After 7 days, if an invite has been neither accepted nor revoked, it expires. In this case, it is removed from the **Members** tab.

Elon.Musk.420@example.com	Invite expires in 7 days	Reinvite	Admin	Edit	Remove	Copy Invite Link
jbezos@example.com	Invite expires in 7 days	Reinvite	Admin	Edit	Remove	Copy Invite Link

Managing Invites

Managing Members

Once a user has accepted an invite, the **Organization > Members** tab offers you this option to modify their membership in your Organization:

- **Remove:** Remove this member from your Organization. After clicking this button, you'll see a confirmation dialog. (Proceeding will not affect this user's access to any other Cribl.Cloud Organizations on which they might be the owner or a member.)

LogStream Cloud Pricing

Beyond the [free tier](#), an optional paid LogStream Cloud account offers support from 8am–5pm, plus the ability to expand to 5 TB/day of throughput. In the Cribl.Cloud portal, select **Request Pricing** to talk with Cribl about upgrading your free account.

You'll pay only for what you use – the data you send to LogStream, and the data sent to external destinations. However, data sent to your AWS S3 storage is always free.

Data Direction	Monthly Charge	Annual Charge
Price per GB sent in to LogStream	\$0.15/GB	\$0.125/GB
Price per GB sent out to external destinations	\$0.15/GB	\$0.125/GB

Example Pricing Scenario

Assume that you want to send 1,000 GB/day to LogStream. You reduce that data by 40% (a standard reduction that we see every day for customers). And you send the remaining 600 GB to an external destination:

$(1,000 \text{ GB/day in} + 600 \text{ GB/day out}) \times \$0.15 \text{ (monthly rate per GB)} = \$240/\text{day}$

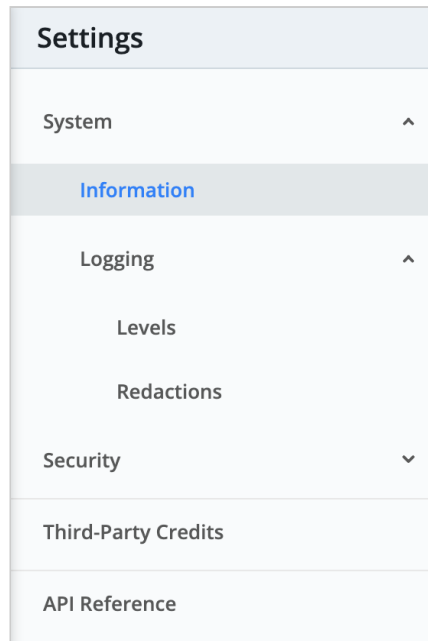
Note that you can send a copy of all of your data to an S3 location of your choice, for no added data-egress charge.

Differences from LogStream Binaries

LogStream Cloud differs from a deployed LogStream binary in the following ways. Keep all these differences in mind as you navigate LogStream's current UI, in-app help (including tooltips), and documentation.

Simplified Administration

Compared to a LogStream binary that you deploy on your own infrastructure, LogStream Cloud's left nav, and particularly its Settings page, are much simpler.



LogStream Cloud's Settings left nav

Here are the key options streamlined out of the Cloud version.

Simplified Distributed Architecture

The **Settings > Worker Processes** and **Settings > Distributed Settings** links are omitted, and the left nav contains no **Worker Groups** or **Mappings** links. LogStream Cloud is configured like a [distributed deployment](#) with a single Worker Group. All Workers will share the same configuration.

Git Preconfigured

The **Settings > Distributed Settings > Git Settings** section is omitted. A local `git` client is preconfigured in your Cribl.Cloud portal. On LogStream Cloud's left nav, use the **Changes** link to commit/push changes to `git`. Select **Deploy** at the UI's top right to deploy your committed changes. LogStream Cloud does not support Git remote repos.

Automatic Restarts and Upgrades

The **Settings > Controls** and **Settings > Upgrade** links are omitted. Cribl handles restarts and version upgrades automatically on your behalf.

Simplified Access Management and Security

There **Settings > Access Management** section is omitted. All users of a given LogStream Cloud instance share a single admin login.

The **Settings > Security** section is omitted. Certificates are predefined for you on the Cribl.Cloud portal's **Data Sources** tab (see [Available Ports and TLS Configurations](#) below). LogStream Cloud does not support KMS secrets stores.

Other Simplified Settings

The **Settings > Licensing** link is omitted. Your license is managed by your parent Cribl.Cloud portal.

The **Settings > Scripts** link is omitted. LogStream Cloud does not support configuring or running shell scripts.

Support Options

The **Settings > Diagnostics** link is omitted. For help with any troubleshooting needs:

- Click the Intercom link at LogStream's lower right.
- Join Cribl's [Community Slack](#) #logstream-cloud channel.
- If you have a [paid account](#), contact Cribl Support.

Available Ports and TLS Configurations

To get data into LogStream Cloud, your Cribl.Cloud portal provides several data sources and ports already enabled for you, plus 10 additional ports (20000 - 20010) that you can use to add and configure more LogStream [Sources](#).

The Cribl.Cloud portal's **Data Sources** tab displays the pre-enabled Sources, their endpoints, reserved and available ports, and protocol details. For the existing Sources listed here, Cribl recommends using the preconfigured endpoint and port to send data into LogStream.

TLS encryption is enabled for you on several Sources, also indicated on the Cribl.Cloud portal's **Data Sources** tab. All TLS is terminated by the Network Load Balancer (NLB) sitting in front of the Workers.

Currently, LogStream Cloud does not enable you to import your own certificates for mutual TLS authentication. LogStream Cloud uses TLS to























provide encryption in the wire, but leaves authentication at the protocol layer – e.g., Splunk HEC or S2S tokens, Kafka authorization, etc.

Overview **Data Sources** Learning Resources

Open Ports Range

20000-20010 ⓘ

Sources Enabled By Default

Name	Type	Ingest Address
in_tcp	TCP	 in.logstream.kind-varahamihira.cribl.cloud:10060 
in_tcp_json	TCP JSON	 in.logstream.kind-varahamihira.cribl.cloud:10070 
in_splunk_tcp	Splunk	 in.logstream.kind-varahamihira.cribl.cloud:9997 
in_splunk_hec	Splunk	 in.logstream.kind-varahamihira.cribl.cloud:8088 
in_elastic	Elasticsearch API	 in.logstream.kind-varahamihira.cribl.cloud:9200 
in_syslog	Syslog	 in.logstream.kind-varahamihira.cribl.cloud:6514 
		 in.logstream.kind-varahamihira.cribl.cloud:9514 
		 in.logstream.kind-varahamihira.cribl.cloud:9514 
http	HTTP	 in.logstream.kind-varahamihira.cribl.cloud:10080 
in_appscope	AppScope	 in.logstream.kind-varahamihira.cribl.cloud:10090 
		 in.logstream.kind-varahamihira.cribl.cloud:10091 

Available ports and TLS certificates

Simplified Source, Collector, and Destination Configuration

LogStream Cloud provides no Filesystem Source, Filesystem Collector, or Filesystem Destination. (A Cloud deployment has no local filesystem to read from or write to.)

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



In a preconfigured Source's configuration, never change the **Address** field, even though the UI shows an editable field. If you change these

fields' value, the Source will not work as expected.

After you create a Source and deploy the changes, it can take a few minutes for the Source to become available in LogStream Cloud's load balancer. However, LogStream will open the port and be able to receive data immediately.

Do Not Enable TLS Within Sources

Several LogStream Cloud Sources' configuration modals include a **TLS Settings (Server Side)** tab, inherited from the LogStream binary's UI. **Do not** set this tab's **Enabled** slider to **Yes**, nor configure any of the resulting fields. Any settings that you configure will conflict with LogStream Cloud Sources' [predefined TLS configurations](#).

Cloud FAQ

Here are some common questions and answers about LogStream Cloud's current state and anticipated evolution.

Are there egress costs to send data to another AWS service in the same Region as LogStream Cloud?

As of August/September 2021, there is a charge, because the data must pass through a public internet interface. Cribl is exploring ways to mitigate this using virtual networks.

Do Worker Nodes provide elastic growth as ingest and egress increase?

As of August/September 2021, it is important to size for anticipated usage. When you anticipate higher throughput, you must communicate this to Cribl, so we can provision additional Workers. Cribl is exploring automated provisioning.

Will LogStream Cloud support a hybrid model with both on-prem and cloud-based Worker Groups?

Later in 2021, Cribl anticipates separating the control plane from the data plane. You will then be able to manage all your LogStream Worker Groups

(cloud and/or on-prem) centrally through the Cloud interface.

How can I send Amazon Kinesis Data Firehose (KDF) data to LogStream Cloud?

1. If you use an AWS load balancer, use only a Classic Load Balancer.
LogStream Cloud currently defaults to using AWS Network Load Balancers (NLBs) in front of each Cloud stack, and KDF is not compatible with NLBs or Application Load Balancers. (For details, see Amazon's [Troubleshooting Amazon Kinesis Data Firehose](#) documentation.)
2. Also, enable duration-based sticky sessions with cookie expiration disabled. (For details, see Amazon's [Duration-Based Session Stickiness](#) documentation.)

As an alternative KDF, you can write to an S3 bucket and use LogStream Cloud's native SQS-based [S3 Source](#). You can also use a Lambda function to define the source data, connecting it to a LogStream Cloud [REST Collector](#) with HTTP discovery.

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** (upper right) > **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 a 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`.

□ You can enter and save `0` or `1`, but LogStream will interpret these entries by attempting to spawn `2` Worker Processes. LogStream will similarly correct for excessive negative offsets by guaranteeing at least `2` Processes.

- **Minimum process count:** Indicates the minimum number of Worker Processes to spawn. Overrides the **Process count**'s effective result, and always enforces at least 2 Processes. (So here again, a 0 or 1 setting is interpreted as 2 Processes.)
- **Memory (MB):** Amount of memory available to each Worker Process, in MB. Defaults to 2048 . (See [Estimating Memory Requirements](#) below.)

i For changes in any of the above controls to take effect, you must click **Save** on the **Manage Processes** page, and then restart the LogStream server via global ⚙ **Settings** (lower left) > **System** > **Controls** > **Restart**. In a distributed deployment, also deploy your changes to the Groups.

Worker Processes' vCPU Requirements

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

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

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 running on Intel or AMD processors 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).

For CPU utilization, see [Capacity and Performance Considerations](#) below.

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

Workers' Independence

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.

For further details about Workers' independence, see [Shared-Nothing Architecture](#).

Capacity and Performance Considerations

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

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

Estimating Core Requirements

Our current guidance for capacity planning depends on the processor type of your bare-metal or VM instance.

Intel/Xeon and AMD Processors

Allocate 1 physical core (2 vCPUs) 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.

Graviton2/ARM64 Processors

Here, 1 physical core = 1 vCPU, but overall throughput is ~20% higher than a corresponding Intel or AMD vCPU. So:

Allocate 1 physical core (1 vCPU) for each 240GB/day of IN+OUT throughput.

To estimate the number of cores needed: Sum your expected input and output volume, then divide by 240GB.

- Example 1: 100GB IN -> 100GB out to each of 3 destinations = 400GB total = 2 physical cores.
- Example 2: 3TB IN -> 1TB out = 4TB total = 17 physical cores.
- Example 3: 4 TB IN -> full 4TB to Destination A, plus 2 TB to Destination B = 10TB total = 42 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.

Measuring CPU Load

You can profile CPU usage on individual Worker Processes.

Single-Instance Deployment

Go to global ⚙ **Settings** (lower left) > **System** > **Worker Processes**, and click **Profile** on the desired row.

ID	Role	PID	Start Time	Options	Restarts	Actions
w:0	LEADER	1400	2021-07-21 17:37:52	--no-warnings --no-deprecation --max-o...	1	Restart Nuke Profile
w:1	WORKER	1443	2021-07-21 17:42:58	--no-warnings --no-deprecation --max-o...	1	Restart Nuke Profile
w:2	WORKER	1413	2021-07-21 17:38:24	--no-warnings --no-deprecation --max-o...	1	Restart Nuke Profile
w:3	WORKER	1428	2021-07-21 17:40:31	--no-warnings --no-deprecation --max-o...	1	Restart Nuke Profile

Worker CPU profiling (single-instance)

Distributed Deployment

This requires a few more steps:

1. Enable [Worker UI Access](#) if you haven't already.
2. Select **Workers** in the left nav.
3. Click on the **GUID** link of the Worker Node you want to profile.
(You will now see that GUID in a **Worker** drop-down at the top left, above an orange header that confirms that you've tunneled through to the Worker Node's UI.)
4. Select **Settings** from that Worker Node's top nav.
5. Select **System** > **Worker Processes** from the resulting side nav.
6. Click **Profile** on the desired Worker Process.

Worker

cribl-w2-7c894c4d... 6bw (3dSadea4)

▼

Sources

Destinations

Routes

Pipelines

Packs

Knowledge

Settings

ALIVE

STATUS

16

CPU

62.13GB

RAM

199.99GB

DISK

Go to Details

Restart

System

Information

General Settings

Worker Processes

Distributed Settings

Logging

Manage Processes

Be careful here!

ID	Role	PID	Start Time	Options	Resta...	Actions
w:0	WORKER	1957	2021-07-21 09:30:11	--no-warnings --no-deprecation --ma...	1	<div>Restart</div> <div>Nuke</div> <div>Profile</div>
w:1	WORKER	1698	2021-07-21 07:25:18	--no-warnings --no-deprecation --ma...	1	<div>Restart</div> <div>Nuke</div> <div>Profile</div>
w:2	WORKER	2238	2021-07-21 11:45:15	--no-warnings --no-deprecation --ma...	1	<div>Restart</div> <div>Nuke</div> <div>Profile</div>

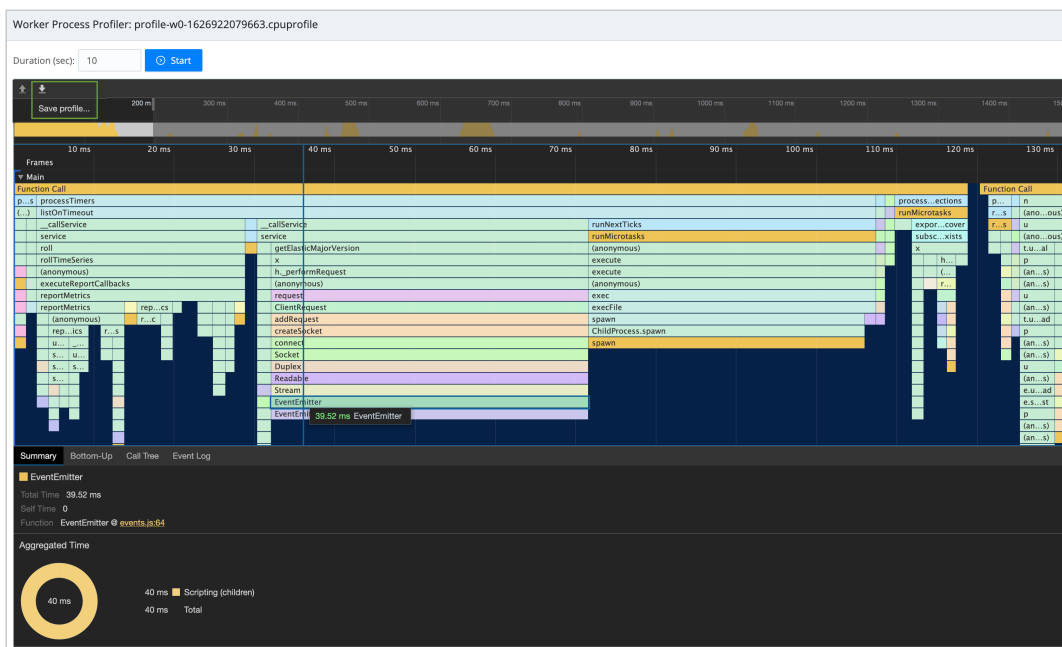
Worker CPU profiling (distributed)

Generating a CPU Profile

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

The default **Duration (sec)** of 10 seconds is typically enough to profile continuous issues, but you might need to adjust this – up to several minutes – to profile intermittent issues. (Longer durations can dramatically increase the lag before LogStream formats and displays the profile data.)

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



Worker CPU profile

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

To save the profile to a JSON file, click the very small tiny minuscule **Save profile (↓)** button we've highlighted at the modal's upper left.

Whether you've saved or not, when you close the modal, you'll be prompted to confirm discarding the in-memory profile data.

i See also: [Diagnosing Issues > Including CPU Profiles](#).

Recommended AWS, Azure, and GCP Instance Types

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

AWS – Intel processors, [Compute Optimized Instances](#). For other options, see [here](#).

Minimum	Recommended
c5d.2xlarge (4 physical cores, 8vCPUs)	c5d.4xlarge or higher (8 physical cores, 16vCPUs)
c5.2xlarge (4 physical cores, 8vCPUs)	c5.4xlarge or higher (8 physical cores, 16vCPUs)

AWS – Graviton2/ARM64 processors, [Compute Optimized Instances](#). For other options, see [here](#).

Minimum	Recommended
c6g.2xlarge (8 physical cores, 8vCPUs)	c6g.4xlarge or higher (16 physical cores, 16vCPUs)
c6gd.2xlarge (8 physical cores, 8vCPUs)	c6gd.4xlarge or higher (16 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 out on the filesystem.

Path Placeholder	Expanded Path
<code>\$CRIBL_HOME</code>	Standalone Install: /path/to/install/cribl/ – referred to below as <code>\$CRIBL_HOME</code> Cribl App for Splunk Install: <code>\$SPLUNK_HOME/etc/apps/cribl/</code>

All paths below are relative to `$CRIBL_HOME` in a [single-instance deployment](#), or to `$CRIBL_HOME/groups/<group-name>/` in a [distributed deployment](#).

Category	Relative Path
Default Configurations Out-of-the-box defaults (rewritable) and libraries (expandable)	default/cribl
Local Configurations User-created integrations and resources	local/cribl
System	(default local)/cribl/cribl.yml

Configuration	See cribl.yml
API Configuration	(default local)/cribl/cribl.yml > [api] section See cribl.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 Configuration	(default local)/cribl/breakers.yml
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 Configuration	(default local)/cribl/functions/<function_name>/... Each function's conf is contained therein.
Roles Configuration	(default local)/cribl/roles.yml RBAC Role definitions. See roles.yml .
Policies Configuration	(default local)/cribl/policies.yml RBAC Policy definitions. See policies.yml .

Configurations and Restart

- Configuration changes resulting from most UI interactions – for instance, changing the order of Functions in a Pipeline, or changing the order of Routes – **do not require restarts**.
- Some configuration changes in the **Settings** UI **do require restarts**. You will be prompted to confirm before restarting.

- All direct edits to configuration files in `(bin|local|default)/cribl/...` **will require restarts**.
- Worker Nodes might temporarily disappear from the Leader's **Workers** tab while restarting.
- When using the [Cribl App for Splunk](#), changes to Splunk configuration files might or might not require restarts. Please check current [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

```
auth: # [object] Authentication Settings
  type: # [string] Type - Select one of the supported authentication provi

# ----- if type is ldap -----

secure: # [boolean] Secure - Enable to use a secure ldap connection (lda
ldapServers: # [array of strings] LDAP servers - List of LDAP servers, e
bindDN: # [string] Bind DN - Distinguished name of entity to authenticat
bindCredentials: # [string] Password - Distinguished Name password used
searchBase: # [string] User search base - Starting point to search LDAP
usernameField: # [string] Username field - LDAP user search field, e.g.
searchFilter: # [string] User search filter - LDAP search filter to appl
groupSearchBase: # [string] Group search base - Starting point to search
groupMemberField: # [string] Group member field - LDAP group search fiel
groupSearchFilter: # [string] Group search filter - LDAP search filter t
groupField: # [string] Group name field - LDAP group field, e.g. cn
connectTimeout: # [number] Connection timeout (ms)
rejectUnauthorized: # [boolean] Reject unauthorized - Valid for secure L
fallback: # [boolean] Fallback on fatal error - Attempt local authentica
groups: # [object]
  default: # [string] Default role - Default role assigned to groups not
  mapping: # [object] Mapping - Group(s) to role(s) mappings

# -----

# ----- if type is splunk -----

host: # [string] Host - Hostname or address of Splunk instance that prov
port: # [number] Port - Port of Splunk instance that provides authentica
ssl: # [boolean] SSL - Whether SSL is enabled on Splunk instance that pr
fallback: # [boolean] Fallback on fatal error - Attempt local authentica
groups: # [object]
  default: # [string] Default role - Default role assigned to groups not
  mapping: # [object] Mapping - Group(s) to role(s) mappings

# -----

# ----- if type is openid -----
```

```

name: # [string] Provider name - The name of the identity provider servi
audience: # [string] Audience - The Audience from provider configuration
client_id: # [string] Client ID - The client_id from provider configurat
client_secret: # [string] Client secret - The client_secret provider con
scope: # [string] Scope - Space separated list of authentication scopes,
auth_url: # [string] Authentication URL - The full path to the provider'
token_url: # [string] Token URL - The full path to the provider's access
userinfo_url: # [string] User Info URL - The full path to the provider's
logout_url: # [string] Logout URL - The full path to the provider's logo
userIdExpr: # [string] User identifier - Expression used to derive userI
rejectUnauthorized: # [boolean] Validate certs - Validate certificates,

```

Example cribl.yml :

```

$CRIBL_HOME/default/cribl/cribl.yml

api:
  host: 0.0.0.0
  port: 9000
  retryCount: 120
  retrySleepSecs: 5
  baseUrl: ""
  # Flag to enable/disable UI. Default: false
  disabled : false
  loginRateLimit: 2/second
  ssl:
    disabled: false
    privKeyPath: /path/to/myKey.pem
    certPath: /path/to/myCert.pem
auth:
  type: local
kms.local:
  type: local
crypto:
  keyPath: $CRIBL_HOME/local/cribl/auth/keys.json
system:
  upgrade: api
  restart: api
  installType: standalone
  intercom: true
workers:
  count: -2
  minimum: 2
  memory: 2048
proxy:
  useEnvVars: true

```

breakers.yml

Cribl's default Event Breaker Library is stored in

`$CRIBL_HOME/default/cribl/breakers.yml` .

`$CRIBL_HOME/default/cribl/breakers.yml`

```
breaker_id: # [object]
  lib: # [string] Library
  description: # [string] Description - Brief description about this rules
  tags: # [string] Tags - One more tags related to this ruleset. Optional.
  rules: # [array] Rules - List of rules. Evaluated in order, top down.
    - name: # [string] Rule Name - Rule Name.
      condition: # [string] Filter Condition - Filter expression (JS) that
      type: # [string] Event Breaker Type - Event Breaker Type
      timestampAnchorRegex: # [string] Timestamp Anchor - Regex to match b
      timestamp: # [object] Timestamp Format - Auto, manual format (strpti
      type: # [string] Timestamp Type
      length: # [number] Length
      format: # [string] Format
      timestampTimezone: # [string] Default Timezone - Timezone to assign
      timestampEarliest: # [string] Earliest timestamp allowed - The earli
      timestampLatest: # [string] Future timestamp allowed - The latest ti
      maxEventBytes: # [number] Max Event Bytes - The maximum number of by
      fields: # [array] Fields - Key value pairs to be added to each event
    - name: # [string] Name - Field Name.
      value: # [string] Value Expression - JavaScript expression to comp
  disabled: # [boolean] Disabled - Allows breaker rule to be enabled o
```

certificates.yml

certificates.yml maintains a list of configured certificates and their parameters.

\$CRIBL_HOME/local/cribl/certificates.yml

```
certificate_id: # [object]
  description: # [string] Description - Brief description of this certific
  cert: # [string] Certificate - Certificate body.
  privKey: # [string] Private key - Certificate private key.
  passphrase: # [string] Passphrase - Passphrase. Optional.
  ca: # [string] CA certificate - Certificate chain. Optional.
  inUse: # [array of strings] Referenced - List of configurations referenc
```

groups.yml

`groups.yml` maintains a list of groups and their configuration versions.

`$CRIBL_HOME/local/cribl/groups.yml`

```
group_id: # [object]
  configVersion: # [string] Config Version - Configuration version that is
```


inputs.yml

inputs.yml contains settings for configuring inputs into Cribl.

\$CRIBL_HOME/default/cribl/inputs.yml

```
inputs: # [object]
  splunk_input: # [object]
    type: # [string] Input Type
    disabled: # [boolean] Disabled - Enable/disable this input
    host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all interfaces)
    port: # [number] [required] Port - Port to listen to.
    tls: # [object] TLS settings (server side)
      disabled: # [boolean] Disabled - Enable TLS

    # ----- if disabled is false -----

    certificateName: # [string] Certificate Name - The name of the prederivative
    privKeyPath: # [string] Private Key Path - Path on server where to find private key
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private key
    certPath: # [string] Certificate Path - Path on server where to find certificate
    caPath: # [string] CA Certificate Path - Path on server where to find CA certificate
    requestCert: # [boolean] Authenticate Client (mutual auth) - Whether to request certificate
    minVersion: # [string] Minimum TLS version - Minimum TLS version to accept
    maxVersion: # [string] Maximum TLS version - Maximum TLS version to accept

    # -----

    ipWhitelistRegex: # [string] IP Allowlist Regex - Regex matching IP addresses
    maxActiveCxn: # [number] Max Active Connections - Maximum number of active connections
    enableProxyHeader: # [boolean] Enable Proxy Protocol - Enable if the client supports it
    metadata: # [array] Fields - Fields to add.
      - name: # [string] Name - Field name
        value: # [string] Value - JavaScript expression to compute field's value
    breakerRulesets: # [array of strings] Event Breaker rulesets - A list of rulesets
    staleChannelFlushMs: # [number] Event Breaker buffer timeout - The amount of time to wait
    authTokens: # [array] Auth tokens - Shared secrets to be provided by any Splunk
      - token: # [string] Token - Shared secrets to be provided by any Splunk
        description: # [string] Description - Optional token description
    pipeline: # [string] Pipeline - Pipeline to process data from this source
  elastic_input: # [object]
    type: # [string] Input Type
    disabled: # [boolean] Disabled - Enable/disable this input
    host: # [string] Address - Address to bind on. Defaults to 0.0.0.0 (all interfaces)
    port: # [number] [required] Port - Port to listen to.
    authTokens: # [array of strings] Auth tokens - Shared secrets to be provided by any Splunk
```

```
tls: # [object] TLS settings (server side)
  disabled: # [boolean] Disabled - Enable TLS

# ----- if disabled is false -----

certificateName: # [string] Certificate Name - The name of the prede
privKeyPath: # [string] Private Key Path - Path on server where to f
passphrase: # [string] Passphrase - Passphrase to use to decrypt pri
certPath: # [string] Certificate Path - Path on server where to find
caPath: # [string] CA Certificate Path - Path on server where to fin
requestCert: # [boolean] Authenticate Client (mutual auth) - Whether
```

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: /.*/
  maxActiveCxn: 0
  authToken: criblmaster
  enabledWorkerRemoteAccess: true
  compression: none
  connectionTimeout: 5000
  writeTimeout: 10000
group: default
envRegex: /^CRIBL_/
tags:
  - tag1
  - tag2
  - tag42
```

- i** As of version 3.0, LogStream's former "master," "whitelist," and "blacklist" entities are respectively renamed "leader," "allowlist," and "blocklist." While some legacy terminology remains within configuration keys/values, this document will reflect that.

jobs.yml

jobs.yml maintains parameters for configured [Collectors](#), corresponding to those listed on the UI's **Manage Collectors** page.

```
$CRIBL_HOME/local/cribl/jobs.yml
```

```
collection_job: # [object]
  collector: # [object]
    type: # [string] Collector type - The type of collector to run.

    # ----- if type is azure_blob -----

    conf: # [object] [required]
      outputName: # [string] Auto-populate from - The name of the predefined
      connectionString: # [string] Connection string - Enter your Azure Co
      containerName: # [string] [required] Container name - Name of the co
      path: # [string] Path - The directory from which to collect data. Te
      extractors: # [array] Path extractors - Allows using template tokens
        - key: # [string] Token - A token from the template path, e.g.: ep
          expression: # [string] Extractor expression - JS expression that
      recurse: # [boolean] Recursive - Whether to recurse through subdirec
      batchSize: # [number] Max Batch Size (objects) - Maximum number o

    # -----

    # ----- if type is filesystem -----

    conf: # [object] [required]
      outputName: # [string] Auto-populate from - Select a predefined conf
      path: # [string] Directory - The directory from which to collect dat
      extractors: # [array] Path extractors - Allows using template tokens
        - key: # [string] Token - A token from the template directory, e.g
          expression: # [string] Extractor expression - JS expression that
      recurse: # [boolean] Recursive - Whether to recurse through subdirec
      batchSize: # [number] Max Batch Size (files) - Maximum number of

    # -----

    # ----- if type is google_cloud_storage -----

    conf: # [object] [required]
      outputName: # [string] Auto-populate from - The name of the predefined
      serviceAccountCredentials: # [string] Service account credentials -
```

```

bucket: # [string] [required] Bucket name - Name of the bucket to co
path: # [string] Path - The directory from which to collect data. Te
extractors: # [array] Path extractors - Allows using template tokens
  - key: # [string] Token - A token from the template path, e.g.: ep
    expression: # [string] Extractor expression - JS expression that
endpoint: # [string] Endpoint - Google Cloud Storage service endpoin
recurse: # [boolean] Recursive - Whether to recurse through subdirec
maxBatchSize: # [number] Max Batch Size (objects) - Maximum number o

# -----

# ----- if type is office365 mgt -----

```

job-limits.yml

job-limits.yml maintains parameters for collection jobs and system tasks. In the UI, you can configure these at global ⚙ **Settings** (lower left) >

General Settings > Job Limits.

\$CRIBL_HOME/default/cribl/job-limits.yml

```
concurrentJobLimit: # [number] Concurrent Job Limit - The total number of
concurrentSystemJobLimit: # [number] Concurrent System Job Limit - The tot
concurrentScheduledJobLimit: # [number] Concurrent Scheduled Job Limit - T
concurrentTaskLimit: # [number] Concurrent Task Limit - The total number o
concurrentSystemTaskLimit: # [number] Concurrent system Task Limit - The n
maxTaskPerc: # [number] Max Task Usage Percentage - Value from 0 to 1 repr
taskPollTimeoutMs: # [number] Task Poll Timeout - The number of millisecon
jobArtifactsReaperPeriod: # [string] Artifact Reaper Period - Time period
finishedJobArtifactsLimit: # [number] Finished Job Artifacts Limit - Maxim
finishedTaskArtifactsLimit: # [number] Finished Task Artifacts Limit - Max
taskManifestFlushPeriodMs: # [number] Manifest Flush Period - The rate at
taskManifestMaxBufferSize: # [number] Manifest Max Buffer Size - The maxim
taskManifestReadBufferSize: # [string] Manifest Reader Buffer Size - The n
schedulingPolicy: # [string] Job Dispatching - The method by which tasks a
jobTimeout: # [string] Job Timeout - Maximum time (assumed in seconds, if
taskHeartbeatPeriod: # [number] Task Heartbeat Period - The heartbeat peri
```

licenses.yml

`licenses.yml` maintains a list of LogStream licenses.

`$CRIBL_HOME/default/cribl/licenses.yml`

`licenses:` # [array of string] - list of licenses

limits.yml

limits.yml maintains parameters for Collector jobs and system tasks. In the UI, you can configure these at global ⚙ **Settings** (lower left) > **General Settings** > **Limits**.

\$CRIBL_HOME/default/cribl/limits.yml

```
samples: # [object] Samples
  maxSize: # [string] Max sample size - Maximum file size for the sample i
minFreeSpace: # [string] Min Free Disk Space - The minimum amount of disk
metricsGCPeriod: # [string] Metrics GC Period - The period at which the sy
metricsMaxCardinality: # [number] Metrics Cardinality Limit - The number o
metricsWorkerIdBlacklist: # [array of strings] Metrics Worker Tracking - L
metricsNeverDropList: # [array of strings] Metrics Never Drop List - List
metricsFieldsBlacklist: # [array of strings] Disable Field Metrics - List
cpuProfileTTL: # [string] CPU Profile TTL - The time to live for collected
```


logger.yml

logger.yml maintains logging levels and redactions, per channel. In the UI, you can configure these at global ⚙ **Settings** (lower left) > **System** > **Logging**.

\$CRIBL_HOME/default/cribl/logger.yml

```
redactFields: # [array of strings] - list of fields to redact
redactLabel: # [string] - redact label
channels: # [object] Logger channels as logger ID/log level pairs. Log lev
  DEFAULT: info
  input:DistMaster: debug
  output:DistWorker: debug
```

mappings.yml

Mapping ruleset configurations are stored in
\$CRIBL_HOME/default/cribl/mappings.yml .

```
$CRIBL_HOME/default/cribl/mappings.yml
```

```
mapping_ruleset_id: # [object]
  conf: # [object]
    functions: # [array] Functions - List of functions to pass data throug
    active: # [boolean]
```

messages.yml

`messages.yml` stores messages displayed in the UI's Messages (LogStream 3.1+) or Notifications (LogStream through 3.0.x) fly-out.

`$CRIBL_HOME/local/cribl/logger.yml`

```
message_id: # [object]
severity: # [string] Severity
title: # [string] Title
text: # [string] Text
time: # [number] Occurrence Time
```

outputs.yml

outputs.yml contains configuration settings for LogStream [Destinations](#).

\$CRIBL_HOME/default/cribl/outputs.yml

```
outputs: # [object]
  default_output: # [object]
    type: # [string] Output Type
    defaultId: # [string,null] Default Output ID - ID of the default output
    pipeline: # [string] Pipeline - Pipeline to process data before sending
    systemFields: # [array of strings] System fields - Set of fields to avoid
    devnull_output: # [object]
      type: # [string] Output Type
      pipeline: # [string] Pipeline - Pipeline to process data before sending
      systemFields: # [array of strings] System fields - Set of fields to avoid
    tcpjson_output: # [object]
      type: # [string] Output Type
      host: # [string] Address - The hostname of the receiver
      port: # [number] [required] Port - The port to connect to on the provider
      authToken: # [string] Auth token - Optional authentication token to use in
      compression: # [string] Compression - Codec to use to compress the data
      throttleRatePerSec: # [string] Throttling - Rate (in bytes per second)
      tls: # [object] TLS settings (client side)
        disabled: # [boolean] Disabled - Enable TLS

    # ----- if disabled is false -----

    rejectUnauthorized: # [boolean] Validate server certs - Reject certs
    servername: # [string] Server name (SNI) - Server name for the SNI (
    certificateName: # [string] Certificate name - The name of the prede
    caPath: # [string] CA certificate path - Path on client in which to
    privKeyPath: # [string] Private key path (mutual auth) - Path on client
    certPath: # [string] Certificate path (mutual auth) - Path on client
    passphrase: # [string] Passphrase - Passphrase to use to decrypt private
    minVersion: # [string] Minimum TLS version - Minimum TLS version to
    maxVersion: # [string] Maximum TLS version - Maximum TLS version to

    # -----

    connectionTimeout: # [number] Connection Timeout - Amount of time (milliseconds)
    writeTimeout: # [number] Write Timeout - Amount of time (milliseconds)
    onBackpressure: # [string] Backpressure behavior - Whether to block, drop

    # ----- if onBackpressure is queue -----
```

```
pqMaxFileSize: # [string] Max File Size - The maximum size to store in
pqMaxSize: # [string] Max Queue Size - The maximum size amount of disk
pqPath: # [string] Queue File Path - The location for the persistent q
pqCompress: # [string] Compression - Codec to use to compress the pers

# -----

pipeline: # [string] Pipeline - Pipeline to process data before sendin
systemFields: # [array of strings] System fields - Set of fields to au
syslog_output: # [object]
type: # [string] Output Type
```

parsers.yml

`parsers.yml` stores configuration data for the [Knowledge > Parsers Library](#).

`$CRIBL_HOME/default/cribl/parsers.yml`

```
parser_id: # [object]
  lib: # [string] Library
  description: # [string] Description - Brief description of this parser.
  tags: # [string] Tags - One more tags related to this parser. Optional.
  type: # [string] Type - Parser/Formatter type to use.
  fields: # [array of strings] List of Fields - Fields expected to be extr
```

policies.yml

policies.yml contains RBAC [Policy](#) definitions. Default example:

```
$CRIBL_HOME/default/cribl/policies.yml
```

```
GroupFull:
  args:
    - groupName
  template:
    - PATCH /master/groups/${groupName}/deploy
    - GroupEdit ${groupName}
GroupEdit:
  args:
    - groupName
  template:
    - '* /m/${groupName}'
    - '* /m/${groupName}/*'
    - GroupRead ${groupName}
GroupCollect:
  args:
    - groupName
  template:
    - POST /m/${groupName}/lib/jobs
    - PATCH /m/${groupName}/lib/jobs/*
    - POST /m/${groupName}/jobs
    - PATCH /m/${groupName}/jobs/*
    - GroupRead ${groupName}
GroupRead:
  args:
    - groupName
  template:
    - GET /m/${groupName}
    - GET /m/${groupName}/*
    - POST /m/${groupName}/preview
    - POST /m/${groupName}/system/capture
    - POST /m/${groupName}/lib/expression
    - GET /master/groups/${groupName}
    - GET /master/workers
    - GET /master/workers/*
    - '* /w/*'
    - GET /master/groups
    - GET /system/info
    - GET /system/info/*
    - GET /system/logs
    - GET /system/logs/group/${groupName}/*
```

- GET /system/settings
- GET /system/settings/*
- GET /system/instance/distributed
- GET /system/instance/distributed/*
- GET /version
- GET /version/*
- GET /version/info
- GET /version/info/*
- GET /version/status
- GET /version/status/*
- GET /mappings

regexes.yml

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

`$CRIBL_HOME/default/cribl/regexes.yml`

```
regex_id: # [object]
  lib: # [string] Library
  description: # [string] Description - Brief description about this regex
  regex: # [string] Regex pattern - Regex pattern. Required.
  sampleData: # [string] Sample data - Sample data for this regex. Optional
  tags: # [string] Tags - One more tags related to this regex. Optional.
```

roles.yml

roles.yml contains RBAC [Role](#) definitions. Default example:

```
$CRIBL_HOME/default/cribl/roles.yml
```

```
admin:
  description: 'Members with admin role have permission to do anything and
  policy:
    - '*' *'
reader_all:
  description: 'Members with reader_all role get read-only access to all W
  policy:
    - GroupRead *
collect_all:
  description: 'Members of this group can run existing collection jobs of
  policy:
    - GroupCollect *
editor_all:
  description: 'Members with editor_all role get read/write access to all
  policy:
    - GroupEdit *
owner_all:
  description: 'Members with owner_all role get read/write access as well
  policy:
    - GroupFull *
user:
  description: 'The base user role allows users to see the system info alo
  policy:
    - GET /system/info
    - GET /system/info/*
    - GET /system/users
    - GET /system/instance/distributed
    - GET /system/instance/distributed/*
    - GET /clui
    - PATCH /ui/*
```

samples.yml

`samples.yml` contains metadata about about stored sample data files (size, number of events, date created, name, etc.). The corresponding sample files reside in `$CRIBL_HOME/data/samples`.

`$CRIBL_HOME/local/cribl/samples.yml`

```
sample_id: # [object]
  sampleName: # [string] File Name - Filename to save the sample as. Requi
  pipelineId: # [string] Associate with Pipeline - Select a pipeline to as
  description: # [string] Description - Brief description about this sampl
  ttl: # [number] Expiration (hours) - Time to live for the sample, the TT
  tags: # [string] Tags - One more tags related to this sample file. Optio
```

schemas.yml

schemas.yml stores configuration data for the [Knowledge > Schema Library](#).

\$CRIBL_HOME/default/cribl/schemas.yml

```
schema_id: # [object]
  description: # [string] Description - Brief description about this schem
  schema: # [string] Schema - JSON schema matching standards of draft vers
```

scripts.yml

scripts.yml stores configuration data for scripts configured at global

⚙ **Settings** (lower left) > **Scripts**:

\$CRIBL_HOME/local/cribl/scripts.yml

```
script_id: # [object]
  command: # [string] Command - Command to execute for this script.
  description: # [string] Description - Brief description about this scrip
  args: # [array of strings] Arguments - Arguments to pass when executing
  env: # [object] Env Variables - Extra environment variables to set when
```

vars.yml

`vars.yml` stores configuration data for the [Knowledge > Global Variables Library](#).

`$CRIBL_HOME/default/cribl/vars.yml`

```
variable_id: # [object]
  lib: # [string] Library
  description: # [string] Description - Brief description of this variable
  type: # [string] Type - Type of variable.
  value: # [string] Value - Value of variable.
  tags: # [string] Tags - One more tags related to this variable. Optional
```

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

LogStream 2.3.x or Later

On LogStream 2.3 or later, you need to configure licensing only on the Leader Node. (See [Managing Licenses](#).) The Leader will push license information down to Worker Groups as part of the [heartbeat](#).

 LogStream 2.3 changed licensing in other ways that might require you to update an existing LogStream 2.2 (or earlier) configuration. In this scenario, please see [Upgrading to LogStream 2.3](#).

LogStream 2.2.x or Earlier

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

- To configure the Master: **Settings > Licensing**.
- To configure Worker Groups: **Worker Groups > [Select a Group] > System Settings > Licensing**.

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 from all LogStream nodes, 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!
```

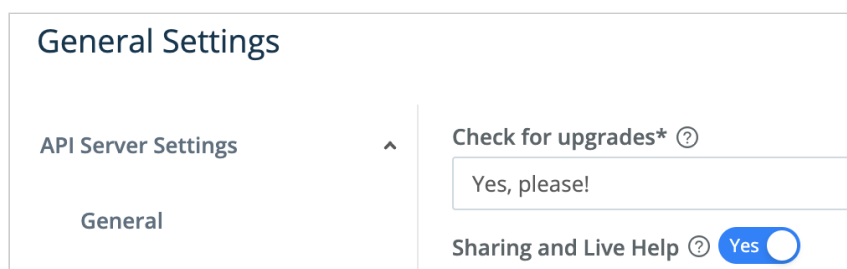
If you get a 302 response code, check whether you've omitted the URL's trailing `/`.

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**.



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 and Out, overall and by Source type and Destination type
- Number of Bytes In and Out, overall and by Source type and Destination type
- Number of Open, Closed, Active Connections
- Number of Routes
- Number of Pipelines

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":"Yrt0MOD1w80zyMYB8WMcEle0tYESMwZw2qIZyTvue0E"}
```

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":"false", "passwd":"Yrt0MOD1w80zyMYB8WMcEle0tYESMwZw2qIZyTvueOE"}
```

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

```
{"username":"user","first":"Elvis","last":"Bath","disabled":"false", "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 that is different from the admin user's password on the Leader Node. This is by design, as a security precaution. But it might lead to situations where administrators cannot log into a Worker Node directly, and must rely on accessing them via the Leader.

To explicitly apply a known/new password to your Worker Node, you set and push a new password to the Worker Group. Here's how, 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 any new password settings will be included.

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` . If you populate the **User info URL** field, you must add `groups` to this list.
- **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 in the **User info URL** response (if configured); otherwise, `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.
- **User info URL** is required, because Okta doesn't encode groups in `id_token`. Azure AD and Google also rely on this field.

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					<input type="text"/>	+ Add New
<input type="checkbox"/>	Username	First Name	Last Name	Email	Roles	
<input type="checkbox"/>	admin	admin	admin	admin	admin	

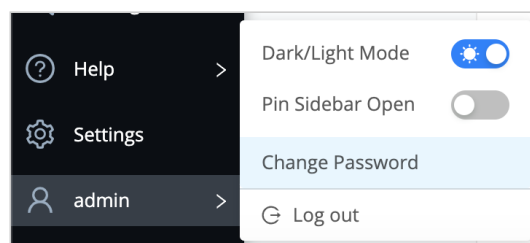
Managing users

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 .

Entering and saving a user's credentials

In LogStream 3.1 and above, logged-in users can change their own LogStream passwords via the User Settings fly-out at the UI's lower left. This fly-out also provides user-specific options to customize the UI.



Self-serve password changes

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.

i 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.
collect_all	Ability to run existing collection jobs on all Worker Groups.
notification_admin	Read/write access to all Notifications .
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			<input type="text"/>	+ Add New
<input type="checkbox"/>	Name	Description	Attached policies	
<input type="checkbox"/>	admin	Members with admin role have permission to do anything and everything I...	* *	
<input type="checkbox"/>	reader_all	Members with reader_all role get read-only access to all Worker Groups.	GroupRead *	
<input type="checkbox"/>	editor_all	Members with editor_all role get read/write access to all Worker Groups.	GroupEdit *	
<input type="checkbox"/>	owner_all	Members with owner_all role get read/write access as well as Deploy perm...	GroupFull *	
<input type="checkbox"/>	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 Policy

Delete Role

Clone Role

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.)

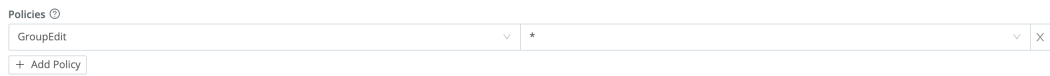
Clone Role: Opens a **New role** version of the modal, duplicating the **Description** and **Policies** of the Role you started with.

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 <code>GroupRead</code> , grants the ability to also change and commit a Worker Group's configuration.
GroupFull	Building on <code>GroupEdit</code> , grants the ability to also deploy a Worker Group.
GroupCollect	Grants the ability to run Collectors on 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

+ Add Policy

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:

Connection Timeout (ms)	admin	
Reject Unauthorized ⓘ	editor_all	
Group Name Field ⓘ	owner_all	
	reader_all	
ROLE MAPPING	user	
Default role ⓘ		
Mapping ⓘ	Enter external group name	Select role(s) to map to
	+ Add Mapping	

Cancel Save

Mapping external user groups to LogStream Roles

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

Reject Unauthorized ⓘ	<input type="radio"/> No	
Group Name Field ⓘ	cn	
ROLE MAPPING		
Default role ⓘ	user	
Mapping ⓘ	devops	owner_all x
	+ Add Mapping	

Cancel Save

External user groups mapped to LogStream Roles

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 Certificate 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.

General Settings

API Server Settings

General

SSL

Advanced

Authentication Settings

Default TLS Settings

Display Settings

Limits

Job Limits

Proxy Settings

Upgrade & Share Settings

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-Options	SAMEORIGIN	X
Referrer-Policy	no-referrer	X

+ Add Header

Prev

Next

Cancel

Save

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	CI
UI	Browser	Cribl LogStream	Default disabled	Default disabled	De di
API	Worker	Master	Default disabled	Default disabled	De di
Worker-to-Master	Worker	Master	Default disabled	Default disabled	De di
Data	Any data	Cribl	Default	Default	De

	sender	LogStream (Source)	disabled	disabled	di
Data	Cribl LogStream (Destination)	Any data receiver	Default disabled	Default disabled	Di di
Authentication	-----	-----	-----	-----	—
• Local	Browser	Cribl LogStream	Default Disabled	N/A	N,
• LDAP	Cribl LogStream	LDAP Provider	Custom	N/A	De Di
• Splunk	Cribl LogStream	Splunk Search Head	Default Enabled	N/A	De 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**.

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 LogStream's secrets store, you can centrally manage secrets that LogStream instances use to authenticate on integrated services. Use this UI section to create and update authorization tokens, username/password combinations, and API-key/secret-key combinations for reuse across the application.

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 on each 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 the following types:

Text: This default type exposes a **Value** field where you directly enter the secret. Text secrets can currently be used only with LogStream's Google Cloud Pub/Sub [Source](#) and [Destination](#), where they are the only supported secret type.

API key and secret key: Exposes **API key** and **Secret key** fields, used to retrieve the secret from a secure endpoint. This is the only secret type supported on LogStream's AWS-based Sources, Collectors, and Destinations, and on our Google Cloud Storage Destination.

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 the [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 the [nodejs docs](#).

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** To integrate an external KMS provider into a distributed deployment, LogStream's Leader Node must have Internet access.

When you initially install a license in distributed mode, a known bug prevents immediate use of KMS features within Worker Groups. Here is the workaround:

1. Open 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:

KMS Settings

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 AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.
- **Manual:** If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials directly or by reference. This is

useful for Workers not in an AWS VPC, e.g., those running a private cloud. It prompts you to provide an **Access key** and a **Secret key**.

Vault AWS IAM Server ID: Value to use for the `Vault-AWS-IAM-Server-ID` header value. This should match the value configured with IAM authentication on Vault.

Vault Role: Authentication role to use in Vault.

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.

Advanced

Enable health check: Whether to perform a health check before migrating secrets data. Defaults to `Yes`.

Health check endpoint: Configurable endpoint to use for validating system health. Defaults to `/v1/sys/health`.

What's Next

➤ [Securing Data](#)

➤ [Access Management](#)

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

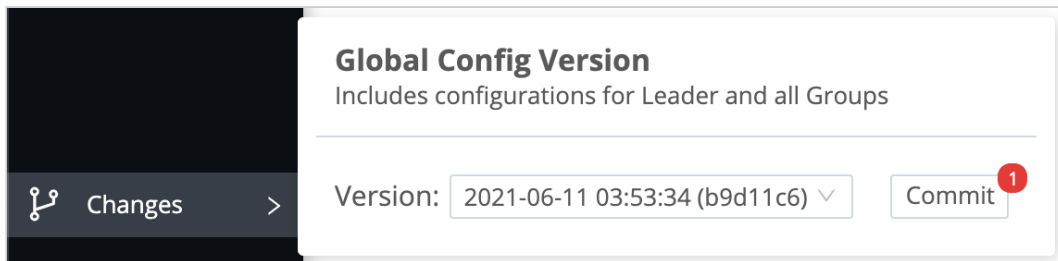
For [distributed deployments](#), `git` **must** be installed and available locally on the host running the Leader Node.

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

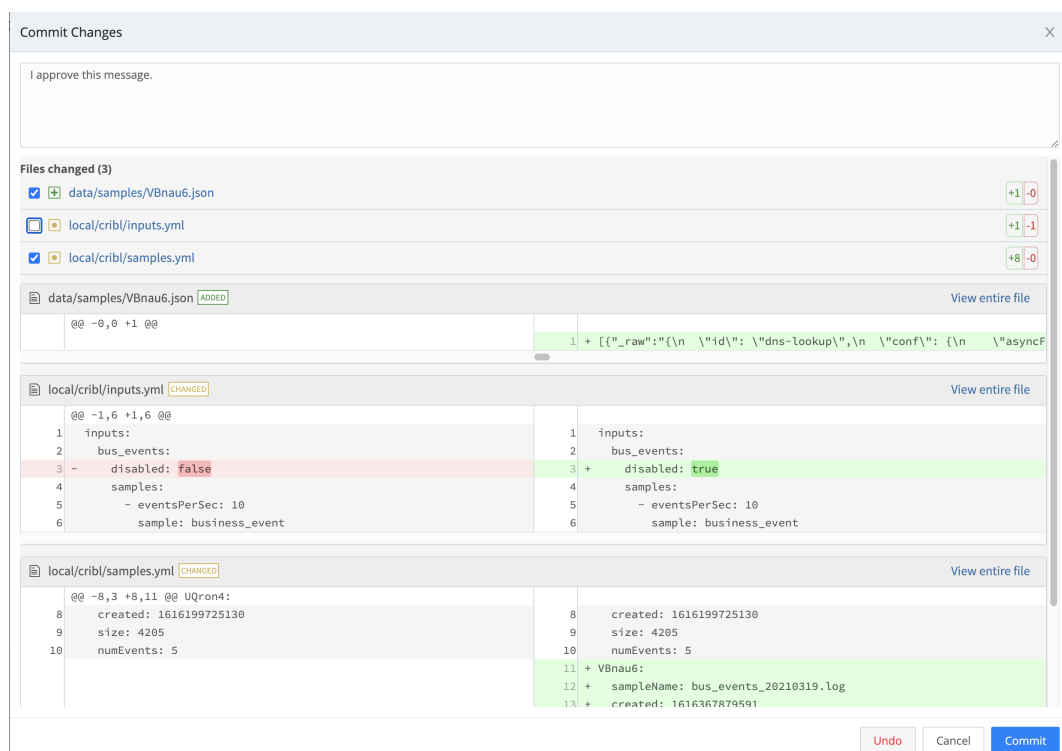
Once Git is installed, you can commit configuration changes using the `git` CLI. You can also commit changes interactively, using LogStream's UI.

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



Changes pending commit

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.



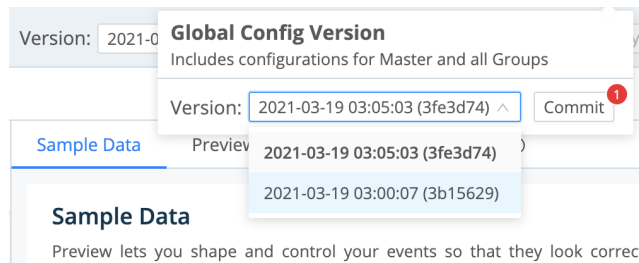
Reviewing a pending commit

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.



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



Undoing earlier commits

Finally, confirm permission for LogStream to restart.

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

57a4d73816cda38c06a6457fb8121c68b3fa14aa

Author: Cribl System <cribl@57a4d73816cda38c06a6457fb8121c68b3fa14aa>

Date: Wed Mar 10 09:03:53 2021 +0000

admin: I approve this message.

Cribl LogStream server will be restarted.

☐ Force Revert

No

Yes

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|ftp|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 [announced](#) its end of 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  **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.

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`

GitLab's **Repository Settings > Push Rules** section includes these two settings of interest:

- As needed, enable **Check whether author is a GitLab user**.
- Understand the consequences of enabling **Prevent committing secrets to Git**. This blocks commits of `.pem` and `.key` files. If you have [certificates](#) or [SSH keys](#) configured, this will break commits from LogStream, throwing only a generic `API Error` in the UI. Check your `git` CLI client for more-specific diagnostics.

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).

The screenshot shows the 'Git Settings' sidebar with 'Scheduled actions' selected. The main panel has a 'Remote URL' field with a placeholder 'Enter remote url'. Below it is the 'Authentication Type' dropdown menu, which is currently set to 'Basic'. Underneath are 'User' and 'Password' fields, both with placeholder text 'Enter user' and 'Enter password' respectively.

Git Authentication Type settings

⚠ GitHub (specifically) does not support Basic authentication.

On the **Git Settings > Scheduled Actions** tab, you can schedule a **Commit**, **Push**, or **Commit & Push** action to occur on a predefined interval.

The screenshot shows the 'Git Settings' sidebar with 'Scheduled actions' selected. The main panel has a 'Scheduled global actions' dropdown menu. The dropdown is open, showing four options: 'None', 'Commit', 'Push', and 'Commit & Push'. The 'Commit & Push' option is highlighted in blue, indicating it is the selected action.

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**.

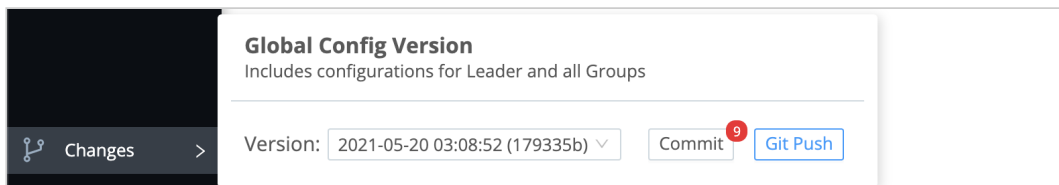
The screenshot shows the 'Git Settings' sidebar with 'Scheduled actions' selected. The main panel shows the 'Commit & Push' action selected in the dropdown. Below it is the 'Schedule' field with the value '0 0 * * *'. A tooltip is visible over the 'Schedule' field, stating 'Cron schedule to run selected git action on.'. Below the schedule field is the 'Commit message' field with the value 'Scheduled daily commit & push'. At the bottom right, there are buttons for 'Prev', 'Next', 'Cancel', and 'Save'.

Saving a Git Scheduled Action

You can schedule only one type of action. To swap to a different type, select it from the **Scheduled global actions** drop-down, and resave. To turn off scheduled Git commands, select **None** from the drop-down, and resave.

Pushing Configs to a Remote Repo

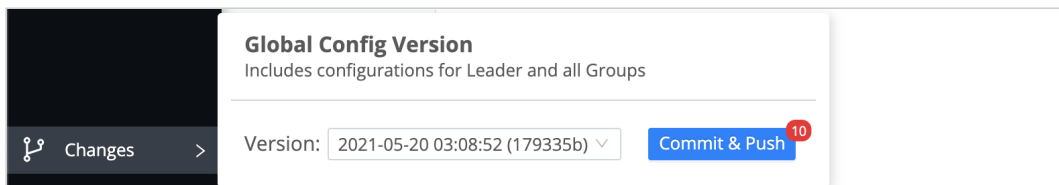
Once you've configured a remote, a **Git Push** button appears in the **Changes** overlay. You can use this to copy committed configuration changes to your remote.



Git Push button

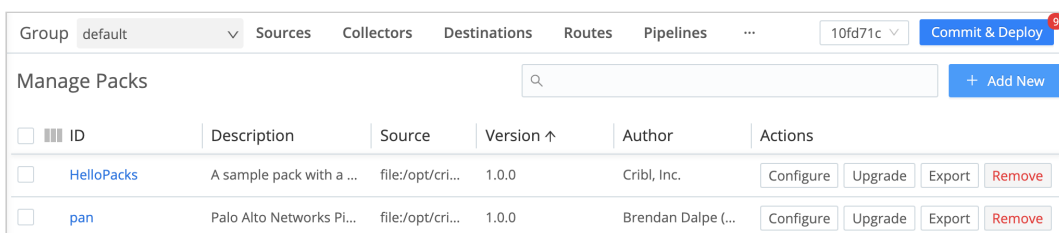
Collapse Actions

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



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.



Git combined actions button for a Worker Group

- Enabling **Collapse Actions** with a remote repo simplifies the **Commit Changes** confirmation dialog to just a commit **Message** text box, and **OK** and **Cancel** buttons, omitting the diff view. If you prefer to inspect configuration changes before committing, keep this option disabled.

Backing Up Configs Out of Band

Once you've configured your remote repo, changes to all LogStream [config files](#) under the `$CRIBL_HOME` directory will normally be backed up to your remote whenever you click the above **Git Push** or **Commit & Push** UI buttons. The exceptions are:

- Any configuration files you've chosen to store outside `$CRIBL_HOME`.
- Any configuration files residing in paths you've [added to](#) `.gitignore`.

To back up these files to your remote, you'll need to either:

- Change the above paths/settings, or
- Use an external `git` client to manually push them.

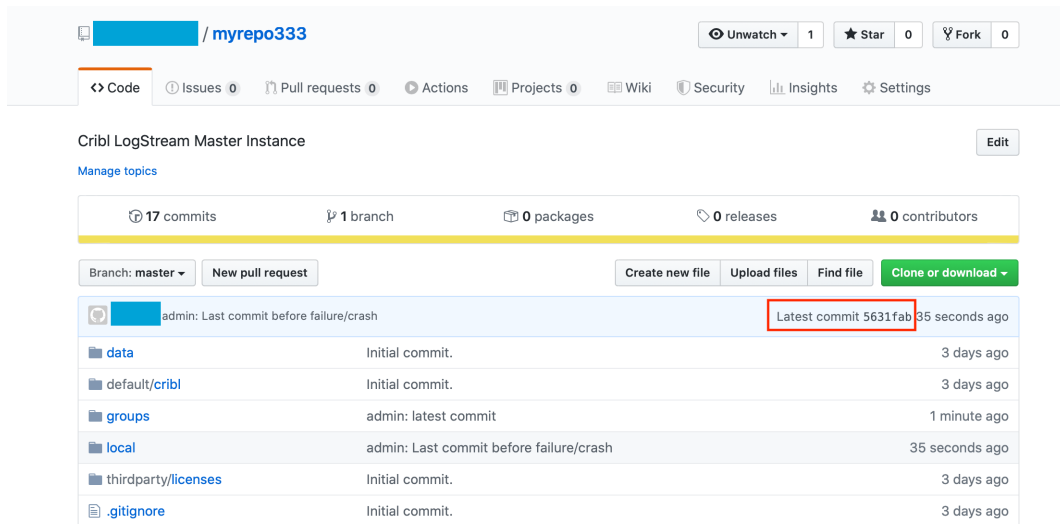
Troubleshooting Push Errors

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

- [Git Push Errors](#)
- [Git Remote Repos & Trusted CAs](#)

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.



Restoring from remote repo

Let's assume that either the entire `$CRIBL_HOME` directory of the Leader is corrupted, or you're replicating the remote repo's config onto a fresh instance. Let's also assume that the remote repo has the form:
`git@github.com:<username>/<reponame>.git` .

1. **Important:** In a directory of choice, untar the **same Cribl LogStream version** that you're trying to restore, but do not start it.

⚠ Make sure you download the **matching** LogStream version. Cribl's [Download page](#) foregrounds LogStream current version, but provides a link to [this archive](#) of prior releases.

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 o
```

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 `.gitignore` file containing a number of patterns/rules, under a section of the file labeled `CRIBL SECTION` .

```
.gitignore
```

```
# Do NOT REMOVE CRIBL and CUSTOM header lines!
# DO NOT REMOVE rules under the CRIBL section as they may be reintroduced
# 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. For details, see [Git Push Errors: Large Files Detected](#).

Files skipped with .gitignore

If you have files that you've set .gitignore to skip, you will need to back them up and restore them by means other than Git. For example, you can periodically copy/rsync them to a backup destination, and then restore them to their original locations after you complete the steps above.

Files specified in .gitignore are not only excluded from pushes to the remote repo, but are also excluded from Worker Group config bundles. When

Workers load a new config that references a skipped (and missing) file, this can produce unexpected results, and usually errors.

For example, if you add `**/auth/**` to `.gitignore`, then any certificate/key files stored in the default `$CRIBL_HOME/local/cribl/auth/certs/` path will be omitted from config deployments, because of a match on the `.../auth/...` subdirectory.

Persistent Queues

LogStream's persistent queuing (PQ) feature helps minimize data loss if a downstream receiver is unreachable. PQ provides durability by writing data to disk for the duration of the outage, and forwarding it upon recovery.

Persistent queues 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 then output them at the rate to which the receiver can sync (as opposed to blocking or dropping them). Only when this queue is full will the output impose backpressure upstream.

Life Without PQ

Backpressure behavior can be configured to one of **Block**, **Drop Events**, or (on Destinations that [support it](#)) **Persistent Queue**. 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 that the sender supports backpressure, too). In general, TCP-based senders support backpressure, but this is not a guarantee: Each upstream application's developer is responsible for ensuring that the application stops sending data once LogStream stops sending TCP acknowledgments back to it.

In **Drop** mode, the Destination will discard new events until the receiver is ready. In some environments, the in-memory queues and their block/drop behavior are acceptable.

PQ + FIFO = Durability

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 Destination 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 on the output side (i.e., after processing).
- Engaged only when **all of that output's receivers** 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.
- TLS-encrypted only for data in flight, and only on Destinations where TLS is supported and enabled. To encrypt data at rest, including disk writes/reads, you must configure encryption on the underlying storage volume(s).

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.
- **Queue-full behavior:** Determines whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.



Minimum Free Disk Space

For queuing to operate properly, you must provide sufficient disk space. 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 minimum is 5GB. Be sure to set this on your Worker Nodes (rather than on the Leader Node) when in distributed mode.

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](#)

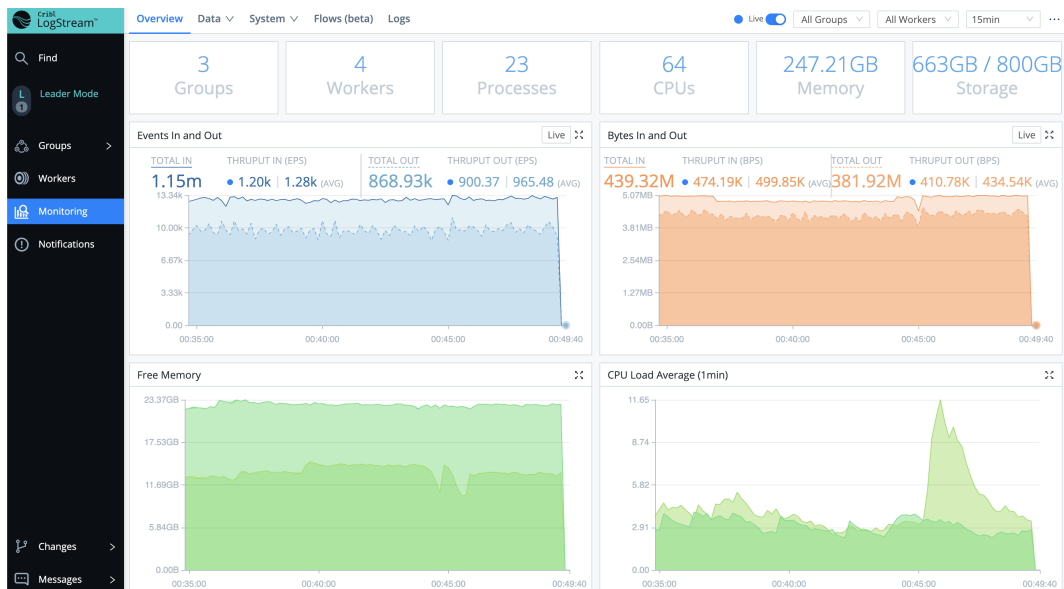
Monitoring Page

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.

The initial view (below) shows aggregate data for all Groups and all Workers. You can use the drop-downs at the upper right to isolate individual Groups, or individual Workers. Here, you can also change the display's granularity from the default 15 min . Coverage is limited to the previous 24 hours (this maximum is not configurable).

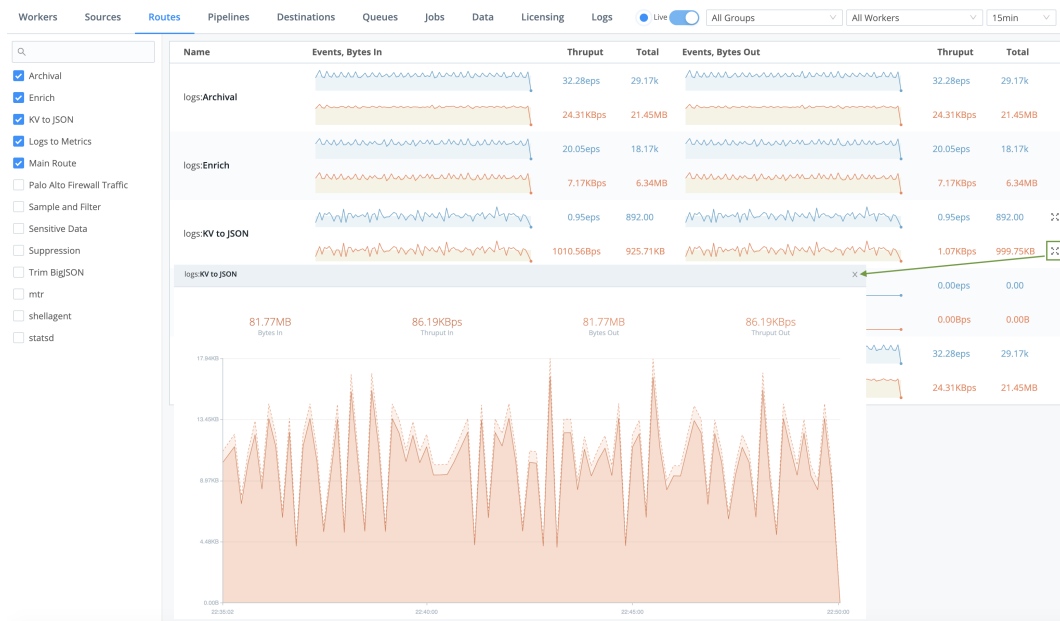
The displayed **CPU Load Average** is an average per Worker Process, updated at 1-minute granularity. (It is not an average for the Worker Node as a whole.)

Byte-related charts show the uncompressed size of processed data. Bytes in/out are measured based on the size of `_raw` (meaning that metrics events will reflect 0 bytes processed, because they include no `_raw` field).



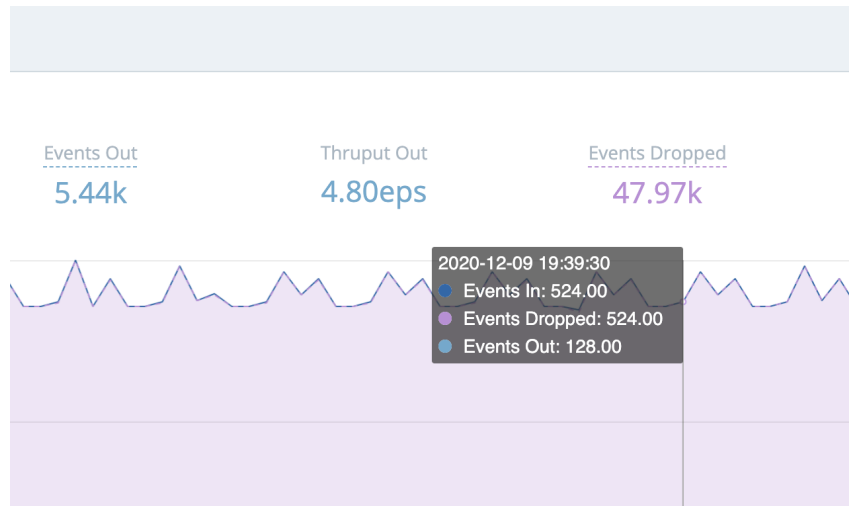
Monitoring page

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.



Sparklines and fly-out

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

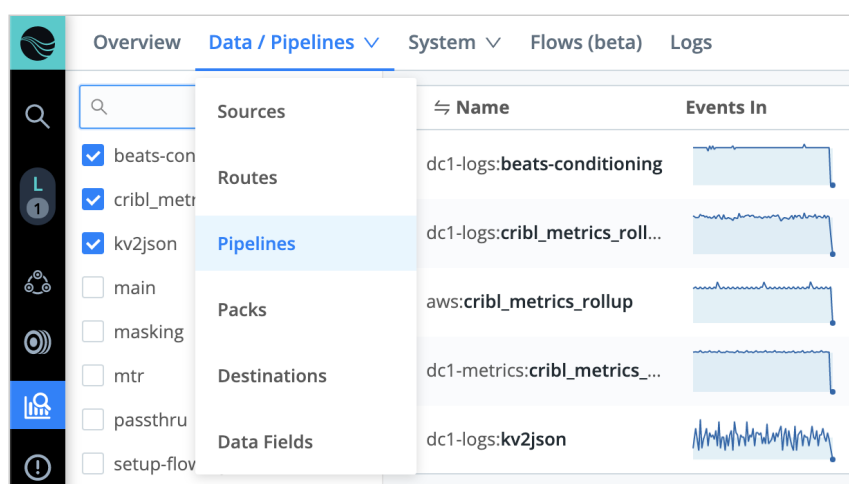


Throughput details

Data Monitoring

From the **Monitoring** page's top nav, open the **Data** submenu to isolate throughput for any of the following:

- Sources
- Routes
- Pipelines
- Packs
- Destinations
- Data Fields

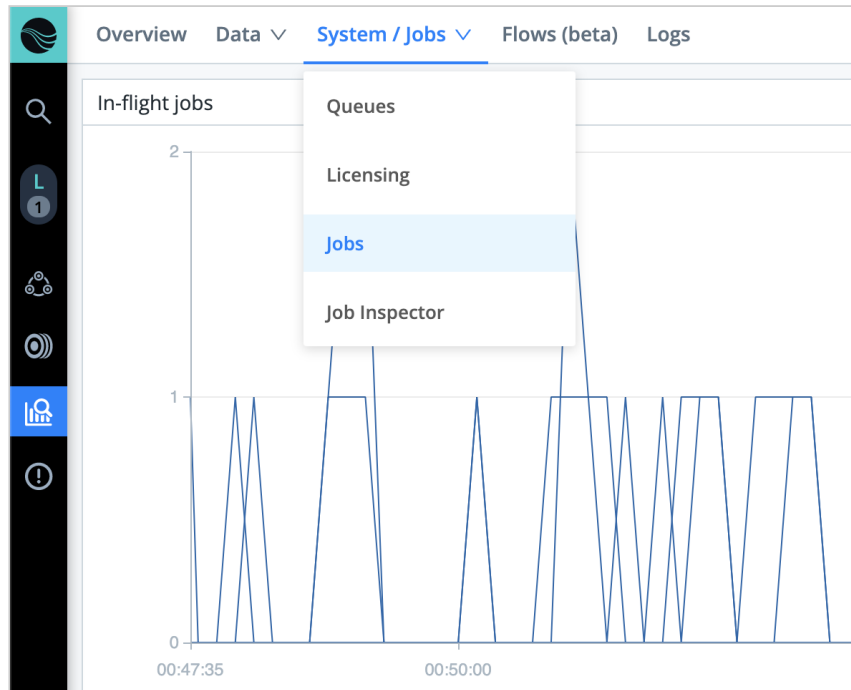


Monitoring > Data submenu (Pipelines selected)

System Monitoring

From the **Monitoring** page's top nav, open the **System** submenu to isolate throughput for any of the following:

- Queues (see [Persistent Queues](#))
- [Licensing](#)
- Jobs (and tasks in-flight, see [Collector Sources](#))
- [Job Inspector](#)



Monitoring > System submenu (Jobs in-flight selected)

Licensing

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

Job Inspector

Select **System > Job Inspector** from the Monitoring page's top nav to view and manage pending, in-flight, and completed collection jobs and their tasks. For details about the resulting page, see [Monitoring and Inspecting Collection Jobs](#).

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.

Internal Logs and Metrics

Select **Logs** from the Monitoring page's top nav. LogStream's [internal logs](#) and [internal metrics](#) provide comprehensive information about an instance's status/health, inputs, outputs, Pipelines, Routes, Functions, and traffic.

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):

The screenshot shows the Cribl Logs page with the following controls highlighted by numbered callouts:

- 1: Worker Process 1 (Source)
- 2: Fields (Main, All, None)
- 3: Main Fields (time, cid, channel, level, message, activeEP, blockedEP, dropped, endtime, inBytes, inEvents, outBytes, outEvents, provider, starttime, total, user)
- 4: Search button
- 5: Search expression, click fields to add to search, shift+click to negate
- 6: Search results (1 hour ago)
- 7: Time column

Time	Event
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: 240 }
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 }
2020-05-22T23:33:58.894Z	{ time: "2020-05-22T23:33:58.894Z", 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 }

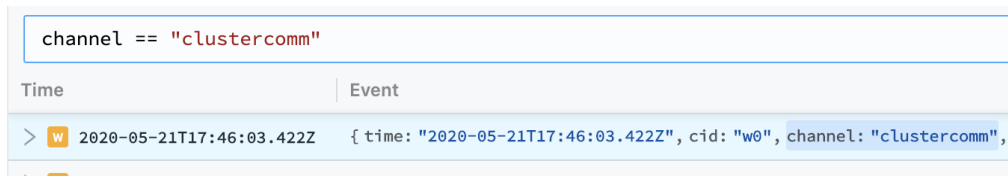
Logs page (controls highlighted)

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.

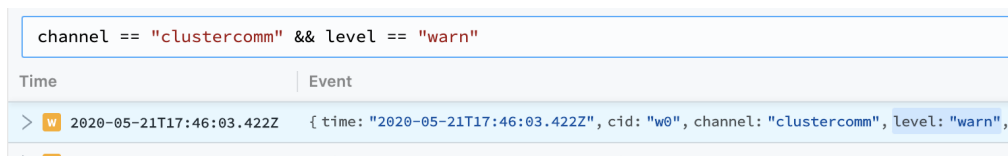
Typeahead assist is available for expression completion:



Click a field in any event to add it to a query:



Click other fields to append them to a query:

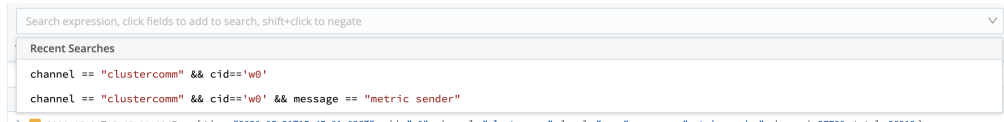


Shift+click to *negate* a field:



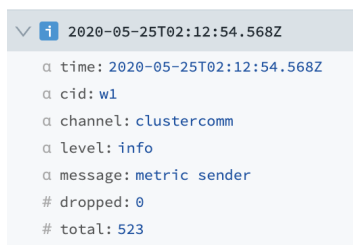
- i** To modify the depth of information that is originally input to the Logs page, see [Logging Settings](#).

6. Click the Search box's history arrow (right side) to retrieve recent queries:



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:



Logging Settings

On LogStream's Settings pages, you can adjust the level (verbosity) of internal logging data processed, per logging channel. You can also redact fields in customized ways. In a distributed deployment, you manage each of these settings per Worker Group.

Change Logging Levels

To adjust logging levels:

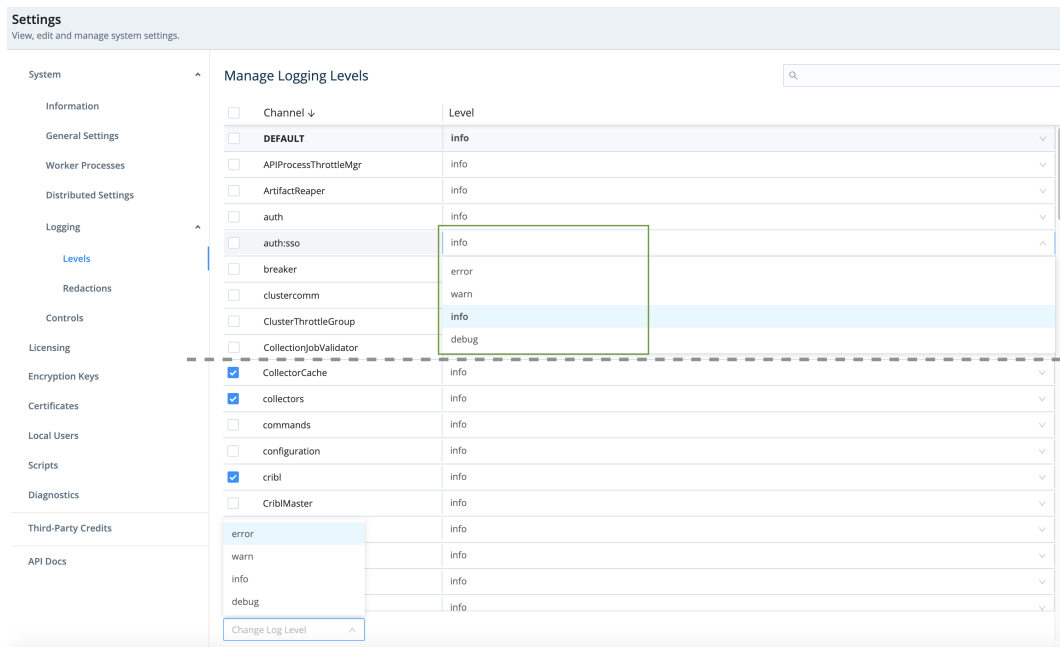
- In a single-instance deployment, or for the Leader Node's own logs, select global **Settings** (lower left) > **System** > **Logging** > **Levels**.
- In a distributed deployment's left nav, first select **Groups**, then click into the group you want to configure. Next, select group **Settings** (upper right) > **System** > **Logging** > **Levels**.

On the resulting **Manage Logging Levels** page, you can:

- Modify one channel by clicking its **Level** column. In the resulting drop-down, you can set a verbosity level ranging from **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.



Manage Logging Levels page

Change Logging Redactions

On the **Redact Internal Log Fields** page, you can customize the redaction of sensitive, verbose, or just ugly data within LogStream's internal logs. To access these settings:

- In a single-instance deployment, or for the Leader Node's own logs, select global ⚙ **Settings** (lower left) > **System** > **Logging** > **Redactions**.
- In a distributed deployment's left nav, first select **Groups**, then click into the group you want to configure. Next, select group **Settings** (upper right) > **System** > **Logging** > **Redactions**.

Redact Internal Log Fields

Custom Redact String ⓘ REDACTED

Additional Fields ⓘ field1 x field3 x

Default Fields ⓘ awsSecretKey passphrase authToken azureStorageAccountKey password workspaceKey team token

Cancel Save

It's easiest to understand the resulting **Redact Internal Log Fields** page's fields from bottom to top:

- **Default fields:** LogStream always redacts these fields. You can't modify this list.
- **Additional 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
<code>curl http(s)://<host>: <port>/api/v1/health</code>	<code>{"status":"healthy"}</code>
<code>curl http(s)://<host>: <port>/api/v1/health</code>	<code>{"status":"healthy","startTime":1617814717110}</code> (see details below)

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.

Counter-type metrics in LogStream do not monotonically increase or decrease. They are reset at the end of each reporting period. LogStream does not report counters when their value is 0. For example, if there aren't any Destinations reporting dropped events then the `total.dropped_events` metric won't be reported because its value would be 0.

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. These metrics are not exported as part of [Cribl Internal Source](#) to routes/pipelines therefore they must be obtained using the REST endpoint documented on this page.

- `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:DatagenWeblog" }},
],
"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 understanding CPU utilization, but it requires you to query all your CPUs individually,

`cribl_wp` Metric Dimension

`cribl_wp` is a useful dimension that identifies the Worker Process that processed each event.

Notifications

In LogStream 3.1 or later, you can configure Notifications about Destinations that report errors, Destinations experiencing backpressure, and pending LogStream license expiration.

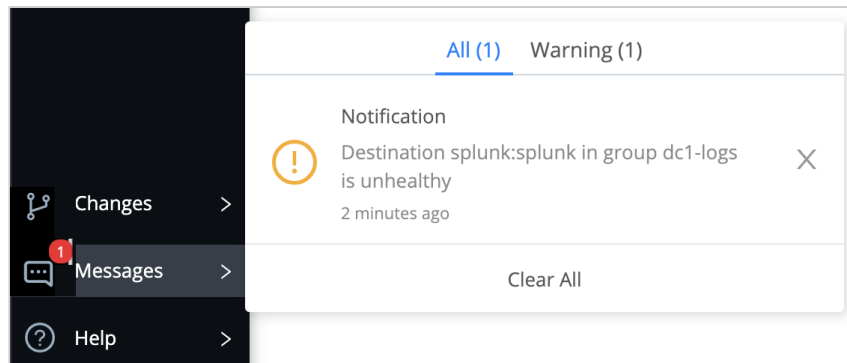
Notifications are not designed to take the place of alerts on your overall infrastructure's health – but they warn you about conditions that could impede expected data output from LogStream.

- Notifications require an Enterprise or Standard [license](#), without which the configuration options described below will be hidden or disabled in LogStream's UI.

Notifications and Targets

Every Notification is sent to one or more **targets**. By default, any Notification that you configure will have a target of `System Messages`. This means that when a Notification is triggered, it will add an indicator on the left nav's **Messages** tab. Click this to view details in a fly-out, as shown below.

All Notifications will also be sent as events to LogStream's [internal logs](#) – both application-wide, and with a filtered view available on affected Destinations. The application-wide logs are recorded as `notifications.log` on the Leader Node. The Leader Node is also responsible for sending all Notifications.



System Messages pane

You can also send any Notification to additional targets, using LogStream's native PagerDuty integration and/or by specifying custom webhooks. For details, see [Configuring Targets](#).

Notifications and RBAC

Notifications work with LogStream's role-based access control. For users with non-administrative permissions, their assigned [Roles and Policies](#) determine the Worker Groups on which they can view Notification messages, and can create and manage Notifications and targets

Configuring Notifications

Destination-state notifications, and license-expiration notifications, are configured separately.

Destination-State Notifications

On individual Destinations, you can configure Notifications that will trigger under these conditions:

- [Destination Backpressure Activated](#)
- [Unhealthy Destination](#)

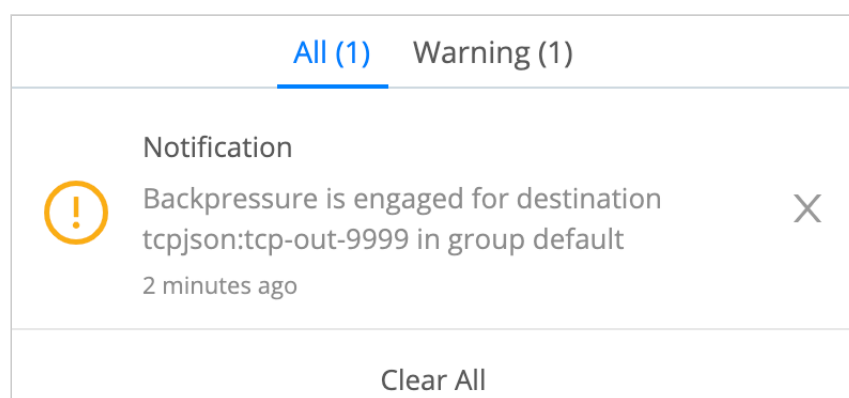
Read on for details about the above conditions, and about how to specify them in the UI's **Condition** field.

Destination Backpressure Activated

This will generate a Notification reading `Backpressure is engaged for Destination <name>` when either of the following events occur:

- The Destination's **Backpressure behavior** is set to `Block`, and backpressure causes outgoing events to block.
- The Destination's **Backpressure behavior** is set to [Persistent Queue](#), but its **Queue-full behavior** is set to `Block`, and a filled queue causes outgoing events to block.

The threshold for the Notification to trigger is: LogStream detected a blocked state during $\geq 5\%$ of the trailing **Time window** that you configure in the [Notification Settings](#) below.



Backpressure notification

Unhealthy Destination

This will generate a `Destination <name> is unhealthy` Notification when the Destination's health has been in "red" status (as indicated on the UI's [Monitoring](#) page) over the trailing **Time window** that you configure in the [Notification Settings](#) below.

The algorithm has slight variations among Destination types, but red status generally means that $\geq 5\%$ of health checks, aggregated over the **Time window**, reported either:

- An error inhibiting the Destination's normal operation, such as a connection error; or
- For multiple-output Destinations like [Splunk Load Balanced](#) or [Output Router](#), $> 50\%$ of the Destination senders in an error state.

Configuring Destination Notifications

To start configuring a Destination-state Notification:

1. Configure and save the Destination.
2. Access this Destination's **Notifications** tab. Either:
 - Click the **Notifications** button on the **Manage...Destinations** page's appropriate row, or
 - Reopen the Destination's config modal, and click its **Notifications** tab.
3. Click **+ Add New** to access the **New Notification** modal shown below.

The image is a composite screenshot of the Splunk interface. The top part shows the 'Notifications' tab for a destination named 'splunk'. It features a table with columns: Notification ID, Group ID, Condition, Targets, Enabled, and Actions. A row is visible for 'Splunk-unhealthy' with condition 'Unhealthy Destination' and target 'sns-notify'. Below this, a 'New Notification' modal is open. The modal has two tabs: 'Notification Settings' (active) and 'Metadata'. The 'Notification Settings' tab contains fields for: ID* (with a 'Yes' toggle), Condition* (set to 'Unhealthy Destination'), Notification targets (a dropdown menu with 'Select one' and a 'Create' button), Default target (set to 'System Messages'), Destination name* (set to 'splunksplunk'), and Time window (set to '60s').

Configuring a Destination Notification (composite screenshot)

The **New Notification** modal provides [Notification Settings](#) and [Metadata](#) tabs, whose controls are listed in the respective sections below.

Notification Settings

ID: Enter a unique ID for this Notification. (Cribl recommends using a string that will make the Notification's purpose clear.)

Condition: Select either **Unhealthy Destination** or **Destination Backpressure Activated**. (You can set triggers for both conditions on the same Destination, but you must configure them as separate Notifications.)

Notification targets: The **Default target** is always always locked to **System Messages**. Click **Add target** to send this Notification to additional targets. You can add multiple targets.

- Use the resulting **Notification targets** drop-down to select any target you've already configured.
- Click **Create** to configure a new target. (See [Configuring Targets](#) for details.)

Destination name: This is locked to the Destination on which you're setting this Notification.

Time window: Sets the threshold period before the Notification will trigger. E.g., the default `60s` will generate a Notification when the Destination has reported the trigger condition over the past 60 seconds. (For % trigger conditions, see [Destination Backpressure Activated](#) and [Unhealthy Destination](#).) To enter alternative numeric values, append units of `s` for seconds, `m` for minutes, `h` for hours, etc.

Metadata

Click **Add field** here to add custom metadata fields to your Notifications, as key-value pairs:

Name: Enter a name for this custom field.

Value: Enter a JavaScript expression that defines this field's value, enclosed in quotes or backticks. (Can evaluate to a constant.)

- Once you've saved your Notification, you can see Notification events specific to this Destination on the Destination config modal's **Events** tab. For a comprehensive view of all Notification events, see the systemwide [Events Tab](#).

Recurrence/Expiration

If a Destination-State Notification's trigger condition persists beyond your configured [Time window](#), expect a new notification to be sent once per **Time window** interval.

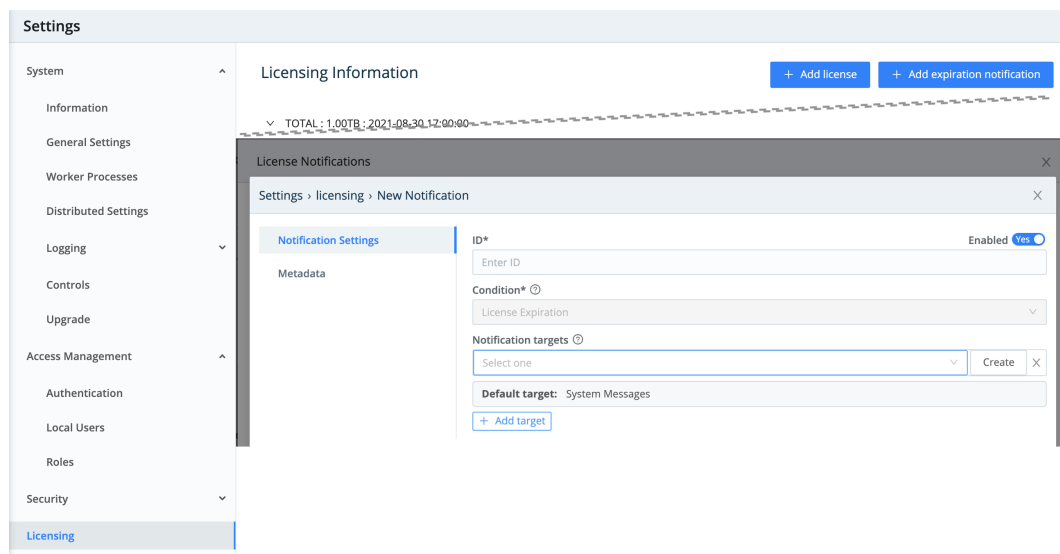
Notifications will cease when the triggering condition clears. There is no explicit "problem cleared" Notification.

License-Expiration Notifications

To prevent interruptions in data throughput, you can configure a Notification that will be triggered two weeks before your LogStream paid [license](#) expires, and then again upon expiration. (If the two-week Notification is cleared from the [Messages](#) tab between those dates, but the license has not been extended, it will trigger again.)

Configuring License-Expiration Notifications

1. Select global [Settings](#) (lower left) > **Licensing**.
2. Click **+ Add expiration notification** to access the **New Notification** modal shown below.



Configuring an expiration Notification (composite screenshot)

This **New Notification** modal provides [Notification Settings](#) and [Metadata](#) tabs, with a subset of the controls available in the [Destination-unhealthy modal](#):

Notification Settings

ID: Enter a unique ID for this Notification.

Condition: This modal's triggering condition is locked to [License Expiration](#).

Notification targets: The **Default target** is always always locked to [System Messages](#). Click **Add target** for each additional target that you want to send this Notification to.

- Use the resulting **Notification targets** drop-down to select any target you've already configured.

- Click **Create** to configure a new target. See [Configuring Targets](#) for details.

Metadata

The options here are identical to those on the [Destination-unhealthy modal](#)'s **Metadata** tab.

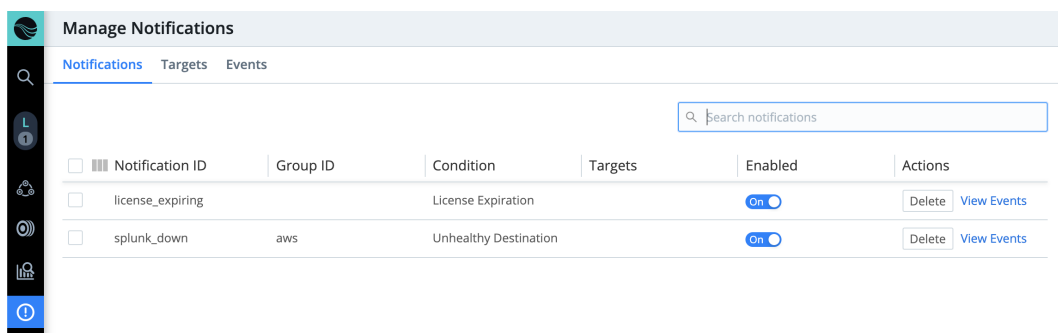
Managing Notifications

The **Manage Notifications** page provides global display and controls for all your configured Notifications, targets, and triggered Events – across all Destinations and all Worker Groups. To access this page:

- In a [distributed deployment](#), click the left nav's (!) **Notifications** tab.
- In a [single-instance deployment](#), click the top nav's **Notifications** tab.

Notifications Tab

This tab lists all your configured [Destination-unhealthy](#) Notifications, across all Destinations, along with any configured [license-expiration](#) Notifications. You can't create new Notifications here, but you can disable or delete existing Notifications; you can also click on any Notification's row to open and modify its configuration.



Manage Notifications						
Notifications Targets Events						
Search notifications						
Notification ID	Group ID	Condition	Targets	Enabled	Actions	
license_expiring		License Expiration		On	Delete	View Events
splunk_down	aws	Unhealthy Destination		On	Delete	View Events

Notifications tab

Targets Tab

This tab is where you centrally configure and manage targets that are available across LogStream, to all Destination- and license-based Notifications. See [Configuring Targets](#) for details.

Events Tab

This tab displays logged events that have been fired by all your configured Notifications. You can filter by search string, and by lookback time.

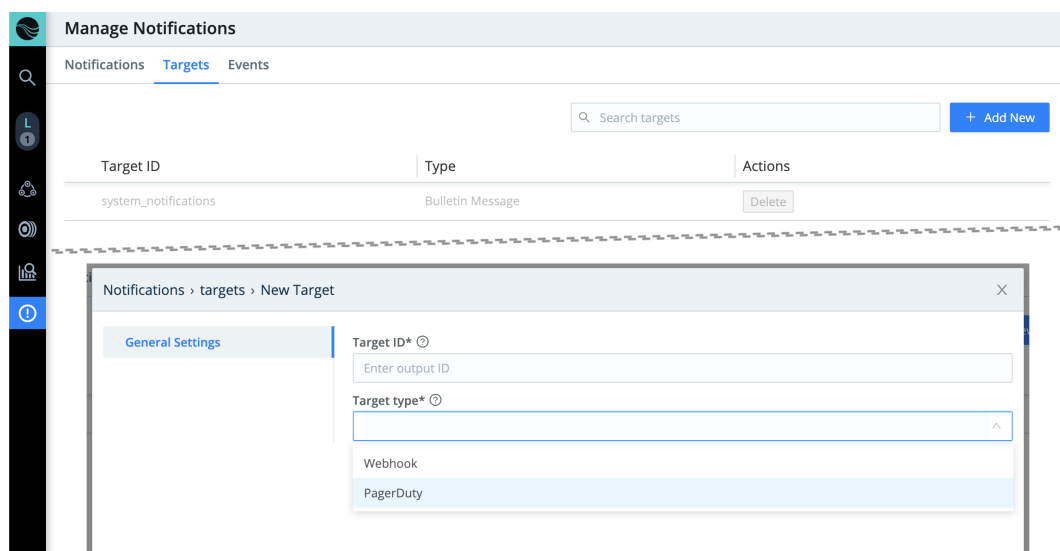
What's Next

In future LogStream releases, Cribl plans to expand the Notifications feature with options to configure additional triggering conditions and time resolutions.

Configuring Targets

To add a new Notification target from the [Manage Notifications](#) page's **Targets** tab:

1. Click **+ Add New** to open the **New Target** modal shown below.
2. Give this target a unique **Target ID**.
3. Set the **Target type** to either [PagerDuty](#) or [Webhook](#). Then configure the target according to the corresponding section below.



Adding a new target (composite screenshot)

- ❏ Notifications require an Enterprise or Standard [license](#), without which all the target configuration options described on this page will be hidden or disabled in LogStream's UI.

PagerDuty Targets

This option sends LogStream Notifications to the [PagerDuty](#) real-time incident response platform, using LogStream's native integration with the PagerDuty API. Select **Target type:** `PagerDuty` to expose the following additional options on the modal's (single) **General Settings** left tab:

Routing key: Enter your 32-character Integration key on a PagerDuty service or global ruleset.

Group: Optionally, specify a PagerDuty default group to assign to LogStream Notifications.

Class: Optionally, specify a PagerDuty default class to assign to LogStream Notifications.

Component: Optionally, a PagerDuty default component value to assign to LogStream Notification. (This field is prefilled with `logstream`.)

Severity: Set the default message severity for events sent to PagerDuty. Defaults to `info`; you can instead select `error`, `warning`, or `critical`. (Will be overridden by the `__severity` value, if set.)

Webhook Targets

This option enables you to send LogStream Notifications to an arbitrary webhook. Select **Target type:** `Webhook` to expose multiple left tabs, with the following configuration options:

General Settings

The added options that appear on this first left tab are:

URL: The endpoint that should receive LogStream Notification events.

Method: Select the appropriate HTTP verb for requests: `POST` (the default), `PUT`, or `PATCH`.

Format: Specifies how to format Notification events before sending them to the endpoint. Select one of the following:

- `NDJSON` (newline-delimited JSON, the default).
- `JSON Array`.
- `Custom`, which exposes these additional fields:
 - **Source expression:** JavaScript expression whose evaluation shapes the event to send to the endpoint. E.g.: `${fieldA}`, `${fieldB}`.

Defaults to `__httpOut` (meaning the value of the `__httpOut` field).

- **Drop when null:** Toggle to `Yes` if you want to drop events where the above **Source expression** evaluates to `null`.
- **Content type:** Defaults to `application/x-ndjson`. You can substitute a different content type for requests sent to the endpoint. This entry will be overridden by any content types set in this modal's **Advanced Settings** tab > **Extra HTTP Headers** section.

Processing Settings

The options on this left tab are identical to those on the [Webhook Destination](#)'s **Processing Settings** tab, except that here, the default **System fields** entry is `cribl_host`.

Advanced Settings

The options on this left tab are identical to those on the [Webhook Destinations](#) **Advanced Settings** tab.

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

⚠ 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.

See notes on [Upgrading from LogStream 2.2 or Prior Versions](#) below.

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 WORKERS | 1 GROUPS | 4 ALIVE | 0 UNHEALTHY | 1 SOFTWARE VERSIONS | 1 CONF VERSIONS

<input type="checkbox"/>	#	GUID	Host	Status	Group	CPU	RAM	Last Time	Start Time	Config Ve...	Version
<input type="checkbox"/>	1	b396df9e-6e49-488d-8479-0b5c	1e9a7eeb5765	alive	default	6	1.94GB	2020-06-05 13...	2020-06-05 13...	8339833	2.1.4-bc26...

Worker Info

GUID: b396df9e-6e49-488d-8479-0b5c

Host: 1e9a7eeb5765

Status: alive

Group: default

CPU: 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

Restart

Worker Details

firstMsgTime: 1591388339455

group: default

id: b396df9e-6e49-488d-8479-0b5c

info: 10 items...

lastMsgTime: 1591389368748

status: healthy

workerProcesses: 2

<input type="checkbox"/>	> 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...
<input type="checkbox"/>	> 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...
<input type="checkbox"/>	> 4	f83a2153-7466-45d9-b2fe-aab2f	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 WORKERS | 1 GROUPS | 4 ALIVE | 0 UNHEALTHY | 1 SOFTWARE VERSIONS | 1 CONF VERSIONS

#

GUID

Host

Status

Group

CPU

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

CPU: 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-RC4-8b3acb4

Restart

Worker Details

firstMsgTime: 1591389998446

group: default

id: b396df9e-6e49-488d-8479-0b501c9dbe65

info: 11 items...

lastMsgTime: 1591391726555

status: healthy

workerProcesses: 4

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

CPU: 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-RC4-8b3acb4

Restart

Worker Details

firstMsgTime: 159139006088

group: default

id: f83a2153-7466-45d9-b2fe-aab2b6edce04

info: 11 items...

lastMsgTime: 1591391734227

status: healthy

workerProcesses: 4

Post-2.1.4 upgrade to 2.2

Upgrade and Rollback via the UI

Page 269 of 1087

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.

- i** These options will work only if all LogStream instances (including Worker Processes) start at v.2.4.4 or higher.

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.

Upgrade Leader

Upgrade Worker Groups

#	Group	Worker nodes	Versions	Actions
1	dc1-logs	2	42.0 (2)	<button>Upgrade</button>
2	aws	1	42.0 (1)	<button>Upgrade</button>
3	dc1-metrics	1	42.0 (1)	<button>Upgrade</button>

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 **Upgrade Group** dialog offers two states: Basic Upgrade and Advanced Upgrade.

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.

LogStream will check to ensure that Workers are upgraded no higher than the Leader's version. Upgrades are performed as the user that was running LogStream on each machine.

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.

- i** When you initiate an upgrade via the UI, the new package is untarred to `$CRIBL_HOME/unpack.<random-hash>.tmp` . This location inherits the permissions you've already assigned to `$CRIBL_HOME` .

Backup and Rollback

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 from LogStream 2.2 or 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/cribl 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:

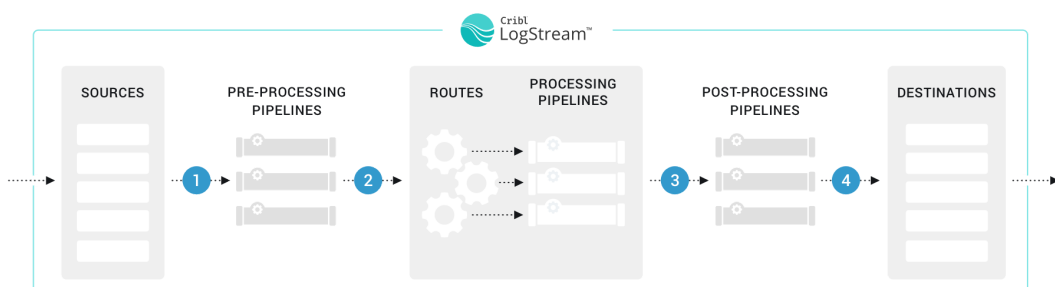
- Fields that start with a double-underscore are known as **internal fields**, and each [Source](#) can add one or many to each event. For example, [Syslog](#) adds both a `__inputId` and a `__srcIpPort` field. Internal fields are used only within Cribl LogStream, and are not passed down to [Destinations](#).
- 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

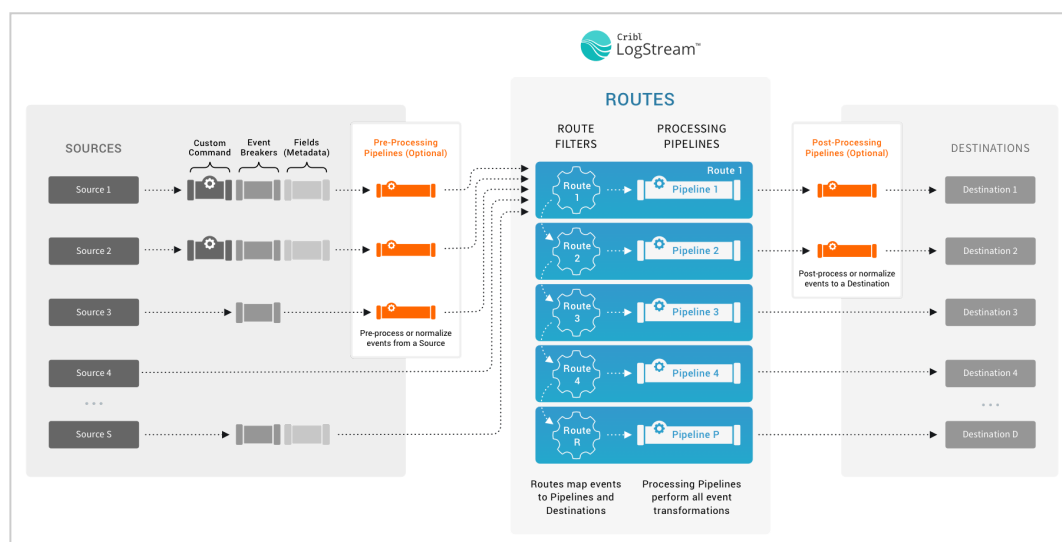
One way to see what an event looks like as it travels through the system is to use the **Capture** feature. While in [Preview](#) (right pane):

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. (Note that because Event Breakers precede tokens, Breakers cannot see or act on tokens.)

4. Auth tokens are applied at this point on Splunk HEC Sources. (Details [here](#).)
5. Fields/Metadata: Optionally, you can add these enrichments to each incoming event. You add fields by specifying key/value pairs, per Source, in a format similar to 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.
6. Auth tokens are applied at this point on HTTP-based Sources other than Splunk HEC (e.g., [Elasticsearch API](#) Sources).
7. Pre-processing Pipeline: Optionally, you can use a single Pipeline to condition (normalize) data from this input before the data reaches the Routes.
8. [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.
9. 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.
10. Post-processing Pipeline: Optionally, you can append a Pipeline to condition (normalize) data from each Processing Pipeline before the data reaches its Destination.
11. [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

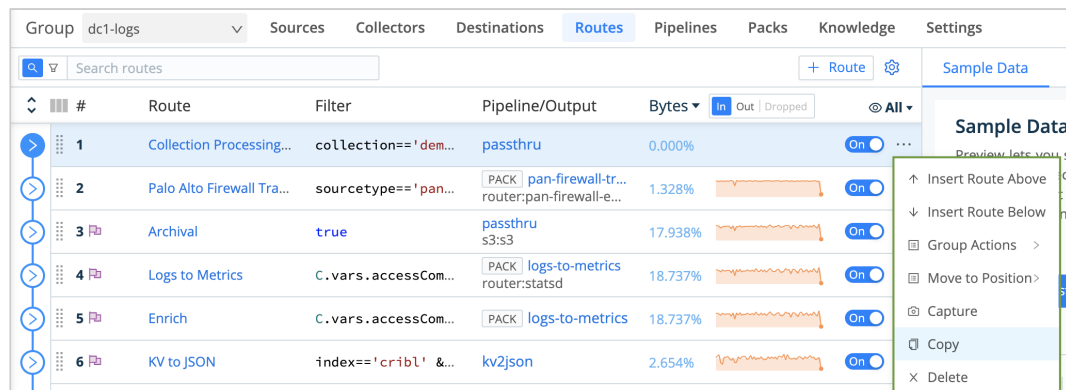
Routes apply filter expressions on incoming events to send matching results to the appropriate Pipeline. Filters are JavaScript-syntax-compatible expressions that are configured with each Route. Examples are:

- `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.

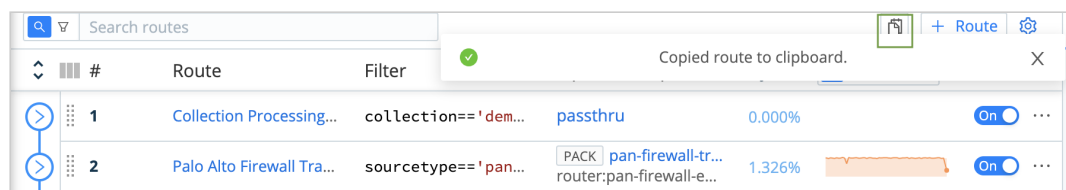
Routes are evaluated in their display order, top->down. The stats shown in the **Bytes/Events** (toggle) column are for the most-recent 15 minutes.

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.



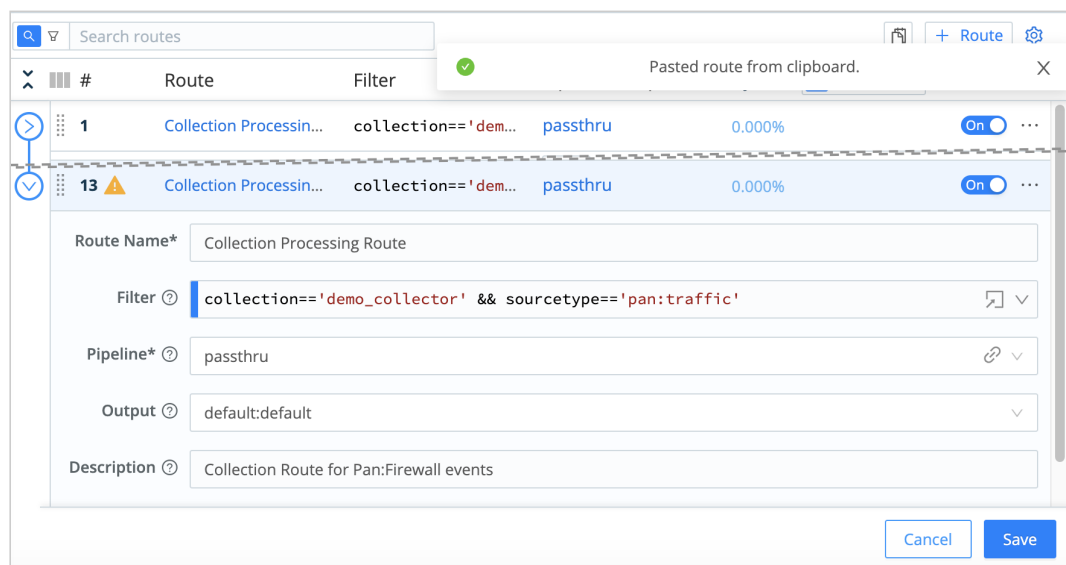
Route > Options menu

Copying a Route displays the confirmation message and the (highlighted) Paste button shown below.



Paste button for copied Route

Pasting creates an exact duplicate of the Route, with a warning indicator to change its duplicate name.



Pasted duplicate Route

Output Destination

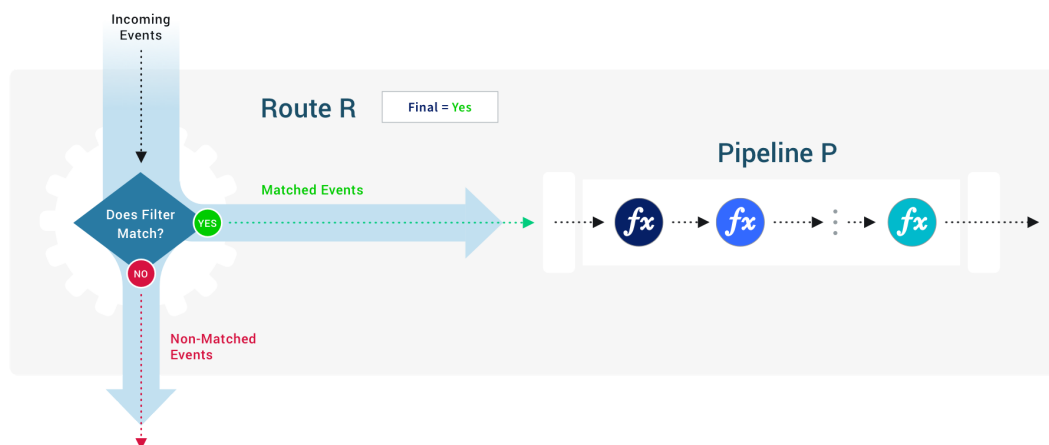
You can configure each Route with an output [Destination](#) that denotes where to send events after they're processed by the Pipeline.

The Final Toggle

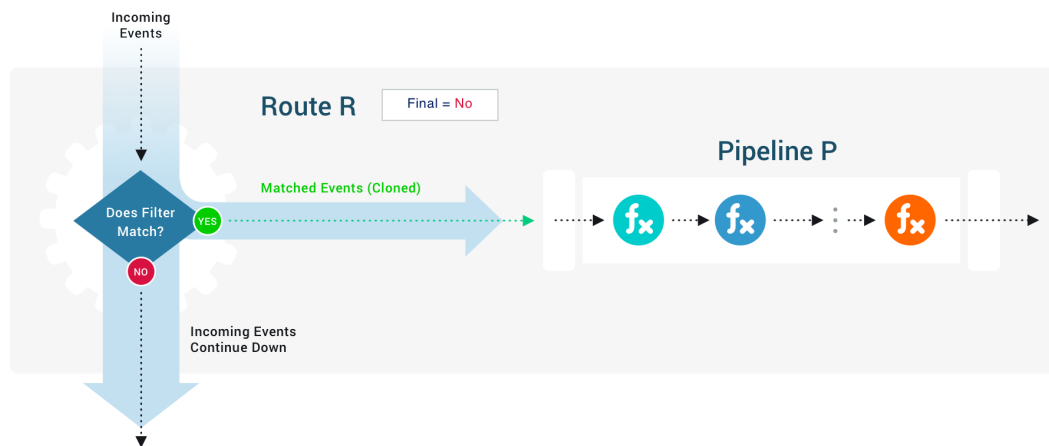
When an event that enters the system and matches a Route-Pipeline pair, it will usually be either:

- 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.

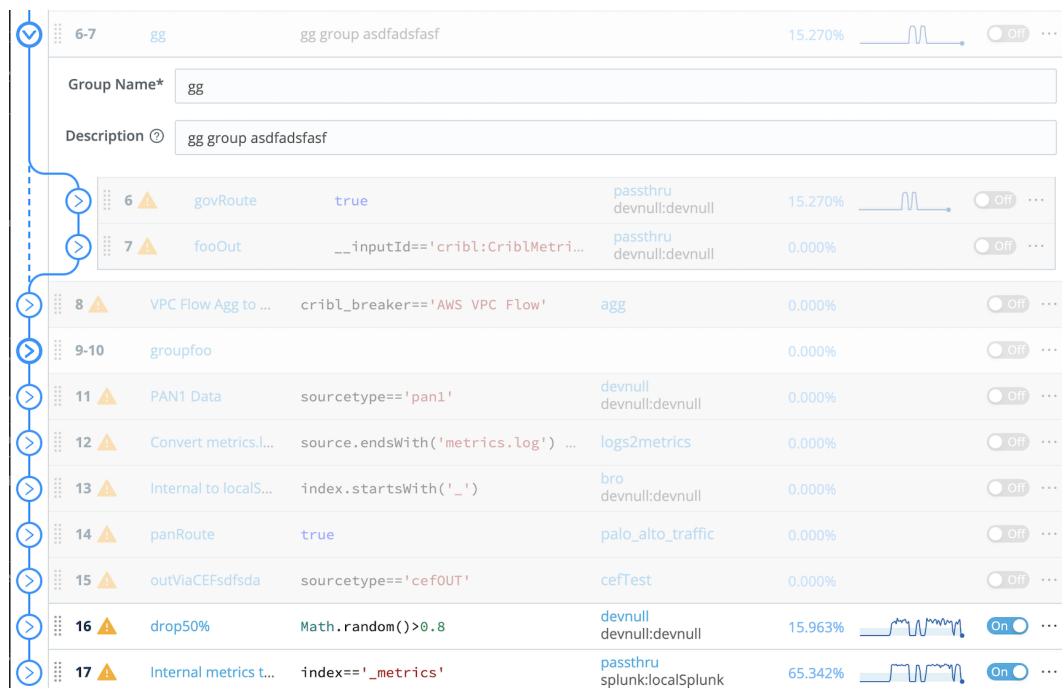


Unreachable Route warning, on hover

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](#)).



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

Data matched by a given [Route](#) is delivered to a Pipeline. Pipelines are the heart of LogStream processing. Each Pipeline is a list of [Functions](#) that work on the data.

-  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). Next, click any displayed Pipeline to see or reconfigure its contained Functions.

Adding Pipelines

To create a new Pipeline, or to import an existing Pipeline to a different LogStream instance, click **+ Pipeline** at the upper right. The resulting menu offer three options:

- **Create Pipeline:** Configure a new Pipeline from scratch, by adding Functions in LogStream's graphical UI.
- **Import from File:** Import an existing Pipeline from a `.json` file on your local filesystem.
- **Import from URL:** Import an existing Pipeline from `.json` file at a remote URL. (This must be a public URL ending in `.json` – the import option doesn't pass credentials to private URLs – and the target file must be formatted as a valid Pipeline configuration.)

Group	default	Sources	Destinations	Routes	Pipelines	Packs	Knowledge	Settings
Search pipelines		Show	All	All	Processing	Pre/Post	+ Pipeline	
<input type="checkbox"/>	Pipeline ...	Description	Route /...	Functions	Outp...	Stats		
<input type="checkbox"/>	cisco_asa	Filter and Sa...	None	11	None	IN 0 OUT 0 ERR	Create Pipeline Import from File Import from URL	
<input type="checkbox"/>	cisco_estr...		None	3	None	IN 0 OUT 0 ERR		

Creating or importing a Pipeline

- To export a Pipeline, see [Advanced Mode \(JSON Editor\)](#).

To import or export a Pipeline along with broader infrastructure (like Knowledge Objects and/or sample data files), see [Packs](#).

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.

Data

Routes

Pipelines

Knowledge

System Settings

Search pipelines

Show

All

All

Processing

Pre/Post

+ Add Pipeline

<div><div><input type="checkbox"/></div><div><div><div><div></div></div></div><div>Pipeline Name</div><div>↑</div></div></div>	Route / Input	Functions	Output	Stats	
<div><div><input type="checkbox"/></div><div><div>wineventlogs</div></div></div>	None	10	None	<div><div>IN 0</div><div>OUT 0</div><div>ERR 0</div></div>	<div>...</div>
<div><div><input type="checkbox"/></div><div><div>passthru</div></div></div>	None	0	None	<div><div>IN 0</div><div>OUT 0</div><div>ERR 0</div></div>	<div>...</div>
<div><div><input type="checkbox"/></div><div><div>palo_alto_traffic</div></div></div>	None	8	None	<div><div>IN 0</div><div>OUT 0</div><div>ERR 0</div></div>	<div>...</div>

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 shortcut works when no field has focus.)

In the condensed Pipelines display above, you can also hover over any Pipeline's **Functions** column to see a horizontal preview of the stack of Functions contained in the Pipeline:

Group	default	Sources	Destinations	Routes	Pipelines	Packs	Knowledge	Settings
<div> <div>Search pipelines</div> <div>Show All</div> <div>All Processing Pre/Post</div> <div>+ Pipeline</div> </div>								
	Pipeline N...	Description	Route ...	Functions	O...	Stats	Actions	
<input type="checkbox"/>	passthru		None	0	No...	IN 0 OUT 0 ERR 0	...	
<input type="checkbox"/>	devnull		None	1	No...	IN 0 OUT 0 ERR 0	...	
<input type="checkbox"/>	main		default	1	d... lt	IN 0 OUT 0 ERR 0	...	
<input type="checkbox"/>	c	<div> <div>f Comment</div> <div>f Rollup Metrics</div> <div>f Eval</div> <div>f Comment</div> <div>f Eval</div> </div>				ERR 0	...	
<input type="checkbox"/>	cribl_metric...		Crib... rics	5	None	IN 0 OUT 0 ERR 0	...	

Preview on hovering over the bottom Pipeline (highlighted in gray)

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				+ Add Function	⚙
Attach Pipeline to Route				IN 3.82k	OUT 3.82k ERR 0
↕	#	Function	Filter	Show All	
➤	1	Eval	true	On	...
➤	2	Eval	message	On	...
➤	3	Eval	severityName && facilityName	On	...
➤	4	Eval	procid== '-'	On	...

Pipeline Functions

- i** You can streamline the above display by organizing related Functions into **Function groups**.

Pipeline Settings

Click the gear button 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 [< Back](#)

Attach Pipeline to Route

Id* ⓘ
cisco_estreamer

Async Function Timeout (ms) ⓘ
1000

Description ⓘ
Enter a description

[Advanced Mode](#) ⓘ

Pipeline Settings

Advanced Mode (JSON Editor)

Once you've clicked the gear button to enter [Pipeline Settings](#), you can click **Edit as JSON** at upper right to edit the Pipeline's definition in a JSON text editor. In this mode's editor, you can directly edit multiple values. You can also use the **Import** and **Export** buttons here to copy and modify existing Pipeline configurations, as `.json` files.

Pipelines > elastic > Advanced Settings Mode [< Back](#)

Attached to Route: [elastic](#)

[Simple Mode](#) [Import](#) [Export](#)

```

1  {
2    "id": "elastic",
3    "conf": {
4      "output": "default",
5      "groups": {},
6      "asyncFuncTimeout": 1000,
7      "functions": [
8        {
9          "id": "drop",
10         "filter": "host!='192.168.1.241'",
11         "disabled": true,
12         "conf": {}
13       }
14     ]
15   }
16 }

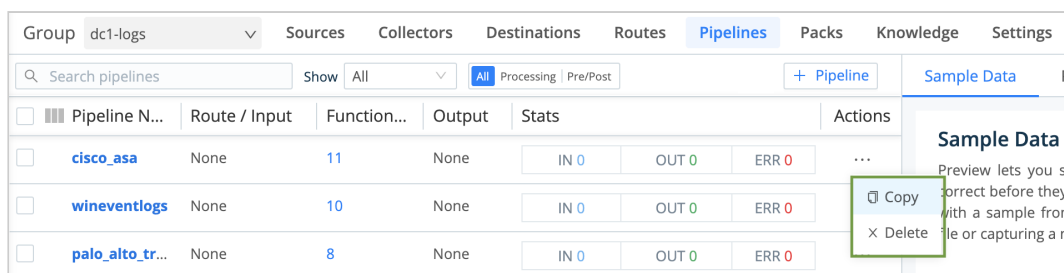
```

Advanced Pipeline Editing

Click **Edit in GUI** at upper right to return to the graphical Pipeline Settings page; then click **Back to** to restore the graphical Pipeline editor.

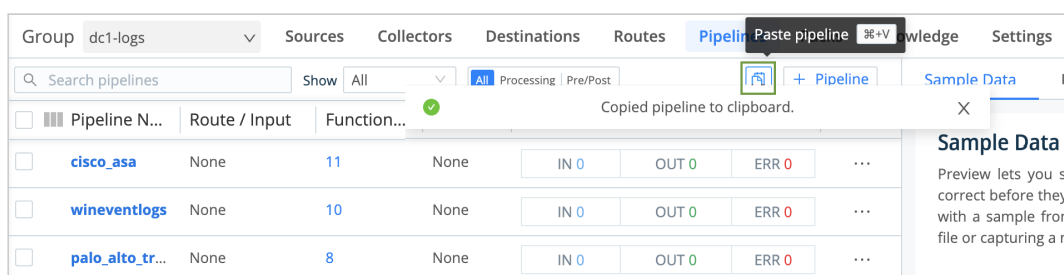
Pipeline Actions

Click a Pipeline's Actions (...) menu to display options for copying or deleting the Pipeline.



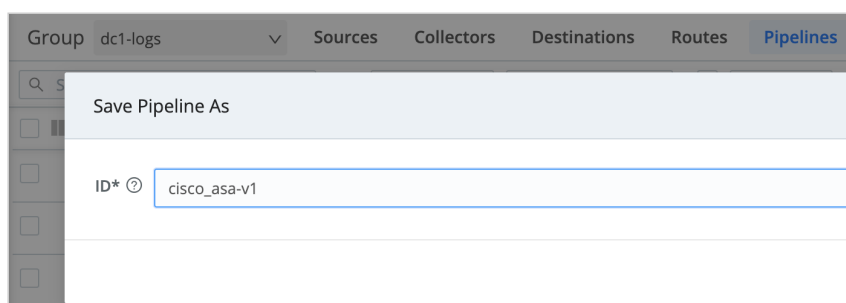
Pipeline > Actions menu

Copying a Pipeline displays the confirmation message and the (highlighted) Paste button shown below.



Paste button for copied Pipeline

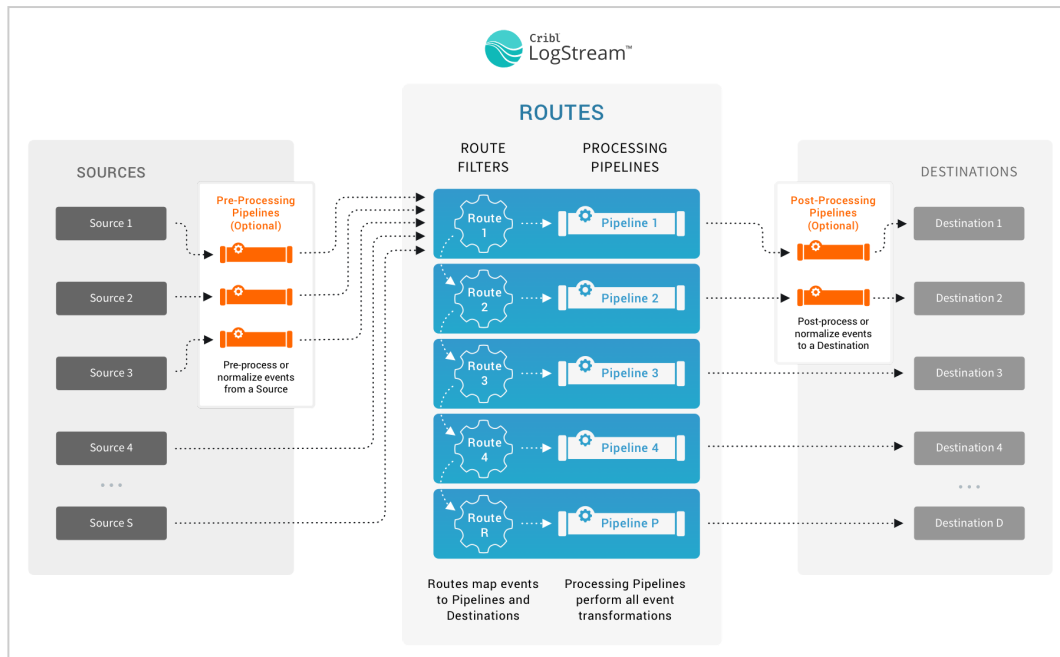
Pasting prompts you to confirm, or change, a modified name for the new Pipeline. The result will be an exact duplicate of the original Pipeline in all but name.



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.



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.

What's Next

➤ [Functions](#)

➤ [Packs](#)

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.

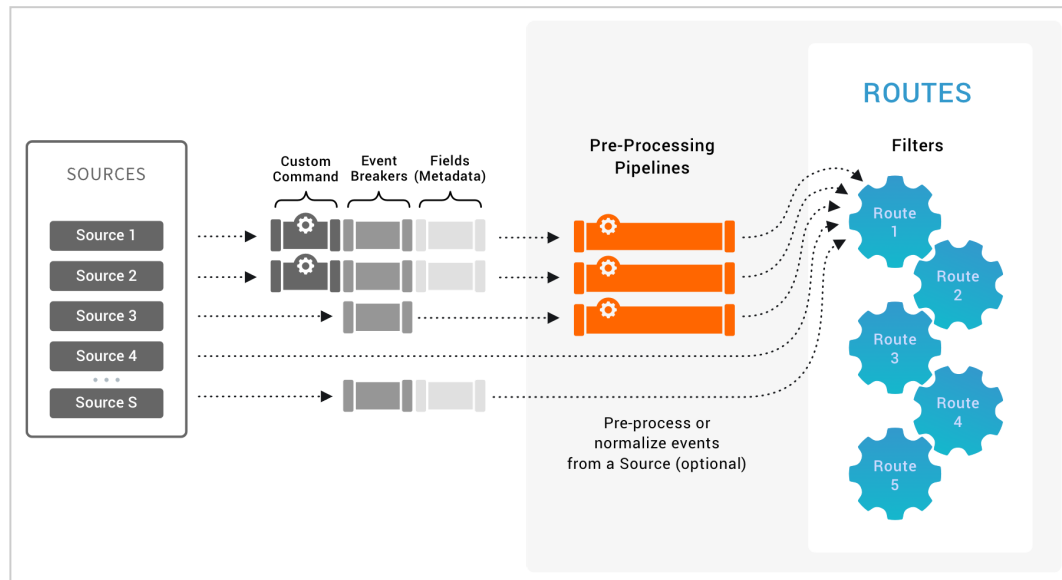
Preview Sample Data

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.

The screenshot shows the LogStream Pipelines interface. At the top, there are tabs for Data, Routes, Pipelines (selected), Knowledge, and System Settings. Below the tabs, the pipeline name 'sample_and_filter' is displayed. A status bar shows 'IN 73.15k', 'OUT 18.72k', and 'ERR 0'. The pipeline is attached to a route named 'Sample and Filter'. The main area shows a stack of five functions:

- 1. **Comment**: Extract the HTTP Status and Sample successes.
- 2. **Regex Extract**: `source=='smart_sample'` (On)
- 3. **Sampling**: `source=='smart_sample'` (On)
- 4. **Filter**: `source=='smart_sample'` (Expanded view below)
- 5. **Drop**: `source=='filter' && !(/\"S+\"s\"S+action=purchase/i.test(_raw))` (On)

The **Filter** function is expanded, showing its configuration:

- Filter**: `source=='smart_sample'`
- Description**: Enter a description
- Final**: ☐ No
- Sampling Rules**:

Filter	Sampling Rate
<code>__status == 200</code>	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.

Functions and Shared-Nothing Architecture

LogStream is built on a shared-nothing architecture, where each Node and its **Worker Processes operate separately, and process events independently of each other**. This means that all Functions operate strictly in a Worker Process context – state is not shared across processes.

This is particularly important to understand for certain Functions that might imply state-sharing, such as [Aggregations](#), [Sampling](#), [Dynamic Sampling](#), [Suppress](#), etc.

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](#)

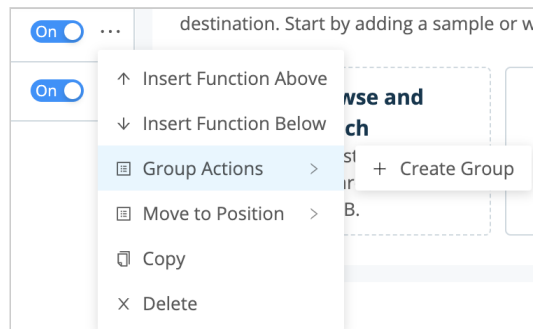
- Serialize / change format (e.g., convert JSON to CSV):
[Serialize](#)
- Convert JSON arrays into their own events:
[JSON Unroll](#), [XML Unroll](#)
- Flatten nested structures (e.g., nested JSON):
[Flatten](#)
- Aggregate events in real-time (i.e. statistical aggregations):
[Aggregations](#)
- Convert 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.

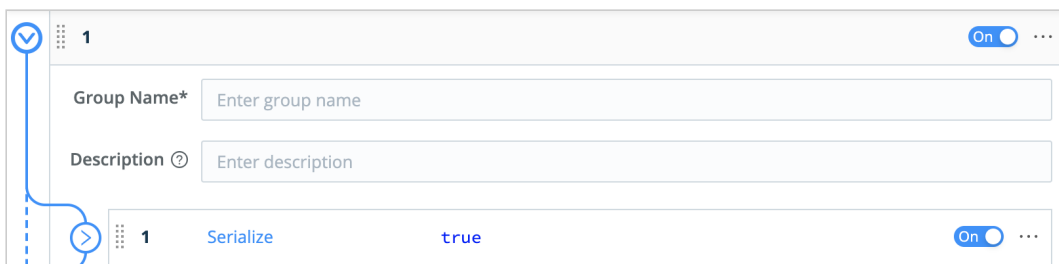
i Function groups work much like [Route groups](#).

To build a group from any Function, click the Function's **⋮** (Options) menu, then select **Group Actions > Create Group**.



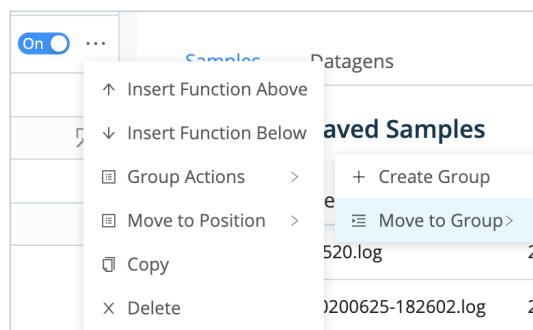
Creating a group

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.

The image shows a web form interface for a group named "Groupy McGroupface". The form has a header bar with a blue checkmark icon, a title "Groupy McGroupface", and a toggle switch set to "On". Below the header, there are two input fields: "Group Name*" with the value "Groupy McGroupface" and "Description ?" with the placeholder text "Enter description". At the bottom of the form, there is a large dashed rectangular box indicating a drag-and-drop target area.

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` . You can access other fields' values via `__e.<fieldName>` .

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

This references https://github.com/d3/d3-time-format#locale_format.

Directives annotated with a (†) symbol might be affected by the locale definition.

%a - abbreviated weekday name. (†)
%A - full weekday name. (†)
%b - abbreviated month name. (†)
%B - full month name. (†)
%c - the locale's date and time, such as %x, %X. (†)
%d - zero-padded day of the month as a decimal number [01,31].
%e - space-padded day of the month as a decimal number [1,31]; equivalent
%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 (%).

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`.

The screenshot displays the LogSteram configuration interface for the 'Auto Timestamp' function. The function is set to 'true' and is applied to the '_raw' source field. The destination field is '_time' and the default timezone is 'Local'. The time expression is `time.getTime() / 1000`. The 'Max Timestamp Scan Depth' is set to 150. The 'Start Scan Offset' is 0 and the 'Default Time' is 'Current Time'. The 'Earliest timestamp allowed' is -420weeks and the 'Future timestamp allowed' is +1week. The interface also shows a sample log event with the following structure:

```
{
  "_raw": "12.130.60.5 - - [23/Apr/2020:13:44:58 +0000] \"GET /static/jquery.js?25SESSIONID=5048590438555SL3659142069FF8145341791ADFF8317519140 HTTP/1.1\" 200 764 \"product.screen?product_id=DP-SAMSGALAX4\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36\"",
  "_time": 1587649498,
  "cribl_breaker": "fallback",
  "cribl_pipe": "beats-conditioning-v1",
  "host": "8c88547db534",
  "index": "cribl",
  "source": "/var/log/httpd/access_log",
  "sourcetype": "access_combined"
}
```

i 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: 210

Click to add **Additional timestamps**:

Regex: `(\d{1,2}\.?\d{2}\.?\d{4}\s\d{1,2}:\d{2}:\d{2}\s\w{2})`

Strptime format: `'%m/%d/%Y %H:%M:%S %p'`

i 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.

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: If enabled, aggregations will be retained for cumulative aggregations when flushing out an aggregation table event. When set to `No` (the default), aggregations will be reset to `0` on flush.

Lag tolerance: The lag tolerance represents the tumbling window tolerance to late events. Must be a valid time string (e.g., `10s`). Must match pattern `\d+[sm]$`.

Idle bucket time limit: The amount of time to wait before flushing a bucket that has not received events. Must be a valid time string (e.g., `10s`). Must match pattern `\d+[sm]$`.

Output Settings

Passthrough mode : 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`.)

Flush on stream close: If enabled (the default), aggregations will flush when an input stream is closed. If set to `No`, the [Time Window Settings](#) will control flush behavior; this can be preferable if (e.g.) your input data consists of many small files.

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.

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 1 hour, Cribl recommends adjusting the **Aggregation memory limit** and/or **Aggregation event limit** to prevent the system from running out of memory.

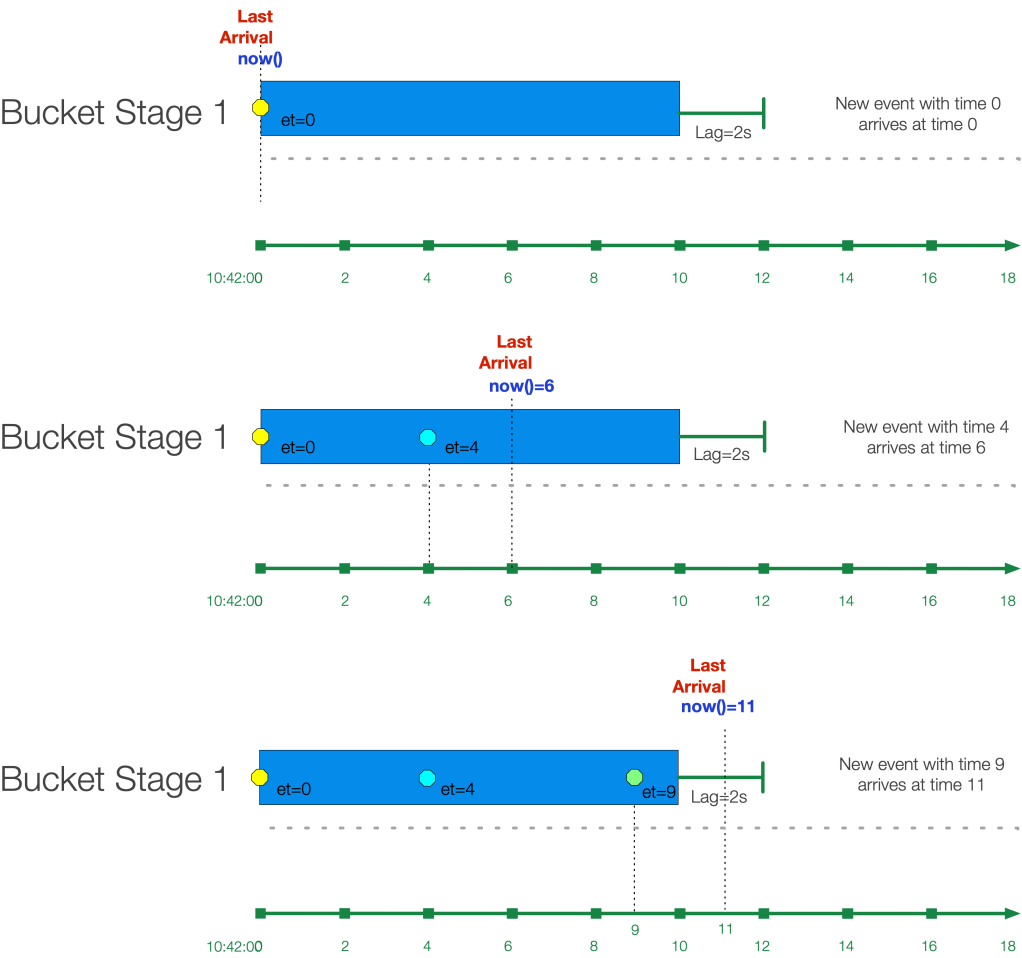
This is especially necessary for high-cardinality data. (Both settings default to unlimited, but we recommend setting defined limits based on testing.)

How Do Time Window Settings Work?

Lag Tolerance

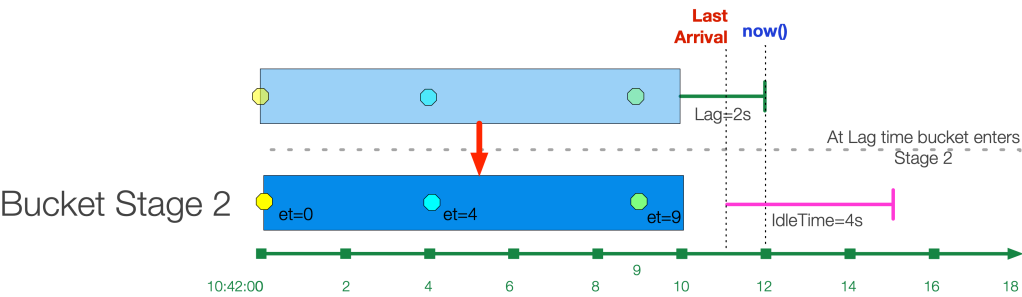
As events are aggregated into windows, there is a good chance that most will arrive later than their event time. For instance, given a `10s` window (`10:42:00 – 10:42:10`), an event with timestamp `10:42:03` might come in 2 seconds later at `10:42:05` .

In several cases, there will also be late, or lagging, events that will arrive **after** the latest time window boundary. For example, an event with timestamp `10:42:04` might arrive at `10:42:12` . Lag Tolerance is the setting that governs how long to wait – after the latest window boundary – and still accept late events.



The "bucket" of events is said to be in Stage 1, where it's still accepting new events, but it's not yet finalized. Notice how in the third case, an event with event time 10:42:09 arrives 1 second past the window boundary at 10:42:11, but it's still accepted because it happens before the lag time expires.

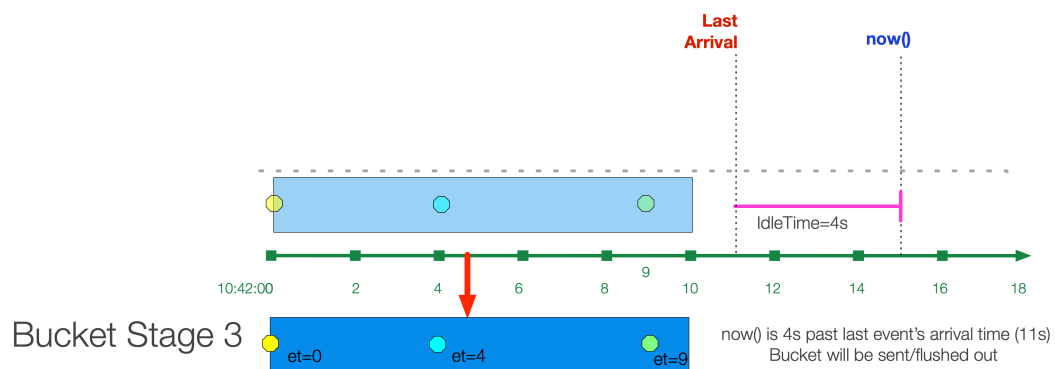
After the lag time expires, the bucket moves to Stage 2.



If the bucket is created from a historic stream, then the bucket is initiated in Stage 2. Lag time is not considered. A "historic" stream is one where the latest time of a bucket is before `now()`. E.g., if the window size is 10s, and `now()`=10:42:42, an event with `event_time=10` will be placed in a Stage 2 bucket with range 10:42:10 - 10:42:20.

Idle Bucket Time Limit

While Lag Tolerance works with event time, Idle Bucket Time Limit works on arrival time (i.e., real time). It is defined as the amount of time to wait before flushing a bucket that has not received events.



After the Idle Time limit is reached, the bucket is "flushed" and sent out of the system.

Examples

Assume we're working with VPC Flowlog events that have the following structure:

```
version account_id interface_id srcaddr dstaddr srcport dstport
protocol packets bytes start end action log_status
```

For example:

```
2 99999XXXXX eni-02f03c2880e4aaa3 10.0.1.70 10.0.1.11 9999 63030
6 6556 262256 1554562460 1554562475 ACCEPT OK
2 496698360409 eni-08e66c4525538d10b 37.23.15.38 10.0.2.232 4373
8108 6 1 52 1554562456 1554562466 REJECT OK
```

Scenario A:

Every 10s, compute sum of `bytes` and output it in a field called `TotalBytes`.

Time Window: 10s

Aggregations: `sum(bytes).as(TotalBytes)`

Scenario B:

Every 10s, compute sum of `bytes` , output it in a field called `TotalBytes` , group by `srcaddr` .

Time Window: 10s

Aggregations: `sum(bytes).as(TotalBytes)`

Group by Fields: `srcaddr`

Scenario C:

Every 10s, compute sum of `bytes` but only where action is `REJECT` , output it in a field called `TotalBytes` , group by `srcaddr` .

Time Window: 10s

Aggregations: `sum(bytes).where(action=='REJECT').as(TotalBytes)`

Group by Fields: `srcaddr`

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](#).

Each Worker Process executes this Function independently on its share of events. For details, see [Functions and Shared-Nothing Architecture](#).

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`

Code

If you need to operate on data in a way that can't be accomplished with LogStream's out-of-the-box Functions, the Code Function enables you to encapsulate your own JavaScript code. This Function is available in LogStream 3.1+, and imposes some restrictions for security reasons.

Restrictions

Generally speaking, anything forbidden in JavaScript [strict mode](#) is forbidden in the context of the Code Function. Specifically, the following are **not allowed**:

- `console` , `eval` , `uneval` , `Function` (constructor) , `Promises` , `setTimeout` , `setInterval` , `global` , `globalThis` , and `window` .

Code Functions **can** include `for` loops, `while` loops, and JavaScript methods such as `map` , `reduce` , `forEach` , `some` , and `every` . For further details, see [Supported JavaScript Options](#).

LogStream's predefined Functions, such as [Eval](#), cover the vast majority of scenarios that users typically need to implement. You should use Code Functions only as a last resort, when you need to construct a complex block of code.

Also, only skilled JavaScript developers should define Code Functions. This is to avoid unintended results – such as creating infinite loops, or otherwise failing to return – that could needlessly add to your throughput burden.

Usage

When added to a Pipeline, the Code Function offers the following configuration options:

Filter: JavaScript filter expression that selects data to be fed through the Function. Defaults to `true` , meaning that all events will be evaluated.

Description: Optionally, add a simple description of this Function.

Final: If true, stops data from being fed to downstream Functions. Defaults to `No` .

Code: The mini-editor where you type your JavaScript code.

Advanced Settings

Maximum number of iterations: The maximum number of iterations per instance of this Code Function. Defaults to `5,000` ; highest allowed value is `10000` .

Notes and Examples

Code Functions always use the special variable `__e` to access the (context) event inside JavaScript expressions.

Possibly the simplest Code Function creates a new field and then assigns it a value:

A Basic Code Function

```
__e['foo'] = 'Hello, Goats!'
```

For more ambitious implementations, see [Code Function Examples](#).

Supported JavaScript Options

With some exceptions, the Code Function supports the options described in the following [MDN JavaScript Guide](#) topics:

- [Expressions and Operators](#)
- [Global variables](#)
- [Control flow and error handling](#)
- [Loops and iteration](#)
- [Functions](#)
- [Numbers and dates](#)

- [Text formatting](#)
- [Regular Expressions](#)
- [Indexed collections](#)
- [Keyed collections](#)
- [Working with Objects](#)

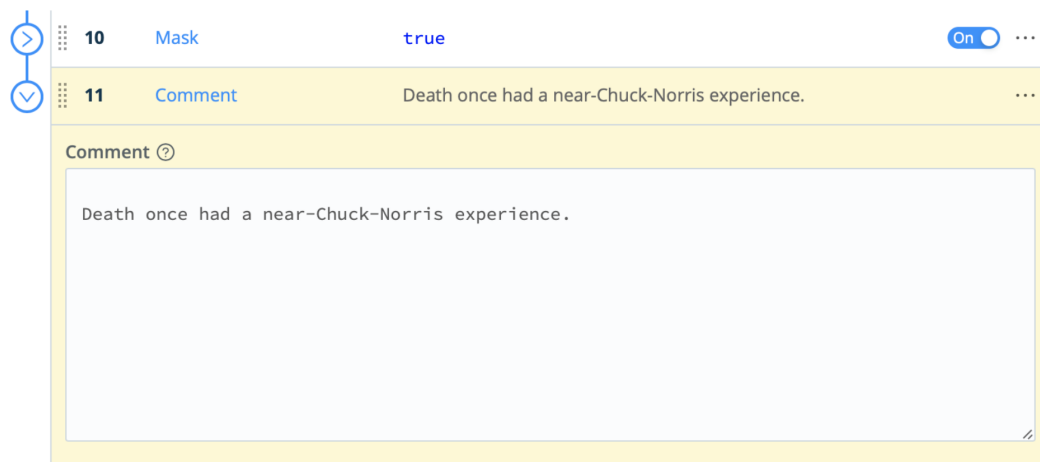
Comment

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.



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...

+ Add Field

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  [Help](#) 









`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.


 2 **DNS Lookup** true ☐ Off


Filter  [Help](#) 








`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	X
⋮	google_ip	reversed_google_ip	X
+ Add field(s)			

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: $\text{Math.ceil}(\text{Math.log}(4000)) = 9$

Period 2 (next 30s) -- Number of events in sample group: 12000 , Sample Rate:
9:1 : Events allowed: 1334
Sample Rate calculation for **next** period: $\text{Math.ceil}(\text{Math.log}(12000)) = 10$

Period 3 (next 30s) -- Number of events in sample group: 2000 , Sample Rate:
10:1 : Events allowed: 200
Sample Rate calculation for **next** period: $\text{Math.ceil}(\text{Math.log}(2000)) = 8$
...

Sample Modes

1. Logarithmic – The sample rate is derived, for each sample group, using a natural log: $\text{Math.ceil}(\text{Math.log}(\text{lastPeriodVolume}))$. This mode is **less aggressive**, and drops fewer events.
2. Square Root – The sample rate is derived, for each sample group, using: $\text{Math.ceil}(\text{Math.sqrt}(\text{lastPeriodVolume}))$. 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

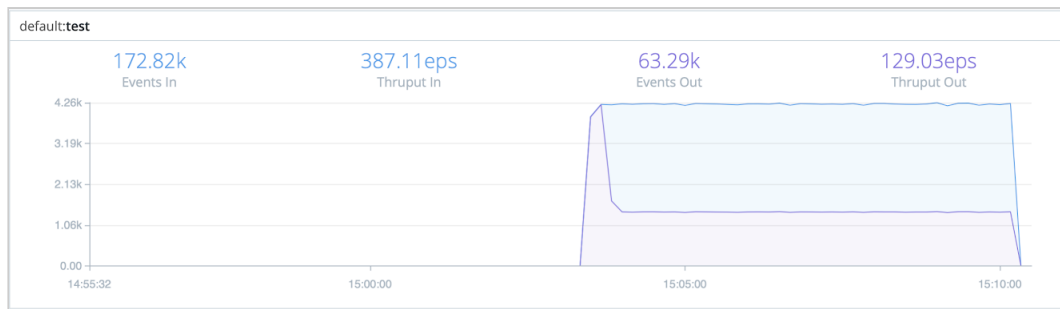
Minimum Events: 3

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](#).

Each Worker Process executes this Function independently on its share of events. For details, see [Functions and Shared-Nothing Architecture](#).

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. Defaults to empty array, which means all fields. Limit to specific fields by typing in their names, separated by hard returns. Supports wildcards (`*`). Supports double-underscore (`__`) internal fields only if individually enumerated – not via wildcards.

Prefix: Prefix string for flattened field names. Defaults to empty.

Depth: Number representing the nested levels to consider for flattening. Minimum `1`. Defaults to `5`.

Delimiter: Delimiter to 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
```

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	<code>__geoip.city.names.en</code>
Country	<code>__geoip.country.names.en</code>
Zip	<code>__geoip.postal.code</code>
Lat	<code>__geoip.location.latitude</code>

Long	<code>__geoip.location.longitude</code>
------	---

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. Click the Expand button at right to open a preview/validation 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 me
```

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 me",  
  "_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": {
```

Event 2:

```
{ "_raw": { "date": "9/25/18 9:10:13.000 PM", "name": "Amrit", "age": 42, "cars": {
```

Event 3:

```
{ "_raw": { "date": "9/25/18 9:10:13.000 PM", "name": "Amrit", "age": 42, "cars": {
```

Event 4:

```
{ "_raw": { "date": "9/25/18 9:10:13.000 PM", "name": "Amrit", "age": 42, "cars": {
```

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 Library](#), 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 modes**, 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 modes** 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
"^([^\s,]+,[^\s,]+,[^\s,]+,THREAT",pan:threat
"^([^\s,]+,[^\s,]+,[^\s,]+,TRAFFIC",pan:traffic
"^([^\s,]+,[^\s,]+,[^\s,]+,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

BEFORE:

```
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2"}
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,THREAT,end,20"}
```

AFTER:

```
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,TRAFFIC,end,2",
 "sourcetype": "pan:traffic"}
{"_raw": "Sep 20 13:03:55 PA-VM 1,2018/09/20 13:03:58,FOOBAR,THREAT,end,20",
 "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

i 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,
  "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,
  "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,
  "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,
  "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. Supports double-underscore (`__`) internal fields only if individually enumerated – not via wildcards.

i Negated terms are 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

+ Add Rule

Apply to Fields ?
_raw x

Mask Function configuration

Result: Vroom vroom!

```

α _raw:
  # age: 42
  allCars: 4 items...
  cars:
    models:
      Trans AM
      Oldsmobile
      Cadillac
      name: GM
    date: 9/25/18 9:10:13.000 PM
    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=90000000000002363, 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.


```

    _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
    cribl_pipe: business_event
    host: 127.0.0.1
    index: cribl
    source: /opt/tibco/tra/apps/ESB/logs/business_event.log
    sourcetype: business_event

```

Event with hashed SSNs

- i** 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. Supports wildcards (`*`) and nested addressing. When empty (the default), Numerify applies to **all** fields. When populated, takes precedence over the **Include expression**.

i Double Negatives

Ignore fields also supports negated terms. 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` ."

Include expression: Optional JavaScript expression to specify fields to numerify. If empty (the default), the Function will attempt to numerify all fields – except those listed in **Ignore fields**, which takes precedence. Use the `name` and `value` global variables to access fields' names/values. (Example: `value != null` .) You can access other fields' values via `__e.<fieldName>` .

Format: Optionally, reformat or truncate the extracted numeric value. Select one of:

- **None:** Applies no reformatting (the default).
- **Floor:** Rounds the number down to the lower adjacent integer (truncates it).
- **Ceil:** Rounds the number up to the higher adjacent integer, removing decimal digits.
- **Round:** Rounds (truncates) the number to a specified number of digits. This option exposes an extra field:
 - **Digits:** Number of digits after the decimal point. Enter a value between 0 – 20 ; defaults to 2 .

Examples

Scenario A:

Assume an event whose text contains a numeric value that must be extracted to perform some numeric analysis. The text looks like this:

```
version=11.5.0.0.1.1588476445
```

We can extract the numeric value by chaining together two Functions:

1. A Regex Extract Function. Set its **Regex** field to `/version=(?<ver>\d+)/` , to capture the first set of digits found in the event string.
2. Then use Numerify.

This captures the substring `11` and converts it to a numeric `11` value.

Scenario B:

Assume email transaction log events like the sample below. The final field is the message's size, in bytes. We want to extract this as a numeric value, for analysis in 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).

The screenshot shows the Cribl Parser configuration interface. At the top, there's a tab labeled '1 Parser' with a status 'true' and a toggle switch 'On'. Below this, the configuration is organized into sections:

- Filter**: A text input field containing 'true'.
- Description**: An empty text input field.
- Final**: A toggle switch set to 'No'.
- Parser Mode***: A dropdown menu showing 'Reserialize'.
- Parser Type***: A dropdown menu showing 'JSON Object'.
- Parser Library**: A dropdown menu showing 'Select from Library'.
- Source Field**: A text input field containing '_raw'.
- List of Fields**: A text input field containing 'Field names'.
- Fields To Keep**: A text input field containing 'Field names'.
- Fields To Remove**: A text input field containing 'values.total.Cxn' and 'startTime'.
- Fields Filter Expression**: A text input field containing '! (name=='level' && value=='info')'.
- Destination Field**: An empty text input 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



#	Event
1	<pre>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 a: 000 b: 001 c: 002 cribl_breaker: Break on newlines cribl_pipe: parser2</pre>

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 by 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:

The screenshot shows the configuration for a Parser Function. The top bar indicates the function is named '1 Parser' and is currently 'true'. The configuration is as follows:

- Filter:** `true`
- Description:** (empty)
- 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:** (empty)

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 Gauge (the default), Counter , or Timer . General definitions that can vary across senders:
 - Gauge : A numeric value that can increase or decrease over time – like a temperature or pressure gauge.
 - Counter : A cumulative numeric value – it can only increase over time.
 - Timer : Generally measures how long a given event type takes (duration), and how often it occurs (frequency).

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
2020-03-05  α  _metric_type: gauge
18:42:52.707  #  _time: 1583451772.707
-05:00  #  _value: 229
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	<code>`metric_name.bytes`</code>	Gauge
packets	<code>`metric_name.packets`</code>	Gauge

Dimensions

Dimensions
<code>action interface_id dstaddr</code>

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
```

i 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,InternalS
```

```

erver,,,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
<code>bytes_sent</code>	<code>`metric.\${host}.bytes_sent`</code>	Counter
<code>bytes_received</code>	<code>`metric.\${host}.bytes_rcvd`</code>	Counter
<code>packets_sent</code>	<code>`metric.\${host}.pkts_sent`</code>	Counter
<code>packets_received</code>	<code>`metric.\${host}.pkts_rcvd`</code>	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 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 of the 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

IN	OUT
1	<pre> raw: <141>2021-06-30T15:45:05.969591-04:00 m087768 filter_instance1[17096]: rppt s=378qs0xcw3 m=1 x=378qs0xcw3-1 mod=mail cmd=msg module=pdr rule=p ass action=continue attachments=0 rcpts=1 routes=Ext... Show more # _time: 1625082305.969 action_XX: continue attachments_XX: 0 cmd_XX: msg cribl_breaker: noBreak1MB cribl_pipe: logicHub duration_XX: 0.301 elapsed_XX: 1.159 guid_XX: [REDACTED] hdr_mid_XX: [REDACTED] hops-ip_XX: 1.2.3.4 m_XX: 1 mod_XX: mail module_XX: pdr qid_XX: [REDACTED] rcpts_XX: 1 routes_XX: External,default_inbound rule_XX: pass s_XX: [REDACTED] size_XX: 4236 spamscore_XX: 0 subject_XX: [External Sender] Re: Blah Blah virusname_XX: </pre>

Example 2 results

Example 3: Multi-Stage Extraction, Complex Events

This example builds on the syntax in [Example 2](#), to tackle 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:"Accep
```

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_directi
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,0=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
_uuid:"001ab86d-d201-4b61-9b64-0fed1a9f059"; 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,0=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_uuid: "001ab86d-d201-4b61-9b64-0fed1a9f059"

```

Example 3 results

- 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.

Examples

Scenario A: Set and Get

This Pipeline demonstrates the use of a pair of Redis Functions. The first Function sets two key-value pairs in Redis. The second Function gets their values, by key, into two corresponding new **Result fields**.

Pipelines > testRedisPipeline

Attach Pipeline to Route

1

Filter

Function

Filter

Show All

IN

100%

100%

100%

Select Fields (0 of 6)

Filter

true

Description

Set keys to Redis

Final

Result field

Command

Key

Args

Result field	Command	Key	Args
Enter result field	set	myfieldAA	420
Enter result field	set	myfieldBB	sample value

+ Add

Redis URL

redis://localhost:6379

> ADVANCED SETTINGS

2

Filter

Function

Filter

Show All

IN

100%

100%

100%

Select Fields (0 of 6)

Filter

true

Description

Read keys from Redis

Final

Result field

Command

Key

Args

Result field	Command	Key	Args
myfield_AA	get	myfieldAA	
myfield_BB	get	myfieldBB	

+ Add

Redis URL

redis://localhost:6379

> ADVANCED SETTINGS

Sample Data

Preview Simple

Preview Full

one-line.log

testRedisPipeline

Run

```

1  [{"raw": {"foo": {"sub_1": "20000", "sub_2": "143", "cr151_1": "160258770005", "index": "asp"}},
2020-10-13
07:40:39.895
00:00
cr151_browser: break on newlines
cr151_pipe: testRedisPipeline
myfield_AA: 420
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'

Scenario B: Multiple-Argument Arrays

This example demonstrates how to configure a Redis Function that supplies an array of multiple arguments to Redis commands (in this example, `lset` and

lrange).

Result field	Command	Key	Args
rs1	rpush	"mylist"	'one'
rs2	rpush	"mylist"	'two'
rs3	rpush	"mylist"	'three'
rs4	lset	"mylist"	[0, 'four']
rs5	lset	"mylist"	[-2, 'five']
rs6	lrange	"mylist"	[0, -1]

Redis Function, arrays of multiple arguments, and sample output

Redis Function

Description: Push arrays of multiple arguments to Redis

Result field: rs1

Command: rpush

Key: "mylist"

Args: 'one'

Result field: rs2

Command: rpush

Key: "mylist"

Args: 'two'

Result field: rs3

Command: rpush

Key: "mylist"

Args: 'three'

Result field: rs4

Command: lset

Key: "mylist"

Args: [0, 'four']

Result field: rs5

Command: lset

Key: "mylist"

Args: [-2, 'five']

Result field: rs6

Command: lrange

Key: "mylist"

Args: [0,-1]

Try This at Home

The Pipeline below contains only this example Function. You can import it into your own LogStream environment, fill in the `url` with your own credentials, and further modify it to meet your needs.

redis-multiple-args.json

```
{
  "id": "redis-multiple-args",
  "conf": {
    "output": "default",
    "groups": {},
    "asyncFuncTimeout": 1000,
    "functions": [
      {
        "filter": "true",
        "conf": {
          "commands": [
            {
              "outField": "rs1",
              "command": "rpush",
              "keyExpr": "\"mylist\"",
              "argsExpr": "'one'"
            },
            {
              "outField": "rs2",
              "command": "rpush",
              "keyExpr": "\"mylist\"",
              "argsExpr": "'two'"
            },
            {
              "outField": "rs3",
              "command": "rpush",
              "keyExpr": "\"mylist\"",
              "argsExpr": "'three'"
            },
            {
              "command": "lset",
              "keyExpr": "\"mylist\"",
              "argsExpr": "[0,'four']",
              "outField": "rs4"
            },
            {
              "outField": "rs5",
              "command": "lset",
              "keyExpr": "\"mylist\"",
              "argsExpr": "[-2,'five']"
            }
          ]
        }
      }
    ]
  }
}
```

```

    {
      "outField": "rs6",
      "command": "lrange",
      "keyExpr": "\"mylist\"",
      "argsExpr": "[0,-1]"
    }
  ],
  "maxBlockSecs": 60,
  "url": "redis://<your-credentials-here>"
},
... ..

```

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.

i This Function cannot operate on internal fields whose names begin with double underscores (`__`).

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. Use the `name` and `value` global variables to access fields' names/values. 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). However, the **Rename fields** strategy will execute first.

Advanced Settings

Parent field wildcard depth: For wildcards specified in **Parent fields**, sets the maximum depth within events to match and rename fields. Enter `0` to match only top-level fields. Defaults to `5` levels down.

Example

Change the `level` field, and all fields that start with `out`, to all-uppercase.

Example event:

```
{
  "inEvents": 622,
  "level": "info",
  "outEvents": 311,
  "outBytes": 144030,
  "activeCxn": 0,
  "openCxn": 0,
  "closeCxn": 0,
  "activeEP": 105,
  "blockedEP": 0
}
```

Rename Fields:

Current name: `level`

New name: `LEVEL`

Rename expression: `name.startsWith('out') ? name.toUpperCase() :`
`name`

Event after Rename:

```
{ "inEvents": 622,  
  "LEVEL": "info",  
  "OUTEVENTS": 311,  
  "OUTBYTES": 144030,  
  "activeCxn": 0,  
  "openCxn": 0,  
  "closeCxn": 0,  
  "activeEP": 105,  
  "blockedEP": 0  
}
```

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

+ Add Rule

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.

- i** Each Worker Process executes this Function independently on its share of events. For details, see [Functions and Shared-Nothing Architecture](#).

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":"init`

```
ializing 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.

Seeing the Results

If you've enabled **Drop suppressed events**, such events will be omitted from logs as they exit this Function. However, the next event allowed through will include a `suppressCount: N` field, whose N value indicates the number of events dropped in the preceding **Suppression period**.

```
9      {} _raw:
2019-05-10      α channel: input:local-redacted
11:55:33.371-04:00      α level: info
                        α message: closed connection
                        α src: 127.0.0.1:63964
                        α time: 2019-05-10T15:55:33.371Z
# _time: 1557503733.371
α cribl_pipe: Dedupe
# suppressCount: 42
```

suppressCount shows number of events dropped

Examples

In the examples below, **Filter** is the Function-level Filter expression:

1. Suppress by the value of the `host` field:

Filter: `true`

Key expression: `host`

Number to allow: 1

Suppression period (sec): 30

Using a datagen sample as a source, generate at least 100 events over 2 minutes.

Result: One event per unique `host` value will be allowed in every 30s. Events without a `host` field will **not** be suppressed.

2. Suppress by the value of the `host` and `port` tuple:

Filter: `true`

Key expression: ``${host}:${port}``

Number to allow: 1

Suppression period (sec): 300

Result: One event per unique `host : port` tuple value will be allowed in every 300s.

⚠ Suppression will **also** apply to events without a `host` or a `port` field. The reason is that if `field` is not present, ``${field}`` results in the literal `undefined`.

3. To **guarantee** that suppression applies **only** to events with `host` and `port`, check for their presence using a Filter:

Filter: `host!=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.

i For further use cases, see Cribl's [Streaming Data Deduplication with Cribl](#) blog post.

Each Worker Process executes this Function independently on its share of events. For details, see [Functions and Shared-Nothing Architecture](#).

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,InternalS
erver,,,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,InternalS
erver,,,not-applicable,vsys1,inside,z1-FW-Transit,ethernet1/2,,All
traffic,2019/10/09
10:19:15,0,1,63712,443,0,0,0x0,udp,deny,60,60,0,1,2019/10/09
10:19:15,0,any,0,0123456789,0x0,Netherlands,10.0.0.0-
10.255.255.255,0,1,0,policy-deny,0,0,0,0,,DMZ-internal,from-
policy,,,0,,0,,N/A,0,0,0,0,1202585d-b4d5-5b4c-aaa2-
d80d77ba456e,0", "_time":1593185606.965,"host":"127.0.0.1", "receive
_time":"2019/10/09
10:19:15", "source_port":"63712", "destination_port":"443", "bytes_re
ceived":"0", "packets_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 `timestamppos` and `timeendpos` fields. It removes `timestamppos` 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 `timestamppos` and `timeendpos`) from `_raw`, and stash it in a field called `time_field`.

Field name: `time_field`

Example event before:

```
{"_raw": "2020-05-22 16:32:11,359 Event [Event=UpdateBillingProvQuote, tim  
"timestamppos":0,
```

```
"timeendpos":23
}
```

To create this example payload, we selected **Sample Data > Paste**, pasted the `_raw` field's original contents into the resulting modal, and then added the two required position fields:

Example event setup

Example event after:

```
{"_raw": "Event [Event=UpdateBillingProvQuote, timestamp=1581426279, prope  
"time_field":"2020-05-22 16:32:11,359"  
}
```

In the Preview pane's **OUT** view, the original timestamp has been removed from `_raw`, and lifted into the new `time_field` we specified in the Function. The `timestartpos` and `timeendpos` fields have been removed.

Example event, saved and transformed

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:

SampleEvent

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.5	38000	5356	?	Ss	2018	2:02	/lib/syst
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	[ksoftirq
root	5	0.0	0.0	0	0	?	S<	2018	0:00	[kworker/
root	7	0.0	0.0	0	0	?	S	2018	3:55	[rcu_sche
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/syst
------	---	-----	-----	-------	------	---	----	------	------	-----------

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	[ksoftirq
------	---	-----	-----	---	---	---	---	------	------	-----------

Event 5:

root	5	0.0	0.0	0	0	?	S<	2018	0:00	[kworker/
------	---	-----	-----	---	---	---	----	------	------	-----------

Event 6:

root	7	0.0	0.0	0	0	?	S	2018	3:55	[rcu_sche
------	---	-----	-----	---	---	---	---	------	------	-----------

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_
# TYPE perf_192_168_1_248_bytes_rcvd counter
```



```
metric_192_168_1_248_bytes_rcvd {destination_ip="160.177.222.249",inbound_  
  
# TYPE perf_192_168_1_248_pkts_sent counter  
metric_192_168_1_248_pkts_sent {destination_ip="160.177.222.249",inbound_i  
  
# TYPE perf_192_168_1_248_pkts_rcvd counter  
metric_192_168_1_248_pkts_rcvd {destination_ip="160.177.222.249",inbound_i
```


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-
2020-06-29      0000-0000-000000000000 ssl_server_cert_status="Not Checked" dest_ip=8.8.8.8 sec_int
12:51:03.551      el_event=No mac_address=00:00:0... Show more
-07:00      # _time: 1593460263.551
      α app_proto: HTTPS
      α client_app: SSL client
      α client_version:
      α connection_id: 21378
      α cribl_breaker: Break on newlines
      α cribl_pipe: rev_dns
      α dest_autonomous_system: 0
      α dest_bytes: 3746
      α dest_host: dns.google
      α dest_ip: 8.8.8.8
```

Collector Sources

LogStream Collectors are a special group of inputs. Unlike other [Sources](#), Collectors are designed to handle intermittent, rather than continuous, data import. You can use Collectors to dispatch on-demand ("ad hoc") collection tasks that fetch, or "replay" (re-ingest), data from local or remote locations.

Collectors also support **scheduled** periodic collection jobs – recurring tasks that can make batch collection of stored data more like continual processing of streaming data. You configure Collectors prior to, and independently from, your configuration of [ad hoc versus scheduled](#) collection runs.

Collectors are integral to Cribl LogStream's larger story about optimizing your data throughput. Send full-fidelity log and metrics data ("everything") to low-cost storage, and then use LogStream Collectors to selectively route ("replay") only needed data to your systems of analysis.

□ Collector Resources

- Video introduction to [Data Collection](#), under 2 minutes.
- Video introduction to [Data Collection Scheduling](#), under 2 minutes.
- Free, fully functional, interactive try-out of Collectors in Cribl's [Data Collection & Replay sandbox](#).
- [Using S3 Storage and Replay](#) walk-through to set up your own replay.

Collector Types

Cribl LogStream currently provides the following Collector options:

- [Filesystem/NFS](#) – enables data collection and replay from local or remote filesystem locations.

- [Azure Blob](#) – enables data collection and replay from Azure Blob Storage objects.
- [Google Cloud Storage](#) – enables data collection and replay from Google Cloud Storage buckets.
- [S3](#) – enables data collection and replay from Amazon S3 buckets or S3-compatible stores.
- [Script](#) – enables data collection and replay via custom scripts.
- [REST](#) – enables data collection and replay via REST API calls. Provides four Discover options, to support progressively more complex (and dynamic) item enumerations.

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 either [scheduled-job-specific](#) or [run-specific](#) settings – such as the run Mode (Preview, Discovery, or Full Run), the Filter expression to match the data against, the time range, 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**
- Select **Monitoring** (side or top nav) > **System** > **Job Inspector** to see the results of recent collection runs. You can filter the display by Worker Group (in [distributed deployments](#)), and by run type and run timing.

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](#), you configure Collectors at the Worker Group level, and Worker Nodes execute the tasks. However, the Leader Node oversees the task distribution, and tries to maintain a fair balance across jobs.

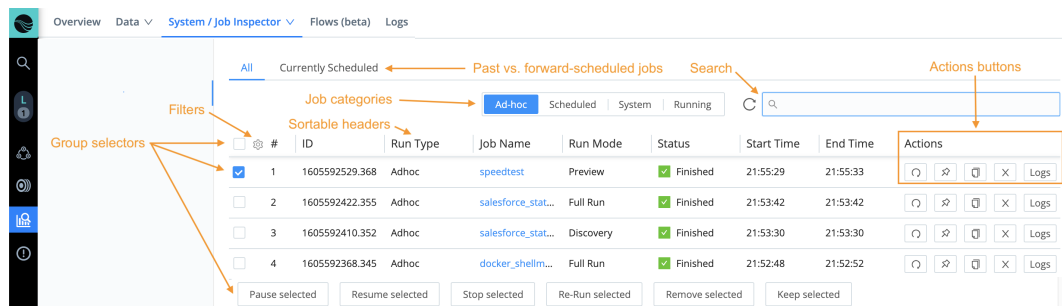
When Workers ask for tasks, the Leader will normally try to assign the next task from a job that has the least tasks in progress. This is known as "Least-In-Flight Scheduling," and it provides the fairest task distribution for most cases. If desired, you can change this default behavior by opening global ⚙ **Settings** (lower left) > **General Settings** > **Job Limits**, and then setting **Job Dispatching** to **Round Robin**.

i More generally: In a distributed deployment, you configure Collectors and their jobs on individual Worker Groups. But you configure Collectors' resource allocation globally in the Leader's global ⚙ **Settings** (lower left) > **General Settings** > **Job Limits** section.

Because the Leader manages Collectors' state, if the Leader instance fails, Collection jobs will fail as well. (This is unlike other [Sources](#), where Worker Groups can continue autonomously receiving incoming data if the Leader goes down.)

Monitoring and Inspecting Collection Jobs

Select **Monitoring** (side or top nav) > **System** > **Job Inspector** to view and manage pending, in-flight, and completed collection jobs and their tasks.



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 forward-scheduled for future execution – including their cron schedule details, last execution, and next scheduled execution. Click **All** to see all jobs initiated in the past, regardless of completion status.
- **Job categories (buttons)**: Select among **Ad hoc**, **Scheduled**, **System**, and **Running**. (At this level, **Scheduled** means scheduled jobs already running or finished.)
- **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).

 Last updated by: Dritan Bitincka

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
➤ Azure Blob Storage
➤ Google Cloud Storage
➤ S3
➤ Script
➤ REST / API Endpoint
➤ Scheduling and Running
➤ Using S3 Storage and Replay

Filesystem/NFS

Cribl LogStream supports collecting data from a local or a remote filesystem location.

How the Collector Pulls Data

When you run a Filesystem/NFS Collector in Discovery mode, the first available Worker returns the list of available files to the Leader Node.

In Full Run mode, the Leader distributes the list of files to process across 1 to N Workers as evenly as possible, based on file size. Each Worker then streams the files from the Filesystem location to itself.

Configuring a Filesystem Collector

From the top nav of a LogStream instance or Group, select **Sources**, then select **Collectors > Filesystem** from the **Data Sources** page's tiles or the **Sources** left nav. Click + **Add New** to open the **Filesystem > New Collector** modal, which provides the following options and fields.

- i** 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.
-

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., DysonV11Roomba960 .

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
<code>/var/log/\${foobar}</code>	<code>foobar: {program: value.split('.') [0]}</code>	<code>/ var/log/syslog.1</code>	<code>{program syslog, foobar: syslog.1}</code>

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](#).

Destructive: If set to `Yes` , the Collector will delete files after collection. Defaults to `No` .

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.

 Last updated by: Dritan Bitincka

What's Next

➤ [Scheduling and Running](#)

Azure Blob Storage

Cribl LogStream 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 **Sources**, then select **Collectors > Azure Blob** from the **Data Sources** page's tiles or the **Sources** left nav. Click + **Add New** to open the **Azure Blob > New Collector** modal, which provides the following options and fields.

i 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.

LogStream supports data collection and replay from Azure's **hot** and **cool** access tiers, but not from the **archive** tier – whose stated retrieval lag, up to several hours, cannot guarantee data availability.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `azure_42-a` .

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
<code>/var/log/\${foobar}</code>	<code>foobar: {program: value.split('.') [0]}</code>	<code>/ var/log/syslog.1</code>	<code>{program syslog, foobar: syslog.1}</code>

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

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Use this default option to enter your Azure Storage connection string directly. Exposes a **Connection string** field for this purpose. (If left blank, LogStream will fall back to `env.AZURE_STORAGE_CONNECTION_STRING` .)
- **Secret:** This option exposes a **Connection string (text secret)** drop-down, in which you can select a stored secret that references an Azure Storage connection string. The secret can reside in LogStream's [internal secrets manager](#) or (if enabled) in an external KMS. A **Create** link is available if you need to generate a new secret.

Connection String Format

Either authentication method uses an Azure Storage connection string in this format:

```
DefaultEndpointsProtocol=[http|https];AccountName=
<your-account-name>;AccountKey=<your-account-key>
```

A fictitious example, using Microsoft's recommended HTTPS option, is:

```
DefaultEndpointsProtocol=https;AccountName=storagesample;AccountK
ey=12345678...32
```

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.

 Last updated by: Dritan Bitincka

Replay

See these resources that demonstrate how to replay data from object storage. Both are written around Amazon S3-compatible stores, but the general principles apply to Azure blobs as well:

- [Data Collection & Replay sandbox](#): Step-by-step tutorial, in a hosted environment, with all inputs and outputs preconfigured for you. Takes about 30 minutes.
- [Using S3 Storage and Replay](#): Guided walk-through on setting up your own replay.

What's Next

➤ [Scheduling and Running](#)

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 **Sources**, then select **Collectors > Google Cloud Storage** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Google Cloud Storage > New Collector** modal, which provides the following options and fields.

- i** 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.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `gcs_24-7` .

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
<code>/var/log/\${foobar}</code>	<code>foobar: {program: value.split('.') [0]}</code>	<code>/ var/log/syslog.1</code>	<code>{program syslog, foobar: syslog.1}</code>

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.

Google Cloud Roles and Permissions

Your Google Cloud service account will need at least the following roles and permissions.


Roles

- `roles/storage.legacyBucketReader`
- `roles/storage.legacyObjectReader`

Permissions

- `storage.buckets.get`
- `storage.objects.get`
- `storage.objects.list`

For additional details, see the Google Cloud [Access Control](#) topic.

 Last updated by: Dritan Bitincka

Replay

See these resources that demonstrate how to replay data from object storage. Both are written around Amazon S3-compatible stores, but the general principles apply to Google Cloud buckets as well:

- [Data Collection & Replay sandbox](#): Step-by-step tutorial, in a hosted environment, with all inputs and outputs preconfigured for you. Takes about 30 minutes.
- [Using S3 Storage and Replay](#): Guided walk-through on setting up your own replay.

What's Next

➤ [Scheduling and Running](#)

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.

Also see our [Using S3 Storage and Replay](#) guided walk-through in this documentation.

How the Collector Pulls Data

When you run an S3 Collector in Discovery mode, the first available Worker returns the list of available files to the Leader Node.

In Full Run mode, the Leader distributes the list of files to process across 1 to N Workers as evenly as possible, based on file size. Each Worker then streams the files from the S3 bucket/path to itself.

- ⚠ LogStream does **not** support data preview, collection, or replay from S3 Glacier or Deep Glacier storage classes, whose stated retrieval lags (variously minutes to 48 hours) cannot guarantee data availability when the Collector needs it.

Configuring an S3 Collector

From the top nav of a LogStream instance or Group, select **Sources**, then select **Collectors > S3** from the **Data Sources** page's tiles or the **Sources** left nav.

Click + **Add New** to open the **S3 > New Collector** modal, which provides the following options and fields.

- i** 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.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `Attic42TreasureChest` .

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
<code>/var/log/\${foobar}</code>	<code>foobar: {program: value.split('.') [0]}</code>	<code>/ var/log/syslog.1</code>	<code>{program syslog, foobar: syslog.1}</code>

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 the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials. 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.

 Last updated by: Dritan Bitincka

What's Next

- Scheduling and Running
- Using S3 Storage and Replay

Script

Cribl LogStream supports flexible data collection configured by your custom scripts.

How the Collector Pulls Data

When you run a Script Collector in Discovery mode, the first available Worker returns one line of data per discovered item. Each item (line) turns into a Collection task on the Leader Node.

In Full Run mode, the Leader passes the item to collect into the script in the `$CRIBL_COLLECT_ARG` variable, and spreads collection across all available Workers.

Configuring a Script Collector

From the top nav of a LogStream instance or Group, select **Sources**, then select **Collectors > Script** from the **Data Sources** page's tiles or the **Sources** left nav. Click + **Add New** to open the **Script > New Collector** modal, which provides the following options and fields.

- i** 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.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID: Unique ID for this Collector. E.g., `sh2GetStuff` .

Discover script: Script to [discover](#) which objects/files to collect. This script should output one task per line in `stdout`. Discovery is especially important in a [distributed deployment](#), where the Leader must track all tasks, and must guarantee that each is run by a single Worker. See [Examples](#) below.

Collect script: Script to perform data collections. Pass in tasks from the Discover script as `$CRIBL_COLLECT_ARG`. Should output results to `stdout`.

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

Telemetry Collector

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`

S3 Collector

In this example, the Discover script retrieves file names from a specified Amazon S3 bucket, and then writes them (one per line) to the standard output. The Collect script processes each line as its `$CRIBL_COLLECT_ARG` , and uses `zcat` to decompress the buckets' data.

Collector type: Script

Discover script: `laws s3api list-objects --bucket <bucket-name> --prefix <subfolder>/ --query 'Contents[].Key' --output text`

Collect script: `aws s3 cp s3://<bucket-name>/$CRIBL_COLLECT_ARG - | zcat -f`


Simple Collector

This example essentially spoofs the Discover script with an `echo` command, which simply announces what the Collect script (itself simple) will do.

Collector type: `Script`

Discover script: `echo "speedtest"`

Collect script: `speedtest --json`

 Last updated by: Dritan Bitincka

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 **Sources**, then select **Collectors > REST** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **REST > New Collector** modal, which provides the following options and fields.

- i** 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.

Collector Settings

The Collector Settings determine how data is collected before processing.

Collector ID

Unique ID for this Collector. E.g., `rest42json` .

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 types** from simplest to most-complex.

- **Discover type: None** matches cases where one simple API call will retrieve all the data you need. This default option suppresses the Discover stage. Use the modal's **Collect** section to specify the endpoint and other details. (Example: Collect a list of configured LogStream Pipelines.)
- **Discover type: Item List** matches cases where you want to enumerate a known list of **Discover items** to retrieve. (Examples: Collect network traffic data that's tagged with specific subnets; or collect weather data for a known list of ZIP codes.) Discovery will return one Collect task per item in the list, and LogStream will spread each of those Collect tasks across all available Workers.
- **Discover type: JSON Response** provides a **Discover result** field where you can (optionally) define Discover tasks as a JSON array of objects. Each entry returned by Discover will generate a Collect task, and LogStream will spread each of those Collect tasks across all available Workers. (Example: Collect data for specific geo locations in the [National Weather Service API](#)'s stream of worldwide weather data. This particular API requires multiple parameters in the request URL – latitude, longitude, etc. – so **Item List** discovery 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-and-then-Collect architecture. (Example: Make a REST call to get a list of available log files, then run Collect against each of those files.) Each item returned will generate a Collect task, and LogStream will spread each of those Collect tasks across all available Workers. As of LogStream 3.0.2, this Discover type supports XML responses.

Collect Settings (Common)

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 are: 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 headers 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

Use this drop-down list to select the pagination scheme for collection results. Defaults to `None`. The other options are:

Response Body Attribute / Response Header Attribute

Select `Response Body Attribute` to extract a value from the response body that identifies the next page of data to retrieve.

Select `Response Header Attribute` to extract this next-page value from the response header. Either of these selections exposes 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.

RFC 5988 – Web Linking

Select this option with APIs that follow [RFC 5988](#) conventions to provide the next-page link in a header. This selection exposes three additional fields:

Next page relation name: Header substring that refers to the next page in the result set. Defaults to `next`, corresponding to the following example link header:

```
<https://myHost/curPage>; rel="self" <https://myHost/nextPage>;  
rel="next"
```

Current page relation name: Optionally, specify the relation name within the link header that refers to the current result set. In this same example, `rel="self"` refers to the current page of results:

```
<https://myHost/curPage>; rel="self" <https://myHost/nextPage>;  
rel="next"
```

Max pages: The maximum number of pages to retrieve. Defaults to `50` pages. 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, use the buttons to select one of these options:

- **None:** Don't use authentication. Compatible with REST servers like AWS, where you embed a secret directly in the request URL.
- **Basic:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret):** Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or click **Create** to configure a new secret.
- **Login:** Enables you to specify several credentials, then perform a POST to an endpoint during the Discover operation. The POST response returns a token, which LogStream uses for later Collect operations.

Login Authentication

Selecting **Login** exposes the following additional fields:

- **Login URL:** URL for the login API call, which is expected to be a POST call.
- **Username:** Login username.
- **Password:** Login password.
- **POST Body:** Template for POST body to send with the login request. The `${username}` and `${password}` variables specify the corresponding credentials' locations in the message.
- **Token Attribute:** Path to the token attribute in the login response body. Supports nested attributes.

- **Authorize Expression:** JavaScript expression used to compute the Authorization header to pass in Discover and Collect calls. Uses `${token}` to reference the token obtained from the login POST request.
 - **Login (credentials secret):** Like **Login** except that you specify a secret which references the credentials, rather than the credentials themselves. Select a stored text secret in the resulting **Credentials secret** drop-down, or click **Create** to configure a new secret.
-

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.

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Other Discover Types

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 are: 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.

 Last updated by: Dritan Bitincka

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 an unintended LogStream shutdown will automatically resume upon LogStream restart.


But regardless of this configuration, if you **explicitly** restart or stop LogStream, this will cancel any currently running jobs. This applies to executing the `./cribl restart` or `./cribl stop` CLI commands, as well as to selecting the UI's global ⚙ **Settings** (lower left) > **Controls** > **Restart** option.

A **scheduled** job interrupted by a shutdown (whether explicit or unintended) 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**

□ See [Job Limits](#) for details on these and other limits that you can set in global **Settings**.

When the above limits delay a Skippable job:

- The Skippable job will be granted slightly higher priority than non-Skippable jobs.
- If the job receives resources to run before its next scheduled run, 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:

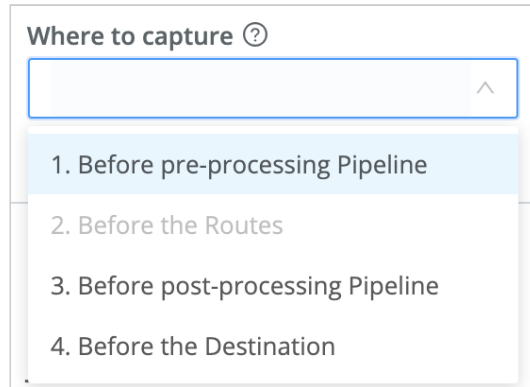
Capture time (sec): Maximum time interval (in seconds) to collect data.

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 .



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: `-1h` , `-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: <code>-</code> for times in the past, <code>+</code> for times in the future, or omit with <code>now</code> .
<time_integer>	Specify any integer, or omit with <code>now</code> .
<time_unit>	Specify the <code>now</code> constant, or one of the following abbreviations: <code>s[econds]</code> , <code>m[inutes]</code> , <code>h[ours]</code> , <code>d[ays]</code> , <code>w[EEKS]</code> , <code>mon[ths]</code> , <code>q[uarters]</code> , <code>y[ears]</code> .
@<snap-to_time_unit>	Optionally, you can append the <code>@</code> modifier, followed by any of the above <code><time_unit></code> 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 `strftime` 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	Matcl
<code>/my/path/\${_time:%Y}/\${_time:%m}/\${_time:%d}/...</code>	<code>/my/</code>
<code>/my/path/\${_time:year=%Y}/\${_time:month=%m}/\${_time:date=%d}/...</code>	<code>/my/</code>
<code>/my/path/\${_time:%Y-%m-%d}/...</code>	<code>/my/</code>

What's Next

➤ [Using S3 Storage and Replay](#)

Job Limits

You can configure global limits that optimize the execution of all Collectors and scheduled jobs (including LogStream system tasks).

General Settings

API Server Settings

General

TLS

Advanced

Default TLS Settings

Display Settings

Custom Login Page

Limits

Job Limits

Proxy Settings

Upgrade & Share Settings

Concurrent Job Limit

10

Concurrent System Job Limit

10

Concurrent Scheduled Job Limit

-2

Concurrent Task Limit

2

Concurrent system Task Limit

1

Max Task Usage Percentage

0.5

Task Poll Timeout

60000

Artifact Reaper Period

30m

Finished Job Artifacts Limit

100

Finished Task Artifacts Limit

500

Manifest Flush Period

100

Manifest Max Buffer Size

1000

Manifest Reader Buffer Size

4kb

Job Dispatching

Least In Flight Tasks

Job Timeout

0

Task Heartbeat Period

60

Job Limits settings

Limits Available

The following controls are available at global ⚙ **Settings** (lower left) > **General Settings > Job Limits**.

□ In a [distributed deployment](#), these limits are set on, and deployed from, the Leader. They are applied at the Worker Group level (except where noted), and trickle down to individual Worker Processes in the group. Task limits are applied at the Worker Process level.

In a [single-instance deployment](#), these limits are set on the single instance, and apply to all its Worker Processes.

Job Limits

Concurrent Job Limit: The total number of jobs that can run concurrently. Defaults to 10 .

i If you see jobs being skipped, this indicates that the **Concurrent Job Limit** for this Group has been reached or exceeded. Here, you need to increase this limit to reduce the number of skippable jobs. Note that, for resource-intensive jobs, this might trigger a need to deploy more Worker Nodes.

Concurrent System Job Limit: The total number of **system** jobs that can run concurrently. Defaults to 10 .

Concurrent Scheduled Job Limit: The total number of **scheduled** jobs that can run concurrently. This limit is set as an offset relative to the **Concurrent Job Limit**. Defaults to -2 .

Task Limits

Concurrent Task Limit: The total number of tasks that a Worker Process can run concurrently. Defaults to 2 .

Concurrent System Task Limit: The number of system tasks that a Worker Process can run concurrently. Defaults to 1 .

Max Task Usage Percentage: Value, between 0 and 1, representing the percentage of total tasks on a Worker Process that any single job may consume. Defaults to 0.5 (i.e., 50%).

Task Poll Timeout: The number of milliseconds that a Worker's task handler will wait to receive a task, before retrying a request for a task. Defaults to 60000 (i.e., 60 seconds).

Completion Limits

Artifact Reaper Period: Interval on which LogStream attempts to reap jobs' stale disk artifacts. Defaults to 30m.

Finished Job Artifacts Limit: Maximum number of finished job artifacts to keep on disk. Defaults to 100.

Finished Task Artifacts Limit: Maximum number of finished task artifacts to keep on disk, per job, on each Worker Node. Defaults to 500.

Task Manifest and Buffering Limits

Manifest Flush Period: The rate (in milliseconds) at which a job's task manifest should be refreshed. Defaults to 100 ms.

Manifest Max Buffer Size: The maximum number of tasks that the task manifest can hold in memory before flushing to disk. Defaults to 1,000.

Manifest Reader Buffer Size: The number of bytes that the task manifest reader should pull from disk. Defaults to 4kb.

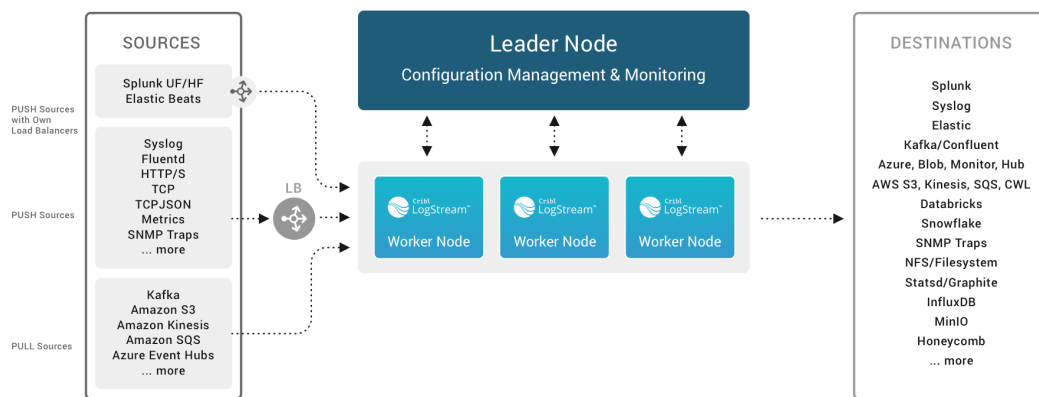
Job Dispatching: The method by which tasks are assigned to Worker Processes. Defaults to Least In-Flight Tasks, to optimize available capacity. Round Robin is also available.

Job Timeout: Maximum time a job is allowed to run. Defaults to 0, for unlimited time. Units are seconds if not specified. Sample entries: 30, 45s, 15m.

Task Heartbeat Period: The heartbeat period (in seconds) for tasks to report back to the Leader/API. Defaults to 60 seconds.

Sources

Cribl LogStream can receive continuous data input 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:

- [Syslog](#)
- [TCP JSON](#)
- [Splunk TCP](#)
- [Splunk HEC](#)
- [Amazon Kinesis Firehose](#)
- [Prometheus Remote Write](#)
- [HTTP/S \(Bulk API\)](#)
- [Raw HTTP/S](#)
- [Elasticsearch API](#)
- [Metrics](#)
- [SNMP Trap](#)

- [TCP \(Raw\)](#)
- [Grafana](#)
- [Loki](#)
- [AppScope](#)

Data from these Sources is normally sent to a set of LogStream Workers through a load balancer. Some Sources, such as [Splunk](#) forwarders, have native load-balancing capabilities, so you should point these directly at LogStream.

PULL Sources

Supported Sources that Cribl LogStream **fetches** data from:

- [Amazon Kinesis Streams](#)
 - [Amazon SQS](#)
 - [Amazon S3](#)
 - [Google Cloud Pub/Sub](#)
 - [Azure Event Hubs](#)
 - [Azure Blob Storage](#)
 - [Office 365 Services](#)
 - [Office 365 Activity](#)
 - [Office 365 Message Trace](#)
 - [Prometheus Scraper](#)
 - [Kafka](#)
-

Internal Sources

Sources that are **internal** to Cribl LogStream:

- [Datagen](#)
 - [Cribl Internal](#)
-

Configuring and Managing Sources

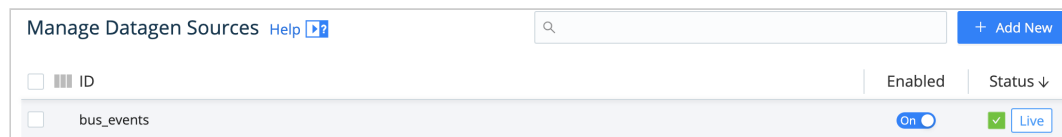
For each Source *type*, you can create multiple definitions, depending on your requirements.

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**.

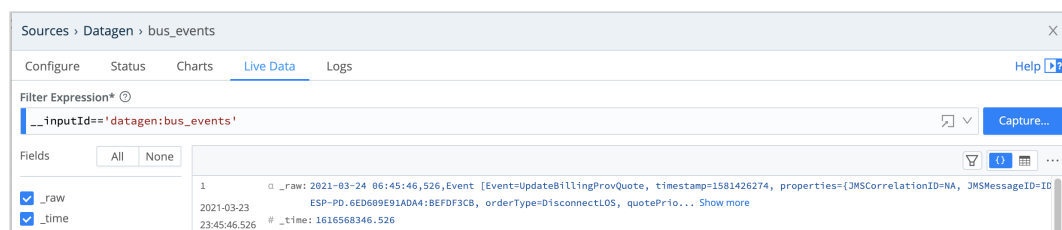
Capturing Source Data

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.



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.



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 **[Push >] 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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (TCP Only)** tab's **Enabled** slider to `Yes`, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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: Enable 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.

TCP Event Delimiter

Within TCP streams, LogStream's Syslog Source currently supports only non-transparent framing to split multiple events, according to [RFC 6587, section 3.4.2](#). Specifically, it supports only `\n` as the trailer character.

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 **[Push >] Splunk > Splunk TCP** from the **Data Sources** page's tiles, or from 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.

 In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes` , nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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: Enable 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.

Fields for this Source:

- __inputId
 - __srcIpPort
 - __source
-

Configuring a Splunk Forwarder

To configure a Splunk forwarder (UF, HF) use the following sample [outputs.conf](#) stanzas:

```
outputs.conf(LogStream)    outputs.conf(LogStream Cloud)

[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
```

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 **[Push >] 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. Defaults to `/services/collector`.

*Note: This **single** endpoint supports both JSON events via `/event` and raw events via `/raw` . See examples below.*

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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes`, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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 on the **Fields (Metadata)** tab 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., consider the following expression: ``${__e['index']} || 'myIndex'}`` Its `L->R` and `OR` logic specifies: If an inbound event includes an `index` field, use that field's value. Otherwise, fall back to the `myIndex` constant defined in this expression.

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 on the **Auth Tokens** tab will normally override fields of the same name in events. But you can specify that fields in events should override these fields' values.

E.g., consider the following expression: ``${__e['index']} || 'myIndex'}`` Its L->R and OR logic specifies: If an inbound event includes an `index` field, use that field's value. Otherwise, fall back to the `myIndex` constant defined in this expression.

Fields here are evaluated and applied **before** any fields specified in the [Fields \(Metadata\)](#) section.

Advanced Settings

Enable Proxy Protocol: Enable if the connection is proxied by a device that supports [Proxy Protocol](#) v1 or v2.

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to 256. Enter 0 for unlimited.

Activity log sample rate: Determines how often request activity is logged at the info level. The default 100 value logs every 100th value; a 1 value would log every request; a 10 value would log every 10th request; etc.

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
- __srcIpPort
- __hecToken

Format and Endpoint Examples

Splunk HEC to LogStream

- 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 - JSON Event Examples Splunk HEC - Raw Event Example

```
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
```

Splunk HEC to LogStream Cloud

- Navigate to LogStream Cloud's Splunk HEC Source > **Auth Tokens** tab.
- Copy your token out of the **Token** field.
- From the command line, use `https` , your Cribl.Cloud portal's **Ingest Endpoint** and port, and the token's value:

Splunk HEC > LogStream Cloud endpoint

```
curl -k "https://in.logstream.<tenant-ID>.cribl.cloud:8088/services/collec  
-H "Authorization: <token_value>" \  
-d '{"event": "Goats are better than ponies."}'{"event": "Goats are bet
```

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 **[Push >] 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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to Yes , nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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: Enable if the connection is proxied by a device that supports [Proxy Protocol](#) v1 or v2.

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`
- `__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 Streams

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 **[Pull >] 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 AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.
- **Manual**: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud.
- **Secret**: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key.

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`

How LogStream Pulls Data

Worker Processes get a list of available shards from Kinesis, and contact the Leader Node to fetch the latest sequence numbers. Based on the sequence number's value, the Worker either resumes the shard reading from where LogStream previously left off, or starts reading from the beginning.

Worker Processes become Kinesis Consumers, and fetch the records for the assigned shards. Every 5 minutes, each Worker Process forwards to the Leader

Node the latest sequence numbers for the shards that Worker Process is responsible for. The Leader Node persists the `shardId > sequenceNumber` mapping to disk.

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 **[Pull >] 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 AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.

Manual

If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret

If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair:** Use the drop-down to select a secret key pair that you've [configured](#) in 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.
-

How LogStream Pulls Data

Workers poll messages from SQS. The call will return a message if one is available, or will time out after 1 second if no messages are available.

Each Worker gets its share of the load from SQS, and it receives a notification of a file newly added to an S3 bucket. By default, SQS returns a maximum of 10 messages in a single poll request.

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 and/or Route filters for further processing.

⚠ LogStream does **not** support data ingestion from buckets saved to S3's Glacier or Deep Glacier storage classes – whose stated retrieval lags (variously minutes to 48 hours) cannot guarantee data availability.

Configuring Cribl LogStream to Receive Data from Amazon S3

From the top nav of a LogStream instance or Group, select **Sources**, then select [Pull >] **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 AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` . Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a

private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair:** Use the drop-down to select a secret key pair that you've [configured](#) in 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.
-

How LogStream Pulls Data

Workers poll message from SQS. The call will return messages if they are available, or will time out after 1 second if no messages are available.

Each Worker gets its share of the load from S3. By default, S3 returns a maximum of 1 message in a single poll request. You can change this default in **Max messages**.

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 **[Pull >] 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 ID: ID of the Pub/Sub topic from which to receive events.

Subscription ID: ID of 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.

Google Cloud Roles and Permissions

Your Google Cloud service account should have at least the following roles on subscriptions:

- `roles/pubsub.subscriber`
- `roles/pubsub.viewer` or `roles/viewer`

To enable LogStream's **Create topic** and/or **Create subscription** options, your service account should have one of the following (or higher) roles:

- `roles/pubsub.editor`
- `roles/editor`

Either `editor` role confers multiple permissions, including those from the lower `viewer`, `subscriber`, and `publisher` roles. For additional details, see the Google Cloud [Access Control](#) topic.

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 LogStream to Receive Data from Azure Event Hubs

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Pull >] 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: The name of the consumer group that includes this LogStream instance. Defaults to `Cribl` .

⚠ To prevent excessive Kafka rebalancing and reduced throughput, each **Group ID** that you specify here should be subscribed to only

one Kafka Topic – i.e., only to the single Topic you specify in **Event Hub name**. This has two implications:

- The **Group ID** should be something other than `$Default` , especially if Event Hubs are stored in shared accounts, where the `$Default` group might be subscribed to other Topics.
- You should configure a **separate** Azure Event Hubs Source for each Group:Topic pair whose events you want to subscribe to.

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 Settings

Enabled: Defaults to `No` . When toggled to `Yes` , all the settings in this section are required.

- **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` .

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Use this default option to enter your Event Hubs connection string. Exposes a **Password** field for this purpose.
- **Secret:** This option exposes a **Connection string (text secret)** drop-down, in which you can select a stored secret that references an Event Hubs connection string. The secret can reside in LogStream's [internal secrets manager](#) or (if enabled) in an external KMS. A **Create** link is available if you need a new secret.

Connection String Format

Either authentication method uses an Azure Event Hubs connection string in this format:

```
Endpoint=sb://<FQDN>;SharedAccessKeyName=<your-shared-access-key-name>;SharedAccessKey=<your-shared-access-key-value>
```

A fictitious example, is:

```
Endpoint=sb://dummysnamespace.servicebus.windows.net/;SharedAccess  
KeyName=DummyAccessKeyName;SharedAccessKey=5d0ntTRytoC24opYThisAsi  
t3is2B+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.

Advanced Settings

Use these settings to fine-tune LogStream's integration with Event Hubs Kafka brokers. For details, see Azure Event Hubs' [recommended configuration](#) documentation. If you are unfamiliar with these parameters, contact Cribl Support to understand the implications of changing the defaults.

Session timeout (ms): Timeout used to detect client failures when using Kafka's group management facilities. (Corresponds to `session.timeout.ms` in the Kafka domain.) If the client sends the broker no heartbeats before this timeout expires, the broker will remove this client from the group, and will initiate a rebalance. Value must be lower than `rebalanceTimeout`. Defaults to `30000` ms, i.e., 30 seconds.

Rebalance timeout (ms): Maximum allowed time for each worker to join the group after a rebalance has begun. (Corresponds to `rebalance.timeout.ms` in the Kafka domain.) If this timeout is exceeded, the coordinator broker will remove the worker from the group. Defaults to `60000 ms`, i.e., 1 minute.

Heartbeat interval (ms): Expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. (Corresponds to `heartbeat.interval.ms` in the Kafka domain.) Value must be lower than `sessionTimeout`, and typically should not exceed 1/3 of the `sessionTimeout` value. Defaults to `3000 ms`, i.e., 3 seconds.

- i** If you observe an excessive number of group rebalances, and/or you observe consumers not regularly pulling messages, try increasing the values of all three of the above parameters.

How LogStream Pulls Data

Azure Event Hubs treat all the Worker Nodes as members of a Consumer Group, and each Worker gets its share of the load from Azure Event Hubs. This is the same process as normal Kafka. By default, Workers will poll every 5 seconds. In the case of Leader failure, Worker Nodes will continue to receive data as normal.

What's Next

➤ [Azure Event Hubs Integrations](#)

Azure Blob Storage

Cribl LogStream supports receiving data from [Azure Blob Storage](#) buckets. LogStream uses [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.

Restrictions

LogStream supports data ingestion from Azure's **hot** and **cool** access tiers, but not from the **archive** tier – whose stated retrieval lag, up to several hours, cannot guarantee data availability.

This Source supports [block blobs](#), but not append blobs, which can change after they are initially created and the create message is sent. Consider using a LogStream [Azure Event Hubs](#) Source if you need to ingest changeable Azure data.

Configuring Cribl LogStream to Receive Data from Azure Blob Storage

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Pull >] Azure > Blob Storage** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Azure Blob > 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}` .

Authentication method: See [Authentication Settings](#) below.

Filename filter: Regex matching file names to download and process. Defaults to `.*` , to match all characters.

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Use this default option to enter your Azure Storage connection string directly. Exposes a **Connection string** field for this purpose. (If left blank, LogStream will fall back to `env.AZURE_STORAGE_CONNECTION_STRING` .)
- **Secret:** This option exposes a **Connection string (text secret)** drop-down, in which you can select a stored secret that references an Azure Storage connection string. The secret can reside in LogStream's [internal secrets manager](#) or (if enabled) in an external KMS. A **Create** link is available if you need a new secret.

Connection String Format

Either authentication method uses an Azure Storage connection string in this format:

```
DefaultEndpointsProtocol=[http|https];AccountName=<your-account-name>;AccountKey=<your-account-key>
```

A fictitious example, using Microsoft's recommended HTTPS option, is:

```
DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=12345678...32
```

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).

- 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.

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

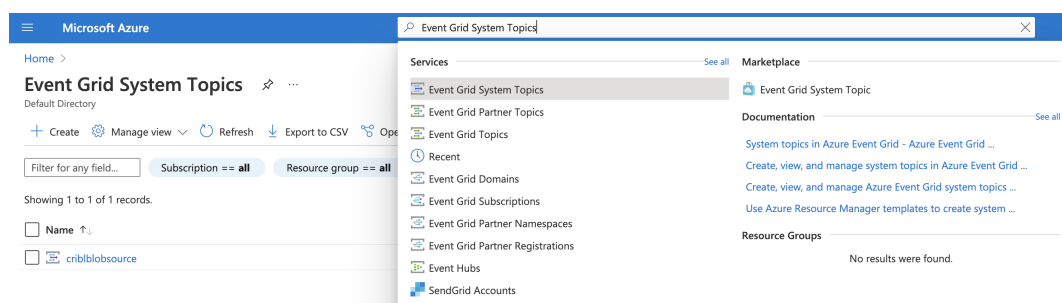
To configure notifications from the Blob storage account in the Azure backend, there are three major steps, outlined below:

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, start at **Event Grid System Topics**:



Azure portal > System topics

Select **+Create** to create a new system topic, then set the **Topic Type** to **Storage Account (Blob)**:

Basics Tags Review + create

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](#)

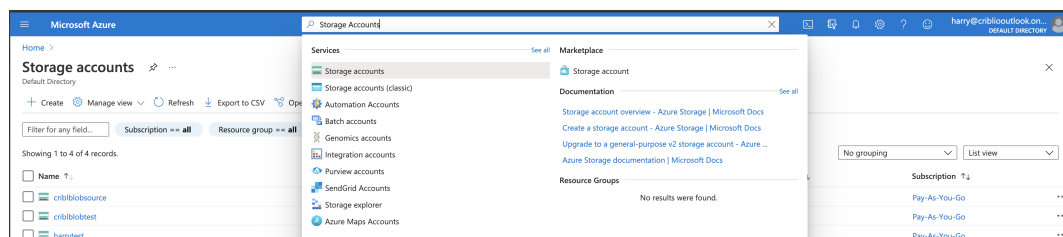
Creating a system topic

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.)

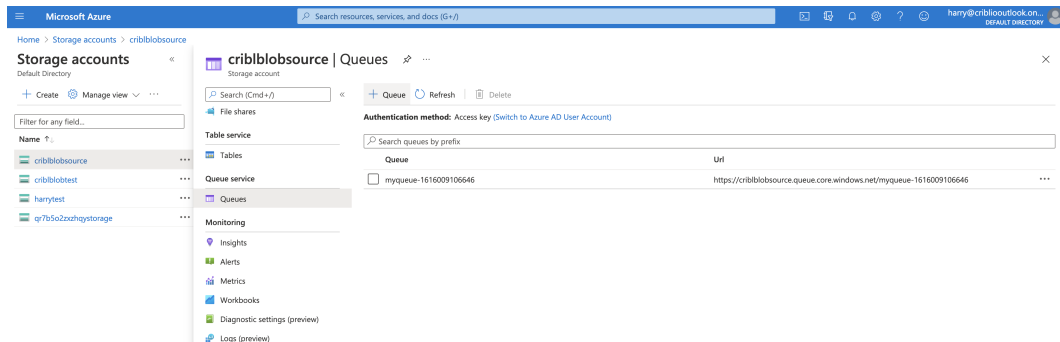
2. Create Storage Queue

Next, navigate to your storage account to create a queue.



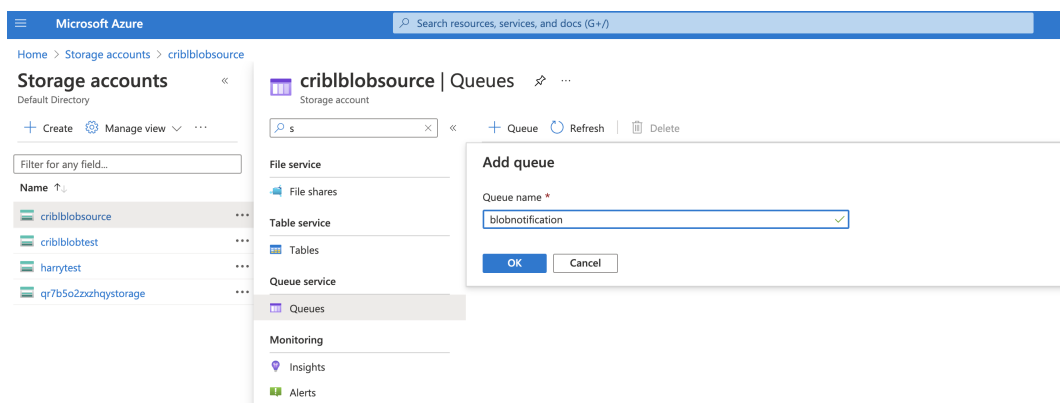
Accessing your storage account

Select the storage account for which you would like to set up notifications. Then, in the submenu, select **Queue service > Queues**:



Accessing queues

Select **Create queue**, and give the queue a name that is meaningful to you.



Adding a queue

3. Configure Storage Account Notifications

Next, set up the storage account that will publish **Blob Create** notifications to the queue, using the system topic. From the **Storage Accounts** menu, select **Events**:

Microsoft Azure

Home > Storage accounts > criblblobsource





Storage accounts

Default Directory

+ Create ⚙️ Manage view ▾ ⋮

Filter for any field...

Name ↑↓

 criblblobsource	⋮
 criblblobtest	⋮
 harrytest	⋮
 qr7b5o2zxzhqystorage	⋮

criblblobsource | Eve

Storage account

s

- Overview
- Activity log
- Tags
- Diagnose and solve problems
- Access Control (IAM)
- Data migration
- Events**
- Storage Explorer (preview)

Settings

- Access keys
- Geo-replication

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 *	<input type="text" value="criblobtest"/>
Event Schema	<input type="text" value="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	<input type="text" value="criblobtest"/>
System Topic Name * ⓘ	<input type="text" value="criblobsource"/>

EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types	<input type="text" value="Blob Created"/>
-----------------------	---

ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

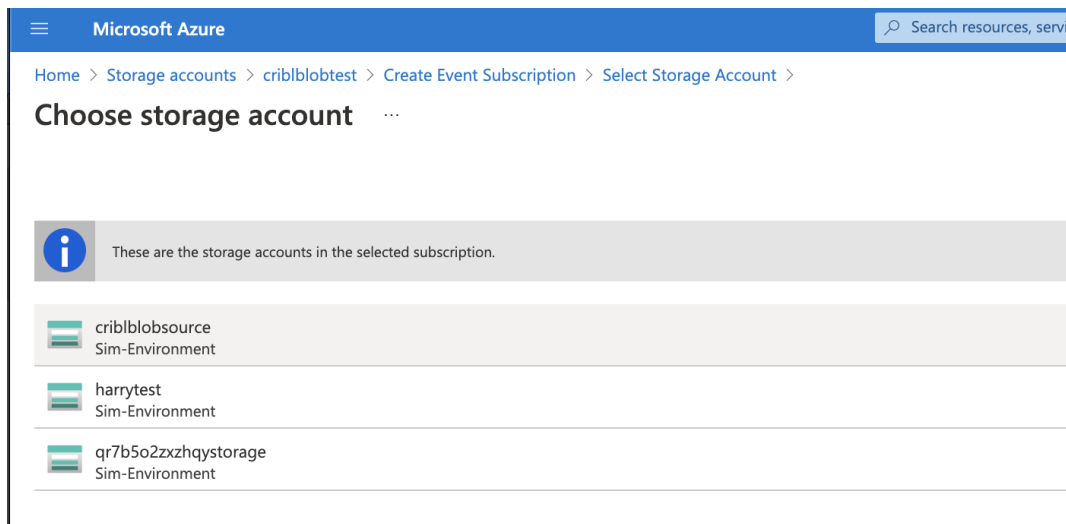
Endpoint Type *	Storage Queues (change)
Endpoint *	Select an endpoint
Use system assigned identity ⓘ	<input type="checkbox"/>

Creating a subscription.

There are a few things to configure here:

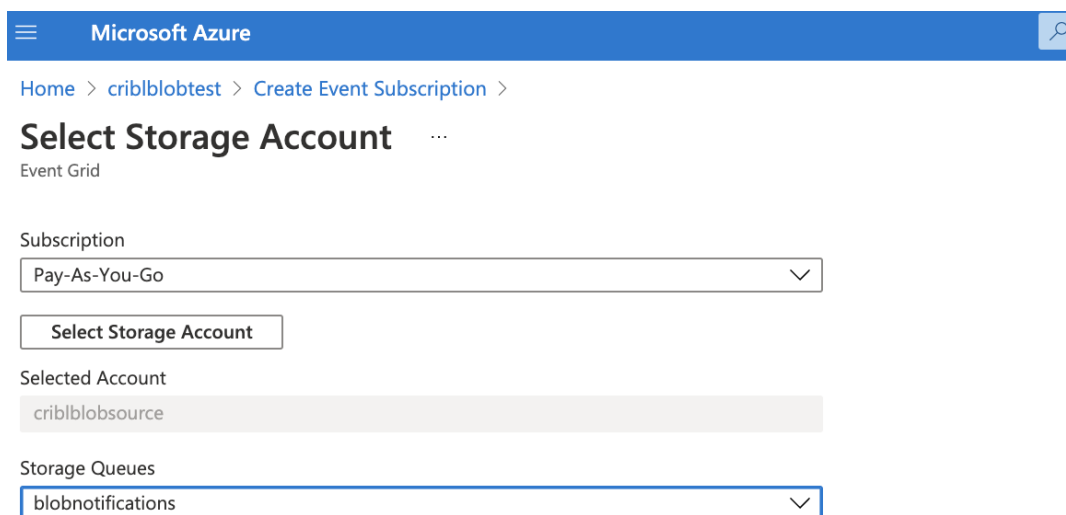
- Enter a **Name** for the subscription.
- In **System Topic Name**, enter the name of the system topic you created in [1. Create System Topic](#) above.
- In **Event Types**, select **Blob Created**, and deselect **Blob Deleted**.
- As the **Endpoint Type**, select **Storage Queues**.
- Click **Select an endpoint**, and click the subscription to use (**Pay-As-You-Go**).

Next, select the storage account on which to add the subscription:



Choosing the storage account

Select the queue you created in [Create Storage Queue](#) above, and click **Confirm Selection** to save the settings.




Selecting the storage account

To complete the process, click **Create**.

Microsoft Azure

Search resources

[Home](#) > [criblblobtest](#) >

 **Create Event Subscription** ...

Event Grid

[Basic](#) [Filters](#) [Additional Features](#)

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

EVENT SUBSCRIPTION DETAILS

Name *

xxxxxx ✓


Event Schema

Event Grid Schema ▼

TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type

 Storage account

Source Resource

criblblobtest

System Topic Name * ⓘ

criblblobsource ✓

EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)


Filter to Event Types

2 selected ▼

ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type *

 Storage Queues [\(change\)](#)

Endpoint *

blobnotifications [\(change\)](#)

Use system assigned identity ⓘ

☐

Create

Creating the subscription

How LogStream Pulls Data

Workers poll messages from Azure Blob Storage using the Azure Event Grid Queue. The call will return a message if one is available, or will time out after 5 seconds if no messages are available.

Each Worker gets its share of the load from Azure Event Grid, and receives a notification of a new file added to an Azure Blob Storage bucket.

By default, the maximum number of messages Azure Event Grid returns in a single poll request is 1 per Worker Process.

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.

Azure AD Permissions

In Azure Active Directory, the application representing your LogStream instance must be granted the following permission to pull data. The permission **Type** must be Application – Delegated is not sufficient:

- ServiceHealth.Read – Required for Office 365 Services.Messages , Services.Current Status ,and Services.Historical Status .

+ Add a permission

✓ Grant admin consent for Cribl

API / Permissions name	Type	Description	Admin consent req...	Status
Office 365 Management APIs (3)				
ActivityFeed.Read	Application	Read activity data for your organization	Yes	Granted for Cribl
ActivityFeed.ReadDlp	Application	Read DLP policy events including detected sensitive data	Yes	Granted for Cribl
ServiceHealth.Read	Application	Read service health information for your organization	Yes	Granted for Cribl

Registered application permissions

Configuring LogStream to Receive Data from the Service API

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Pull >] 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, for example:

```
*/${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`

How LogStream Pulls Data

The Office 365 Services Source retrieves data using LogStream scheduled Collection jobs, which include Discover and Collection phases. The Discover phase task returns the URL of the content to collect.

In the Source's **General Settings > Content Types > Interval** column, you configure the polling schedule for each Content Type independently.

The job scheduler spreads the Collection tasks across all available Workers. The collected content is paginated, so the collection phase might include multiple calls to fetch data.

Viewing Scheduled Jobs

This Source executes LogStream's [scheduled collection jobs](#). Once you've configured and saved the Source, you can view those jobs' results by reopening the Source's config modal and clicking its **Job Inspector** tab.

Each content type that you enabled gets its own separate scheduled job.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring > System > Job Inspector > Currently Scheduled** tab.

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.

Azure AD Permissions

In Azure Active Directory, the application representing your LogStream instance must be granted the following permissions to pull data. Each permission's **Type** must be Application – Delegated is not sufficient:

- ActivityFeed.Read – Required for all Content Types except DLP.All .
- ActivityFeed.ReadDlp – Required for the DLP.All Content Type.

+ Add a permission

✓ Grant admin consent for Cribl

API / Permissions name	Type	Description	Admin consent req...	Status
Office 365 Management APIs (3)				
ActivityFeed.Read	Application	Read activity data for your organization	Yes	✓ Granted for Cribl
ActivityFeed.ReadDlp	Application	Read DLP policy events including detected sensitive data	Yes	✓ Granted for Cribl
ServiceHealth.Read	Application	Read service health information for your organization	Yes	✓ Granted for Cribl

Registered application permissions

Office 365 Subscriptions

LogStream does not support starting/stopping Office 365 subscriptions. You can start subscriptions either via another Office 365 API client, or simply via

`curl` commands. We document the `curl` command method below in [Starting Content Subscriptions](#).

Configuring LogStream to Receive Data from the Activity API

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Pull >] 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`

Starting Content Subscriptions

Content subscriptions (a different concept from the O365 subscription plans) are required in order for LogStream to be able to begin retrieving O365 data. There is a separate subscription required for each Content Type. If you are using an existing Azure-registered application ID that already has subscriptions started, then you can ignore this section. But if you are:

- Using a newly registered application ID, and therefore never had any subscriptions started, or
- Reusing an application ID that had subscriptions started, but are currently stopped

...then you will need to use this procedure to manually start the necessary subscriptions.

This is a two-step process. The first command obtains an auth token, which is used in the second command to actually start the subscription. To execute these commands, you'll need the same information (i.e. client secret, application ID, and tenant ID) that you already require to configure this Source in LogStream's GUI. Replace those three variables as appropriate in the commands below.

1.

```
curl -d "client_secret=<client  
secret>&resource=https://manage.office.com&client_id=<app  
id>&grant_type=client_credentials" -X POST  
https://login.windows.net/<tenant id>/oauth2/token
```

Here is an example of each command executed and expected output:

```
ay5rxlczPgylPsSqMiTqLeSf438i3g9riZltK7g2WonZFStF7gewTlPWLqllGi2FY7  
-cEwjWGeDjGH_UQ3j_gkHNOVR9t7JtjqEwS4ObA-ky32GMRDvw"}
```

Example Command #2

```
$ curl -d "" -H "Authorization: Bearer  
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Im5PbzNaRHJPRFhFSzFqS1  
doWHNsSFJfS1hFZyIsImtpZCI6Im5PbzNaRHJPRFhFSzFqS1doWHNsSFJfS1hFZyJ9  
.eyJhdWQiOiJodHRwczovL21hbmFnZS5vZmZpY2UuY29tIiwiaXNzIjoiaHR0cHM6L  
y9zdHMud2luZG93cy5uZXQvZmFiZDI0NGYtZGZhOS00MjMzLTg2ZGEtMTYwYzZjMzA  
xNTRhLyIsImldCI6MTYyMjA4NTUyOSwibmJmIjoxNjIyMDg1NTI5LCJleHAiOiE2M  
jIwODk0MjksImFpbyI6IkUyWmdZSmdTSGLWcjR0VW45bDhnU0ZwNzMvTlBQT09Iiw  
iYXBwaWQiOiIwMGZhYzVmYS1jNDRiLTRmYmItYWIXOC1lNDdiZjJkNjYzNTkiLCJhc  
HBpZGFjciI6IjEiLCJpZHAiOiJodHRwczovL3N0cy53aW5kb3dzLm5ldC9mYWJkMjQ  
0Zi1kZmE5LTQyMzY0ODZkYS0xNjBjNmMzMDE1NGEvIiwib2lkIjoiaZTlMTkzOGMtM  
mIzMi00MDY2LTgyMmUtNDhlNTQ5NGEzMTljIiwicmgiOiIwLkFWb0FueVM5LXFuZk0  
wS0cyYFlNYkRBVLN2ckYtZ0JMeEx0UHF4amtlX0xXWTFsYUFBQS4iLCJyb2xlcYI6W  
yJBY3Rpdml0eUZlZWQuUmVhZERscCI6IlNlcnZpY2VIZWFSdGguUmVhZCIsIkFjdG  
L2aXR5RmVLZC5SZWFKIl0sInN1YiI6ImU2ZTE5MzhjLTJiMzItNDA2Ni04MjJlLTQ4Z  
TU0OTRhMzE5YyIsInRpZCI6ImZhYmQyNDRmLWRmYTktNDIzMy04NmRhLTE2MGM2Yz  
MwMTU0YSIsInV0aSI6IkdiUUNweFFDZ2tLNWlZMnV3OHdZQUEiLCJ2ZXIiOiIwLkFj  
Q.RyXNthPtvVBsd6UJdF1e4F6qhYw1fGC0GAcQjK54zzZOM5C6n57QviK-w8ea-  
gbQQv_e8mGuPWd7_-  
NTPcjKQwwt1hElpVjnudhyHL9HPRMD__scKAxmrvKpURk_42FqxWEJCuD_NEzQSoC  
Jibyg8RmbNCRbe4Qq3-6Pd_3LEqXUrSX30Y00yg82-yjbJhipa_aP0-  
SRYskDbYwQN1hciGddnISHvINc-  
ay5rxlczPgylPsSqMiTqLeSf438i3g9riZltK7g2WonZFStF7gewTlPWLqllGi2FY7  
-cEwjWGeDjGH_UQ3j_gkHNOVR9t7JtjqEwS4ObA-ky32GMRDvw" -X POST  
https://manage.office.com/api/v1.0/12345678-aaaa-4233-cccc-  
160c6c30154a/activity/feed/subscriptions/start?  
contentType=Audit.AzureActiveDirectory
```

Output:

```
{"contentType": "Audit.AzureActiveDirectory", "status": "enabled", "w  
ebhook": null}
```

Note there is no output when executing this second command with a `stop` operation.

You'll need to execute the second command for each Content Type whose logs you wish to collect. Use the exact strings below to specify Content Types in that

command:

- Audit.AzureActiveDirectory
- Audit.Exchange
- Audit.SharePoint
- Audit.General
- DLP.All

How LogStream Pulls Data

The Office 365 Activity Source retrieves data using LogStream scheduled Collection jobs, which include Discover and Collection phases. The Discover phase task returns the URL of the content to collect.

In the Source's **General Settings > Content Types > Interval** column, you configure the polling schedule for each Content Type independently.

The job scheduler spreads the Collection tasks across all available Workers. The collected content is paginated, so the collection phase might include multiple calls to fetch data.

Viewing Scheduled Jobs

This Source executes LogStream's [scheduled collection jobs](#). Once you've configured and saved the Source, you can view those jobs' results by reopening the Source's config modal and clicking its **Job Inspector** tab.

Each content type that you enabled gets its own separate scheduled job.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring > System > Job Inspector > Currently Scheduled** tab.

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.

Office 365 Setup

Your Office 365 service account should include a role with `Message Tracking` and `View-Only Recipients` permissions, assigned to the Office 365 user that will integrate with LogStream.

Configuring LogStream to Receive Office 365 Message Trace Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Pull >] 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`
-

How LogStream Pulls Data

The Office 365 Message Trace Source uses a scheduled REST Collector. It runs one collection task every **Poll interval**, and a single Worker will process the collection. The data is paginated, so the Worker might make multiple calls to fetch the data.

Viewing Scheduled Jobs

This Source executes LogStream's [scheduled collection jobs](#). Once you've configured and saved the Source, you can view those jobs' results by reopening the Source's config modal and clicking its **Job Inspector** tab.

Each content type that you enabled gets its own separate scheduled job.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring > System > Job Inspector > Currently Scheduled** tab.

TCP JSON

Cribl LogStream supports receiving of data over TCP in JSON format (see the 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 **[Push >] 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).

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Use this default option to enter the shared secret that clients must provide in the `authToken` header field. Exposes an **Auth token** field for this purpose. (If left blank, unauthenticated access will be permitted.) A **Generate** link is available if you need a new secret.
 - **Secret:** This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the `authToken` header field value described above. The secret can reside in LogStream's [internal secrets manager](#) or (if enabled) in an external KMS. A **Create** link is available if you need a new secret.
-

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.

 In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to **Yes**, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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: Enable 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`
- `__srcIpPort`

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``).

Examples

Testing TCP JSON In

This first example simply tests that data is flowing in through the Source:

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`

LogStream to LogStream Cloud

This second example demonstrates using TCP JSON to send data from one LogStream instance to a downstream LogStream Cloud instance. We assume that the downstream Cloud instance uses LogStream Cloud's **default** TCP JSON Source configuration.

So all the configuration happens on the upstream instance's [TCP JSON Destination](#). Replace the `<tenant-ID>` placeholder with the tenant ID from your [Cribl Cloud portal](#).

TCP JSON Destination Configuration

On the upstream LogStream instance's Destination, set the following field values to match the target Cloud instance's defaults:

General Settings

Address: `in.logstream.<tenant-ID>.cribl.cloud` – you can simply copy/paste your Cribl Cloud portal's **Ingest Endpoint** here

Port: `10070`

TLS Settings (Client Side)

Enabled: Yes

Validate server certs: Yes

TCP (Raw)

Cribl LogStream supports receiving of data over TCP. (See examples and header options [below](#).)

i 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 **[Push >] 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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes`, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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: Enable 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 Examples

Every new TCP connection may contain an **optional** header line, with an `authToken` and a list of fields and values to add to every event. To use the LogStream Cloud sample, copy the `<token_value>` out of your LogStream Cloud TCP Source.

Sample test.raw (LogStream) Sample test.raw (LogStream Cloud)

```
{"authToken":"myToken42", "fields": {"region": "us-east-1", "AZ":"az1"}}  
  
this is event number 1  
this is event number 2
```

Enabling the Example – LogStream

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`

Enabling the Example – LogStream Cloud

Use netcat with `--ssl` and `--ssl-verify` :

Command-line test

```
cat test.raw | nc --ssl --ssl-verify in.logstream.<tenant-ID>.cribl.cloud
```

HTTP/S (Bulk API)

Cribl LogStream supports receiving data over HTTP/S using the Cribl Bulk API, [Splunk HEC](#), or [Elastic Bulk API](#).

i Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring LogStream to Receive Data over HTTP(S)

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Push >] 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.

 In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes` , nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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: Enable if the connection is proxied by a device that supports [Proxy Protocol](#) v1 or v2.

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to 256. Enter 0 for unlimited.

Activity log sample rate: Determines how often request activity is logged at the info level. The default 100 value logs every 100th value; a 1 value would log every request; a 10 value would log every 10th request; etc.

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
- __srcIpPort
- __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

LogStream

The examples in this section demonstrate sending HTTP data into a LogStream binary that you manage on-prem, or on a VM. To set up these 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
```

i For Splunk HEC, the token specification can be either `Splunk <token>` or `<token>` .

LogStream Cloud – Single Event

1. Generate and copy a token in your LogStream Cloud instance's HTTP Source > **General Settings**.
2. From the command line, use `https` , your Cribl.Cloud portal's **Ingest Endpoint** and port, and the token's value:

LogStream Cloud – Single Event

```
curl -k https://in.logstream.<tenant-ID>.cribl.cloud:10080/cribl/_bulk -H
```

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 **[Push >] 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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to Yes , nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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.

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.)

i 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 **[Push >] 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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes`, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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: Enable if the connection is proxied by a device that supports [Proxy Protocol](#) v1 or v2.

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Activity log sample rate: Determines how often request activity is logged at the `info` level. The default `100` value logs every 100th value; a `1` value would log every request; a `10` value would log every 10th request; etc.

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`
 - `__srcIpPort`
 - `__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 the left (LogStream) example:

```
filebeat.yml (LogStream)    filebeat.yml (LogStream Cloud)

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 LogStream to Receive Data from Kafka Topics

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Pull >] 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: Enter the names of topics to subscribe to. Press `Enter` / `Return` between multiple entries.

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.

 In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Client Side)** tab's **Enabled** slider to Yes , nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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.

Advanced Settings

Use these settings to fine-tune LogStream's integration with Kafka topics. If you are unfamiliar with these parameters, contact Cribl Support to understand the implications of changing the defaults.

Session timeout (ms): Timeout used to detect client failures when using Kafka's group management facilities. If the client sends the broker no heartbeats before this timeout expires, the broker will remove this client from the group, and will initiate a rebalance. Value must be between the broker's configured `group.min.session.timeout.ms` and `group.max.session.timeout.ms`. Defaults to `30000` ms, i.e., 30 seconds. For details, see the [Kafka documentation](#).

Rebalance timeout (ms): Maximum allowed time for each worker to join the group after a rebalance has begun. If the timeout is exceeded, the coordinator broker will remove the worker from the group. Defaults to `60000` ms, i.e., 1 minute. For details, see the [Kafka documentation](#).

Heartbeat interval (ms): Expected time between heartbeats to the consumer coordinator when using Kafka's group management facilities. Value must be lower than `sessionTimeout`, and typically should not exceed 1/3 of the `sessionTimeout` value. Defaults to `3000` ms, i.e., 3 seconds. For details, see the [Kafka documentation](#).

- i** If you observe an excessive number of group rebalances, and/or you observe consumers not regularly pulling messages, try increasing the values of all three of the above parameters.

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)

How LogStream Pulls Data

Kafka treats all the Worker Nodes as members of a Consumer Group, and Kafka manages each Node's data load. By default, Workers will poll every 5 seconds. In the case of Leader failure, Worker Nodes will continue to receive data as normal.

Metrics

Cribl LogStream supports receiving metrics in these wire formats/protocols: [StatsD](#), [StatsD Extended](#), and [Graphite](#). Automatic protocol detection happens on the first line received over a TCP connection or a UDP packet. Lines not matching the detected protocol are dropped.

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 **[Push >] 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: Enable 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

Data Format/Protocol Examples

StatsD

Format: `MetricName:value|type`

StatsD Example

```
metric1:100|g  
metric2:200|ms  
metric.dot.3:300.16|c
```

See the StatsD [repo](#).

StatsD Extended

Format: `MetricName:value|type|#dim=value,dim2=value`

StatsD Extended Example

```
metric1:100|g|#dim1=val1,dim2=val2,dim3=val3  
metric2:200|ms|#dim1=val1,dim2=val2,dim3=val3  
metric.dot.3:300.16|c|#dim1=val1,dim2=val2,dim3=val3
```

Graphite

Format: MetricName[;dim1=val1[;dim2=val2]] value time

Graphite Example with Dimensions

```
metric1;dim1=val1;dim2=val2 100 9999
metric2;dim1=val1;dim2=val2 200 9999
metric.dot.3;dim1=val1;dim2=val2 300.16 9999.16
```

Graphite Example without Dimensions

```
metric1 100 9999
metric2 200 9999
metric.dot.3 300.16 9999.16
```

See the Graphite (also known as Carbon) [plaintext protocol](#).

SNMP Trap

Cribl LogStream supports receiving data from SNMP Traps.

i 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 **[Push >] 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`
- `__srcIpPort` : In this particular Source, this field uses a pipe (`|`) symbol to separate the source IP address and the port, in this format:
`event.__srcIpPort = ${rInfo.address}|${rInfo.port};`
- `__snmpVersion` : Acceptable values are `0` , `2` , or `3` . These respectively indicate SNMP v1, v2c, and v3.
- `__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 Remote Write

Cribl LogStream supports receiving metric data from Prometheus instances that are configured to send data via the [remote write](#) protocol.

i Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring LogStream to Receive Metrics from Prometheus Remote Write Sources

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Push >] Prometheus > Remote Write** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Prometheus Remote Write > 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`.

Port: Enter the port number to listen on..

Remote Write API endpoint: Enter the absolute path on which to listen for Prometheus requests. Defaults to `/write`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/write`.

Authentication

Select one of the following options for authentication:

- **None:** Don't use authentication.

- **Auth token:** Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header, or click **Generate** if you need a new token.
- **Auth token (text secret):** Provide an HTTP token referenced by a secret. Select a stored text secret in the resulting drop-down, or click **Create** to configure a new secret.
- **Basic:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials. Click **Generate** if you need a new password.
- **Basic (credentials secret):** Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or click **Create** to configure a new secret.

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.

 In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to **Yes**, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with Cribl.Cloud's predefined TLS configuration.

You can ingest Prometheus remote write data on any of your Cribl.Cloud portal's open ports `20000 – 20010`, all of which have TLS enabled.

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: Enable if the connection is proxied by a device that supports [Proxy Protocol](#) v1 or v2.

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256`. Enter `0` for unlimited.

Keep alive timeout (seconds): Maximum time to keep a socket connection open to wait for additional data, after the last response was sent. When the incoming request frequency is high, increase this from the default 5 seconds, to avoid creating a new connection per request. (By default, Prometheus will attempt to keep connections open for up to 5 minutes.)

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`
- `__srcIpPort`

Detecting Metrics' Types

Because Prometheus remote write requests don't specify metrics' types, LogStream applies the following rules to determine the type as we ingest them:

- If the metric's name ends with `_total` , `_sum` , `_count` , or `_bucket` , the type is set to `counter` .
- Otherwise, the metric's type is set to `gauge` .

This is consistent with the type detection practiced by other services implementing the remote write protocol. See, for example, [New Relic's](#) and [Elastic's](#) documentation.

Note that LogStream supports the `timer` type in addition to `counter` and `gauge` .

Prometheus Scraper

Cribl LogStream supports receiving batched data from Prometheus targets. This is a pull Source; to ingest Prometheus streaming data, see [Prometheus Remote Write](#).

i Type: **Pull** | TLS Support: **No** | Event Breaker Support: **No**

This Source does not currently support Prometheus metadata.

Configuring LogStream to Scrape Prometheus Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Pull >] Prometheus > Scraper** 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: Specify the dimensions to include in events. Defaults to `host` and `source`.

Discovery type: Use this drop-down to select a discovery mechanism for targets. See [Discovery Type](#) below for the options – each of which exposes additional controls above or below the next two fields. (Some options also add extra tabs to the modal.)

Poll interval: Specify how often (in minutes) to scrape targets for metrics. Defaults to `15`. This value must be an integer that divides evenly into `60`.

Log level: Set the verbosity level to one of `debug` , `info` (the default), `warn` , or `error` .

Discovery Type

Use this drop-down to select a discovery mechanism for targets. To manually enter a targets list, use [Static](#) (the default). To enable dynamic discovery of endpoints to scrape, select [DNS](#) or [AWS EC2](#). Each selection exposes different controls and/or tabs, listed below.

Static Discovery

The `Static` option adds a **Targets** field, in which you enter a list of specific Prometheus targets from which to pull metrics.

Values can be in URL or `host[:port]` format, e.g.:

`http://localhost:9090/metrics` , `localhost:9090` , or `localhost` . If you specify only `host[:port]` , the endpoint will resolve to:

`http://host[:port]/metrics` . For further options, see [Target Discovery for DNS](#).

DNS Discovery

The `DNS` option adds a [Target Discovery](#) tab to the modal, and adds two extra fields to its **General Settings** tab:

- **DNS names:** Enter a list of DNS names to resolve.
- **Record type:** Select the DNS record type to resolve. Defaults to `SRV` (Service). Other options are `A` or `AAAA` .

AWS EC2 Discovery

The `AWS EC2` option adds [Target Discovery](#) and [Assume Role](#) tabs to the modal, and adds one extra field to the **General Settings** tab:

- **Region:** Select the AWS region in which to discover EC2 instances with metrics endpoints to scrape.

Authentication (Prometheus)

Use the **Authentication Method** buttons to select one of these authentication options for Prometheus:

- **Manual:** In the resulting **Username** and **Password** fields, enter Basic authentication credentials corresponding to your Prometheus targets.
 - **Secret:** This option exposes a **Secret** drop-down, in which you can select a stored secret that references your credentials described above. The secret can reside in LogStream's [internal secrets manager](#) or (if enabled) in an external KMS. Click **Create** if you need to configure a new secret.
-

Assume Role

With the [AWS EC2](#) target discovery type, you can configure [AssumeRole](#) behavior on AWS.

- **Enable for EC2:** Toggle to `Yes` if you want to use `AssumeRole` credentials to access EC2.
 - **AssumeRole ARN:** Enter the Amazon Resource Name (ARN) of the role to assume.
 - **External ID:** Enter the External ID to use when assuming the role.
-

Target Discovery

Setting the [Discovery type](#) drop-down to [DNS](#) or [AWS EC2](#) exposes this tab. These two discovery types expose different controls here.

Target Discovery for DNS

Setting the [Discovery type](#) drop-down to [DNS](#) exposes the following **Target Discovery** fields.

Metrics protocol: Select `http` (the default) or `https` as the protocol to use when collecting metrics.

Metrics port: Where discovered targets' metrics URLs lack port numbers, this value sets the port to append to those URLs. Defaults to `9090`.

Metrics path: Specify a path to use when collecting metrics from discovered targets. Defaults to `/metrics`.

Target Discovery for AWS

Setting the [Discovery type](#) drop-down to [AWS EC2](#) exposes the following **Target Discovery** controls. The first controls is a special case:

- **Authentication method:** Select the **Auto**, **Manual**, or **Secret** button to determine how LogStream will authenticate against AWS. Each selection changes the fields displayed on this tab – see [AWS Authentication Options](#) for details.

These remaining controls are displayed for all **Authentication method** selections:

- **Metrics protocol:** Select `http` (the default) or `https` as the protocol to use when collecting metrics.
- **Metrics port:** Specify the port number to append to the metrics URL for discovered targets. Defaults to `9090`.
- **Metrics path:** Specify a path to use when collecting metrics from discovered targets. Defaults to `/metrics`.
- **Use public IP:** The `Yes` default uses the public IP address for discovered targets. Toggle to `No` to use a private IP address.
- **Search filter:** Click **+ Add filter** to apply filters when searching for EC2 instances. Each filter row provides two columns:
 - **Filter name:** Select standard [attributes](#) from the drop-down, or type in custom attributes.
 - **Filter values:** Enter values to match within this row's attribute, Press `Enter` between values. (If you specify no values, the search will return only `running` EC2 instances.)

AWS Authentication Options

Auto: This default option uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by

reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud. This option displays the same fields as [Auto](#), plus:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. This option displays the same fields as [Auto](#), plus:

- **Secret key pair:** Use the drop-down to select a secret key pair that you've [configured](#) in LogStream's internal secrets manager or (if enabled) an external KMS. Click **Create** if you need to configure a key pair.

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`

How LogStream Pulls Data

The Prometheus Source retrieves data using LogStream scheduled Collection jobs. You determine the schedule using your **Poll interval** entry.

With the `DNS` or `AWS EC2` **Discovery Type**, these jobs include both Discover and Collection phases. The Discover phase runs on a single Worker, and returns 1 collection task per discovered target.

The job scheduler spreads the Collection tasks across all available Workers.

Viewing Scheduled Jobs

This Source executes LogStream's [scheduled collection jobs](#). Once you've configured and saved the Source, you can view those jobs' results by reopening the Source's config modal and clicking its **Job Inspector** tab.

Each content type that you enabled gets its own separate scheduled job.

You can also view these jobs (among scheduled jobs for other Collectors and Sources) in the **Monitoring > System > Job Inspector > Currently Scheduled** tab.

Grafana

Cribl LogStream supports receiving metric and log data from [Grafana Agent](#) instances via the Prometheus [remote write](#) specification. The Grafana Agent uses [Prometheus](#) for metrics collection and [Grafana Loki](#) for log collection.

i Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring LogStream to Receive Metrics and Logs from Grafana Agent Sources

From the top nav of a LogStream instance or Group, select **Sources**, then select **[Push >] Grafana** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Grafana > 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`.

Port: Enter the port number to listen on.

Remote Write API endpoint: Absolute path on which to listen for Grafana Agent's remote write requests. Defaults to `/api/prom/push`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/api/prom/push`.

Logs API endpoint: Absolute path on which to listen for Loki logs requests. Defaults to `/loki/api/v1/push`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/loki/api/v1/push`.

Authentication

The **Authentication** tab provides separate **Loki** and **Prometheus** sections, enabling you to configure these inputs separately. The two sections provide identical options.

Select one of the following options for authentication:

- **None:** Don't use authentication.
- **Auth token:** Enter the bearer token that must be included in the authorization header.
- **Auth token (text secret):** Provide an HTTP token referenced by a secret. Select a stored text secret in the resulting drop-down, or click **Create** to configure a new secret.
- **Basic:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret):** Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or click **Create** to configure a new secret.

TLS Settings (Server Side)

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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes`, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with Cribl.Cloud's predefined TLS configuration.

You can ingest Grafana data on any of your Cribl.Cloud portal's open ports `20000 – 20010`, all of which have TLS enabled.

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.

Max active requests: Maximum number of active requests allowed for this Source, per Worker Process. Defaults to `256` . Enter `0` for unlimited.

Keep alive timeout (seconds): Maximum time to keep a socket connection open to wait for additional data, after the last response was sent. When the incoming request frequency is high, increase this from the default `5` seconds, to avoid creating a new connection per request. (By default, Grafana Agent's embedded Prometheus instance will attempt to keep connections open for up to 5 minutes.)

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`
- `__srcIpPort`
- `__labels` (for log events only – will contain all the labels found in each event's corresponding Loki stream)

Detecting Metrics' Types

Because Prometheus remote write requests don't specify metrics' types, LogStream applies the following rules to determine the type as we ingest them:

- If the metric's name ends with `_total` , `_sum` , `_count` , or `_bucket` , the type is set to `counter` .
- Otherwise, the metric's type is set to `gauge` .

This is consistent with the type detection practiced by other services implementing the remote write protocol. See, for example, [New Relic's](#) and [Elastic's](#) documentation.

Note that LogStream supports the `timer` type in addition to `counter` and `gauge` .

Loki

Cribl LogStream supports receiving log data from [Grafana Loki](#) via an [adaptation](#) of the [Protobuf](#) (Protocol Buffers) specification.

i Type: **Push** | TLS Support: **YES** | Event Breaker Support: **No**

Configuring LogStream to Receive Loki Logs Data

From the top nav of a LogStream instance or Group, select **Sources**, then select **Loki** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Loki > 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`.

Port: Enter the port number to listen on.

Logs API endpoint: Absolute path on which to listen for Loki logs requests. Defaults to `/loki/api/v1/push`, which will (in this example) expand as: `http://<your-upstream-URL>:<your-port>/loki/api/v1/push`.

Authentication

Use the **Authentication type** buttons to select how Loki's [Promtail](#) agent will authenticate against LogStream::

- **None:** Don't use authentication.

- **Auth token:** Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Auth token (text secret):** Provide an HTTP token referenced by a secret. Select a stored text secret in the resulting drop-down, or click **Create** to configure a new secret.
- **Basic:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret):** Provide username and password credentials referenced by a secret. Select a stored text secret in the resulting **Credentials secret** drop-down, or click **Create** to configure a new secret.

TLS Settings (Server Side)

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 server certs:** Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).
- **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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes`, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with Cribl.Cloud's predefined TLS configuration.

You can ingest Grafana data on any of your Cribl.Cloud portal's open ports `20000 – 20010`, all of which have TLS enabled.

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: Enable if the connection is proxied by a device that supports [Proxy Protocol](#) v1 or v2.

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`
- `__srcIpPort`
- `__labels` (will contain all the labels found in each event's corresponding Loki stream)

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 **[Push >] 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.

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Use this default option to enter the shared secret clients must provide in the `authToken` header field. Click **Generate** if you need a new auth token. If empty, unauthenticated access will be permitted.
- **Secret:** This option exposes an **Auth token (text secret)** drop-down, in which you can select a stored secret that references the auth token described above. The secret can reside in LogStream's [internal secrets manager](#) or (if enabled) in an external KMS. Click **Create** if you need to configure a new secret.

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.

⚠ In a [Cribl.Cloud deployment](#), do not set the **TLS Settings (Server Side)** tab's **Enabled** slider to `Yes`, nor configure any of the tab's resulting TLS fields. Any settings that you configure here would conflict with the LogStream Cloud Source's predefined TLS configuration.

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`
- `__srcIpPort`

Examples

LogStream Cloud – TLS

- An `in_appscope` TLS Source is preconfigured for you in LogStream Cloud, using port `10090` .
- Send it AppScope data using this command:

```
./scope run -c in.logstream.<tenant-ID>.cribl.cloud:10090 -- ps -ef
```

LogStream Cloud – TCP

- Add a new AppScope Source and assign it **Port:** `10091` .
- Send it AppScope data using this command:

```
./scope run -c tcp://in.logstream.<tenant-ID>.cribl.cloud:10091 \  
> -- curl -so /dev/null https://wttr.in/94105
```

Datagen

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 **[Internal >] Datagen** from the **Data Sources** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Datagen > 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`

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 **[Internal >] 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')	<div>On</div>	<div><div></div>Live</div>
CriblMetrics	Cribl internal metrics. (Field source will be set to 'cribl')	<div>On</div>	<div><div></div>Live</div>

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.

Omitting sourcetype

As of LogStream 3.0.2, you can easily drop the `sourcetype` attribute from metrics events, leaving only `event_sourcetype`. This will prevent duplicate `sourcetype` events from being routed to Destinations.

To do this: In the same `cribl-metrics_rollup` pre-processing Pipeline (or a clone) that you attach to your Source, enable the final Eval Function, which applies this **Filter** expression to remove the `sourcetype` field:


```
_metric && _metric.startsWith('cribl.logstream.sourcetype.')
```

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.

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](#)
- [Amazon Kinesis Streams](#)
- [Amazon CloudWatch Logs](#)
- [Amazon SQS](#)
- [Azure Monitor Logs](#)
- [Azure Event Hubs](#)
- [Google Cloud Pub/Sub](#)
- [StatsD](#)
- [StatsD Extended](#)
- [Graphite](#)

- [TCP JSON](#)
- [Syslog](#)
- [Kafka](#)
- [Elasticsearch](#)
- [Honeycomb](#)
- [New Relic](#)
- [SNMP Trap](#)
- [InfluxDB](#)
- [Wavefront](#)
- [SignalFx](#)
- [Sumo Logic](#)
- [Datadog](#)
- [Prometheus](#)
- [Grafana Cloud](#)
- [Loki](#)
- [Webhook](#)

Non-Streaming Destinations

Destinations that accept events in groups or batches are referred to as non-streaming Destinations:

- [Amazon S3 Compatible Stores](#)
- [Azure Blob Storage](#)
- [Google Cloud Storage](#)
- [Google Chronicle](#)
- [Filesystem/NFS](#)
- [MinIO](#)

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:

- **Default:** Here, you can specify a default output from among your configured Destinations.
- **Output Router:** Flexible "meta-destination." Here, you can configure rules that route data to multiple configured Destinations.
- **DevNull:** An output that simply drops events. Preconfigured and active when you install LogStream, so it requires no configuration. Useful for testing.
- **SpaceOut:** This experimental Destination is undocumented. Be careful!

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**.

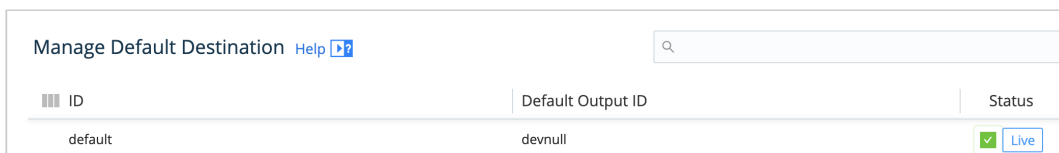
Configuring Destinations

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**.

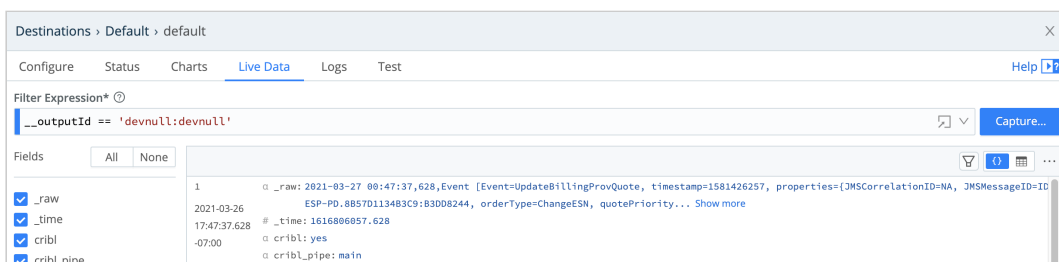
Capturing Outgoing Data

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 configuration row.



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.



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:** Toggle to `No` if you want the event to be checked against other rules lower in the stack.

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
<code>host.startsWith('66')</code>	San Francisco	Yes
<code>method=='POST' method=='GET'</code>	Seattle	No
<code>method=='POST' method=='GET'</code>	Los Angeles	Yes
<code>true</code>	New York	Yes

Splunk Single Instance

Splunk Enterprise is a streaming Destination type. You can also use this Destination to send data to a **free** Splunk Cloud instance. However, for a **Standard** Splunk Cloud instance whose `../default/outputs.conf` file contains multiple indexer entries, you must instead use LogStream's [Splunk Load Balanced](#) Destination.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** drops the newest events from being sent out of LogStream, and throws away incoming data, while leaving the contents of the PQ unchanged.

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 `ca`cert , `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`.

Authentication method: Use the buttons to select one of these options:

- **Manual:** In the resulting **Auth token** field, enter the shared secret token to use when establishing a connection to a Splunk indexer.
- **Secret:** This option exposes an **Auth token (text secret)** drop-down, in which you can select a [stored secret](#) that references the auth token described above. A **Create** link is available to store a new, reusable secret.

Notes about Forwarding to Splunk

- Data sent to Splunk is not compressed.
- The only `ack` from indexers that LogStream listens for and acts upon is the shutdown signal described in [Minimize in-flight data loss](#) above.
- 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 possible 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	<i>time=0s ---> time=300s</i>	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	<i>time=300s ---> time=600s</i>	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 at the bottom of **General Settings**. When set to `No` (the default), instead displays the [Destinations](#) section at the bottom of **General Settings**.

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** is set to its `No` default. 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 `300` seconds, i.e., 5 minutes.

- Each Worker Process performs its own indexer discovery according to the above settings.

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 [capabilities](#): `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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking. **Drop new data** drops the newest events being sent out of LogStream, throws away incoming data, and leaves the contents of the PQ unchanged.

TLS Settings (Client Side)

Enabled defaults to `No` . When toggled to `Yes` :

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

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`.

Authentication method: Use the buttons to select one of these options:

- **Manual:** In the resulting **Auth token** field, enter the shared secret token to use when establishing a connection to a Splunk indexer.
- **Secret:** This option exposes a **Auth token (text secret)** drop-down, in which you can select a [stored secret](#) that references the auth token described above. A **Create** link is available to store a new, reusable secret.

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 will need to extract the private and public keys from the Splunk-provided Splunk Cloud Universal Forwarder [credentials package](#). You will also need to reference the CA Certificate located in the same package.

You can reuse many of the settings in this Splunk Cloud package to set up Splunk Cloud Destinations. Use the following steps:

Step 1. Extract the `splunkclouduf.spl` package on the LogStream instance that you will be connecting to Splunk Cloud. You will have a folder that looks something like this:

```
100_my-splunk-cloud_splunkcloud
  /default/
    outputs.conf
    limits.conf
    your-splunk-cloud_server.pem
    your-splunk-cloud_cacert.pem
```

Step 2. (optional) Test connectivity to Splunk Cloud, using the Root CA certificate:

```
openssl s_client -CA 100_<your-splunk-cloud>_splunkcloud/default/my-splunk
```

To test the connection, you can use any of the URLs listed in the `[tcpout:splunkcloud]` stanza's `outputs.conf` section.

□ You can simplify Steps 3 and 4 below by dragging and dropping (or uploading) the `.pem` files into LogStream's **New Certificates** modal. See [SSL Certificate Configuration](#).

Step 3. Extract the private key from the Splunk Cloud certificate. At the prompt, you will need the `sslPassword` value from the `outputs.conf`. Using Elliptic Curve keys:

```
openssl ec -in 100_<your-splunk-cloud>_splunkcloud/default/<your-splunk-cl
```

If you are using RSA keys, instead use:

```
openssl rsa 100_<your-splunk-cloud>_splunkcloud/default/<your-splunk-cloud
```

Step 4. Extract the public key for the Server Certificate:

```
openssl x509 -in 100_<your-splunk-cloud>_splunkcloud/default/<your-splunk-
```

Step 5. In the Splunk Load Balanced Destination's TLS Settings (Client Side) section, enter the following:

- CA Certificate Path: Path to `<your-splunk-cloud>_cacaert.pem`.
- Private Key Path (mutual auth): Path to `private.pem` ([Step 3](#) above).
- Certificate Path (mutual auth): Path to `server.pem` ([Step 4](#) above).

□ In a [distributed deployment](#), enter this Destination configuration on each Worker Group that forwards to Splunk Cloud. Then commit and deploy your changes.

Step 6. In a [distributed deployment](#), enable [Worker UI access](#), and verify that the Certificate files have been distributed to individual workers. If they are not present, copy the Certificate files to the Workers, using exactly the same paths you used at the Group level.

Notes About Forwarding to Splunk

- Data sent to Splunk is **not** compressed.
- The only `ack` from indexers that LogStream listens for and acts upon is the shutdown signal described in [Minimize in-flight data loss](#) above.
- 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.

- You can copy and paste the Splunk Cloud servers from the [tcpout:splunkcloud] stanza] into the Splunk Load Balanced Destination's **General Settings** > **Destinations** section. E.g., from the example stanza below, you would copy only the bolded contents:

```
[tcpout:splunkcloud]
```

```
server= inputs1.your-splunk-cloud.splunkcloud.com:9997, inputs2.your-splunk-cloud.splunkcloud.com:9997, inputs3.your-splunk-cloud.splunkcloud.com:9997, inputs4.your-splunk-cloud.splunkcloud.com:9997, inputs5.your-splunk-cloud.splunkcloud.com:9997, inputs6.your-splunk-cloud.splunkcloud.com:9997, inputs7.your-splunk-cloud.splunkcloud.com:9997, inputs8.your-splunk-cloud.splunkcloud.com:9997, inputs9.your-splunk-cloud.splunkcloud.com:9997, inputs10.your-splunk-cloud.splunkcloud.com:9997, inputs11.your-splunk-cloud.splunkcloud.com:9997, inputs12.your-splunk-cloud.splunkcloud.com:9997, inputs13.your-splunk-cloud.splunkcloud.com:9997, inputs14.your-splunk-cloud.splunkcloud.com:9997, inputs15.your-splunk-cloud.splunkcloud.com:9997
```

```
compressed=false
```

- From limits.conf , copy the [thruput] value, and paste it into the Splunk Load Balanced Destination's **Advanced Settings** tab > **Throttling** setting.

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`).

 For Splunk Cloud endpoints, change the default `http:` prefix to:
`https:`

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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.)

Validate server certs: Toggle to `Yes` to 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).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle 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. Enable Round-robin DNS to better balance distribution of events between Splunk HEC servers.
- See [Splunk's documentation](#) on editing `fields.conf` to ensure the visibility of index-time fields sent to Splunk by 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.

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 one of these options:

Auto: This default option uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair:** Use the drop-down to select an API key/secret key pair that you've [configured](#) in LogStream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

Enable for S3: Toggle to `Yes` to use Assume Role credentials to access S3.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role. This is required only when assuming a role that requires this ID in order to delegate third-party access. For details, see [AWS' documentation](#).

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

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 .

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.

Remove staging dirs: Toggle to Yes to delete empty staging directories after moving files. This prevents the proliferation of orphaned empty directories.

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: Encryption type for uploaded objects – used to enable encryption on data at rest. Defaults to no encryption; the other options are Amazon S3 Managed Key or AWS KMS Managed Key .

□ AWS S3 always encrypts data in transit using HTTPS, with default one-way authentication from server to clients. With other S3-compatible stores (such as our native [MinIO Destination](#)), use an `https://` URL to invoke in-transit encryption. Two-way authentication is not required to get encryption, and requires clients to possess a certificate.

Signature version: Signature version to use for signing S3 requests. Defaults to v4 .

Reuse connections: Whether to reuse connections between requests. 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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** drops the newest events from being sent out of LogStream, and throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

Use the **Authentication Method** buttons to select an AWS authentication method.

Auto: This default option uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` . Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.

- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair:** Use the drop-down to select an API key/secret key pair that you've [configured](#) in LogStream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

Enable for Kinesis Streams: Toggle to `Yes` to use Assume Role credentials to access Kinesis Streams.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role.

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. Con"
{ "_raw": "07-03-2018 18:33:51.136 -0700 INFO  TcpOutputProc - Connection to
...
```

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

Use the **Authentication Method** buttons to select an AWS authentication method.

Auto: This default option uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` . Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.

- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair:** Use the drop-down to select an API key/secret key pair that you've [configured](#) in LogStream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

Enable for CloudWatch Logs: Toggle to `Yes` to use Assume Role credentials to access CloudWatch Logs.

AssumeRole ARN: Enter the Amazon Resource Name (ARN) of the role to assume.

External ID: Enter the External ID to use when assuming role.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

Use the **Authentication Method** buttons to select an AWS authentication method.

Auto: This default option uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` . Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. The **Secret** option exposes this additional field:

- **Secret key pair:** Use the drop-down to select an API key/secret key pair that you've [configured](#) in LogStream's secrets manager. A **Create** link is available to store a new, reusable secret.

Assume Role

Enable for SQS: Toggle to **Yes** to use Assume Role credentials to access SQS.

AWS account ID: Enter the SQS queue owner's AWS account ID. Leave empty if the SQS queue is in the same AWS account where this 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.

Authentication method: See [Authentication Settings](#) below.

Container name: Enter the container name. (A container organizes a set of blobs, similar to a directory in a file system.)

Create container: Toggle to **Yes** to create the configured container in Azure Blob Storage if one 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.

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` .

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Use this default option to enter your Azure Storage connection string directly. Exposes a **Connection string** field for this purpose. (If left blank, LogStream will fall back to `env.AZURE_STORAGE_CONNECTION_STRING` .)
- **Secret:** This option exposes a **Connection string (text secret)** drop-down, in which you can select a [stored secret](#) that references an Azure Storage connection string. A **Create** link is available to store a new, reusable secret.

Connection String Format

Either authentication method uses an Azure Storage connection string in this format:

```
DefaultEndpointsProtocol=[http|https];AccountName=
<your-account-name>;AccountKey=<your-account-key>
```

A fictitious example, using Microsoft's recommended HTTPS option, is:

```
DefaultEndpointsProtocol=https;AccountName=storagesample;AccountK
ey=12345678...32
```

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.

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.

Remove staging dirs: Toggle to `Yes` to delete empty staging directories after moving files. This prevents the proliferation of orphaned empty directories.

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 data 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. (In the Azure Dashboard, see **Workspace > Advanced settings**.)

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

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.

Azure Monitor Limitations

The Azure Monitor Logs architecture limits the number of columns per table, characters per column name, and other parameters. For details, see Microsoft's [Azure Monitor Service Limits](#) topic.

Azure will drop logs if your data exceeds these limits. To diagnose this, you can search in the Azure Data Explorer console with a query like this:

```
Operation | summarize count() by Detail
```

...for error messages of this form:

```
Data of type <type> was dropped: The number of custom fields
<number> is above the limit of 500 fields per data type.
```

Notes on HTTP-based Outputs

- 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. Enable Round-robin DNS to better balance distribution of events between destination cluster nodes.

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`.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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`.

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials. The password is your 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://dummysamespace.servicebus.windows.net/;SharedAc
cessKeyName=dummyaccesskeyname;SharedAccessKey=5d0ntTRytoC24op
YThisAsit3is2B+0GY1US/fuL3ly=
```

- **Secret:** This option exposes a **Password (text secret)** drop-down, in which you can select a [stored secret](#) that references the credentials described above. A **Create** link is available to store a new, reusable secret.

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`

What's Next

➤ [Azure Event Hubs Integrations](#)

Google Chronicle

Cribl LogStream supports sending data to [Google Chronicle](#), a cloud service for retaining, analyzing, and searching enterprise security and network telemetry data. This is a non-streaming Destination type.

To define a Google Chronicle Destination, you need to [obtain an API key](#) from Google. If you want LogStream or an external KMS to manage the API key, [configure](#) a key pair that references the API key.

Configuring Cribl LogStream to Output to Chronicle

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Google Chronicle** from the **Data Destinations** page's tiles or the Destinations left nav. Click **+ Add New** to open the **Chronicle > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Chronicle output definition.

Send events as: `Unstructured` is the only currently supported format. Cribl plans to add [UDM](#) (Unified Data Model) support in a future release.

Log type: Select an application log type to send to Chronicle. (Google Chronicle expects all batches for a given Destination to have the same log type.) Can be overwritten by the `__logType` event field.

Log text field: Specify the event field that contains the log text to send. If you do not specify a log text field, LogStream sends a JSON representation of the whole event.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

The Google Chronicle API key is required to complete this part of the Destination definition.

Use the **Authentication Method** buttons to select one of these options:

- **Manual:** In the resulting **API key** field, enter your Google Chronicle API key.
- **Secret:** This option exposes a **Secret** drop-down, in which you can select a [stored secret](#) that references your Google Chronicle API key. A **Create** link is available to store a new, reusable secret.

Processing Settings

Post-Processing

Pipeline: 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.

System fields: Specify any fields you want LogStream to automatically add to events using this output. Wildcards are supported.

Advanced Settings

Validate server certs: Toggle to **Yes** to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to **Yes** to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle to **Yes** if you want LogStream to compress the payload body before sending.

Request timeout: Enter an amount of time, in seconds, to wait for a request to complete before aborting it.

Request concurrency: Enter the maximum number of ongoing requests to allow before blocking.

Max body size (KB): Enter a maximum size, in KB, for the request body.

Max events per request: Enter the maximum number of events to include in the request body. Defaults to **0** (unlimited).

Flush period (sec): Enter the maximum time to allow between requests. Be aware that small values could cause the payload size to be smaller than the configured **Max body size**.

Extra HTTP Headers: Click **+ Add Header** to insert extra headers as **Name/Value** pairs.

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.

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.strptime(_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 one of these options:

- **Manual:** With this default option, authentication is via HMAC (Hash-based Message Authentication Code). To create a key and secret, see Google Cloud's [Managing HMAC Keys for Service Accounts](#) documentation. This option exposes these two fields:
 - **Access key:** Enter the HMAC access key.
 - **Secret key:** Enter the HMAC secret.
- **Secret:** This option exposes a **Secret key pair** drop-down, in which you can select a [stored secret](#) that references the secret key pair described above. A **Create** link is available to store a new, reusable secret.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 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.

Add Output ID: Whether to append output's ID to staging location. Defaults to `Yes`.

Remove staging dirs: Toggle to `Yes` to delete empty staging directories after moving files. This prevents the proliferation of orphaned empty directories.

Endpoint: The Google Cloud Storage service endpoint. Typically, there is no reason to change the default <https://storage.googleapis.com> endpoint.

Object ACL: Select an Access Control List to assign to uploaded objects. Defaults to `private` .

Storage class: Select a storage class for uploaded objects.

Signature version: Signature version to use for signing requests. Defaults to `v4` .

Reuse connections: Whether to reuse connections between requests. 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 ID: ID of the Pub/Sub topic to send events to.

Create topic: Toggle to `Yes` if you want LogStream to create the topic on Pub/Sub if it does not exist.

Ordered delivery: Toggle to `Yes` if you want LogStream to send events in the order that they arrived in 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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

Use the **Authentication Method** buttons to select one of these options:

- **Auto:** This option uses the environment variables `PUBSUB_PROJECT` and `PUBSUB_CREDENTIALS` , and requires no configuration here.
- **Manual:** This default option displays a **Service account credentials** field for you to enter the contents of your service account credentials file (a set of JSON keys), as downloaded from Google Cloud.

To insert the file itself, click the upload button at this field's upper right. As an alternative, you can use environment variables, as outlined [here](#).

- **Secret:** This option exposes a drop-down in which you can select a [stored secret](#) that references the service account credentials described above. A **Create** link is available to store a new, reusable secret.

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` .

Google Cloud Roles and Permissions

Your Google Cloud service account should have at least the following roles on topics:

- `roles/pubsub.publisher`
- `roles/pubsub.viewer` or `roles/viewer`

To enable LogStream's **Create topic** option, your service account should have one of the following (or higher) roles:

- `roles/pubsub.editor`
- `roles/editor`

Either `editor` role confers multiple permissions, including those from the lower `viewer`, `subscriber`, and `publisher` roles. For additional details, see the Google Cloud [Access Control](#) topic.

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 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.

Queue fallback behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** drops the newest events from being sent out of LogStream, and throws away incoming data, while leaving the contents of the PQ unchanged.

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 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.

Queue fallback behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** drops the newest events from being sent out of LogStream, and throws away incoming data, while leaving the contents of the PQ unchanged.

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 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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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.

Authentication method: See [Authentication Settings](#) below.

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 .

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** In the resulting **Auth token** field, you can optionally enter an auth token to use in the connection header.
- **Secret:** This option exposes an **Auth token (text secret)** drop-down, in which you can select a [stored secret](#) that references the `authToken` header field value described above. A **Create** link is available to store a new, reusable secret.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

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`

There's also this one, which deserves its own section:

- `__syslogout`

Exporting `__syslogout` vs. `_raw`

If you choose to send `__syslogout` to downstream services, it is exclusive – it becomes the entire syslog message sent. Neither `_raw`, nor any other metadata, will be sent downstream. The result will not be a proper syslog message unless you hand-build the event yourself.

Reconstructing `__syslogout`

You'll need to add `_time` to the payload. For example, in an [Eval](#) Function, you could use **Evaluate Fields** to build up `__syslogout` in an expression like this:

Name	Value Expression
__syslogout	<\${pri}>\${C.Time.strftime(_time,"%b %d %H:%M:%S")} \${host} \${appname}[\${procid}]: \${_raw}

Above, pri encodes the severity + facility ,according to the syslog protocol. Here's that example expression in a full Function:

3 Eval __syslog_test

Filter: __syslog_test

Description: Enter a description

Final: ☐ No

Evaluate Fields

Name	Value Expression
appname	'using_syslogout'
severity	1
facility	3
pri	(8 * facility) + severity
procid	'7777'
__syslogout	'<\${pri}>\${C.Time.strftime(_time,"%b %d %H:%M:%S")} \${host} \${appname}[\${procid}]: \${_raw}'

+ Add Field

Keep Fields: Enter field names

Remove Fields: Enter field names

Adding '_time __syslogout to construct a valid syslog message

Enhancing _raw with syslog Decorations

An easier approach is to put the message content in _raw , and construct the syslog "envelope" around _raw by including the severity , priority , facility , procid , msgid ,and appname fields as required.

Here's an alternative Eval Function that illustrates this:

4 Eval ! __syslog_test

Filter: ! __syslog_test

Description: Enter a description

Final: ☐ No

Evaluate Fields

Name	Value Expression
severity	1
facility	3
procid	8889
appname	'using_raw'

+ Add Field

Keep Fields: _raw *severity *facility *procid _time *appname

Remove Fields: *

Adding syslog decorations to _raw

Both Eval Functions are provided in this example Pipeline:

syslog_loop.json

```
{
  "id": "syslog_loop",
  "conf": {
    "output": "default",
    "groups": {},
    "asyncFuncTimeout": 1000,
    "functions": [
      {
        "filter": "true",
        "conf": {
          "mode": "reserialize",
          "type": "json",
          "srcField": "_raw",
          "dstField": "_raw",
          "keep": [
            "resource",
            "path",
            "httpMethod"
          ],
          "remove": []
        },
        "id": "serde",
        "disabled": false
      },
      {
        "filter": "true",
        "conf": {
          "clones": [
            {
              "__syslog_test": "true"
            }
          ]
        },
        "id": "clone",
        "disabled": false
      },
      {
        "filter": "__syslog_test",
        "conf": {
          "add": [
            {
              "name": "appname",
              "value": "'using_syslogout'"
            },
            {
              "name": "severity",
              "value": "1"
            },
            {
              "name": "facility",
              "value": "3"
            }
          ]
        }
      }
    ]
  }
}
```


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.

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.

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.
-

Advanced Settings

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.

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.

Remove staging dirs: Toggle to `Yes` to delete empty staging directories after moving files. This prevents the proliferation of orphaned empty directories.

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, drop, or queue incoming 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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

TLS Settings (Client Side)

Enabled Defaults to `No`. When toggled to `Yes`:

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

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. Defaults to `PLAIN` .
`SCRAM-SHA-256` and `SCRAM-SHA-512` are also available.

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Secret:** This option exposes a **Credentials secret** drop-down, in which you can select a [stored secret](#) that references the credentials described above. A **Create** link is available to store a new, reusable secret.

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. When toggled to `Yes`, see [Authentication Settings](#) below.

Backpressure behavior: Specify whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Secret:** This option exposes a **Credentials secret** drop-down, in which you can select a [stored secret](#) that references the credentials described above. A **Create** link is available to store a new, reusable secret.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle 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 resolution.
- `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. Enable Round-robin DNS to better balance distribution of events between Elasticsearch cluster nodes.

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`.)

Authentication method: See [Authentication Settings](#) below.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays a field for you to enter the **API key** for the team to which the dataset belongs.
- **Secret:** This option exposes a **API key (text secret)** drop-down, in which you can select a [stored secret](#) that references the API key described above. A **Create** link is available to store a new, reusable secret.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle 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. Enable Round-robin DNS to better balance distribution of events between destination cluster nodes.

New Relic Logs & Metrics

Cribl LogStream supports sending events to the [New Relic Log API](#) and the [New Relic Metric API](#).

- i** As of LogStream v.3.1.2, this Destination is updated to authenticate using New Relic's [Ingest License API key](#). (New Relic will retire the Insights Insert API keys, which this Destination previously used for authentication.)

Also as of v.3.1.2, LogStream provides a separate [New Relic Events](#) Destination that you can use to send ad hoc (loosely structured) events to New Relic via the [New Relic Event API](#).

Configuring Cribl LogStream to Output to New Relic

From the top nav of a LogStream instance or Group, select **Destinations**, then select **New Relic Ingest > Events** 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.

Authentication method: Select one of the following buttons.

- Manual:** This default option exposes an **API key** field. Directly enter your New Relic Ingest License API key, as you created or accessed it from New Relic's **account** drop-down. (For details, see the [New Relic API Keys](#) documentation.)

- **Secret:** This option exposes an **API key (text secret)** drop-down, in which you can select a [stored secret](#) that references a New Relic Ingest License API key. A **Create** link is available to store a new, reusable secret.

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`. For the `Persistent Queue` option, see the section just below.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle 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 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.

Verifying the New Relic Destination

Once you've configured log and/or metrics sources, create one or more Routes to send data to New Relic.

In New Relic, you can create visualizations incorporating the LogStream-supplied data, then add them to new or existing dashboards as widgets.

Logs and metrics land in two different places in New Relic.

Log Queries

To access and query log data:

- Navigate to the New Relic home screen's **Logs** header option, and click the (+) button at right.
 - Then to build your queries, use the **Find logs where** input field, and add desired columns to the table view below the graph,.
-

Metrics Queries

To access and query metrics data:

- From the New Relic home screen, *Click **Browse Data > Metrics > Can Search for metricNames**.
- Then, customize time range and dimensions to build the desired logic for your queries.
- Alternatively, you can use [NRQL](#) to build your own query searches.

New Relic Events

Cribl LogStream (v.3.1.2 and higher) supports sending events to New Relic via the [New Relic Event API](#). Use this Destination to export ad hoc events that New Relic ingestion treats as [custom events](#).

To export structured log and/or metric events, use LogStream's [New Relic Logs & Metrics](#) Destination.

Configuring LogStream to Output Events to New Relic

From the top nav of a LogStream instance or Group, select **Destinations**, then select **New Relic Ingest > Events** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Events > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Destination.

Authentication method: Select one of the following buttons.

- **Manual:** This default option exposes an **API key** field. Directly enter your New Relic Ingest License API key, as you created or accessed it from New Relic's **account** drop-down. (For details, see the [New Relic API Keys](#) documentation.)
- **Secret:** This option exposes an **API key (text secret)** drop-down, in which you can select a [stored secret](#) that references a New Relic Ingest License API key. A **Create** link is available to store a new, reusable secret.

Region: Select which New Relic region endpoint to use.

Account ID: Enter your New Relic account ID. (You can access this ID from New Relic's **account** drop-down, by selecting **Manage your plan**.)

Event type: Default `eventType` to apply when not specified in an event. You can use arbitrary values, as long as they do not conflict with New Relic [reserved words](#).

i Where an `eventType` is specified in an event, it will override this value.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`. For the `Persistent Queue` option, see the section just below.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Processing Settings

Post-Processing

Pipeline: Pipeline to process data before sending the data out via this output.

System fields: A list of fields to automatically add to events that use this output. By default, includes `cribl_pipe` (identifying the 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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Defaults to `Yes`, meaning compress the payload body before sending.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests 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.

Max events per request: Maximum number of events to include in the request body. Defaults to `0`, allowing unlimited events.

Flush period (sec): Maximum time between requests. Low values can cause the payload size to be smaller than the configured **Max body size**. Defaults to `1` second.

Extra HTTP headers: Click **+ Add Header** to insert extra headers as **Name/Value** pairs.

Verifying the New Relic Events Destination

Once you've configured event sources, create one or more Routes to send data to New Relic.

In New Relic, you can create visualizations incorporating the LogStream-supplied data, then add them to new or existing dashboards as widgets.

Alternatively, in the New Relic backend, you can select **Query you data** (top nav) > **Events** (left tab), and then select the event type you exported from LogStream.

To view more events, change the time frame at the upper right. To see raw events, click **Raw data** on the right.

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](#) (versions 1.x and 2.0.x) and [InfluxDB Cloud](#).

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. When toggled to `Yes` , see [Authentication Settings](#) below.

Backpressure behavior: Select whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
 - **Secret:** This option exposes a **Credentials secret** drop-down, in which you can select a [stored secret](#) that references the credentials described above. A **Create** link is available to store a new, reusable secret.
-

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle 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.

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.

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 one of these options:

Auto: This default option uses the AWS instance's metadata service to automatically obtain short-lived credentials from the IAM role attached to an EC2 instance. The attached IAM role grants LogStream Workers access to authorized AWS resources. Can also use the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` . Works only when running on AWS.

Manual: If not running on AWS, you can select this option to enter a static set of user-associated IAM credentials (your access key and secret key) directly or by reference. This is useful for Workers not in an AWS VPC, e.g., those running a private cloud. The **Manual** option exposes these corresponding additional fields:

- **Access key:** Enter your AWS access key. If not present, will fall back to the `env.AWS_ACCESS_KEY_ID` environment variable, or to the metadata endpoint for IAM role credentials.
- **Secret key:** Enter your AWS secret key. If not present, will fall back to the `env.AWS_SECRET_ACCESS_KEY` environment variable, or to the metadata endpoint for IAM credentials.

Secret: If not running on AWS, you can select this option to supply a stored secret that references an AWS access key and secret key. This option exposes a **Secret key pair** drop-down, in which you can select a [stored secret](#) that references the set of user-associated IAM credentials described above. A **Create** link is available to store a new, reusable secret.

Processing Settings

Post-Processing

Pipeline: Pipeline 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 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.

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.

Remove staging dirs: Toggle to `Yes` to delete empty staging directories after moving files. This prevents the proliferation of orphaned empty directories.

Region: Region where the MinIO service/cluster is located. Leave blank when using a containerized MinIO.

Object ACL: ACL (Access Control List) to assign to uploaded objects. Defaults to `Private`.

Storage class: Select a storage class for uploaded objects. Defaults to `Standard`.

Server-side encryption: Server side encryption type for uploaded objects. Defaults to `none`.

Signature version: Signature version to use for signing MinIO requests. Defaults to `v4`.

Reuse connections: Whether to reuse connections between requests. 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`

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.

Authentication method: See [Authentication Settings](#) below.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays an **API key** field for you to enter your Wavefront API auth token. See Wavefront's [Generating an API Token](#) topic.
- **Secret:** This option exposes an **Auth token (text secret)** drop-down, in which you can select a [stored secret](#) that references the API auth token described above. A **Create** link is available to store a new, reusable secret.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle 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 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.

Authentication method: See [Authentication Settings](#) below.

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` .

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays an **Auth token** field for you to enter your SignalFx API access token. See SignalFx's [Manage Tokens](#) topic.
 - **Secret:** This option exposes an **Auth token (text secret)** drop-down, in which you can select a [stored secret](#) that references the API access token described above. A **Create** link is available to store a new, reusable secret.
-

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

Compress: Toggle 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 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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

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. Enable Round-robin DNS to better balance distribution of events between destination cluster nodes.

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.

Authentication method: See [Authentication Settings](#) below.

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` .

Authentication Settings

Use the **Authentication method** buttons to select one of these options:

- **Manual:** Displays a field for you to enter an **API key** that is available in your Datadog profile.
- **Secret:** This option exposes an **API key (text secret)** drop-down, in which you can select a [stored secret](#) that references the API access token described above. A **Create** link is available to store a new, reusable secret.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

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](#)

Prometheus

Cribl LogStream can send metric events to targets and third-party platforms that support Prometheus' [remote write](#) specification. This is a streaming Destination type.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

Use the **Authentication type** buttons to select one of these options:

- **None:** Don't use authentication.
- **Auth token:** Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Auth token (text secret):** This option exposes a **Token (text secret)** drop-down, in which you can select a [stored text secret](#) that references the bearer token described above. A **Create** link is available to store a new, reusable secret.
- **Basic:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
- **Basic (credentials secret):** This option exposes a **Credentials secret** drop-down, in which you can select a [stored text secret](#) that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

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. The default expression – `name.replace(/\\./g, '_\\')` – replaces all `.` characters in a metric's name with the Prometheus-supported `_` character. Use the `name` global variable to access the metric's name. You can access event fields' values via `__e.<fieldName>`.

Send metadata: Whether to generate and send metrics' metadata (`type` and `metricFamilyName`) along with the metrics. 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. Enable Round-robin DNS to better balance distribution of events between Prometheus cluster nodes.

Grafana Cloud

Cribl LogStream can send data to two of the services available in [Grafana Cloud](#): [Loki](#) for logs and [Prometheus](#) for metrics. The Grafana Cloud Destination shapes events appropriately for Loki and Prometheus, and routes events to the correct endpoint for each service. This is a streaming Destination type.

Preparing Prometheus and Loki to Receive Data from LogStream

To define a Grafana Cloud Destination, you need a [Grafana Cloud account](#).

While logged in to your Grafana account, navigate to the Grafana Cloud Portal, which should be located at `https://grafana.com/orgs/<your-organization-name>`, and complete the following steps.

Obtain an API key, setting its Role to `MetricsPublisher`. If you want LogStream or an external KMS to manage the API key, [configure](#) a key pair that references the API key.

In the Prometheus tile, click **Send Metrics** to open the Prometheus configuration page. Write down:

- Your **Remote Write Endpoint** URL, for example: `https://prometheus-blocks-prod-us-central1.grafana.net/api/prom/push`.
- Your Prometheus **Username**.

In the Loki tile, click **Send Logs** to open the Loki configuration page. Write down:

- Your **Grafana Data Source settings** URL, for example: `https://logs-prod-us-central1.grafana.net`.
- Your Loki **User ID**.

Decide what type of authentication to use and prepare accordingly:

- If you choose Basic authentication, the username (**Username** in Prometheus, **User** in Loki) and password (simply your Grafana API key) will remain separate.
- If you choose token-based authentication, construct your tokens by concatenating username, colon (:), and password, for example `12345:c0QvDj6sJGFS3Bk2MguBW==` . Because the Prometheus and Loki usernames differ, you need to construct a separate token for each service.

Configuring Cribl LogStream to Output to Grafana Cloud

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Grafana Cloud** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Grafana Cloud > New Destination** modal, which provides the following fields.

General Settings


Output ID: Enter a unique name to identify this Grafana Cloud output definition.

Loki URL: The endpoint to send log events to, e.g.: `https://logs-prod-us-central1.grafana.net` . This is the **Grafana Data Source settings** URL you wrote down earlier.

Prometheus URL: The endpoint to send metric events to, e.g.: `https://prometheus-blocks-prod-us-central1.grafana.net/api/prom/push` . This is the **Remote Write Endpoint** URL you wrote down earlier.

Backpressure behavior: Whether to block, drop, or queue events when all receivers are exerting backpressure.

Persistent Queue Settings

-  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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

The **Authentication** tab provides separate **Loki** and **Prometheus** sections, enabling you to configure these inputs separately. The two sections provide identical options.

Use the **Authentication method** buttons to select one of these options:

- **Auth token:** Enter the bearer token that must be included in the authorization header. Use the token that you constructed earlier. In Grafana Cloud, the bearer token is generally built by concatenating the username and the API key, separated by a colon. E.g.: `<your-username>:<your-api-key>` .
- **Auth token (text secret):** This option exposes a drop-down in which you can select a [stored text secret](#) that references the bearer token described above. A **Create** link is available to store a new, reusable secret.
- **Basic:** This default option displays fields for you to enter HTTP Basic authentication credentials. **Username** is the Loki **User** or Prometheus **Username** that you wrote down earlier. **Password** is your API key in the Grafana Cloud domain.

- **Basic (credentials secret):** This option exposes a **Credentials secret** dropdown, in which you can select a [stored text secret](#) that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

Processing Settings

Metric events can have dimensions, and log events have labels. Dimensions, labels, and their values are determined by several different settings in LogStream. This section explains how that works, along with other kinds of settings.

Loki uses labels to define separate streams of logging data. This is a key concept. Cribl recommends that you familiarize yourself with the [information](#) and documentation Grafana provides about labels in Loki.

One canonical example is processing logs from servers in three environments: production, staging, and testing. You could create a label named `env` whose possible values are `prod`, `staging`, and `test`.

One basic principle is that if you set too many labels, you can end up with too many streams.

Post-Processing

Pipeline: Pipeline to process data before sending the data out using this output.

System fields: A list of fields to automatically add to events that use this output—both metric events, as dimensions; and, log events, as labels. Supports wildcards.

By default, includes `cribl_host` (LogStream Node that processed the event) and `cribl_wp` (LogStream Worker Process that processed the event). On the Loki side, this creates different streams, which prevents Loki from rejecting some events as being out of order when different Nodes or Worker Processes are emitting at different rates.

Other options include:

- `cribl_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

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 `Yes` .

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.


Compress: When the **Message format** is `JSON` , you can toggle this slider to `Yes` to GZIP-compress the data before sending to Grafana Cloud. (Applies only to Loki's JSON payloads. This slider is hidden when the **Message format** is `Protobuf` , because both Prometheus' and Loki's Protobuf implementations are [Snappy](#)-compressed by default.)

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.

 Loki and Prometheus might complain about entries being delivered out of order when **Request concurrency** is set `> 1` and any of **Flush period (sec)**, **Max body size (KB)**, or **Max events per request** are set to low values.

Flush period (sec): Maximum time between requests. Low values could cause the payload size to be smaller than its configured maximum. Defaults to `1` .

Extra HTTP headers: Name/Value pairs to pass as additional HTTP headers.

Metric renaming expression: A JavaScript expression that can be used to rename metrics. The default expression – `name.replace(/\\./g, '_\\')` – replaces all `.` characters in a metric's name with the Prometheus-supported `_` character. Use the `name` global variable to access the metric's name. You can access event fields' values via `__e.<fieldName>` .

Message format: Whether to send events as `Protobuf` (the default) or `JSON` .

Logs message field: The event field to send as log output, for example: `_raw` . All other event fields are discarded. If left blank, LogStream sends a JSON representation of the whole event.

Logs labels: Name/value pairs where the value can be a static or dynamic expression that has access to all log event fields.

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 Prometheus as a metric event. If `__criblMetrics` is absent, LogStream will treat the event as a log and send it to Loki.

The internal field `__labels` specifies labels to add to log events. If a label is set in both the `__labels` field and in **Logs labels** and/or **System fields**, LogStream sends the value from `__labels` to Loki. Setting the `__labels` field in a Pipeline gives you a quick way to experiment with the logs being sent.

If there are no labels set (this would happen when **System fields**, **Logs labels**, and `__labels` are all empty), LogStream adds a default `source` label, which prevents Loki from rejecting events. The `source` label the concatenation of `cribl` , underscore (`_`), source type, colon (`:`), source-name, where source name and type are values in the `__inputId` event field, for example:
`cribl_metrics:in_prometheus_rw` . If `__inputId` is missing, `source` is set to `cribl` .

Notes on HTTP-based Outputs

- The **Advanced Settings > Compress** toggle determines whether to compress the payload body before sending to Loki only. The toggle setting does not apply to Prometheus payloads, which are always compressed using [Snappy](#).
- 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. Enable Round-robin DNS to better balance distribution of events between Grafana Cloud nodes.

Loki

Cribl LogStream can send log events to Grafana's [Loki](#) log aggregation system. This is a streaming Destination type.

Configuring Cribl LogStream to Output to Loki

From the top nav of a LogStream instance or Group, select **Destinations**, then select **Loki** from the **Data Destinations** page's tiles or the **Destinations** left nav. Click **+ Add New** to open the **Destinations > Loki > New Destination** modal, which provides the following fields.

General Settings

Output ID: Enter a unique name to identify this Loki output definition.

Loki URL: The endpoint to send events to, e.g.: `https://logs-prod-us-central1.grafana.net`.

Backpressure behavior: Whether to block, drop, or queue events when all receivers are exerting backpressure.

Authentication

Use the **Authentication type** buttons to select one of these options:

- **None:** Don't use authentication.
- **Auth token:** Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
- **Auth token (text secret):** This option exposes a drop-down in which you can select a [stored text secret](#) that references the bearer token described above. A **Create** link is available to store a new, reusable secret.

- **Basic:** This default option displays fields for you to enter HTTP Basic authentication credentials. **Username** is the Loki **User**. **Password** is your API key in the Grafana Cloud domain.
- **Basic (credentials secret):** This option exposes a **Credentials secret** drop-down, in which you can select a [stored text secret](#) that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.

Persistent Queue Settings

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Processing Settings

Loki uses labels to define separate streams of logging data. This is a key concept. Cribl recommends that you familiarize yourself with the [information](#) and documentation Grafana provides about labels in Loki.

One canonical example is processing logs from servers in three environments: production, staging, and testing. You could create a label named `env` whose possible values are `prod`, `staging`, and `test`.

One basic principle is that if you set too many labels, you can end up with too many streams.

Post-Processing

Pipeline: Pipeline to process data before sending the data out using this output.

System fields: A list of fields to automatically add to log events as labels. Supports wildcards.

By default, includes `cribl_host` (LogStream Node that processed the event) and `cribl_wp` (LogStream Worker Process that processed the event). On the Loki side, this creates different streams, which prevents Loki from rejecting some events as being out of order when different Nodes or Worker Processes are emitting at different rates.

Other options include:

- `cribl_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

Compress: When the **Message format** is `JSON`, you can toggle this slider to `Yes` to GZIP-compress the data before sending to Loki. (When the **Message format** is `Protobuf`, data is always [Snappy](#)-compressed, so this slider is hidden.)

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

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`.

Request timeout: Amount of time (in seconds) to wait for a request to complete before aborting it. Defaults to `30`.

Request concurrency: Maximum number of concurrent requests before blocking. This is set per Worker Process. Defaults to 1 .

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 15 .

Extra HTTP headers: Name/Value pairs to pass as additional HTTP headers.

Message format: Whether to send events as Protobuf (the default) or JSON .

Logs message field: The event field to send as log output, for example: `_raw` . All other event fields are discarded. If left blank, LogStream sends a JSON or Protobuf representation of the whole event.

Logs labels: Name/value pairs where the value can be a static or dynamic expression that has access to all log event fields.

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.

Queue-full behavior: Whether to block or drop events when the queue is exerting backpressure (because disk is low or at full capacity). **Block** is the same behavior as non-PQ blocking, corresponding to the **Block** option on the **Backpressure behavior** drop-down. **Drop new data** throws away incoming data, while leaving the contents of the PQ unchanged.

Authentication

Use the **Authentication type** buttons to select one of these options:

- **None:** Don't use authentication.
 - **Auth token:** Use HTTP token authentication. In the resulting **Token** field, enter the bearer token that must be included in the HTTP authorization header.
 - **Auth token (text secret):** This option exposes a **Token (text secret)** drop-down, in which you can select a [stored text secret](#) that references the bearer token described above. A **Create** link is available to store a new, reusable secret.
 - **Basic:** Displays **Username** and **Password** fields for you to enter HTTP Basic authentication credentials.
 - **Basic (credentials secret):** This option exposes a **Credentials secret** drop-down, in which you can select a [stored text secret](#) that references the Basic authentication credentials described above. A **Create** link is available to store a new, reusable secret.
-

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

Validate server certs: Toggle to `Yes` to reject certificates that are **not** authorized by a CA in the **CA certificate path**, nor by another trusted CA (e.g., the system's CA).

Round-robin DNS: Toggle to `Yes` to use round-robin DNS lookup. When a DNS server returns multiple addresses, this will cause LogStream to cycle through them in the order returned.

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.

Use Cases

See these examples of configuring a Webhook Destination to integrate with specific services:

- [Webhook/BigPanda Integration](#)
- [Webhook/Sumo Logic Integration](#)

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. Enable Round-robin DNS to better balance distribution of events between destination cluster nodes.

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 [Help](#) 

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 LogStream deployment of any size, 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 a Pipeline's configuration references Lookup file(s), LogStream will import the Pipeline only if the Lookups are available in their configured locations. 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.

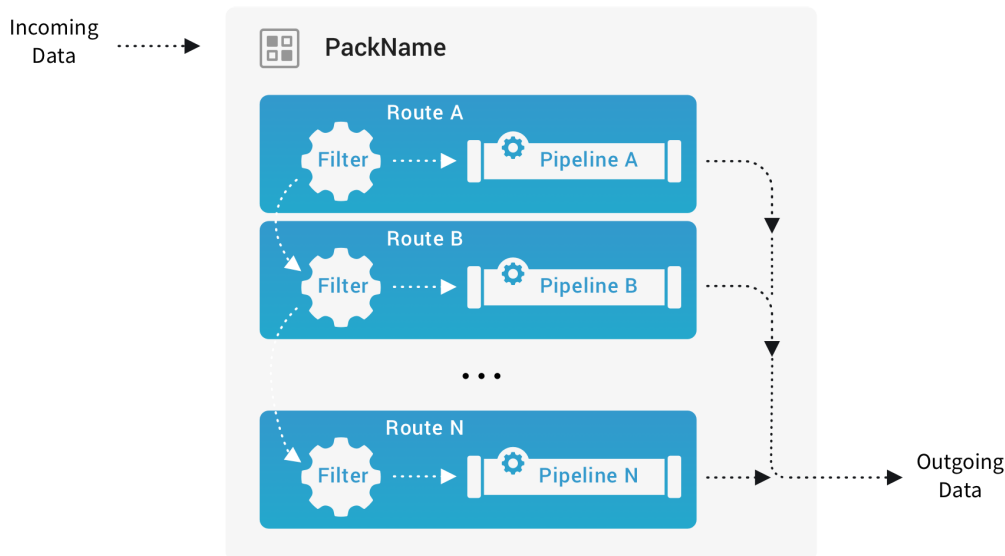
What Is a Pack?

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.

Where Can I Get Some Packs?

Easy now. See [The Cribl Pack Dispensary™](#) below.

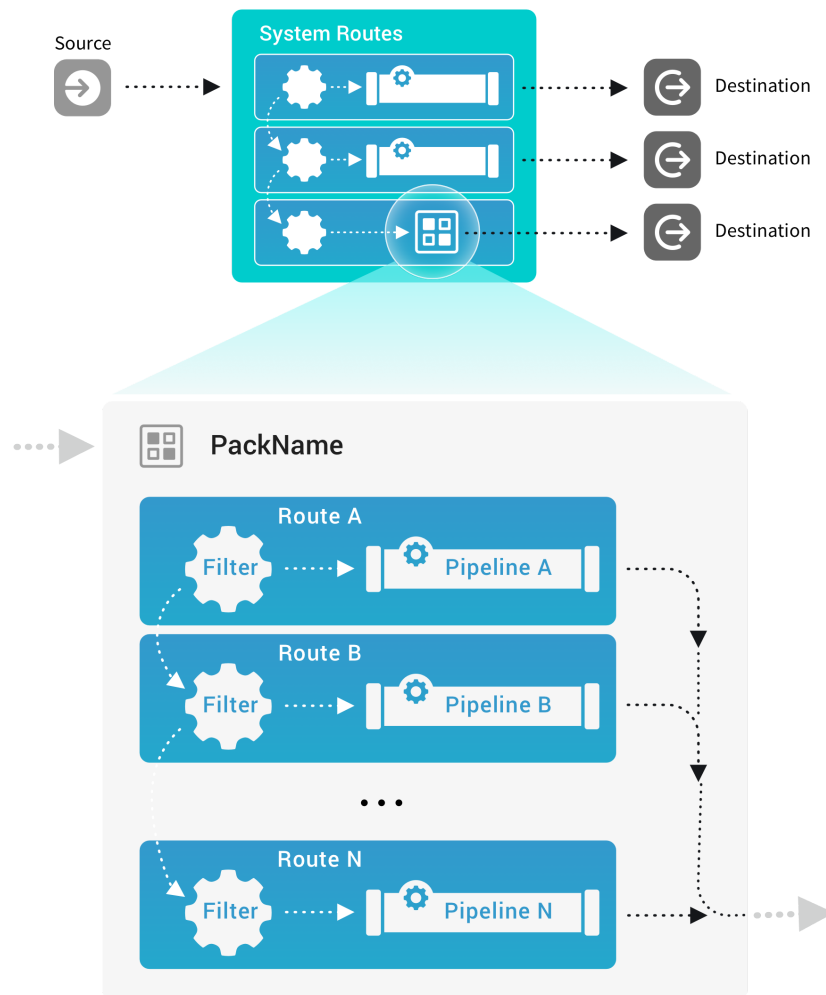
Using Packs

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 the display with a **PACK** badge, as you can see here in the Routing table:

#	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

The **PACK** badge is also displayed when you click into a resource – shown here on one of the Routes from the above table:

#	Route	Filter	Pipeline/Output	Bytes	In	Out	Dropped	⌵ All
2	Palo Alto Firewall Tra...	sourcetype=='pan:...	PACK pan-firewall-tr... router:pan-firewall-ex...	1.071%				On ...

Route Name* Palo Alto Firewall Traffic

Filter ② `sourcetype=='pan:traffic'`

Pipeline* ② `PACK pan-firewall-traffic`

Output ② router:pan-firewall-external

Description ② Enter a description

Final ② ☒ Yes

PACK badge on a Pack connected to a Route

LogStream's **Monitoring** page includes a **Packs** link where you can monitor Packs' throughput.

Accessing Packs

You access Packs differently, depending on your [deployment type](#).

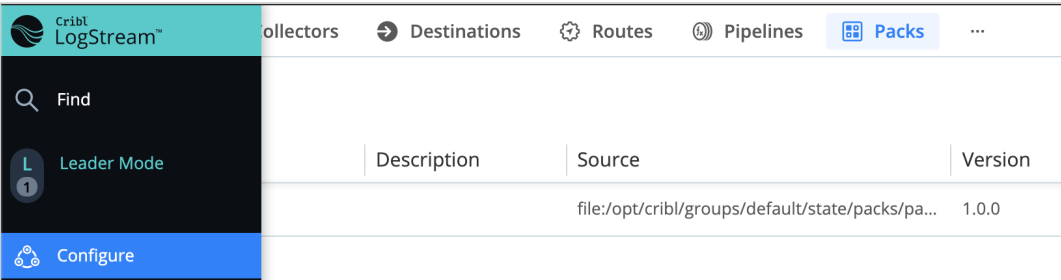
Single-Instance

In a single-instance deployment, Packs are global. From LogStream's top-level navigation, just click **Packs**.

Packs, single-instance navigation

Distributed/Default Worker Group

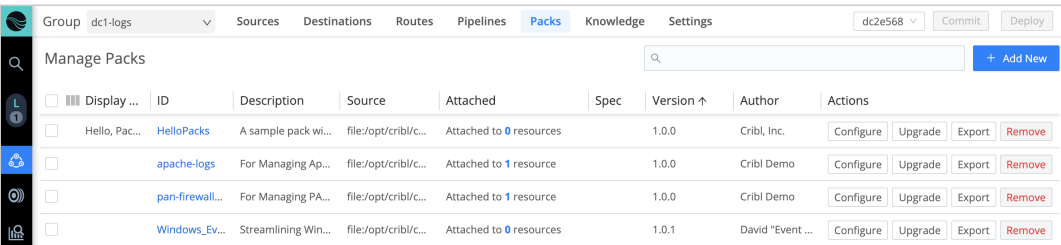
In a distributed deployment with the default single Worker Group (Leader mode), select **Configure** from the left nav, then **Packs** from the resulting top nav.



Packs, Leader mode

Distributed/Mutliple Worker Groups

In a distributed deployment with multiple Worker Groups (Leader mode), Packs are associated with (and installed within) Worker Groups. Navigate to the parent Worker Group, then select **Packs** from that Group's top nav.



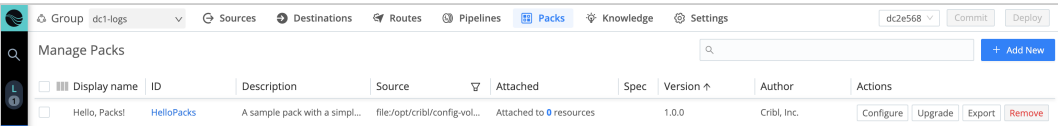
Worker Group > Manage Packs page

□ 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

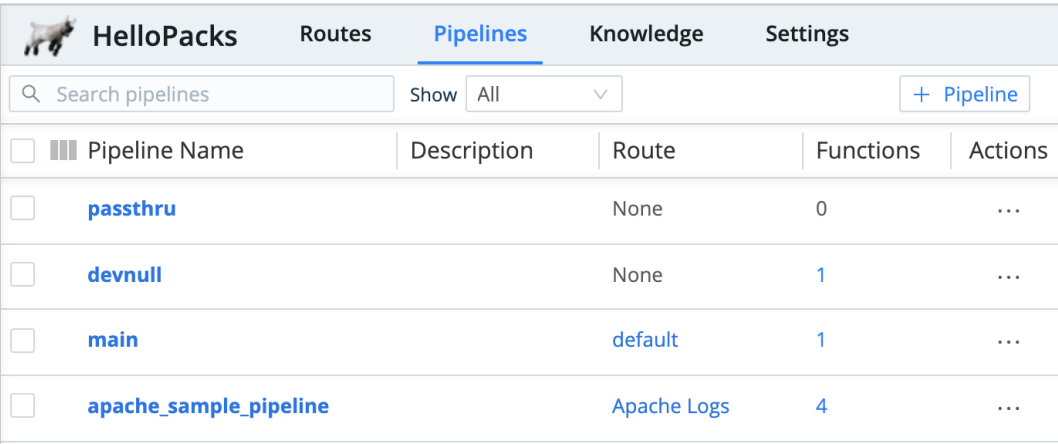
To unpack Packs, use the above instructions (per deployment type) to navigate to the **HelloPacks** example Pack shipped with LogStream. On the **Manage Packs** page, click this Pack's row to see its Pack's configuration.



Display name	ID	Description	Source	Attached	Spec	Version	Author	Actions
Hello, Packs!	HelloPacks	A sample pack with a simple...	file/opt/cribl/config.vol...	Attached to 0 resources		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.

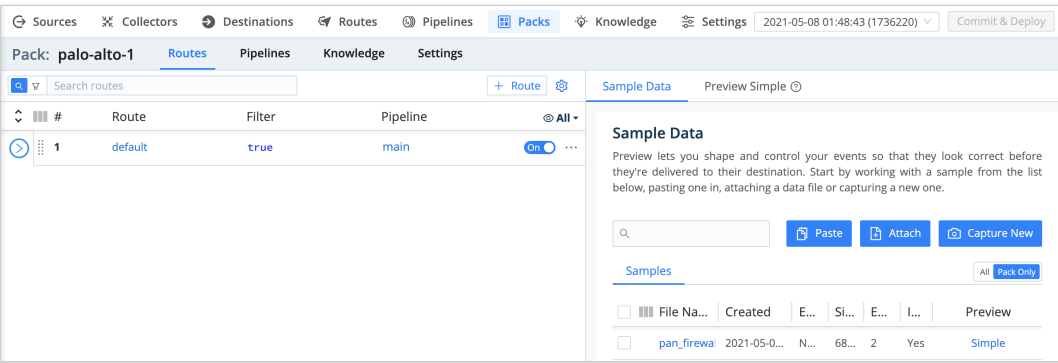


Pipeline Name	Description	Route	Functions	Actions
passthru		None	0	...
devnull		None	1	...
main		default	1	...
apache_sample_pipeline		Apache Logs	4	...

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	On/Off
1	default	true	main	On

File Na...	Created	E...	Si...	E...	I...	Preview
pan_firewa	2021-05-0...	N...	68...	2	Yes	Simple

Configuring a Pack

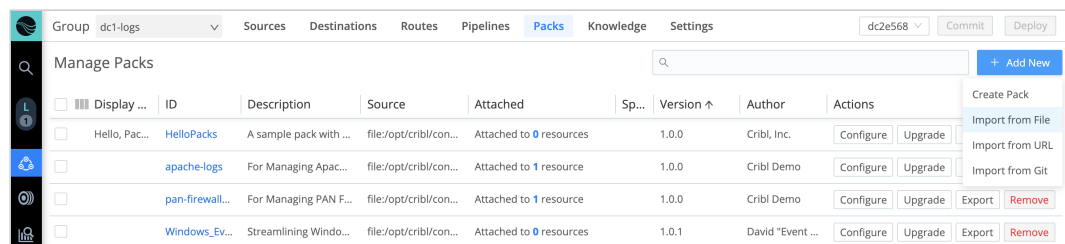
The left pane's links give you access to configuration objects specific to this Pack. In the right pane, you can toggle between displaying **All** sample files available on your LogStream instance, versus samples internal to the the **Pack Only**.

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. Select your desired **Import from source**: [File](#), [URL](#), or [Git repo](#).



Importing a Pack

⚠ Custom Functions

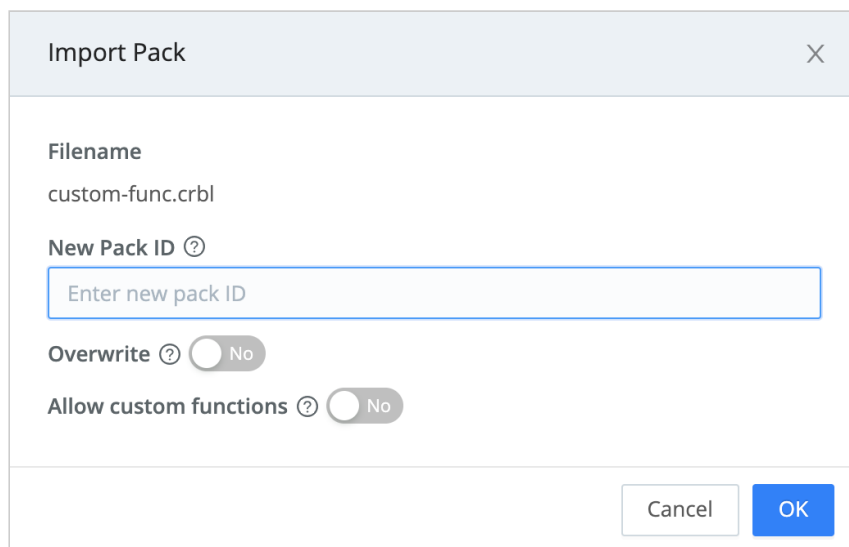
Packs can include Pipelines containing custom functions, which can (in turn) run arbitrary JavaScript. Before you install a Pack, make sure it comes from a provider you trust, such as the [Cribl Pack Dispensary](#) or your own organization.

LogStream 3.1.2 and higher provide an additional layer of protection: All Pack import modals provide an **Allow custom functions** slider. In the slider's default **No** position, if LogStream detects custom functions in the specified Pack, it will block the import with an error message. If you trust the Pack's provider, toggle the slider to **Yes**, and the import will proceed normally.

Import from File

To import a Pack (`.crbl` file) from your local filesystem:

1. From the + **Add New** submenu, select **Import from File**.
2. From the resulting File Open dialog, select the file to import.
3. Optionally, give the pack an explicit, unique **New Pack ID**. (For details about this option, see [Upgrading an Existing Pack](#) below.)
4. Where appropriate (see just [above](#)), enable **Allow custom functions**.
5. Click **OK** to confirm the import.



Import Pack

Filename
custom-func.crbl

New Pack ID ?
Enter new pack ID

Overwrite ? ☐ No

Allow custom functions ? ☐ No

Cancel OK

Importing from a file

Import from URL

To import a Pack from a known, public or internal, URL:

1. From the + **Add New** submenu, select **Import from URL**.
2. Enter a valid URL for the Pack's source. (This field's input is validated for URL format, but not for accuracy, before you submit the modal.)
3. Optionally, give the pack an explicit, unique **New Pack ID**. (See [Upgrading an Existing Pack](#).)
4. Where appropriate, enable **Allow custom functions**. (See [Custom Functions](#).)
5. Click **OK** to confirm the import.

Import from URL

URL* ?

New Pack ID ?

Enter new pack ID

Overwrite ? ☐ No

Allow custom functions ? ☐ No

Cancel OK

Optionally, import the Pack with a different, unique ID.

Confirming file import from URL

- To import a Pack from a public URL, LogStream's Leader Node (or single instance) requires Internet access. A [distributed deployment](#)'s Leader can then deploy the Pack to Workers even if the Workers lack Internet access.

Import from Git Repos

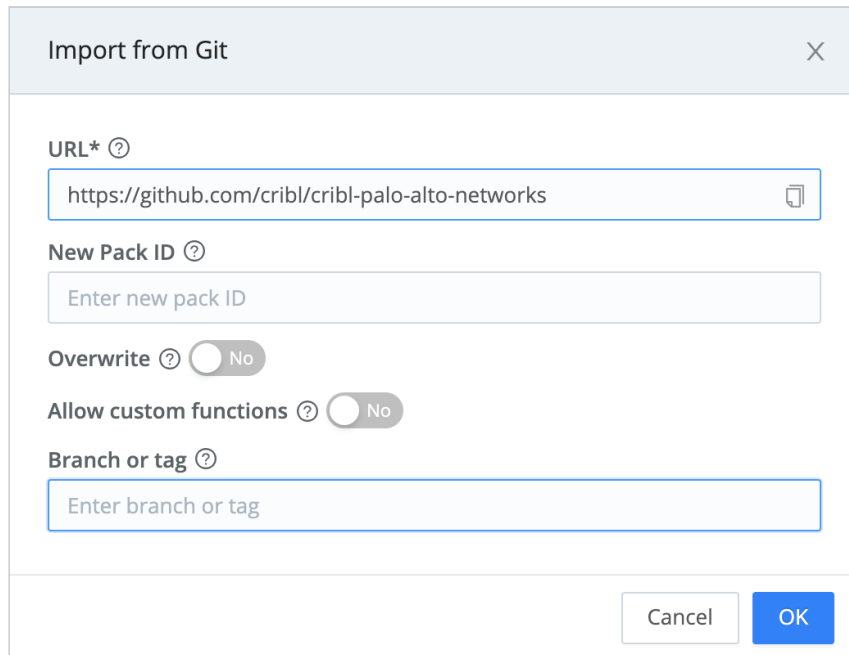
To import a Pack from a known public or private Git repo:

1. From the **+ Add New** submenu, select **Import from Git**.
2. Enter the source repo's valid URL.

This field's input is validated for URL format, but not for completeness or accuracy, before you submit the modal. When targeting a private repo, use the format: `https://<username>:<token/password>:<repo-address>` . Public repos need only `https://<repo-address>` , as shown in the example below.

3. Optionally, give the pack an explicit, unique **New Pack ID**. (See [Upgrading an Existing Pack](#).)
4. Optionally, enter a **Branch or tag** to filter the import source using the repo's metadata. You can specify a branch (such as `master`) or a tag (such as a release number: `0.5.1` , etc.).

5. Where appropriate (see [Custom Functions](#)), enable **Allow custom functions**.
6. Click **OK** to confirm the import.



Importing from a Git repo

- To import a Pack from a public repo, LogStream's Leader Node (or single instance) requires Internet access. A [distributed deployment](#)'s Leader can then deploy the Pack to Workers even if the Workers lack Internet access.

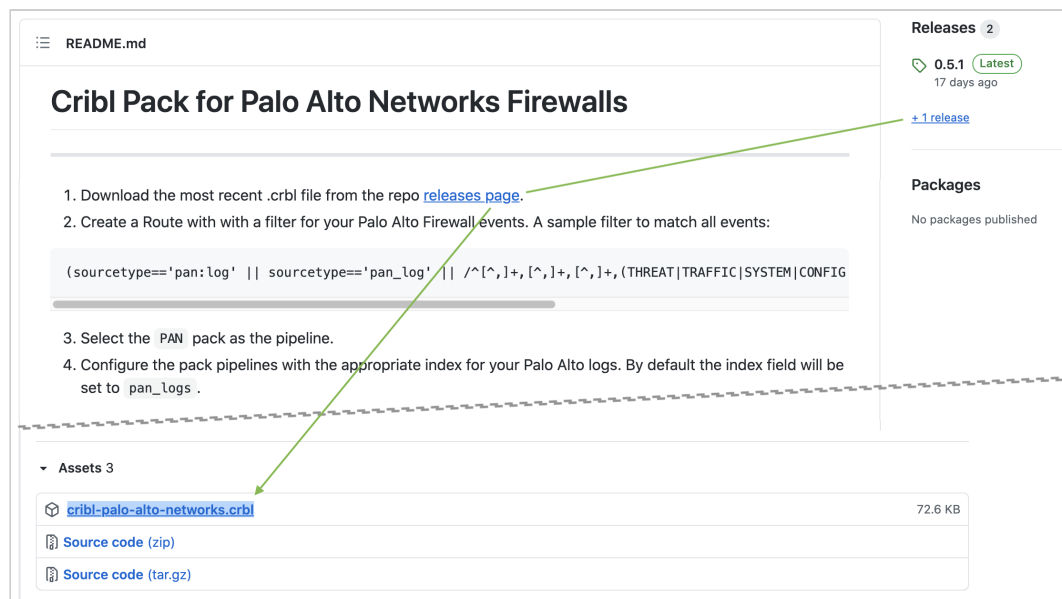
The Cribl Pack Dispensary™

You might be wondering, "This **Import from Git** option is nice, but how do I discover a reliable repo from which to pull Packs that add useful features to LogStream?"

For starters, Cribl is proud to point you to the [Cribl Pack Dispensary™](#). Here, Cribl's own engineers have seeded several strains of high-productivity LogStream configurations. Because this repo is a place to share good stuff, we expect many new hybrids to sprout from the community. Cribl will test and curate submissions to ensure the quality of the repo's contents.

You can install Dispensary Packs directly through LogStream's UI (see [Import from Git Repos](#) above). However, if you prefer, you can click through to any

Dispensary repo's release page, download the corresponding `.crbl` file, and then [upload the file](#) into LogStream.



Downloading a `.crbl` file from the Cribl Pack Dispensary's Web UI

Upgrading an Existing Pack

Each Pack that is installed within a given Worker Group (or single-instance deployment) must have a unique ID. The ID is based on the Pack's internal configuration – not its container's file name, nor on its Display name.

If you import a Pack whose internal ID matches an installed Pack – whether an update, or just a duplicate – you'll be prompted to assign a unique **New Pack ID** to import it as a separate Pack.

Import Pack

Filename

HelloPacks.crbl

New Pack ID ?

Enter new pack ID

Overwrite ?

No

Allow custom functions ?

No

Cancel

OK

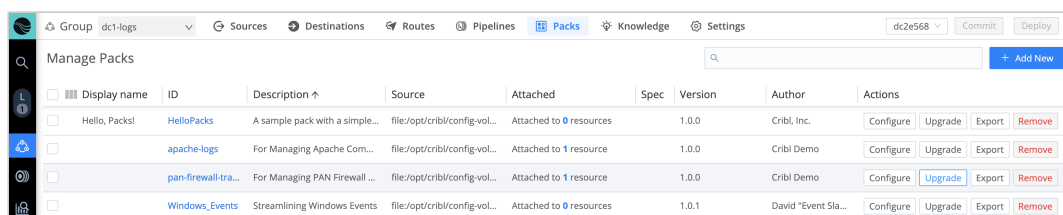
Renaming a Pack on import

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.

Each Pack within a LogStream instance must have a unique **Pack ID**, so you cannot share an ID between two (or more) installed Packs.

To explicitly upgrade an existing Pack, you can instead click the **Upgrade** button on its row.



Display name	ID	Description	Source	Attached	Spec	Version	Author	Actions
Hello, Packs!	HelloPacks	A sample pack with a simple...	file:/opt/cribl/config-vol...	Attached to 0 resources		1.0.0	Cribl, Inc.	Configure Upgrade Export Remove
apache-logs	apache-logs	For Managing Apache Com...	file:/opt/cribl/config-vol...	Attached to 1 resource		1.0.0	Cribl Demo	Configure Upgrade Export Remove
pan-firewall-tra...	pan-firewall-tra...	For Managing PAN Firewall ...	file:/opt/cribl/config-vol...	Attached to 1 resource		1.0.0	Cribl Demo	Configure Upgrade Export Remove
Windows_Events	Windows_Events	Streamlining Windows Events	file:/opt/cribl/config-vol...	Attached to 0 resources		1.0.1	David "Event Sla...	Configure Upgrade Export Remove

Upgrading an existing Pack

- If you've modified an installed Pack, LogStream will block 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.
Each Pack within a LogStream instance must have a separate **Pack ID**, but you can assign arbitrary **Display names**.
5. Click **OK** to save the Pack.

New Pack

X

Display name ?

Custom Pack

Pack ID* ?

Custom_Pack

Version* ?

0.0.1

Description ?

Encapsulates 42 amazing features

Author ?

Cribl.io

Cancel

OK

Creating a Pack

6. On the **Manage Packs** page, click the new Pack's row to open the Pack.

Manage Packs									
<input type="checkbox"/>	Display name	ID	Description ↑	Source	Attached	Spec	Version	Author	Actions
<input type="checkbox"/>	Hello, Packs!	HelloPacks	A sample pack with a simple...	file:/opt/cribl/config-vo...	Attached to 0 resources		1.0.0	Cribl, Inc.	Configure Upgrade Export Remove
<input type="checkbox"/>	Custom Pack	Custom_Pack	Encapsulates 42 amazing fe...	file:/opt/cribl/config-vo...	Attached to 0 resources		0.0.1	Cribl.io	Configure Upgrade Export Remove
<input type="checkbox"/>	apache-logs		For Managing Apache Com...	file:/opt/cribl/config-vo...	Attached to 1 resource		1.0.0	Cribl Demo	Configure Upgrade Export Remove
<input type="checkbox"/>	pan-firewall-tra...		For Managing PAN Firewall ...	file:/opt/cribl/config-vo...	Attached to 1 resource		1.0.0	Cribl Demo	Configure Upgrade Export Remove
<input type="checkbox"/>	Windows_Events		Streamlining Windows Events	file:/opt/cribl/config-vo...	Attached to 0 resources		1.0.1	David "Event Sla...	Configure Upgrade Export Remove

Manage Packs page

7. 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.) and display settings. If you're developing a new Pack to share, you'll want to use this interface to populate the Pack's README and display logo.

1. From the Pack's submenu, select Settings.

	#	Route	Filter	Pipeline	
>	1	default	true	main	On <input type="checkbox"/> ...

Pack Settings

2. To populate the Pack's README file, toggle **View** to **Edit**, replace the placeholder markdown content, and **Save**.

View **Edit**

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

Edit

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

Pack info

Version ⓘ

1.0.1

Description ⓘ

Encapsulates 42 amazing features

Author ⓘ

Cribl.io

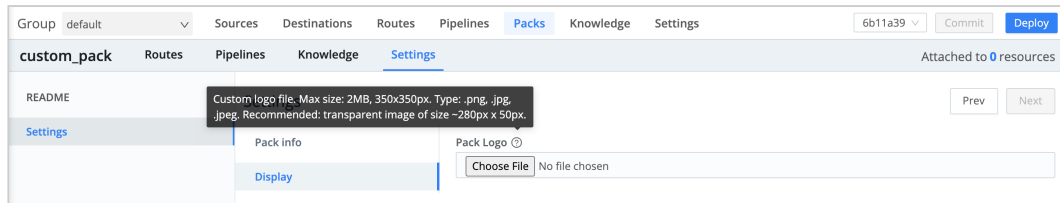
Prev Next

Editing Pack's metadata

4. To add a Pack logo, click the Pack's **Settings > Display** left tab.

Cribl recommends adding a logo to each custom Pack, to visually distinguish the Pack's UI from the surrounding LogStream UI (as well as

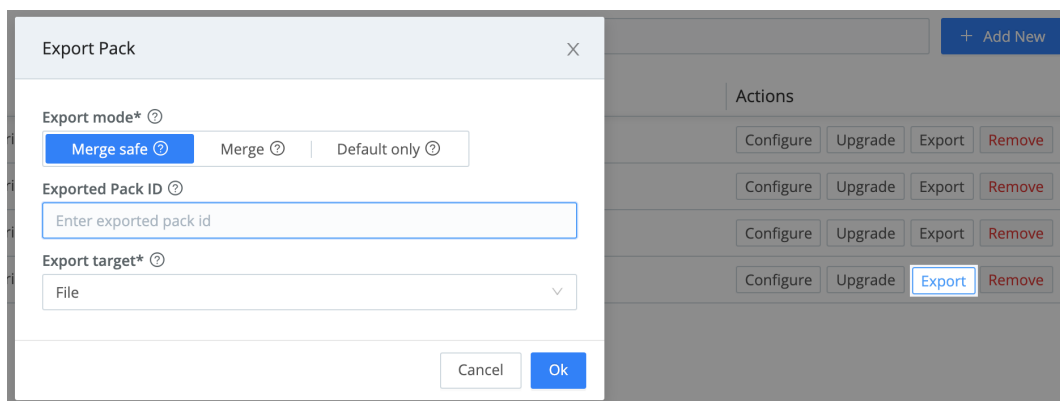
from other Packs). You can upload a .png or .jpg / .jpeg file, up to a maximum size of 2MB and 350x350px. Cribl recommends a transparent image, sized approximately 280x50px.



Editing Pack's display (logo) settings

Exporting a Pack

To export a newly created or modified Pack, click its **Export** button on the Packs page.



Exporting a Pack

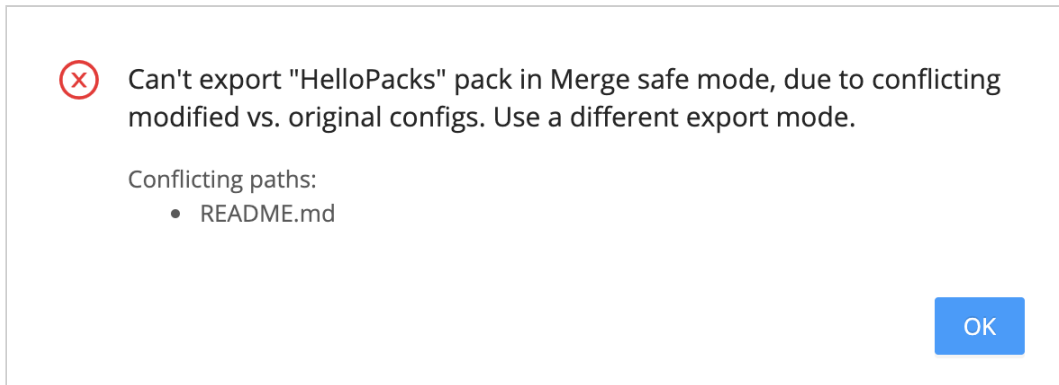
The resulting Export Pack modal provides the following options.

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:



Merge safe error

If you encounter this error, use the **Merge** or **Default only** export mode instead.

Export Target

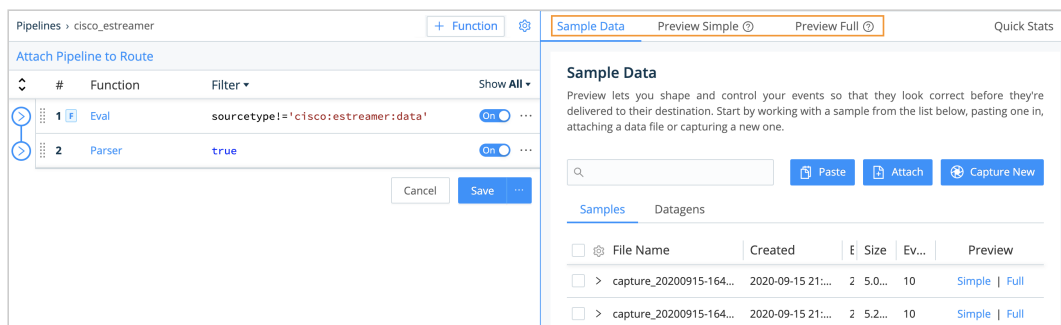
The options here are:

- **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.

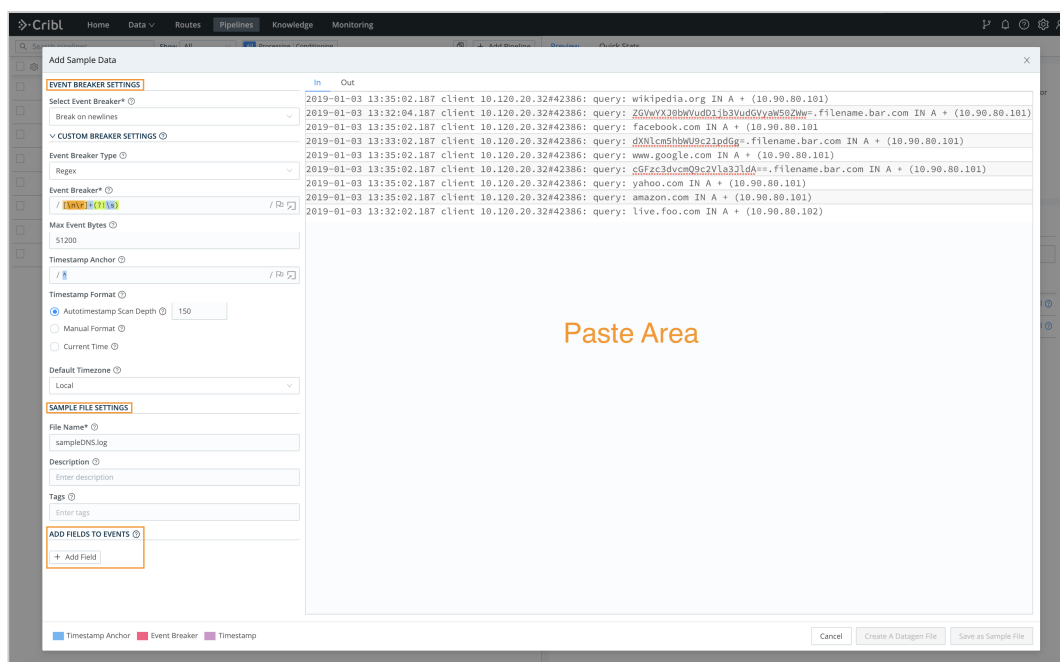


Preview options

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.



Add Sample Data modal

Paste Area

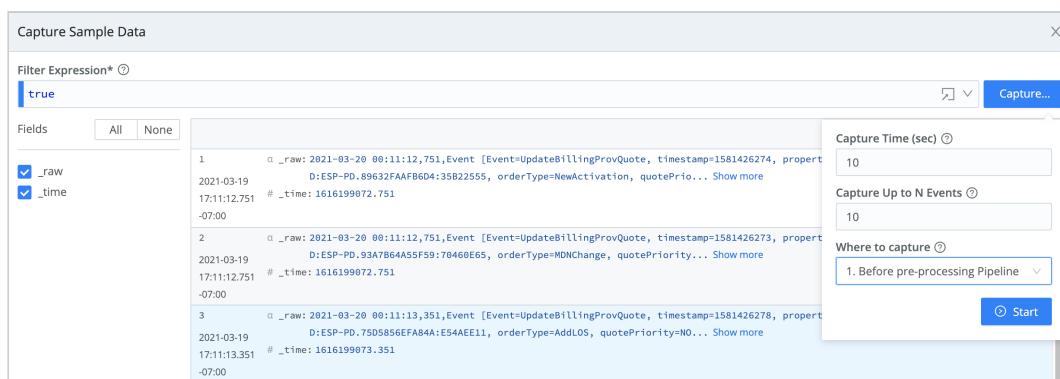
This is where the content of the paste (or uploaded file) is displayed.

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

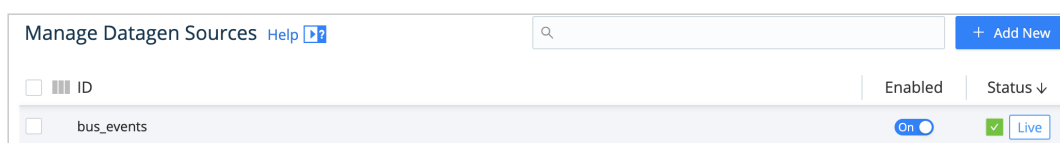
The **Capture New** button opens a slightly different modal – it does not require event breaking. In the composite screenshot below, we've already captured some events using the **Capture** drop-down.



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.



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.



Destination modal > Live Data tab

Controlling Sample Size

To prevent in-memory samples from getting unreasonably large, samples input by any means (Capture/Live Data, Attach;/upload, or Paste) are constrained by a limit set at global **Settings** (lower left) > **General Settings** > **Limits** >

Max sample size. The default limit is 256KB , and you can adjust this upward or downward.

Fields

In the **Capture Sample Data** and **Live Data** modals, 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 that 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:

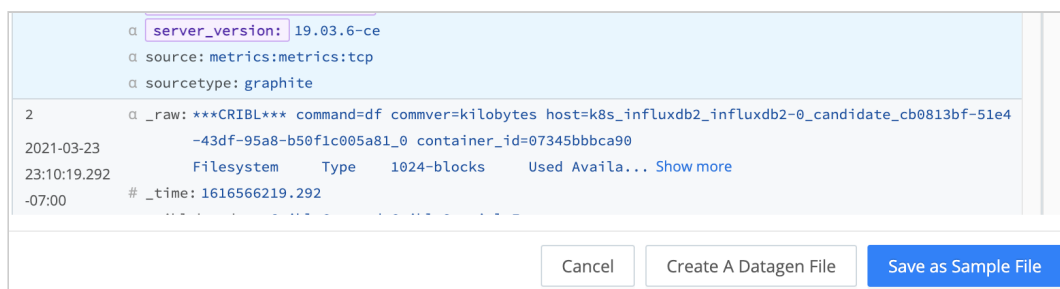
Symbol	Meaning
α	string
#	numeric
b	boolean
m	metric
{ }	JSON object
[]	array

On JSON objects and arrays, you'll also see:

Symbol	Meaning
+	expandable
-	collapsible

Saving Sample Data

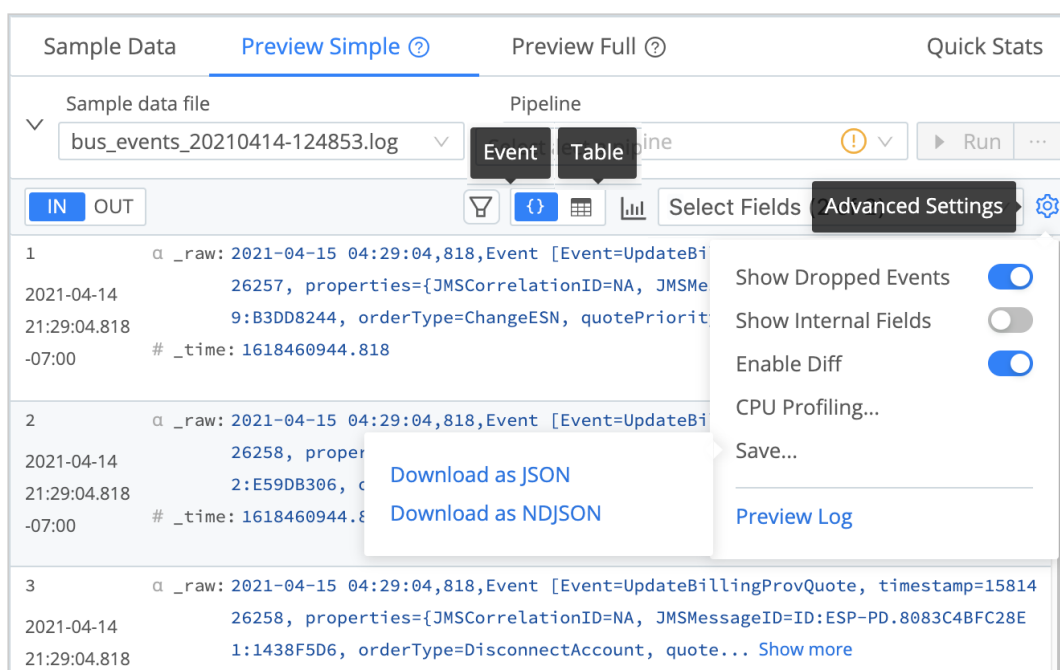
The Preview pane's **Add Sample Data** or **Capture Sample Data** modal, once you've successfully populated it with data, provides options to save the data as a sample and/or datagen file. Click the appropriate button, accept or modify the default/generated file name and other options, and confirm the save.



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.



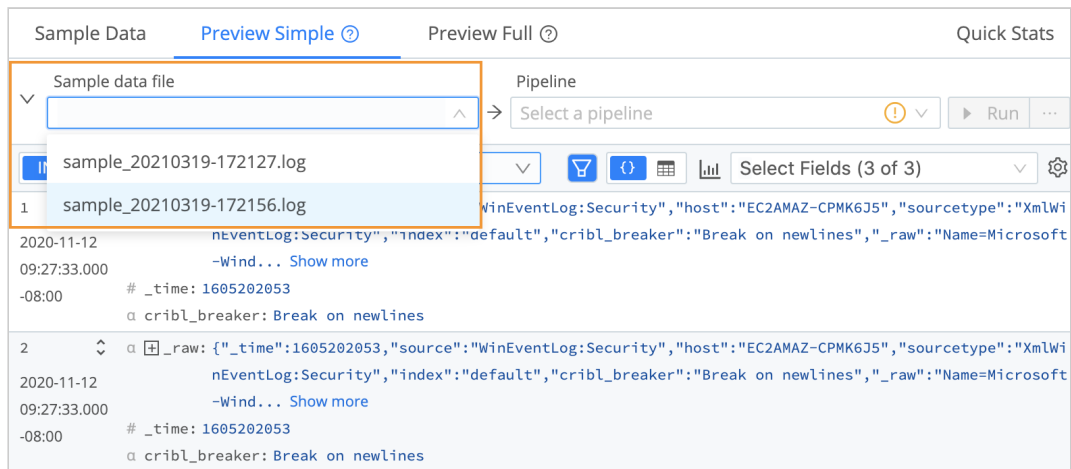
Event, Table, and Advanced options (composite screenshot)

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

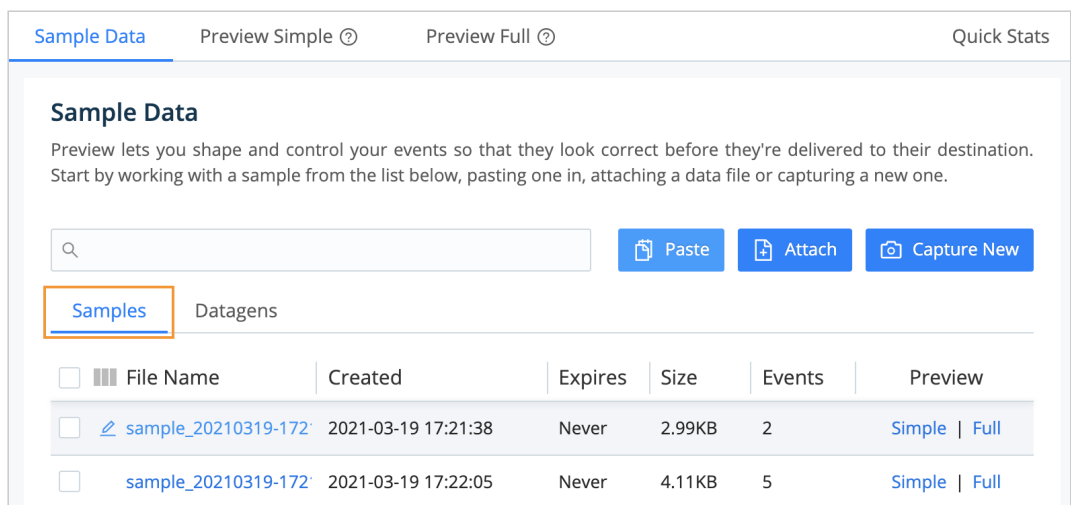
As you add more samples to your system, you can easily access them via the **Sample data file** drop-down.



The screenshot shows the 'Sample Data' tab in a software interface. At the top, there are tabs for 'Sample Data', 'Preview Simple', 'Preview Full', and 'Quick Stats'. The 'Sample Data' tab is active. Below the tabs, there is a 'Sample data file' dropdown menu. The dropdown is open, showing two options: 'sample_20210319-172127.log' and 'sample_20210319-172156.log'. The second option is highlighted. To the right of the dropdown, there is a 'Pipeline' section with a 'Select a pipeline' dropdown and a 'Run' button. Below the pipeline section, there is a 'Select Fields (3 of 3)' dropdown. The main area of the 'Sample Data' tab displays a list of samples. The first sample is 'sample_20210319-172156.log'. It has a status icon, a date '2020-11-12', a time '09:27:33.000', and a size '-08:00'. The sample is expanded, showing a preview of the event log data. The preview shows a JSON-like structure with fields like 'source', 'host', 'sourcetype', and 'index'. The second sample is 'sample_20210319-172127.log', also with a status icon, date '2020-11-12', time '09:27:33.000', and size '-08:00'. It is also expanded, showing a similar preview of event log data.

Selecting an existing sample

You can also manage and modify sample files via the **Samples** tab highlighted below.

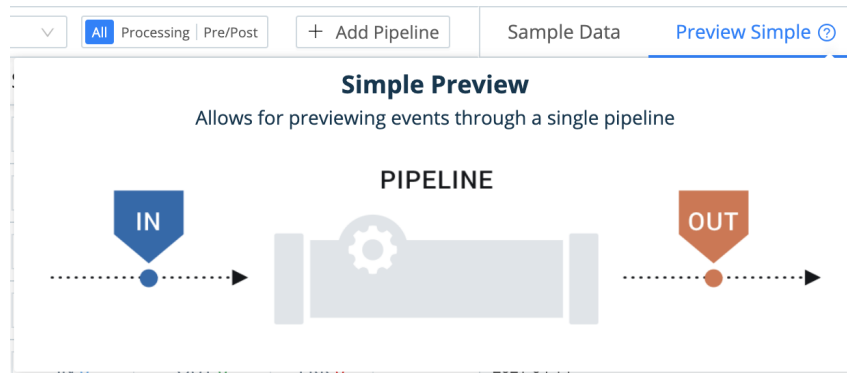


The screenshot shows the 'Samples' tab in a software interface. At the top, there are tabs for 'Sample Data', 'Preview Simple', 'Preview Full', and 'Quick Stats'. The 'Sample Data' tab is active. Below the tabs, there is a 'Sample Data' section with a description: 'Preview lets you shape and control 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.' Below the description, there is a search bar and three buttons: 'Paste', 'Attach', and 'Capture New'. Below the buttons, there is a 'Samples' tab, which is highlighted. Below the 'Samples' tab, there is a table of sample files. The table has columns for 'File Name', 'Created', 'Expires', 'Size', 'Events', and 'Preview'. Two samples are listed: 'sample_20210319-172' and 'sample_20210319-172'. The first sample has a status icon, a date '2021-03-19 17:21:38', a size '2.99KB', and 2 events. The second sample has a status icon, a date '2021-03-19 17:22:05', a size '4.11KB', and 5 events. Both samples have a 'Preview' column with links for 'Simple' and 'Full'.

Managing sample files

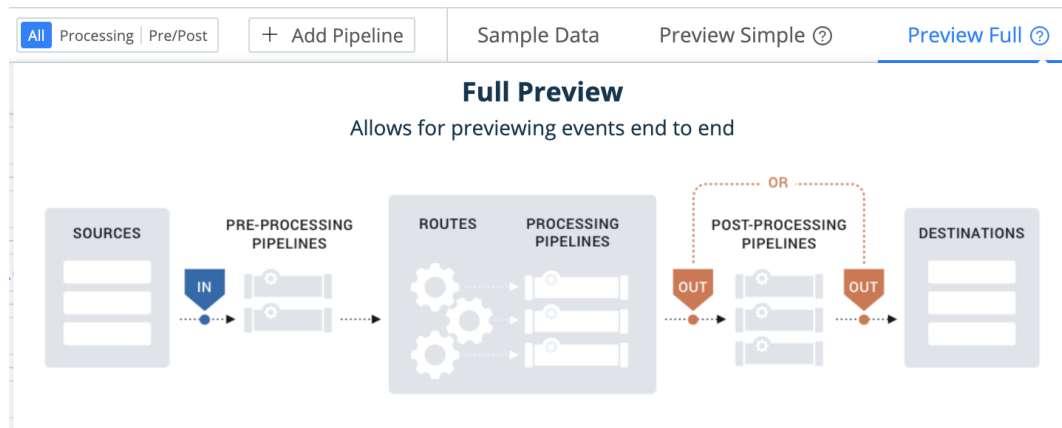
Simple Versus Full Preview

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

Delete Sample

Create Datagen from Sample

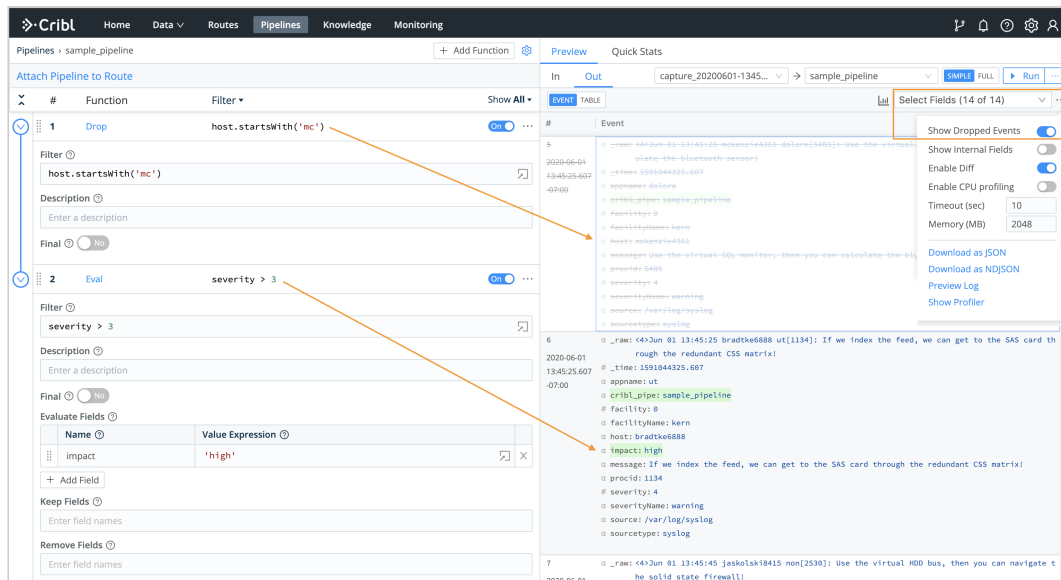
Clone Sample

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:

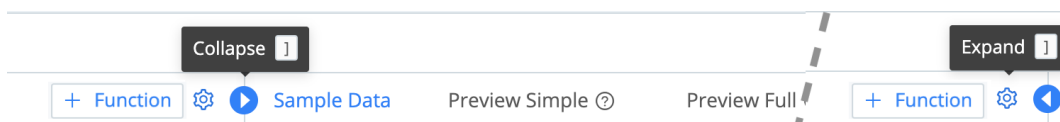
- **Dropped events:** When events are dropped, the **OUT** tab displays them as grayed-out text, with strikethrough. You can control their display using the **Advanced Settings** menu's **Show Dropped Events** slider.
- **Added fields:** When LogStream's processing adds new fields, these fields are highlighted 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.



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: <code>0</code> Keys: keyId: <code>0</code> , keyId: <code>1</code> Users: <code>U0</code> , <code>U1</code>	datasetA Users: <code>U0</code> , <code>U1</code> , <code>U2</code>

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: <code>1</code> Keys: keyId: <code>11</code> , keyId: <code>22</code>	datasetA Users: <code>U0</code> , <code>U1</code> , <code>U2</code>

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](#).

Encryption Keys

Get Key Bundle

+ Add New

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, residing in: `$SPLUNK_HOME/etc/apps/cribl/local/cribl/auth/`.

Modifying Keys

When you update keys by editing the `keys.json` file, you must add them back to the directories above (respectively, on a single instance or on a distributed deployment's Leader Node).

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

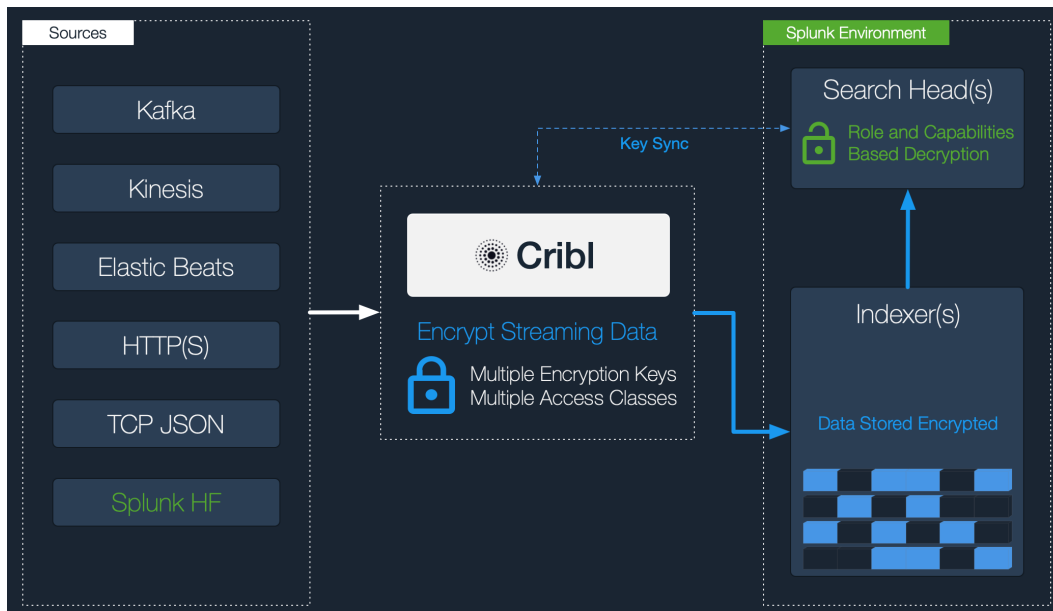
In Splunk, capability names should follow the format `cribl_keyclass_N`, where `N` is the Cribl Key Class. For example, a role with capability `cribl_keyclass_1` has access to all key IDs associated with key class `1`.

Capability Name	Corresponding Cribl Key Class
<code>cribl_keyclass_1</code>	1
<code>cribl_keyclass_2</code>	2

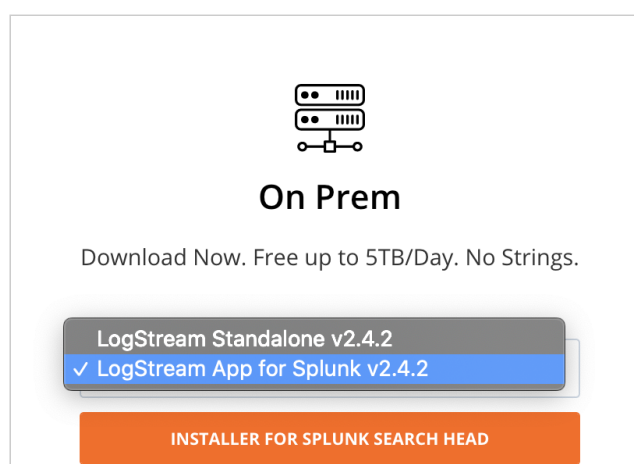
...	...
cribl_keyclass_N	N

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 .`



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: `$CRIBL_HOME/bin/cribl mode-searchhead` . (When installed as a Splunk app, `$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 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
--------------	---------------

AddVariable

Run Delete Script

Settings > Manage Scripts page

The Manage Scripts page provides the following fields:

- **ID:** Unique ID for this script.
- **Command:** Command to execute for this script.
- **Description:** Brief description about this script. Optional.
- **Arguments:** Arguments to pass when executing this script.
- **Env variables:** Extra environment variables to set when executing script.

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 Simple ?

Preview Full ?

Quick Stats

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.

Paste

Attach

Capture New

Samples

Datagens

<input type="checkbox"/>	File Name	Created	E	Size	Eve...	Actions
<input type="checkbox"/>	> weblog.log	2019-11-26 11:39:08	N	42.9...	100	Setup Datagen
<input type="checkbox"/>	> syslog.log	2019-11-26 11:39:08	N	42.5...	100	Setup Datagen
<input type="checkbox"/>	> apache_common.log	2019-11-26 11:39:08	N	22.1...	100	Setup Datagen
<input type="checkbox"/>	> 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**.

The screenshot shows the configuration page for 'Apache Syslog Data Generators'. On the left, there's a sidebar with 'General Settings' selected. The main area has 'Input ID*' set to 'Apache Syslog Data Generators'. Below, the 'Datagens' section contains a table with two entries: 'apache_common.log' with 4 events per second and 'syslog.log' with 8 events per second. At the bottom, there are buttons for 'Delete Source', 'Clone Source', 'Prev', and 'Next'.

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.

The screenshot shows the 'Sources' tab in the Monitoring section. A search filter 'datagen' is applied, showing one source: 'Apache Syslog Data Gen...'. The table displays metrics for Events, Thruput, Total, and Bytes, along with a 'Status' column indicating it is 'Live'.

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

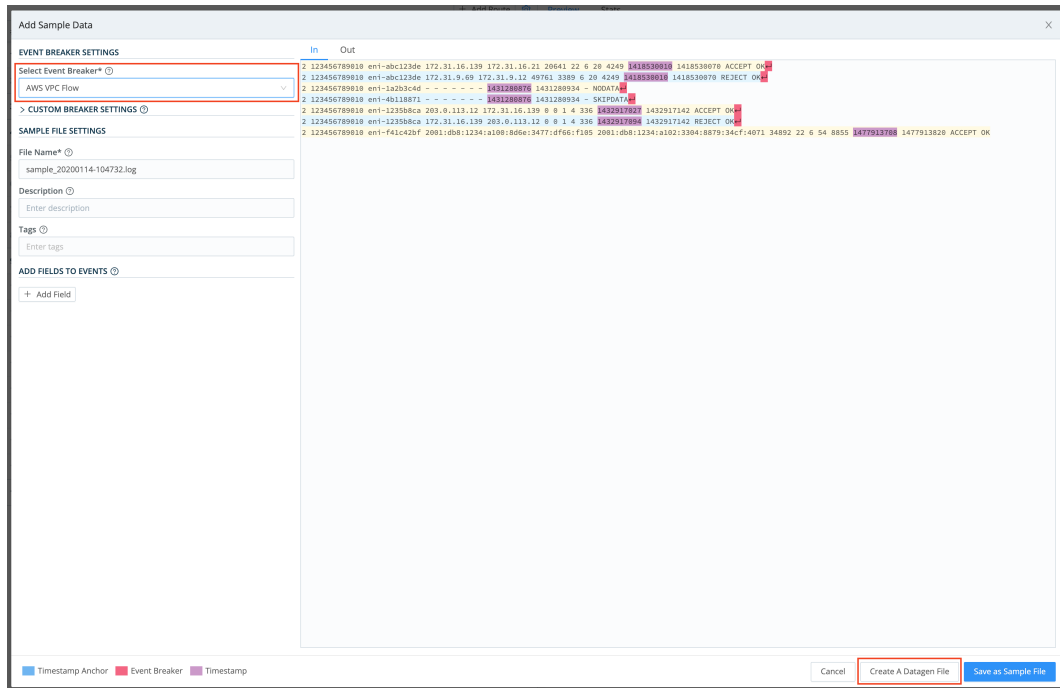
```
2 123456789010 eni-abc123de 172.31.16.139 172.31.16.21 20641 22 6 20 4249
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1
2 123456789010 eni-1a2b3c4d - - - - - 1431280876 1431280934 - NODATA
2 123456789010 eni-4b118871 - - - - - 1431280876 1431280934 - SKIPDATA
2 123456789010 eni-1235b8ca 203.0.113.12 172.31.16.139 0 0 1 4 336 1432917
2 123456789010 eni-1235b8ca 172.31.16.139 203.0.113.12 0 0 1 4 336 1432917
2 123456789010 eni-f41c42bf 2001:db8:1234:a100:8d6e:3477:df66:f105 2001:db
```

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**.



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**.

Create Datagen File

DATAGEN SETTINGS

Template timestamp format*

Time field

Datagen File Name*

CUSTOM TIME SETTINGS

Timestamp Anchor

Default Timezone

Timestamp Format ☒ Autotimestamp Scan Depth ☐ Manual Format ☐ Current Time ☐

#	Event
1	123456789010 ent-abc123de 172.31.16.139 172.31.16.21 20641 22 0 20 4249 \$!timestamp!\$! 1438538078 ACCEPT OK
2	123456789010 ent-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 \$!timestamp!\$! 1438538078 REJECT OK
3	123456789010 ent-1a2b3c4d - - - - - \$!timestamp!\$! 1431280934 - NODATA
4	123456789010 ent-4b118871 - - - - - \$!timestamp!\$! 1431280934 - SKIPDATA
5	123456789010 ent-123508ca 203.0.113.12 172.31.16.139 8 0 1 4 336 \$!timestamp!\$! 1432937342 ACCEPT OK
6	123456789010 ent-123508ca 172.31.16.139 203.0.113.12 8 0 1 4 336 \$!timestamp!\$! 1432937342 REJECT OK
7	123456789010 ent-f41c420f 2061:db8:1234:aa00:b00e:3477:d766:f105 2061:db8:1234:aa02:3304:b879:34c7:4071 34892 22 6 54 8855 \$!timestamp!\$! 1477919828 ACCEPT OK

Cancel **Save as Datagen File**

Saving a named Datagen template

To confirm that the Datagen file has been created, check **Preview > Datagens**.

Sample Data Preview Simple Preview Full Quick Stats

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.

Samples **Datagens**

<input type="checkbox"/>	File Name	Created ↑	Expires	Size	Events	Actions
<input type="checkbox"/>	> vpc-flow-datagen.log	2020-01-14 11:00:26	Never	960.00B	7	Setup Datagen
<input type="checkbox"/>	> weblog.log	2019-11-26 14:39:08	Never	42.40KB	100	Setup Datagen
<input type="checkbox"/>	> syslog.log	2019-11-26 14:39:08	Never	19.97KB	100	Setup Datagen
<input type="checkbox"/>	> apache_common.log	2019-11-26 14:39:08	Never	17.12KB	100	Setup Datagen
<input type="checkbox"/>	> 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.

Manage Datagens Sources

Help

+

Add New

ID

Apache_Syslog_Data_Generators

On

Live

General Settings

Processing Settings

Fields (metadata)

Pre-processing

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	X
syslog.log	8	X
	10	X

+

businessevent.log

palo_alto_traffic.log

snmp-v1.log

snmp-v2c.log

snmp-v3.log

syslog.log

vpc-flow-datagen.log

weblog.log

Delete Source

Clone Source

Prev

Next

Cancel

Save

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>
```

Not all commands have sub-commands.

To see help for any command, append the `--help` option, for example:

```
./cribl vars --help
./cribl vars get --help
./cribl vars get -i myArray --help
```

The `scope` command is an exception: it has no `--help` option, but it has [its own CLI Reference](#) in the AppScope documentation.

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 a list of commands with a description (help) for each. Defaults to a selection of generally useful commands.

Usage

```
./cribl help [-a]
```

Options

`-a` - Display the list of all commands, except for ``scope``.

Sample Response

```
Cribl LogStream - 3.1.0-f765e418
Usage: [sub-command] [options] [args]

Commands:
```

help	- Display help
mode-master	- Configure LogStream as a master instance
mode-single	- Configure LogStream as a single instance
mode-worker	- Configure LogStream as a worker instance
reload	- Reload Cribl LogStream
restart	- Restart Cribl LogStream
boot-start	- Start Cribl LogStream
status	- Status of Cribl LogStream
stop	- Stop Cribl LogStream
version	- Print Cribl LogStream version
auth	- Cribl LogStream Auth
boot-start	- Enable/Disable Cribl LogStream boot-start
diag	- Manage diagnostics bundles
git	- Manage worker groups config
keys	- Manage encryption keys
nc	- Listen on a port for traffic and output stats and da
node	- Execute a JavaScript file
pack	- Manage Cribl Packs
pipe	- Feed stdin to a pipeline
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.

Usage

```
./cribl mode-master <options> <args>
```

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 f
[-c <certPath>]	- Server path containing certificates (in PEM form
[-u <authToken>]	- Optional authentication token to include as part
[-i <ipWhitelistRegex>]	- Regex matching IP addresses that are allowed to

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.

Usage

```
./cribl mode-single [--help]
```

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> <options> <args>
```

The `-H <host> -p <port>` parameters are required.

Options

<code>-H <host></code>	- Leader Node's Hostname or IP address.
<code>-p <port></code>	- Leader Node's cluster communications port (defaults to 8080).
<code>[-n <certName>]</code>	- Name of saved certificate.
<code>[-k <privKeyPath>]</code>	- Server path containing the private key (in PEM format).
<code>[-c <certPath>]</code>	- Server path containing certificates (in PEM format) to use.
<code>[-u <authToken>]</code>	- Authentication token to include as part of the connection.
<code>[-e <envRegex>]</code>	- Regex that selects environment variables to report to Leader.
<code>[-t <tags>]</code>	- Tag values to report to Leader.
<code>[-g <group>]</code>	- Worker Group to report to Leader.

Sample Response

Settings updated.
You will need to restart LogStream before your changes take full effect.

pack

Manages [Cribl Packs](#).

Usage

```
./cribl pack <sub-command> <options> <args>
```

Sub-commands and Options

export	- Export Cribl Packs, args:
-m <mode>	- Mode to export. Accepts: merge_safe, merge, default_
[-o <filename>]	- Where to export the pack on disk.
[-n <name>]	- Name to override the installed pack's name on export
[-g <group>]	- The worker group to execute within
install	- Install a Cribl Pack, args:
[-d]	- Run install in debug.
[-f]	- Force install.
[-n <name>]	- Name of the pack to install; defaults to source.
[-g <group>]	- The worker group to execute within.
list	- List Cribl Packs, args:
[-v]	- Display all pack info.
[-g <group>]	- The worker group to execute within.
uninstall	- Uninstall a Cribl Pack, args:
[-d]	- Run uninstall in debug.
[-g <group>]	- The worker group to execute within.
upgrade	- Upgrade a Cribl Pack, args:
[-d]	- Run upgrade in debug.
[-s <source>]	- Provide the pack source.
[-m <minor>]	- Only upgrade to minor version.
[-g <group>]	- The worker group to execute within.

Sample Response

id	version	spec	displayName	author	description

HelloPacks	1.0.0	----	Hello, Packs!	Cribl, Inc.	A sample pack with

reload

Reloads Cribl LogStream. Executes immediately.

Usage

```
./cribl reload [--help]
```

Sample Response

Reload request submitted to Cribl LogStream

restart

Restarts Cribl LogStream. Executes immediately.

 Executing this command cancels any running [collection jobs](#).

Usage

```
./cribl restart [--help]
```

Sample Response

```
Stopping Cribl LogStream, process 18
.....
Cribl LogStream is not running
Starting Cribl LogStream...
...
Cribl LogStream started
```

start

Starts Cribl LogStream. Executes immediately. Upon first run, echoes LogStream's default login credentials.

Usage

```
./cribl start <options> <args>
```

Options

```
[-d <dir>] - Configuration directory
[-r <role>] - Process role
```

Sample Response

```
Starting Cribl LogStream...
...
Cribl LogStream started
```

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.

Usage

```
./cribl status [--help]
```

Sample Response

```
Cribl LogStream Status

Address: http://172.17.0.3:9000
Mode: master
Status: Up
Software Version: 3.1.0-f765e418
Config Version: 347079c
Master: 0.0.0.0:4200
PID: 4100
GUID: e706052a-ace9-4511-a7c7-b58a414a07d3
```

stop

Stops Cribl LogStream. Executes immediately.

 Executing this command cancels any running [collection jobs](#).

Usage

```
./cribl stop [--help]
```

Sample Response

```
Stopping Cribl LogStream, process 3951
.....
```

Cribl LogStream is not running

version

Displays Cribl LogStream version. Executes immediately.

Usage

```
./cribl version [--help]
```

Sample Response

```
Software Version: 3.1.0-f765e418
```

auth

Log into or out of Cribl LogStream.

Usage

```
./cribl auth <sub-command> <options> <args>
```

Sub-commands and Options

login	- Login to Cribl LogStream, args:
[-h <oldHost>]	- undefined
[-H <host>]	- Host URL (e.g. http://localhost:9000)
[-u <username>]	- Username
[-p <password>]	- Password
[-f <file>]	- File with credentials
logout	- Logout 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

```
./cribl boot-start <sub-command> <options> <args>
```

Sub-commands and Options

<code>disable</code>	- Disable Cribl LogStream boot-start, args:
<code>[-m <manager>]</code>	- Init manager (systemd initd)
<code>[-c <configDir>]</code>	- Config directory for the init manager
<code>enable</code>	- Enable Cribl LogStream boot-start, args:
<code>[-m <manager>]</code>	- Init manager (systemd initd)
<code>[-u <user>]</code>	- User to run Cribl LogStream as
<code>[-c <configDir>]</code>	- Config directory for the init manager

Sample Response

```
Enabling Cribl LogStream to be managed by initd...
boot-start enable command needs root privileges...
Enabled Cribl LogStream to be managed by initd as user=root.
```

diag

Manages diagnostic bundles.

Usage

```
./cribl pack <sub-command> <options> <args>
```

Sub-commands and Options

create	- Creates diagnostic bundle for Cribl LogStream, args:
[-d]	- Run create in debug mode
[-j]	- Do not append '.txt' to js files
list	- List existing Cribl LogStream diagnostic bundles
send	- Send LogStream diagnostics bundle to Cribl Support,
-c <caseNumber>	- Cribl Support Case Number
[-p <path>]	- Diagnostic bundle path (if empty then new bundle wil

Sample Response

```
Created Cribl LogStream diagnostic bundle at /opt/cribl/diag/logstream-zed
```

git

Manages Worker Groups configuration.

Usage

```
./cribl pack <sub-command> <options> <args>
```

Sub-commands and Options

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-groups	- List worker groups.

Sample Response

Successfully committed version 7c04de1

groups

Deprecated. See [git](#) .

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

`./cribl keys <sub-command> <options> <args> -g <group>`

Sub-commands and Options

<code>add</code>	- Add encryption keys, args:
<code>[-c <keyclass>]</code>	- key class to set for the key
<code>[-k <kms>]</code>	- KMS to use, must be configured, see <code>cribl.yml</code>
<code>[-e <expires>]</code>	- expiration time, epoch time
<code>[-i]</code>	- use an initialization vector
<code>-g <group></code>	- Group ID
<code>list</code>	- List encryption keys, args:
<code>-g <group></code>	- Group ID

Sample Response

Adding key succeeded. Key count=1

nc

Listens on a port for traffic, and outputs stats and data. (Netcat-like utility.)

Usage

`./cribl nc -p <port> <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 <throttle>]	- throttle rate in (unit)/sec, where units can be KB,

Sample Response

```
2021-08-20T22:44:30.457Z - starting server on 0.0.0.0:9999
2021-08-20T22:44:30.462Z - server listening 0.0.0.0:9999
2021-08-20T22:44:31.461Z - messages: 0, socks: 0, thruput: 0MBps
2021-08-20T22:44:32.466Z - messages: 0, socks: 0, thruput: 0MBps
...
2021-08-20T22:44:39.212Z - got connection: 127.0.0.1:37190
2021-08-20T22:44:39.213Z - got connection: 127.0.0.1:37192
```

node

Run with no options, displays a command prompt, as shown here:

```
>
```

To execute a JavaScript file, you can enter path/filename at the prompt.

With the `-v` option, prints the version of NodeJS that is running.

With `-e` , evaluates a string. Write to console to see the output, for example:

```
./cribl node -e 'console.log(Date.now())'
1629740667695
```

Usage

```
./cribl node <options> <args>
```

Options

[-e <eval>]	- String to eval
[-v]	- Prints NodeJS version

Sample Response

v14.15.1

pipe

Feeds stdin to a pipeline.

Usage

```
./cribl pipe -p <pipelineName> <options> <args>
```

Examples:

```
cat sample.log | ./cribl pipe -p <pipelineName>
cat sample.log | ./cribl pipe -p <pipelineName> 2>/dev/null
```

Options

- p - Pipeline to feed data thru
- [-d] - Include dropped events
- [-c] - Perform CPU profiling
- [-a] - Optional Cribl Pack context

Sample Response

```
...
{"time":"2021-08-20T20:37:00.017Z","cid":"api","channel":"commands","level":1}
{"time":"2021-08-20T20:37:00.019Z","cid":"api","channel":"pipe:main","level":1}
{"time":"2021-08-20T20:37:00.021Z","cid":"api","channel":"pipe:main","level":1}
{"time":"2021-08-20T20:37:00.022Z","cid":"api","channel":"commands","level":1}
{"time":"2021-08-20T20:37:00.028Z","cid":"api","channel":"GrokMgr","level":1}
...
```

scope

Greps your apps by the syscalls. Executes immediately.

See the [AppScope CLI Reference](#) for usage and examples.

vars

Manages LogStream [Global Variables](#).

Usage

```
./cribl vars <sub-command> <options> <args>
```

Sub-commands and Options

Sub-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 global variables, 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

Sample Response

```
[
  {
    "type": "number",
    "lib": "cribl",
    "description": "Sample number variable ",
    "value": "42",
    "tags": "cribl,sample",
    "id": "theAnswer"
  }
]
```

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
<pre>x = 42 newFoo = foo.slice(30)</pre>	<pre>(Math.random() * 42) 3 + 4 'foobar' '42'</pre>

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.Crypto.Hmac

(method) `Crypto.Hmac(value: string | Buffer, secret: string, algorithm: string = 'sha256', outputFormat: 'base64' | 'hex' | 'latin1' = 'hex')`: string

Generates an [HMAC](#) that can be added to events, or can be used to validate events that contain an HMAC. (Available in LogStream v.3.1.2+.)

@param – `value` – The data to encrypt, as a string. (When the `outputFormat` is invalid or undefined, this parameter is returned as the digest, via a Buffer.)

@param – `secret` – The secret key used to generate the MAC, as a string.

@param – `algorithm` – The hash algorithm used to generate the MAC, as a string. Defaults to 'sha256' .Run `openssl list -digest-algorithms` to see the list of available algorithms.

@param – `outputFormat` – One of 'base64' , 'hex' ,or 'latin1' . Defaults to 'hex' .

@returns – The calculated HMAC digest on success; 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 Luhn'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 Luhn'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 Luhn'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 Luhn'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 Luhn'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 Luhn 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.isIPv4AllInterfaces()`

(method) `Net.isIPv4AllInterfaces(value: string): boolean`

Determine if the value supplied is the IPv4 all-interfaces address, typically used to bind to all IPv4 interfaces – i.e., `'0.0.0.0'`. (Available in LogStream v.3.1.2+.)

@param – `value` – the IP address to test.

`C.Net.isIPv6AllInterfaces()`

(method) `Net.isIPv6AllInterfaces(value: string): boolean`

Determine if the value supplied is an IPv6 all-interfaces address, typically used to bind to all IPv6 interfaces – one of: `'::'`, `'0:0:0:0:0:0:0:0'`, or `'0000:0000:0000:0000:0000:0000:0000:0000'`. (Available in LogStream v.3.1.2+.)

@param – `value` – the IP address to test.

`C.Net.ipv6Normalize()`

(method) `Net.ipv6Normalize(address: string): string`

Normalize an IPV6 address based on [RFC draft-ietf-6man-text-address-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.Secret – Secrets-Management Functions

`C.Secret()`

(method) `Secret(id: string, type: 'keypair') => IPairSecret`

(method) `Secret(id: string, type: 'text') => ITextSecret`

(method) `Secret(id: string, type: 'credentials') => ICredentialsSecret`

(method) `Secret: (id: string, type?: string): ISecret`

Returns a secret matching a specified ID.

@param `id` – ID of the secret.

@param `type` – optional type of the secret.

@returns the specified secret.

Example: `C.Secret('victorias', 'text')` will return a text secret with ID 'victorias' (or undefined if no such secret exists).

See [Securing > Secrets](#) 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 – Optionally, override the default

\$CRIBL_HOME/data/lookups/model_relative_entropy_top_domains.csv
model used to test the input. Create a custom lookup file with the same column and value structure as the default, and store it in the same path, as model_relative_entropy_<custom-name>.csv . To reference it, pass your <custom-name> substring as the modelName parameter.

@returns – The relative entropy value, or -1 in case of an error.

□ When using modelName in a [distributed deployment](#), the corresponding paths are \$CRIBL_HOME/groups/<worker-group-name>/data/lookups/. Creating your custom lookup file [via LogStream's UI](#) will automatically set the appropriate paths.

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.strptime()`

(method) `Time.strptime(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

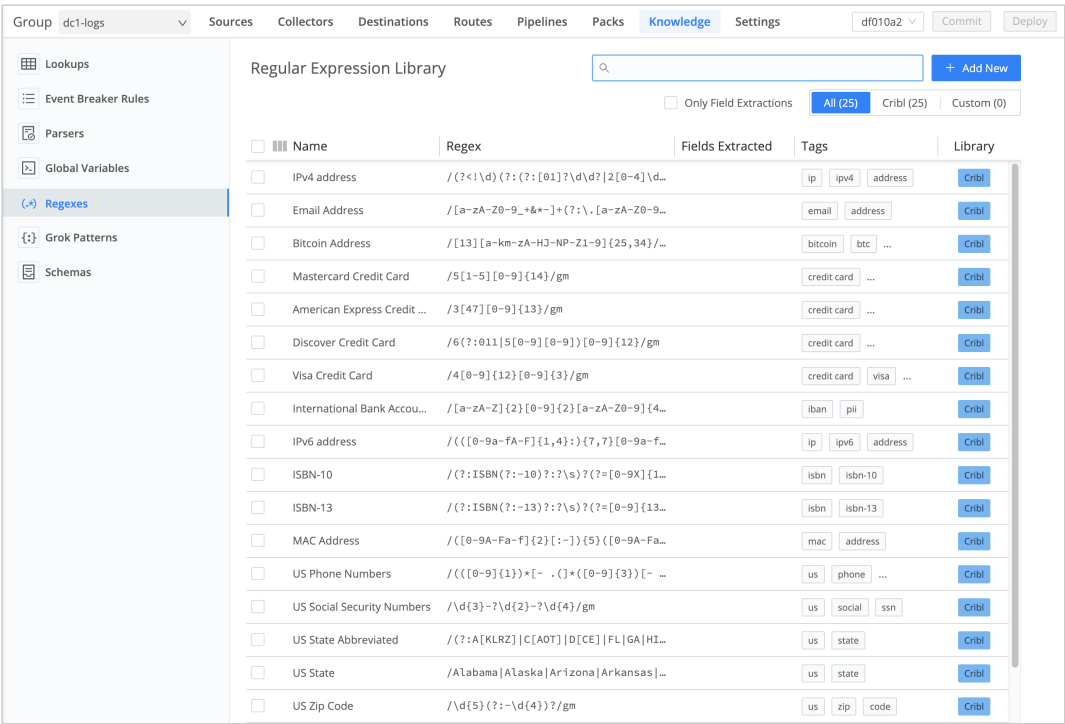
Cribl LogStream Version.

KNOWLEDGE

Regex Library

What Is the Regex Library

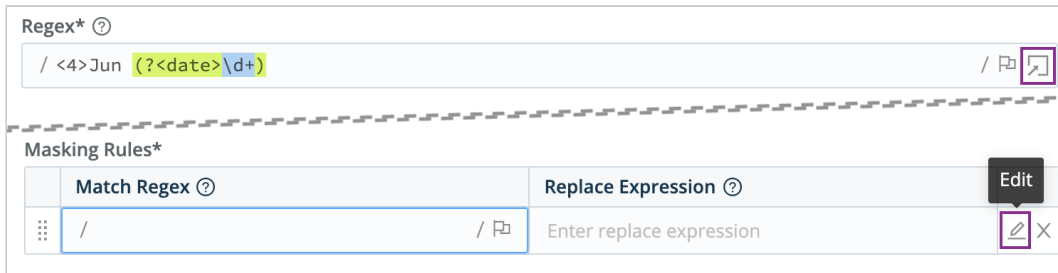
Cribl LogStream ships with a Regex Library that contains a set of pre-built common regex patterns. This library serves as an easily accessible repository of regular expressions. The Library is searchable, and you can assign tags to each pattern for further organization or categorization. The Library is located under **Knowledge > Regex Library**.



Regular Expression Library

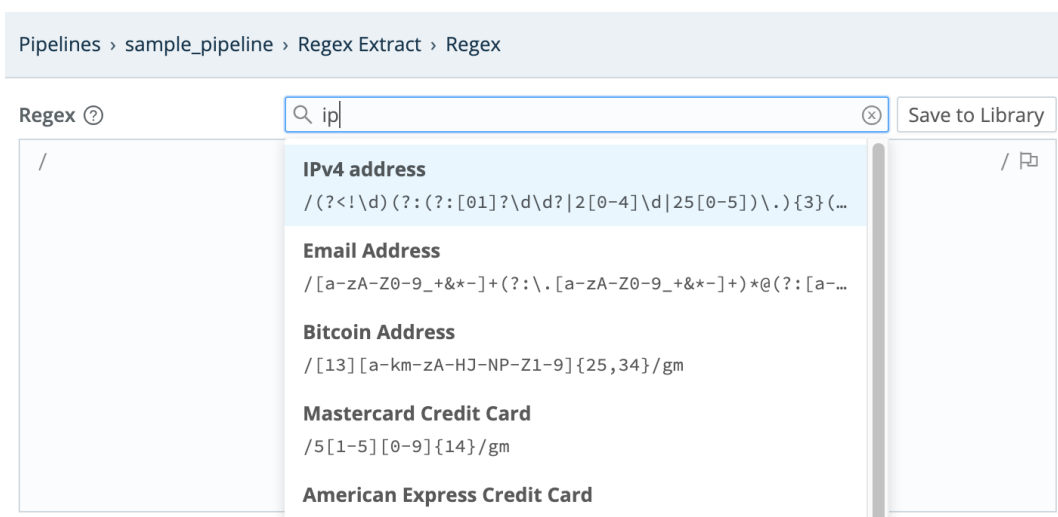
Using Library Patterns

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

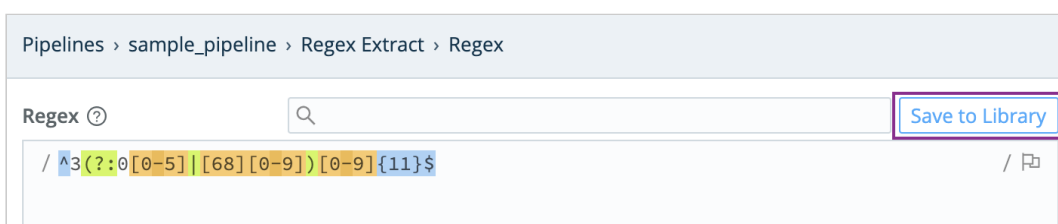
In the resulting Regex or Rules modal, Regex Library patterns will appear as typeahead options. Click a pattern to paste it in. You can then use the pattern as-is, or modify it as necessary.



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.



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**.

Groups > dc1-logs > Knowledge > Grok Patterns > New Grok pattern file

Filename*

Content*

1

- 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. To see how Event Breakers interact with the rest of LogStream's data flow, see [Event Processing Order](#).

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.

Group	dcl-metrics	Sources	Destinations	Routes	Pipelines	Packs	Knowledge	Settings	c508d4d	Commit	Deploy
Lookups	Event Breaker Rulesets										
Event Breaker Rules	<input type="text"/>										
Parsers	All (9) Cribl (8) Custom (1)										
Global Variables											
Regexes											
Grok Patterns											
Schemas											
	<input type="checkbox"/>	ID	Description	Tags	Library						
	<input type="checkbox"/>	AWS Ruleset	Event breaking rules for common AWS data sour...	flowlogs elb alb ...	Cribl						
	<input type="checkbox"/>	Apache Ruleset	Event breaking rules for Apache Common and C...	apache common ...	Cribl						
	<input type="checkbox"/>	Cisco Ruleset	Event breaking rules for common Cisco data sou...	cisco asa estreamer	Cribl						
	<input type="checkbox"/>	Palo Alto Ruleset	Event breaking rules for common Palo Alto data ...	palo traffic threat ...	Cribl						
	<input type="checkbox"/>	Bro Ruleset	Event breaking rules for Bro logs	bro	Cribl						
	<input type="checkbox"/>	Cribl	Event breaking rules for new line delimited json ...	cribl ndjson	Cribl						
	<input type="checkbox"/>	Office 365			Cribl						
	<input type="checkbox"/>	Cribl - Do Not Break Ruleset	Special rule to NOT break streams into events (Ca...		Cribl						
	<input type="checkbox"/>	Cribl Command	Break up events from shellmetrics		Custom						

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.

Groups > dc1-metrics > Knowledge > Event Breaker Rules > AWS Ruleset

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	Earliest timestamp allowed	Future timestamp allowed	Max Event Bytes	Fields	Enabled	Actions
1	AWS CloudTrail...	/CloudTrail...	JSON Array		Format: %Y-%m-%...	utc			51200		Yes	
2	AWS VPC Flow Logs...	/^\d+\s+\d+\										

Rules within an example (AWS) ruleset that ships with LogStream

- 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:

Groups > dc1-metrics > Sources > TCP > in_tcp

Configure Status Charts Live Data Logs Connected Destinations Help

General Settings

TLS Settings (Server Side)

Processing Settings

Custom Command

Event Breakers

Fields (Metadata)

Pre-Processing

Event Breaker rulesets

- 1 **AWS Ruleset** Event breaking rules for common AWS data sources (5 rules)
- 2 **Cisco Ruleset** Event breaking rules for common Cisco data source (3 rules)
- 3 **Palo Alto Ruleset** Event breaking rules for common Palo Alto data source (4 rules)

System Default Rule Filter Condition: true Event Breaker: /\n\r]+(?!\s)/ Timestamp Anchor: ^/ Timestamp Format: Auto:150 Default Timezone: Local Max Event Bytes: 51200

+ Add ruleset

Event Breaker buffer timeout

10000

Three Event Breaker rulesets added to a Source

Rule Example

This rule breaks on newlines and uses Manual timestamping **after** the sixth comma, as indicated by this pattern: `^(?:[^\s,]*,){6}`.

Event Breaker Rule

Rule Name* Palo Alto Traffic

Filter Condition* /\n\r]+(?!\s)/

Event Breaker* /\n\r]+(?!\s)/

Timestamp Anchor* ^/

Timestamp Format* Manual Format

Autotimestamp Scan Depth* 150

Manual Format* %Y/%m/%d %H:%M:%S

Current Time

Default Timezone* Local

Max Event Bytes* 51200

ADD FIELDS TO EVENTS

+ Add Field

Timestamp Anchor Event Breaker Timestamp

Cancel OK

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.

The screenshot shows the 'Add new Event Breaker rule' modal in Cribl. The modal is titled 'Groups > default > Knowledge > Event Breaker Rules > Cribl > Rules'. It has a 'Rule Name' field, a 'Filter Condition' dropdown set to 'true', and an 'EVENT BREAKER SETTINGS' section. This section includes an 'Enabled' toggle (checked), an 'Event Breaker Type' dropdown (set to 'Regex'), an 'Event Breaker' field with a regex pattern, a 'Max Event Bytes' field (set to 51200), and a 'TIMESTAMP SETTINGS' section. The timestamp settings include a 'Timestamp Anchor' dropdown (set to 's'), a 'Timestamp Format' dropdown (set to 'Autotimestamp Scan Depth'), an 'Autotimestamp Scan Depth' field (set to 150), and a 'Manual Format' field. There is also a 'Default Timezone' dropdown (set to 'Local') and two fields for 'Earliest timestamp allowed' (-420weeks) and 'Future timestamp allowed' (+1week). At the bottom, there is an 'ADD FIELDS TO EVENTS' section with a '+ Add Field' button. The main area of the modal is a large text box with the placeholder text 'Paste your events here or upload a sample file'. There is an 'Upload Sample File' button in the top right corner. At the bottom right, there are 'Cancel' and 'OK' buttons.

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

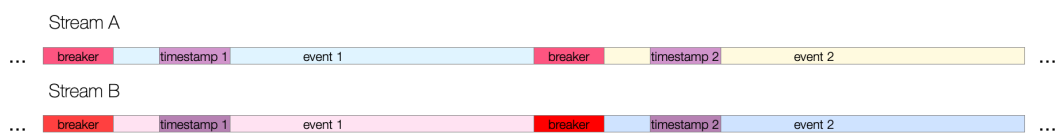
After a breaker pattern has been selected, it will apply on the stream **continuously**. See below for specific information on different [Event Breaker Types](#).

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 be 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.

⚠ The highest **Max Event Bytes** value that you can set is 128 MB (134217728).

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 yo
  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 yo
  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 fe
  "_time": 1589920332
}

--- output event 2 ---
{
  "_raw": "2020-05-19 16:32:12 moen3628 ipsum[5213]: Use the mobile TCP fe
  "_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
#types  time    string  addr    port    addr    port    enum
1331904608.080000 -      192.168.204.59 137      192.168.204.255 137
1331904609.190000 -      192.168.202.83 48516    192.168.207.4   53

--- output event 1 ---
{
  "_raw": "1331904608.080000      -      192.168.204.59 137      192.168.2",
  "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    192.168.2",
  "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","eve
{"eventVersion":"1.05","eventTime":"2020-04-08T01:35:56Z","eventSource":"e

--- output event 1 ---
{
  "_raw": "{\"eventVersion\":\"1.05\",\"eventTime\":\"2020-04-08T01:35:55Z
  "_time": 1586309755,
  "cribl_breaker": "j-array"
}

--- output event 2 ---
{
  "_raw": "{\"eventVersion\":\"1.05\",\"eventTime\":\"2020-04-08T01:35:56Z
  "_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","lev
{"time":"2020-05-25T18:00:54.246Z","cid":"w0","channel":"clustercomm","lev

--- output event 1 ---
{
  "_raw": "{\"time\":\"2020-05-25T18:00:54.201Z\",\"cid\":\"w1\",\"channel
"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
"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"}
{"level":"debug","ts":"2021-02-02T10:38:56.365Z","caller":"sdk/sync.go:42"}

--- output event 1 ---
{
  "_raw": "{\"level\":\"debug\",\"ts\":\"2021-02-02T10:38:46.365Z\",\"call
  \"_time\": 1612262326.365
}

--- output event 2 ---
{
  "_raw": "{\"level\":\"debug\",\"ts\":\"2021-02-02T10:38:56.365Z\",\"call
  \"_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\", \"
  \"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\\\", \"3",
    "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

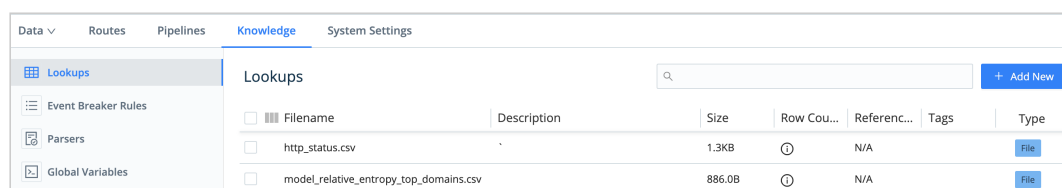
Event Breaker rulesets shipped by Cribl will be listed under the **Cribl** tag, while user-built rulesets will be listed under **Custom**. Over time, Cribl will ship more patterns, so this distinction allows for both sets to grow independently. In the case of an ID/Name conflict, the Custom pattern takes 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 are processed by the [Lookup Function](#). You can access the Lookups library, which provides a management interface for all lookups, under **Knowledge > 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 (`.mddb`) 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	1	N/A		File
model_relative_entropy_top_domains.csv		886.0B	1	N/A		File

Lookups Library

How Does the Library Work

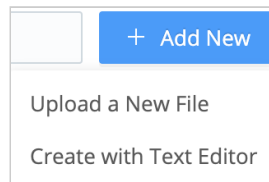
In single-instance deployments, all files handled by the interface are stored in `$CRIBL_HOME/data/lookups` . In [distributed deployments](#), the storage path on the Leader Node is `$CRIBL_HOME/groups/<groupname>/data/lookups/` for each Worker Group.

- 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.

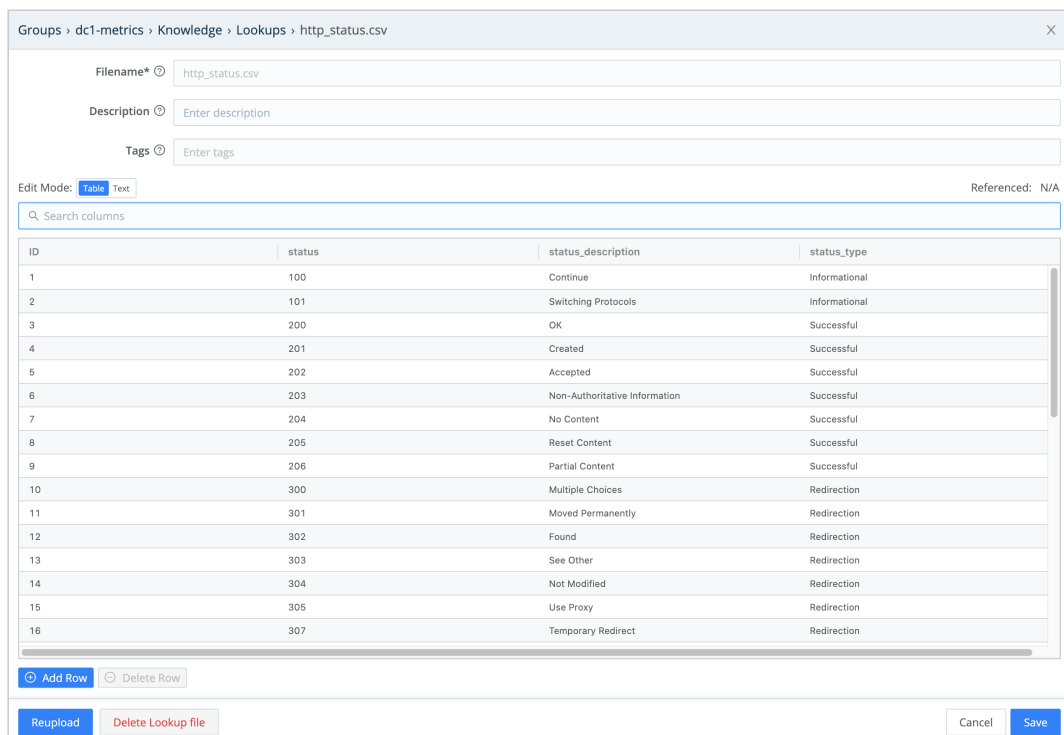


Adding a lookup file

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.




Editing in table mode

Groups › dc1-metrics › Knowledge › Lookups › http_status.csv X

Filename*  http_status.csv

Description  Enter description

Tags  Enter tags

Edit Mode: Table **Text** Referenced: N/A

Note: values in header and data rows should be comma delimited.

1	status,status_description,status_type
2	100,Continue,Informational
3	101,Switching Protocols,Informational
4	200,OK,Successful
5	201,Created,Successful
6	202,Accepted,Successful
7	203,Non-Authoritative Information,Successful
8	204,No Content,Successful
9	205,Reset Content,Successful
10	206,Partial Content,Successful
11	300,Multiple Choices,Redirection
12	301,Moved Permanently,Redirection
13	302,Found,Redirection
14	303,See Other,Redirection
15	304,Not Modified,Redirection
16	305,Use Proxy,Redirection
17	307,Temporary Redirect,Redirection
18	400,Bad Request,Client Error
19	401,Unauthorized,Client Error
20	402,Payment Required,Client Error
21	403,Forbidden,Client Error
22	404,Not Found,Client Error
23	405,Method Not Allowed,Client Error
24	406,Not Acceptable,Client Error
25	407,Proxy Authentication Required,Client Error
26	408,Request Timeout,Client Error
27	409,Conflict,Client Error
28	410,Gone,Client Error
29	411,Length Required,Client Error
30	412,Precondition Failed,Client Error
31	413,Request Entity Too Large,Client Error

Reupload

Delete Lookup file

Cancel

Save

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:

$$\text{Lookup file's uncompressed size (MB)} * 2.25 \text{ (to } 2.75) * \text{numWorkerProcesses} = \text{Extra RAM required for lookup}$$

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.

Maximum Table Size

Aside from the memory requirements above, the Node.js runtime imposes a hard limit on the size of lookup tables that LogStream can handle. No table can contain more than 16,777,216 (i.e., 2^{24}) rows. If a lookup exceeds this size, attempting to load the lookup will log errors of the form: `failed to load function...Value undefined out of range...`

Other Scenarios

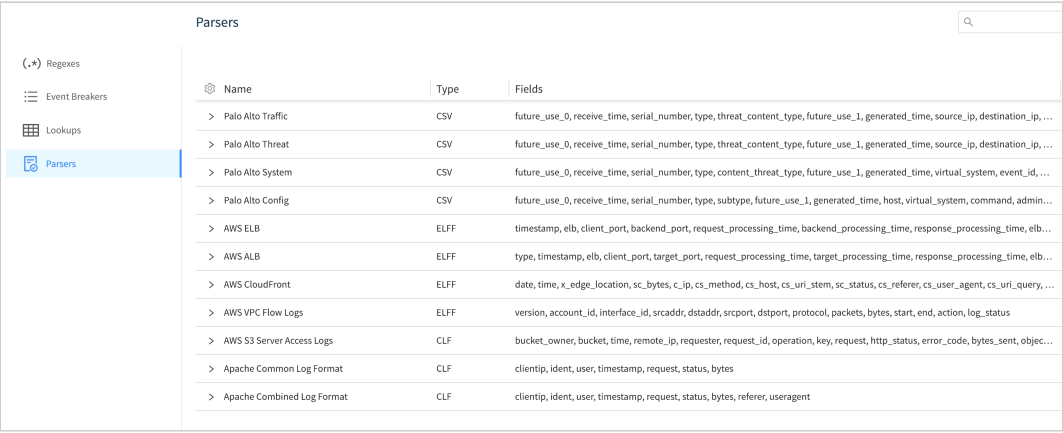
See also:

- [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.



The screenshot shows the 'Parsers' library interface. On the left is a sidebar with navigation options: Regexpes, Event Breakers, Lookups, and Parsers (which is selected). The main area displays a table of parsers. The table has three columns: Name, Type, and Fields. The parsers listed include Palo Alto Traffic, Palo Alto Threat, Palo Alto System, Palo Alto Config, AWS ELB, AWS ALB, AWS CloudFront, AWS VPC Flow Logs, AWS S3 Server Access Logs, Apache Common Log Format, and Apache Combined Log Format.

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**.

☐ New Parser

Custom

Id*

Enter id

Description ⓘ

Enter description

Parser Type* ⓘ

Select one

List of Fields.
Required.* ⓘ

Field names

Tags ⓘ

Enter tags

Advanced Mode

Cancel

Save

Adding a new parser

Schema Library

What Are Schemas

Schemas are JSON definitions that are used to validate JSON events. They're based on the popular [JSON Schema standard](#), and LogStream supports schemas matching that standard's [Drafts 0 through 7](#).

You can find the schema library under **Knowledge > Schemas**. Its purpose is to provide an interface for creating, editing, and maintaining schemas.

You validate a schema using this built-in method:

```
C.Schema('<schema_name>').validate(<object_field>).
```

You can call this method anywhere in LogStream that supports JavaScript expressions. 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",
    "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.

#	Function	Filter	Show All	TABLE	Select Fields (8 of 8)
1	Eval	true	ON	Event	
	Filter	true			
	Description	Enter a description			
	Final	NO			
	Evaluate Fields				
	Name	Value Expression			
	isValid	C.Schema('schenal').validate(employee)			
	+ Add Field				
	Keep Fields	Enter field names			
	Remove Fields	Enter field names			
1				#_ctm=1578661447.911	
2020-01-09				cribl_breaker: Break on newlines	
15:24:07.911				cribl_pipe: myPipeline	
-05:00				employee:	
				age: 21	
				firstName: John	
				lastName: Doe	
				isValid: true	
2				#_ctm=1578661447.911	
2020-01-09				cribl_breaker: Break on newlines	
15:24:07.911				cribl_pipe: myPipeline	
-05:00				employee:	
				age: 43	
				firstName: John	
				lastName: Doe	
				isValid: false	
3				#_ctm=1578661447.911	
2020-01-09				cribl_breaker: Break on newlines	
15:24:07.911				cribl_pipe: myPipeline	
-05:00				employee:	
				firstName: John	
				lastName: Doe	
				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:



The screenshot shows the configuration for an Eval function in Splunk. At the top, it says "3 Eval" and "sourcetype=='access_combined'" with an "On" toggle. Below this, there are several sections: "Filter" with a text input containing "sourcetype=='access_combined'", "Description" with a text input containing "Add 'dc' index-time field to events", "Final" with a "No" radio button, "Evaluate Fields" with a "+ Add Field" button, "Keep Fields" with a text input containing "Enter field names", and "Remove Fields" with a text input containing "Enter field names".

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

+ Add Field

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

+ Add Field

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.

The screenshot shows the Splunk configuration interface for an Eval function. The top bar indicates the function is named 'Eval' and is currently 'On'. The search query is `sourcetype=='access_combined' && host.startsWith...`. The 'Filter' section contains the same query. The 'Description' field is empty. The 'Final' toggle is set to 'No'. The 'Evaluate Fields' table has one entry: 'dc' with the value expression `'nyc-42'`. The 'Keep Fields' section is empty. The 'Remove Fields' section contains the text `date_*`. At the bottom right are 'Cancel', 'Save', and a menu icon.

Name	Value Expression
dc	'nyc-42'

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.

i 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 (b
allowed,S3,"Connection established and close attempt by responder seen (bu
allowed,RST0,"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 SY
dropped,RSTRH,"Responder sent a SYN ACK followed by a RST, we never saw a
dropped,SH,"Originator sent a SYN followed by a FIN, we never saw a SYN AC
dropped,SHR,"Responder sent a SYN ACK followed by a FIN, we never saw a SY
allowed,OTH,"No SYN seen, just midstream traffic (a 'partial connection' t

```

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.cs
```

v

(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	X

+ Add field(s)

Output field(s) ?

Output Field Name from Lookup ?	Lookup Field Name in Event ?	Default Value ?	
action	action	Enter default value	X

+ Add field(s)

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	X
	priority	priority	X

+ Add field(s)

Output field(s)

	Output Field Name from Lookup	Lookup Field Name in Event	Default Value	
	severity	severity	Enter default value	X

+ Add field(s)

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
 + Add 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 ⓘ
<code>__status == 200</code>	5

+ Add Rule

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+)/`

The screenshot shows the 'Regex Extract' configuration page in Kibana. The top bar indicates the function is 'On' and the filter is 'sourcetype=='access_combined' || sourcetype=='aws:s3:accesslogs''. The 'Filter' field contains the same expression. The 'Description' field is 'Extract status from access logs'. The 'Final' toggle is set to 'No'. The 'Regex*' field contains the pattern `/ HTTP\/\d\.\d"\s(?:<__status>\d+)/`. The 'Additional Regex' section has a '+ Add Regex' button. The 'Source Field' is set to `_raw`.

Defining the Regex Extract Function

3. Add a Sampling Function to sample 5:1 when `__status==200`.
4. Save.

The screenshot shows the 'Sampling' configuration page in Kibana. The top bar indicates the function is 'On' and the filter is 'sourcetype=='access_combined' || sourcetype=='aws:s3:accesslogs''. The 'Filter' field contains the same expression. The 'Description' field is 'Check for status 200 and sample 5:1'. The 'Final' toggle is set to 'No'. The 'Sampling Rules' section contains a table with one rule:

Filter	Sampling Rate
<code>__status == 200</code>	5

Below the table is a '+ Add Rule' button.

Sampling success responses

Note About Sampling

Each time an event goes through the **Sampling** Function, an index-time `sampler::<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	<code>sourcetype=='aws:cloudfront:accesslogs'</code>
Amazon ELB Access Logs	<code>sourcetype=='aws:elb:accesslogs'</code>

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
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1
```

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
[+ Add 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 [?]
<code>__action=="ACCEPT"</code>	5

[+ Add Rule](#)

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	<code>sourcetype=='cisco:asa'</code>
Related sourcetypes to consider sampling:	<code>sourcetype=='cisco:fwm'</code> <code>sourcetype=='cisco:pix'</code>

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:

7 Mask source=='pii_naivecc' On ...

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)

+ Add Rule

Apply to Fields ⓘ

_raw x

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:** `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=XXXXXXXXXXXXXXXXXX`

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.

□ The steps below assume access to a command line and (more importantly) to your OS' filesystem. Where you lack such access – for example, in a [LogStream Cloud](#) deployment – load lookup files of all sizes via LogStream's UI, as outlined in [Lookups Library](#).

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.

i 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	<ul style="list-style-type: none"> • 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

```
$ 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

```
$ 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 JavaScript 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	<code>__masks.split(' ').indexOf('ssn') > -1</code>	On ...
3	Mask	<code>__masks.split(' ').indexOf('credit_card') > -1</code>	On ...
4	Mask	<code>__masks.split(' ').indexOf('auth_token') > -1</code>	On ...
5	Mask	<code>__masks.split(' ').indexOf('bearer_token') > -1</code>	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

+ Add 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.

IN	OUT
1	<pre> 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 # _time: 1612577911.286 az: a cribl_breaker: Break on newlines cribl_pipe: setting_index_by_region_availability_zone domain: compute.amazonaws.com host: ec2-35-162-133-145 region: us-west1 </pre>
2	<pre> 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 # _time: 1612582410.302 az: b cribl_breaker: Break on newlines cribl_pipe: setting_index_by_region_availability_zone domain: compute.amazonaws.com host: ec2-48-169-111-182 region: us-east2 </pre>
3	<pre> 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 # _time: 1612592951.841 az: a cribl_breaker: Break on newlines cribl_pipe: setting_index_by_region_availability_zone domain: compute.amazonaws.com host: ec2-21-187-232-201 region: asia-northeast3 </pre>

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 x

index x

sourcetype x

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	europa.+	emea_index_tier	cloud-init
4	northamerica.+	na_index_tier	cloud-init
5	southamerica.+	latam_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 ?

Help ?

true

Description ?

Lookup index and sourcetype using regex matching

Final ?

No

Lookup file path (.csv, .csv.gz)* ?

region_index_sourcetype.csv

Match Mode ?

Match Type ?

Regex

Most Specific

Lookup field(s) ?

	Lookup Field Name in Event ?	Corresponding Field Name in Lookup ?	
⋮	region	Enter field name	X

+ Add field(s)

Output field(s) ?

+ Add field(s)

> ADVANCED SETTINGS

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      q _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]: Lo
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-aw
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
      q az: a
      q cribl_breaker: Break on newlines
      q cribl_pipe: setting_index_by_region_availability_zone
      q domain: compute.amazonaws.com
      q host: ec2-35-162-133-145
      q index: usa_index_tier
      q region: us-west1
      q sourcetype: cloud-init

2      q _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', 'ucloudinit.s... Show more
21:33:30.302      # _time: 1612582410.302
-06:00      q az: b
      q cribl_breaker: Break on newlines
      q cribl_pipe: setting_index_by_region_availability_zone
      q domain: compute.amazonaws.com
      q host: ec2-48-169-111-182
      q index: usa_index_tier
      q region: us-east2
      q sourcetype: cloud-init

3      q _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      q az: a
      q cribl_breaker: Break on newlines
      q cribl_pipe: setting_index_by_region_availability_zone
      q domain: compute.amazonaws.com
      q host: ec2-21-187-232-201
      q index: apac_index_tier
      q region: asia-northeast3
      q 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 ?

Help ?

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')

+ Add Field

Results of the Eval Function and Replace Method

The `host_ip` field is now added to the events, displayed below host :

IN	OUT	
1	<pre>q _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 q az: a q cribl_breaker: Break on newlines q cribl_pipe: setting_index_by_region_availability_zone q domain: compute.amazonaws.com q host: ec2-35-162-133-145 q host_ip: 35.162.133.145 q index: usa_index_tier q region: us-west1 q sourcetype: cloud-init</pre>	
2	<pre>q _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 q az: b q cribl_breaker: Break on newlines q cribl_pipe: setting_index_by_region_availability_zone q domain: compute.amazonaws.com q host: ec2-48-169-111-182 q host_ip: 48.169.111.182 q index: usa_index_tier q region: us-east2 q sourcetype: cloud-init</pre>	
3	<pre>q _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 q az: a q cribl_breaker: Break on newlines q cribl_pipe: setting_index_by_region_availability_zone q domain: compute.amazonaws.com q host: ec2-21-187-232-201 q host_ip: 21.187.232.201 q index: apac_index_tier q region: asia-northeast3 q sourcetype: cloud-init</pre>	

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

Help

Description

Append region and az to sourcetype

Final

No

Evaluate Fields

Name	Value Expression
sourcetype	`\${sourcetype}_\${region}_\${az}`

+ Add Field

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!

IN	OUT
1	<pre> 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 # _time: 1612577911.286 az: a cribl_breaker: Break on newlines cribl_pipe: setting_index_by_region_availability_zone domain: compute.amazonaws.com host: ec2-35-162-133-145 host_ip: 35.162.133.145 index: usa_index_tier region: us-west1 sourcetype: cloud-init-us-west1_a </pre>
2	<pre> 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 # _time: 1612582410.302 az: b cribl_breaker: Break on newlines cribl_pipe: setting_index_by_region_availability_zone domain: compute.amazonaws.com host: ec2-48-169-111-182 host_ip: 48.169.111.182 index: usa_index_tier region: us-east2 sourcetype: cloud-init-us-east2_b </pre>
3	<pre> 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 # _time: 1612592951.841 az: a cribl_breaker: Break on newlines cribl_pipe: setting_index_by_region_availability_zone domain: compute.amazonaws.com host: ec2-21-187-232-201 host_ip: 21.187.232.201 index: apac_index_tier region: asia-northeast3 sourcetype: cloud-init-asia-northeast3_a </pre>

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 domain",
  },
  {
    "filter": "true",
    "conf": {
      "matchMode": "regex",
      "matchType": "specific",
      "reloadPeriodSec": 60,
      "addToEvent": false,
      "inFields": [
        {
          "eventField": "region"
        }
      ],
      "ignoreCase": false,
      "file": "region_index_sourcetype.csv"
    },
    "id": "lookup",
    "disabled": false,
    "description": "Lookup index and sourcetype using regex matching"
  },
  {
    "filter": "true",
    "conf": {
      "add": [
        {
          "name": "host_ip",
          "value": "host.replace(/\\w+-(\\d+)-(\\d+)-(\\d+)-(\\d+)/. '$'

```

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

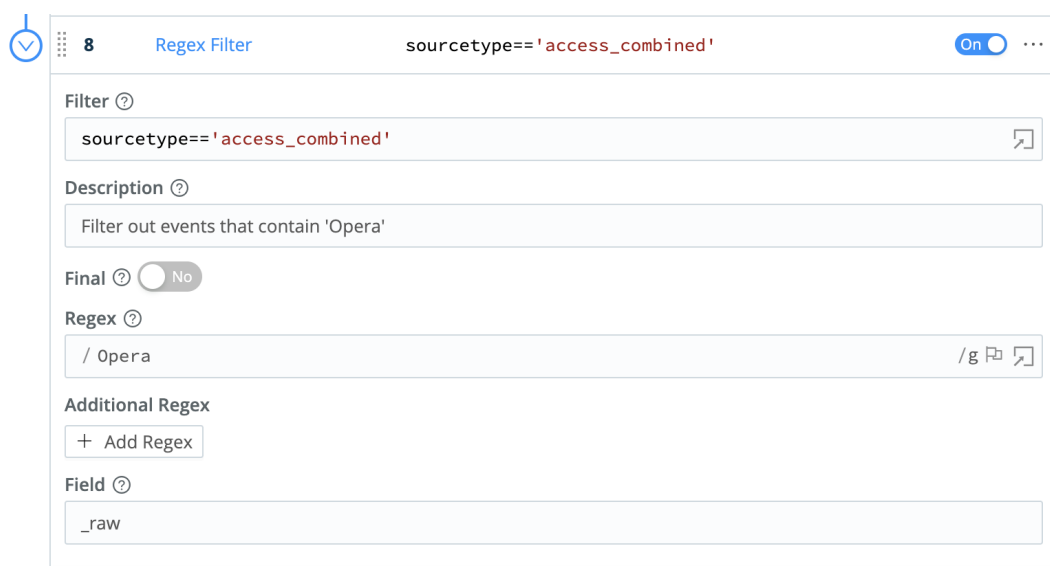
To filter events in real time, we use the out-of-the-box **Regex Filter** Function. This is similar to nullqueueing with TRANSFORMS in Splunk, but the matching condition is way more flexible.

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:



The screenshot shows the configuration for a 'Regex Filter' function in Splunk. The function is named '8 Regex Filter' and is currently 'On'. The configuration includes:

- Filter**: A text field containing the expression `sourcetype=='access_combined'`.
- Description**: A text field containing the text 'Filter out events that contain 'Opera''.
- Final**: A toggle switch set to 'No'.
- Regex**: A text field containing the pattern `/ Opera`.
- Additional Regex**: A section with a '+ Add Regex' button.
- Field**: A text field containing the field name `_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:

The screenshot shows the 'Regex Filter' configuration window in Kibana. At the top, it displays '8' filters and the current filter's name 'Regex Filter'. The filter expression is `sourcetype=='access_combined' && host.endsWith('...'` with a toggle set to 'On'. The 'Filter' section contains the full expression `sourcetype=='access_combined' && host.endsWith('dnto.ca')`. The 'Description' section has the text 'Filter out events that contain 'Opera''. The 'Final' section has a toggle set to 'No'. The 'Regex' section contains the pattern `/ Opera`. The 'Additional Regex' section has a '+ Add Regex' button. The 'Field' section has the field name `_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`.)

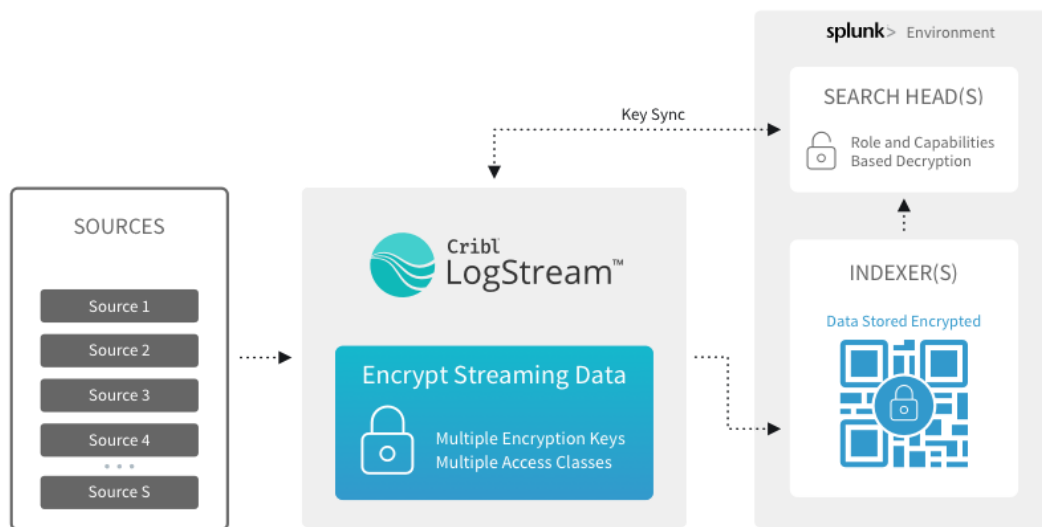
Keys and Key Classes

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 + Add New

To protect against accidental changes, key parameters can *only* be modified through configuration files. Get Key Bundle

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

Description

Encryption Algorithm*

KMS for this key*

Key Class*

Expiration time

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
```

...

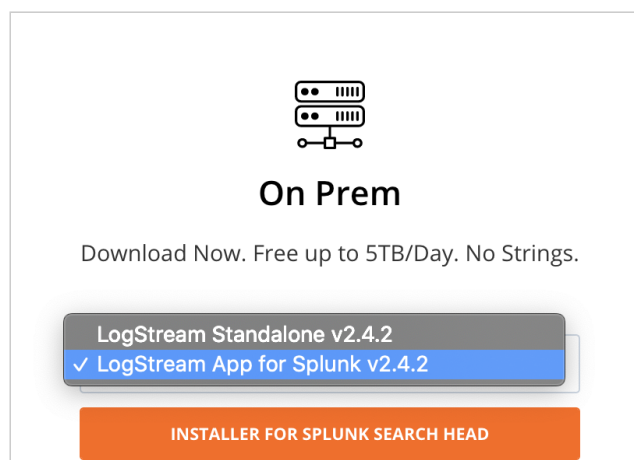
```
$CRIBL_HOME/bin/cribl keys add -c <N> -i -g uk
```

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

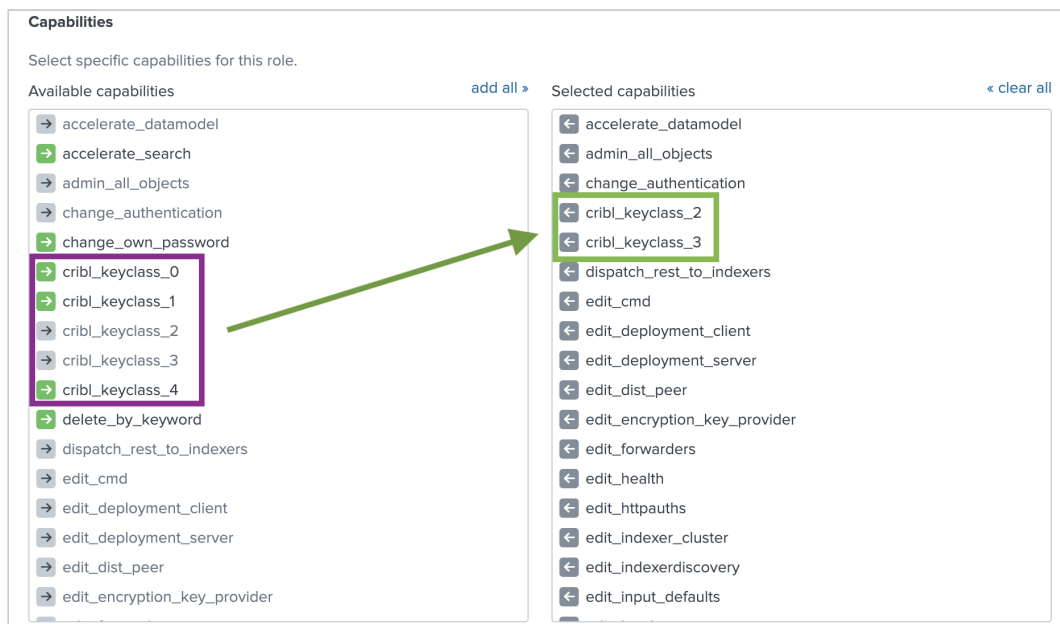
- Download the Cribl/LogStream App for Splunk from Cribl's [Download LogStream](#) page: In the **On Prem** section, select the Splunk app from the drop-down list, as shown. Clicking the orange button downloads a file named:
`cribl-splunk-app-<version-#>-<hash-#>-linux-x64.tgz` .



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.



Selecting capabilities

- 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.

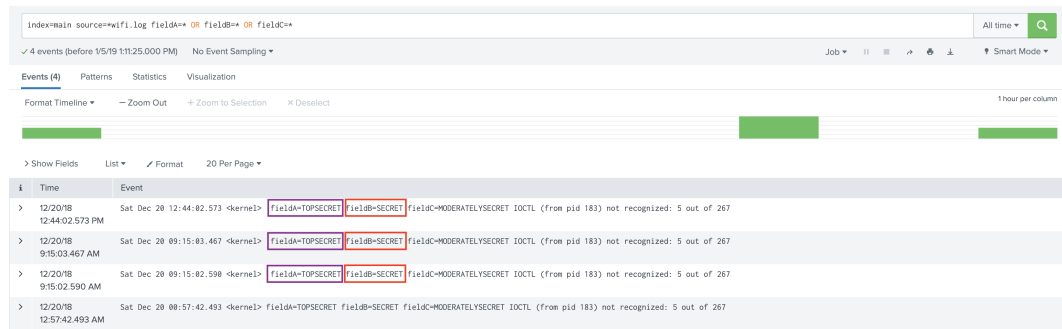
Sync/copy these files over to their counterparts on the search head (decryption side). In a non-Splunk integration, you would copy these assets to wherever decryption will take place.

i Modifying Keys

When you update keys by editing the `keys.json` file, you must add them back to 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`.



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

Filter true

Description

Final Capture field values

Encrypt

Match Regex	Replace Expression
/fieldA=(\w+)/	`\${g1}\${C.Crypto.encrypt(g2,1)}`
/fieldB=(\w+)/	`\${g1}\${C.Crypto.encrypt(g2,2)}`

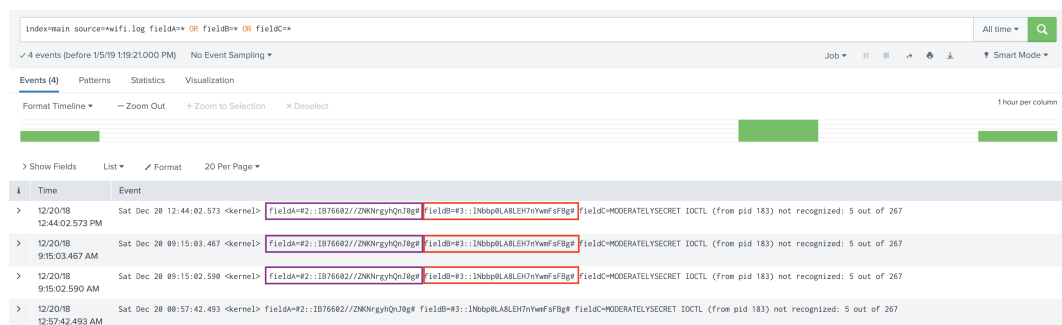
Add Rule

Apply to Fields

_raw

Encrypting two fields with separate key classes

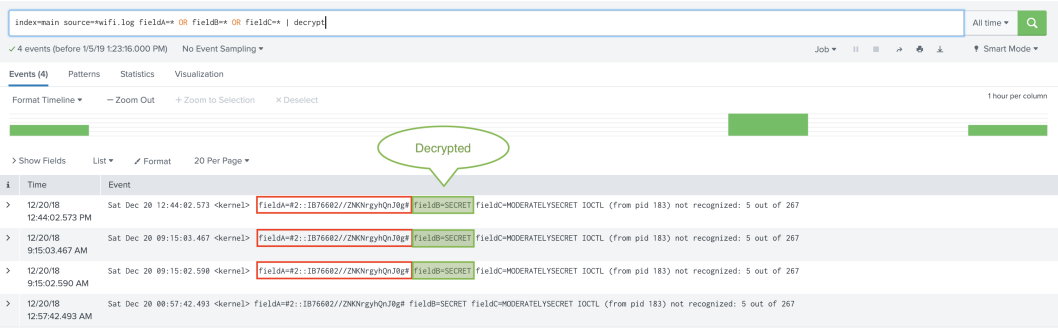
After Encryption: again, notice the values of `fieldA` and `fieldB`.



Time	Event
12/20/18 12:44:02.573 PM	Sat Dec 20 12:44:02.573 <kernel> fieldA=#2::1B76692//ZM0Nrgyh0n3Bgt fieldB=#3::1NbapLABLEH7nWmfSfBgt 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::1B76692//ZM0Nrgyh0n3Bgt fieldB=#3::1NbapLABLEH7nWmfSfBgt 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::1B76692//ZM0Nrgyh0n3Bgt fieldB=#3::1NbapLABLEH7nWmfSfBgt 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::1B76692//ZM0Nrgyh0n3Bgt fieldB=#3::1NbapLABLEH7nWmfSfBgt 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 .



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      |   _raw: <167>2020-02-27T19:00:23.468Z esxi-3.madronecg.com Hostd: verbose hostd[D2C2B70] [Originator@68  
2020-02-27       |         76 sub=PropertyProvider] RecordOp ASSIGN: guest.disk, 9. Sent notification immediately.  
11:00:23.468    | # _time: 1582830023.468  
-08:00          |   appname: Hostd  
                | # facility: 20  
                |   facilityName: local4  
                |   host: esxi-3.madronecg.com  
                |   message: verbose hostd[D2C2B70] [Originator@6876 sub=PropertyProvider] RecordOp ASSIGN: guest.disk,  
                |           9. Sent notification immediately.  
                | # severity: 7  
                |   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
2020-02-27      ent notification immediately.
11:00:23.468 # _time: 1582830023.468
-08:00      α appname: Hostd
      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.

Pipelines > syslog-conditioning + Add Function 

Attach Pipeline to Route IN 4.14k OUT 4.14k ERR 0

#	Function	Filter	
1	Eval	true	On ...

Filter 



true 

Description 

Enter a description

Final  No

Evaluate Fields 


Name	Value Expression	
sourcetype	'syslog'	 
source	__inputId	 


+ Add Field


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 volume

Final  No


Evaluate Fields 


Name	Value Expression	
_raw	message	 

+ Add Field


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`.

The screenshot shows the configuration for the third Eval Function. The title bar indicates it is function 3, named 'Eval', with the filter condition `severity != null && facility != null` and a status of 'On'. The configuration panel includes a 'Filter' section with the same condition, a 'Description' field with the text 'Drop redundant fields', a 'Final' toggle set to 'No', an 'Evaluate Fields' section with an 'Add Field' button, a 'Keep Fields' section with an input field for 'Enter field names', and a 'Remove Fields' section with two fields: `severity` and `facility`, each with a delete icon.

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`.

The screenshot shows the configuration for the fourth Eval Function. The title bar indicates it is function 4, named 'Eval', with the filter condition `procid=='-'` and a status of 'On'. The configuration panel includes a 'Filter' section with the same condition, a 'Description' field with the text 'Remove useless values for procid', a 'Final' toggle set to 'No', an 'Evaluate Fields' section with an 'Add Field' button, a 'Keep Fields' section with an input field for 'Enter field names', and a 'Remove Fields' section with one field: `procid`, with a delete icon.

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.

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.

4

Firewall Lo... __inputId.startsWith('sysl... firewall_activi... 45.4... On ...

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

#	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

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.



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.

The screenshot shows the LogStream interface. On the left, a pipeline named 'elastic_common_schema' is configured with five steps: 1. Comment (Extract each element from the original event.), 2. Regex Extract (Filter: true), 3. Lookup (Filter: true), 4. Comment, and 5. Eval (Filter: true). The pipeline is attached to the route 'weblogs_to_elastic'. On the right, the 'Preview' tab shows a table of events. The first event is a GET request from 192.73.30.102 to 1558296286. The second event is a GET request from 192.73.30.102 to 1558296287. The table shows the raw event data and the transformed JSON output for each event.

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.)

The screenshot shows the 'Manage SplunkTCP Sources' interface. At the top, there's a search bar and an 'Add New' button. Below is a table with columns: ID, Address, Port, IP Whitelist Regex, TLS, Enabled, and Status. A row for 'New Splunk source' is expanded, showing 'Disabled' and a toggle switch. To the left is a sidebar with settings categories: General Settings (selected), TLS Settings (server side), Processing Settings, Event Breakers, Fields (metadata), Pre-processing, and Advanced Settings. The main area displays the configuration for the selected source, including fields for Input ID, Address (0.0.0.0), Port (with a blue border), and IP Whitelist Regex. At the bottom right are 'Prev', 'Next', 'Cancel', and 'Save' buttons.

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

Regex		
/ (?<__method>GET HEAD POST PUT DELETE CONNECT OPTIONS TRACE)	/	X
/ HTTP\/(?<__version>[0-9\.\.]*)"	/	X
/ \s(?<__status>\d\d\d)\s	/	X
/ (?<__ip_address>(?:[0-9]{1,3}\.){3}[0-9]{1,3})\s	/	X
/ (?<__url>\s\/([^\s]*))	/	X

+ Add Regex

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.

3LookuptrueOn...

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`.

4GeoIPtrueOn...

Filter ⓘ
true

Description ⓘ
Enter a description

Final ⓘ ☐ No

GeoIP File (.mddb) ⓘ
/opt/maxmind/GeoLite2-City.mddb

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.

6

Eval

true

On

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.

Discover

Search

KQL

Aug 25, 2020 @ 21:00:00.0 → Aug 26, 2020 @ 00:00:00.0

Refresh

cribl

Search field names

Filter by type

Selected fields

Available fields

8 hits

Aug 25, 2020 @ 21:00:00.000 - Aug 26, 2020 @ 00:00:00.000

Count

Time

@timestamp per 5 minutes

Aug 25, 2020 @ 22:43:55.000

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: UNET source.address: 173.73.8.192 source.ip: 173.73.8.192 host.hostname: ip-172-26-11-87 host.os.kernel: 4.19.8-10-cloud-amd4 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, fette::9e:df:fc29:8354 host.name: ip-172-26-11-87 host.id: 15c9b07e8b9ccsa9ede375d377ea host.mac: 02:9e:8d:28:83:34 host.architecture: x86_64

Expanded document

Table

JSON

{
 "_index": "cribl",
 "_type": "_doc",
 "_id": "inCw0Q87b4900BuJqg",
 "_version": 1,
 "_score": null,
 "_source": {
 "source": {
 "geo": {
 "continent_name": "North America",
 "region_iso_code": "VA",
 "city_name": "Springfield",

Page 923 of 1087

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?

```
IN OUT
1
2021-01-05
10:09:07.8
-06:00

  <?xml version='1.0' encoding='UTF-16'>
    <Provider Name='Microsoft-Windows-Security-Auditing' Guid='{54849625-5478-49
94-ASB4-3CB3B328C38D}'>
      <EventID>4625</EventID>
      <Version>0</Version>
      <Level>0</Level>
      <Task>12544</Task>
      <Opcode>0</Opcode>
      <Keywords>0x8010000000000000</Keywords>
      <TimeCreated SystemTime='2020-11-12T17:27:33.608596000Z'>
        <EventRecordID>1598720</EventRecordID>
        <Correlation>
          <Execution ProcessID='740' ThreadID='9
48'>
            <Channel>Security</Channel>
            <Computer>EC2AMAZ-CPMG635.cribl.poc</Computer>
            <Security>
              <System>
                <EventData>
                  <Data Name='SubjectUserSid'>S-1-0-0</Data>
                  <Data 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'>0x23
13</Data>
                  <Data Name='SubStatus'>0xc000006a</Data>
                  <Data Name='LogonType'>3</Data>
                  <Data Name='LogonProcessName'>NTLMSsp</Data>
                  <Data Name='AuthenticationPackage
name'>NTLM</Data>
                  <Data Name='WorkstationName'></Data>
                  <Data Name='TransmittedServices'></Data>
                  <Data Name='LinkPackageName'></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>
              </System>
            </Security>
          </Execution>
        </Correlation>
      </TimeCreated>
    </Provider>
  </Event>
</xml>

# _time: 1609552209.078
cribl_breaker: Break on newlines
```

	_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%

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:

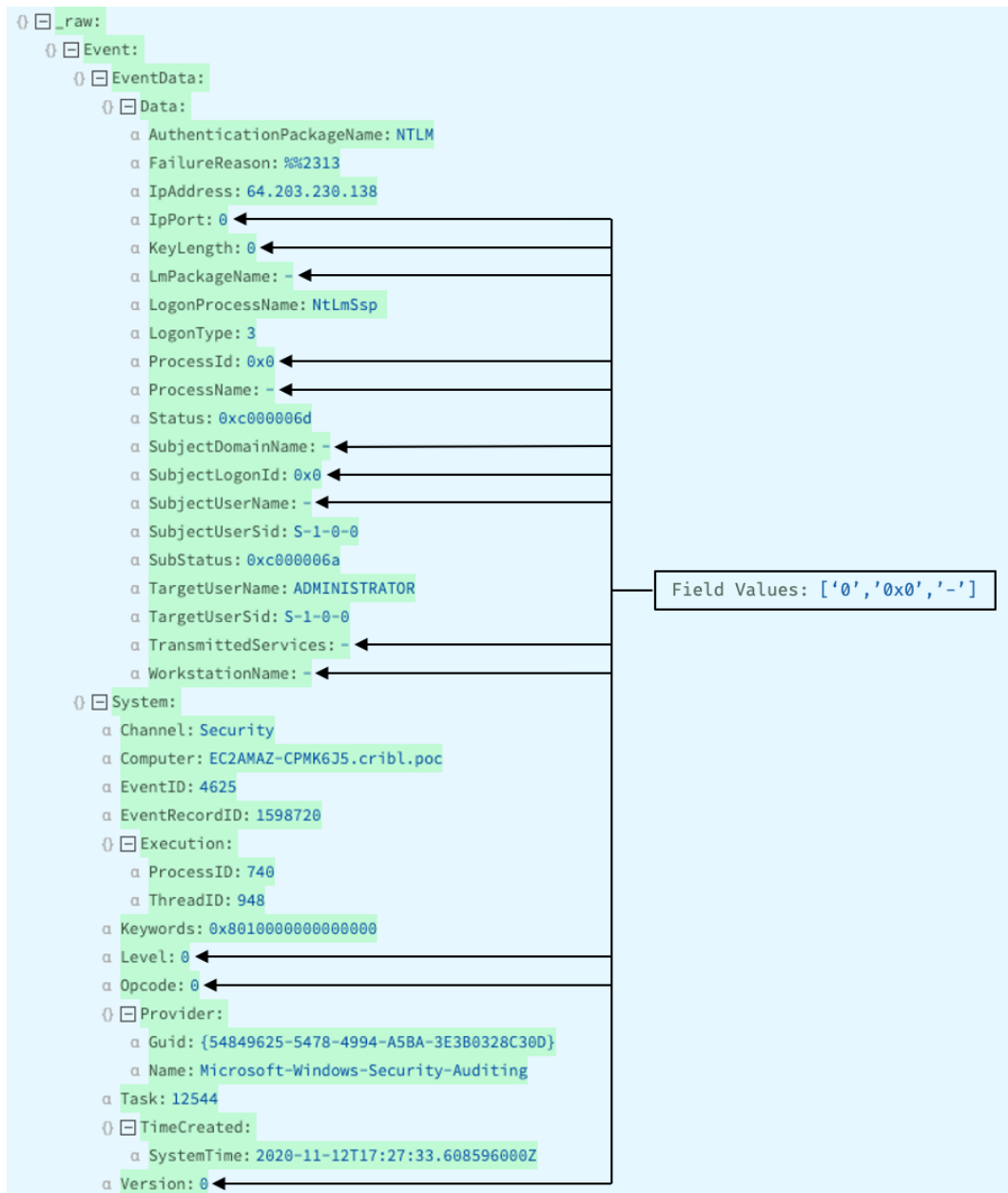
Page 926 of 1087

	_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'`, `'0x0'`, or `'-'`) bloat events, demanding extra infrastructure and storage:



Let's amplify the reduction by removing all of the fields whose values are in the set: ['0', '0x0', '-']. This improved version of our Eval Function parses the Windows XML event, and discards ['0', '0x0', '-'] values. (Its preceding row also tidies up events by removing tabs and curly braces, and replacing newlines and returns with commas.) The result is an even **smaller** prettified JSON object:

2

Eval

true

On ...

Filter ?

Help ?

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', '0x0', '-'] are removed:

```

{} _raw:
  {} Event:
    {} EventData:
      {} Data:
        α AuthenticationPackageName: NTLM
        α FailureReason: %%2313
        α IPAddress: 64.203.230.138
        α LogonProcessName: NtLmSsp
        α LogonType: 3
        α Status: 0xc000006d
        α SubjectUserSid: S-1-0-0
        α SubStatus: 0xc000006a
        α TargetUserName: ADMINISTRATOR
        α TargetUserSid: S-1-0-0
      {} System:
        α Channel: Security
        α Computer: EC2AMAZ-CPMK6J5.cribl.poc
        α EventID: 4625
        α EventRecordID: 1598720
      {} Execution:
        α ProcessID: 740
        α ThreadID: 948
        α Keywords: 0x8010000000000000
      {} Provider:
        α Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
        α Name: Microsoft-Windows-Security-Auditing
        α Task: 12544
      {} TimeCreated:
        α 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:

3 Flatten On

Filter ? Help
true

Description ?
Enter a description

Final ? ☐ No

Fields ?
_raw X

Prefix ? Depth ? Delimiter ?
Enter prefix 10 -

Using Flatten, we've successfully created top-level fields from the nested JSON structure:

```

raw: <Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider-Name='Microsoft-Windows-Security-Auditing'-Guid='{54849625-5478-4994-A5BA-3E3B0328C30D}'><EventID>4625</EventID></System></Event>
raw_Event_EventData_Data_AuthenticationPackageName: NTLM
raw_Event_EventData_Data_FailureReason: %%2313
raw_Event_EventData_Data_IpAddress: 64.203.230.138
raw_Event_EventData_Data_LogonProcessName: NtLmSsp
raw_Event_EventData_Data_LogonType: 3
raw_Event_EventData_Data_Status: 0xc000006d
raw_Event_EventData_Data_SubjectUserSid: S-1-0-0
raw_Event_EventData_Data_SubStatus: 0xc000006a
raw_Event_EventData_Data_TargetUserName: ADMINISTRATOR
raw_Event_EventData_Data_TargetUserSid: S-1-0-0
raw_Event_System_Channel: Security
raw_Event_System_Computer: EC2AMAZ-CPMK6J5.cribl.poc
raw_Event_System_EventID: 4625
raw_Event_System_EventRecordID: 1598720
raw_Event_System_Execution_ProcessID: 740
raw_Event_System_Execution_ThreadID: 948
raw_Event_System_Keywords: 0x8010000000000000
raw_Event_System_Provider_Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
raw_Event_System_Provider_Name: Microsoft-Windows-Security-Auditing
raw_Event_System_Task: 12544
raw_Event_System_TimeCreated_SystemTime: 2020-11-12T17:27:33.608596000Z
_time: 1609852209.078
cribl_breaker: Break on newlines
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 ? Help ?

Description ?

Final ? ☐ No

Base Fields ?

Rename Fields ?

Renaming Expression ?

The resulting field names are now much more compact, and easier to work with and manage:

```

1 <_raw: <Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider-Name='Microsoft-Windows-Security-Auditing'-Guid='{54849625-5478-4994-A5BA-3E3B0328C30D}'><EventID>4625</EventID><Channel>Security</Channel><Source>NTLM</Source><Computer>EC2AMAZ-CPMK6J5.cribl.poc</Computer><CriblBreaker>Break on newlines</CriblBreaker><CriblPipe>Parse_Windows_XML_Events</CriblPipe><EventID>4625</EventID><EventRecordID>1598720</EventRecordID><FailureReason>%%2313</FailureReason><Guid>{54849625-5478-4994-A5BA-3E3B0328C30D}</Guid><IpAddress>64.203.230.138</IpAddress><Keywords>0x8010000000000000</Keywords><LogonProcessName>NtLmSsp</LogonProcessName><LogonType>3</LogonType><Name>Microsoft-Windows-Security-Auditing</Name><ProcessID>740</ProcessID><Status>0xc000006d</Status><SubjectUserSid>S-1-0-0</SubjectUserSid><SubStatus>0xc000006a</SubStatus><SystemTime>2020-11-12T17:27:33.608596000Z</SystemTime><TargetUserName>ADMINISTRATOR</TargetUserName><TargetUserSid>S-1-0-0</TargetUserSid><Task>12544</Task><ThreadID>948</ThreadID></Event></System></Event>
```

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` :

The screenshot shows the configuration for the 'Serialize' function. The 'Filter' is set to 'true'. The 'Description' field is empty. The 'Final' toggle is set to 'No'. The 'Type' is set to 'Key=Value Pairs' and the 'Library' is 'Select from Library'. The 'Fields to serialize' list contains '!*', 'cribl_breaker', and '*'. The 'Source Field' is 'Source field name' and the 'Destination Field' is '_raw'. The 'Clean Fields' toggle is set to 'No'.

In the Preview pane, the `_raw` field is now back, serialized into compact, tidy key-value pairs:

```

# _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
# AuthenticationPackageName: NTLM
# Channel: Security
# Computer: EC2AMAZ-CPMK6J5.cribl.poc
# cribl_breaker: Break on newlines
# cribl_pipe: Parse_Windows_XML_Events
# EventID: 4625
# EventRecordID: 1598720
# FailureReason: %2313
# Guid: {54849625-5478-4994-A5BA-3E3B0328C30D}
# IPAddress: 64.203.230.138
# Keywords: 0x8010000000000000
# LogonProcessName: NtLmSsp
# LogonType: 3
# Name: Microsoft-Windows-Security-Auditing
# ProcessID: 740
# Status: 0xc000006d
# SubjectUserSid: S-1-0-0
# SubStatus: 0xc000006a
# SystemTime: 2020-11-12T17:27:33.608596000Z
# TargetUserName: ADMINISTRATOR
# TargetUserSid: S-1-0-0
# Task: 12544
# 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:

6

Eval

On

Filter

true

Help

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:

```

raw: Name=Microsoft-Windows-Security-Auditing Guid={54849625-5478-4994-A5BA-3E3B0320C300} 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
cribl_breaker: Break on newlines
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 parameters: None

Collect headers:

accept: 'application/json'

x-rapidapi-key:

'e4068647ffmsh65536596798f49dp17e998jsn342bac862377'

x-rapidapi-host: 'matchilling-chuck-norris-jokes-v1.p.rapidapi.com'

useQueryString: true

Pagination: None

Authentication: None

Event Breaker: JSON Newline Delimited – use LogStream's built-in **Cribl > ndjson** rule, and associate it with the Collector to parse the JSON document.

Jobs > Collectors > chuck-jokes

Event Breakers
Fields (Metadata)
Result Routing
Advanced Settings

> DISCOVER

< COLLECT

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	'e4868647ffmsh65536596798f49dp17e998jsn342bac862377'
x-rapidapi-host	'matchilling-chuck-norris-jokes-v1.p.rapidapi.com'
useQueryString	true

+ Add header

Pagination*

None

> AUTHENTICATION

▶ Run
⌚ Schedule
Delete Collector
Clone Collector

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:

Previewing chuck-jokes:1613406562.9

Filter Expression*
true

Fields
All
None

	Value
1	{ categories: [], created_at: 2020-01-05 13:42:25.985626, icon_url: https://assets.chucknorris.host/img/avatar/chuck-norris.png, id: Cv13HPbHTLiBecApP8AAQ, updated_at: 2020-01-05 13:42:25.985626, url: https://api.chucknorris.io/jokes/Cv13HPbHTLiBecApP8AAQ, 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 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.

Collect headers: None

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

Custom Command

Event Breakers

Fields (Metadata)

Result Routing

Advanced Settings

Collector type* ?
REST

> DISCOVER

▼ COLLECT

Collect URL* ?
'http://www.neowsapp.com/rest/v1/neo/browse?api_key=oDa6w0fjsKEb1N3bMA5dMLhatMJ4WC5Xt0BTrLrk'

Collect method* ?
GET

Collect parameters ?
+ Add parameter

Collect headers ?
+ Add header

Pagination* ?
Response Body Attribute

Response Attribute* ?
next

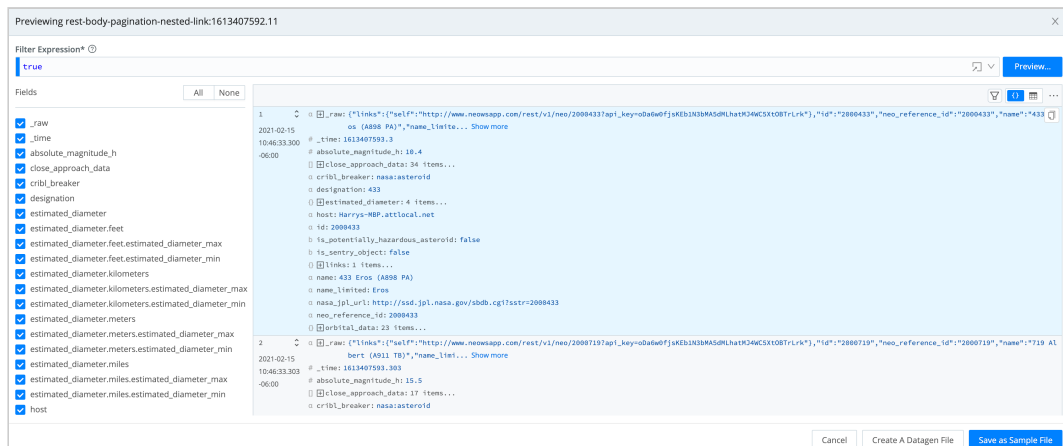
Max Pages* ?
10

> 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.



Paginated events



This API allows a certain number of calls/month. Cribl recommends that you not schedule this Collector – run it ad-hoc, for testing only.

3. HTTP GET with Pagination via Response Body Attribute

This example uses Response Body Attribute pagination, which returns a token that is passed as a request parameter to retrieve subsequent pages of data. The only difference between this example and [Example 2](#) is how the Response Body Attribute is used.



To authenticate against the GreyNoise endpoint used in this example, set up a trial account according to GreyNoise's [Setting Up a Trial Account](#) documentation.

Discover type: None

Collect URL: 'https://api.greynoise.io/v2/experimental/gnql'

Collect method: GET

Collect parameters:

query: 'last_seen:1d'

scroll: `\${scroll}`

Collect headers:

accept: 'application/json'

key: '<your-GreyNoise-API-key-here>'

Pagination: Response Body Attribute

Response attribute: scroll

Max pages: 10 (or 0 to pull all data)

Authentication: None

Event Breaker: JSON Array – use the configuration shown here:

greynoise

Filter Condition* ?

true

EVENT BREAKER SETTINGS

Enabled ? ☒ Yes

Event Breaker Type* ?

JSON Array

Array Field ?

data

JSON Extract Fields ? ☒ Yes

Timestamp Field ?

Enter timestamp field

Max Event Bytes ?

51200

TIMESTAMP SETTINGS

Timestamp Anchor* ?

/

Timestamp Format* ?

☐ Autotimestamp Scan Depth ?
☐ Manual Format ?
☒ Current Time ?

Default Timezone ?

Local

Event Breaker configuration

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.

Jobs > Collectors > gretynoise

Meta (metadata)

Result Routing

Advanced Settings

Collect URL*

`*https://api.greynoise.io/v2/experimental/gnql*`

Collect method*

GET

Collect parameters

Name	Value
query	<code>'last_seen:Id'</code>
scroll	<code>'\${scroll}'</code>

+ Add parameter

Collect headers

Name	Value
key	<code>'sqcPG38hoTnklZ6VxbawHISkgFzaOGAMgwchAAmOwxGgOHDt68nh8uU7ADf5Msuc0'</code>
accept	<code>'application/json'</code>

+ Add header

Pagination*

Response Body Attribute

Response Attribute*

scroll

Max Pages*

0

> AUTHENTICATION

Run Schedule Delete Collector Clone Collector

Collector configuration for HTTP GET, paginated via Response Body Attribute

Collector Output

Previewing rest-body-pagination-nested-link:1613408462.12

Filter Expression*

`true`

Fields

All None

☒ _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 2021-02-15 11:01:02.826 # _time: 1613408462.826
-06:00 # absolute_magnitude_h: 10.4
      # close_approach_data: 34 items...
      # cribl_breaker: nasa:asteroid
      # designation: 433
      # estimated_diameter: 4 items...
      # host: Harrys-MBP.attlocal.net
      # id: 2008433
      # is_potentially_hazardous_asteroid: false
      # is_sentry_object: false
      # links: 1 items...
      # name: 433 Eros (A000 PA)
      # name_titled: Eros
      # nasa_jpl_url: http://ssd.jpl.nasa.gov/sbdb.cgi?str=2008433
      # neo_reference_id: 2008433
      # orbital_data: 23 items...

2 2021-02-15 11:01:02.828 # _time: 1613408462.828
-06:00 # absolute_magnitude_h: 13.5
      # close_approach_data: 17 items...
      # cribl_breaker: nasa:asteroid
  
```

Cancel Create A Datalog File Save as Sample File

Paginated events



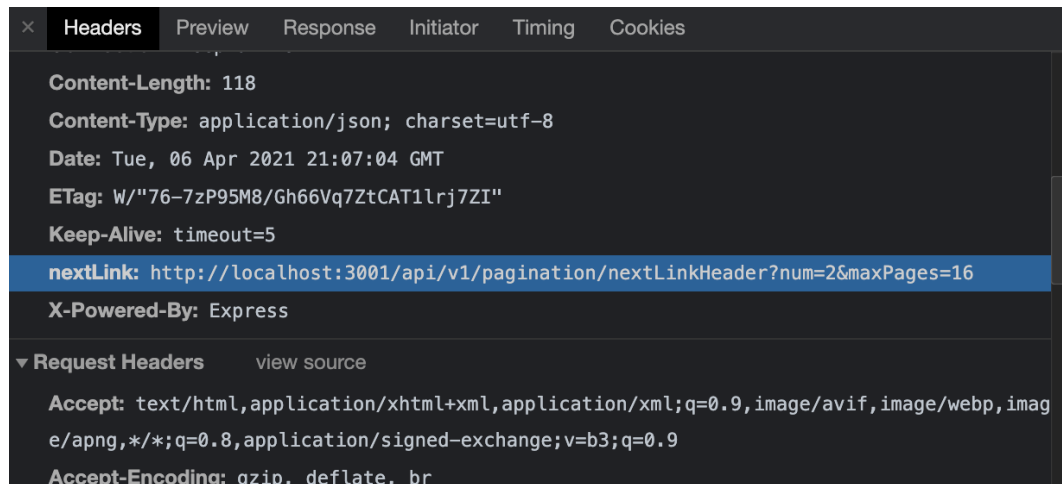
This API allows a certain number of calls/month. Cribl recommends that you not schedule this Collector – run it ad-hoc, for testing only.

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:



Next-page URL passed as Response Header Attribute

Collector Configuration

Discover type: None

Collect URL: '<http://localhost:3001/api/v1/pagination/nextLinkHeader?num=1&maxPages=16>'

Collect parameters: None

Collect headers: None

Pagination: Response Header Attribute

Response attribute: nextLink

Authentication: None

Event Breaker: None

- You can modify the `maxPages` URL parameter to control how many pages this call returns.

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 be 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 parameters: None

Discover headers: None

Discover data field: items

Collect URL: 'http://localhost:9000/api/v1/system/logs/' +
C.Encode.uri(`\${id}`)

Collect parameters: None

Collect headers: None

Pagination: None

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

Jobs > Collectors > cribl-logs	
Collector Settings	Collector ID* ⓘ cribl-logs
Result Settings ^	Collector type* ⓘ REST
Custom Command	DISCOVER
Event Breakers	Discover Type* ⓘ HTTP Request
Fields (Metadata)	Discover URL* ⓘ <code>*http://localhost:9000/api/v1/system/logs*</code>
Result Routing	Discover method* ⓘ GET
Advanced Settings	Discover parameters ⓘ + Add parameter
	Discover Headers ⓘ + Add Header
	Discover Data Field ⓘ Items
	COLLECT
	Collect URL* ⓘ <code>*http://localhost:9000/api/v1/system/logs/' + C.Encode.uri('\${id}')</code>
	Collect method* ⓘ GET
	Collect parameters ⓘ + Add parameter
	Collect headers ⓘ + Add header
	Pagination* ⓘ None
	AUTHENTICATION
	Authentication* ⓘ Login
	Login URL* ⓘ <code>*http://localhost:9000/api/v1/auth/login*</code>
	Username* ⓘ *****
	Password* ⓘ *****
	POST Body* ⓘ <code>{ "username": "\${username}", "password": "\${password}" }</code>
	Token Attribute* ⓘ token
	Authorize Expression* ⓘ <code>*Bearer \${token}*</code>

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"
    }
  ]
}
```

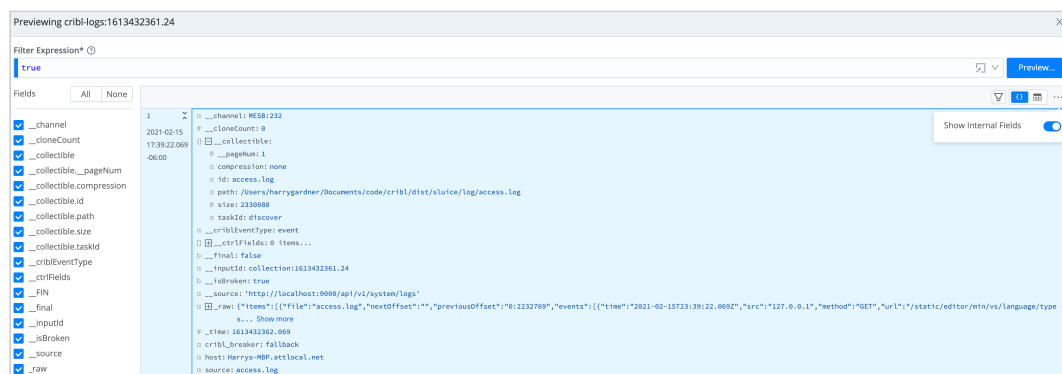
The **Discover Data Field** is used to define the array in Discover results that contains the list of items to discover. Here, each item is an object, with an attribute ID that is referenced in the Collect calls. So the Discover call generates a list of items for which Collect tasks will be created.

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:



__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}`)
```

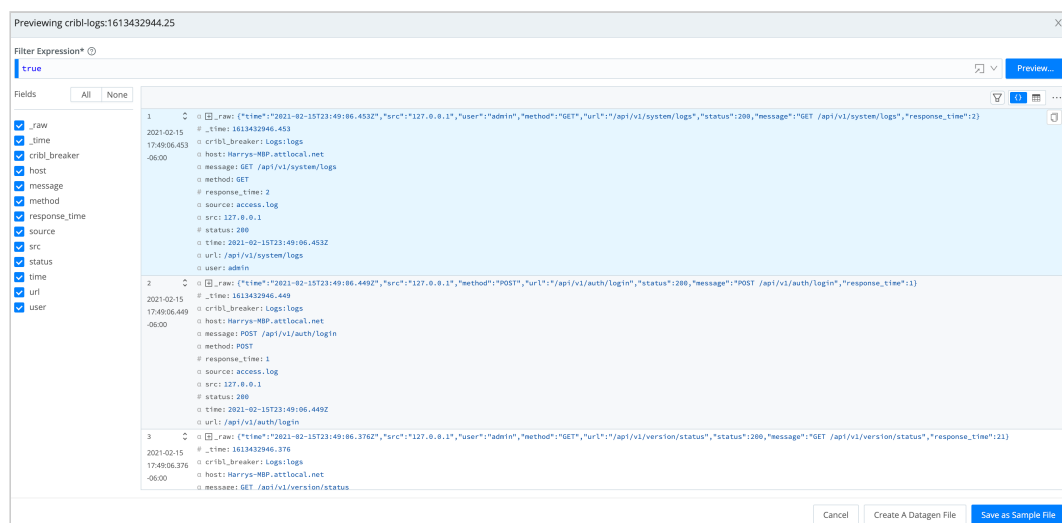
Because the variable is used in the path, and it might contain unsafe ASCII characters (specifically, space), we need to URL-encode the variable. This is the only case where the REST Collector requires URI encoding – variables that are defined directly as part of the URL. (Request parameters, not contained 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:



Collected data

i If this example fails with errors of the form `statusCode: 429...` Too many requests – see [Common Errors and Warnings](#) to resolve this by relaxing the login rate limit.

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 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}

The screenshot shows the configuration for a collector named 'Item-list-weather2'. The interface is divided into several sections: Collector Settings, Result Settings, Custom Command, Event Breakers, Fields (Metadata), Result Routing, and Advanced Settings. The 'Fields (Metadata)' section is currently selected, showing a table of fields to be collected. The 'Collect parameters' table has columns for Name and Value, with rows for 'type' (Link), 'units' (imperial), and 'q' (\${id}). The 'Collect headers' table also has columns for Name and Value, with rows for 'x-rapidapi-host' (community-open-weather-map.p.rapidapi.com), 'x-rapidapi-key' (78934c846cmsh70cb53f75a8a54bp119d21jsn29df549b4fd6), and 'useQueryString' (true). The 'Pagination' is set to 'None' and 'Authentication' is set to 'None'.

Name	Value
type	'Link'
units	'imperial'
q	\${id}

Name	Value
x-rapidapi-host	'community-open-weather-map.p.rapidapi.com'
x-rapidapi-key	'78934c846cmsh70cb53f75a8a54bp119d21jsn29df549b4fd6'
useQueryString	true

Collector configuration for Discovery via Item List

Jobs > Collectors > item-list-weather2		
Collector Settings	Fields	
Result Settings		
Custom Command		
Event Breakers		
Fields (Metadata)		
Result Routing		
Advanced Settings		
	Name	Value
	job	'weather-\${__collectible.id}'
	city	'\${__collectible.id}'
	+ Add Field	

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:

Previewing item-list-weather2:1613484194.11		
Filter Expression* <code>true</code>		
Fields	All	None
<input checked="" type="checkbox"/> __channel	<input checked="" type="checkbox"/> __cloneCount	<input checked="" type="checkbox"/> __collectible
<input checked="" type="checkbox"/> __collectible__pageNum	<input checked="" type="checkbox"/> __collectible.compression	<input checked="" type="checkbox"/> __collectible.id
<input checked="" type="checkbox"/> __collectible.taskid	<input checked="" type="checkbox"/> __criblEventType	<input checked="" type="checkbox"/> __inputId
<input checked="" type="checkbox"/> __isBroken	<input checked="" type="checkbox"/> __raw	<input checked="" type="checkbox"/> __time
<input checked="" type="checkbox"/> city	<input checked="" type="checkbox"/> cod	<input checked="" type="checkbox"/> count
<input checked="" type="checkbox"/> cribl_breaker	<input checked="" type="checkbox"/> host	<input checked="" type="checkbox"/> job
<input checked="" type="checkbox"/> list	<input checked="" type="checkbox"/> message	
1	<pre> { "__channel": "MESB:49", "__cloneCount": 0, "__collectible": { "__pageNum": 1, "compression": "none", "id": "Nashville", "taskId": "discover", "__criblEventType": "event", "__inputId": "collection:1613484194.11", "isBroken": true, "raw": { "message": { "accurate": "200", "count": 5, "list": [{ "id": "4644585", "name": "Nashville", "coord": { "lat": 36.1699, "lon": -86.7844 }, "main": { "temp": 10.4, "feels_like": 9.18, "temp_min": 10.4, "temp_max": 10.4 } }] } }, "time": "1613484194.11", "city": "Nashville", "cod": "200", "count": 5, "cribl_breaker": "ND_350k:rdjson", "host": "Harrys-MBP.attlocal.net", "job": "weather-Nashville", "list": 0, "message": "accurate" }, "time": "1613484194.11" } </pre>	
2	<pre> { "__channel": "MESB:190", "__cloneCount": 0, "__collectible": { "count": 4, "items": [{ "id": "1613484194.11", "message": { "accurate": "200", "count": 5, "list": [{ "id": "4644585", "name": "Nashville", "coord": { "lat": 36.1699, "lon": -86.7844 }, "main": { "temp": 10.4, "feels_like": 9.18, "temp_min": 10.4, "temp_max": 10.4 } }] } }], "time": "1613484194.11", "city": "Nashville", "cod": "200", "count": 5, "cribl_breaker": "ND_350k:rdjson", "host": "Harrys-MBP.attlocal.net", "job": "weather-Nashville", "list": 0, "message": "accurate" }, "time": "1613484194.11" } </pre>	

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 headers: None

Collect parameters:

lat: `\${lat}`

lon: `\${lon}`

appid: '438d61a1db9e713240b30140e9ddfea2'

Pagination: None

Authentication: None

Event Breaker: JSON Newline Delimited – Use a rule like **Cribl > ndjson** to parse each event and extract fields.

Fields (Metadata):

job: `weather-\${__collectible.city}`

city: `\${__collectible.city}`

The screenshot shows the Cribl Collector configuration for a collector named 'weather.json-response'. The 'Discover' section is expanded, showing a 'Discover Result' JSON object. The 'items' array contains a single object with fields: 'city' (Nashville), 'lat' (36.174465), 'lon' (86.787960), 'city' (Dallas), 'lat' (32.779167), 'lon' (-96.808893), 'city' (Denver), 'lat' (39.742043), 'lon' (-104.991531). The 'Collect' section is also expanded, showing a 'Collect URL' with a placeholder for the city name, and 'Collect parameters' with fields for 'lat', 'lon', and 'appid'.

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

The screenshot shows the Cribl Collector output preview for a collector named 'weather.json-response:1613486848.16'. The 'Filter Expression' is set to 'true'. The 'Fields' list on the left shows various metadata fields. The main area displays a preview of the collected data, showing a JSON object with fields like 'channel', 'lat', 'lon', 'city', 'time', 'base', 'clouds', 'coord', 'coord.lat', 'coord.lon', 'dt', 'host', 'id', 'job', 'main', 'main.feels_like', and 'main.grnd_level'.

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.

Using S3 Storage and Replay

Cribl LogStream's Replay options offer organizations fundamentally new ways to manage data, by providing an easy way to selectively ingest, **and re-ingest**, data into systems of analysis. Let's walk through how to use this feature, step by step.

For simplicity, we'll treat the storage destination here as Amazon S3, although it could just as easily be MinIO, or any of several other options.

Choosing JSON vs. Raw Format

You can write data out of LogStream in either of two formats – JSON or raw.

JSON

With this option, the parsed event, with all metadata and modifications it contains at the time it reaches the Destination step, will be wrapped in a JSON object. Each event is one line. This is newline-delimited JSON (abbreviated NDJSON). For example, here is syslog data using the JSON format option:

```
9    {"_raw":"<95>Aug 01 17:56:51 wintheiser6653 consec
10    {"_raw":"<4>Aug 01 17:56:51 torp3545 minus[9617]:
11    {"_raw":"<4>Aug 01 17:56:51 moen3628 ipsum[5213]:
12    {"_raw":"<27>Aug 01 17:56:51 von6811 ut[2280]: The
13    {"_raw":"<76>Aug 01 17:56:51 okuneva3386 earum[523
14    {"_raw":"<8>Aug 01 17:56:51 kling1251 sed[9036]: "
```

Raw

With this option, the contents of the event's `_raw` field – unparsed, at the time it reaches the Destination step – are written out in plain text. Each event is one line. Here's the same syslog data, unmodified and unparsed, written out with the raw option:

```
1 <95>Aug 01 17:56:51 wintheiser6653 consequu
2 <4>Aug 01 17:56:51 torp3545 minus[9617]: Us
3 <4>Aug 01 17:56:51 moen3628 ipsum[5213]: Us
4 <27>Aug 01 17:56:51 von6811 ut[2280]: The A
5 <76>Aug 01 17:56:51 okuneva3386 earum[5214]
```

Which Format?

Cribl recommends using the default JSON format. Here, timestamp extractions and other vital enhancements should all have been performed already. Reusing that information makes sense, and will make your replay simpler.

Notice that the raw screen capture above is just the original data. There are cases where maybe you want this; but usually, having the preprocessed event is more desirable.

Writing the Data Out

The Worker Nodes stage files until certain limits are reached: Time open, idle time, size, or number of files. These settings are available on the Destination configuration modal's **Advanced Settings** tab (see S3 details [here](#)).

Once any of the configured limits is reached, the Worker gzips the file and drops it into the object store. If you reach the open-file limit, the oldest file will be targeted.

The S3 Destination's settings also allow you to define how the uploaded files are partitioned. Host, time, sourcetype, source – all the metadata is available to you for this purpose. When you're replaying data from the store, these partitions will be handy, to make your replay searches faster.

We can map segments of the path back to variables (including `time`) that you can use to zero in on the exact logs you need to replay, without requiring checking `_raw` .

□ For examples of the expressions Cribl recommends, see the example [Destination definition](#) below.

Retrieving the Events

With events stored in an object store, we can now point LogStream to that store to Replay selected events back through the system. We want to use data in the file path as an initial level of filtering, to exclude as much data as we can from download. Object retrieval and unpacking imposes a big resource hit in the Replay process, so minimize your impact radius. Searching against `_raw` data is also possible, but should be secondary to `_time` , `sourcetype` , `index` , `host` , etc.

Retrieval details: The Leader picks one Node to do the discovery exploration, to find the potential objects that are in play. That list of targets is then doled out to the Worker Group to actually pull down the objects, and to examine them for content matches, before executing final delivery. All Worker Nodes share the workload of retrieving and re-injecting the data.

Finally, we need to process the data coming back to extract each event. As with any incoming data stream on a compatible Source, LogStream can use default, or custom, Event Breaker definitions. In this case, we recommended above to use JSON as the format of the events when we write to disk, so we'll use the **Cribl** Event Breaker ruleset on this Source. This Event Breaker contains a newline-delimited-JSON definition.

Setting Up and Running Replay

With the above concepts established, let's put them to work. We'll round-trip data through our Destination and replay it in LogStream.

The Store

Object storage, or any shared storage, will work. As long as all the LogStream Nodes can see it, we're ready to go. For the purposes of this post, let's stick with an S3-compatible store.

You'll need all the credentials, keys, secret keys, endpoints, etc., required to access the bucket that you intend to use in your object store. (For details on cross-account access, see [this blog post](#).) Obviously, the bucket should be able to grow to the size intended for your long-term archival needs.

⚠ LogStream Collectors and Replay features are not compatible with "deep-freeze" storage classes that have long retrieval times. This excludes the S3 Glacier and Deep Glacier storage tiers, and also excludes Azure's archive tier.

The Destination Definition

In your LogStream Worker Group config, create a new [S3 Destination](#). The screenshot below is an example built for this demo.

Use the Destination's **Advanced Settings** tab to adjust the limits, if needed. The defaults are fine for most situations, but depending on your partitioning scheme, we could be talking about 100+ files. So **make sure your staging area has enough space**.

In particular, set the **Max open files** option appropriately. It overrides the size and time limits. We recommend that the staging area be its own volume, so that you don't fill up a more-vital volume by mistake.

Output ID* ?

archival

S3 Bucket Name* ?

'logstream'

Region ?

US West (N. California)

Staging Location* ?

/opt/cribl/staging

Key Prefix* ?

C.vars.MYENV

Partitioning Expression ?

`\${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%`

Data Format ?

json

File Name Prefix Expression ?

`\${C.Time.strftime(_time ? _time : Date.now() / 1000, '%H%M')}`

Compress ?

gzip

The screen capture above includes the partitioning and filename prefix expressions. Below is the full text of each expression. In a nutshell, we're using time and other metadata to construct a path in the object store, which will be useful to us at replay time:

```
Year/Month/Day/index/host/sourcetype/HHMM-foobar.gz
```

Partitioning (one line):

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%Y/%m/%d')}/${index
```

Filename:

```
`${C.Time.strftime(_time ? _time : Date.now() / 1000, '%H%M')}`
```

We've also included a Key Prefix from LogStream's **Knowledge > Global Variables**. You could use this to partition your logs by environment, or any other qualifier. But this is optional, not a required field.

The Archival Route

Create a new LogStream Route called `Archival` that matches everything you want to archive. In this case, we've set it to `!__replayed`, and set this internal field in the Source (see the next step). Any event that is **not** coming from the defined Replay process will be archived.

Select the `passthru` Pipeline, or create an empty Pipeline and use that. Select the S3 Destination you created above. And finally, make sure **Final** is **not set**. We want data to flow through this Route, not stop at it. So, position the `Archival` Route at or near the top of the Routing table. Save your work, commit, and deploy.

9

FreeNAS Archive

!__replayed

passthru
minio:mini

Route Name*

FreeNAS Archive

Filter ?

!__replayed

Pipeline* ?

passthru

Output ?

minio:archival

Description ?

Archive all events except those coming from a replay

Final ?

☐ No

Once saved and deployed, data should be flowing to your object store. Check your work: Go to the [Monitoring](#) dashboard and select **Destinations** at the top. You want to see a green checkmark on the right side for your S3 Destination. If it's not green, find out why.

- i

You can also use an S3-capable browser, like Cyberduck, to check the files more manually. (Specific troubleshooting steps are outside the scope of this doc.)

385.85Bps320.27KB☒ Live

The Replay Collector

From the top nav of your LogStream Worker Group or single instance, select **Sources > Collectors > S3**, and create a new S3 Collector with the ID `Replay` . Use the **Auto-populate from** option to pull in the configs from your S3 Destination.

In the **Path** field, we need to establish the tokens to extract from the path on the S3 store. If you used the recommended partitioning scheme in the above example [Destination definition](#), we recommend specifying these tokens here:

```
/${MYENV}/${_time:%Y}/${_time:%m}/${_time:%d}/${index}/${host}/${$sourcety
```


If you chose to leave out `C.Vars.MYENV`, exclude the first segment. It is **vital** that this scheme match your partitioning scheme in the Destination definition.

With these tokens defined, you can now define filters that exclude and include relevant files before LogStream is required to download and open them. This cuts down on the work required, by orders of magnitude.

- This is a core concept. We want to avoid having to open files to check for matches, by instead relying on key data in the path. LogStream can immediately exclude, without downloading, the files that have no chance of matching our target events.

Using `_time`, `sourcetype`, `host`, and `index`, it can accurately zero in on the target files. After this high-level filtering, LogStream will download and interrogate the contents of what's left, and will send the matches along to the Routing table.

Finally, under **Result Settings > Event Breakers**, select the `Cribl` Ruleset. This Ruleset understands how to parse the JSON packaged events stored by the Archive Destination.

Event Breaker rulesets ?

1	Cribl	Event breaking rules for new line delimited json data (1 rule)
System Default Rule Filter Condition: <code>true</code> Event Breaker: <code>/[\n\r]+(?!s)/</code>		
Local Max Event Bytes: 51200		
+ Add ruleset		

To make it easier to identify events that have been replayed, create a field (**Result Settings > Fields**) named `__replayed` and set it to `true`. You could filter on this field in Routes and Pipelines later. Because it's a double-underscore field, it's internal-only, and won't be passed on to your final destinations. In a previous step, we used it to prevent replayed events from being re-archived:

Name	Value
__replayed	true

+ Add Field

Now save your Collector, and commit and deploy.

Testing Replay

After you've accumulated some data in your S3 store, head over to Pipelines and start a capture. For the filter, use `__replayed`, and run it for 300 seconds, 10 max events.

Once it's running, in a new browser window, navigate back to Collectors, and run your Replay Collector. Filter on `true`, and set the **Earliest** time to `-1h` and the **Latest** to `now`.

You can run it in Preview mode to make sure you get results, and then come back and do a full run. (Or you can just do a full run right off the bat if you're confident.)

With a running job, you can click on the job ID to follow its progress. You can also pop back over to the Capture browser window or tab, and you should see events there.

In a full run, the events will proceed through your Routes as normal, and will land wherever your original Routes and Pipelines dictate. In this case, they landed in Splunk, and we could easily see duplicate events – that is, **exact** duplicate events – in the 1-hour timeframe that the Collector job defined.

Once defined, this Collector can be controlled via [scheduling](#), manual runs, or API calls. And in production use, when configuring the job's **Run configuration** or **Schedule configuration** modal, you'd want to fill in the **Filter** expression to meet your needs.

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 Configuration 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.

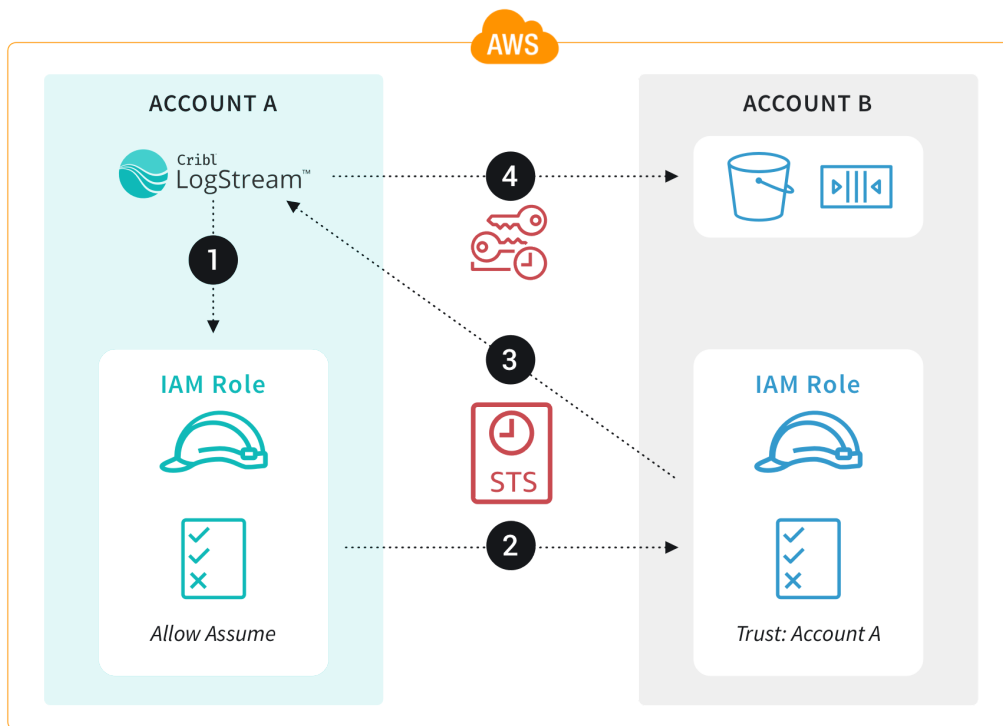
AWS Cross-Account Data Collection

Cribl uses multiple AWS accounts for different purposes, including our public [LogStream Cloud](#) service (deployment details [here](#)). We built our account strategy from the ground up using the AWS Control Tower framework, as summarized in our [Logging in a Multi-Account AWS Environment](#) blog post.

However, some organizations use a legacy account structure that doesn't consolidate logs to a single account. This creates a challenge in collecting data or logs from peer accounts, as permission boundaries now need to be crossed. Whether you need to collect data from, or write data to, another account, the `AssumeRole` permission allows for cross-account access without the need to generate static IAM keys.

AssumeRole Example

A usage example is a central logging account that has access to other organization accounts to consume logs, but not to perform other actions. Let's say we have two AWS accounts, A and B. We want account A to be able to access resources in account B. We can build a policy inside account B that allows permissions to access the target resources. We can then specify, in account B, that we trust account A to be allowed to use this role. The diagram below illustrates how the `AssumeRole` action permits the Trusted Account (A) access to resources in the Trusting Account (B).



Using STS and temporary credentials to access an S3 bucket and SQS queue across accounts

Here's how LogStream would work with `AssumeRole` permissions inside AWS:

1. The LogStream Worker has an EC2 instance role attached.
2. The IAM role in Account A permits the EC2 instance to assume the role in Account B (and Account B trusts Account A).
3. Temporary IAM credentials are returned to the EC2 instance.
4. LogStream uses the temporary IAM credentials to access the resources in Account B.

Configure IAM AssumeRole Permissions

First, in account A, we build a policy that allows only the ability to assume the role inside account B. This policy restricts users from being able to access any resources that they don't need to see. We can also revoke this trust relationship at any time, without having to worry about an account still having keys and (therefore) access to the data.

In our example, we want to access VPC Flow logs inside an S3 bucket in account B (ID 222222222222) from account A (ID 111111111111). We'll start building the two policies in account B, and then move to account A.

Account B Configuration

In account B, we build a policy to enable access to the S3 bucket (`vpc-flow-logs-for-cribl`) with the least privileges required for LogStream. This policy is the one that changes depending on what you need to accomplish – e.g., reading from an S3 bucket, writing to Kinesis Streams, etc.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::vpc-flow-logs-for-cribl/*"
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::vpc-flow-logs-for-cribl"
    }
  ]
}
```

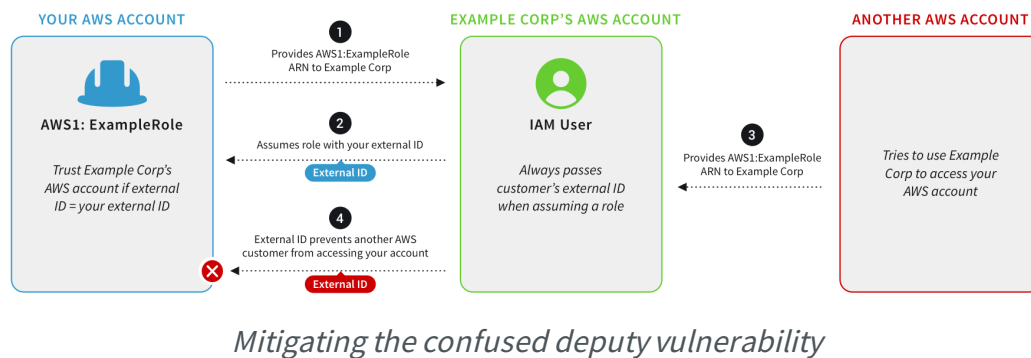
Next, we need to attach a Trust Relationship to Account B's IAM role to permit the `AssumeRole` action from account A:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:role/account-a-logstream-assumero"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "cribl-s3cre3t"
        }
      }
    }
  ]
}
```

Why an External ID Condition in the Trust Policy?

It is important to [configure an AWS External ID](#), especially if you have third parties accessing your AWS accounts. The External ID protects from the [confused deputy problem](#), where a third party obtains access through an

intermediary. The External ID is not a password or secret, but it should still be protected from accidental sharing.



Account A Configuration

In account A, we next configure a new IAM role with the following policy. For our example, we only want the role to be able to use the `AssumeRole` action, but you can add additional statements to meet your needs:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::222222222222:role/account-b-logstream-role-to"
  }
}
```

Since we need our EC2 instance to be able to assume this role, we will configure the Trust Relationship as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Configure LogStream

Now that our `AssumeRole` policies have been built, we can configure `LogStream` to assume the correct role to access the resources we need. Configure your Source, Collector, or Destination with the appropriate `AssumeRole` ARN and External ID. While this screenshot shows an S3 collector specifically, all AWS sources and destinations support Assume Role functionality.

Groups › default › Jobs › Collectors › vpc-flow-logs-for-cribl

Collector Settings

Result Settings ^

Custom Command

Event Breakers

Fields (Metadata)

Result Routing

Advanced Settings

Collector ID* ⓘ

vpc-flow-logs-for-cribl

Collector type* ⓘ

S3

Auto-populate from ⓘ

Select ▼

S3 bucket* ⓘ

'vpc-flow-logs-for-cribl'

Region ⓘ

US West (Oregon)

Path ⓘ

AWSLogs/\${aws_account_id}/vpcflowlogs/\${region}/\${_time:%Y}/\${_time:%m}/\${_time:%d}/

Path extractors ⓘ

+ Add extractor

Recursive ⓘ ☒ Yes

Max Batch Size (objects)

10

> AUTHENTICATION

▼ ASSUME ROLE

Enable Assume Role ⓘ ☒ Yes

AssumeRole ARN ⓘ

arn:aws:iam::222222222222:role/account-b-logstream-role-to-assume

External ID ⓘ

cribl-s3cre3t

> ADDITIONAL S3 SETTINGS

LogStream Assume Role configuration

OpenID + Azure AD Configuration

This page outlines how to integrate Azure Active Directory with LogStream's SSO/OpenID Connect [authentication](#).

Configure Azure AD App

Start at the Azure portal to configure an OpenID Connect provider:
<https://portal.azure.com/>.

Register Your Azure AD App

1. Open the **Azure Active Directory** Service.
2. In the left nav's **Manage** section, select **App registrations**.
3. Add a new registration. For details, see Microsoft's [Quickstart: Register an Application](#) topic.

In the example below, substitute the appropriate callback URL for your own LogStream Leader instance.

Register an application ...

*** Name**

The user-facing display name for this application (this can be changed later).

LogStream ✓

Supported account types

Who can use this application or access this API?

☒ Accounts in this organizational directory only (Cribl only - Single tenant)
☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
☐ Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web ✓ <https://leader.cribl.io:9000/api/v1/auth/authorization-code/callback> ✓

Registering an Azure AD app

Get the Azure AD App's Basic Credentials

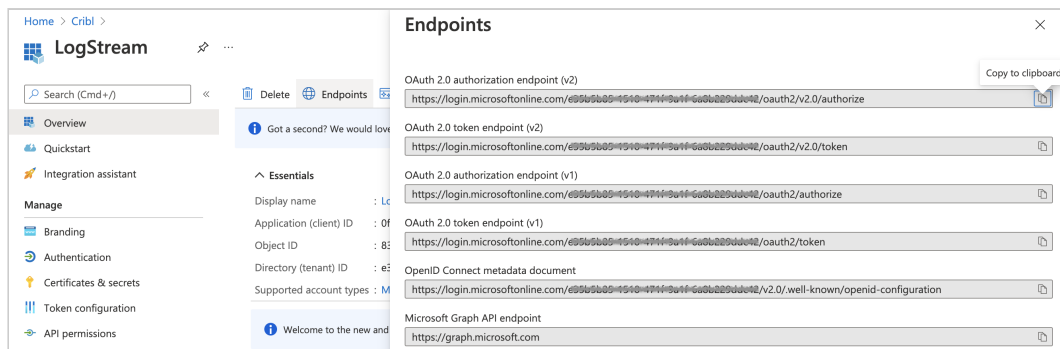
You'll need to copy and paste these credentials into LogStream's **Authentication** page [below](#).

1. You can find the OIDC Client ID on the new app's **Overview** page, as the **Application (client) ID**.

The screenshot shows the 'Overview' page for an Azure AD application named 'LogStream'. The 'Application (client) ID' is highlighted in a green box. The page also displays the 'Directory (tenant) ID' and 'Object ID'. On the right, it shows 'Supported account types' as 'My organization only', 'Redirect URIs' as '1 web, 0 spa, 0 public client', and 'Application ID URI' as 'Add an Application ID URI'. The 'Managed application in L...' is 'LogStream'.

Finding the OIDC Client ID

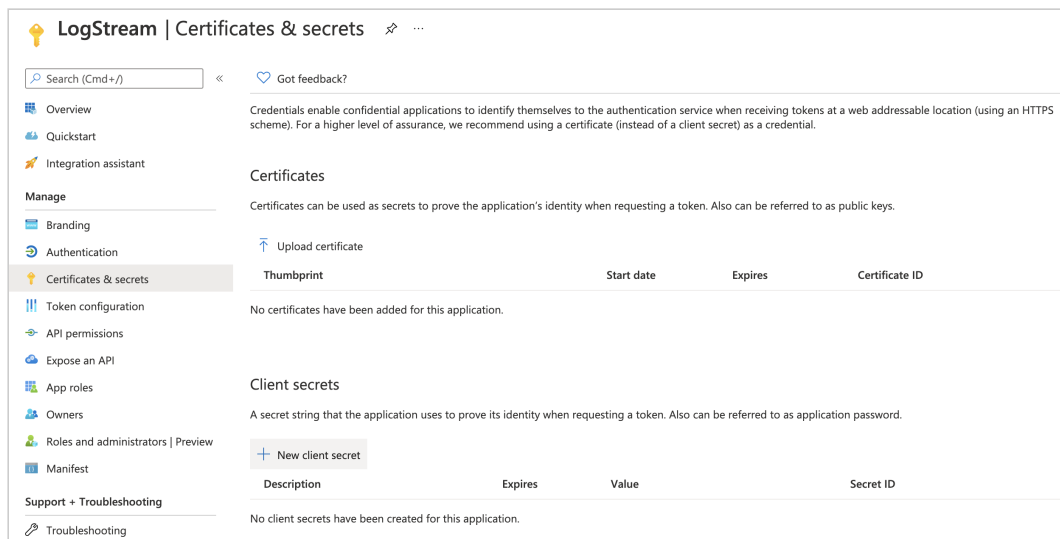
2. Click the **Endpoints** button at the page top to display the OAuth endpoints. You can use either the v2 or the v1 endpoints.



Copying OAuth 2 v2 endpoints

Create and Copy a Client Secret

1. To create a client secret: From the Azure portal's left nav, select **Certificate & secrets**. Then select **New client secret**.



Accessing client secrets

2. Add a new client secret with a descriptive name, and an expiration timeframe.

Add a client secret

Description

logstream_master

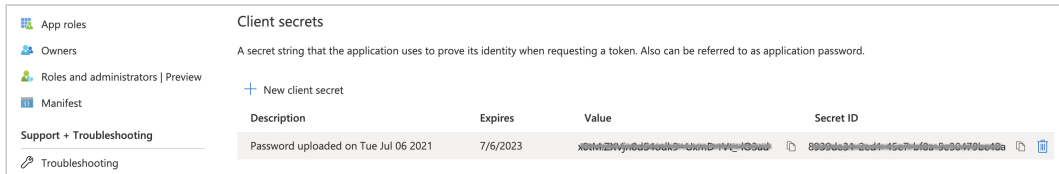
Expires

24 months

Adding a client secret

3. Click **Add**.

4. Immediately copy the **Value** and **Secret ID** from the resulting page. You'll need to paste the **Value** into LogStream's **Authentication** > **Client secret** field [below](#).



Copy that secret!

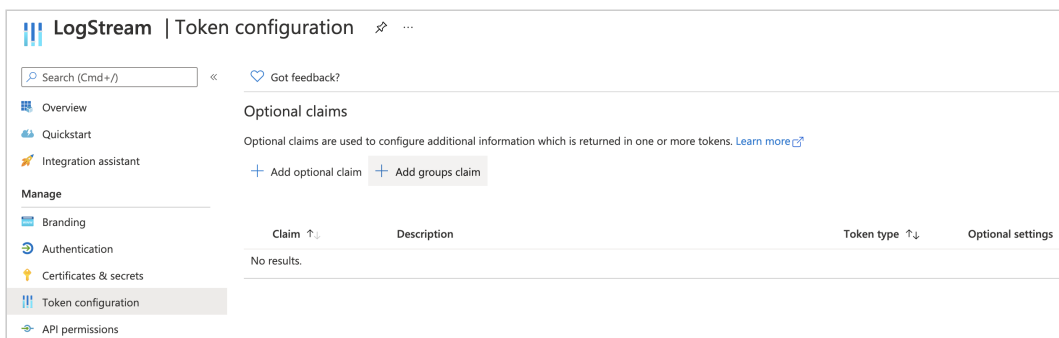


This is the only time the secret is shown! Make sure you copy it while it's visible. (If you missed your chance, you can start over by creating a new secret.)

Configure Token and Claims

Here, you'll add the groups claim to the OIDC ID token.

1. From the Azure portal's left nav, select **Token configuration**, then select **Add groups claim**.



Configuring a token

2. Configure the groups claim as necessary, then click **Add**.

Edit groups claim

Adding the groups claim applies to Access, ID, and SAML token types. [Learn more](#)

Select group types to include in Access, ID, and SAML tokens.

- ☐ Security groups
- ☐ Directory roles
- ☐ All groups (includes distribution lists but not groups assigned to the application)
- ☒ Groups assigned to the application

Customize token properties by type

- ▼ ID
- ▼ Access
- ▼ SAML

i Unless you synchronize Azure AD with your on-premises Active Directory, AD will return only GUIDs for your group names. If you've synchronized, you'll then be able to configure returning the `sAMAccountName` instead.

3. Your token is now configured, and you're all done on the Azure side.

Token configuration

«

Got feedback?

Optional claims

Optional claims are used to configure additional information which is returned in one or more tokens. [Learn more](#)

+ Add optional claim + Add groups claim

Claim ↑↓	Description	Token type ↑↓	Optional settings
groups	Optional formatting for group claims	ID, Access, SAML	Default ***

Configure LogStream Authentication

Switch to LogStream, and navigate to its global ⚙ **Settings** (lower left) > **Access Management** > **Authentication** page. Configure this as indicated below (with reactions):

- **Type: OpenID Connect.** This will expose relevant fields, setting several default values and placeholders.
- **Provider name:** Enter an arbitrary identifier for this Azure AD integration.
- **Audience:** Enter your LogStream Leader instance's base URL. Use the format: `https://<your-domain.ext>:9000`
- **Client ID:** Enter your Azure AD **Application (client) ID**. (In the Azure portal, see [above](#) to copy this from your app's **Overview** page.)
- **Client secret:** Enter the **Client secret** > **Value** that you [earlier](#) generated and copied from the Azure app's **Certificates & secrets** page.
- **Scope:** Accept the default openid profile email scopes.
- **Authentication URL:** Paste the **OAuth 2.0 authorization endpoint** that you copied [above](#) from the Azure app's **Overview** > **Endpoints** drawer.
- **Token URL:** Paste the **OAuth 2.0 token endpoint** that you copied [above](#) from the Azure app's **Overview** > **Endpoints** drawer.
- **User Info URL, Logout URL:** Leave both fields blank.
- **User identifier:** Adjust this based on the endpoint you choose (v1 or v2) [above](#). In v2, the `preferred_username`, `name`, and `email` fields are set, matching this field's default values.

In v1, only the `name` field is included in the token by default, so an acceptable entry here might be: ``${unique_name || upn || username || name}``. You can check the token fields returned by [enabling](#) debug-level logging on LogStream's `auth:sso` channel.

- Change the **Filter type** to `User info filter`.
- Optionally, enable **Allow local auth** as a fallback login method.

Authentication Settings

Type* ? OpenID Connect ▼

Provider name ? Azure AD

Audience* ? https://leader.cribl.io:9000

Client ID* ? *****

Client secret* ? 🔑

Scope ? openid profile email

Authentication URL* ? https://login.microsoftonline.com/*****/oauth2/authorize

Token URL* ? https://login.microsoftonline.com/*****/oauth2/token

User Info URL ? Enter user info url

Logout URL ? Enter logout url

User identifier ? `{preferred_username || name || email}` 🔗

Validate certs ? ☒ Yes ☐ No

Filter type ? User info filter ✕

Group name field ? groups

Allow local auth ? ☒ Yes ☐ No

User info filter ? true 🔗

Cancel Save

Sample LogStream Authentication Settings for Azure AD (v2 endpoints)

User identifier ? `{unique_name || upn || username || name}` 🔗

Validate certs ? ☒ Yes ☐ No

Filter type ? User info filter ✕

Group name field ? groups

Allow local auth ? ☒ Yes ☐ No

User info filter ? true 🔗

Sample User identifier entry for v1 endpoints

Map Azure AD Groups to LogStream Roles

Next, map your Azure AD groups to LogStream [Roles](#). The group names might appear as GUIDs. You can translate these on the Azure AD **Groups** page.

Unless you synchronize Azure AD with your on-premises Active Directory, you will need to obtain the Group GUIDs from the Azure AD **Groups** page. Place these GUIDs in the mappings box and then choose the appropriate LogStream Role. Here is a simple example.

Groups | All groups

Cribl - Azure Active Directory

All groups
Deleted groups
Diagnose and solve problems

Settings
General
Expiration

New group
Download groups
Delete
Refresh
Columns
Preview features

This page includes previews available for your evaluation. View previews →

Search groups

Add filters

Name	Object Id	Group Type	Membership Type
<input checked="" type="checkbox"/> AC All Company	83166456-7077-455a-b05d-24d007e90ede	Microsoft 365	Assigned
<input checked="" type="checkbox"/> CR Cribl	bb9e64e3-095e-4e95-b3cc-e67f99a34b20	Microsoft 365	Assigned

Azure AD groups...

ROLE MAPPING

Default role ⓘ
user

Mapping ⓘ

83166456-7077-455a-b05d-24d007e90ede	: reader_all x	X
bb9e64e3-095e-4e95-b3cc-e67f99a34b20	: admin x	X

+ Add Mapping

...mapped to LogStream Roles

SSO/Okta Configuration

If you are a LogStream admin and want to offer single sign-on (SSO) to your LogStream users, you first choose OpenID Connect as the authentication type, then choose an SSO provider for OpenID Connect. Once configuration is complete (several steps later), the LogStream login page will send users to the SSO provider login page.

The provider can be Okta or Google, among others. This page describes how to configure SSO with Okta as the provider. **SSO with Okta is only supported in LogStream versions 3.0.0 and newer.**

In Okta, admins organize their users in groups. In LogStream, there are no user groups, but there are [roles](#). Your task includes **mapping** Okta groups to LogStream roles.

- Mapping groups to roles is only possible for LogStream deployments that are in Distributed mode with an Enterprise license applied.
- If you are running LogStream in Single-instance mode, you cannot map Okta groups to LogStream roles, although you can still set up SSO with Okta.

As you think through how best to map your Okta groups to LogStream roles, keep these principles in mind:

- A LogStream role can map to more than one Okta group.
- An Okta group can map to more than one LogStream role.

The example mappings below illustrate these principles. Clearly, the groups in Mapping **b** and **c** each map to multiple roles. And both the `reader_all` and `editor_cloud` roles map to multiple groups.

Mapping	Okta Group	LogStream Role(s)
a.	Cribl Admins	admin

b.	Cloud Admins	reader_all , editor_cloud
c.	Security Team	reader_all , editor_cloud , editor_firewall

- If a user has multiple roles, LogStream applies the union of the most permissive levels of access.
- LogStream automatically assigns the `default` role to any user who has no mapped roles.

Integrate Okta with LogStream

Log in to your Okta tenant admin console.

In the left menu, select **Applications > Applications**.

Click **Create App Integration**.

- For **Sign-in method**, select `OIDC - OpenID Connect`.
- For **Application type**, select `Web Application`.

Click **Next** to open the **New Web App Integration** page.

- In the **App integration name** field, enter `Cribl LogStream`.
- (Optional) In the **Logo** field, upload the Cribl logo.
 - Use a logo from the Cribl [Media Kit](#).
- In the **Sign-in redirect URIs** field, replace the default with your Leader base URL and `/api/v1/auth/authorization-code/callback` as the path. This is the LogStream [callback API endpoint](#).
- (Optional) In the **Sign-out redirect URIs** field, append `/login` to the pre-filled path.
- In the **Assignments > Controlled access** area:
 - If all your Okta users need access to LogStream, select **Allow everyone in your organization to access**.
 - To permit specific Okta groups to access LogStream, select **Limit access to selected groups**. Then, in the field below, add the groups you want to include. After you finish creating the app, if you need to add or remove groups, do that in the **Applications > Assignments** tab.

Click **Save**.

- Okta should show an `Application Created Successfully` message.

New Web App Integration

General Settings

App integration name

Cribl LogStream

Logo (Optional) ?

Grant type

Client acting on behalf of itself

☐ Client Credentials

Client acting on behalf of a user

☒ Authorization Code☐ Refresh Token☐ Implicit (Hybrid)

Learn More

Sign-in redirect URIs

Okta sends the authentication response and ID token for the user's sign-in request to these URIs

https://leader:9000/api/v1/auth/authorization-code/callback

+ Add URI

Learn More

Sign-out redirect URIs (Optional)

After your application contacts Okta to close the user session, Okta redirects the user to one of these URIs.

https://leader:9000/login

+ Add URI

Learn More

Trusted Origins

Base URIs (Optional)

Required if you plan to self-host the Okta Sign-In Widget. With a Trusted Origin set, the Sign-In Widget can make calls to the authentication API from this domain.

+ Add URI

Learn More

Assignments

Controlled access

☐ Allow everyone in your organization to access☒ Limit access to selected groups

Cribl Admins x

Save

Cancel

Completing the new app integration in Okta

Copy your Okta App's Client ID and Client Secret

In the **Client Credentials** panel, copy both the **Client ID** and **Client Secret**, and temporarily store them locally. You will need them in the next step, when you configure LogStream.

Configure LogStream

In LogStream, select **Settings > Access Management > Authentication**.

- Choose `OpenID Connect` from the **Type** dropdown.
- Choose `Okta` from the **Provider** dropdown.
- In the **Audience** field, enter your LogStream UI base URL. Do **not** append a trailing slash.
- In the **Client ID** and **Client secret** fields, enter the respective values that you copied from the Okta UI in the previous step.
- If your LogStream is in Enterprise Distributed mode:
 - In the **Scope** field, add the scope `groups` to the default space-separated list of scopes, which is `openid profile email`.
- Obtain the authentication, token, user info, and logout URLs for your Okta app, by sending a request to the the OpenID Connect Discovery endpoint.
 - This endpoint has the URL `https://<tenant>.okta.com/.well-known/openid-configuration` where `<tenant>` is your Okta tenant name, for example `https://dev-12345678.okta.com/.well-known/openid-configuration`
 - You can view the discovery document in your web browser, or use [jq](#) to extract the needed values, as in the following example:

```
curl -s https://<tenant>.okta.com/.well-known/openid-configuration | jq '. | {"auth": (.authorization_endpoint), "token": (.token_endpoint), "userinfo": (.userinfo_endpoint), "logout": (.end_session_endpoint)}'

{
  "auth": "https://dev-416897.oktapreview.com/oauth2/v1/autho
  "token": "https://dev-416897.oktapreview.com/oauth2/v1/toke
```



```

"userinfo": "https://dev-416897.oktapreview.com/oauth2/v1/u
"logout": "https://dev-416897.oktapreview.com/oauth2/v1/log
}

```

- Populate the **Authentication URL** **Token URL** fields with the respective auth and token URLs.
- If you configured Okta to use groups, populate the **User info URL** field with the userinfo URL.
 - This is necessary because Okta does not send group information in the `id_token` passed to LogStream.
- If you want **Account > Log out** in LogStream to log the user out **globally**, populate the **Logout URL** field with the `logout` URL. This means that when a user clicks the **Accounts > Log out** link in LogStream, they are logged out of **both** LogStream and Okta.

Authentication Settings

Type* ⓘ

OpenID Connect

Provider name ⓘ

Okta

Audience* ⓘ

http://leader:9000

Client ID* ⓘ

Client secret* ⓘ

Scope ⓘ

openid profile email groups

Authentication URL* ⓘ

https://dev-416897.oktapreview.com/oauth2/default/v1/authorize

Token URL* ⓘ

https://dev-416897.oktapreview.com/oauth2/default/v1/token

User Info URL ⓘ

https://dev-416897.oktapreview.com/oauth2/default/v1/userinfo

Logout URL ⓘ

https://dev-416897.oktapreview.com/oauth2/default/v1/logout

User identifier ⓘ

`\${name} || preferred_username || email`

Validate certs ⓘ

Yes

Filter type ⓘ

User info filter

Group name field ⓘ

groups

Allow local auth ⓘ

Yes

User info filter ⓘ

true

Configuring the Response to the Okta /userinfo Endpoint

An Okta tenant's user groups can be mastered either inside Okta, outside Okta, or both.

When the /userinfo endpoint is queried, Okta returns the appropriate groups membership of the user back to LogStream:

- For groups mastered inside Okta only, the app should pass a `Filter` type groups claim to LogStream.
- For groups mastered outside Okta (e.g., Active Directory), or both inside and outside, the app should pass an `Expression` type groups claim back to LogStream.

See the Okta documentation on [dynamic allow lists](#) and using Okta [together](#) with Active Directory.

In Okta, you should still be in the panel for the app you created. If not, you can get there by opening **Applications > Applications** and selecting the app.

- For groups mastered **inside** Okta only, complete [this](#) procedure.
- For groups mastered **outside** Okta, or both inside and outside, complete [this](#) procedure.

Configuration for Groups Inside of Okta

Open the **Sign On** tab.

In the **OpenID Connect ID Token** panel:

- Click **Edit** to change the value of **Groups claim filter** to `groups` and show filter options.
- Leave **Groups claim type** set to `Filter`.
- Choose **Matches regex** from the dropdown, and enter `.*` as the regex.
- Click **Save**.

OpenID Connect ID Token

Cancel

Issuer	https://dev-410007.oktapreview.com		
Audience	okta-410007-dev-410007		
Claims	Claims for this token include all user attributes on the app profile.		
Groups claim type	<div>Filter</div>		
Groups claim filter ?	<div>groups</div>	<div>Matches regex</div>	<div>.*</div>

Using Groups Claim

Save

Cancel

Role mapping, beginning

Configuration for Groups Outside of Okta

Open the **Sign On** tab.

In the **OpenID Connect ID Token** panel:

- Click **Edit** to change the value of **Groups claim filter** to `groups` and show filter options.
- Set **Groups claim type** set to `Expression`.
- In the **input expression**, enter an expression field that matches the groups you want passed to LogStream. See the Okta documentation for more details.

- For example, to match on Active Directory groups that contain the string `okta`, use the following expression:

```
Groups.contains("active_directory", "cribl", 10)
```

- Click **Save**.

OpenID Connect ID Token

Cancel

Issuer

https://dev-110000.oktapreview.com

Audience

00000000-0000-0000-0000-000000000000

Claims

Claims for this token include all user attributes on the app profile.

Groups claim type

Expression

Groups claim expression ?

groups

Groups.contains("active_directory_group")

Using Groups Claim

Save

Cancel

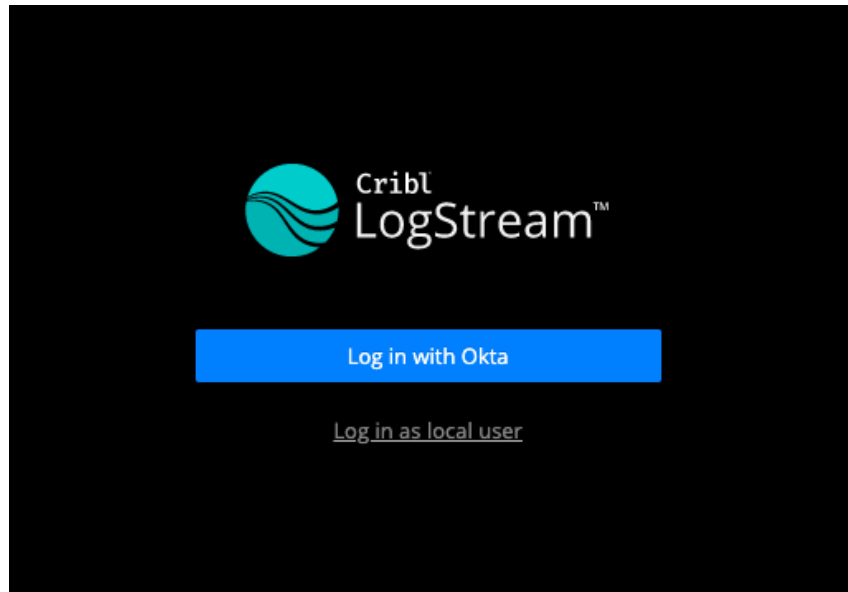
Map Okta Group Names to LogStream Roles

In LogStream, select **Settings > Access Management > Authentication**.

- Cribl recommends that you set the `default_role` to `user`, meaning that this role will be assigned to users who are not in any groups.
- Add mappings as needed.
 - The Okta group names in the left column are case sensitive and must match the values returned by Okta (the ones you saw earlier when configuring Okta).

Verify that SSO with Okta is Working

Log out of LogStream, and verify that Okta is now an option in the login screen:



Logging in with Okta

- Click **Log in with Okta**.
- You should be redirected to Okta to authenticate yourself.
- The OpenID connect flow should complete the authentication process.

Code Function Examples

The [Code Function](#) introduced in LogStream version 3.1 is a powerful way to transform events without needing to write a [custom function](#). Using JavaScript methods such as `map`, `reduce`, `forEach`, `some`, and `every` is possible.

Cribl still recommends that you use LogStream's basic, built-in Functions, like [Eval](#), as much as possible. But these use cases demonstrate some basic ways to use the new Code Function to solve questions asked in the Cribl's Slack Community.

Basic Examples

Example Event Data

The first several examples below use the following JSON object as a sample event. You can copy and paste this event into LogStream's **Sample Data** pane. Add the `Do Not Break` Ruleset to your Source, select the `noBreak1MB` Event Breaker, and under your Source's **Advanced Settings**, enable `Parse JSON Event`.

```
{
  "cpus": [
    {"number": 1, "name": "CPU1", "value": 2.3},
    {"number": 2, "name": "CPU2", "value": 3.1},
    {"number": 3, "name": "CPU3", "value": 5.1},
    {"number": 4, "name": "CPU4", "value": 1.3}
  ],
  "arch": "Intel x64"
}
```

Accessing Fields in an Event

To access a field inside an event, you can use the `__e` [special variable](#). The `__e` prefix allows for access to the (context) event inside a JavaScript

expression. For example, if you want to access the extracted field `field-name`, use the following syntax:

```
__e['field-name']
```

In other words, think of your code executing in a context like this:

```
function(__e: Event) {  
  // your code here  
}
```

Eval a Field

To create a new field, it is as simple as assigning the field to a value. For example, if you want to create a field `test` with the value `Hello, Goats!`, use the following syntax:

```
__e['test'] = 'Hello, Goats!'
```

JSON Filter

To filter the `cpus` array inside the event, you can use the `filter` function to keep only certain values, based on a logic condition. In the example below, only entries where the value is greater than or equal to `3` are kept, and placed into a new field called `cpus_filtered`.

```
__e['cpus_filtered'] = __e['cpus'].filter(entry => entry.value >= 3)
```

IN OUT	
1	# _time: 1623342662.776
2021-06-10	α arch: Intel x64
11:31:02.776	[] cpus:
-05:00	{} 0:
	α name: CPU1
	# number: 1
	# value: 2.3
	{} 1:
	α name: CPU2
	# number: 2
	# value: 3.1
	{} 2:
	α name: CPU3
	# number: 3
	# value: 5.1
	{} 3:
	α name: CPU4
	# number: 4
	# value: 1.3
	[] cpus_filtered:
	{} 0:
	α name: CPU2
	# number: 2
	# value: 3.1
	{} 1:
	α name: CPU3
	# number: 3
	# value: 5.1

Code Function example: JSON Filter

Reduce

The `reduce` method allows you to summarize data across an array, with a returned accumulator value. In the example below, a new field, `cpus_reduce`, will be created with a value of `11.8`.

```
__e['cpus_reduce'] = __e['cpus'].reduce((accumulator, entry) => accumulat
```


IN OUT	
1	# _time: 1623342662.776
2021-06-10	α arch: Intel x64
11:31:02.776	[] cpus:
-05:00	{} 0:
	α name: CPU1
	# number: 1
	# value: 2.3
	{} 1:
	α name: CPU2
	# number: 2
	# value: 3.1
	{} 2:
	α name: CPU3
	# number: 3
	# value: 5.1
	{} 3:
	α name: CPU4
	# number: 4
	# value: 1.3
	# cpus_reduce: 11.8

Code Function example: Reduce

Some/Every

The `some` and `every` methods return a boolean result (`true` / `false`).

The `some` method checks that there is at least one logic validation that returns `true` . In the example below, `cpus_some` would be set to `true` because there is at least one object with a value greater than or equal to `3` .

```
__e['cpus_some'] = __e['cpus'].some(entry => entry.value >= 3)
```

The `every` method checks that every entry in the array returns `true` . If so, the result is `true` ; otherwise, this returns `false` . In the example below, the value for `cpus_every` would be `false` , because not all values in the event are greater than or equal to `10` .

```
__e['cpus_every'] = __e['cpus'].every(entry => entry.value >= 10)
```

IN OUT	
1	# _time: 1623342662.776
2021-06-10	α arch: Intel x64
11:31:02.776	[] cpus:
-05:00	{} [] 0:
	α name: CPU1
	# number: 1
	# value: 2.3
	{} [] 1:
	α name: CPU2
	# number: 2
	# value: 3.1
	{} [] 2:
	α name: CPU3
	# number: 3
	# value: 5.1
	{} [] 3:
	α name: CPU4
	# number: 4
	# value: 1.3
	b cpus_every: false
	b cpus_some: true

Code Function example: Some/Every

Advanced Examples

Transform a Specific Field

In this example, each `cpus` member will have its `name` field transformed to lowercase. [Object.assign](#) is used to keep the original object, while assigning the `name` field to the desired value.

```
__e['cpus'] = __e['cpus'].map(entry => Object.assign(entry, {'name': entry
```

IN		OUT
1		#_time: 1623342662.776
2021-06-10		α arch: Intel x64
11:31:02.776		[] cpus:
-05:00		{} 0:
		α name: cpu1
		# number: 1
		# value: 2.3
		{} 1:
		α name: cpu2
		# number: 2
		# value: 3.1
		{} 2:
		α name: cpu3
		# number: 3
		# value: 5.1
		{} 3:
		α name: cpu4
		# number: 4
		# value: 1.3
		{} 4:
		α name: CPU1
		# number: 1
		# value: 2.3
		{} 5:
		α name: CPU2
		# number: 2
		# value: 3.1
		{} 6:
		α name: CPU3
		# number: 3
		# value: 5.1
		{} 7:
		α name: CPU4
		# number: 4
		# value: 1.3

Code Function example: Transform a Field

forEach

The `forEach` method loops over each element of an array. However, unlike the `map` method, it does not return the value, and it requires a separate temporary variable for result collection.

```
let test = {};
__e['cpus'].forEach((entry, index) => test[entry.name] = entry.value);
__e['cpus_foreach'] = test;
```

IN OUT	
1	# _time: 1623342662.776
2021-06-10	α arch: Intel x64
11:31:02.776	[] cpus:
-05:00	{} 0:
	α name: CPU1
	# number: 1
	# value: 2.3
	{} 1:
	α name: CPU2
	# number: 2
	# value: 3.1
	{} 2:
	α name: CPU3
	# number: 3
	# value: 5.1
	{} 3:
	α name: CPU4
	# number: 4
	# value: 1.3
	{} cpus_foreach:
	# CPU1: 2.3
	# CPU2: 3.1
	# CPU3: 5.1
	# CPU4: 1.3

Code Function example: forEach

You could also accomplish the same result by replacing the middle line above with the `map` method below:

```
__e['cpus'].map(item => test[item.name] = item.value);
```

Building a New Array

In this example, we create a new array to include some of the original values from each object in the `cpus` array, but we also dynamically inject a new field containing the `arch` field (CPU architecture) from the original event's top level.

```
__e['cpus_new'] = __e['cpus'].map(entry => {
  const container = {};

  for (const [key, value] of Object.entries(entry)) {
    container[key] = value;
  }
});
```

```

    }

    container['arch'] = __e['arch'];

    delete container.name;
    return container;
  })

```



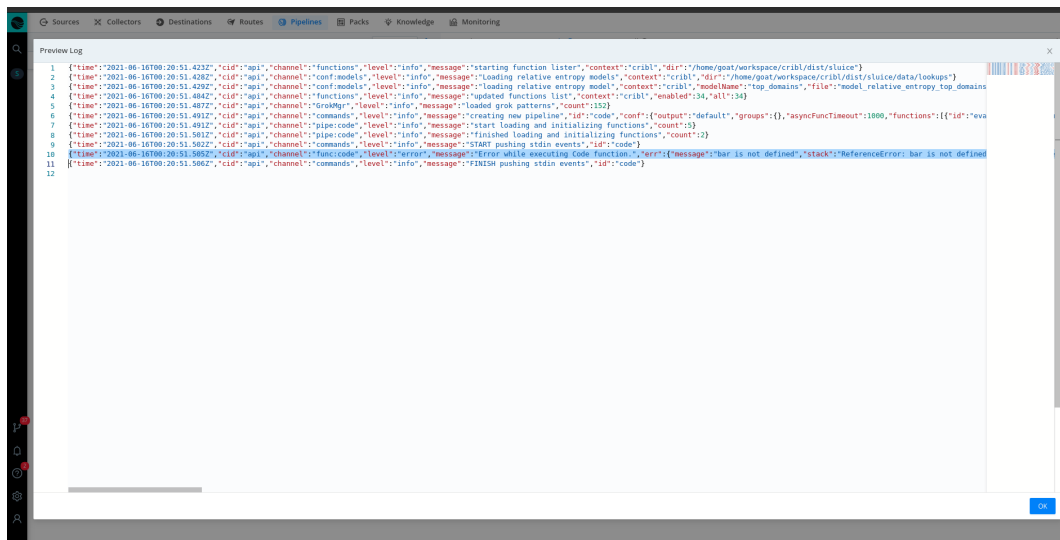
Code Function example: Build a new array

Managing and Troubleshooting Code Execution

LogStream provides the following options for tuning the logic you build with the Code Function.

Logging

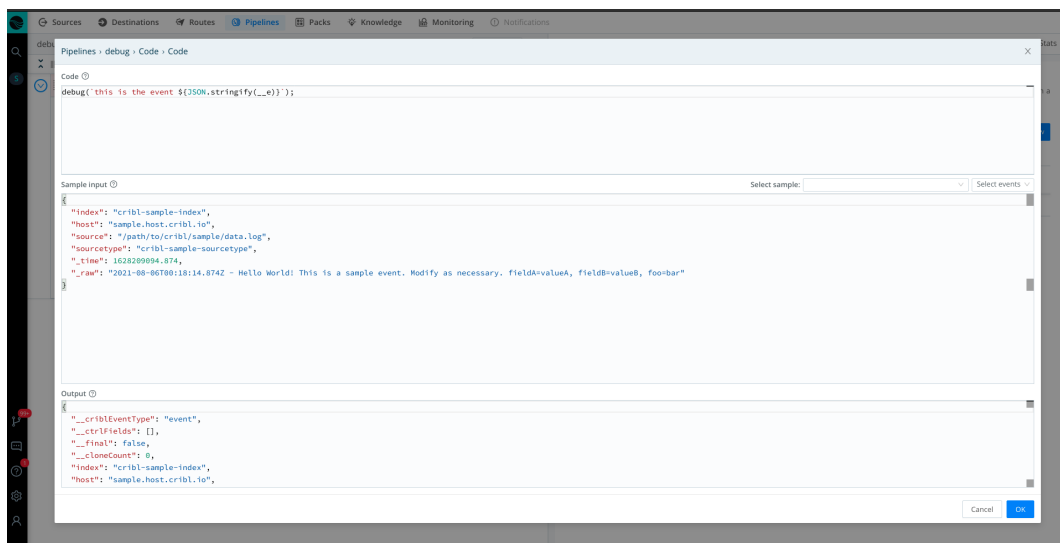
Users can access the log messages generated by their Function from the **Preview Log**.



Log messages from a Code Function

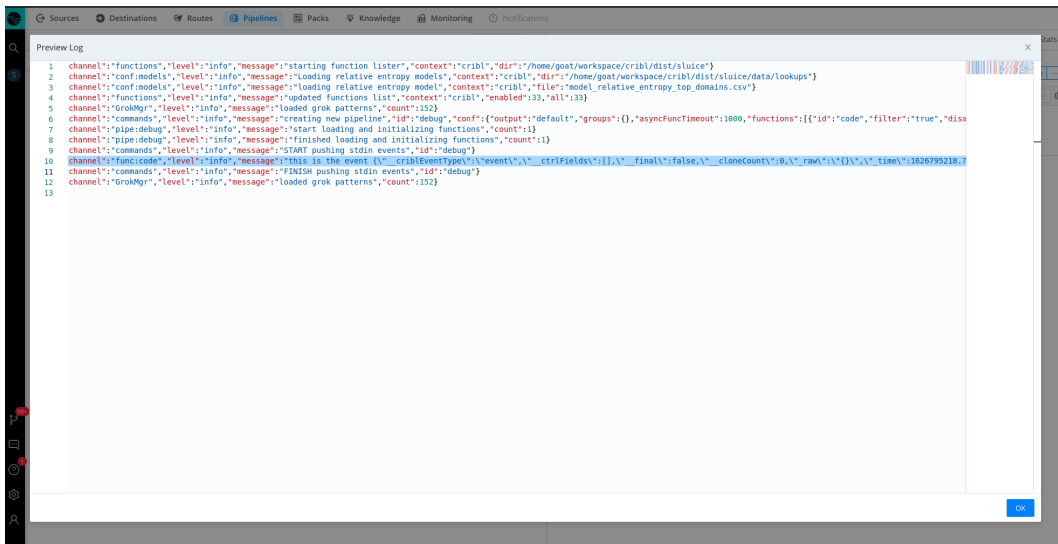
Debugging

The Code Function's execution context defines a helper function called `debug` that can be used for debugging purposes.



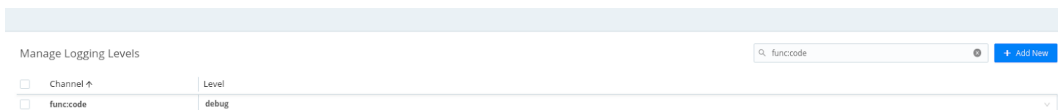
Debugging a Code Function

Messages logged by this `debug` helper function are shown in the Preview Log by default.



Debugging a Code Function with the Preview Log

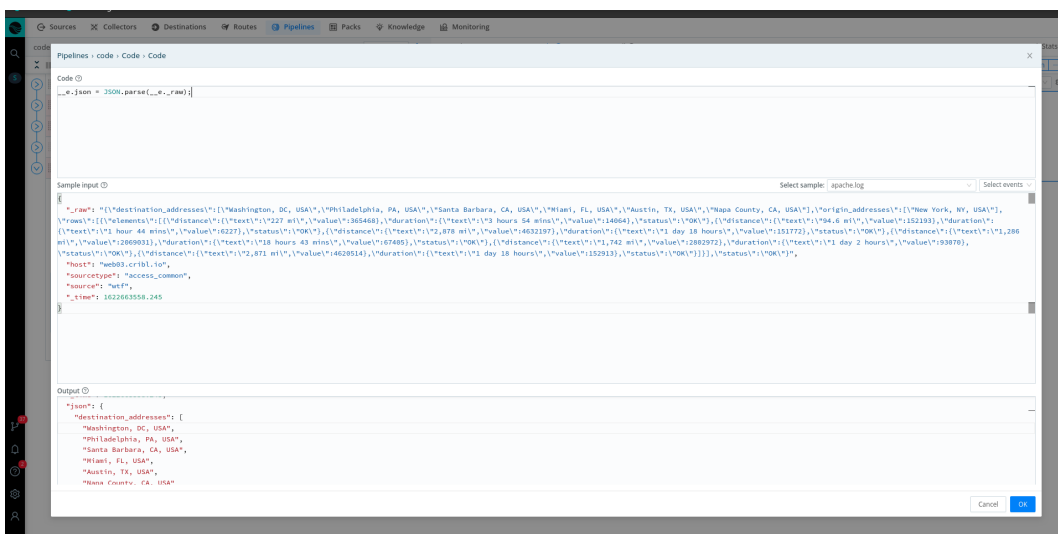
You can also route these messages to regular logs, although this requires setting the Function's log level (`func:code`) to `debug` .



Debugging a Code Function with regular logs

Previewing

By using the expanded editor, users can run the expression against a sample event, and can preview the transformation:



Previewing a Code Function's transformation of an event

Infinite-Loop Protection

The Code Function keeps watching user-defined functions to detect infinite loops that would cause processing to hang. To limit the number of iterations allowed per instance of your Code Function, adjust the **Advanced Settings > Maximum number of iterations** option. This defaults to 5,000 ; the maximum number allowed is 10,000 . Once the limit is reached, the Code Function will stop processing whatever is after the statement that exhausted the allowed maximum.

▼ ADVANCED SETTINGS

Maximum number of iterations. ⓘ

5000

Infinite-Loop protection in a Code Function

Loops

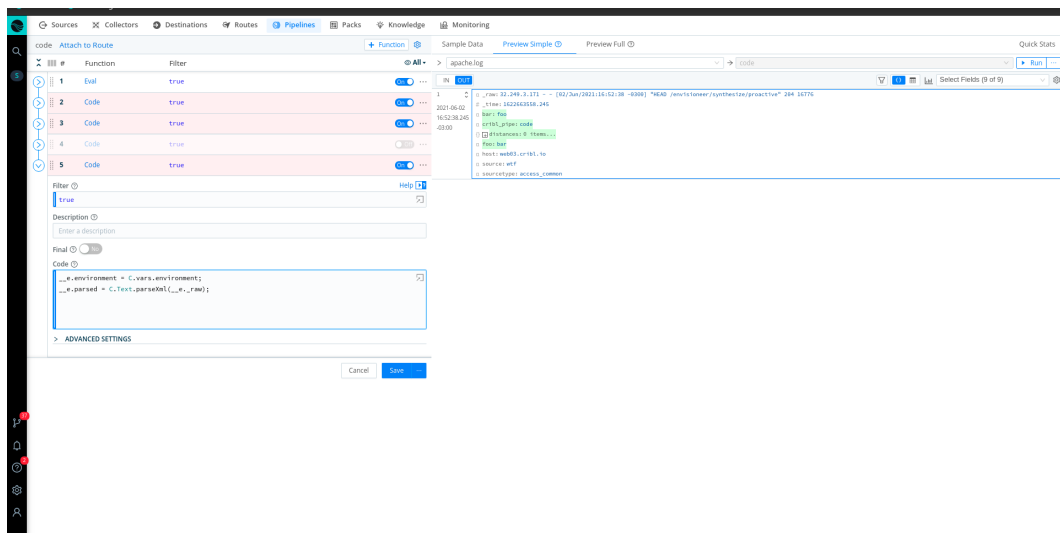
All JavaScript loops and statements are allowed: `for` , `for-of` , `while` , `do-while` , `switch` , etc.

Functions

Users can define their own functions to better organize their code. Both traditional and arrow functions are allowed.

Access to C.* Global Object

From the Code's Function body, you can access LogStream's global C.* object and its [methods/expressions](#).



The global C. object of a Code Function*

Setup Guides

The following topics provide expanded configuration options for selected LogStream integrations ([Collectors](#), [Sources](#), [Destinations](#), and [Notifications](#)):

- [Azure Event Hubs Integrations](#)
- [Splunk Cloud and BYOL Integrations](#)
- [Webhook/BigPanda Integration](#)
- [Webhook/Sumo Logic Integration](#)

Azure Event Hubs Integrations

You can create an Azure Event Hub which sends data to a LogStream Azure Event Hubs Source.

Your Azure Event Hubs account must be at the Standard (or a higher) pricing tier, because the Basic pricing tier does not support the [PLAIN authentication method](#) LogStream uses for Event Hubs.

Prepare Azure Event Hubs to Send Data to LogStream

First, create an Azure Event Hub as described in the [Azure documentation](#).

For purposes of this tutorial, we assume that in the **Create Namespace** page, you will use `CriblTest` as your **Namespace name**.

Create Namespace ...

Event Hubs

Basics Tags Review + create

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource group *

[Create new](#)

INSTANCE DETAILS

Enter required settings for this namespace, including a price tier and configuring the number of units (capacity).

Namespace name * .servicebus.windows.net

Location *

i The region selected supports Availability zones. Your namespace will have Availability Zones enabled. [Learn more.](#)

Pricing tier ([View full pricing details](#)) *

Throughput Units *

Enable Auto-Inflate ☐

Creating an Event Hubs Namespace

Collect the information you will need when configuring LogStream:

1. In the **Deployment** page, click **Go to Resource**.
2. Write down the **Host Name**.

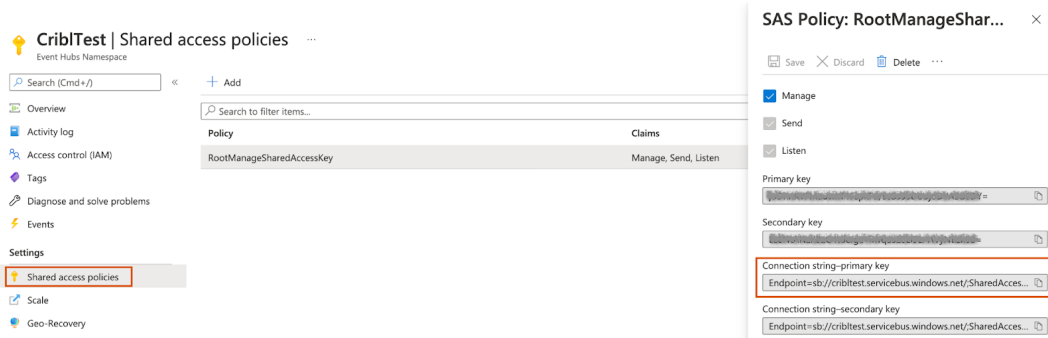
The screenshot shows the 'CriblTest' Event Hubs Namespace page in the Azure portal. The left sidebar has 'Shared access policies' highlighted. The main content area shows the 'Essentials' tab with the following details:

Property	Value
Resource group (change)	Test-Environment
Status	Active
Location	East US
Subscription (change)	Azure subscription 1
Subscription ID	73a631f2-229a-44b6-9343-4fd32ddc384a
Host name	CriblTest.servicebus.windows.net
Tags (change)	Click here to add tags

Below the essentials, there are three status indicators: NAMESPACE CONTENTS (1 EVENT HUB), DATA SURFACE (ENABLED), and ZONE REDUNDANCY (ENABLED). At the bottom, there is a 'Show data for the last:' dropdown menu with options: 1 hour, 6 hours, 12 hours, 1 day, 7 days, and 30 days.

Finding the Host name and Shared access policies

3. Click **Shared Access Policies** to open the page where you can select policies for your Event Hubs Namespace, and then click `RootManageSharedAccessKey` to show details for that policy.



Viewing Shared Access Policies

4. Copy and securely store the **Connection String-primary key**.

Configure the Azure Event Hubs Source in LogStream

1. From the top nav of a LogStream instance or Group, select **Sources**, then select **Azure Event Hubs** from the **Pull** page's tiles or the **Sources** left nav. Click **+ Add New** to open the **Azure Event Hubs > New Source** modal.
2. In the **General Settings** tab, enter the following values:
 - **InputId:** LogStream .
 - **Brokers:** To the Host Name you wrote down earlier, append port 9093, and enter the result. For example:
`CriblTest.servicebus.windows.net:9093` .
 - **Event Hub Name:** The name of your Azure Event Hub, for example: `CriblTest` . This is equivalent to a [Kafka topic](#).
 - **Group ID:** Keep (or change, if desired) the default value (`Cribl`).

Sources > Azure Event Hubs > New Source

General Settings
TLS Settings (Client Side)
Authentication
Processing Settings
Fields (Metadata)
Pre-Processing
Advanced Settings

Input ID* ⓘ
LogStream
__inputId=='eventhub:LogStream' ⓘ
Brokers* ⓘ
CriblTest.servicebus.windows.net:9093
+ Add Broker
Event Hub name* ⓘ
CriblTest ×
Group ID ⓘ
Cribl
From beginning ⓘ ☒ Yes

The General Settings tab

6. In the **Authentication** tab, enter or select the following values:

- **SASL mechanism:** PLAIN (the only supported option).
- **Username:** \$ConnectionString (the default generated by Azure).
- **Authentication Method:** Select **Manual** to use the Connection String Key generated by Azure Event Hubs.
- **Password:** Enter the **Connection String-primary key** that you [copied](#) earlier.

Sources > Azure Event Hubs > LogStream

Configure
Status
Charts
Live Data
Logs
Help ⓘ

General Settings
TLS Settings (Client Side)
Authentication
Processing Settings
Fields (Metadata)
Pre-Processing
Advanced Settings

Enabled ⓘ ☒ Yes
SASL mechanism* ⓘ
PLAIN
Username* ⓘ
\$ConnectionString
Authentication method ⓘ
Manual ⓘ Secret ⓘ
Password* ⓘ
Endpoint-sb://cribltest.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=

The Authentication tab

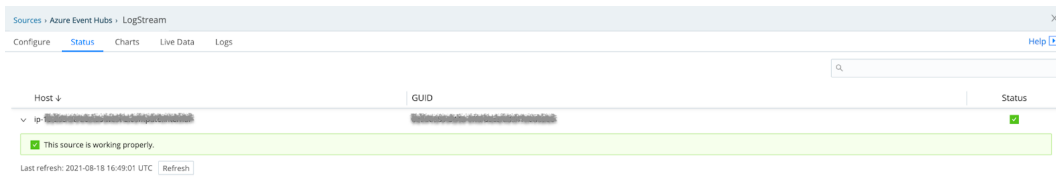
Verify that Data is Flowing from your Azure Event Hub to LogStream

Before you can verify that data is reaching LogStream, you must ensure that it is flowing out of your Azure Event Hub in the first place.

One option is to configure a [Datagen](#) and a [Route](#) to send data to the Event Hub destination. We'll assume you have done that, or gotten data flowing from your Azure Event Hub in some other way.

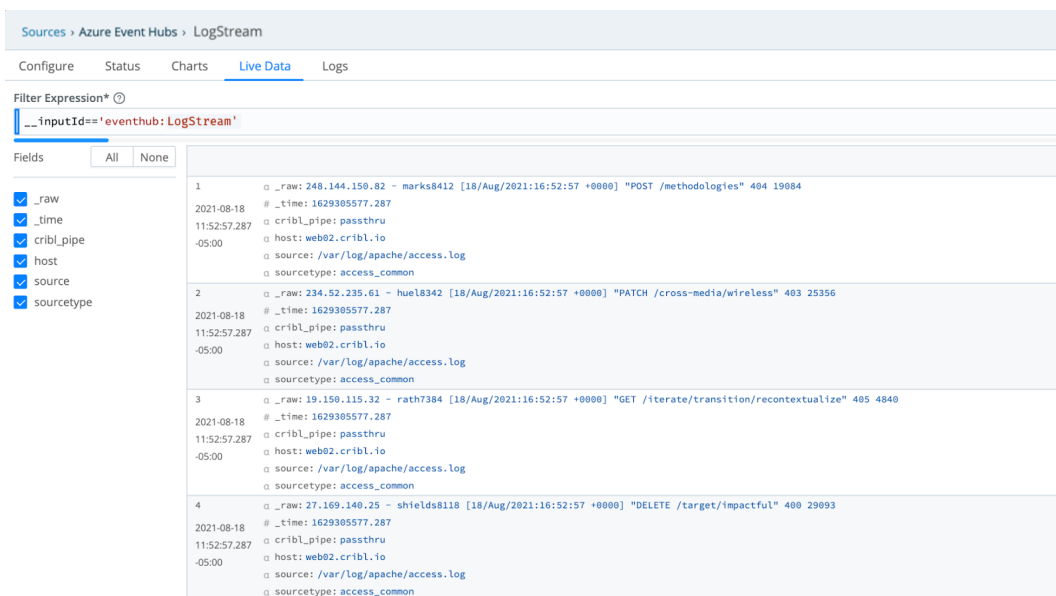
1. In LogStream, open the **Sources > Azure Event Hubs > LogStream** page.

This should show your Source, with a message confirming that it is working properly.



A working Source

2. Open the **Live Data** tab. You should see the data that is flowing from your Azure Event Hub to LogStream.



Viewing Live Data

Splunk Cloud and BYOL Integrations

LogStream can send data to these flavors of Splunk Cloud:

- The free, single-instance trial version.
- A distributed Splunk Cloud instance with clustered indexers.
- A Bring Your Own License (BYOL) deployment, either in a non-Splunk cloud or on-prem.

You have a choice of two methods for sending the data:

- [Splunk HEC](#) (HTTP Event Collector).
- The S2S (Splunk-to-Splunk) protocol.

Of all the possible combinations, three have proven most useful in the field:

- Using Splunk HEC with the trial version of Splunk.
- Using S2S with a distributed instance of Splunk.
- Using S2S with a BYOL deployment of Splunk.

When to Use Splunk HEC

Splunk HEC is fast and easy to set up. Under the hood, it uses the HTTP/S protocol. This offers better compression than S2S, which is a binary protocol.

The Splunk HEC endpoints are virtual endpoints, front-ended with load balancers – ELB for AWS, or GLB for GCP. This provides good load-balancing.

Cribl generally recommends using Splunk HEC for integrating with Splunk Cloud, because (1) it requires fewer connections than S2S, and therefore consumes less memory; and (2) because its superior compression yields lower egress costs.

When to Use S2S

S2S allows each LogStream Worker Process to connect to multiple indexers concurrently, which distributes data very effectively. This helps significantly with Splunk search, by placing a smaller burden on a larger number of indexers. This support for concurrent connections is the main advantage of S2S. Consider S2S if you plan to route all your data through LogStream first, and you prioritize search performance.

S2S enables TLS compression by default. Do not confuse TLS compression with the `compressed` setting in the Splunk `inputs.conf` file, which is a different thing, and is for non-TLS connections only.

See the Splunk documentation about the `compressed` [setting](#), and about TLS, which Splunk configuration files still [refer to](#) as SSL.

Using Splunk HEC

Identify Your Splunk HEC Endpoint

In Splunk Cloud, identify your HEC endpoint, as described in the Splunk [documentation](#). Here are some example URL patterns for HEC endpoints:

- Free version:
`https://inputs.<cloud_stack_name>:8088`
- Paid Version in AWS:
`https://http-inputs-<cloud_stack_name>:443`
- Paid version in GCP:
`https://http-inputs.<cloud_stack_name>:443`

A HEC endpoint for a paid version of Splunk Cloud on AWS, for a company called "Acme Group," might look like this:

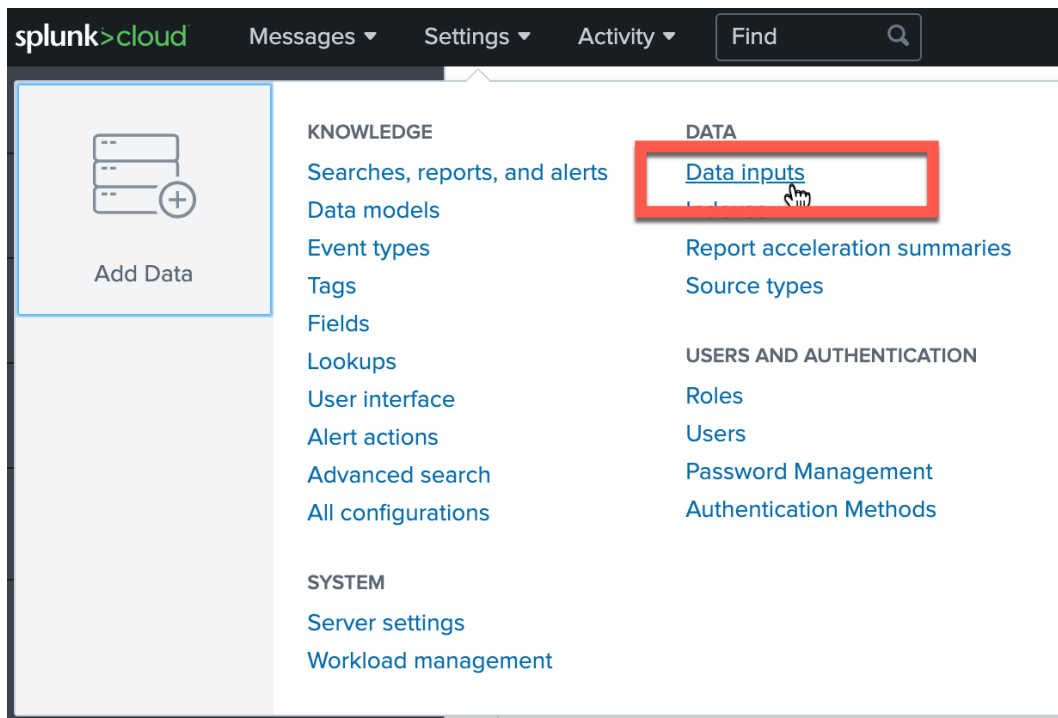
```
https://http-inputs-acmegroup.splunkcloud.com:443
```

Copy the endpoint URL for use when configuring LogStream in the next section.

Create HEC Tokens

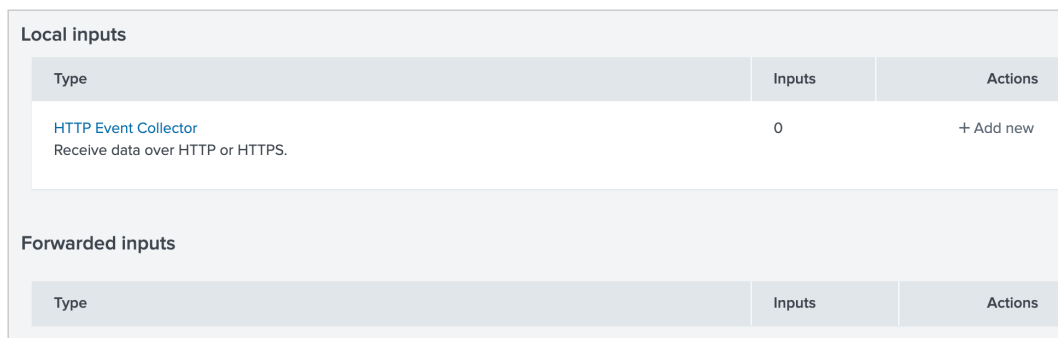
You need to create at least one HEC token. For deployments where you set up routing to individual indexes, or you use HEC tokens for RBAC on Splunk, you will create multiple HEC tokens.

1. In the Splunk UI, open the **Settings** menu and click **Data Inputs**.



Settings > Data inputs

2. In the **HTTP Event Collector** section of the resulting modal, click **+ Add new**.



Adding a new HTTP Event Collector

3. Name the new token and click **Next**.

Add Data

Select Source Input Settings Review Done

< Back Next >

HTTP Event Collector

Configure tokens that clients can use to send data over HTTP or HTTPS.

Configure a new token for receiving data over HTTP. [Learn More](#)

Name

Source name override ?

Description ?

Enable indexer acknowledgement ☐

Adding a new token

- Do not add any indexes. This way HEC can write to any index. If you prefer a default index other than `main`, choose it from the **Default index** drop-down.

Add Data

Select Source Input Settings Review Done

< Back Review >

Input Settings

Optionally set additional input parameters for this data input as follows:

Source type

The source type is one of the default fields that the Splunk platform assigns to all incoming data. It tells the Splunk platform what kind of data you've got, so that the Splunk platform can format the data intelligently during indexing. And it's a way to categorize your data, so that you can search it easily.

Automatic Select New

App context

Application contexts are folders within a Splunk platform instance that contain configurations for a specific use case or domain of data. App contexts improve manageability of input and source type definitions. The Splunk platform loads all app contexts based on precedence rules. [Learn More](#)

App Context

Index

The Splunk platform stores incoming data as events in the selected index. Consider using a "sandbox" index as a destination if you have problems determining a source type for your data. A sandbox index lets you troubleshoot your configuration without impacting production indexes. You can always change this setting later. [Learn More](#)

Select Allowed Indexes

Available item(s)

- history
- lastchanceindex
- main
- summary

add all >

Selected item(s) < remove all

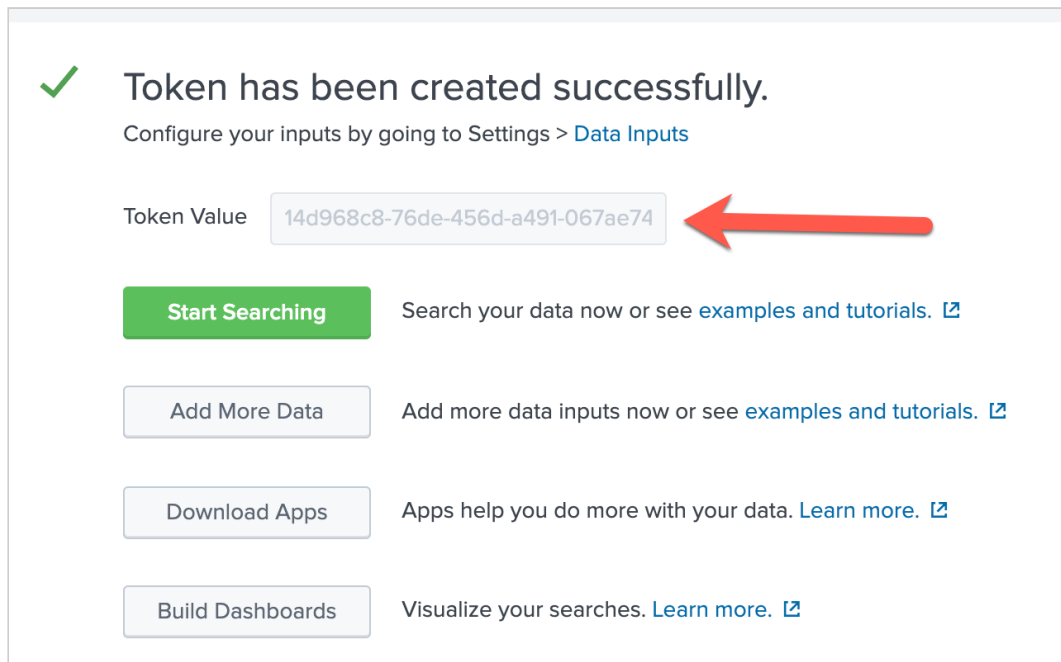
Select indexes that clients will be able to select from.

Default Index [Create a new index](#)

Leave empty to allow writing to any index

Indexes for the new token

5. Once the token has been created, copy it for use when configuring LogStream in the next section.



Copying the token value

Add a Splunk HEC Destination in LogStream

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.

In the **General Settings** tab:

- Grab the values that you copied in the previous section, and paste them into the **Splunk HEC Endpoint** and **HEC Auth Token** fields, respectively. Be sure to specify HTTPS, because the endpoint will default to HTTP.
- Click **Save**.
- Click **Commit & Deploy**.

Groups > default > Destinations > HEC > SplunkCloudHEC

Configure Status Charts Live Data Logs Test

General Settings

Processing Settings ^

Post-Processing

Advanced Settings

Output ID* ⓘ

SplunkCloudHEC

Splunk HEC Endpoint* ⓘ

https://inputs.prd-p-y8yoy.splunkcloud.com:8088/services/collector/event

HEC Auth Token* ⓘ

14d968c8-76de-456d-a491-067ae7470d0b

Backpressure behavior ⓘ

Block

Free Splunk Cloud HEC endpoint

HEC Auth token created in prior step

Populating General Settings with values from Splunk

Verify that Data is Flowing from LogStream to Splunk Cloud

- Be sure you have committed and deployed the newly created configuration. Otherwise, data will not flow to Splunk Cloud, and verification will fail.

1. In LogStream, open the Splunk HEC Destination that you created in the previous section.

- In the configuration modal's **Test** tab, click **Run Test**.
- You should see a **Success** message.

Groups > default > Destinations > HEC > SplunkCloudHEC

Configure Status Charts Live Data Logs **Test** ⓘ

Select worker: ip-10-255-12-93.us-west-2.compute.inter... Run Test

Select sample: Select events

Test input ⓘ

```
{
  "_raw": "178.146.234.131 - Qiv41020 [03/Jun/2021:04:29:02 +0000] \"GET /efficient/wireless/relationships/deliver/\" 204 2338",
  "host": "web03.cribl.io",
  "sourcetype": "access_common",
  "source": "/var/log/apache/access.log",
  "_time": 1622694542.68,
  "cribl_test": "cribl-SplunkCloudHEC"
},
{
  "_raw": "247.43.114.229 - Qiv41020 [03/Jun/2021:04:29:02 +0000] \"DELETE /embrace/portals/\" 502 12637",
  "_time": 1622694542.68,
  "cribl_test": "cribl-SplunkCloudHEC"
},
{
  "_raw": "67.207.231.248 - Qiv41020 [03/Jun/2021:04:29:02 +0000] \"PUT /synergistic/\" 503 9386",
  "_time": 1622694542.68,
  "cribl_test": "cribl-SplunkCloudHEC"
},
{
  "_raw": "14.86.68.219 - Qiv41020 [03/Jun/2021:04:29:02 +0000] \"DELETE /paradigms/reinvent/optimize/action-items/\" 405 25172",
  "_time": 1622694542.68,
  "cribl_test": "cribl-SplunkCloudHEC"
}
```

Test Results ⓘ Last run: 2021-06-03 04:29:04 UTC

✓ Success

Testing the HEC Destination

2. In Splunk, search on `index=main cribl_pipe=*` . Events that you sent from the LogStream **Test** tab should appear in the search results.

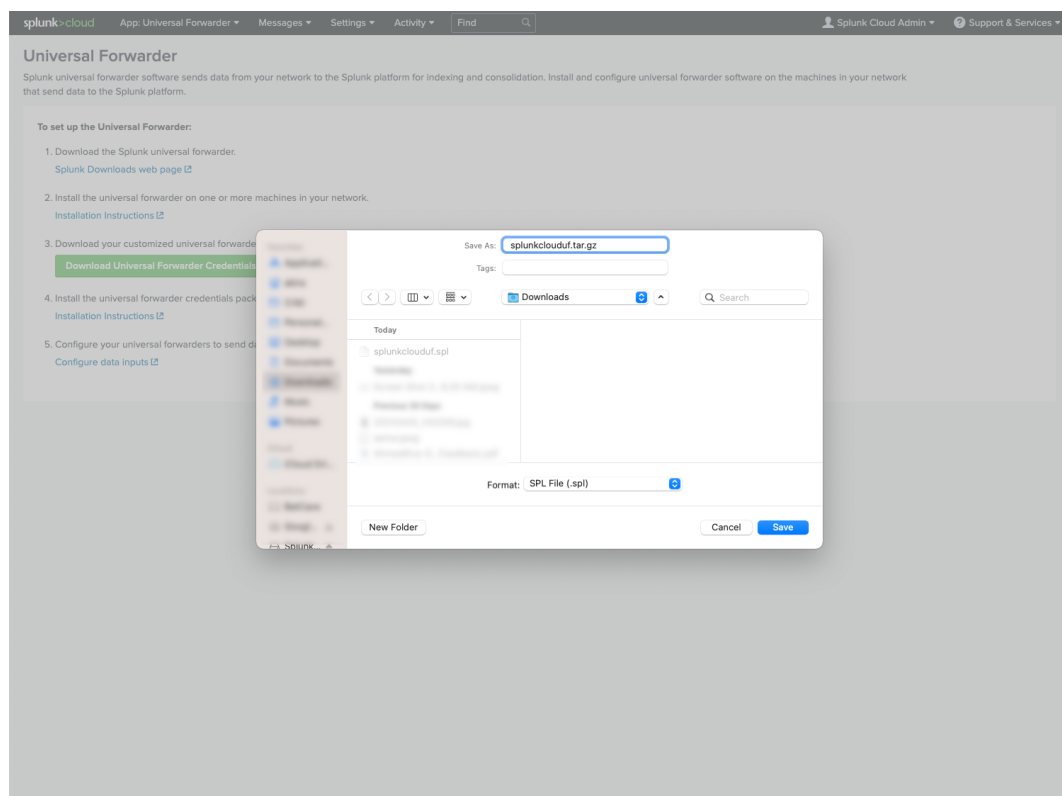
i	Time	Event
>	6/3/21 4:29:02.680 AM	206.249.78.234 - Qiv41020 [03/Jun/2021:04:29:02 +0000] "GET /e-enable/out-of-the-box" 204 895 host = web01.cribl.io source = /var/log/apache/access.log sourcetype = access_common
>	6/3/21 4:29:02.680 AM	14.86.68.219 - Qiv41020 [03/Jun/2021:04:29:02 +0000] "DELETE /paradigms/reinvent/optimize/action-items" 405 25172 host = inputs.prd-p-y8yoy.splunkcloud.com:8088 source = http:Cribl sourcetype = httpevent
>	6/3/21 4:29:02.680 AM	67.207.231.248 - Qiv41020 [03/Jun/2021:04:29:02 +0000] "PUT /synergistic" 503 9386 host = inputs.prd-p-y8yoy.splunkcloud.com:8088 source = http:Cribl sourcetype = httpevent
>	6/3/21 4:29:02.680 AM	247.43.114.229 - Qiv41020 [03/Jun/2021:04:29:02 +0000] "DELETE /embrace/portals" 502 12637 host = inputs.prd-p-y8yoy.splunkcloud.com:8088 source = http:Cribl sourcetype = httpevent
>	6/3/21 4:29:02.680 AM	178.146.234.131 - Qiv41020 [03/Jun/2021:04:29:02 +0000] "GET /efficient/wireless/relationships/deliver" 204 2338 host = web03.cribl.io source = /var/log/apache/access.log sourcetype = access_common

Events flowing to Splunk

Using S2S

Prepare Splunk Cloud for LogStream Integration

1. In Splunk Cloud, download the Splunk Cloud Universal Forwarder credentials app to your desktop.
2. Change the file suffix from `.spl` to `.tar.gz` .



Changing the credentials app file suffix

3. Untar/unzip the directory to expose the files.

100_prd-p-y8yoy_splunkcloud	--	Folder
default	--	Folder
limits.conf	235 bytes	Document
server.conf	97 bytes	Document
prd-p-y8yoy_cacert.pem	4 KB	printabl...archive
prd-p-y8yoy_server.pem	7 KB	printabl...archive
outputs.conf	417 bytes	Document
local	--	Folder
outputs.conf	142 bytes	Sublim...cument
splunkclouduf.tar.gz	6 KB	gzip co...archive
splunkclouduf.spl	6 KB	Document

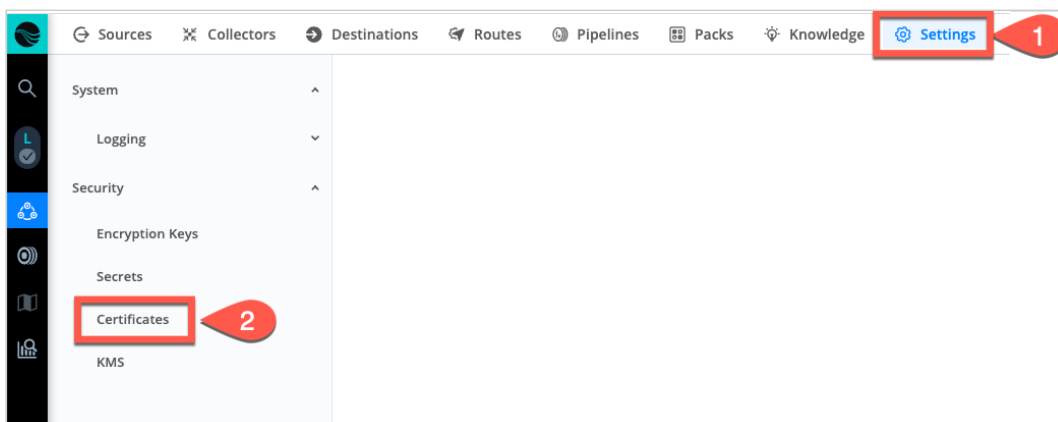
Credentials app files

4. Locate the following files. You will need them when you configure LogStream in the next section.

- ./default/<SplunkCloudInstanceName>_cacert.pem
- ./default/<SplunkCloudInstanceName>_server.pem
- ./default/outputs.conf
- ./local/outputs.conf

Configure Certificate Settings in LogStream

1. In LogStream, select **Groups** > <group-name> (or **Configure** > default) from the left nav. Then, at the upper right, select **Settings** > **Certificates**.



The Settings > Certificates submenu

2. Populate each field below with the specified content:

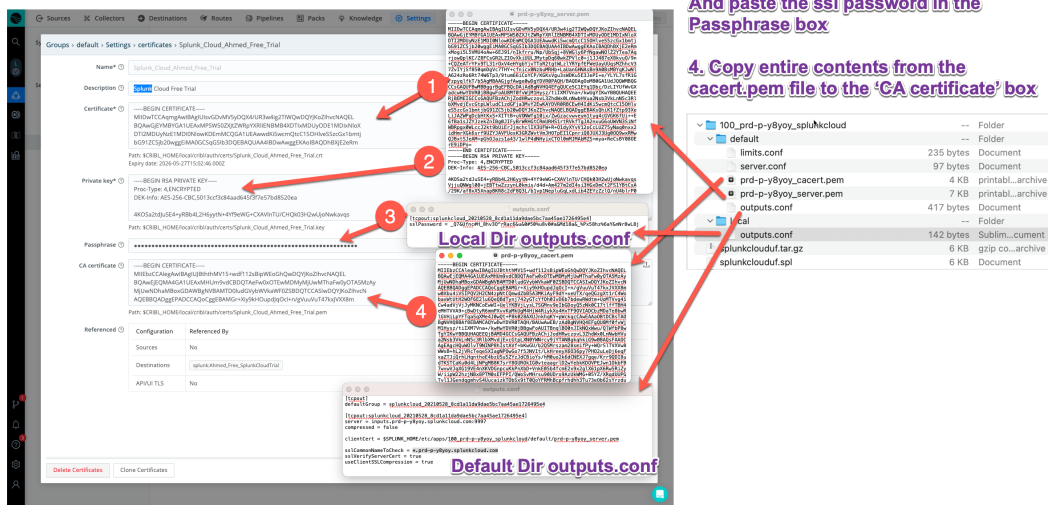
- **Certificate:** Drag and drop the `server.pem` file.
- **Private Key:** Copy and paste just the private key section of the `server.pem` file.
- **Passphrase:** Copy and paste just the SSL password from the `../local/outputs.conf` file.
- **CA certificate:** Drag and drop the `cacert.pem` file.

1. Copy entire contents from `server.pem` to the 'certificate' box

2. From within the `server.pem`, locate the private key section and paste it in the 'private key' box

3. From the `local/outputs.conf`, copy and paste the ssl password in the Passphrase box

4. Copy entire contents from the `cacert.pem` file to the 'CA certificate' box



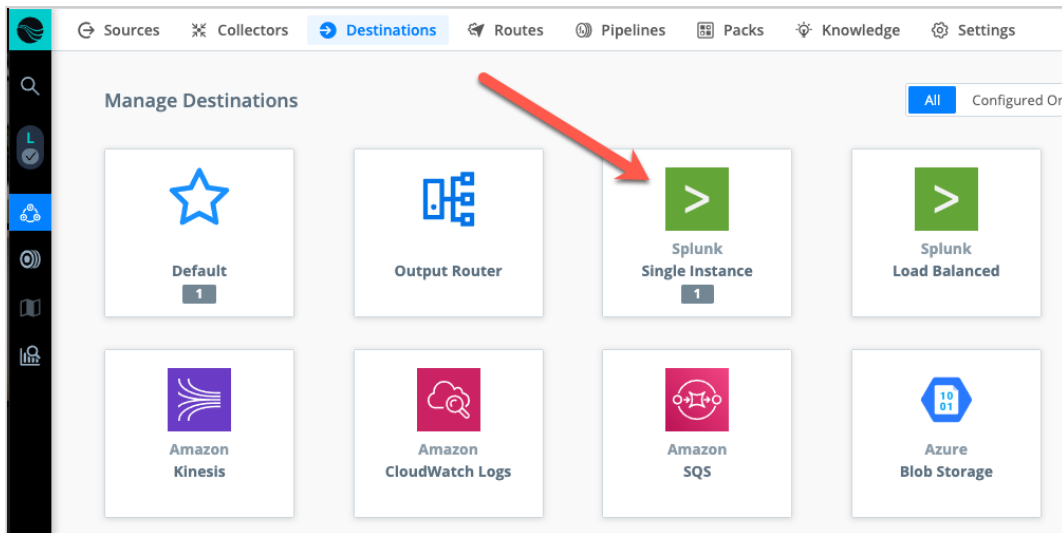
Copying certificate values

Add a Splunk Destination in LogStream

The type of Destination to add depends on what form of Splunk you're using:

- For a trial version of Splunk Cloud, select **Splunk Single Instance**.
- For a paid version of Splunk Cloud, select **Splunk Load Balanced**. This is required because any paid version of Splunk Cloud will have multiple indexer entries in the `../default/outputs.conf` file.

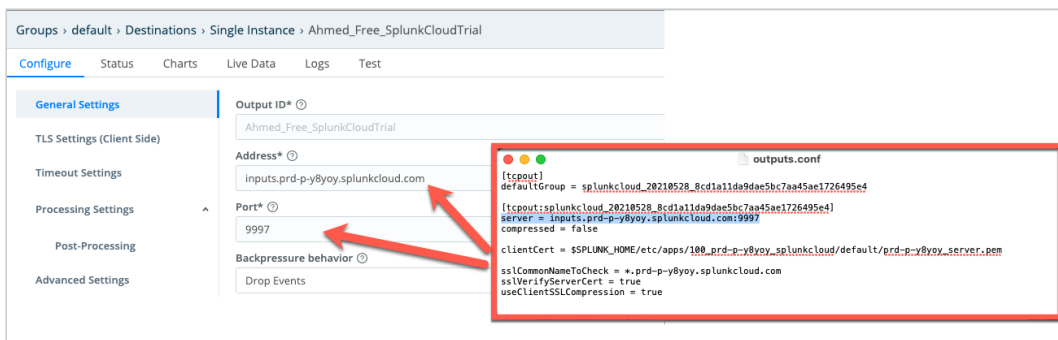
1. From the top nav of a LogStream instance or Group, select **Destinations**, then select **Splunk > Load Balanced** or **Single Instance** from the **Data Destinations** page's tiles or the **Destinations** left nav. Then click **+ Add New** to open the **Load Balanced** or **Single Instance > New Destination** modal.



Creating the Splunk Destination

2. In the **General Settings** tab, populate the **Address** and **Port** fields.

- From the `./default/outputs.conf` file you copied in the previous section, divide the value of the `server` line between the two fields shown in the screenshot below.



The General Settings tab

3. In the **TLS Settings (Client Side)** tab:

- From the **Certificate name** drop-down, select the certificate that you created.
- From the `./local/outputs.conf` file, paste the `sslPassword` value into the **Passphrase** field.
- Click **Save**.
- Click **Commit and Deploy**.

Groups > default > Destinations > Single Instance > Ahmed_Free_SplunkCloudTrial

Configure Status Charts Live Data Logs Test

General Settings

TLS Settings (Client Side)

Timeout Settings

Processing Settings

Post-Processing

Advanced Settings

Enabled ☒ AutoFill? ☒

Validate server certs ☒

Server name (SNI)

Enter server name

Certificate name

Splunk_Cloud_Ahmed_Free_Trial

Private key path (mutual auth)

Private key path (mutual auth)

Certificate path (mutual auth)

Passphrase

Minimum TLS version

Maximum TLS version

From the drop-down, select the certificate added in the prior step. It will populate the fields below

Copy and paste the sslPassword from the ../local/outputs.conf file to the Passphrase box

```

[outputs.conf]
sslPassword = _074uTncn#L8hv30*7Bac66u0958u0v0@uM18ak_NPa58h:46aYsnR8uL8j

```

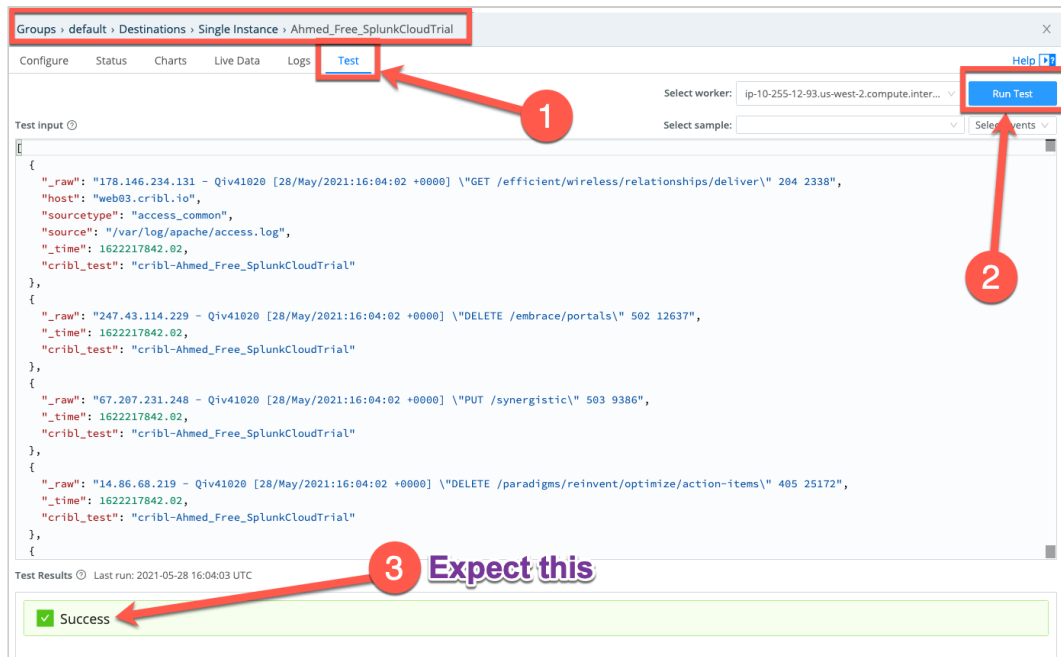
TLS Settings

Verify that Data is Flowing from LogStream to Splunk Cloud

- Be sure you have committed and deployed the newly created configuration. Otherwise, data will not flow to Splunk Cloud, and verification will fail.

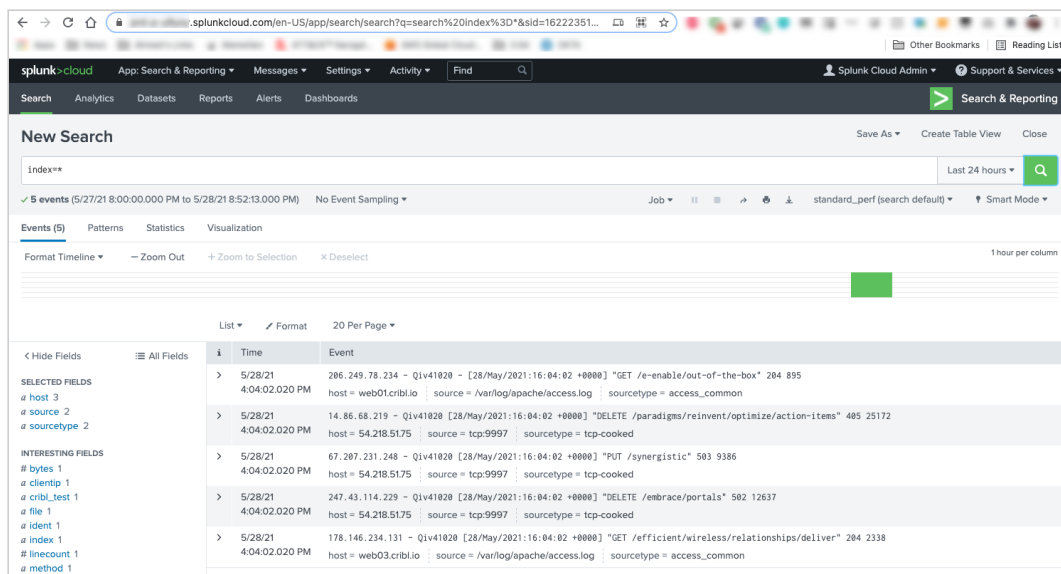
1. In LogStream, open the Destination that you created in the previous section.

- In the configuration modal's **Test** tab, click **Run Test**.
- You should see a **Success** message.



Testing the Splunk Destination

2. In Splunk, search on `index=main cribl_pipe=*`. Events that you sent from the LogStream Test tab should appear in the search results.



Events flowing to Splunk

Using S2S with Splunk BYOL

Before you begin configuring this option, you should already have Splunk Universal Forwarders configured to send data securely to your Splunk environment. This enables you to:

- Re-use content from the `.pem` and `outputs.conf` files already in use on those Forwarders.
- Follow the procedures in the in the [previous section](#) to add the certificate to LogStream Cloud.
- Then, reference the certificate in the Splunk Destination configuration.

If you need to secure your Splunk indexers, see the Splunk [documentation](#).

When Your Data Source is a Splunk Forwarder

If a Splunk [Universal or Heavy Forwarder](#) is the source of the data you want to send to Splunk Cloud:

- In LogStream, create a [Splunk TCP Source](#) to receive data from the Splunk Forwarder.
- This process includes configuring the Splunk Forwarder to point to the new Source in LogStream, and (optionally) securing the communication with TLS.

Webhook/BigPanda Integration

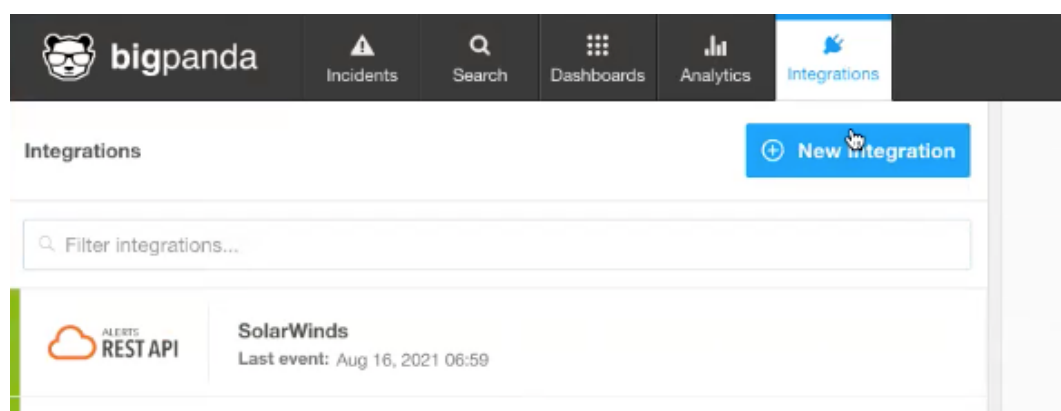
You can configure LogStream to send Webhook notifications to the [BigPanda](#) IT Ops platform. These notifications arrive in BigPanda as [Alerts](#), which BigPanda correlates into [Incidents](#).

Before you begin, you should have an Admin account on a BigPanda Cloud instance.

Prepare BigPanda to Receive Data from LogStream

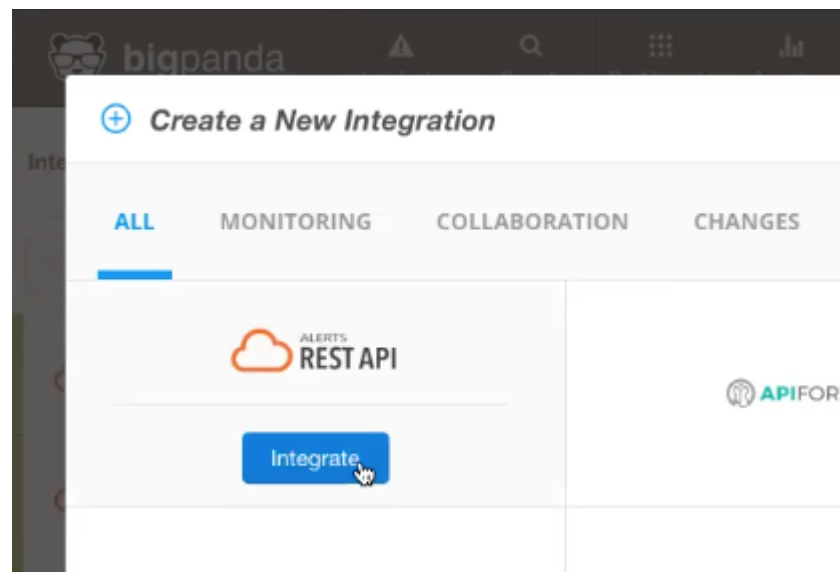
- i** The BigPanda **App Key** and **Access Token** are separate and independent. The **Access Token** is a 32-character string that is part of the value that BigPanda generates for the `Authorization` HTTP header. (It functions like an auth token or bearer token.)

1. Log into your BigPanda Cloud instance as an Admin.
2. In the **Integrations** tab, click **New Integration**.



The Integrations tab

3. In the **Create a New Integration** modal, select **Alerts REST API** and click **Integrate**.



The Create a New Integration modal

- This opens the **Alerts API Integration** page.

Incidents
Search
Dashboards
Analytics
Integrations

Integrations > Alerts API integration

ALERTS
REST API

Description

Easily push custom alert types to BigPanda using our REST API for Alerts. View your custom alerts alongside insights from the rest of your monitoring systems, so you can understand and respond faster to critical issues.

Have Questions?

[Compare supported versions and types](#)
[Contact support](#)

FAQ

Where can I manage existing REST API application? >

1 Create an App Key

First create an App Key. Note: you'll need a separate App Key for each integrated system.

Generate App Key

2 Select One Of The Following

3 Make a REST Call From Your Monitoring System

Configure the integrated system to call the Alerts API endpoint:

Use the following HTTP headers:

Authorization: Bearer

Content-Type: application/json

The JSON payload can contain all or a subset of the following fields:

Field	Description
app_key	Application key from the first step.
status	Status of the alert. One of [ok, critical, warning, acknowledged].
host / service	Main object that caused the alert. Can be the associated host or, if a host isn't relevant, a fields, consider specifying the primary and secondary properties.

The Alerts API Integration page

- In the **Create an App Key** section, generate an App Key named `Cribl LogStream` . You'll need the App Key when configuring LogStream in the next section. LogStream will insert the App Key into every event it sends to BigPanda.
- Store the following information from the **Make a REST Call From Your Monitoring System** section:
 - The Alerts API endpoint URL.
 - The `Authorization` HTTP header. This should consist of the word `Bearer` , a space, and a 32-character string. The 32-character string will be your **Access Token**.
 - The `Content-Type` HTTP header. This should be `application/json` .

Page 1025 of 1087

You'll need the URL and header values when configuring LogStream in the next section.

6. View your completed Cribl LogStream integration in the **Integrations** tab.

Configure the Webhook Destination in LogStream

1. 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.
2. In the **Configure > General Settings** tab, enter or select the following values:
 - **URL:** Enter the Alerts API endpoint URL, for example:
`https://api.bigpanda.io/data/v2/alerts`
 - **Method:** POST
 - **Format:** Custom
 - **Content type:** application/json
3. In the **Authentication** tab, select an **Authentication type**.
 - You can select **Auth token**, and then enter the Access Token (the 32-character string you wrote down [earlier](#)) in the **Token** field.
 - Alternatively, select **Auth token (text secret)** to expose the **Secret** drop-down, in which you can select a [stored secret](#) that references the Access Token. A **Create** link is available to store a new, reusable secret.
4. Click **Save**, then **Commit & Deploy**. You are now ready to test your Webhook Destination's communication with Big Panda.
5. In the **Test** tab, enter the following test input, substituting your own App Key value for the `<your_app_key>` placeholder shown:

```
[
  {
    "app_key": "<your_app_key>",
    "status": "critical",
    "host": "production-database-1",
    "timestamp": 1402302570,
    "check": "CPU overloaded",
    "description": "CPU is above upper limit (70%)",
```



```
    "cluster": "production-databases",  
    "my_unique_attribute": "myUniqueValue987654321"  
  }  
]
```

5. Click **Run Test**.

This should send an alert to BigPanda.

BigPanda Alerts API Requirements

HTTP payloads sent to the BigPanda Alerts API must satisfy rules that are beyond the scope of this topic. For details, see the BigPanda documentation about [Alert Properties](#) and [Integration Diagnostics](#).

However, at at minimum, three fields are required:

1. `app_key` .
2. `status` .
3. `host` OR `service` OR `application` OR `device` .

Thus, the test input shown above works even if you omit all but the first three fields.

There are other possibilities for the third field, but they require understanding how BigPanda determines the `primary_property` of an Alert, plus some additional BigPanda configuration. See the BigPanda links above for details.

Verify that BigPanda is Receiving Notifications and Events

In the BigPanda **Incidents** tab, you should see an Incident whose Source is `Cribl LogStream` . The details of the test input you sent from the Webhook Destination should appear in an Alert within that Incident. If so: It works!

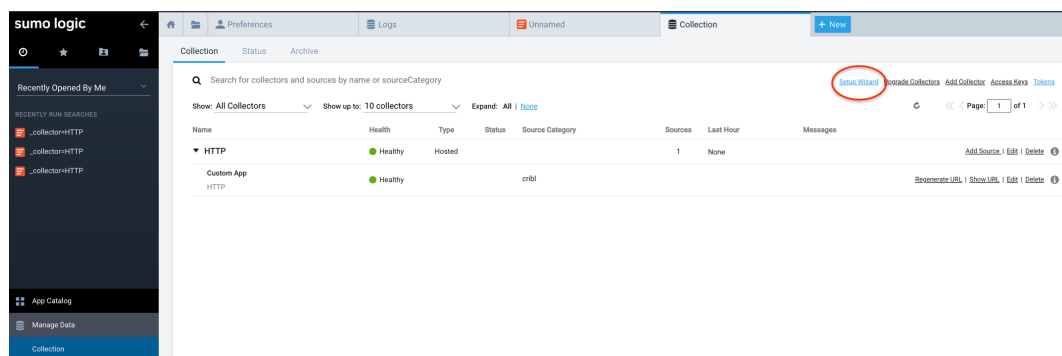
Webhook/Sumo Logic Integration

You can configure LogStream to send raw event data through a [Webhook Destination](#) to Sumo Logic's log management and analytics service, which can then apply [field extraction rules](#) that you configure.

Before you begin, you need an Admin account in Sumo Logic.

Prepare Sumo Logic

1. In the Sumo Logic **Manage Data > Collection** tab, click **Setup Wizard**.



The Collection Setup Wizard

2. In the **Start Streaming Data to Sumo** tile, click **Get Started** to open the **Select Data Type** page.
3. Click **Your custom App** to open the **Set Up Collection** page.
4. Select **HTTP Source** to open the **Configure Source: HTTP Source** page.
5. In the **Source Category** field, enter a tag for Sumo Logic to add to each event. The purpose of this tag is to make it easy for Sumo Logic search to identify events coming from LogStream. In LogStream, you can override the value you enter here, either on a per-event basis, or globally in a configuration file.

sumo logic

Select Data Type

Set Up Collection

Configure Source

Finish

Configure Source: HTTP Source

1. Enter a Source Category that will help you search your logs later.

Source Category ?

cribl2

2. Select a time zone for your log file.

☒ Use time zone from log file. If none present use:

(UTC) Etc/UTC

☐ Ignore time zone from log file and instead use:

(UTC) Etc/UTC

Back

Next

Entering a Source Category

6. Click **Next**. Sumo Logic then displays the HTTP Source URL, which you will need when configuring the Webhook Destination in LogStream.

sumo logic

Select Data Type

Set Up Collection

Configure Source

Finish

Configure Source: HTTP Source

1. An HTTP Source has been automatically configured for you. Copy the following URL.

https://endpoint6.collection.us2.sumologic.com/receiver/v1/http/Z4W0edwMMXpT0uP2e04G03G0W12S0wv2M
LUMW1qM5p2m120eFnyUdpgG0uMjg0H03Q6m07F0301n0of00q220m0q00
eRD0g==

Copy

2. Use the URL as the target for your Source. [Learn more](#)
[View cURL example](#)

Back

Next

The HTTP Source URL

7. Click **Copy** to capture the URL.
8. Click **Next** to finish configuring the HTTP Source.

Configure LogStream's Webhook Destination

1. 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.
2. In the **Configure > General Settings** tab, enter or select the following values:
 - **URL:** Enter the Sumo Logic HTTP Source URL.
 - **Method:** POST
 - **Format:** Custom
 - **Source Expression** = `_raw`
 - As needed, add more fields (e.g., `host`) for Webhook to include in the raw output.
 - **Content type:** `application/text`

The screenshot shows the 'General Settings' tab for a Webhook destination named 'Sumo_Webhook'. The configuration fields are as follows:

- Output ID:** Sumo_Webhook
- URL:** `https://endpoint6.collection.us2.sumologic.com/receiver/v1/http/...`
- Method:** POST
- Format:** Custom
- Source expression:** `_raw`
- Drop when null:** ☒
- Event delimiter:** `\n`
- Content type:** `application/text`
- Backpressure behavior:** Block

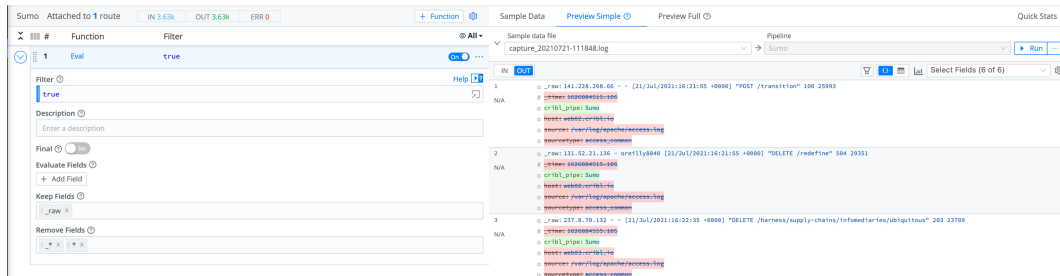
Webhook Destination General Settings

3. In the **Authentication** tab, select an **Authentication type**.
 - Select the option that supports (1) your [Sumo Logic authentication](#) configuration, and (2) whether or not you're using a secret in LogStream to reference your Sumo Logic access ID and key.
4. In the **Post-Processing** tab, remove `cribl_pipe` from **System Fields**.
5. Click **Save**, then **Commit & Deploy**.

Create a Pipeline

Create a LogStream Pipeline to remove all except the **Source Expression** fields. Add an **Eval** Function with the following values:

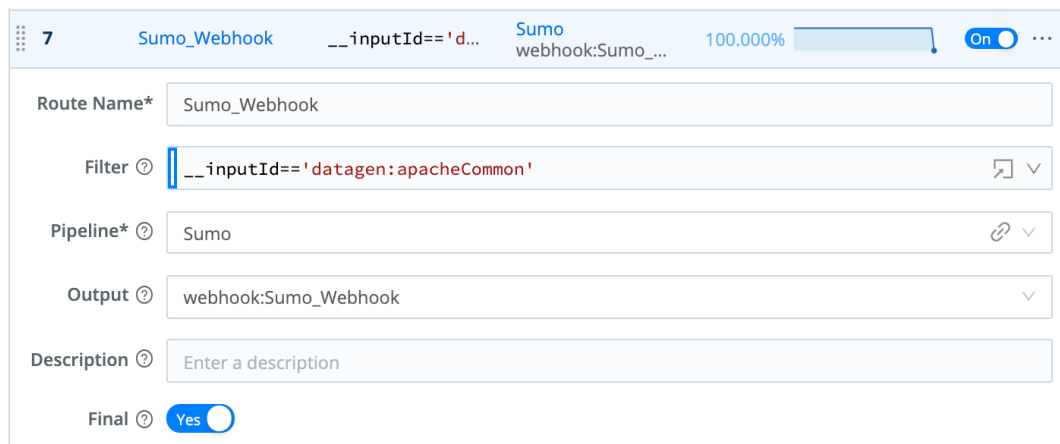
- **Keep Fields:** Specify the same field(s) you specified in **Source Expression** when you created the Webhook Destination (e.g., `_raw` , `hosts`).
- **Remove Fields:** `_*` and `*` , to remove all other fields.



Example Pipeline

Create a Route

Configure a Route similar to the example below, replacing the names/IDs shown with the actual names/IDs from your setup.

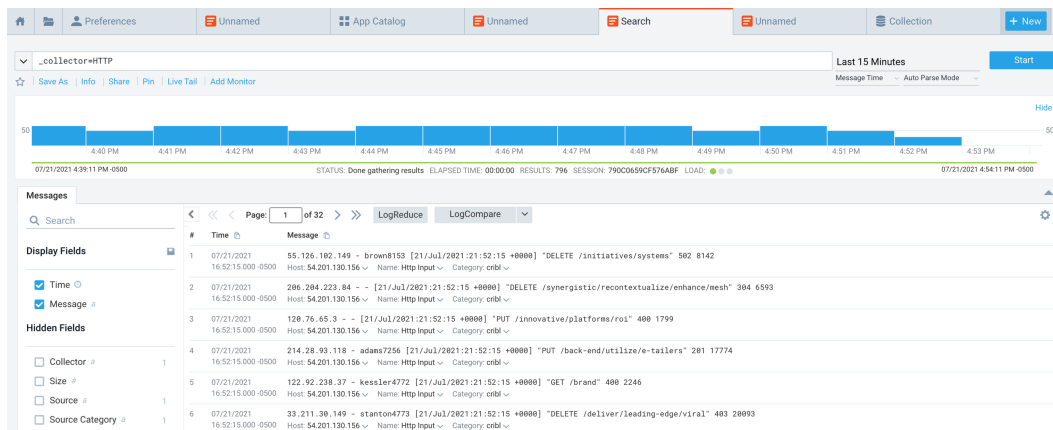


Example Route

Verify that Sumo Logic Is Receiving Data

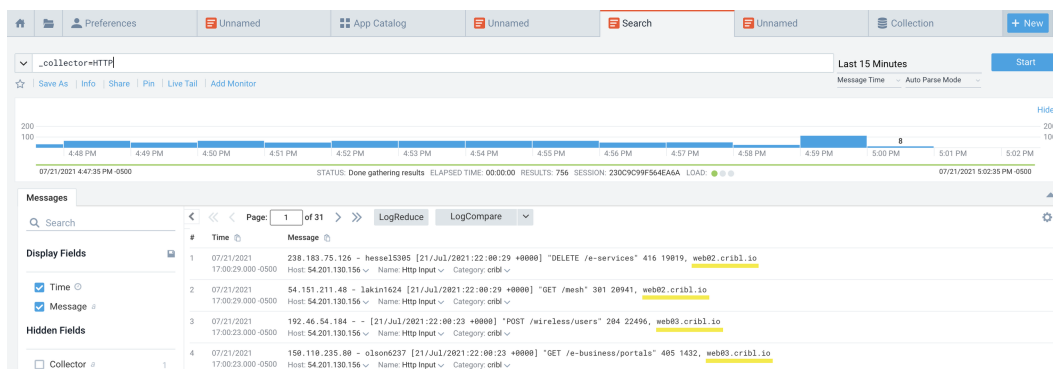
In Sumo Logic, search for `_collector=HTTP` . The event data returned should be broken out into fields.

If your Webhook Destination specifies **Source Expression** = `_raw` , you should see output similar to this:



Sumo Logic Search Results

If your Webhook Destination specifies **Source Expression** = `_raw , host , you` should see output similar to this, where the added field (`host`) appears at the end of the raw string.



Sumo Logic Search Results with an Added Field

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



Introducing Cribl LogStream

01:45



Conceptual overview of LogStream's capabilities

Concept – Notifications

Cribl Concept - Notifications

01:07

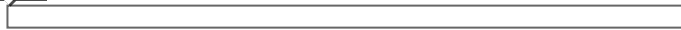


Brief walkthrough of LogStream 3.1+'s Notification capabilities

Concept – The Code Function

Cribl Concept: The Code Function

01:31



Brief walkthrough of LogStream 3.1+'s Code Function

Concept: LogStream Cloud

Concept: LogStream Cloud

01:59 |



Conceptual walkthrough of Cribl's LogStream Cloud offering
Sign up for a Cribl.Cloud account at: <https://cribl.io/logstream-cloud/>

Concept: AppScope

Concept-AppScope

02:30 |



Brief walkthrough of Cribl's open-source [AppScope](#) project

Concept: Receive

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

01:40



A quick overview of LogStream's data collection capabilities

Concept: Pipelines

A brief conceptual walkthrough of how Pipelines work in LogStream

Concept: Routes

A brief conceptual walkthrough of how LogStream routes data

Concept: Role-Based Access Control

Cribl Concept-Role Based Access Control

01:43



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: Notifications

Cribl How-To: Notifications

02:25

A video player interface showing a progress bar. The progress bar is a horizontal line with a small vertical marker at the beginning, indicating the current position in the video.

A quick walkthrough of sending LogStream [Notifications](#) via [Webhooks](#) to a [Slack target](#).

How-to: The Code Function

Cribl How-To: The Code Function

02:32

A video player interface showing a progress bar. The progress bar is a horizontal line with a small vertical marker at the beginning, indicating the current position in the video.

A quick walkthrough of how to use LogStream 3.1+'s [Code Function](#).

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

01:44



A quick walkthrough of the steps to create a LogStream Pipeline

How-to: Creating a Route

How-To: Creating a Route

from **Cribl**

02:36

A walkthrough of creating a Route in LogStream

How-to: Configuring an 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

02:46

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

02:03



A brief walkthrough on scheduling data collection jobs in LogStream

How-to: Securing Logstream

Securing LogStream

from **Cribl**

14:25



Use TLS to secure a LogStream distributed deployment

How-to: Logstream Fault Tolerance

LogStream Fault Tolerance

15:11

Recover a failed Leader Node from a remote Git repository

How-to: Worker Group-to-Worker Group Communication

LogStream Worker Group to Worker Group

from Cribl

10:53

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

If you're managing your own LogStream deployment ([single-instance](#) or [distributed](#)), you 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. That site works only when using the `cribl diag` command or uploading using the LogStream UI. (So connecting directly to it with your web browser will fail.)

With a [Cribl.Cloud](#) deployment, [contact](#) Cribl Support to gather a diag bundle on your behalf.

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
```

Including CPU Profiles

If Cribl Support asks you to grab CPU profiles of Worker Processes, follow these steps:

1. Use `top` or `htop` on the Worker Node to identify Worker PIDs consuming a lot of CPU.
2. See [Sizing & Scaling > CPU Profiling](#) for instructions on accessing the UI's **Profile** options (for your deployment type), and generating and saving profiles.
3. Find the Worker Processes matching the PIDs you identified above.
4. Click **Profile** on each. Start with the default 10-second **Duration**.
5. Once the profile is displayed, save it to a JSON file. (See details at the above link.)
6. Repeat steps 3–6 for other CPU-intensive Worker Processes.
7. Upload the profile JSON files to Cribl Support.

i On an already CPU-starved Worker Node, profiling might fail with an error message, or just hang. In this case, you might need a few retries to get a successful profile.

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

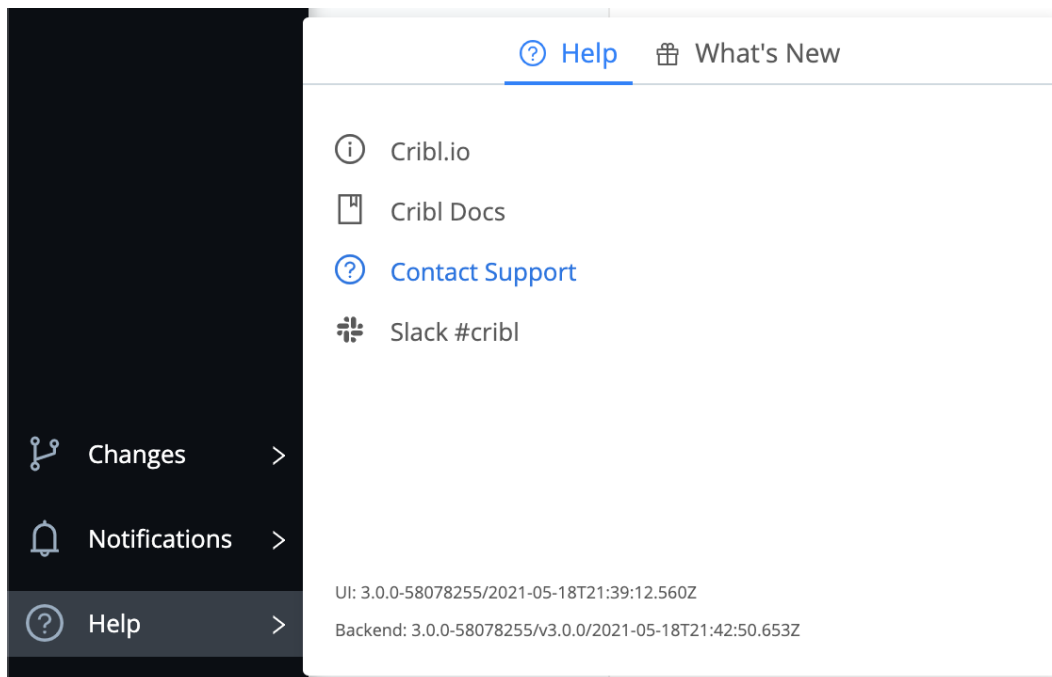
You have several options for opening a support case.

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.



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

2021-09-14 – v.3.1.1 – Modifying Collector > Preview > Capture settings can break Capture elsewhere

Problem: Modifying the capture settings in a Collector's Run > Preview > Capture modal can improperly modify the Filter expression in Capture modals for other Collectors, Sources, and Routes/Pipelines.

Fix: Planned for LogStream 3.1.2.

2021-09-10 – v.3.1.1 – Worker Group certificate name drop-down shows certificates from the Leader

Problem: When configuring certificates at **Groups** > <group-name> > **Settings** > **API Server Settings** > **TLS**, certificates configured on the Leader incorrectly appear on the **Certificate name** drop-down.

Fix: Planned for LogStream 3.1.2.

2021-09-10 – v.3.1.1 – Git Collapsed Actions broken in 3.1.1

Problem: With **Collapsed Actions** enabled, clicking the **Commit & Push** button has no effect. (The **Commit & Deploy** button works properly.)

Workaround: Disable **Collapsed Actions**, to restore separate **Commit** and **Git Push** buttons.

Fix: Planned for LogStream 3.1.2.

2021-09-08 – All versions through v.3.1.1 – UDP support is currently IPv4-only

Problem: Where Sources and Destinations connect over UDP, they currently support IPv4 only, not IPv6. This applies to Syslog, Metrics, and SNMP Trap Sources; and to Syslog, SNMP Trap, StatsD, StatsD Extended, and Graphite Destinations.

Workaround: Integrate via IPv4 if possible.

Fix: IPv6 support is planned for LogStream 3.1.2.

2021-09-02 – v.3.1.0 – Google Pub/Sub authentication via proxy environment variable fails

Problem: When LogStream's Google Cloud Pub/Sub Source and Destination attempt authentication through a proxy, using the `https_proxy` environment variable, they send an HTTP request to

<http://www.googleapis.com:443/oauth2/v4/token>. This request fails with 504/502 errors. The root cause is a mismatch in dependency libraries, whose correction has been identified, but requires broader testing.

Workaround: Configure the proxy in `transparent` mode, to avoid relying on environment variables.

Fix: Planned for LogStream 3.1.2.

2021-08-24 – v.3.1.0 – High CPU load with LogStream version 3.1.0

Problem: LogStream 3.1.0 added code-execution safeguards that inadvertently increased CPU load, and decreased throughput, with several Functions and most expressions.

Workaround: Downgrade to v.3.0.4.

Fix: Upgrade to v.3.1.1.

2021-08-18 – v.3.1.0 – Clone Collector option broken

Problem: Clicking a 3.1.0 Collector modal's **Clone Collector** button simply closes the modal. (If you have unsaved changes, you'll first be challenged to confirm closing the parent modal – but the expected cloned modal won't open.)

Workaround: Click + **Add New** to re-create your original Collector's config from scratch, adding any desired modifications.

Fix: In LogStream 3.1.1.

2021-08-18 – All versions through 3.1.0 – Tabbed code blocks broken on in-app docs

Problem: When docs with tabbed code blocks are opened in the Help drawer, the default (leftmost) tab seizes focus. Other tabs will not display when clicked.

Workaround: Click the blue/linked page title atop the Help drawer to open the

same page on docs.cribl.io, where all tabs can be selected.

Fix: In LogStream 3.1.1.

2021-08-14 – v.3.1.0 – Splunk Load Balanced Destination does not migrate auth type

Problem: In a Splunk Load Balanced Destination with **Indexer discovery** enabled and a corresponding **Auth token** defined, upgrading to LogStream 3.1.0 corrupts the **Auth token** field's value.

Workarounds: Set the **Authentication method** to **Manual** and resave the token's value.

Fix: In 3.1.1.

2021-08-11 – v.3.1.0 – C.Secret() values are undefined in Collectors

Problem: Calling the [C.Secret\(\)](#) internal method within a [Collector](#) field resolves incorrectly to an undefined substring. E.g., in URL fields, `C.Secret()` values will resolve to `/undefined/` path substrings.

Workarounds: 1. Use `C.vars` and a Global Variable, instead of using this method. 2. Root cause is that `C.Secret()` in Collectors and Pipeline Functions has access only to secrets that were created before the last restart. Therefore, restart Worker Processes to refresh the method's access.

Fix: In 3.1.1.

2021-08-10 – v.3.1.0 – Pre-processing Pipelines break Flows display

Problem: Attaching a pre-processing Pipeline to a Source breaks the **Monitoring > Flows (beta)** page's display. Attempting to remove Sources/Destinations from that page's selectors throws a cryptic Sankey error.

Workaround: Temporarily detach pre-processing Pipelines if you want to check Flows.

Fix: Planned for LogStream 3.1.1, but couldn't reproduce the error.

2021-07-26 – v.3.0.x–3.1.0 – Packs with orphaned lookups block access to Worker Groups

Problem: If a Pack references a lookup file that's missing from the Pack, pushing the Pack to a Worker Group will block access to the Group's UI. You will

see an error message of the form: "The Config Helper service is not available because a configuration file doesn't exist... Please fix it and restart LogStream."

Workaround: On the Leader Node, review the config helper logs (`$CRIBL_HOME/log/groups/<group>/*.log`) to see which references are broken. (In a single-instance deployment, see `$CRIBL_HOME/log/*.log`.) Then manually resolve these references in the Pack's configuration.

Fix: Planned for LogStream 3.1.1, but couldn't reproduce the error.

2021-07-20 – v.3.0.3 – Can't add Functions to a Pipeline named `config`

Problem: You cannot add Functions to a Pipeline if the Pipeline is named `config`, because this name conflicts with the reserved route for the **Create Pipeline** dialog.

Workarounds: Don'tcha name your Pipelines `config`.

Fix: Version TBD.

2021-07-06 – All versions through 3.1.x – Duplicate Workers/Worker GUIDs

Problem: Multiple Workers have identical GUIDs. This creates problems in [Monitoring, upgrading](#) and versioning, etc., because all Workers show up as one.

Cause: This is caused by configuring one Worker and then copying its `cribl/` directory to other Workers, to quickly bootstrap a deployment.

Workaround: Don't do this! Instead, use the [Bootstrap Workers from Leader](#) endpoint.

Fix: Planned for LogStream 3.2.0.

2021-07-02 – v.3.0.2–3.0.3 – Sample file's last line not displayed upon upload

Problem: When uploading (attaching) a sample data file, the file's final line is not displayed in the **Add Sample Data** modal.

Workarounds: This is a UI bug only. LogStream correctly processes the complete sample data, which should show up when viewing the sample afterwards (e.g., within a Pipeline's preview pane).

Fix: In LogStream 3.1.0.

2021-07-02 – All versions – Date fields misleadingly preview with string symbol

Problem: In [Preview or Capture](#), incoming events like `_raw` will be displayed in the right pane with an `α` symbol that indicates string data. However, calling `new Date()` and then `C.Time.strptime()` [methods](#) in an [Eval](#) Function will return `null` on the OUT tab.

Cause: Due to the nature of JSON serialization, the incoming event's `Date` field is misleadingly subsumed under the event's `α` string symbol. It's actually a structured type, not a string...yet.

Workaround: If you see unexpected `null` results, stringify the datetime field as you extract it, e.g.: `new Date().toISOString()`. Feeding the resulting field to Time methods should return datetime strings as expected.

2021-06-22 – v.3.0.0–3.0.2 – Internal

`C.Text.relativeEntropy()` method – broken typeahead and preview

Problem: The `C.Text.relativeEntropy()` internal method is missing from JavaScript expressions' typeahead drop-downs. You can manually type or paste in the method, and save your Function and Pipeline, but LogStream's right Preview pane will (misleadingly) always show the method returning `0`.

Workarounds: Use other means (such as the **Live** button) to preview and verify that the method is (in fact) returning valid results.

Fix: In LogStream 3.0.3.

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: In 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: In 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: In 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: In LogStream 3.1.0.

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.

2021-05-03 – v.2.4.4–3.01 – Rollup Function suppresses sourcetype metrics

Problem: `sourcetype` metrics can be suppressed when the Cribl Internal > CriblMetrics Source is enabled and the `cribl_metrics_rollup` pre-processing Pipeline is attached to a Source.

Workarounds: Disabling the pre-processing pipeline restores `sourcetype` and any other missing data. However, without the rollup, a much higher data volume will be sent to the indexing tier.

Fix: In LogStream 3.0.2.

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: In LogStream 3.0.2.

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: In 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: In 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: In LogStream 3.0.

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: In LogStream 2.4.1.

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 `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 v1.7 (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.
- NodeJS serves as the LogStream backend and has a large collection of well documented errors of its own. Some of system-level are listed below with the appropriate action to take. The remainder are documented at https://nodejs.org/docs/latest-v14.x/api/errors.html#errors_node_js_error_codes, specifically in the Common system errors section of that page.
- 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.

Web UI

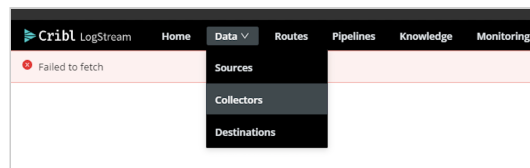
Where: In LogStream's UI.

Error: "Failed to fetch"

Cause: This occurs only when accessing **Data > Collectors**. It can occur with ad blockers, and occurs with the Brave browser (because the newly loaded page's URL includes the string `collector`).

Recommendation: If you have an ad blocker, allowlist the `https://<hostname>:9000/jobs/collectors` (single-instance) URL or the

`https://<masternode>:9000/m/<group_name>/jobs/collectors` (Leader)
URL.



Failed to fetch

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.

Warning: "KMS saved, but secret exists in destination location. No data migrated."

Cause: This will appear when you attempt to save KMS Settings with a **Secret Path** entry that references a secret that already exists in HashiCorp Vault. It can also occur with OpenID Connect remote authentication. LogStream has aborted the remote write to avoid overwriting an external shared secret.

Recommendation: On the Leader Node, manually edit `kms.yml` to use `provider: local`. Then restart LogStream to correct the path conflict.

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.

```
message: Initialization error
error:
  message: bind EACCES 0.0.0.0:514
  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.

```
message: Initialization error
error:
  message: bind EADDRINUSE 0.0.0.0:9514
  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 TCP source

Where: These events will be in the Worker Process logs.

Error: "connection rejected" with a reason of "Too many connections"

Cause: The maximum number of active Splunk TCP connections has been exceeded per worker process. The default is 1000.

Recommendation: In the Splunk TCP input's Advanced Settings configuration, increase the Max Active Connections value, set it to 0 for unlimited, and/or increase the # of worker processes the Worker Node(s) are using..

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.

```

  channel: task
  level: error
  message: task execution failure
  host: worker248
  jobId: 1618350773.7.adhoc.NWS
  reason:
    message: Invalid URL: undefined
    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

Error: "statusCode: 429...Too many requests"

Cause: This response is triggered by rapidly repeated authentication requests from the Collector's Discover and Collect phases. It's especially likely when different Workers run multiple Collect tasks.

Recommendation: Navigate to **Settings > General > Advanced** and gradually increase the **Login Rate Limit** until this error response is no longer returned.

AWS Sources/Destinations & S3-Compatible Stores

Where: These events will be in the Worker Process logs.

Error: "message": "Inaccessible host: sqs.us-east-1.amazonaws.com'. This service may not be available in the us-east-1' region.", "stack": "UnknownEndpoint: Inaccessible host: sqs.us-east-1.amazonaws.com'. This service may not be available in the us-east-1' region."

Cause: If this is persistent rather than intermittent then it could be caused by TLS negotiation failures. For example, AWS SQS currently does not support

TLS 1.3. If intermittent then a network-related issue could be occurring such as DNS-related problems.

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.

```
message: Bucket does not exist!
bucket: bucket1
endpoint: https://minio.mydomain.com:9090
error:
  message: Missing credentials in config
  stack: Error: connect ETIMEDOUT 169.254.169.254:80
        at TCPConnectWrap.afterConnect [as oncomplete]
```

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 Leader when the Worker can't connect to the Leader Node to send metrics data. When triggered by cluster communication, the warning will be in the Worker's API log.

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.

```

  channel: output:minio
  level: error
  message: Bucket does not exist!
  bucket: bucket1
  endpoint: http://minio.mydomain.com:9090
  error:
    message: Parse Error: Expected HTTP/
    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

Pipelines/Functions

Where: These events will be in the Worker Process logs.

Error: "failed to load function...Value undefined out of range..."

Cause: A [Lookup Function](#) attempted to load a lookup table that exceeded Node.js' hard size limit of 16,777,216 (i.e., 2^{24}) rows.

Recommendation: Split the lookup table to smaller tables, or use the [Redis Function](#).

```
Preview Log
1  s sage:"starting function lister","dir":"/opt/cribl/groups/default"}
2  s sage:"updated functions list","enabled":32,"all":32}
3  "Loading relative entropy models","dir":"/opt/cribl/groups/default/data/lookups"}
4  "Loading relative entropy model","modelName":"top_domains","file":"model_relative_entropy_top_domains.csv"}
5  s sage:"loading CSV lookup","path":"/opt/cribl/groups/default/data/lookups/model_relative_entropy_top_domains.csv"}
6  s sage:"done loading CSV lookup","path":"/opt/cribl/groups/default/data/lookups/model_relative_entropy_top_domains.csv","elapsed":5,"rowCount":38}
7  "Successfully loaded records into model","numRows":18,"modelName":"top_domains","path":"/model_relative_entropy_top_domains.csv"}
8  s sage:"loaded grok patterns","count":152}
9  s sage:"profiler started"}
10 s sage:"creating new pipeline","id":"lookup","conf":{"output":"default","groups":{"},"asyncFuncTimeout":1000,"functions":[{"id":"serde","filter":"true","disabled":null,"conf":{"mode":"extra
11 message:"start loading and initializing functions","count":3}
12 message:"Creating Lookup: ","matchMode":"exact","matchType":"first","file":"maxRowsPlusOne.csv.gz"}
13 s sage:"loading CSV lookup","path":"/opt/cribl/groups/default/data/lookups/maxRowsPlusOne.csv.gz"}
14 message:"failed to load function","error":{"message":"Value undefined out of range for undefined options property undefined","stack":"RangeError: Value undefined out of range for undefi
15 message:"finished loading and initializing functions","count":1}
16 s sage:"START pushing stdin events","id":"lookup"}
17 s sage:"FINISH pushing stdin events","id":"lookup"}
18 s sage:"profiler stopped"}
19 s sage:"loading CSV lookup","path":"/opt/cribl/groups/default/data/lookups/maxRowsPlusOne.csv.gz"}
20
```

Oversized lookup table error

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.

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.

Distributed Management

General Settings

Master Settings

TLS Settings

Git Settings

General

Remote

Scheduled actions

Enabled

YesAutofill?

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 – 2.880.0
@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
google_protobuf – 3.15.6
google_cloud_pubsub – 2.12.0
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
ldaps – 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