



Cribl LogStream Documentation Manual

Version: v2.0

INTRODUCTION	5
About Cribl	5
Basic Concepts	7
DEPLOYING CRIBL	9
Before Deploying	9
Standalone Deployment	12
Single Instance Deployment	13
Distributed Deployment	16
Splunk App Deployment	23
Scaling	28
Config Files	30
cribl.yml	32
inputs.yml	34
outputs.yml	35
licenses.yml	37
regexes.yml	38
breakers.yml	40
mappings.yml	41
instance.yml	42
Licensing	43
User Authentication	45
Persistent Queues	47
Securing and Monitoring	49
Upgrading	51
Diagnosing	53
Uninstalling	54
WORKING WITH CRIBL	55
Routes	55
Pipelines	58
Functions	61
Auto Timestamp	63
Aggregations	65
CEF Serializer	70
Clone	72
Comment	73
Drop	74

Dynamic Sampling	75
Eval	77
Flatten	79
GeoIP	81
JSON Unroll	82
Lookup	84
Mask	86
Numerify	87
Parser	88
Publish Metrics	93
Regex Extract	95
Regex Filter	97
Sampling	98
Serialize	99
Suppress	100
Tee	102
XML Unroll	103
Prometheus Publisher (beta)	106
Reverse DNS (beta)	107
Sources	108
Splunk	110
Syslog	112
TCP JSON	114
HTTP(S)	117
Kafka	120
Kinesis Streams	123
Azure Event Hubs	125
Metrics	127
SQS	129
S3	131
Cribl Internal	134
Destinations	135
Splunk	138
Splunk Load Balanced	140
Splunk HEC	144
S3 Compatible Stores	146
Kinesis Streams	148
SQS	150

CloudWatch Logs	152
Filesystem/NFS	153
Elasticsearch	155
Honeycomb	157
TCP JSON	159
Syslog	161
Kafka	163
Azure Blob Storage	166
Azure Event Hubs	168
Azure Monitor Logs	170
StatsD	172
StatsD Extended	173
Graphite	174
Output Router	175
Data Preview	176
Securing Data	180
Encryption	181
Decryption	185
Scripts	187
EXPRESSION REFERENCE	188
Introduction	188
Cribl Expressions	191
KNOWLEDGE	197
Regex Library	197
Event Breakers	199
Lookups Library	203
Parsers Library	205
USE CASES	206
Ingest-time Fields	206
Ingest-time Lookups	209
Sampling	213
Access Logs: Apache, ELB, CDN, S3 etc.	215
Firewall Logs: VPC Flow Logs, Cisco ASA etc.	218
Masking and Obfuscation	221
Regex Filtering	225
Encrypting Sensitive Data	227
KNOWN ISSUES	233
Known Issues	233

THIRD PARTY SOFTWARE

234

Current List

234

INTRODUCTION

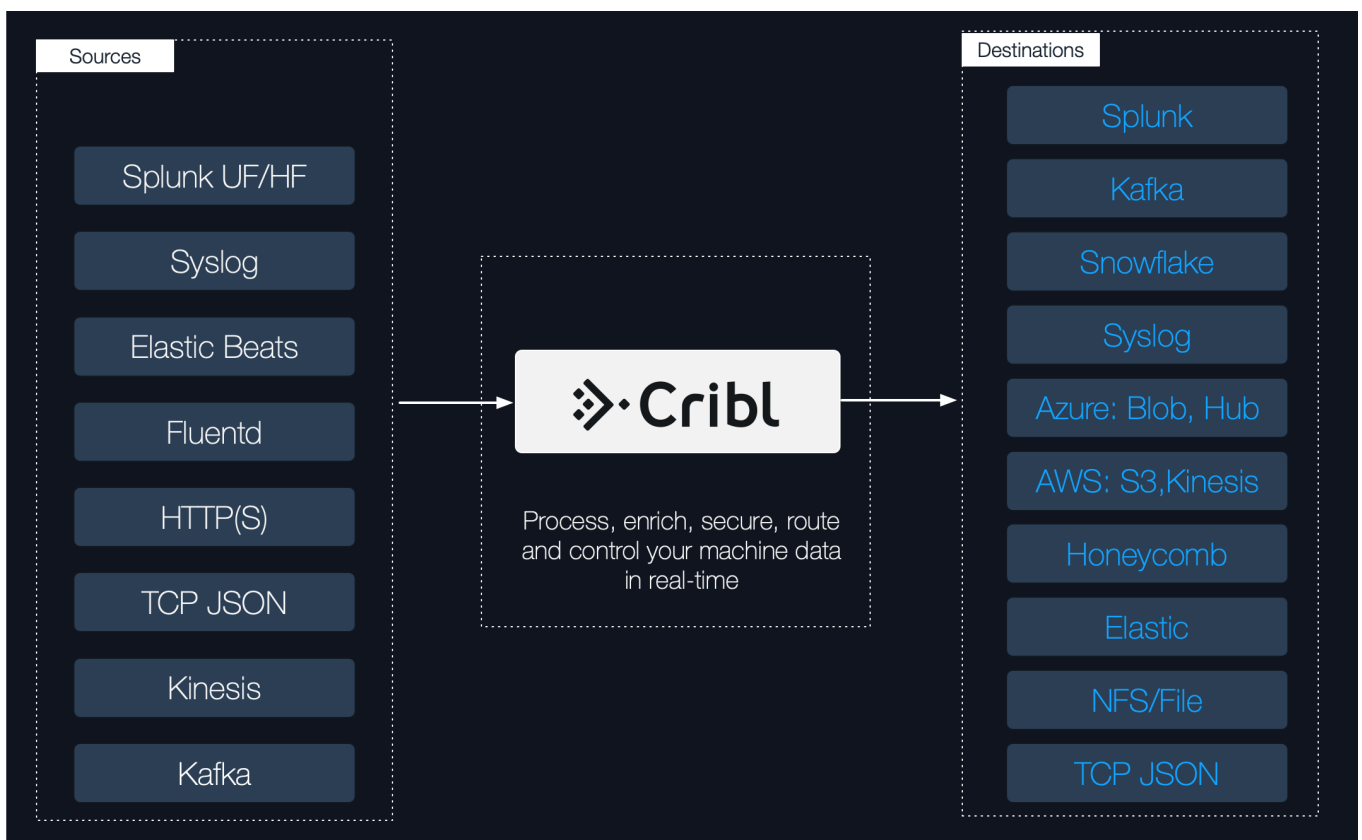
About Cribl

Getting started with Cribl

What is Cribl?

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 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 .



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

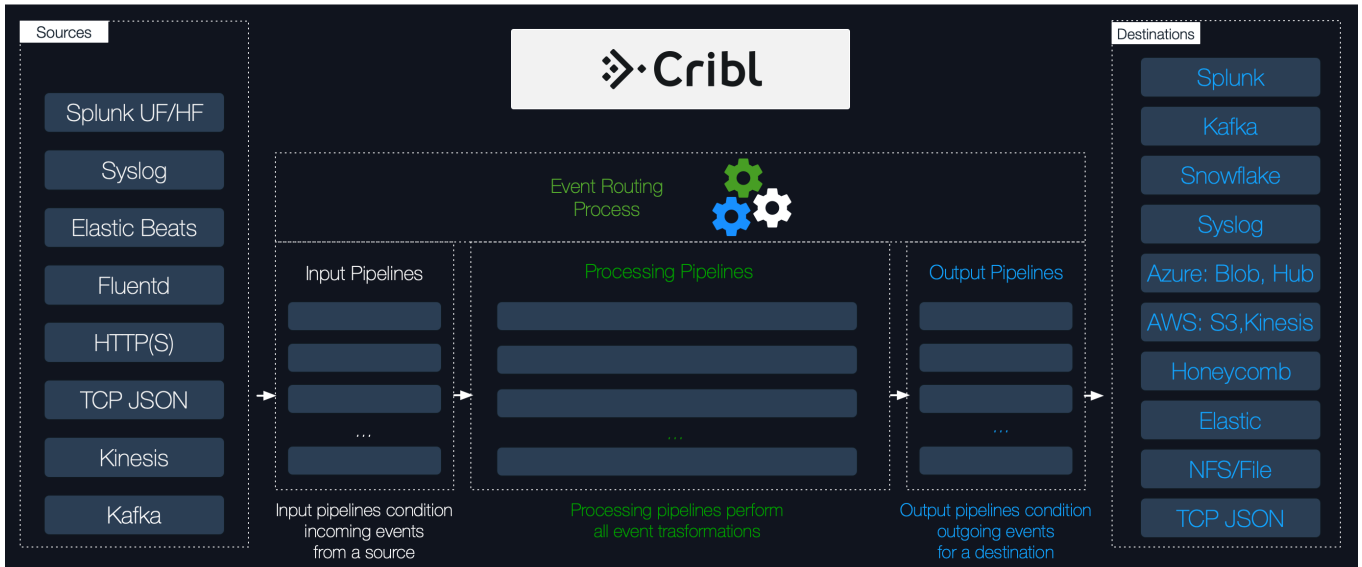
Who is Cribl for?

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

Basic Concepts

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

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



Functions

At its core a **function** is a piece of code that executes on an event and it encapsulates the smallest amount of processing that can happen to that event. For instance, a simple function can be one that replaces the term `foo` with `bar` on each event. Another one can hash or encrypt `bar` and yet another can add a field, say, `dc=jfk-42` to any event with `source=*us-nyc-application.log`. Functions process each event that passes thru them. To help improve performance, functions can be optionally configured with filters to limit processing scope on matching events only. More details on functions.

Pipelines

A series of functions is called a pipeline and the order in which they are executed matters. Events are delivered at the beginning of a pipeline (by a Route, see below) and as they're processed by a function they are passed to the next one down the line. Events only move forward, towards the end of the pipeline and eventually out of the system. More details on pipelines.

Routes

Routes evaluate incoming events against filter expressions to find the appropriate pipeline to send them to. Routes are **evaluated in order**. A Route can be associated **only with one** pipeline and one output. By default, a Route-Pipeline-Output tuple will consume matching events. If the `Final` flag is disabled, one or more clones are sent down the pipeline while the original event continues down the rest of the routes. This is very useful in cases where the same set of events needs to be processed differently and delivered to different destinations. More details on routes.

DEPLOYING CRIBL

Before Deploying

Deployment Options

There are two deployment options for Cribl; standalone or as a Splunk app. Your exact choice will depend on your requirements. Both packages are available for download [here](#).

Requirements and Supported Platforms

- **OS:**
 - Linux: RedHat, CentOS, Ubuntu, AWS Linux, Suse (64bit)
 - macOS 10.13 and 10.14
- **System:**
 - +4CPUs
 - +4GB RAM
 - 5GB free disk space (more if persistent queuing is enabled)

Recommended AWS Instance Types

c5d.2xl or higher (8vCPUs, 16GB RAM, 200GiB NVMe SSD)

c5.2xl or higher (8vCPUs, 16GB RAM, EBS)

c3.2xl or higher (8vCPUs, 15GB RAM, 2x160GiB SSD)

(the higher the CPU clock, the better)

i As of v1.7 Node is no longer a runtime dependency.

Network Ports

*Cribl needs these ports to be available by default:

- Cribl UI. Default: `9000` (both options)

- Cribl HTTP In. Default: 10080 (Standalone)
- Splunk to Cribl data port. Default: localhost:10000 (Cribl App for Splunk)
- | criblstream Splunk search command to Cribl. Default: localhost:10420 (Cribl App for Splunk)

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` .
Note: For Splunk to Cribl the corresponding `server` attribute in `[tcpout:cribl]` defined by default in `default/outputs.conf` , on Splunk side, can be overridden by re-defining it in `local/outputs.conf` (Splunk conf file precedence applies - local overrides default).

`$SPLUNK_HOME/etc/apps/cribl/local/outputs.conf`

```
[tcpout:cribl]
server=127.0.0.1:<myPort>
```

Performance Considerations

Like most data processing applications, Cribl's expected resource utilization will be commensurate with the type of processing that is occurring. For instance, a function that adds a static ingest-time field on an event will likely perform faster than one that is applying a regex to finding and replace a string. At the time of this writing:

- Cribl processing will use about 2 CPUs (i.e. 4 vCPUs)
- Cribl processing happens in-memory
- Cribl processing does not require significant disk allocation.

Security Considerations

At the time of this writing:

- With the Cribl App for Splunk package, data flow from Splunk to Cribl is **confined to** `localhost:10000` and/or `localhost:10420`
- The control plane (UI/API) runs on port `9000` and it's authenticated either locally or against Splunk's admin role.

Standalone Deployment

Deployment guide to get you started with Cribl

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

When volume is low and/or amount of processing is light, you can get started with a single instance deployment. See performance considerations. To accommodate increased load, you will need to scale up and perhaps out with multiple instances.

Single Instance Deployment

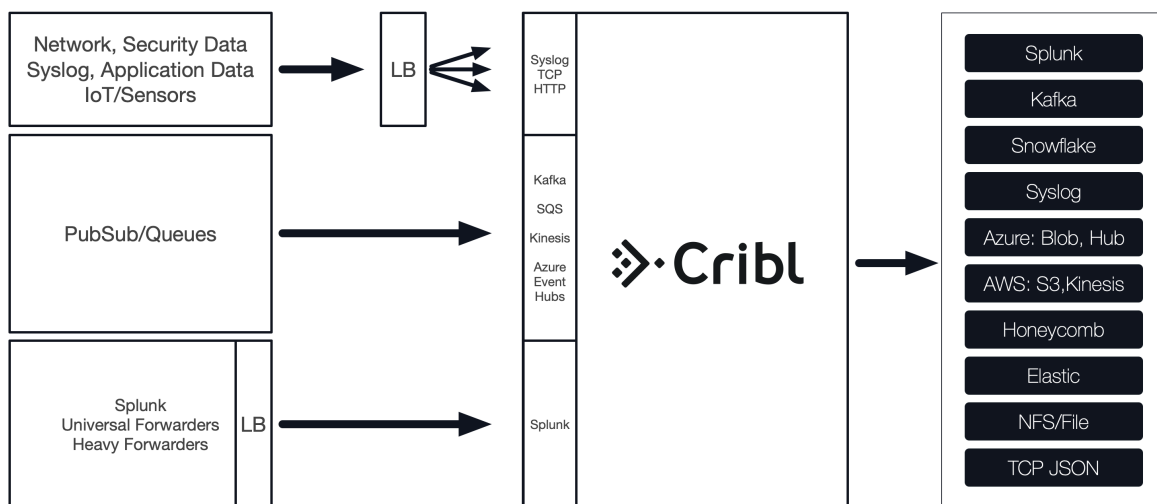
Distributed Deployment

Single Instance Deployment

Getting started with Cribl on a single instance.

For small volume or light processing environments or for test and evaluation use-cases a single instance of Cribl may be sufficient to serve all inputs, processing of events and sending to outputs without needing any others. To implement a single instance Cribl deployment see below.

Architecture



System Requirements

- **OS:** Linux (RedHat, CentOS, Ubuntu, AWS Linux), MacOS/Darwin
- **System:** +4CPUs, +4GB RAM

Note: 1 CPU here means a physical CPU core. I.e. 2 CPUs = 4 virtual/hyperthreaded CPUs

Installing on Linux/Mac

- Select an instance where to install and get the Cribl package here.
- Ensure that ports 10080 and 9000 are available. See here.
- Un-tar in a directory of choice, say, /apps/
 - e.g., `tar xvzf cribl-<version>-<build>-<arch>.tgz`

Running

Go to `$(CRIBL_HOME)` directory - this is where the package was extracted e.g. `/apps/cribl/` - and use `.bin/cribl.sh` to:

- **Start:** `./bin/cribl.sh start [--force]`
- **Stop:** `./bin/cribl.sh stop [--force]`
- **Reload:** `./bin/cribl.sh reload [--force]`
- **Restart:** `./bin/cribl.sh restart [--force]`
- **Get status:** `./bin/cribl.sh status`

Next, go to `http://<hostname>:9000` and login with default credentials (`admin:admin`) to start configuring Cribl with Sources, Destinations or start creating Routes and Pipelines.

Running on Linux with systemd

Sample systemd File

```
[Unit]
Description=Systemd service file for cribl.
After=network.target

[Service]
Type=forking
Restart=on-failure
RestartSec=5
PIDFile=/opt/cribl/pid/cribl.pid
ExecStart=/opt/cribl/bin/cribl.sh start
ExecStop=/opt/cribl/bin/cribl.sh stop
ExecStopPost='/bin/rm -f /opt/cribl/pid/cribl.pid'
ExecReload=/opt/cribl/bin/cribl.sh reload
TimeoutSec=60

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

Scaling Up

A single instance Cribl installation can be configured to scale up and utilize as many resources on the host as required. See [Scaling](#) section for more details.

Distributed Deployment

Getting started with Cribl on a distributed deployment.

Distributed Deployment

To sustain higher incoming data volumes and/or increased processing you can scale from a single instance to a multi-instance, distributed deployment. Instances in the deployment serve all inputs, process events and send to outputs independently. They are managed centrally by a single master node which is responsible for keeping configurations in sync and tracking & monitoring their activity metrics.

Concepts

Single Instance - a normal Cribl instance running by itself.

Master Node - a Cribl instance running on **master** mode used to centrally author configurations and monitor a distributed deployment.

Worker Node - a Cribl instance running as a **managed worker** whose configuration is fully managed by a Master Node.

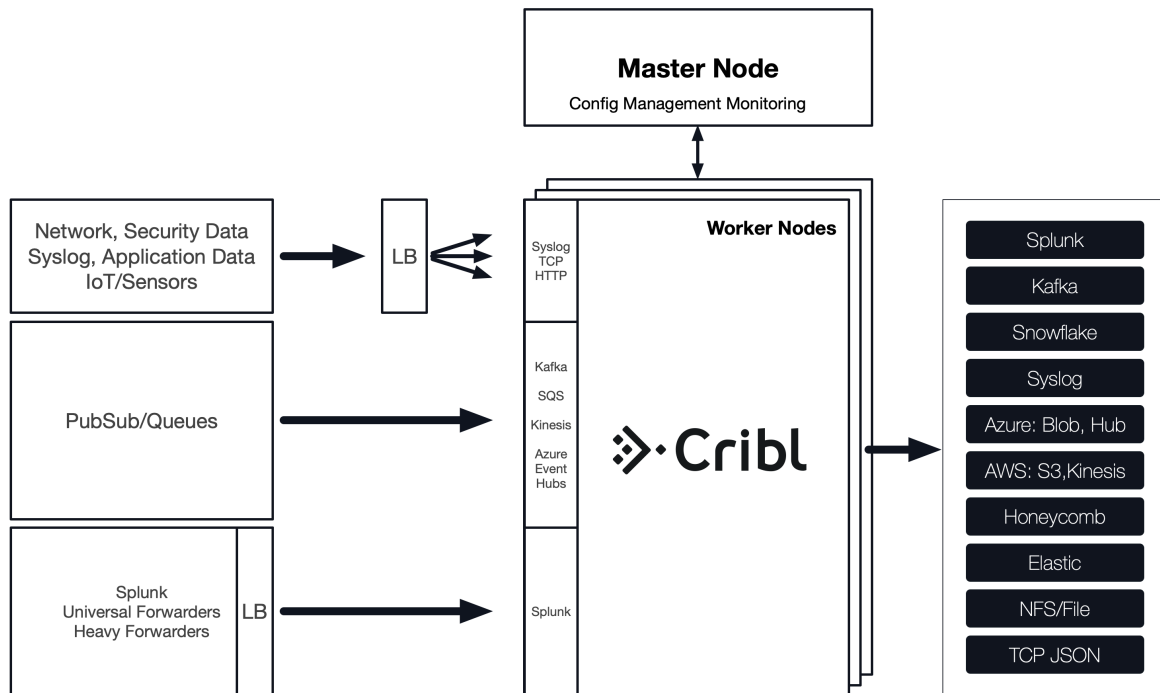
Worker Group - a collection of Worker Nodes that share the same configuration.

Worker Process - a process within a Single Instance or Worker Nodes that handles data inputs, processing and output

Mapping Ruleset - an ordered list of Filters used to map Workers to Worker Groups.

Note about Worker Nodes: The local running config of a Worker Node can be manually overridden/changed, but changes won't persist on the filesystem.

Architecture



Installing and Running

- See Single Instance Deployment as procedures are identical.
- In distributed deployment Workers communicate out to Master Node on port 4200 . Ensure that Master is reachable on that port.

i Git is required to be available on Master Node. See below for details.

Setting up Master and Workers Nodes

1. Configuring a Master Node

Using the UI:

In **Settings | Distributed Management**, select Mode **Master**, supply required Master settings: Address, Port (and optional settings if used), and restart.

Or, through `instance.yml` :

In `$CRIBL_HOME/local/_system/instance.yml` , under `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>
    compression: none
    connectionTimeout: 5000
    writeTimeout: 10000
```

2. Configuring a Worker Node

Through the UI:

In **Settings | Distributed Management**, select Mode **Worker**, supply required Master settings: Address, Port (and optional settings if used), and restart.

Or, through `instance.yml` :

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

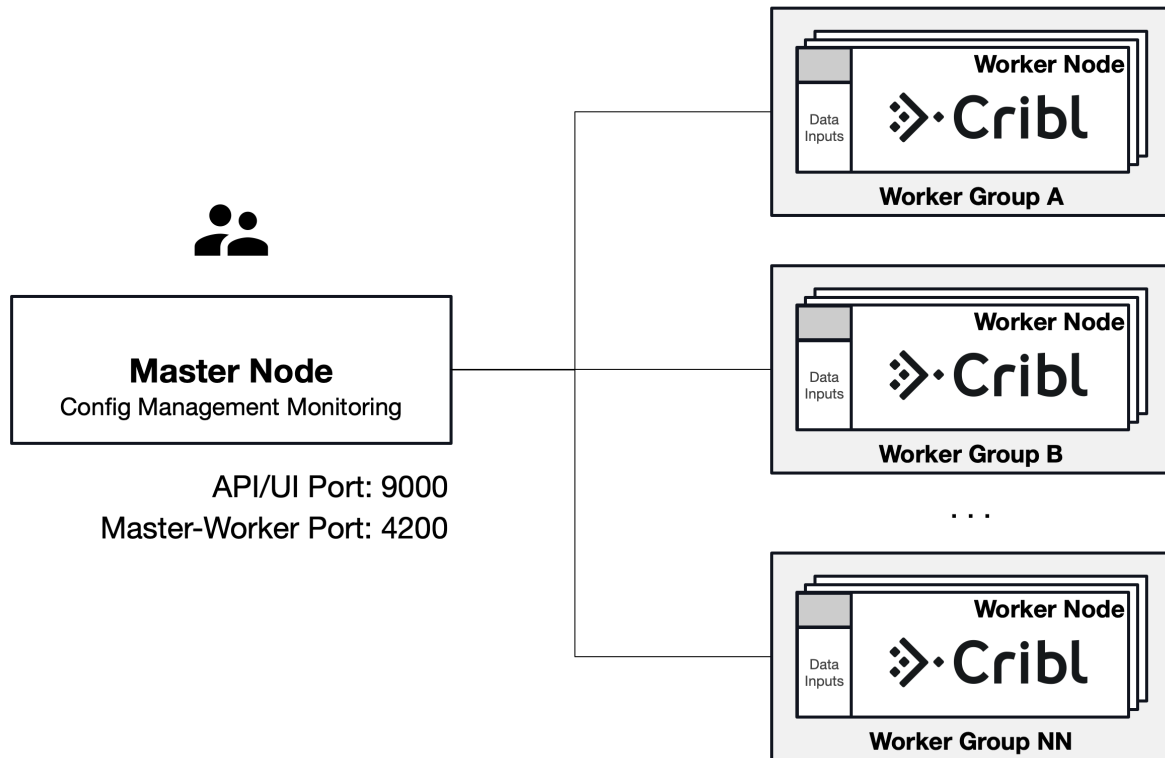
`$CRIBL_HOME/local/_system/instance.yml`

```
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
```

How do Workers and Master Work Together

The Master Node has two primary roles:

1. Serve as a central location for Workers' operational metrics. The Master ships with a monitoring console that has a number of dashboards covering almost every operational aspect the deployment.
2. Serve as a central location for authoring, validating, deploying and synchronizing configurations across Worker Groups.



Workers will periodically send a heartbeat to the Master which 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 Master tracks with the connection.

When a Worker Node checks-in with the Master:

- The Master uses Mapping Rules to find a Worker Group for each worker
- The Worker Node pulls its Group's updated configuration bundle

Config Bundle Management

Config bundles are compressed archives of all config files and associated data that a Worker needs to operate. The Master creates bundles upon Deploy and manage 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 Master 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, go to **Worker Groups** top level menu and click **+ Add New**.

Configuring a Worker Group

Clicking on the newly created group will present you with an interface for **authoring and validating** its configurations. You can configure everything for this Group as if it were a single Cribl instance - using the same exact visual interface for Routes, Pipelines, Sources, Destinations and System Settings.

Mapping Workers to Worker Groups

Mapping Rulesets are used to map Workers to Worker Groups. **Only one Mapping Ruleset can be active at any one time**. A ruleset is a list of rules that evaluate Filter expressions on the information that Workers send to the Master. The ruleset behavior is similar to Routes where the order matters and the Filter section support full JS expressions. The ruleset matching strategy is first-match and one Worker can belong to only one Worker Group. At least one Worker Group should be defined and present in the system.

Example: All hosts that satisfy this condition: IP address starts with `10.10.42` with more than 6 CPUs OR `CRIBL_HOME` environment variable contains `w0` , belong to **Group420**.

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

To create a Mapping Ruleset, go to **Mappings** top level menu and click **+ Add New**. Click on the newly created item and start adding rules by clicking on **+ Add Rule**. While working with or tuning rules, the Preview on 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 Master. To do that go to Mappings and click **Activate** on the required ruleset. You can also 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 it ranks in mapping priority.

When an instance runs as Master 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

The typical workflow for deploying configurations is the following:

1. **Work on Configs**
2. **Commit (and optionally Push)**
3. **Deploy**

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

Configuration Files

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

On the managed Worker, after configs have been pulled, lookups are extracted under: `$CRIBL_HOME/data/` .

Note: *some configuration changes will require restarts while many others only reloads. See here for details. Restarts/Reloads are handled automatically by the Worker.*

Using git

Git is required to be available locally on the host where the Master Node will run. **Configuration changes must be committed to git before they're deployed.** If you don't have it installed check here for details on how to get started.

To verify that `git` is available, run this command

```
git --version
```

and ensure that you get a response similar to this:

```
git version 2.20.1 (Apple Git-117)
```

The Master node uses `git` to:

- manage configuration versions across worker groups
- allow users to have an audit trail of all configuration changes
- allow users to display diffs between current and previous config versions

Support For Remotes

Git remote repositories are supported. You can configure git with a remote either from CLI or through the UI via Settings > Distributed Settings > Git Settings. When a remote is configured a `Git Push` button is displayed at the Global/Top Level config management.

Note, reverting back to a previous commit is only supported from the CLI.

Auto Scaling Workers and Load Balancing Incoming Data

If data flows in via Load Balancers make sure to register all instances. Each Cribl 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>

Splunk App Deployment

Deploying Cribl App for Splunk

In a Splunk environment, Cribl can be installed and configured as a Splunk app and depending on your requirements and architecture, it can run either on a Search Head, Heavy Forwarder (strongly advised) or an Indexer.

Running on a Search Head (SH)

When running on a SH, Cribl is set on **mode-searchhead**, the default mode for the app. It listens for **localhost traffic** generated by a custom command - `| criblstream`. The command is used to forward search results to the Cribl instance's TCPJSON input on port `10420` but it's also capable of sending to any other Cribl instance listening for TCPJSON. Once in Cribl, 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 single click.

Installing the Cribl App for Splunk on a SH

- Select an instance where to install
- Ensure that ports `10000`, `10420` and `9000` are available. See Before Deploying 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 login with a Splunk **admin** role credentials.

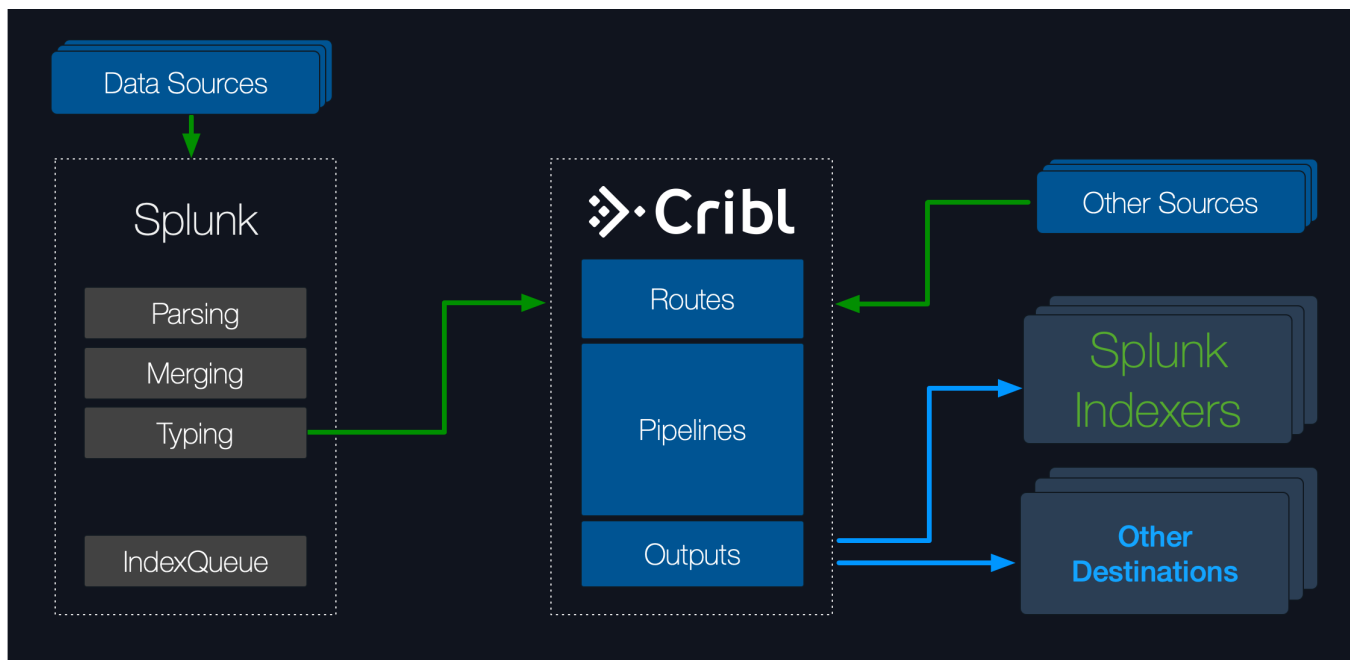
Typical Use Cases for Search Head mode

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

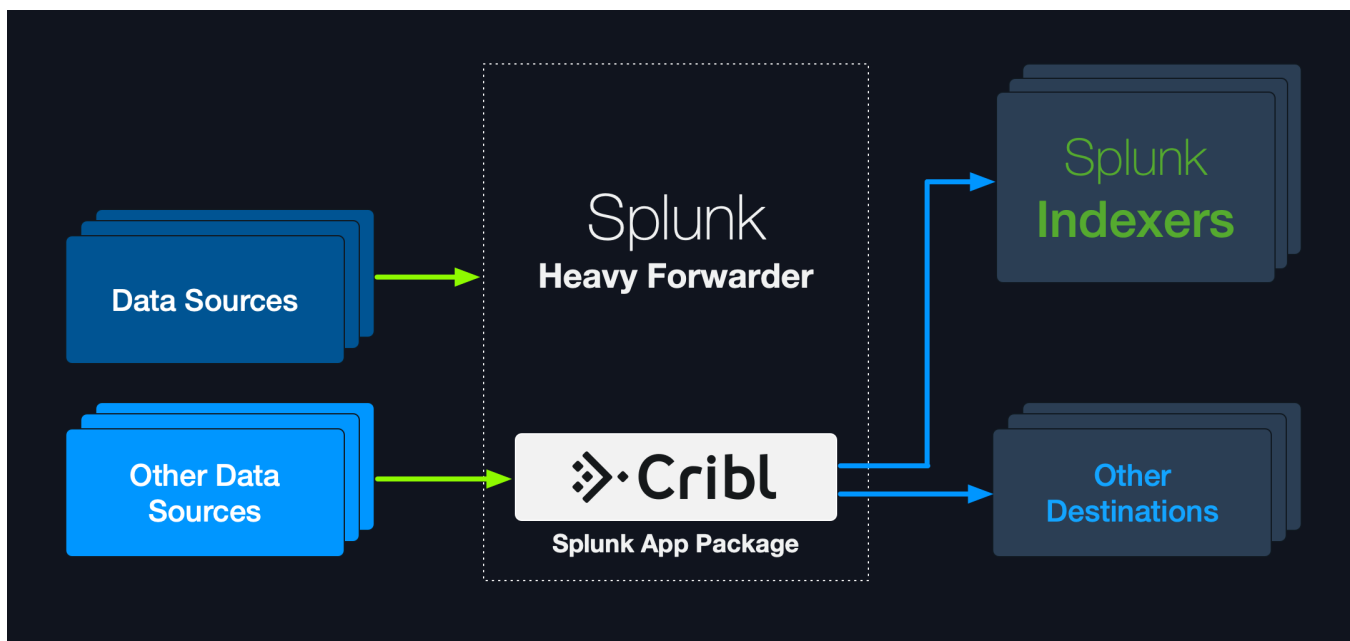
Running on a Heavy Forwarder (HF)

When running on an HF, Cribl is set on **mode-hwf**, and receives events from the local Splunk process per routing configurations in `props.conf` and `transforms.conf`. Data is first parsed and processed by Splunk

pipelines and then by Cribl. By default all data except internal indexes are routed out to Cribl right after the Typing pipeline.



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



Installing the Cribl App for Splunk on a HF

- Select an instance where 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 in **mode-hwf**: `$SPLUNK_HOME/etc/apps/cribl/bin/cribl mode-hwf`
 - **Note**: SPLUNK_HOME environment variable must be defined
- Restart the Splunk instance
- Go to `https://<instance>:9000` and login with a Splunk **admin** role credentials.

i Note about Splunk warnings

If you come across messages similar to below, on startup, or in logs:

```
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)
```

please ignore them. They are benign warns.

Relevant configurations in Cribl App for Splunk on a HF

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

apps/cribl/default/outputs.conf

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

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

apps/cribl/default/inputs.conf

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

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 with a subset of your data

The `props.conf` stanza above will apply the above transforms to **everything**. Depending on your requirements you may want to target 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 thru Cribl.

outputs.conf

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

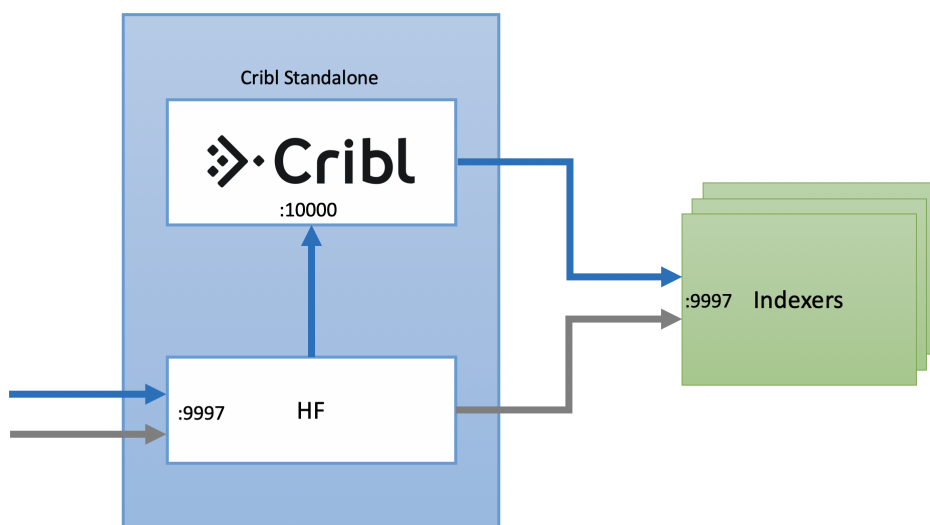
```
[tcpout:myIndexers]  
server=<list of indexers>  
...
```

props.conf

```
[<bluedata>  
TRANSFORMS-toCribl = route2cribl  
...
```

transforms.conf

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

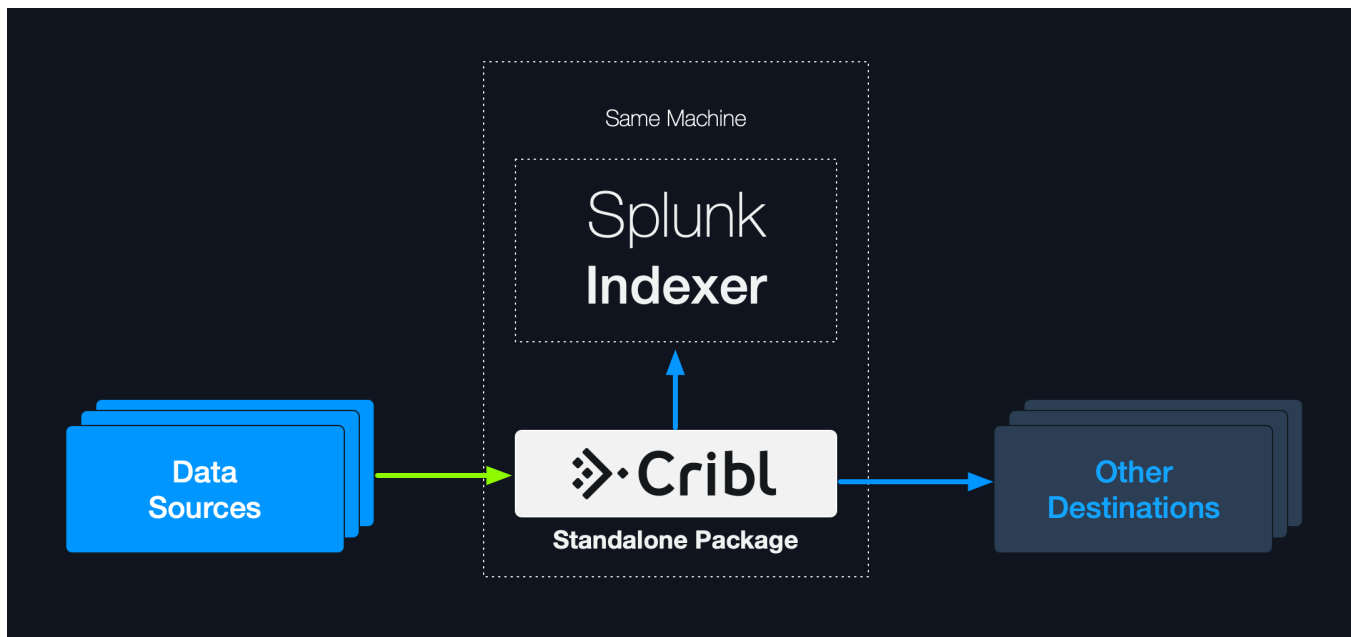


Configure Cribl to send data to Splunk Indexers

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

Running on a Indexer

Cribl can natively accept data **streams** (un-broken events) or **events** from sources. In this case, data comes **directly** into Cribl which processes it then sends it downstream, including the local Splunk indexer instance. This is exactly like a Standalone Deployment but using a Splunk Indexer instance as the host.



Scaling

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

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

Scale Up

A single instance Cribl installation can be configured to scale up and utilize as many resources on the host as required. Allocation of resources is governed thru Worker Processes Settings section in General Settings.

Memory (MB): Amount of memory available to each worker process, in MB. Defaults to 2048 .

Process Count: Indicates the number of worker processes to spawn. Each worker process will utilize up to 1 CPU core. Negative numbers can be used to tie the number of workers relative to the number of CPUs in the system. Any setting less than 1 is interpreted as $\text{number of CPUs} - \text{setting}$.

For example, assuming a system with 12 CPUs:

- If **Process Count** is set to 4 , then Cribl will spawn 4 processes, using up to 4 CPU cores, leaving 8 free.
- If **Process Count** is set to -2 , then Cribl will spawn 10 processes (12-2), using up to 10 CPU cores, leaving 2 free.

 There are guardrails in place that **prevent** spawning more processes than CPU cores.

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

1. If data comes into a single socket then it will be processed by a single process. I.e. **to get the full benefits of multiple worker processes it's important that data comes into from multiple sockets.**
E.g., it is better to have five connections each bringing in 200GB/day than one doing 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.

Scale Out

When data volume, processing needs or other requirements exceed those that a single instance can sustain, a Cribl 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 section for more details.

Config Files

Understanding Configuration Paths and Files

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

`$CRIBL_HOME`

Standalone Install:

`/path/to/install/cribl/`

Splunk App Install:

`$SPLUNK_HOME/etc/apps/cribl/`

All paths below relative to `$CRIBL_HOME`

Default Cribl Configuration	<code>default/cribl</code>
Local Cribl Configuration	<code>local/cribl</code>
Cribl Configuration	<code>(default local)/cribl/cribl.yml</code> See <code>cribl.yml</code>
API Configuration	<code>(default local)/cribl/api.yml</code>
Source Configuration	<code>(default local)/cribl/inputs.yml</code> See <code>inputs.yml</code>
Destination Configuration	<code>(default local)/cribl/outputs.yml</code> See <code>outputs.yml</code>
License Configuration	<code>(default local)/cribl/licenses.yml</code>
Regexes Configuration	<code>(default local)/cribl/regexes.yml</code>
Breakers Configuration	<code>(default local)/cribl/breakers.yml</code>
Limits Configuration	<code>(default local)/cribl/limits.yml</code>
Pipelines Configuration	<code>(default local)/cribl/pipelines/<pname></code> Each pipeline's conf is contained therein
Routes Configuration	<code>(default local)/cribl/pipelines/routes.yml</code>

Functions	(default local)/cribl/functions/<function_name> Each function's code, conf is contained therein
Functions Conf	(default local)/cribl/functions/<function_name>/... Each function's conf contained therein.

Configurations and Restart

- Any configuration changes resulting from UI interactions, for instance, changing the order of functions in a pipeline, or changing the order of routes, **do not require restarts**.
- All Cribl configuration file changes resulting from direct file manipulations in (bin|local|default)/cribl/... **will require restarts**.
- In the case of a Cribl App for Splunk, Splunk configurations file changes may or may not require restarts. Please check with recent Splunk docs.

Configuration Layering and Precedence

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

! Editing Configuration Files Manually

When config files **must** be edited manually, all changes should be done in `local`.

cribl.yml

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

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

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


inputs.yml

inputs.yml contains settings for configuring inputs into Cribl.

\$CRIBL_HOME/default/cribl/inputs.yml

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

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

outputs.yml

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

\$CRIBL_HOME/default/cribl/outputs.yml

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

```
bucket: s2.bucket.address.here
# Prefix to append to files before uploading
destPath: keyprefix
# AWS API key, if not present will fallback on env.AWS_ACCESS_KEY_ID, or the meta-data enc
awsApiKey: key
# AWS Secret Key. If left blank, Cribl will fallback on env.AWS_SECRET_ACCESS_KEY, or the
awsSecretKey: secretkey
# Staging destination path. Writable by Cribl.
stagePath: /tmp/foo
# Partition schema for outputted files
partitionExpr: >-
  `${host}/${sourcetype}`
# Format of the output data
format: json
# The output filename prefix
baseFileName: CriblOut
# Compression options. None | Gzip
compress: none
# Maximum uncompressed output file size
maxFileSizeMB: 32
# Maximum amount of time to keep inactive files open.
maxFileOpenTimeSec: 300
# Maximum amount of time to keep inactive files open.
maxFileIdleTimeSec: 30
# Maximum number of files to keep open concurrently.
maxOpenFiles: 100
```

licenses.yml

licenses.yml maintains a list of licenses for Cribl.

\$CRIBL_HOME/default/cribl/licenses.yml

licenses:

List of license keys

- eyJ0eXAiOiJKV1QiLCJhasdfasfasdfdasfasdfa-Abo2_ogVbR_5VKeAe1Z1Tc5b-TKQax9R1ywno0G8guis2RC0s

regexes.yml

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

\$CRIBL_HOME/default/cribl/regexes.yml

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

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

  xoxa-23984754863-2348975623103

  xoxb-23984754863-2348975623103

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


breakers.yml

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

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

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

mappings.yml

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

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

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

```

instance.yml

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

`$CRIBL_HOME/local/_system/instance.yml`

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

Licensing

Every Cribl version ships with a **Free** license that allows for processing of up to 100GB/day on a single node deployment. Free licenses are restricted to one node per email address, and will require phoning home with telemetry metadata (see below). **Sales Trial** and **Enterprise** licenses do not require phoning home, are not restricted to a single node, and are entitled to a certain amount of daily ingestion volume.

Licenses can be managed in **Settings | Licensing**.

✔ The latest license expires on: 2019-11-30T12:00:00+00:00

License Types

Enterprise License

This is a Cribl standard license available for purchase. Contact Cribl Sales at sales@cribl.io for more information.

Sales Trial License

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

Free License

A license type that allows for processing of up to 100GB/day on a single node deployment. Free licenses ship with the download, are restricted to one node per email address, and will require phoning home with telemetry metadata.

Combining License Types

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

- If there are any unexpired Enterprise licenses - use only Enterprise 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 licenses - use only free licenses to compute the effective license

Upon an Enterprise license expiration, system will fallback to Sales Trial and Free types to compute a future expiration date. An expired Sales Trial license cannot use Free to fallback to.

i License Expiration Behavior

Upon license expiration, Crill will backpressure and block all incoming data.

Telemetry Data

If you are on the **Free** license, your instance will periodically share usage and deployment performance **metadata** with Cribl. The data will be sent to phonehome.cribl.io and Cribl will use it **only** to make decisions about product development and improved customer experience.

If you would like this feature disabled in order to deploy on your environment, please reach out to Cribl Sales at sales@cribl.io, and we will work with you to issue another license that does that.

Data Shared Per Interval (roughly very minute):

Version

Instance's GUID

Earliest, Latest Time

Number of Events In, Out

Number of Bytes In, Out

Number of Open, Closed, Active Connections

Number of Routes

Number of Pipelines

User Authentication

Cribl supports both **local**, **Splunk** and **LDAP** authentication.

Local Authentication

Local user management in Cribl is done through **Settings > Local Users**. All changes made to users are persisted in a file located in `$CRIBL_HOME/local/cribl/auth/users.json`.

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

Line format:

```
{"username":"user","first":"Elvis","last":"Bath","disabled":"false",  
"passwd":"GIBBERISH/aPNs8RtU5o9Lu2WE0j17XUA="}
```

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

To manually add/change a password just add a `password` key as such: `"password":"plainText"`, in the corresponding line. The plaintext passwords will be hashed during the next file's reload and the `password` key will be deleted.

To setup local authentication:

Navigate to **Settings > General Settings > Authentication Settings** and select **Local**

cribl.secret file

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

 **Backup and secure access to this file by applying strict permissions. E.g. 600.**

Splunk Authentication

This is very helpful when deploying alongside Splunk and authentication is only available to users with Splunk `admin` role. To setup Splunk authentication:

Navigate to **Settings > General Settings > Authentication Settings** and select **Splunk**

- Host: Splunk host address (typically a search head)
- Port: Splunk management port (defaults to `8089`)

- SSL: Set to `Yes` if enabled
- Fallback to local: Attempt local authentication if Splunk authentication is unsuccessful. Defaults to `false`.

LDAP Authentication

LDAP authentication is supported and can be setup as follows:

Navigate to **Settings > General Settings > Authentication Settings** and select **LDAP**

- Secure: Enable to use a secure ldap connections (`ldaps://`). Disable for unsecure (`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.
- Search Base: Starting point to search LDAP for users, e.g., `'dc=example,dc=org'`
- Search Field: LDAP user search field, e.g.: `cn` or `uid` .
- Fallback to Local: Attempt local authentication if LDAP authentication is down or mis-configured. Defaults to `No` .
- Reject Unauthorized: Valid for secure LDAP connections, set true to reject unauthorized server certificates.

Persistent Queues

Persistent queuing is a feature that helps minimize data loss if a downstream receiver (output) is unreachable. Durability is provided by writing data on disk for the duration of the outage.

How does Persistent Queueing Work

Each output has an in memory queue that helps it absorb temporary imbalances in inbound and outbound data rates. E.g., if there is an inbound burst of data the output will store events in the queue and output them at the rate that the receiver can sink, as opposed to blocking or dropping them. Only when this queue is full the output will backpressure upstream. Backpressure behavior can be configured to either **block** or **drop**. In block mode the output will refuse to accept new data until the receiver is ready. The system will back propagate block "signals" all the way to the sender (assuming they support backpressure, too). In drop behavior, the output will drop new events until the receiver is ready.

While in some environments the in memory queues and their block/drop behavior are acceptable, in others where more durability is required (i.e. outages last longer than memory queues can sustain), or when upstream senders do not support back pressure (e.g. ephemeral/network senders), persistent queues can be engaged to help minimize data loss. In this case, once the in-memory queue is full, the output will write its data to disk, then, when the receiver is ready, it will start draining the queues (in first in, first out).

Persistent Queues Are:

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

Using Persistent Queueing

Persistent Queueing is available only for certain streaming destinations. Non-streaming destinations, such as Filesystem or S3 have their own inherent resilience and do not support Persistent Queueing.

Destinations that support persistent queuing

Splunk

Syslog

TCP JSON

Configuring Persistent Queueing

Persistent Queueing is configured individually for each output that supports it. To enable it, go to output's configuration page and select **Persistent Queueing** under **Backpressure Behavior**.

Max File Size: The maximum size to store in each queue file before closing and optionally compressing (KB, MB, etc). Defaults to 1 MB .

Maximum Queue Size: The maximum size amount of disk space the queue is allowed to consume. Once reached, queueing is stopped and backpressure is applied (KB, MB, etc).

Queue File Path: The location for the persistent queue files. Will be of form `your/path/here/<worker-id>/<output-id>` . Defaults to `$CRIBL_HOME/state/queues` .

Compression: Codec to use to compress the persisted data. Defaults to None.

Minimum Free Disk Space

Sufficient disk space is required for queuing to operate properly. Minimum disk space is configured in [Settings | System Settings | Limits | Min Free Disk Space](#).

Securing and Monitoring

Securing

Cribl's API/UI access can be secured by configuring SSL. You can use your own private keys and certs or you can generate a pair with OpenSSL:

```
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.

Key and Cert can be configured via Settings > System Settings > API Server Settings. Alternatively, you can manually use **privKeyPath** and **certPath** attributes in the `api` section in `local/cribl.yml`. E.g.,

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
...
```

Monitoring (ToDo)

To get an operational posture of a single instance deployment the following can be used:

- **Stats Page:** exposes information about traffic in and out of the system. It tracks events, bytes, split by data fields over time, and a range of system metrics.
- **Cribl.log:** contains comprehensive information about the status of the instance, its inputs, outputs, pipelines, routes, functions and traffic metrics.

In a distributed deployment, all Workers forward their metrics to Master which then natively consolidates them to provide a deployment-wide view.

Forwarding Cribl's internal data to your preferred log and metrics monitoring solution is also supported. To send internal data out of Cribl, go to **Sources** and enable **Cribl Internal**. This will send `cribl.log` down the routes and pipelines just like another data source.

In addition, each Cribl instance exposes a health endpoint - typically used in conjunction with a Load Balancer - that can be used to make operational decisions.

Health Check Endpoint	Healthy Response
curl http://<host>:<port>/api/v1/health	{"status":"healthy"}

Upgrading

Except for upgrading to a major version, or from a Beta version and to its GA successor, upgrades for both packages are done as below. (Direct upgrades from a Beta to a GA version are **not** supported. To get the GA version running, a new install is required.)

i Supported Upgrade Paths

v1.7.x ==> v2.0

v1.6.x or below ==> v1.7.x ==> v2.0

Standalone Package Upgrade Steps

- Stop Cribl process
- Untar/unzip the new version on top of the old one
- Restart

Splunk App Package Upgrade Steps

i See special note below if upgrading to v1.7.

- Stop Splunk
- Untar/unzip the new app version on top of the old one
- Restart

Special Note: Upgrading Splunk App Package to v1.7

Contrary to prior versions, in v1.7 the Splunk App package defaults to Search Head Mode. If you have Cribl deployed as a Heavy Forwarder app then follow these steps to upgrade.

- Stop Splunk
- Untar/unzip the new app version on top of the old one

- Convert to HF mode by running: `$SPLUNK_HOME/etc/apps/cribl/bin/cribl mode-hwf`
- Restart

Diagnosing

Cribl system configuration as well as recent log output is accessible from the UI through **Settings** (top right) | **Diag.**

System Info:

Running system information including but not limited to:

- Cribl Build Versions
- System: Uptime | Memory | CPU | Network

Pipelines:

Full, running configuration of all Pipelines.

Route Configuration:

Full running configuration of all Routes.

Inputs:

Input configurations.

Outputs :

Outputs configurations.

cribl.log Logs:

Most recent log messages emitted by Cribl.

Diag Bundle

To create a diag bundle, click on Export Diag Bundle and all the relevant configuration and recent logs will be archived and downloaded to your local machine. Share this bundle with Cribl team when trying to troubleshoot a problem. **Please make sure that all sensitive configuration data (such as API keys, secrets etc) are scrubbed before sharing.**

Uninstalling

Uninstalling the Standalone version

- Stop Cribl (stopping main process)
- Backup necessary configurations/data
- Remove the directory where Cribl is installed

Uninstalling the Splunk App version

- Stop Splunk
- Backup 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 CRIBL

Routes

What are Routes

Before incoming events are transformed by a processing pipeline, Cribl uses a set of filters to first select a **subset** to deliver to the correct pipeline. This process is done via routes.

How do Routes Work

Routes apply filter expressions on incoming events to send matching results to the appropriate pipeline. Filters are JS-syntax compatible expressions, e.g., `source=='foo.log' && fieldA=='bar'`, `true`, etc. that are configured with each route. There can be multiple routes in the system but a route can only be associated with **one** pipeline.

i Routes are evaluated in order, top down.

#	Name	Description/Filter	Pipeline	% Events	
1	Route	Route data to other systems	route	17.498%	On
2	Sensitive Data	Mask Sensitive Data	masking	11.251%	On
3	Logs to Metrics	Convert Access Combined Logs to Splunk Metrics	logs_to_metrics	10.001%	On
4	Sample and Filter	Clean up voluminous weblog data for our needs	sample_and_filter	20.002%	On
5	Enrich	Use lookups to add context to events	enrich	20.002%	On
6	JWT Decode	Send auth failures with JWT to decode JWT	jwt_to_json	3.746%	On
7	Main Route	Catchall for routing data through main pipeline	main	17.498%	On

In this example, incoming events will be evaluated against the route named **Route** first, then **Sensitive Data**, then **Logs to Metrics** and so on. At the end, the **Main** route serves as a catch-all for any event that does not

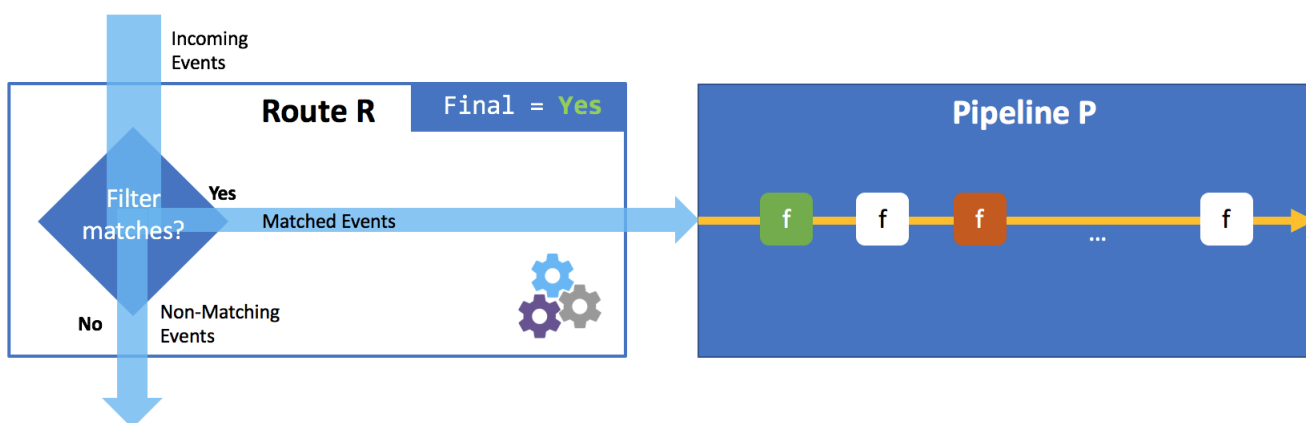
match any of the other routes. If a route needs to be applied before another, simply drag it on top of it. In addition, you can turn routes On/Off inline as necessary.

Output Destination

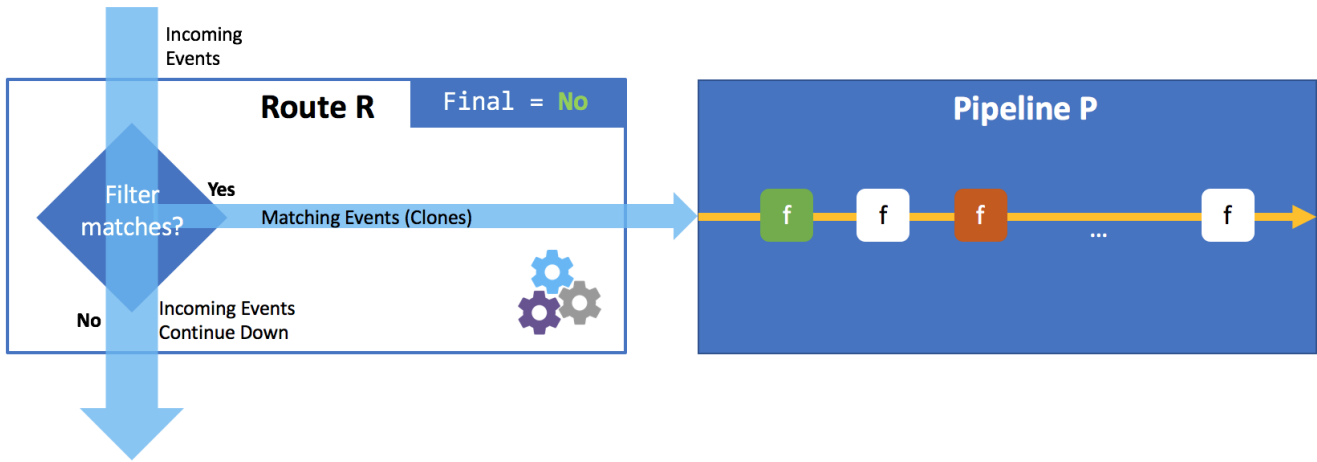
Routes can be configured with an output destination which denotes where to send events after they're processed by the pipeline. This destination **overrides** the one set at the pipeline level.

The `Final` Toggle

An event that enters the system and matches a route-pipeline pair in most cases it will either be dropped by a function or optionally transformed and exit the system. This is ensured by the `final` toggle in route settings. It defaults to `Yes` and means that matched events will be **consumed** by that route and not evaluated against any other routes that sit below it.



If the toggle is set to `No`, clone(s) of the matching events are processed by the configured pipeline and the original events are allowed to continue their trip downstream to be evaluated and/or processed by other route-pipeline pairs.



This is very useful in cases where the same set of events needs to be processed differently and delivered to different destinations. Each clone can be decorated with KV pairs as necessary.

Final Flag and Cloning Considerations

Depending on your **cloning** needs you may want to follow a **most specific first** or **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 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 may 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 list of routes and they are a UI artifact only - i.e. while in a group routes maintain their global position order.

Pipelines

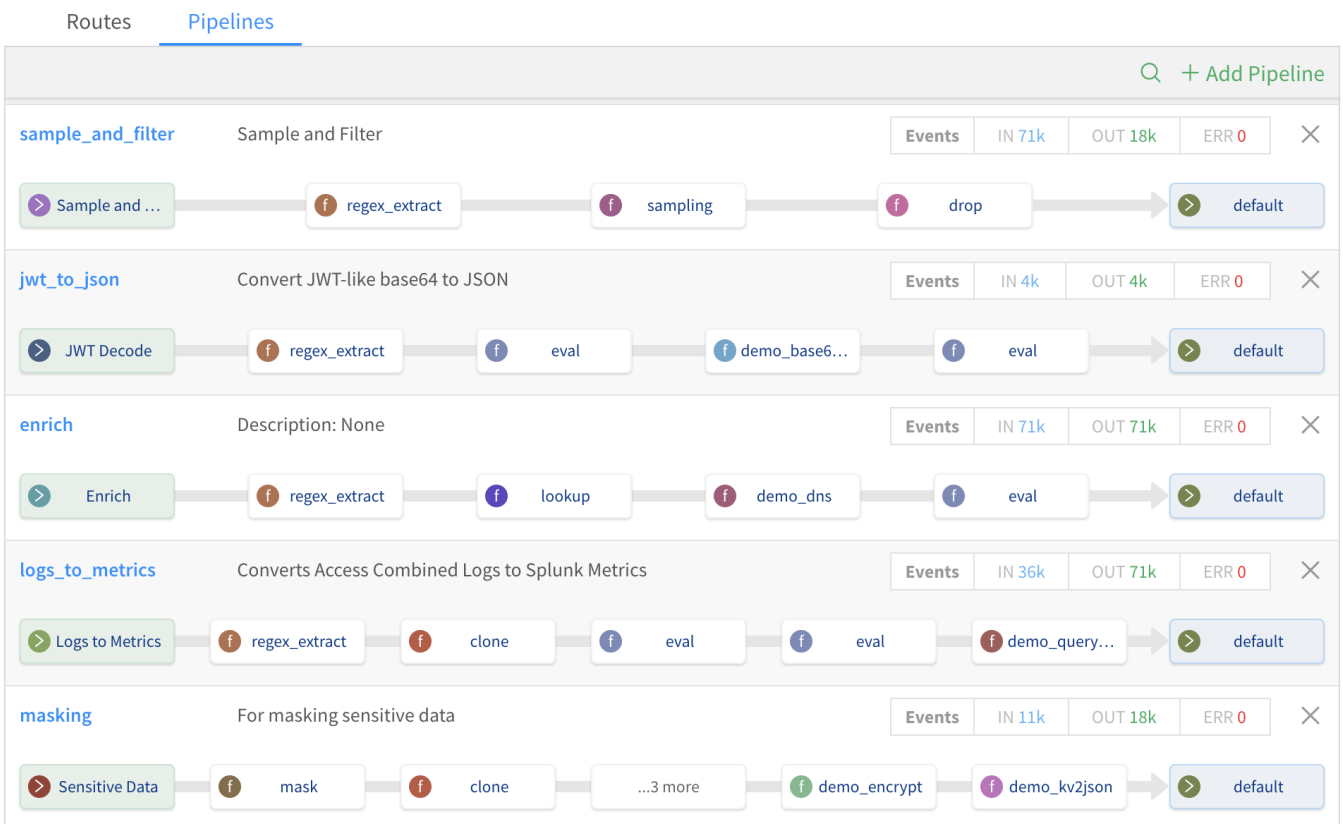
What are Pipelines

After the data has been matched by a route it gets delivered to a pipeline. A pipeline is set of functions that work on the data and that are composed in a very specific list. Similar to routes, the order in which the functions are listed matters.

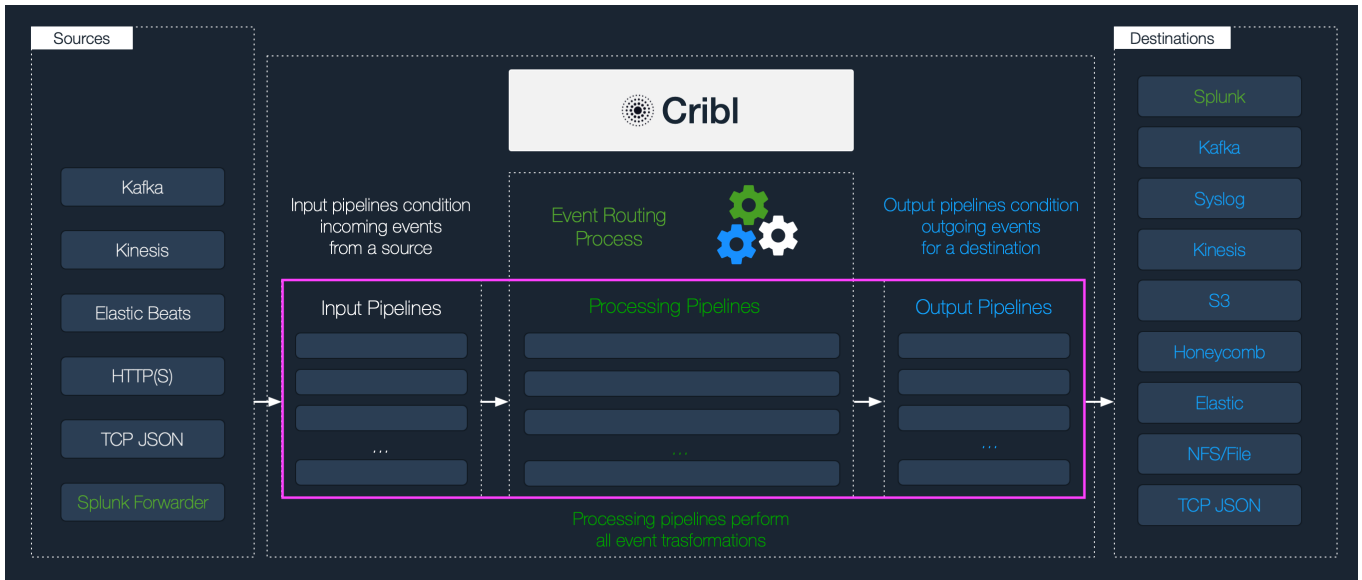
i Functions in a pipeline are evaluated in order, top down.

How do Pipelines Work

Events are always delivered at the beginning of a pipeline via a route . They are processed by each function, in order. A pipeline of chained functions will always move events in the direction that points outside of the system. This is on purpose so as to keep the design simple and avoid potential loops.



Types of Pipelines



Input Pipelines

These are pipelines that are attached to a Source (or Input) for the purposes of conditioning the events **before** they're delivered to a Processing Pipeline. They are optional and typical use cases are event formatting or when applying functions to *all* events of that input . E.g. extract the `message` field from all Elastic Sources before pushing events to various processing pipelines.

Processing Pipelines

These are the classic event processing pipelines.

Output Pipelines

These pipelines that are attached to a Destination (or Output) for the purposes of conditioning the events **before** they're sent out. Typical use cases are applying functions that transform or shape events per receiver requirements. E.g., ensure that a `_time` field exists for all events bound to a Splunk receiver.

Destination Selection in Processing Pipelines

Pipelines can be configured with an output destination but it is considered a best practice to define the destination at the route level instead. This makes the pipeline independent and reusable. (Note that destinations defined at route level **overrides** those at the pipeline level).

Other Considerations

Functions in a pipeline are equipped with their own filters. Even though they're not required, it advised that they're used as often as possible. Similar to 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, f_1 and f_2 and if f_1 operates on source 'foo' and f_2 that operates on source 'bar' it may make sense to apply source=='foo' and source=='bar' filters on each one respectively.

Functions

What are Functions

When events enter a pipeline they're processed by a series of functions therein. 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. We using the term "processing" here to mean a variety of possible options; from string replacement, to 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 last 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 JS code that are invoked on each event that passes thru them. To help improve performance, functions can be configured with filters to further scope their invocation on 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 thru. In addition, you can turn functions On/Off inline as necessary.

The screenshot shows the Splunk Pipelines configuration interface for a pipeline named "sample_and_filter". The interface is divided into several sections:

- Pipeline Overview:** Shows the pipeline name "sample_and_filter" and options to "+ Add Function" and "Show All".
- Function List:** A table listing functions in the pipeline:

#	Name	Filter	Toggle	Close
1	Regex Extract	source=='smart_sample'	On	X
2	Sampling		On	X
3	Drop	source=='filter' && !(/\"S+\s\S+action=purchase/i.test(_raw))	On	X
- Function Configuration (for the selected 'Drop' function):**
 - Filter:** A text input field containing the filter expression `source=='smart_sample'`.
 - Description:** An empty text input field.
 - Final:** A toggle switch set to "No".
 - Sampling Rules:** A table with columns "Filter" and "Sampling Rate".

Filter	Sampling Rate
<code>__status == 200</code>	5

The Final Toggle

Similar to the `Final` toggle in routes, the `Final` toggle here controls the flow of events at the function level.

- `Off` (**default**): means that matching events processed by this function will be passed down to the next function in the pipeline.
- `On` : means that this function is the last one that the matching events will be applied to. All others coming down the pipeline will be skipped.

Out of the Box Functions

Cribl ships with several functions out of the box and you can chain them together to meet your requirements. Expand the list of **Functions** on the left and the Use Cases section for more details.

Custom Functions

At the time of this custom functions are not yet supported.

Auto Timestamp

Description

The `Auto Timestamp` function extracts time to a destination field given a source field in the event.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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`.

Source Field: Field to search for a timestamp. Defaults to `_raw`.

Destination Field: Field to place extracted timestamp in. Defaults to `_time`. Nested addressing supported.

Default Timezone: Timezone to parse timestamps lacking timezone info. Defaults to `Local`.

Advanced Settings

Time Expression: Expression to use to format extracted time. Current time, as a Javascript Date object, is in global `time`. Defaults to `time.getTime() / 1000`.

Max Timestamp Scan Depth: Maximum string length where to look for a timestamp.

Additional Timestamps: Add Regex/Strptime pairs to extract additional timestamp formats.

- **Regex:** Regex with first capturing group matching the timestamp.
- **Strptime Format:** Timestamp in strptime format.

Format Reference:

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

```
%a - abbreviated weekday name.*
%A - full weekday name.*
%b - abbreviated month name.*
%B - full month name.*
%c - the locale's date and time, such as %x, %X.*
%d - zero-padded day of the month as a decimal number [01,31].
%e - space-padded day of the month as a decimal number [ 1,31]; equivalent to %_d.
%f - microseconds as a decimal number [000000, 999999].
```


%H - hour (24-hour clock) as a decimal number [00,23].
%I - hour (12-hour clock) as a decimal number [01,12].
%j - day of the year as a decimal number [001,366].
%m - month as a decimal number [01,12].
%M - minute as a decimal number [00,59].
%L - milliseconds as a decimal number [000, 999].
%p - either AM or PM.*
%Q - milliseconds since UNIX epoch.
%s - seconds since UNIX epoch.
%S - second as a decimal number [00,61].
%u - Monday-based (ISO 8601) weekday as a decimal number [1,7].
%U - Sunday-based week of the year as a decimal number [00,53].
%V - ISO 8601 week of the year as a decimal number [01, 53].
%w - Sunday-based weekday as a decimal number [0,6].
%W - Monday-based week of the year as a decimal number [00,53].
%x - the locale's date, such as %-m/%-d/%Y.*
%X - the locale's time, such as %-I:%M:%S %p.*
%y - year without century as a decimal number [00,99].
%Y - year with century as a decimal number.
%Z - time zone offset, such as -0700, -07:00, -07, or Z.
%% - a literal percent sign (%).

Directives marked with an asterisk (*) may be affected by the locale definition.

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 empty - 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]$`.

Aggregate(s): 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 labelled `<aggFunction>_<fieldName>`. If there are conflicts, the last one wins. For example, given two aggregates; `sum(bytes).where(action=='REJECT')` and `sum(bytes)`, the latter one, i.e. `sum_bytes` is the winner.

Group by Fields: Fields to group aggregates by.

Evaluate Fields: Set of key-value pairs to evaluate and add/set. Fields are added in context of an aggregated event, before they're sent out. Does not apply to passthru events.

Time Window Settings

Cumulative Aggregations: Determines if the aggregations should be reset to 0 or retained for cumulative aggregations when flushing out an aggregation table event. Defaults to `No`.

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

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

Output Settings

Passthrough Mode : Determines whether or not to passthrough the original events along with the aggregation events. Defaults to No .

Sufficient Statistics Mode: Determines whether or not to output only the sufficient statistics for the supplied aggregations. Defaults to No .

Metrics Mode: Determines whether or not to output aggregates as metrics or events. Defaults to No .

Advanced Settings

Aggregation Event Limit: The maximum number events to include in any given aggregation event. Defaults to unlimited.

Aggregation Memory Limit: The memory usage limit to impose upon aggregations. Defaults to unlimited (i.e. amount of memory in system).

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.

`max(expr:FieldExpression)` : Returns the maximum value of the 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.

`variance(expr:FieldExpression)` : Returns the sample variance of the values of the parameter.

`variancep(expr:FieldExpression)` : Returns the population variance of the values of the parameter.

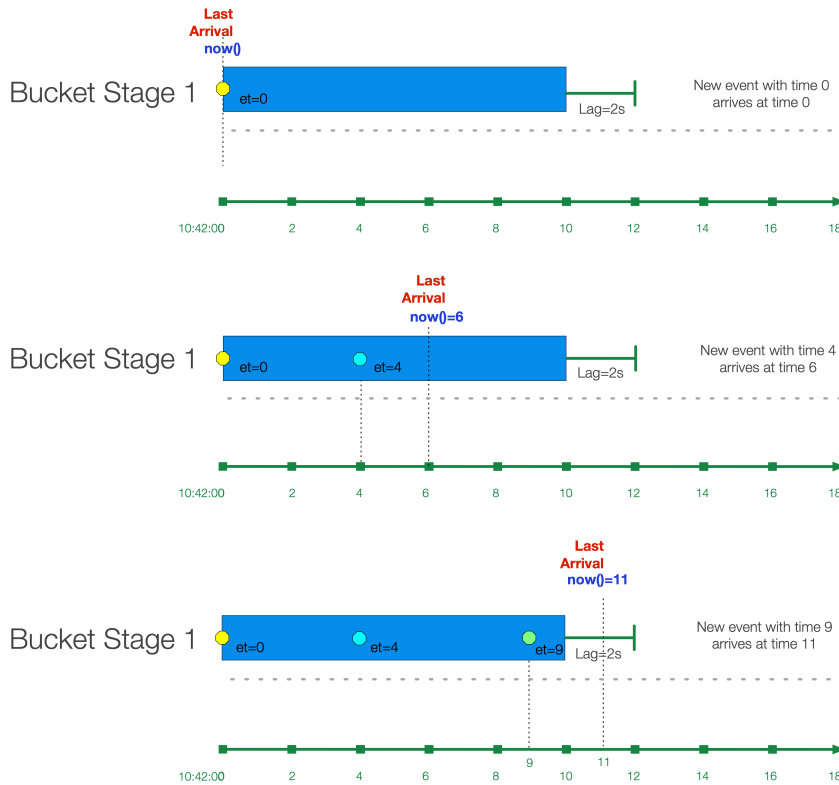
How do time window settings work?

Lag Tolerance

As events are aggregated into windows there is a good chance that most will arrive later than their event time. For instance, given a 10s window 10:42:00 - 10:42:10 an event with timestamp 10:42:03 may 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 may arrive at 10:42:12. Lag Tolerance is the setting that governs how long to wait, after the latest window boundary and still accept late events.

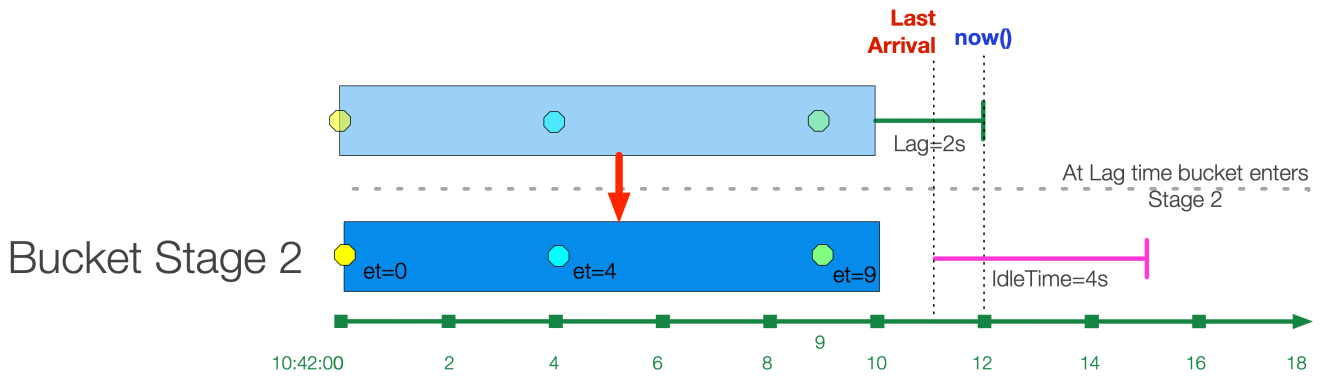
10s Window Aggregation

Settings: Lag=2s, IdleTime=4s



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

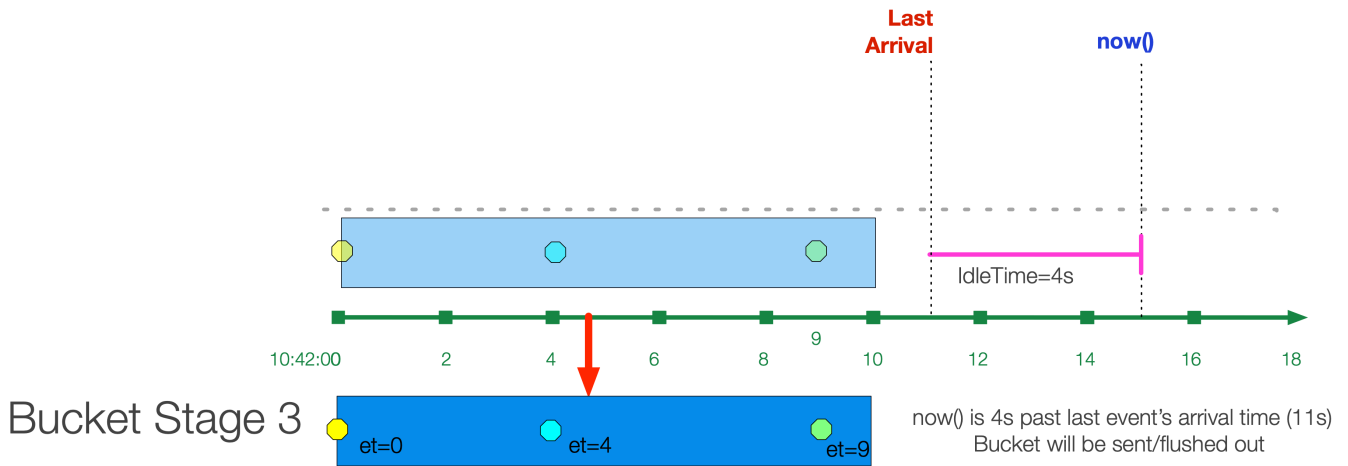
After Lag time expires, bucket moves to Stage 2.



If the bucket is created from a historic stream then 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 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 send 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')`

CEF Serializer

Description

The `CEF Serializer` takes a list of fields and/or values and formats them in Common Event Format (CEF) standard.

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 empty - 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. 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. Fields names and values will be written in `key=value` format. Names are selected from dropdown and values can be computed with JS expression or can be constants.

Clone

Description

The `clone` function clones events with optional added fields.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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 following fields set

Fields: Set of key-value pairs to add. Nested addressing supported.

Examples (coming soon)

Comment

Description

The `Comment` function adds a text comment in the pipeline

Usage

Comment: Text input field to add comment.

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 empty - 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`.

Dynamic Sampling

Description

The `Dynamic Sampling` function filters out events based on an expression, a sample mode and volume.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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`.

Sample Mode: Defines how sample rate will be derived. Supported methods:

- Square Root: `sqrt(previousPeriodCount)`
- Logarithmic: `log(previousPeriodCount)`. Defaults to Logarithmic.

Sample Group Key: Expression used to derive sample group key. For example: `${domain}:${statusCode}`. 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 num events received for a sample group is less than min a sample rate of 1:1 is used. Defaults to `30`.
- **Max Sampling Rate.** Maximum Sampling rate. If computed sampling rate is above this value it will be clamped down to it.

How does dynamic sampling work

Compared to static sampling where users must select a sample rate apriori, Dynamic Sampling allows for **automatically adjusting** sampling rates based on incoming data volume per sample group. The function allows users to only set the aggressiveness/coarseness of this adjustment. Square Root is more aggressive than Logarithmic setting.

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}`` it will be associated with `1.2.3.42:1234` sample group.

Note: If Sample Group Key is left at default ``${host}`` 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 defaults to 30 seconds). Once Sample Period seconds have elapsed, a sample rate will be derived based on the configured **Sample Mode** using sample group's event volume during the **previous** sample period.

For example, assume a Logarithmic Sample Mode:

Period 0 (first 30s): Number of events in sample group: `1000` , Sample Rate: `1:1` , Events allowed: `ALL`
Sample Rate calculation for **next** period: `Math.ceil(Math.log(1000)) = 7`

Period 1 (next 30s) -- Number of events in sample group: `4000` , Sample Rate: `7:1` : Events allowed: `572`
Sample Rate calculation for **next** period: `Math.ceil(Math.log(4000)) = 9`

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

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

...

Sample Modes:

1. Logarithmic - The sample rate is derived for each sample group using

`Math.ceil(Math.log(lastPeriodVolume))` (natural log). This mode is **less aggressive** and drops fewer events.

2. Square Root - The sample rate is derived for each sample group using

`Math.ceil(Math.sqrt(lastPeriodVolume))` . This mode is **more aggressive** and drops more events.

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 empty - 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. Left-hand side input is the key name, right-hand side is a JS expression to compute the value (can be constant). Nested addressing supported.

Keep Fields: List of fields to keep. Wildcards (*) and nested addressing supported. Takes precedence over Remove Fields (below).

Remove Fields: List of fields to remove. Wildcards (*) and nested addressing supported. supported. Cribl internal fields that start with `__` (double underscore) cannot be remove via wildcard. Instead they need to be specified individually. For example, `__myField` cannot be removed by specifying `__myF*`. A field matching an entry in Keep (wildcard or not) and Remove will not be removed. This is useful for implementing "remove all but" functionality. For example, to only keep `_time`, `_raw`, `source`, `sourcetype`, `host` we can specify them all in Keep while specifying `*` in Remove.

- *Note:* Negated terms are supported in both **Keep Fields** and **Remove Fields**. List is order sensitive when negated terms are used. E.g., `!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 a multivalued field `myTags`

- **Name:** `myTags`
- **Value Expression:** `login=='error' ? [...myTags, 'error'] : myTags`

See Ingest-time Fields for more examples.

Advanced Usage Notes

Note 1:

The Eval function has the ability to execute expressions without assigning their value to the field of an event. This can be done by simply leaving the left-hand side input empty and having the right hand side do the assignment.

- **Simple Example:** `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.
- **Another Example:** 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.

Note 2:

The Eval function can also be used to set and unset control fields (e.g., `_TCP_ROUTING` in Splunk) via this syntax: `_ctrl.<name>` . They can only be referenced 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, set the value to `undefined` . These fields are normally not needed for event computations and modifying them **is suggested to be done only by experts**. Please reach out to Cribl team 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 empty - 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`.

Fields: List of top level fields to include for flattening. Supports `*` wildcards. Defaults to empty array which means all fields.

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

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

Delimiter: Delimiter to be used for flattening. Defaults to `_` (underscore).

Example

Assume an input event like this:

input

```
{ "accounting" : [ { "firstName" : "John", "lastName" : "Doe", "age" : 23 }, { "firstName" : 'Mary', "lastName" : "Smith", "age" : 25 } ] }
```

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": 25
}
```



```
"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,  
}
```

GeoIP

Description

The `GeoIP` function enriches events with geo fields given an IP address. It is optimized for binary databases such as Maxmind's GeoIP

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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. If the database file is located within the lookup directory then the file doesn't have to be an absolute path.

IP Field: Field name where to find an IP to lookup, can be nested. Defaults to `ip`.

Result Field : Field name where to store the GeoIP lookup results. Defaults to `geoip`.

JSON Unroll

Description

The `JSON Unroll` function accepts a proper JSON event with an array of elements and converts them into individual events.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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 of each element in the new event. Leave empty to expand the array element.

Examples

Assume you have an incoming event as below:

sample.json

```
{  "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

```
{"date": "9/25/18 9:10:13.000 PM", "name": "Amrit", "age": 42, "car": {"name": "Blackberry", "models": [
```

Event 2

```
{"date": "9/25/18 9:10:13.000 PM", "name": "Amrit", "age": 42, "car": {"name": "Fiat", "models": ["500",
```

Event 3

```
{"date": "9/25/18 9:10:13.000 PM", "name": "Amrit", "age": 42, "car": {"name": "GM", "models": ["Trans A
```

Event 4

```
{"date": "9/25/18 9:10:13.000 PM", "name": "Amrit", "age": 42, "car": {"name": "Ford", "models": ["Fiest
```

Lookup

Description

The `Lookup` function enriches events with external fields. CSV lookup table files are supported.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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 location of the lookup file. Environment variables can be referenced via `$`, e.g. `$HOME/file.csv`.

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 Wildcard Match Mode, this attribute further 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, and `All` will return all matches in output as arrays. Defaults to `First Match`.

Reload Period (sec): Periodically check the underlying file for modtime changes and reload if necessary. Use `-1` to disable. Defaults to `60`.

Add to raw event: Whether to append the looked up values to `_raw` field as key=value pairs. Defaults to `No`.

Lookup Fields (.csv): Field(s) which 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 event field name. This input is optional.

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 lookup field name. This input is optional. Nested addressing supported.

Examples

See Ingest-time Lookups for examples.

Mask

Description

The `Mask` function masks, or replaces patterns in events.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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. Default to empty.

- **Match Regex:** Pattern to replace. Use `/g` to replace all matches e.g. `/(bar)/g`
- **Replace Expression:** A JS expression or literal to replace the matching content.

Apply To Fields: Fields where to apply the masking rules. Defaults to `_raw`. Wildcards (*) and nested addressing supported.

- *Note:* Negated terms are also supported. List is order sensitive when negated terms are used. E.g., `!foobar, foo*` means "All fields that start with 'foo' except foobar". `!foo*, *` means "All fields except for those that start with 'foo'".

Examples

See Masking and Obfuscation for examples.

Numerify

Description

The `Numerify` function converts fields of an event that are numbers to type of `number` .

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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: Fields to NOT numerify. By default numerify will apply to **all** fields. Wildcards (*) and nested addressing supported.

- *Note:* Negated terms are also supported. List is order sensitive when negated terms are used. E.g., `!foobar`, `foo*` means "All fields that start with 'foo' except foobar". `!foo*`, `*` means "All fields except for those that start with 'foo'".

Parser

Description

The `Parser` function can be used to extract fields out of events or reserialize (re-write) events with a subset of fields. Reserialization will **maintain** the format of the event. For example, if an event contains comma delimited fields and `fieldA` and `fieldB` are filtered out, their positions will be set to null and not deleted completely.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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`.

Parser Mode: Operating mode. **Extract** creates new fields. **Reserialize** will extract, filter fields and then reserialize. **Serialize** will put fields in a certain format. Defaults to `Extract`.

Source Field: Field which contains text to be parsed. Not usually needed in `Serialize` Mode.

Destination Field: Field name where to add extracted and serialized fields to. `Extract` and `Serialize` Mode only.

Type: Parser/Formatter type to use. Options: `CSV`, `JSON`, `K=V Pairs`, `Extended Log File Format (ELFF)`, `Common Log Format (CLF)`

Library: Browse Parser/Formatter library.

List of Fields: Fields expected to be extracted, in order. If not specified parser will auto-generate.

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.

- Note:* Negated terms are supported in both **Fields to Remove** and **Fields to Keep**. List is order sensitive when negated terms are used. E.g., `!foobar`, `foo*` means "All fields that start with 'foo' except foobar".
`!foo*`, `*` means "All fields except for those that start with 'foo'".

Fields Filter Expression: Expression evaluated against `{index, name, value}` context of each field. Return truthy to keep, falsy to remove field. Index is zero based.

Destination Field: Field where to add extracted fields to (`Extract` mode only).

How do Fields to Keep, Fields to Remove and Fields Filter Expression interact

Order or priority: **Fields to Keep** > **Fields to Remove** > **Fields Filter Expression**

If a field is in **Fields to Keep** and **Fields to Remove**, **Fields to Keep** takes precedence.

If a field is in **Fields to Remove** and in **Fields Filter Expression**, **Fields to Remove** takes precedence.

Example 1

Assume we have an event with KV pairs as below:

```
<timestamp> a=000,b=001,c=002,d=003,e=004,f=005,g1=006,g2=007,g3=008, ...
```

To extract all fields we can select K=V Pairs from Parser Type.

Scenario A: Keep fields `a`, `b`, `c`. Drop the rest.

Expected result: `a`, `b`, `c`

- Fields to Keep: `a`, `b`, `c`
- Fields to Remove: `*`
- Fields Filter Expression: `<empty>`

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`

Note: `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 an index 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 those that end in `Cxn` in the `values.total.` path

Parser Function Configuration:

The screenshot shows a configuration window for a parser function. At the top, it says "1 Parser" and "true". There are several sections with input fields and dropdown menus:

- 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 with "Reserialize" selected.
- Parser Type***: A dropdown menu with "JSON Object" selected.
- Parser Library**: A dropdown menu with "Select from Library" selected.
- 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.

JSON event after 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 3

Assume we have an event with KV pairs as below:

```
<timestamp> 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 selecting K=V Pairs from Parser Type. Then proceed with another Parser function right below it.

Scenario A: Serialize fields `a`, `b`, `c`, `d` in CSV format

Expected result: `_raw` field will have this value `000,001,002,003`

Parser 2

- Operation Mode: Serialize
- Source Field: <empty>
- Destination Field: <empty>
- Type: CSV
- List of Fields: `a`, `b`, `c`, `d` (needed for positional formats)

Scenario B: Serialize fields `a`, `b`, `c` in JSON format, under a field called `bar`

Expected result: `bar` field will be set to: `{"a":"000","b":"001","c":"002","d":"003"}`

Parser 2

- Operation Mode: Serialize
- Source Field: <empty>
- Destination Field: `bar`
- Type: JSON
- List of Fields: <empty>
- Fields to Keep: `a`, `b`, `c`, `d`

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 empty - 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`.

Metrics: List of metrics from event to extract and format. Formatted metrics can be used by a destination to pass metrics to a metrics aggregation platform.

- **Event Field Name:** The name of the field in event containing the metric value.
- **Metric Name Expression:** JavaScript expression to evaluate metric field name. Defaults to Event Field Name.
- **Metric Type:** Type of metric.

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. Defaults to `!_* *`.

- *Note:* **Dimensions** supports wildcards and negated terms. List is order sensitive when negated terms are used. E.g., `!foobar, foo*` means "Keep all dimensions that start with 'foo' except foobar". `!foo*, *` means "Keep all dimensions except for those that start with 'foo'".

Overwrite: If true overwrite previous metric specs, otherwise append. Defaults to `No`.

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
```

... and we want to use values of `bytes` and `packets` as metrics across these dimensions: `action` , `interface_id` and `dstaddr` .

Metrics:

Event Field Name	Metric Name Expression	Metric Type
<code>bytes</code>	<code>`metric_name.bytes`</code>	Gauge
<code>packets</code>	<code>`metric_name.packets`</code>	Gauge

Dimensions:

Dimensions
<code>action interface_id dstaddr</code>

OUTPUT

```
{
  "action": "REJECT",
  "interface_id": "eni-02f03c2880e4aaa3",
  "dstaddr": "10.0.1.11",
  "metric_name.bytes": 262256,
  "metric_name.packets": 6556,
}
```

Regex Extract

Description

The `Regex Extract` function extract fields with regex named groups. (In Splunk these will be index-time fields). Fields that start with `__` (double underscore) are special fields in Cribl. They are ephemeral and 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 empty - 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`.

Regex: Regex literal with named capturing groups, e.g. `(?<foo>bar)` or special `_NAME_N` and `_VALUE_N` capturing groups which extract **both name and value** of a field e.g., `(?<_NAME_0>[^\s]+)=(?<_VALUE_0>[^\s]+)`. Defaults to empty. See Examples below.

Source Field: Field where to perform regex field extraction. Nested addressing supported. Defaults to `_raw`.

Advanced Settings:

- **Max Exec:** The maximum number of times to apply the Regex to source field, used by `_NAME_N` and `_VALUE_N` capturing groups. Named capturing groups will always use a value of 1. Defaults to 100.
- **Field Name Format Expression:** Expression to format field names when *NAME* capturing groups are used. The **original** field name is in global `name`. E.g., to append `XX` to all field names: ``_${name}_XX``. If not specified names will be sanitized using regex: `/^[_0-9]+|^[a-zA-Z0-9_]+/g`.

Examples

Assume a simple event that looks like this: `metric1=23 metric2=42 dc=23 abc=xyz`

1. Extract only the `metric1` field:

Regex: `metric1=(?<metric1>\d+)`

Result: `metric1:"23"`

2. Extract all k=v pairs:

Regex: (?<_NAME_0>[^\s]+)=(?<_VALUE_0>[^\s]+)

Result: metric1:"23" , metric2:"42" , dc"23" , abc:"xyz"

Regex Filter

Description

The `Regex Filter` function will filter out events based on regex match.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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`.

Regex: Regex to text against. Defaults to empty.

Field: Name of the field to apply the regex on (defaults to `_raw`). Nested addressing supported.

Examples

See [Regex Filtering](#) for examples.

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 empty - 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`.

Sampling Rules: Events matching these rules will be sampled at the given rate

Filter: Filter expression matching events to be sampled. Use `true` to match all.

Sampling Rate: Integer, picks one out of N matching events.

Examples

See `Sampling` for examples.

Serialize

Description

The `Serialize` function can be used to serialize the content of an event into a pre-defined format.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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` .

Type: Data output format. Defaults to `CSV` .

Library: Browse Parser/Formatter library.

Fields To Serialize: Required for CSV, ELFF and CLF. All other formats support wildcard field lists.

Source Field: Field containing object to serialize. Leave blank to serialize top level event fields.

Destination Field: Field to serialize data to. Defaults to `_raw` .

Examples

Assume a simple event that looks like this: `{"time":"2019-08-`

`25T14:19:10.240Z","channel":"input","level":"info","message":"initializing input","type":"kafka"}`

1. Serialize these fields: `_time`, `channel`, `level`, `type` in CSV format into a new destination field called `test`

Type: `CSV`

Fields to Serialize: `_time channel level type`

Destination Field: `test`

Result: `_raw: 1566742750.24,input,info,kafka`

Suppress

Description

The `Suppress` function suppresses events over a period of time based on a key expression evaluation.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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` .

Key Expression: Suppression key expression used to uniquely identify events to suppress. For example, ``${ip}:${port}`` will use 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 (seconds): 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` .

Advanced Settings

Maximum Cache Size : The maximum number of keys that can be cached before idle entries are removed. Leave at default unless you understand the implications of changing. Defaults to `50000`

Suppression Period Timeout: The number of suppression periods 'Suppression Period' of inactivity before a cache entry is considered idle. Leave at default unless you understand the implications of changing. Defaults to `2` .

Num Events to Trigger Cache Clean-Up: Check cache for idle sessions every $N \times$ events when cache size is $>$ 'Maximum Cache Size'. Leave at default unless you understand the implications of changing. Defaults to `10000` .

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): `300`

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

! READ THIS!

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

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

Filter: `host!=undefined && port!=undefined`

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

Number to Allow: `1`

Suppression Period (sec): `300`

3. 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 those that qualify will be added a field `suppress=1` which can be used downstream to further transform them.

Tee

Description

The `Tee` tees events out to a command of choice, via `stdin`, one JSON formatted event per line.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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`.

Command: Command to execute and feed events to.

Args: Command arguments.

Environment variables: Environment variables to set or overwrite.

Communication Protocol:

Data is passed to the command through its **`stdin`** using this 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 (coming soon)

XML Unroll

Description

The `XML Unroll` function accepts a proper XML event with a set of elements and converts them into individual events.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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`.

Unroll Elements Regex: Path to 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: Add a field with this name, containing the index at which the item was located, starting from 0. In Splunk this will be an index-time field. Nested addressing supported. Defaults to: `unroll_idx`

Pretty Print: Whether to pretty print the output XML.

Examples

Assume you have an incoming event as below and we want to break all the `Child` elements and inherit `myID`, and `branchLocation`.

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>
</Parent>
```



```
<Child>
  <state>CA</state>
  <city>Oakland</city>
</Child>
<Child>
  <state>CA</state>
  <city>San Francisco</city>
</Child>
</Parent>
```

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 (beta)

Description

The `Prometheus Publisher` function allows for metrics to be published to a Prometheus compatible metrics endpoint.

- In current implementation endpoint is: `<cribl-host>:<api-port>/metrics`
- The function should **follow** `Publish Metrics` or `Aggregations` functions.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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`.

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: Determines whether or not the event should be consumed once published. Defaults to `No`.

Update Mode: Determines whether or not the publisher overwrites or update the published output. Defaults to `Yes`.

Reverse DNS (beta)

Description

The `Reverse DNS` function resolve hostnames using an IP address.

Usage

Filter: Filter expression (JS) that selects data to be fed through the function. Defaults to empty - 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 Fields

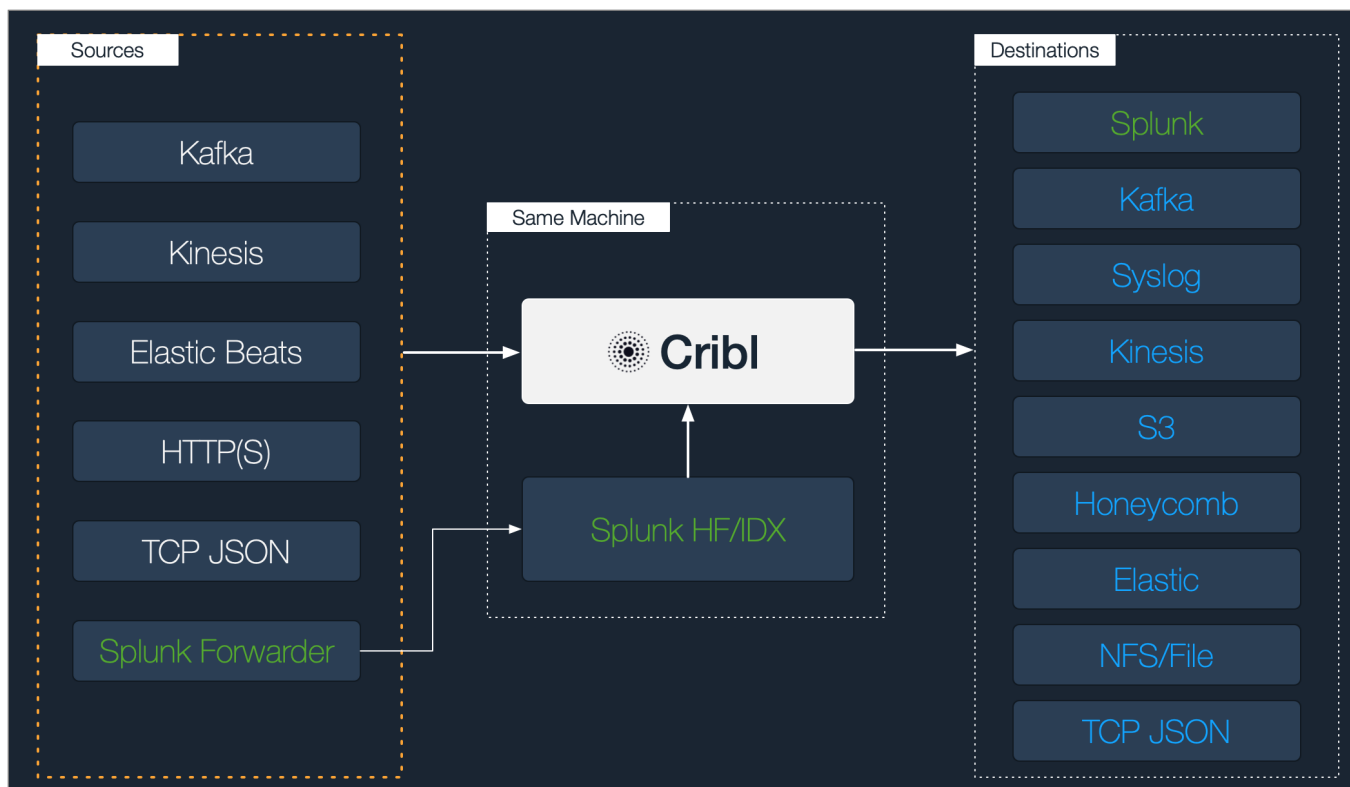
Lookup Field Name: The name of the field containing the IP address to lookup. If the field value is not in ipv4 or ipv6 format, the lookup is skipped.

Output Field Name: Name of field to add resolved the hostname as, leave blank to overwrite the lookup field.

Reload Period (minutes): How often (in minutes) to refresh DNS cache. Use 0 to disable. Defaults to `60` .

Sources

You can send data to Cribl from various sources, including Splunk, HTTP, Elastic Beats, Kinesis, Kafka and TCP JSON.



Sources

The following source types are supported.

- Splunk
- TCP JSON
- HTTP
- Kafka
- Kinesis Streams
- Azure Event Hubs
- Cribl Internal

Configuring and Managing Sources

For each source type users can create multiple definitions depending on your requirements.

To configure sources, click on **Sources**, select the desired type from the left vertical menu then click **Add New**.

Splunk

Cribl supports receiving of Splunk data data from Universal or Heavy Forwarders.

Configuring Cribl to receive Splunk data.

While on **Sources** screen, select **Splunk** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this Splunk source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Address**: Enter hostname/IP to listen for Splunk parsed data. E.g. `localhost` or `0.0.0.0` .
- **Port**: Enter port number.
- **IP Whitelist Regex**: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` i.e. all IPs.

TLS Settings (server side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Private Key Path**: Path on server where to find the private key to use in PEM format. Path can reference `$ENV_VARS`.
 - **Passphrase**: Passphrase to use to decrypt private key.
 - **Certificate Path** : Path on server where to find certificates to use in PEM format. Path can reference `$ENV_VARS`.
 - **CA Certificate Path** : Path on server where to find CA certificates to use in PEM format. 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` .
 - **Validate Client Certs**: Require server to reject any connection which is not authorized with the list of supplied CAs. Defaults to `No` .

Event Breaker Settings

- **Event Breaker Rulesets:** A list of event breaking rulesets that will be applied to the input data stream before being sent through the routes. Defaults to `System Default Rule` .
- **Event Breaker Buffer Timeout:** The amount of time in milliseconds the event breaker will wait for new data to be sent to a specific channel before flushing the data stream out as-is to the routes. Defaults to `10000` .

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data from this input before being sent through the routes.

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

`__inputId`

Syslog

Cribl supports receiving of data over syslog data.

Configuring Cribl to receive data over syslog.

While on **Sources** screen, select **Syslog** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this Syslog source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Address**: Enter hostname/IP to listen for Splunk parsed data. E.g. `localhost` or `0.0.0.0` .
- **UDP Port**: Enter UDP port number to listen on. Not required if listening on TCP.
- **TCP Port**: Enter TCP port number to listen on. Not required if listening on UDP.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data from this input before being sent through the routes.
- **Max Buffer Size (events)** : Maximum number of events to buffer when downstream is blocking.
- **IP Whitelist Regex**: Regex matching IP addresses that are allowed to send data. Defaults to `.*` i.e. all IPs.
- **Default Timezone**: Timezone to assign to timestamps without timezone info. Defaults to `local` .

TLS Settings (TCP ONLY)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Private Key Path**: Path on server where to find the private key to use in PEM format. Path can reference `$ENV_VARS`.
 - **Passphrase**: Passphrase to use to decrypt private key.
 - **Certificate Path** : Path on server where to find certificates to use in PEM format. Path can reference `$ENV_VARS`.

- **CA Certificate Path** : Path on server where to find CA certificates to use in PEM format. 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 .
- **Validate Client Certs**: Require server to reject any connection which is not authorized with the list of supplied CAs. Defaults to No .

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

__srcIpPort

TCP JSON

Cribl supports receiving of data over TCP in JSON format.

Configuring Cribl to receive TCP JSON data.

While on **Sources** screen, select **TCP JSON** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this TCP JSON source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Address**: Enter hostname/IP to listen for TCP JSON data. E.g. `localhost` or `0.0.0.0` .
- **Port**: Enter port number.
- **IP Whitelist Regex**: Regex matching IP addresses that are allowed to establish a connection. Defaults to `.*` i.e. all IPs.
- **Shared secret (authToken)**: Shared secret to be provided by any client (in `authToken` header field). If empty, unauthenticated access will be permitted.

TLS Settings (server side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Private Key Path**: Path on server where to find the private key to use in PEM format. Path can reference `$ENV_VARS`.
 - **Passphrase**: Passphrase to use to decrypt private key.
 - **Certificate Path** : Path on server where to find certificates to use in PEM format. Path can reference `$ENV_VARS`.
 - **CA Certificate Path** : Path on server where to find CA certificates to use in PEM format. 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` .
 - **Validate Client Certs**: Require server to reject any connection which is not authorized with the list of supplied CAs. Defaults to `No` .

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data from this input before being sent through the routes.

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

```
__inputId
```

Format

At the time of this writing, TCP JSON events are expected in new line 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` . Header line is **optional** when `authToken` is not set. 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", "fieldA":"valueA", "i  
{ "host":"myOtherHost", "source":"myOtherSource", "_raw": "{\"message\": \"Something informative
```

Note: 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"}
```

`_time`, `host` and `source` become their corresponding fields in Splunk. The value of `_raw` becomes the actual body of the event and `fieldA` becomes an index-time field. (`fieldA::valueA`)

Example

1. Configure Cribl 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 host: `cat test.json | nc <myCriblHost> 10001`

HTTP(S)

Cribl supports receiving of data over HTTP/S using the Elastic Bulk API or Cribl Bulk API.

Configuring Cribl to receive data over HTTP(S)

While on **Sources** screen, select **HTTP** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this HTTP(S) source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Address**: Enter hostname/IP to listen for HTTP(S) data. E.g. `localhost` or `0.0.0.0` .
- **Port**: Enter port number.
- **Shared secret (authToken)**: Shared secret to be provided by HTTP client in header (as `Authorization: <authToken>`). If empty, unauthenticated access **will be permitted**.
- **Elastic API Endpoint (Bulk API)**: Absolute path where to listen for the Elastic API requests. At the moment only `_bulk` is available. Others are faked as success. Use empty string to disable. Default to `/elastic` .
- **Cribl HTTP Event API**: Absolute path where to listen for Cribl HTTP API requests. Use empty string to disable. Defaults to `/cribl` .
- **Splunk HTTP Event Collector API**: Absolute path where to listen for the Splunk HTTP Event Collector API requests. Use empty string to disable. Defaults to `/services/collector` .
Note, this implementation is an **event**, and not **raw** endpoint. More here. To send data to it from a HEC client use `/services/collector/event` .
- **Splunk HEC Acks**: Whether to enable Splunk HEC acknowledgements. Defaults to `No` .

TLS Settings (server side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **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** : Path on server where to find certificates to use in PEM format. Path can reference \$ENV_VARS.
- **CA Certificate Path** : Path on server where to find CA certificates to use in PEM format. 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 .
- **Validate Client Certs**: Require server to reject any connection which is not authorized with the list of supplied CAs. Defaults to No .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data from this input before being sent through the routes.

Internal Fields

Cribl 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 be used to make processing decisions by functions.

Field(s) for this source:

__inputId
 __id (Elastic In)
 __type (Elastic In)
 __index (Elastic In)

Format & Endpoint

At the time of this writing, HTTP(S) events are expected to use the following format:

1. A JSON record per event.

Sample Event Format

```
{ "_time":1541280341, "_raw":"this is a sample event ", "host":"myHost", "source":"mySource", '
{"_time":1541280341, "host":"myOtherHost", "source":"myOtherSource", "_raw": "{\\"message\\":\\"$
```

Note 1: Events can be sent as separate POSTs but it is **highly** recommended that multiple of them are newline delimited, grouped and POSTed together.

Note 2: if a 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 as below:

```
{"_time":1541280341, "host":"myHost", "source":"mySource", "_raw":"this is a sample event ",  
"fieldA":"valueA"}
```

`_time` , `host` and `source` become their corresponding fields in Splunk. The value of `_raw` becomes the actual body of the event and `fieldA` becomes an index-time field. (`fieldA::valueA`)

Example

1. Configure Cribl to listen on port `10080` for HTTP (default). Set `authToken` to `myToken42` .
2. Send a payload to your Cribl host.

Cribl Single Event Example: Cribl Multiple Events Example: Splunk HEC Event Endpoint

Cribl Endpoint:

```
-----  
curl -k http://<myCriblHost>:10080/cribl/_bulk -H 'Authorization: myToken42' -d '{"_raw":"this
```


Kafka

Cribl supports receiving of data records from a Kafka cluster.

Configuring Cribl to receive data from Kafka topics.

While on **Sources** screen, select **Kafka** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Brokers**: List of Kafka brokers to use to, eg. `localhost:9092`.
- **Topics**: List of topics to subscribe to.
- **Group ID**: The name of the consumer group this Cribl instance belongs to.
- **From Beginning**: Whether to start reading from earliest available data, relevant only during initial subscription. Defaults to `Yes` .

TLS Settings (client side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Validate Server Certs**: Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No` .
 - **Server Name (SNI)**: Server Name Indication.
 - **CA Certificate Path** : Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth)**: Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
 - **Certificate Path (mutual auth)** : Path on client where to find certificates to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required**.
 - **Passphrase**: Passphrase to use to decrypt private key.

Schema Registry (For AVRO encoded data with schema stored in Confluent Schema Registry)

Kafka Schema Registry Authentication

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Schema Registry URL**: URL for access to the Confluent Schema Registry. e.g.,
`http://<hostname>:8081`

TLS Settings (client side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Validate Server Certs**: Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No` .
 - **Server Name (SNI)**: Server Name Indication.
 - **CA Certificate Path** : Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth)**: Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Certificate Path (mutual auth)** : Path on client where to find certificates to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Passphrase**: Passphrase to use to decrypt private key.

Authentication Settings

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **SASL Mechanism**: SASL authentication mechanism to use. Select one.
 - **Username**: Username.
 - **Password**: Password.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data from this input before being sent through the routes.

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

`__inputId`

`__topicIn` (indicates Kafka topic that event came from. See `__topicOut` in Kafka Destination)

`__schemaId` (when using Schema Registry)

Kinesis Streams

Cribl supports receiving of data records from Amazon Kinesis Streams.

Configuring Cribl to receive data from Kinesis Streams

While on **Sources** screen, select **Kinesis** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this Kinesis Stream source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Stream Name**: Kinesis stream name (not ARN) to read data from.
- **Shard Selection Expression**: A JS 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` .
- **Shard Iterator Start**: Location where 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 , CloudWatch Logs, Event Per Line, and New Line JSON). Defaults to `Cribl` .
- **API Key**: API key, if not present will fallback on `env.AWS_ACCESS_KEY_ID`, or the meta-data endpoint for IAM credentials.
- **Secret Key**: Secret key, if not present will fallback on `env.AWS_SECRET_ACCESS_KEY`, or the meta-data endpoint for IAM credentials.
- **Region**: Region where the Kinesis Stream is located.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data from this input before being sent through the routes.
- **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`

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

`__inputId`

Azure Event Hubs

Cribl supports receiving of data records from Azure Event Hubs.

Configuring Cribl to receive data from Azure Event Hubs.

While on **Sources** screen, select **Azure Event Hubs** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Brokers**: List of Event Hub Kafka brokers to connect to, e.g., `yourdomain.servicebus.windows.net:9093` . The hostname can be found in the host portion of the primary or secondary connection string in Shared Access Policies.
- **Event Hub Name**: The name of the Event Hub (a.k.a. Kafka Topic) to subscribe to.
- **Group ID**: Specifies the name of the consumer group this Cribl instance belongs to, should always be `$Default` for Event Hub.
- **From Beginning**: Whether to start reading from earliest available data, relevant only during initial subscription. Defaults to `Yes` .

Authentication Settings

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **SASL Mechanism**: SASL authentication mechanism to use, `PLAIN` is the only mechanism currently supported for Event Hub Kafka brokers.
 - **Username**: The username for authentication, for Event Hub this should always be `$ConnectionString` .
 - **Password**: Connection String Primary or Secondary key from Event Hub workspace.

TLS Settings (client side)

- **Disabled** Defaults to `No` .

- **Validate Server Certs:** For Event Hub, this should always be false. Defaults to No .

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data from this input before being sent through the routes.

Metrics

Cribl supports receiving of metrics in these wire formats/protocols: StatsD, StatsD Extended, Graphite. Automatic protocol detection will happen on the first line received over a TCP connection or a UDP packet. Lines not matching the detected protocol will be dropped.

Configuring Cribl to receive metrics.

While on **Sources** screen, select **Metrics** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this Syslog source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Address**: Enter hostname/IP to listen to. Defaults to `0.0.0.0` .
- **UDP Port**: Enter UDP port number to listen on. Not required if listening on TCP.
- **TCP Port**: Enter TCP port number to listen on. Not required if listening on UDP.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data from this input before being sent through the routes.
- **Max Buffer Size (events)** : Maximum number of events to buffer when downstream is blocking. Defaults to `1000` .
- **IP Whitelist Regex**: Regex matching IP addresses that are allowed to send data. Defaults to `.*` i.e. all IPs.

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

`__srcIpPort`

`__metricsInType`

Metric Event Schema

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
....

Sufficient information will be placed into a field called `__criblMetric` such that these events can be properly serialized out to any metric outputs (independent of the input type).

SQS

Cribl supports receiving of events from Amazon Simple Queuing Service.

Configuring Cribl to receive data from Amazon SQS

While on **Sources** screen, select **SQS** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this SQS source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Queue Name**: The name of the AWS SQS queue to read events from.
- **Queue Type**: The queue type used (or created). Defaults to `Standard` .
- **Create Queue**: Create queue if it does not exist.
- **API Key**: API key, if not present will fallback on `env.AWS_ACCESS_KEY_ID`, or the meta-data endpoint for IAM credentials.
- **Secret Key**: Secret key, if not present will fallback on `env.AWS_SECRET_ACCESS_KEY`, or the meta-data endpoint for IAM credentials.
- **Region**: Region where SQS queue is located.
- **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` .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data from this input before being sent through the routes.
- **Num Receivers**: The number of receiver processes to run, the higher the number the better throughput at the expense of CPU overhead. Defaults to `3` .

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

__sqsMetadata

S3

Cribl supports receiving of data from Amazon S3 using eventnotifications through SQS.

The source S3 bucket needs to be configured to send `s3:ObjectCreated:*` events to an SQS queue either directly (easiest) or via SNS. See below for configuration guidelines.

Configuring Cribl to receive data from Amazon S3

While on **Sources** screen, select **S3** from the vertical menu, then click **Add New**:

- **Input Id**: Enter a unique name to identify this SQS source definition.
- **Disabled** : Enable/disable toggle for this input. Defaults to `No` . I.e. Input is enabled.
- **Queue Name**: The name of the AWS SQS queue to monitor for S3 notifications.
- **Filename Filter**: Regex matching file names to download and process. Defaults to `.**`
- **API Key**: API key, if not present will fallback on `env.AWS_ACCESS_KEY_ID`, or the meta-data endpoint for IAM credentials.
- **Secret Key**: Secret key, if not present will fallback on `env.AWS_SECRET_ACCESS_KEY`, or the meta-data endpoint for IAM credentials.
- **Region**: Region where SQS queue is located.
- **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` .

Event Breaker Settings

- **Event Breaker Rulesets**: A list of event breaking rulesets that will be applied to the input data stream before being sent through the routes. Defaults to `System Default Rule` .
- **Event Breaker Buffer Timeout**: The amount of time in milliseconds the event breaker will wait for new data to be sent to a specific channel before flushing the data stream out as-is to the routes. Defaults to `10000` .

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data from this input before being sent through the routes.
 - **Visibility Timeout (Seconds):** Used to override the number of seconds message can be in-progress before deletion. Leave blank to use the default. This is important for processing of large files, the file must be downloaded and processed in this amount of time or the events will be duplicated.
 - **SQS Pollers:** Number of concurrent SQS event pollers, each poller can fetch 10 notifications at one time. Defaults to 2 .

How to configure S3 to send event notifications to SQS.

1. Create a SQS Queue. Note its ARN.
2. Replace its access policy with the one below by selecting the queue, and in the **Permissions** tab, click **Edit Policy Document (Advanced)**.
3. In Amazon S3 console, add a notification configuration publish events of the `s3:ObjectCreated:*` type to the SQS queue.

SQS Access Policy

```
{
  "Version": "2012-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SQS:SendMessage"
      ],
      "Resource": "SQS-queue-ARN",
      "Condition": {
        "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket-name" }
      }
    }
  ]
}
```

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this source:

__sqsMetadata

Cribl Internal

Cribl allows for capturing and sending its own internal logs through routes and pipelines.

Configuring Cribl Internal logs to behave as a data source.

While on **Sources** screen, select **Cribl Internal** from the vertical menu, then toggle **Enable** or **Disable**:

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data from this input before being sent through the routes.

Cribl Internal Log Fields

Fields below will be added to this source:

`source` , set to `cribl`

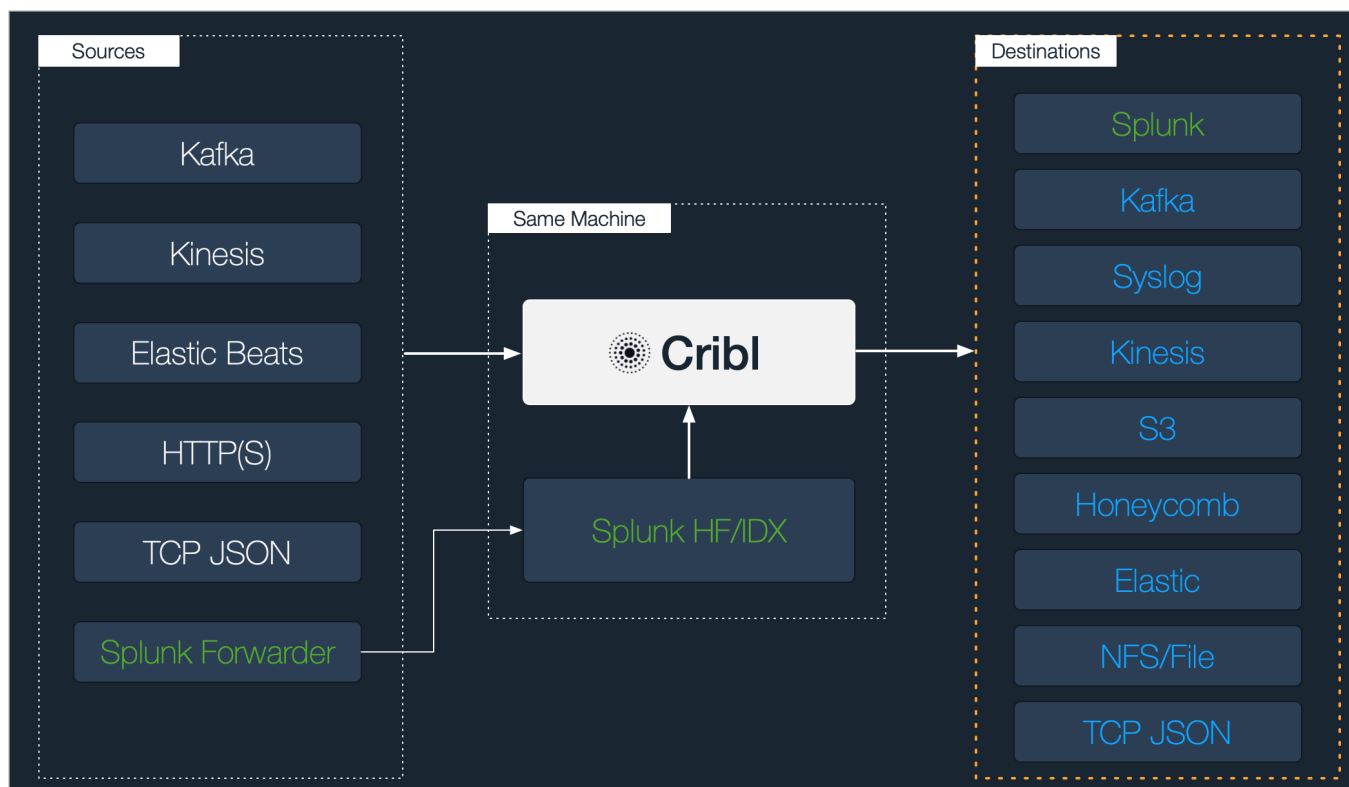
`host` , set to value of hostname of the Cribl instance

Note: use these fields to guide these events through Routes.

Destinations

You can send data processed through Cribl to other various destinations.

In Cribl, each pipeline can be independently configured with a destination definition. See the "Destinations" block on right below for a list of destination types.



How does it work

When non-streaming destination definitions are associated with a pipeline, Cribl will use a staging directory in the local filesystem to format and write outputted events. After a set of conditions (below) is met, typically file size and number of files, data is then compressed and then moved or copied to the final destination. An inventory of open, or in-progress files is kept in the root of staging directory in order to avoid having to walk that directory at startup. This can get expensive if staging is the final directory. At startup, Cribl will check for any left over files in progress from prior sessions and ensure they're moved/copied to final destination. The process of moving to final destination is delayed after startup (default 30 sec) and (b) processing of these files is paced at one per service period (default 1 second).

There are a number of **conditions** that govern when files are closed and rolled out:

1. File reaches its configured max size

2. File reaches its configured max open time
3. File reaches its configured max idle time

If a new file needs to be open, Cribl will enforce the number of max open files, by closing them in the order in which they were opened.

Delivery Policies

There is always at least one destination configured in Cribl. This is referred to as the **default** destination. In this version of Cribl, while each pipeline can be associated with any destination definition, in the event that that destination is unreachable, Cribl will send data to **default**. In the event that **default** is unavailable, the data will be dropped.

Destination Types

Streaming

Destinations that accept events in real-time and support back-pressure are referred to as streaming destinations.

Supported destinations:

- Splunk
- Splunk Load Balanced
- Splunk HEC
- AWS Kinesis Streams
- AWS CloudWatch Logs
- Elasticsearch
- Honeycomb
- Kafka
- Syslog
- TCP JSON
- Azure Blob Storage
- Azure Event Hubs
- Azure Monitor Logs
- StatsD
- StatsD Extended
- Graphite

Non-Streaming

Destinations that accept events in groups or batches are referred to as non-streaming destinations. Supported destinations:

S3 Compatible Stores

Filesystem/NFS

Configuring Destinations

For each destination type users can create multiple definitions depending on their requirements.

To configure destinations, click on **Destinations**, select the desired type from the left vertical menu then click **Add New**.

Splunk

Splunk Enterprise is a streaming destination type.

Configuring Cribl to output to **Splunk** destinations

While on **Destinations** screen, select **Splunk** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this Splunk destination definition.
- **Host**: Hostname of the Splunk receiver.
- **Port**: Port number.
- **Backpressure Behavior**: Whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to Block.
- **Nested Field Serialization**: Specifies how to serialize nested fields into index-time fields. Defaults to None.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.

TLS Settings (client side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Validate Server Certs**: Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No` .
 - **Server Name (SNI)**: Server Name Indication.
 - **CA Certificate Path** : Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth)**: Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Certificate Path (mutual auth)** : Path on client where to find certificates to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Passphrase**: Passphrase to use to decrypt private key.

Note: If you have a **single** .pem file with cacert, key and cert sections therein, enter it in all these inputs above: **CA Certificate Path, Private Key Path (mutual auth), Certificate Path (mutual auth)**.

Notes about forwarding to Splunk

- If events have a Cribl 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.

Splunk Load Balanced

Splunk is a streaming destination type and with **Splunk Load Balanced** output you can load balance data out to multiple Splunk receivers.

How does load balancing work

Cribl will attempt to load balance outbound data as fairly as possible across all receivers. Data is sent to all receivers simultaneously and the amount sent to each depends on these parameters:

1. Respective destination **weight**
2. Respective destination **historical data**

By default, historical data is tracked for 300s and it is used to influence the traffic sent to each destination so as to ensure that differences decay over time and 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 amount of data. Suppose further that the load balance stats period is set at 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. A will start the interval with **60** and B with **40**. To determine how many events A and B receive during this interval, Cribl will use their weights and their stats as follows:

Total number 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: $150 - 40 = 110$

End of interval 2 totals: $A=120+90=210$, $B=80+110=190$, a difference of **20 events** and a ratio of **1.1:1**.

Over the subsequent intervals, the difference becomes exponentially less pronounced and insignificant and thus the load gets balanced fairly.

Configuring Cribl to output to load balance to multiple **Splunk** destinations

While on **Destinations** screen, select **Splunk Load Balanced** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this Splunk LB destination definition.
- **DNS Resolution Period (s)**: Re-resolve any hostnames every this many seconds and pick up destinations from A records. Defaults to 60s.
- **Exclude Current Host IPs**: Exclude all IPs of the current host from the list of any resolved hostnames. Defaults to Yes.
- **Load Balance Stats Period (s)**: Lookback traffic history period. Defaults to 300s. *(Note that If multiple receivers are behind a hostname (i.e. multiple A records) all resolved IPs will inherit the weight the host, unless each IP is specified separately. In Cribl load balancing, IP settings take priority over those from hostnames.)*
- **Indexer Discovery**: Automatically discover indexers in indexer clustering environment. Defaults to No .
When toggled to Yes :
 - **Site**: Clustering site of the indexers from where indexers need to be discovered. In case of single site cluster, it defaults to `default` site.
 - **Cluster Master URI**: Full URI of Splunk Cluster Master (scheme://host:port).
 - **Auth Token**: Authentication token required to authenticate to cluster master for indexer discovery.
 - **Refresh Period**: Time interval in seconds between two consecutive indexer list fetches from cluster master. Defaults to `60` .

Note: To enable token authentication on Cluster Master, follow these steps here.

- **Backpressure Behavior**: Whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to Block.
- **Destinations**: Set of Splunk receivers where to load balance data to.

- **Host:** Hostname of the Splunk receiver.
- **Port:** Port number to send data to.
- **TLS:** Whether to inherit TLS configs from group setting or disable TLS. Defaults Inherit.
- **TLS Servername:** Servername to use if establishing a TLS connection. If not specified defaults to connection host (iff not an IP), otherwise the global TLS settings.
- **Weight:** The weight to use for load balancing purposes.

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data before sending it out using this output.

TLS Settings (client side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Validate Server Certs:** Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No` .
 - **Server Name (SNI):** Server Name Indication.
 - **CA Certificate Path:** Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth):** Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Certificate Path (mutual auth):** Path on client where to find certificates to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Passphrase:** Passphrase to use to decrypt private key.

Note: If you have a *single* .pem file with cacert, key and cert sections therein, enter it in all these inputs above: **CA Certificate Path, Private Key Path (mutual auth), Certificate Path (mutual auth).**

SSL Configuration for Splunk Cloud - Special Note

To connect to Splunk Cloud you **may** need to extract the private and public key from the Splunk provided Splunk Cloud Certificate (typically bundled in an app)

Step 1: Test connectivity to Splunk Cloud using the Root CA certificate

```
openssl s_client -CApath path_to_ca.pem -connect hostnameToSplunkCloud:9997
```

Step 2: Extract the Private key from Splunk Cloud Certificate. At the prompt you will need the sslPassword value in `outputs.conf` bundled with the Splunk Cloud app.

```
openssl ec -in path_to_server_cert.pem -out private.pem
```

Step 3: Extract the Public Key for Server Certificate

```
openssl x509 -in path_to_server_cert.pem -out server.pem
```

Step 4: In Cribl, in the destination TLS section enter the following:

- **CA Certificate Path:** Path to CA Certificate
- **Private Key Path (mutual auth):** Path to `private.pem` (above)
- **Certificate Path (mutual auth):** Path to `server.pem` (above)**

Notes about forwarding to Splunk

- If events have a Cribl 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.

Splunk HEC

Splunk HEC is a streaming destination type. In a typical deployment, Cribl will be installed/co-located in a Splunk heavy forwarder and if this output is enabled it can send data out to a Splunk HEC destination thru the **event endpoint**.

Configuring Cribl to output to **Splunk HEC** destinations

While on **Destinations** screen, select **Splunk HEC** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this Splunk HEC destination definition.
- **Splunk HEC Endpoint**: URL to an Splunk HEC endpoint where to send events, e.g.
`http://myhost.example.com:8088/services/collector/event`
- **HEC Auth Token**: Splunk HEC authentication token.
- **Next Processing Queue**: Specify the next Splunk processing queue to send the events after HEC processing. Defaults to **indexQueue**.
- **Default _TCP_ROUTING**: Specify the value of `_TCP_ROUTING` field for events that do not have `_ctrl._TCP_ROUTING` set. Defaults to `nowhere`. **Note**: this is useful only when this data is expected to be further routed to another destination by the HEC receiver.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Request Concurrency**: Maximum number of ongoing requests before blocking. Defaults to 5.
- **Max Body Size (KB)**: Maximum size, in KB, of the request body. Defaults to 4096.
- **Flush Period (s)**: Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to 1.
- **Extra HTTP Headers**: Name/Value pairs to pass as additional HTTP headers.

Notes on HTTP based outputs

- Cribl 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 next request.
- When resolving destination's hostname Cribl will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

S3 Compatible Stores

S3 is a non-streaming destination type. Cribl does **not** have to run on AWS in order to deliver data to S3. Stores that are S3-compatible will work with this destination type.

Configuring Cribl to output to **S3** destinations.

While on **Destinations** screen, select **S3** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this S3 destination definition.
- **S3 Bucket**: Enter an S3 Bucket where to upload the data.
- **Key Prefix**: Prefix to append to files before uploading.
- **Staging Location**: Local filesystem location where to buffer files before compressing and moving to final destination. It is advisable that this location stable and high performance.
- **API Key**: Enter your AWS API Key. If left blank, Cribl will fallback on `env.AWS_ACCESS_KEY_ID`, or the meta-data endpoint for IAM credentials.
- **Secret Key**: Enter your AWS Secret Key. If left blank, Cribl will fallback on `env.AWS_SECRET_ACCESS_KEY`, or the meta-data endpoint for IAM credentials.
- **Region**: Region where the S3 bucket is located.
- **Endpoint**: S3 service endpoint. If empty the endpoint will be automatically constructed from the region.
- **Signature Version**: Signature version to use for signing S3 requests.
- **Partitioning Expression**: JS expression to define how files are partitioned and organized. If left blank, Cribl will fallback on `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**: The output filename prefix. Defaults to `CriblOut`
- **Compress**: Data compression format used before moving to final destination. Default `none`. It is recommended that `gzip` is used.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.

- **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 .

Note

Cribl will close files when **either** of Max File Size (MB) or Max File Open Time (Sec) conditions are met.

- **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. Default: 30 .
- **Max Open Files:** Maximum number of files to keep open concurrently. When over, the oldest open files will be closed and moved to final output location. Default: 100 .

Internal Fields

Cribl uses a set of internal fields to assist in forwarding data to a destination.

Field(s) for this destination:

__partition

Kinesis Streams

Cribl can PUT events into **Amazon Kinesis Data Streams** records of up to 1MB uncompressed. Cribl does **not** have to run on AWS in order to deliver data to a Kinesis Data Stream.

Configuring Cribl to output to **Amazon Kinesis Data Streams**

While on **Destinations** screen, select **Kinesis** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this S3 destination definition.
- **Stream Name**: Kinesis Data Stream name where to send events.
- **API Key**: Enter your AWS API Key. If left blank, Cribl will fallback on `env.AWS_ACCESS_KEY_ID`, or the meta-data endpoint for IAM credentials.
- **Secret Key**: Enter your AWS Secret Key. If left blank, Cribl will fallback on `env.AWS_SECRET_ACCESS_KEY`, or the meta-data endpoint for IAM credentials.
- **Region**: AWS Region where the Kinesis Data Stream is located.
- **Signature Version**: Signature version to use for signing Kinesis stream requests.
- **Put Request Concurrency**: Maximum number of ongoing put requests before blocking.
- **Maximum Record Size**: Maximum size (KB) of each individual record before compression. For non compressible data 1MB is the max recommended size.
- **Flush Period (sec)**: Maximum time between requests. This could cause the payload size to be smaller than max.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.

Format

At the time of this writing, the outputted events use the following record format:

- Header line containing information about the payload (currently one type as follows)
- New Line Delimited JSON (that is, each Kinesis record will contain multiple events in **ndjson** format)

Record payloads (including header and body) will be gzip compressed and then Kinesis will base64 encode them.

Sample Kinesis Record

```
{"format":"ndjson","count":8,"size":3960}  
{"_raw":"07-03-2018 18:33:51.136 -0700 ERROR TcpOutputFd - Read error. Connection reset by peer"  
{"_raw":"07-03-2018 18:33:51.136 -0700 INFO TcpOutputProc - Connection to 127.0.0.1:10000 closed"  
...
```

SQS

Cribl supports sending events to Amazon Simple Queuing Service.

Configuring Cribl to send data to Amazon SQS

While on **Destinations** screen, select **SQS** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this SQS destination.
- **Queue Name**: The name of the AWS SQS queue to send events to.
- **Queue Type**: The queue type used (or created). Defaults to `Standard`.
- **Message Group Id**: This parameter applies only to FIFO queues. The tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are processed in a FIFO manner. Use event field `__messageGroupId` to override this value.
- **Create Queue**: Create queue if it does not exist.
- **API Key**: API key, if not present will fallback on `env.AWS_ACCESS_KEY_ID`, or the meta-data endpoint for IAM credentials.
- **Secret Key**: Secret key, if not present will fallback on `env.AWS_SECRET_ACCESS_KEY`, or the meta-data endpoint for IAM credentials.
- **Region**: Region where SQS queue is located.
- **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`.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data from this input before being sent through the routes.
- **Max Queue Size**: Maximum number of queued batches before blocking. Defaults to `5`.
- **Max Record Size (KB)**: Maximum size (KB) of each individual record, per the SQS spec the max allowed value is 256 KB. Defaults to `256`.

- **Flush Period (sec):** Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to 1 .
- **Max Concurrent Requests:** The maximum number of in-progress API requests before backpressure is applied. Defaults to 10 .

Internal Fields

Cribl 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 can be used to make processing decisions by functions.

Field(s) for this Destination:

`__messageGroupId`

CloudWatch Logs

Cribl supports sending of data over to Amazon CloudWatch Logs. This is a streaming destination type. Cribl does **not** have to run on AWS in order to deliver data to a CloudWatch Logs.

Configuring Cribl to output to **Amazon CloudWatch Logs**

While on **Destinations** screen, select **Kinesis** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this S3 destination 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 instance. E.g., `myStream_myHost_myOutputId` .
- **API Key**: Enter your AWS API Key. If left blank, Cribl will fallback on `env.AWS_ACCESS_KEY_ID`, or the meta-data endpoint for IAM credentials.
- **Secret Key**: Enter your AWS Secret Key. If left blank, Cribl will fallback on `env.AWS_SECRET_ACCESS_KEY`, or the meta-data endpoint for IAM credentials.
- **Region**: Region where the CloudWatchLogs is located.
- **Signature Version**: Signature version to use for signing CloudWatchLogs requests. Defaults to `v4` .
- **Max Queue Size**: Maximum number of queued batches before blocking. Defaults to `5` .
- **Maximum Record Size**: Maximum size (KB) of each individual record before compression. For non compressible data 1MB is the max recommended size. Defaults to `1024` .
- **Flush Period (sec)**: Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to `1` .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.

Filesystem/NFS

Filesystem is a non-streaming destination type that Cribl can use to output files to a local or a network attached filesystem (NFS).

Configuring Cribl to output to **Filesystem** destinations.

While on **Destinations** screen, select **Filesystem** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this Filesystem destination definition.
- **Output Location**: Final destination for the output files.
- **Staging Location**: Local filesystem location where to buffer files before compressing and moving to final destination. It is advisable that this location stable and high performance.
- **Partitioning Expression**: JS expression to define how files are partitioned and organized. If left blank, Cribl will fallback on `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**: The output filename prefix. Defaults to `CriblOut`
- **Compress**: Data compression format used before moving to final destination. Default `none`. It is recommended that `gzip` is used.
- **Max File Size (MB)**: Maximum uncompressed output file size. Files of this size will be closed and moved to final output location. Defaults to `32`.
- **Max File Open Time (Sec)**: Maximum amount of time to write to a file. Files open for longer than this will be closed and moved to final output location. Defaults to `300`.

Note

Cribl will close files when **either** of Max File Size (MB) or Max File Open Time (Sec) conditions are met.

- **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 over, the oldest open files will be closed and moved to final output location. Defaults to `100` .

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data before sending it out using this output.

Internal Fields

Cribl uses a set of internal fields to assist in forwarding data to a destination.

Field(s) for this destination:

`__partition`

Elasticsearch

Cribl can send events to an **Elasticsearch** cluster using the Bulk API.

Configuring Cribl to output to **Elasticsearch**

While on **Destinations** screen, select **Elasticsearch** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this Elasticsearch destination definition.
- **Bulk API URL**: Specify a URL to an Elasticsearch cluster where to send events, e.g.
`http://myElasticCluster.example.com:9200/_bulk`
- **Index**: Elasticsearch Index where to send events. Note that this value can be overwritten by 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

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Request Concurrency**: Maximum number of ongoing requests before blocking. Defaults to 5.
- **Max Body Size (KB)**: Maximum size, in KB, of the request body. Defaults to 4096.
- **Flush Period (s)**: Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to 1.
- **Extra HTTP Headers**: Name/Value pairs to pass as additional HTTP headers.

Internal Fields

Cribl uses a set of internal fields to assist in forwarding data to a destination.

Field(s) for this destination:

`__id`

`__type`

`__index`

Notes on HTTP based outputs

- Cribl 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 next request.
- When resolving destination's hostname Cribl will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

Honeycomb

Cribl supports sending of events to a **Honeycomb** dataset.

Configuring Cribl to output to **Honeycomb**

While on **Destinations** screen, select **Honeycomb** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this Honeycomb destination definition.
- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Dataset Name**: Name of the dataset where to send events. E.g. `iLoveObservabilityDataset`
- **Team**: Team id where the dataset belongs. E.g. `teamWilde`

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Request Concurrency**: Maximum number of ongoing requests before blocking. Defaults to 5.
- **Max Body Size (KB)**: Maximum size, in KB, of the request body. Defaults to 4096.
- **Flush Period (s)**: Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to 1.
- **Extra HTTP Headers**: Name/Value pairs to pass as additional HTTP headers.

Then, click **Save**.

Notes on HTTP based outputs

- Cribl 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 next request.
- When resolving destination's hostname Cribl will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

TCP JSON

Cribl supports sending of data over TCP in JSON format. **TCP JSON** is a streaming destination type.

Configuring Cribl to output in **TCP JSON** format

While on **Destinations** screen, select **TCP JSON** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination definition.
- **Host**: Hostname of the receiver.
- **Port**: Port number.
- **Auth Token**: Optional authentication token to include as part of the connection header. Defaults to empty.
- **Backpressure Behavior**: Whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block`.
 - **Compression**: Codec to use to compress the data before sending. Defaults to `None`.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.

TLS Settings (client side)

- **Disabled** defaults to `Yes`. When toggled to `No`:
 - **Validate Server Certs**: Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No`.
 - **Server Name (SNI)**: Server Name Indication.
 - **CA Certificate Path**: Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth)**: Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Certificate Path (mutual auth)**: Path on client where to find certificates to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**

Format

At the time of this writing, TCP JSON events are sent in new line delimited JSON format:

1. A header line. Can be empty. E.g. `{}` . If Auth Token is enabled it will be included here as a field called `authToken` . In addition, if events contain common fields they will be included here under `fields` .
2. A JSON event/record per line.

See an example [here](#).

Syslog

Cribl supports sending of data over syslog via TCP. **Syslog** is a streaming destination type.

Configuring Cribl to output in **Syslog** format

While on **Destinations** screen, select **Syslog** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination definition.
- **Host**: Hostname of the receiver.
- **Port**: Port number.
- **Facility**: Message facility, can be overwritten by `event.__facility` . Defaults to `user-level` .
- **Severity**: Message severity, can be overwritten by `event.__severity` . Defaults to `notice` .
- **App Name**: Application name to add to syslog messages, can be overwritten by `event.__appname` . Defaults to `Cribl` .
- **Message Format**: The syslog message format supported by the receiver. Defaults to `RFC3164` .
- **Backpressure Behavior**: Whether to block, drop, or queue events when all receivers in this group are exerting backpressure. Defaults to `Block` .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.

TLS Settings (client side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Validate Server Certs**: Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No` .
 - **Server Name (SNI)**: Server Name Indication.
 - **CA Certificate Path** : Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth)**: Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**

- **Certificate Path (mutual auth)** : Path on client where to find certificates to use in PEM format. Path can reference \$ENV_VARS. **Use only if mutual auth is required.**

Internal Fields

Cribl uses a set of internal fields to assist in forwarding data to a destination.

Field(s) for this destination:

__priority
__facility
__severity
__procid
__appname
__msgid
__syslogout

Kafka

Cribl supports sending of data over to a Kafka topic. **Kafka** is a streaming destination type.

Configuring Cribl to output to **Kafka**

While on **Destinations** screen, select **Kafka** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination definition.
- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Brokers**: List of Kafka brokers to connect to, eg. localhost:9092.
- **Topic**: The topic where to publish events. Can be overwritten using field `__topic` in event.
- **Acknowledgments**: Control the number of required acknowledgments.
- **Record Data Format**: Format to use to serialize events before writing to Kafka. Defaults to `JSON`.
- **Compression**: Codec to use to compress the data before sending to Kafka. Defaults to `gzip`.

TLS Settings (client side)

- **Disabled** defaults to `Yes`. When toggled to `No`:
 - **Validate Server Certs**: Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No`.
 - **Server Name (SNI)**: Server Name Indication.
 - **CA Certificate Path**: Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth)**: Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Certificate Path (mutual auth)**: Path on client where to find certificates to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Passphrase**: Passphrase to use to decrypt private key.

Schema Registry (For AVRO encoded data with schema stored in Confluent Schema Registry)

Kafka Schema Registry Authentication

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **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 Settings (client side)

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **Validate Server Certs**: Require client to reject connections to servers whose certs are not signed by one of the supplied CAs. Defaults to `No` .
 - **Server Name (SNI)**: Server Name Indication.
 - **CA Certificate Path** : Path on client where to find CA certificates to use to verify the server's cert in PEM format. Path can reference `$ENV_VARS`.
 - **Private Key Path (mutual auth)**: Path on client where to find the private key to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Certificate Path (mutual auth)** : Path on client where to find certificates to use in PEM format. Path can reference `$ENV_VARS`. **Use only if mutual auth is required.**
 - **Passphrase**: Passphrase to use to decrypt private key.

Authentication Settings

Authentication parameters to use when connecting to brokers. Using TLS is highly recommended.

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **SASL Mechanism**: SASL authentication mechanism to use. Select one.
 - **Username**: Username.
 - **Password**: Password.

Advanced Settings

- **Conditioning Pipeline:** Pipeline to process data before sending it out using this output.
- **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 (s):** Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to 1 .

Internal Fields

Cribl uses a set of internal fields to assist in forwarding data to a destination.

Field(s) for this destination:

__topicOut

__key

__headers

__keySchemaIdOut

__valueSchemaIdOut

Azure Blob Storage

Azure Blob Storage is a non-streaming destination type. Cribl does **not** have to run on Azure in order to deliver data to it.

Configuring Cribl to output to **Azure Blob Storage**.

While on **Destinations** screen, select **Azure > Azure Blob Storage** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this S3 destination definition.
- **Account Name**: Enter your Azure Storage Account Name. If left blank, Cribl will fallback on `env.AZURE_STORAGE_ACCOUNT`.
- **Account Key**: Enter your Azure Storage Key. If left blank, Cribl will fallback on `env.AZURE_STORAGE_KEY`.
- **Container Name**: A container organizes a set of blobs, similar to a directory in a file system.
- **Create Container**: Toggle to create the configured container in Azure Blob Storage if it does not already exist.
- **Blob Prefix**: Prefix to append to files before uploading.
- **Staging Location**: Local filesystem location where to buffer files before compressing and moving to final destination. It is advisable that this location stable and high performance.
- **Partitioning Expression**: JS expression to define how files are partitioned and organized. If left blank, Cribl will fallback on `event.__partition`. Defaults to ``${host}/${sourcetype}``
- **Data Format**: Format of the output data. Defaults to `json`.
- **File Name Prefix**: The output filename prefix. Defaults to `CriblOut`
- **Compress**: Data compression format used before moving to final destination. Default `none`. It is recommended that `gzip` is used.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **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 .

Note

Cribl will close files when **either** of Max File Size (MB) or Max File Open Time (Sec) conditions are met.

- **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. Default: 30 .
- **Max Open Files:** Maximum number of files to keep open concurrently. When over, the oldest open files will be closed and moved to final output location. Default: 100 .

Internal Fields

Cribl uses a set of internal fields to assist in forwarding data to a destination.

Field(s) for this destination:

__partition

Azure Event Hubs

Cribl supports sending of data over to Azure Event Hubs. This is a streaming destination type.

Configuring Cribl to output to Azure Event Hubs

While on **Destinations** screen, select **Azure | Event Hubs** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination definition.
- **Brokers**: List of Event Hub Kafka brokers to connect to, e.g., `yourdomain.servicebus.windows.net:9093` . The hostname can be found in 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 publish events. Can be overwritten using field `__topicOut` .
- **Acknowledgments**: Control the number of required acknowledgments. Defaults to `Leader` .
- **Record Data Format**: Format to use to serialize events before writing to the Event Hub Kafka brokers. Defaults to `JSON`
- **Compression**: Codec to use to compress the data before sending to Event Hub Kafka brokers. Defaults to `gzip` .

Authentication Settings

Authentication parameters to use when connecting to brokers. Using TLS is highly recommended.

- **Disabled** defaults to `Yes` . When toggled to `No` :
 - **SASL Mechanism**: SASL authentication mechanism to use, `PLAIN` is the only mechanism currently supported for Event Hub Kafka brokers.
 - **Username**: The username for authentication, for Event Hub this should always be `$ConnectionString` .
 - **Password**: Connection String Primary or Secondary key from Event Hub workspace.

TLS Settings (client side)

- **Disabled** Defaults to No .
- **Validate Server Certs**: For Event Hub, this should always be false. Defaults to No .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **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 (s)**: Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to 1 .

Internal Fields

Cribl uses a set of internal fields to assist in forwarding data to a destination.

Field(s) for this destination:

__topicOut

__key

__headers

__keySchemaIdOut

__valueSchemaIdOut

Azure Monitor Logs

Cribl supports sending of data over to Azure Monitor Logs. This is a streaming destination type.

Configuring Cribl to output to Azure Monitor Logs

While on **Destinations** screen, select **Azure | Monitor Logs** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination definition.
- **Workspace Id**: Azure Log Analytics Workspace ID, see Azure Dashboard Workspace->Advanced settings.
- **Workspace Key**: Azure Log Analytics Workspace Primary or Secondary Shared Key, see Azure Dashboard Workspace->Advanced settings.
- **Log Type**: The Record Type of events sent to this LogAnalytics workspace. Defaults to `Cribl`.
- **Resource ID**: Optional Resource ID of the Azure resource the data should be associated with. This populates the `_ResourceId` property and allows the data to be included in resource-centric queries. If this field isn't specified, the data will not be included in resource-centric queries.

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Request Concurrency**: Maximum number of ongoing requests before blocking. Defaults to `5`.
- **Max Body Size**: Maximum size, in KB, of the request body. Defaults to `4096`.
- **Flush Period (s)**: Maximum time between requests. This could cause the payload size to be smaller than max. Defaults to `1`.
- **Extra HTTP Headers**: Name/Value pairs to pass as additional HTTP headers.

Notes on HTTP based outputs

- Cribl 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 next request.
- When resolving destination's hostname Cribl will pick the first IP in the list for use in the next connection. Round-robin DNS would help with event balancing.

StatsD

Cribl supports sending of data over to a StatsD destination. This is a streaming destination type.

Configuring Cribl to output via StatsD.

While on **Destinations** screen, select **Metrics | StatsD** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination 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` .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Max record Size (Bytes)**: Used when Protocol is UDP, specifies the maximum size (Bytes) of packets sent to the destination. Also known as the MTU for the network path to the destination system. Defaults to `512` .
- **Flush period (sec)**: Used when Protocol is TCP to specify how often buffers should be flushed resulting in records sent to the destination. Defaults to `1` .

StatsD Extended

Cribl supports sending of data over to a StatsD destination. This is a streaming destination type.

Configuring Cribl to output via StatsD Extended.

While on **Destinations** screen, select **Metrics | StatsD Extended** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination 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` .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Max record Size (Bytes)**: Used when Protocol is UDP, specifies the maximum size (Bytes) of packets sent to the destination. Also known as the MTU for the network path to the destination system. Defaults to `512` .
- **Flush period (sec)**: Used when Protocol is TCP to specify how often buffers should be flushed resulting in records sent to the destination. Defaults to `1` .

Graphite

Cribl supports sending of data over to a Graphite backend destination. This is a streaming destination type.

Configuring Cribl to output to a Graphite backend.

While on **Destinations** screen, select **Metrics | Graphite** from the vertical menu, then click **Add New**:

- **Output Id**: Enter a unique name to identify this destination 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` .

Advanced Settings

- **Conditioning Pipeline**: Pipeline to process data before sending it out using this output.
- **Max record Size (Bytes)**: Used when Protocol is UDP, specifies the maximum size (Bytes) of packets sent to the destination. Also known as the MTU for the network path to the destination system. Defaults to `512` .
- **Flush period (sec)**: Used when Protocol is TCP to specify how often buffers should be flushed resulting in records sent to the destination. Defaults to `1` .

Output Router

Output Routers are meta-destinations that allow for output selection based on rules. Rules are evaluated in order, top->down, with first match being the winner.

Configuring Cribl to output to an Output Router

While on **Destinations** screen, select **Output Router** from the vertical menu, then click **Add New**:

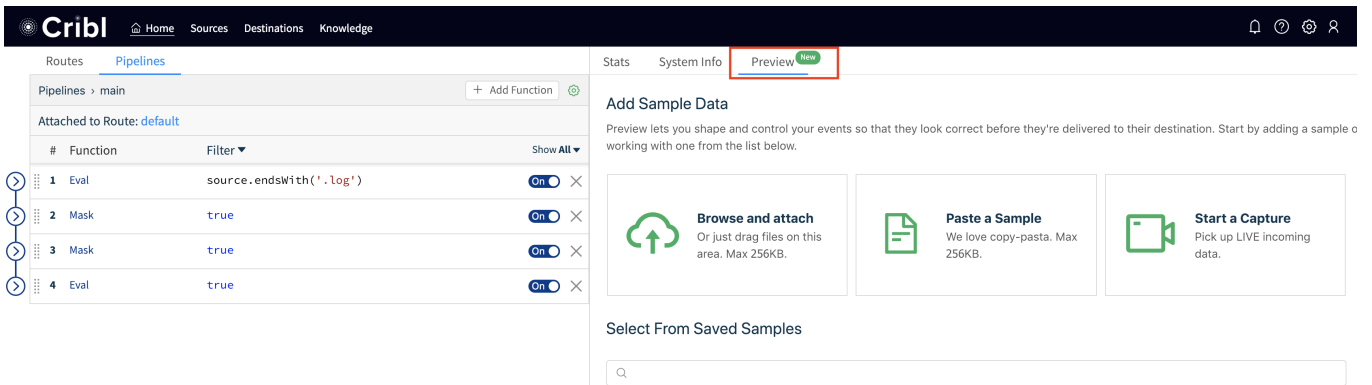
- **Router Name:** Enter a unique name to identify this router definition.
- **Rules:** A list of event routing rules.
 - **Filter Expression:** JavaScript expression to select events to send to output.
 - **Output:** Output where to send matching events.
 - **Final:** Flag to control whether to stop the event from being checked against other rules. Defaults to `Yes`.

Notes

- An Output Router cannot reference another. This is by design so as to avoid cycles.
- **Events that do not match any of the rules are dropped.** Use a catchall rule to change this behavior.
- No conditioning can be done here. Use Conditioning Pipelines at Source tier..
- Data can be cloned by turning the `Final` flag to `No` (set to `Yes` default, i.e. no cloning).

Data Preview

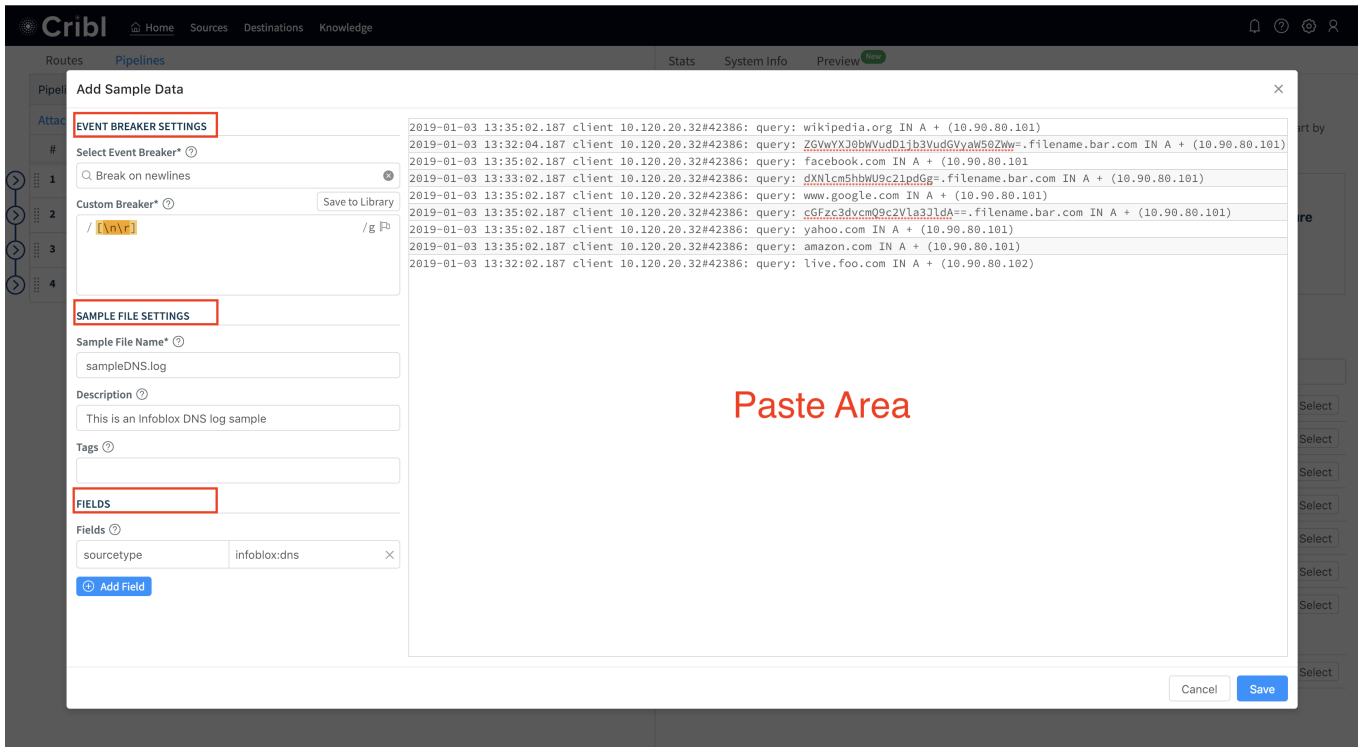
Data Preview is a feature that allows for visual inspection of events as they make their trip into a pipeline. It helps users shape and control events before they're delivered to a destination as well as assists with troubleshooting functions. It works by taking a set of Sample events, passing them thru the pipeline and displaying the result on a different pane. Anytime a function is modified, added or removed, the pipeline changes and so does its output.



While in a pipeline, samples can be added through one of the supported options: Upload, Paste or Capture. The Upload and Paste options work with content that needs to be broken into events, while the Capture 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 screen similar to below. *The Capture screen is slightly different in that there is no need for event breaking.*



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 how to break the file or paste content into events. Breaking will occur at the **start** of the match. Cribl ships with several common breaker patterns out of box but custom breakers can be configured. The UI here is interactive and you can iterate until you find the exact pattern.

Fields

The Fields section allows users to add or overwrite key-value pairs on the sample.

In Tab: Displaying samples on the way IN to the pipeline

There are three display options for the event: Text, JSON, and Table and each can be useful depending on the type of data being previewed. As you add more samples to your system you can easily access them via the dropdown on the top right: **All Samples > Sample Name**

Routes Pipelines

Pipelines > exfil + Add Function

Attach Pipeline to Route

#	Function	Filter	Show All
1	Regex Extract	sourcetype='infoblox:dns'	On
2	Lookup	sourcetype='infoblox:dns'	On
3	Drop	sourcetype='infoblox:dns' && rank != undefined	On
4	Eval	sourcetype='infoblox:dns'	On

Stats System Info Preview

In Out

All Samples > sampleDNS.log

TEXT JSON TABLE Select Fields (2 of 2)

#	Event
1	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: wikipedia.org IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
2	{ "_raw": "2019-01-03 13:32:04.187 client 10.120.20.32#42386: query: ZGvWYX30bWudD1jb3VudGVyaW50ZWw=.filename.bar.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
3	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: facebook.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
4	{ "_raw": "2019-01-03 13:33:02.187 client 10.120.20.32#42386: query: dXNlcm5hbWU9c2lpdGg=.filename.bar.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
5	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: www.google.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
6	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: cGFzc3dvcmQ9c2Vla3JldA==.filename.bar.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
7	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: yahoo.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
8	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: amazon.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" }
9	{ }

Out Tab: Displaying samples on the way OUT of the pipeline

As data traverses functions in the pipeline, events may be modified and some may be dropped altogether. When they're dropped, they are displayed as grayed-out with strikethrough text in the OUT tab. You can control their display with the Show Dropped toggle. When new fields are added, as shown above, they're highlighted green. Fields to be displayed are controlled with the **Selected Fields** dropdown.

Routes Pipelines

Pipelines > exfil + Add Function

Attach Pipeline to Route

#	Function	Filter	Show All
1	Regex Extract	sourcetype='infoblox:dns'	On
2	Lookup	sourcetype='infoblox:dns'	On
3	Drop	sourcetype='infoblox:dns' && rank != undefined	On
4	Eval	sourcetype='infoblox:dns'	On

Filter

sourcetype='infoblox:dns'

Description

Compute Shannon entropy, subdomain length and potential_exfil field

Final Yes No

Evaluate Fields

Name	Value Expression
decoded_payload	subdomain && C.Decode.base64(subdomain, 'utf8-vali...
entropy	subdomain && Math.round(C.Text.entropy(subdomain)*...
potential_exfil	decoded_payload && 'yes'
query_label_len	decoded_payload && subdomain.length

+ Add Field

Keep Fields

Field names

Remove Fields

Field names

Stats System Info Preview

In Out

All Samples > sampleDNS.log

TEXT JSON TABLE Show Dropped Select Fields (10 of 10)

#	Event
1	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: wikipedia.org IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" "domain": "wikipedia.org" "rank": 101 "cribl_pipeline": "exfil" }
2	{ "_raw": "2019-01-03 13:32:04.187 client 10.120.20.32#42386: query: ZGvWYX30bWudD1jb3VudGVyaW50ZWw=.filename.bar.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" "subdomain": "ZGvWYX30bWudD1jb3VudGVyaW50ZWw=", "domain": "bar.com" "decoded_payload": "department=counterintel" "entropy": 4.164 "potential_exfil": "yes" "query_label_len": 32 "cribl_pipeline": "exfil" }
3	{ "_raw": "2019-01-03 13:35:02.187 client 10.120.20.32#42386: query: facebook.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" "domain": "facebook.com" "rank": 101 "cribl_pipeline": "exfil" }
4	{ "_raw": "2019-01-03 13:33:02.187 client 10.120.20.32#42386: query: dXNlcm5hbWU9c2lpdGg=.filename.bar.com IN A + (10.90.80.101)" "sourcetype": "infoblox:dns" "subdomain": "dXNlcm5hbWU9c2lpdGg=", "domain": "bar.com" "decoded_payload": "username=smith" "entropy": 4.122 "potential_exfil": "yes" "query_label_len": 20 "cribl_pipeline": "exfil" }

Securing Data

Cribl can be used to encrypt sensitive data in real-time and route it to an end system. Decrypted retrieval can be implemented on a per-system basis. At the time of this writing decryption is supported only when Splunk is the end system.

- Data Encryption
- Data Decryption

Encryption

Encryption of data in motion

With Cribl you can encrypt fields or patterns within events in real-time 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 that describes the content to be replaced. The **Replace Expression** is a JS expression or literal to replace matched content. `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 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 key class

@param - keyId - encryption keyId, takes precedence over keyclass

@param - defaultVal - what to return if encryptions 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 key encryption 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 are collection 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 id `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>	<code>datasetA</code> Users: <code>U0</code> , <code>U1</code> , <code>U2</code>

User `U1` has been given access to an **additional** keyclass, `1` which contains key id `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 to 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> Users: <code>U1</code>	<code>datasetA</code> Users: <code>U0</code> , <code>U1</code> , <code>U2</code>

Configuring Keys with CLI

When using the `local` key management system, encryption keys in Cribl are encrypted with `$(CRIBL_HOME)/local/cribl/auth/cribl.secret` and stored in `$(CRIBL_HOME)/local/cribl/auth/keys.json`. Cribl monitors `keys.json` file for changes every 60 seconds.

Note: when installed as a Splunk app, `$(CRIBL_HOME)` is `$(SPLUNK_HOME)/etc/apps/cribl`.

Keys are added and listed using the `keys` command.

Listing keys

```
$(CRIBL_HOME)/bin/cribl keys list
```

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/cribld 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

Adding a key to keyclass 1 with no expiration date.

```
$CRIBL_HOME/bin/cribld keys add -c 1 -i
```

Sample Command Output

Adding key: success. Key count=1

Listing keys to verify key generation

```
$CRIBL_HOME/bin/cribld keys list
```

Sample Command Output

keyId	algorithm	keyclass	kms	created	expires	useIV
1	aes-256-cbc	1	local	1545243364.342	0	true

Configuring Keys with UI

The key management interface can be accessed through **Settings | Encryption Keys** . Here you can list and add new keys. To protect against accidental changes, once saved a key's parameters can only be edited through

configuration files.

Encryption Keys

Get Key Bundle

Search

+ 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

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. `cribl.secret` and `keys.json` in `$CRIBL_HOME/local/cribl/auth` **in the Cribl instance where encryption happened** should be synced/copied over to the one on the Search Head/decrypting side. When using the UI, these files can be downloaded through the **Get Key Bundle** button.

Decryption

Decryption of data

At the time of this writing, decryption is supported only when Splunk is the end system. Decryption in Splunk can be done by users of any role with permissions to the `decrypt` command. Further restrictions can be applied when capabilities are used. See below for more.

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

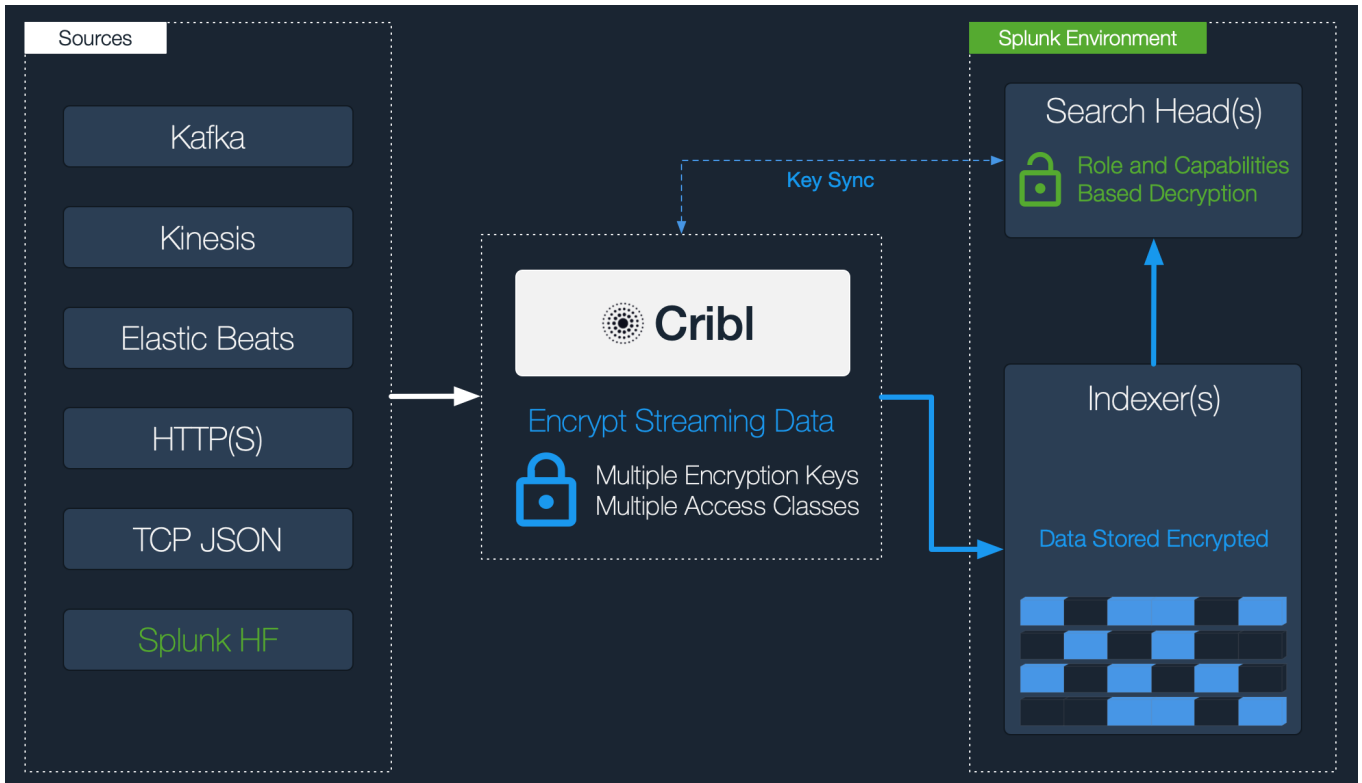
To ensure that keys don't get distributed to all search peers, including ones that your search head can search but you don't have full control over, `decrypt` is scoped to run locally on the installed search head.

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>	<code>1</code>
<code>cribl_keyclass_2</code>	<code>2</code>
...	...
<code>cribl_keyclass_N</code>	<code>N</code>

Configuring Splunk Search Head to Decrypt Data



- Install Cribl App for Splunk on your Search Head. As of v1.7, the app will run on search head mode by default. If previously installed and later modified, you can convert to search head mode with:
`$CRIBL_HOME/bin/cribl mode-searchhead` . When installed as a Splunk app `$CRIBL_HOME` is `$SPLUNK_HOME/etc/apps/cribl` .
- Assign permissions to the `decrypt` command per your requirements.
- Assign capabilities to your Roles per your requirements. If you'd like to create more capabilities ensure that they follow the naming convention defined above.
- Sync `auth/(cribl.secret|keys.json)` . To successfully decrypt data, the `decrypt` command will need access to the same keys that were used to encrypt. `cribl.secret` and `keys.json` in `$CRIBL_HOME/local/cribl/auth` **in the Cribl instance where encryption happened** should be synced/copied over to the one on the Search Head/decrypting side. When using the UI, these files can be downloaded through the **Get Key Bundle** button.

Scripts

Admins can run scripts (e.g., shell scripts) from within Cribl by configuring and executing them thru **Settings > Scripts**. They are typically used to call custom automation jobs or in general trigger tasks on demand. For example, you can use Scripts to run an Ansible job, or place a call to another automation system, when Cribl configs are updated.

Manage Scripts + Add New

● 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*

Command*

Description

Arguments

Env Var Name	Env Var Value
<input type="text"/>	<input type="text"/>

+ Add Variable

Run Delete Script

Command: Command to execute for this script.

Arguments: Arguments to pass when executing this script

Env Variables: Extra environment variables to set when executing script

! With great power comes great responsibility!

Scripts will allow you to execute almost anything on the system where Cribl is running. Make sure you understand the impact of what you're executing before you do so!

EXPRESSION REFERENCE

Introduction

As data travels a Cribl pipeline, it is operated on by a series of functions. Functions are fundamentally Javascript code.

Functions that ship with Cribl 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
<code>x = 42</code>	<code>(Math.random() * 42)</code>
<code>newFoo = foo.slice(30)</code>	<code>3 + 4</code>
	<code>'foobar'</code>
	<code>'42'</code>

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. They 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:vpctest'`

Truthy	Falsy
<code>true</code>	<code>false</code>
<code>42</code>	<code>null</code>
<code>-42</code>	<code>undefined</code>

3.14	0
"foo"	NaN
Infinity	' '
-Infinity	""

Value Expressions

Values 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')`

Considerations and best practices for creating predictable expressions

- In a value expression ensure that the source variable is not **null**, **undefined** or **empty**. For example, if you want to have a field called `len` to be assigned the length of a field called `employeeID` but you're not sure if `employeeID` exists, instead of `employeeID.length` you can use a safer shorthand as such: `(employeeID || '').length`.
- If a field does not exist (undefined) and you're doing a comparison with its properties the boolean expression will **always** evaluate to false. For example, if `employeeID` is undefined, then both of these expressions `employeeID.length > 10`, and `employeeID.length < 10` will evaluate to false.
- `==` means equal to, while `===` means equal value and equal type.. For example, `5 == 5` evaluates to **true**, while `5 === "5"` evaluates to **false**.
- 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 can do: `(age >= 18) ? 'adult' : 'minor'`

Wildcard Lists

Wildcards 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 term. They also accept strings that start with 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 whitelists and blacklists.

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 .

Cribl Expressions

Native Cribl 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:

Name	Value Expression
myNewField	C.Mask.sha1(myOtherField)

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 where 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 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 encryptions fails for any reason, if unspecified the original value is returned

@returns - - if encryption succeeds the encrypted value, otherwise defaultVal if specifier, 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 and returns a string or Buffer depending on 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 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 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 (eg. 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 it's hex representation (lower case). 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.Mask - Data Masking Functions

C.Mask.CC

(method) Mask.CC(value: string, unmasked?: number, maskChar?: string): string

Check that 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 unmasked digits, 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 that 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 unmasked digits, 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 that 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 that 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 unmasked digits, 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, imei etc) By default the mod 10 of the checksum is returned, pass mod = 0 to get actual checksum

@param - value a string whose digits you want to perform the Luhn checksum on

@param - mod return checksum module 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 given value

@param - value compute 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 or the result, or if a string use it's 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 or the result, or if a string use it's length

@param - char pattern which 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 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.Net - Network Functions

`C.Net.cidrMatch()`

(method) `Net.cidrMatch(cidrIpRange: string, ipAddress: string): boolean`

Determines if the supplied IPv4 `ipAddress` is inside the range of addresses identified by `cidrIpRange`. For

example: `C.Net.cidrMatch('10.0.0.0/24', '10.0.0.100')` returns `true`

@param - `cidrIpRange` - IPv4 address range in cidr format. E.g., 10.0.0.0/24

@param - `ipAddress` - The IPv4 IP address to test for inclusion in `cidrIpRange`

`C.Net.ipv6Normalize()`

(method) `Net.ipv6Normalize(address: string): string`

Normalize an IPV6 address based on RFC draft-ietf-6man-text-addr-representation-04

@param - `address` - the IPV6 address to normalize

`C.Net.isPrivate()`

(method) `Net.isPrivate(address: string): string`

Determine if the supplied IPv4 address is in the range of private addresses per RFC1819.

@param - `address` - address to test

C.os - System Functions

`C.confVersion`

Returns Cribl config version.

`C.os.hostname()`

Returns hostname of system running this Cribl instance.

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 compute Shanon entropy of.

@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 compute the hash of

@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, false otherwise.

`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, false otherwise.

`C.Text.relativeEntropy()`

(method) `Text.relativeEntropy(bytes: any, modelName?: string): number`

Computes the relative entropy of the given buffer or string

@param - bytes - value to compute relative entropy of

@param - string modelName - The name of the model to test string with.

@returns - the relative entropy value or -1 in case of an error

C.Time - Time Functions

`C.Time.strftime()`

(method) `Time.strftime(date: number | Date, format: string, utc?: boolean): string`

Format a `[Date][1]` or number as a time string using `[strftime specifier][2]` [1]: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date [2]: <https://github.com/d3/d3-time-format#api-reference>

@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 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][2]` - if successful a `[Date][1]` object is returned otherwise null.

[1]: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date [2]:

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

@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 local time

@param - strict - whether to return null if there are any extra characters after timestamp

@returns - the parsed date or null if the specifier did not match

`C.Time.timestampFinder()`

(method) `Time.timestampFinder(utc?: boolean): AutoTimeParser`

KNOWLEDGE

Regex Library

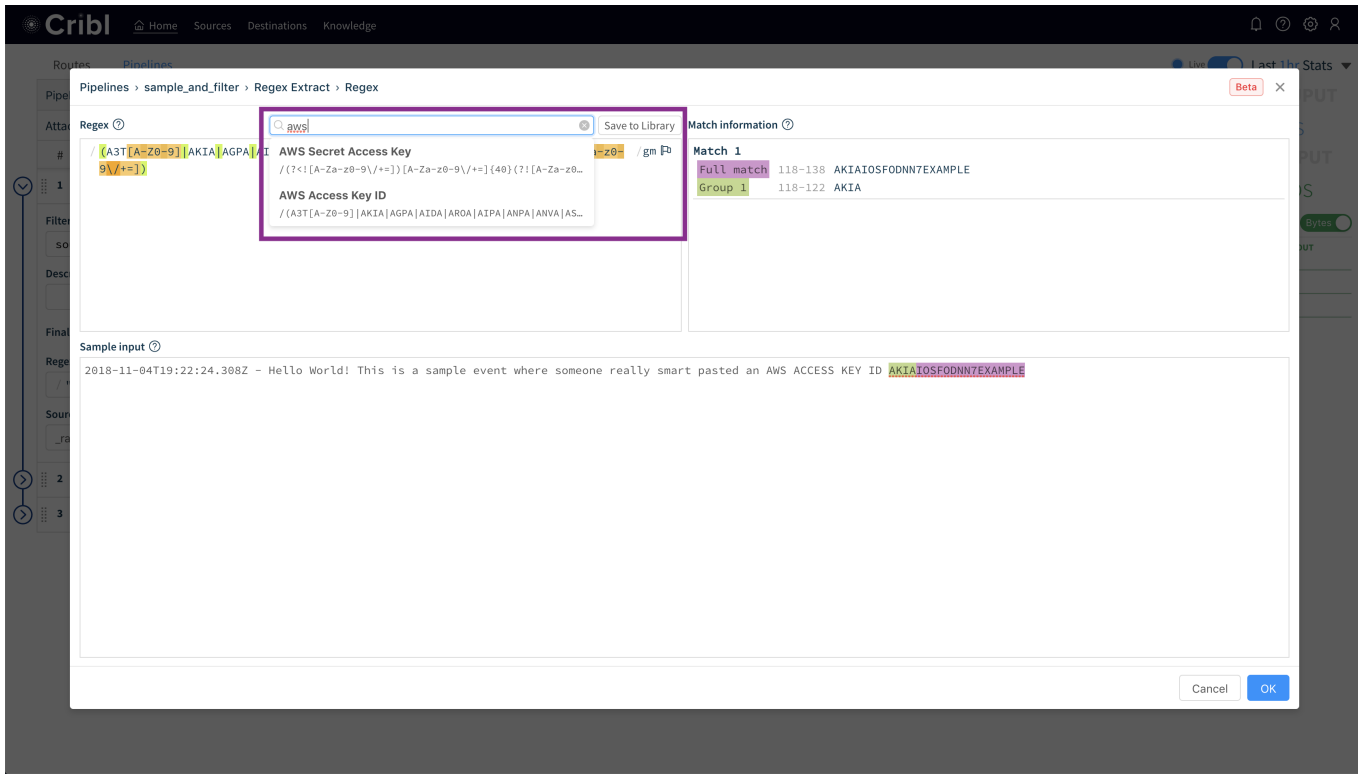
What is the Regex Library

As of v1.1 Cribl LogStream ships with a Regex Library that contains a set of pre-built common regex patterns. The goal of the library is to serve as an easily accessible repository of regular expressions. The library is searchable and each pattern can be tagged if further organization or categorization is needed. The library can be found under **Knowledge | Regex Library** .

Category	Regex	Tags
IPv4 address	/([0-1]?[0-9]?[0-4]?[0-5]?[0-9])\.[0-1]?[0-9]?[0-4]?[0-5]?[0-9]...	ip, ipv4, address
Email Address	/[a-zA-Z0-9_+&*~]+(?:\.[a-zA-Z0-9_+&*~]+)*@[a-zA-Z0-9...]	email, address
Bitcoin Address	/[13][a-km-zA-HJ-NP-Z1-9]{25,34}/gm	bitcoin, btc, address
Mastercard Credit Card	/5[1-5][0-9]{14}/gm	credit card, mastercard, pii
American Express Credit Card	/3[47][0-9]{13}/gm	credit card, american express, pii
Discover Credit Card	/6(?:011 5[0-9]{0-1})[0-9]{12}/gm	credit card, discover, pii
Visa Credit Card	/4[0-9]{12}[0-9]{3}/gm	credit card, visa, pii
International Bank Account Numbers (IBAN)	/[a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{4}[0-9]{7}([a-zA-Z0-9]?...	iban, pii
IPv6 address	/([0-9a-fA-F]{1,4}){7,7}([0-9a-fA-F]{1,4}) ([0-9a-fA-F]...	ip, ipv6, address
ISBN-10	/(?:ISBN(?:-10)?\s)?(?:=[0-9X]{10}\$ (?=[0-9]+[-\s])...	isbn, isbn-10
ISBN-13	/(?:ISBN(?:-13)?\s)?(?:=[0-9]{13}\$ (?=[0-9]+[-\s])...	isbn, isbn-13
MAC Address	/([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})/gm	mac, address
US Phone Numbers	/((([0-9]{1})+[-.()*(0-9){3}[-.])*[0-9]{3}[-.])*[0-9]...	us, phone, number
US Social Security Numbers	/\d{3}-?\d{2}-?\d{4}/gm	us, social, ssn

How does it work

As of this this version, the Library contains 25 patterns shipped by Cribl. A pattern can be used as-is in a Function or can be modified as necessary and new, custom patterns can be added by users.



Cribl vs. Custom and Priority

Patterns shipped by Cribl will be listed under the **Cribl** tab while those built by users will be found under **Custom**. Over time Cribl 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.

Event Breakers

What are Event Breakers

Event breakers are regex patterns and timestamp definitions that assist in breaking incoming streams of data into events. The Event Breakers management interface can be found under **Knowledge | Event Breakers** . Event Breaker rules and ruleset can be edited, added, deleted, searched and tagged as necessary.

Event Breaker Rulesets

Search: + Add New

Tags: All (5) Cribl (5) Custom (0)

ID	Description	Tags	Library						
▼ AWS Ruleset	Event breaking rules for common AWS data sources	flowlogs elb alb loadbalancer cdn	Cribl						
Id* AWS Ruleset									
Description Event breaking rules for common AWS data sources									
Tags flowlogs x elb x alb x loadbalancer x cdn x									
Rules	Rule Name	Filter Condition	Event Breaker	Timestamp Anchor	Timestamp Format	Default Timezone	Max Event Bytes	Fields	Actions
1	AWS VPC Flow	/^\d+(s+d+s+eni-w+*...)	[n\ r]+	(?= \d{10} \s \d{10})	Format: %s	UTC	1024		⌵ ⌵ ⌵
2	AWS ALB	/^(?:https? h2 wss?)\s\d...	[n\ r]+	\w+\s	Format: %Y-%m-%dT%...	Local	4096		⌵ ⌵ ⌵
3	AWS ELB	/^\d+-(\d+)-\d+*(?:(\d+),\...	[n\ r]+		Format: %Y-%m-%dT%...	Local	4096		⌵ ⌵ ⌵
4	AWS Cloudfront Web	/^\d+-(\d+)-(\d+)-(\d+)-\...	[n\ r]+		Format: %Y-%m-%d %...	UTC	4096		⌵ ⌵ ⌵

+ Add Rule

Clone Ruleset Advanced Mode

> Apache Ruleset	Event breaking rules for Apache Common and Combined logs	apache common combined	Cribl
> Cisco Ruleset	Event breaking rules for common Cisco data source	cisco asa estreamer	Cribl
> Palo Alto Ruleset	Event breaking rules for common Palo Alto data source	palo traffic threat system config	Cribl
> Bro Ruleset	Event breaking rules for Bro logs	bro	Cribl

How do Event Breakers work

Event Breaker Rules

Rules define configurations needed to break down a stream of data into discrete events.

Filter Condition: As a stream of data moves into the engine, a rule's filter expression is applied. If it evaluates to `true` , the rule configurations are engaged for the entire duration of that stream. Else, the next rule down the line is evaluated.

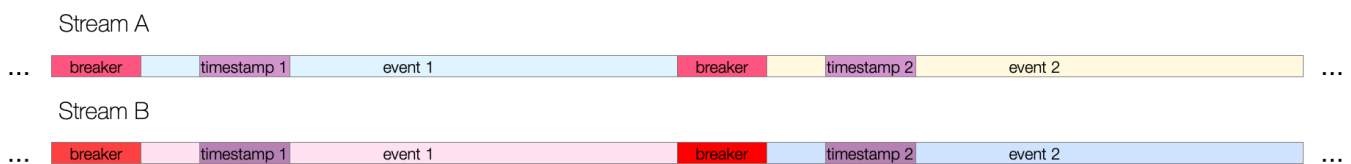
Event Breaker: After a breaker regex pattern has been selected it will apply on the stream **continuously**. Breaking will occur at the beginning of the match and the matched content will be consumed/thrown away. If necessary, a positive lookahead regex can be used e.g., `(?=pattern)` to keep the content. Capturing groups are **not allowed** to be used anywhere in the event breaker pattern as they will further break the stream. This is often

undesirable.

Breaking will also occur if Max Event Bytes has been reached. (See below for default value).

Timestamping: After events are synthesized out streams, timestamping will be attempted. First, a timestamp anchor will be located inside the event. Next, starting there, the engine will: try to scan up to a configurable depth into the event and autotimestamp, OR, timestamp using a manually supplied `strptime` format OR timestamp the event with 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 manual option, the anchor needs to lead the engine **right before** the timestamp pattern begins.



Fields: After events have been timestamped one or more fields can be added. Their values can be fully evaluated using JS expressions.

Rule Defaults:

Filter Condition defaults to `true`

Event Breaker to `[\n\r]+`

Timestamp anchor to `^`

Timestamp format to `Auto` and a scan depth of `150` bytes,

Max Event Bytes to `51200`

Default Timezone to `Local`

Rule Example: Break on newlines and use Manual timestamping **after** the sixth comma, as indicated by this pattern: `^(?:[^\,]*,){6}` .

Event Breaker Rule

Rule Name* Upload Sample File

Filter Condition* in Out

Event Breaker* in Out

Timestamp Anchor* in Out

Timestamp Format* Autotimestamp Scan Depth Manual Format Current Time

Default Timezone

Max Event Bytes

ADD FIELDS TO EVENTS

Timestamp Anchor
 Event Breaker
 Timestamp

Cancel

Event Breaker Rulesets

Rulesets are **collections of rules** that are associated with Sources. Rules within a ruleset are ordered and evaluated top->down. One ore more rulesets can be associated with a source and they too, are evaluated top->down. First rule that matches goes into effect for a stream from a source.

Rulesets and Rules - Ordered

Ruleset A

Rule 1

Rule 2

...

Rule n

...

Ruleset B

Rule Foo

Rule Bar

...

Rule FooBarn

Here's an example of 5 rulesets associated with a Source:

Manage Splunk Sources

+ Add New

ID	Host	Port	IP Whitelist Regex	TLS	Enabled	Status
local-splunk	192.168.1.12	10000	/:/	Disabled	On	●

Input Id*

Disabled No

Host*

Port*

Ip Whitelist Regex / R ↗

> **TLS SETTINGS (SERVER SIDE)**

> **EVENT BREAKERS**

Event Breaker Rulesets

1	AWS Ruleset	Event breaking rules for common AWS data sources (4 rules)	v x
2	Cisco Ruleset	Event breaking rules for common Cisco data source (3 rules)	v x
3	Palo Alto Ruleset	Event breaking rules for common Palo Alto data source (4 rules)	v x
4	Apache Ruleset	Event breaking rules for Apache Common and Combined logs (2 rules)	v x
5	Bro Ruleset	Event breaking rules for Bro logs (1 rule)	v x

System Default Rule Filter Condition: Event Breaker: Default Timezone: Max Event Bytes:

+ Add Ruleset

Event Breaker Buffer Timeout

> **ADVANCED SETTINGS**

Delete Source Clone SourceCancel Save

Default Rule: there is a system default rule that sits at the bottom of the Ruleset/Rule hierarchy that goes into effect if there are no matching rules. See Defaults above.

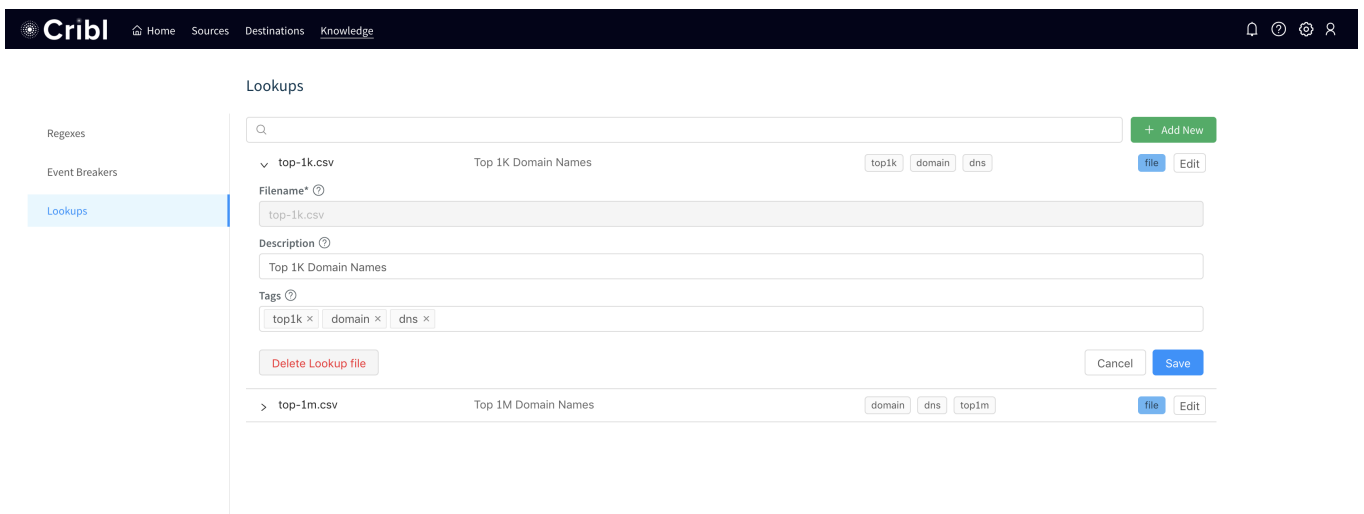
Cribl vs. Custom

Event Breaker Ruleset shipped by Cribl will be listed under the **Cribl** tab while those built by users will be found under **Custom**. Over time Cribl 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.

Lookups Library

What are Lookups

Lookups are data tables that can be used in Cribl to enrich events as they're processed by the Lookup Function. The Lookups library can be found under **Knowledge | Lookups** and its goal is to provide a management interface for all lookups. The library is searchable and each lookup can be tagged as necessary.



How does it work

The management interface allows for lookups to be added, deleted and edited. All files handled by the interface are stored in `$CRIBL_HOME/data/lookups`.

Edit Lookup - top-1k.csv

🔍 Search columns

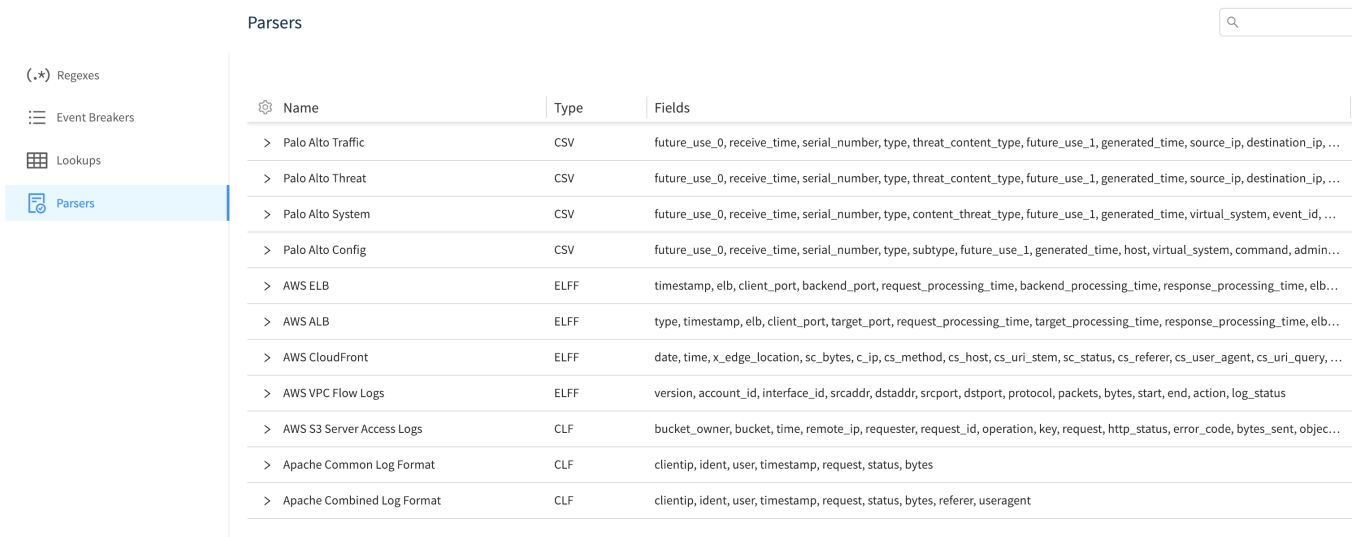
ID	rank	site
1	1	google.com
2	2	youtube.com
3	3	facebook.com
4	4	baidu.com
5	5	wikipedia.org
6	6	qq.com
7	7	taobao.com
8	8	yahoo.com
9	9	reddit.com
10	10	amazon.com
11	11	tmall.com
12	12	google.co.in
13	13	twitter.com
14	14	sohu.com
15	15	live.com
16	16	vk.com
17	17	yandex.ru
18	18	google.co.jp
19	19	sina.com.cn
20	20	weibo.com
21	21	instagram.com

Cancel Save

Parsers Library

What are Parsers

Parsers are definitions and configurations for the Parser Function. The library can be found under **Knowledge | Parsers** and its goal is to provide an interface for creating and editing Parsers. The library is searchable and each parsers can be tagged as necessary.



The screenshot displays the 'Parsers' library interface. On the left, a sidebar contains navigation options: '(.*') Regexes', 'Event Breakers', 'Lookups', and 'Parsers' (which is highlighted). The main area is titled 'Parsers' and includes a search bar. Below the search bar is a table listing various parsers.

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

USE CASES

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 leave it empty and apply it 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 & 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. The interface includes a title bar with a checkmark icon, a list icon, the number '2', the name 'Eval', and an 'On' toggle switch. Below the title bar, there are three main sections: 'Filter' with a search box containing 'sourcetype=='access_combined'', 'Description' with a text box containing 'Add 'dc' index-time field to events', and 'Final' with a toggle switch set to 'No'. At the bottom, there are two sections: 'Evaluate Fields' and 'Remove Fields', each with a blue 'Add Field' button.

- Next, let's click on Add Field, add our `dc` field and Save.

The screenshot shows the 'Eval' configuration window. At the top, there is a 'Filter' field containing the expression `sourcetype=='access_combined'`. Below it is a 'Description' field with the text 'Add 'dc' index-time field to events'. The 'Final' toggle is set to 'No'. Under 'Evaluate Fields', there is a table with one row:

Name	Value Expression
dc	'nyc-42'

Buttons for '+ Add Field' and '+ Add Field' are visible at the bottom.

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:

The screenshot shows the 'Eval' configuration window with an updated filter. The 'Filter' field now contains the expression `sourcetype=='access_combined' && host.startsWith('web-01')`. The 'Description' field remains 'Add 'dc' index-time field to events'. The 'Final' toggle is still 'No'. The 'Evaluate Fields' table is identical to the previous screenshot:

Name	Value Expression
dc	'nyc-42'

Buttons for '+ Add Field' and '+ Add Field' are visible at the bottom.

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

Removing fields can be done by either listing or wildcarding of field names. Let's see how we can remove all fields that start with `date_..`

- First make sure you have a route & pipeline configured to match desired events.
- Next, let's add a **Eval** function to it (similar to above)
- Next, in the Remove Fields section add `date_*` and hit Save.

The screenshot shows the configuration for an Eval function in Splunk. The interface includes the following sections:

- Filter:** A text input field containing the search filter: `sourcetype='access_combined' && host.startsWith('web-01')`.
- Description:** A text input field containing: `Add 'dc' index-time field to events`.
- Final:** A radio button control with 'No' selected.
- Evaluate Fields:** A table with two columns: 'Name' and 'Value Expression'.

Name	Value Expression
dc	'nyc-42'
- Remove Fields:** A text input field containing `date_*`.

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, say `.csv` files we use the 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`
- Temporally correct looked-up information is required. For example, when you have a highly dynamic infrastructure and you need to resolve a resource (e.g. a container) name to its address you can't afford to do it at search/run-time as the resource and its records may no longer exist. External (non `.csv`) lookups are coming soon.

Working with lookups - Example 1

Let's assume we have the following lookup file and given the field `conn_state` in an event we would like to add a corresponding ingestion-time field called `action`

bro_conn_state.csv

```
action,"conn_state","conn_state_meaning"
dropped,S0,"Connection attempt seen, no reply."
allowed,S1,"Connection established, not terminated."
allowed,SF,"Normal establishment and termination."
blocked,REJ,"Connection attempt rejected."
allowed,S2,"Connection established and close attempt by originator seen (but no reply from res
allowed,S3,"Connection established and close attempt by responder seen (but no reply from orig
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 SYN-ACK from the resp
dropped,RSTRH,"Responder sent a SYN ACK followed by a RST, we never saw a SYN from the (purpor
dropped,SH,"Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder
dropped,SHR,"Responder sent a SYN ACK followed by a FIN, we never saw a SYN from the originatc
allowed,OTH,"No SYN seen, just midstream traffic (a 'partial connection' that was not later cl
```

- First make sure you have a route & pipeline configured to match desired events.
- Next, let's add a Lookup function to it with these settings:

Lookup file path: \$SPLUNK_HOME/etc/apps/Splunk_TA_bro/lookups/bro_conn_state.csv

Note that Environment variables are allowed in 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`

2 Lookup On ×

Filter ?
sourcetype=='bro'

Description ?
Add ingest-time field action to all events with sourcetype bro

Final ?
 No

Lookup file path (.csv)* ?
\$SPLUNK_HOME/etc/apps/Splunk_TA_bro/lookups/bro_conn_state.csv

Lookup field(s) ?

Lookup Field Name in Event ?	Corresponding Field Name in Lookup ?	
conn_state	conn_state	×

+ Add field(s)

Output field(s) ?

Output Field Name from Lookup ?	Lookup Field Name in Event ?	
action	action	×

+ Add field(s)

To confirm, verify that this search returns expected results: `sourcetype="bro" action::allowed` . Change `action` value as necessary.

Working with lookups - Example 2

Let's assume we have the following lookup file and given **both** 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 & pipeline configured to match desired events.
- Next, let's add a Lookup function to it 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 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

2 Lookup
On X

Filter ?

Description ?

Final ?

 No

Lookup file path (.csv)* ?

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 ?	
⋮	severity	severity	X

+ Add field(s)

To confirm, verify that this search returns expected results: `sourcetype="cisco:sourcefire" severity::medium`. Change `severity` value as necessary.

Sampling

Sampling at ingest-time

Let's say that you wanted troubleshoot with and analyze **highly verbose/voluminous** data, for example, CDN logs, ELB Access Log or VPC Flows but you were concerned about storage requirements and search performance. With Sampling you can bring in enough samples so that your analysis remains statistically significant but you can also do all the troubleshooting necessary.

See the example below or more details: 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' 5:1 (and all others 1:1). This should lower the volume of all verbose successes (200s) but still bring in **ALL** potentially erroneous events (400s, 500s etc) that can be used for troubleshooting.

- First make sure you have a route & pipeline configured to match desired events.
- Next, let's add a **Regex Extract** function and extract the status field from `_raw` and let's call it `__status`. Remember, fields that start with `__` are special fields in Cribl and can be used anywhere in a pipeline.



The screenshot shows the configuration for a 'Regex Extract' function in Cribl. The interface includes a 'Filter' field with the value `sourcetype=='access_combined'`, a 'Description' field with the text 'Extract status from access logs', a 'Final' toggle set to 'No', a 'Regex' field with the pattern `/"\s(?:<__status>\d+)/`, and a 'Source Field' field with the value `_raw`. The function is currently turned 'On'.

Next, let's add a **Sampling** function, 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
On

Filter ?

Description ?

Final ? No

Sampling Rules ?

	Filter ?	Sampling Rate ?	
⋮	__status == 200 ?	5	×

+ Add Rule

To confirm that sampling works, compare the event count of all 200 s before and after. In addition, each time an event goes thru the Sampling function an index-time field is added to it `sampled::<rate>` . You can use that to 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, CDN logs, such as those from Amazon Cloudfront, Amazon S3 Server Access Logs, AWS ELB Access Logs etc. For large installations, oftentimes bringing in everything to an analytics tool is 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), and the other much larger majority, contains information when looked at in aggregate (e.g., successes to determine traffic patterns etc.). It would be great if we could find a middle ground. With Cribl Sampling you can.

- Ingest enough sample events from the majority category so 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. **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+)/`

2 Regex Extract On

Filter [?](#)

`sourcetype=='access_combined' || sourcetype=='aws:s3:accesslogs'`

Description [?](#)

Extract status from access logs

Final [?](#) No

Regex [?](#)

`/HTTP\/\d\.\d"\s(?<__status>\d+)/`

Source Field [?](#)

`_raw`

3. Add a Sampling function to sample 5:1 when `__status==200`

4. Save.

3 Sampling On

Filter [?](#)

`sourcetype=='access_combined' || sourcetype=='aws:s3:accesslogs'`

Description [?](#)

Check for status 200 and sample 5:1

Final [?](#) No

Sampling Rules [?](#)

Filter ?	Sampling Rate ?
<code>__status == 200</code>	5

[+ Add Rule](#)

Note About Sampling

Each time an event goes thru 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 may need a different `__status` extraction regex:

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] (https://www.cisco.com/c/en/us/td/docs/security/asa/syslog/b_syslog.html), and other technologies such as Juniper, Checkpoint, pfSense etc.

Similar to Access Logs, bringing in **everything** for operational analysis may be cost-prohibitive and Sampling with Cribl can help.

- Ingest enough sample events from the majority category so 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

VPC Flow Logs is a feature that enables you to capture information about the IP traffic going to and from network interfaces in your VPC. Flow log data can be published to Amazon CloudWatch Logs and Amazon S3.

Typical VPC Flow Logs look like this:

Flow Log Records for Accepted and Rejected Traffic

```
2 123456789010 eni-abc123de 172.31.16.139 172.31.16.21 20641 22 6 20 4249 1418530010 141853007
2 123456789010 eni-abc123de 172.31.9.69 172.31.9.12 49761 3389 6 20 4249 1418530010 141853007
```

Let's use a **very simple** filter condition and only look for `ACCEPT s`.

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)/`

2 Regex Extract
On

Filter ?

Description ?

Final ? No

Regex ?

Source Field ?

3. Add a Sampling function to sample 5:1 when `__action=="ACCEPT"`
4. Save.

3 Sampling
On

Filter ?

Description ?

Final ? No

Sampling Rules ?

Filter ?	Sampling Rate ?
<div style="display: flex; align-items: center;"> ⋮ <input style="margin-left: 5px;" type="text" value="__action == 'ACCEPT'"/> ✖ </div>	5

+ Add Rule

Note About Sampling

Each time an event goes thru 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 may need a different `__action` extraction regex:

Cisco ASA Logs

`sourcetype=='cisco:asa'`

Related sourcetypes to consider sampling:

sourcetype=='cisco:fwsn'

sourcetype=='cisco:pix'

Masking and Obfuscation

Masking and anonymization of data in motion.

To mask patterns in real-time we use the out-of-the-box **Mask** function. This is similar to `sed` but with much powerful functionality.

Masking Capabilities

The Masking function accepts multiple replacement rules and 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:

4 Mask 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

[+](#) Add Field

Masking Examples

Let's look at the **various** ways that we can mask a string like this: `cardNumber=214992458870391` . The **Regex Match** we'll use is: `/((cardNumber=)(\d+)/g` . In this example:

- `g0` = `cardNumber=214992458870391`
- `g1` = `cardNumber=`
- `g2` = `214992458870391`

i Replace Expression Evaluation

Replace Expression 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"``

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.` 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

Random Masking with default character length (4):

- **Replace Expression:** ``${g1}${C.Mask.random()}``
- **Result:** `cardNumber=HRhc`

Random Masking with defined character length:

- **Replace Expression:** ``${g1}${C.Mask.random(7)}``
- **Result:** `cardNumber=neNSm8r`

Random Masking with length preserving replacement:

- **Replace Expression:** ``${g1}${C.Mask.random(g2)}``
- **Result:** `cardNumber=DroJ73qmyaro51u3`

Repeat Masking with default character length (4):

- **Replace Expression:** ``${g1}${C.Mask.repeat()}``
- **Result: Result:** `cardNumber=XXXX`

Repeat Masking with defined character choice and length:

- **Replace Expression:** ``${g1}${C.Mask.repeat(6, 'Y')}``
- **Result:** `cardNumber=YYYYYY`

Repeat Masking with length preserving replacement:

- **Replace Expression:** ``${g1}${C.Mask.repeat(g2)}``
- **Result:** `cardNumber=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.

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 that contain the pattern `Opera` in `_raw` :

- First make sure you have a route & 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. The interface includes a title bar with a dropdown arrow, a tab labeled '4 Regex Filter', and an 'On' toggle switch. The configuration fields are as follows:

- Filter**: `sourcetype='access_combined'`
- Description**: Filter out events that contain 'Opera'
- Final**: No
- Regex**: `/Opera/g`
- Field**: `_raw`

To confirm, 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 `bar.com` , change the filter input as below:

4 Regex Filter On

Filter ⓘ
`sourcetype=='access_combined' && host.endsWith('bar.com')`

Description ⓘ
Filter out events that contain 'Opera'

Final ⓘ No

Regex ⓘ
`/Opera/g`

Field ⓘ
_raw

This is a very flexible method for filtering incoming events in real-time on almost any arbitrary conditions.

Encrypting Sensitive Data

Encryption at ingest-time

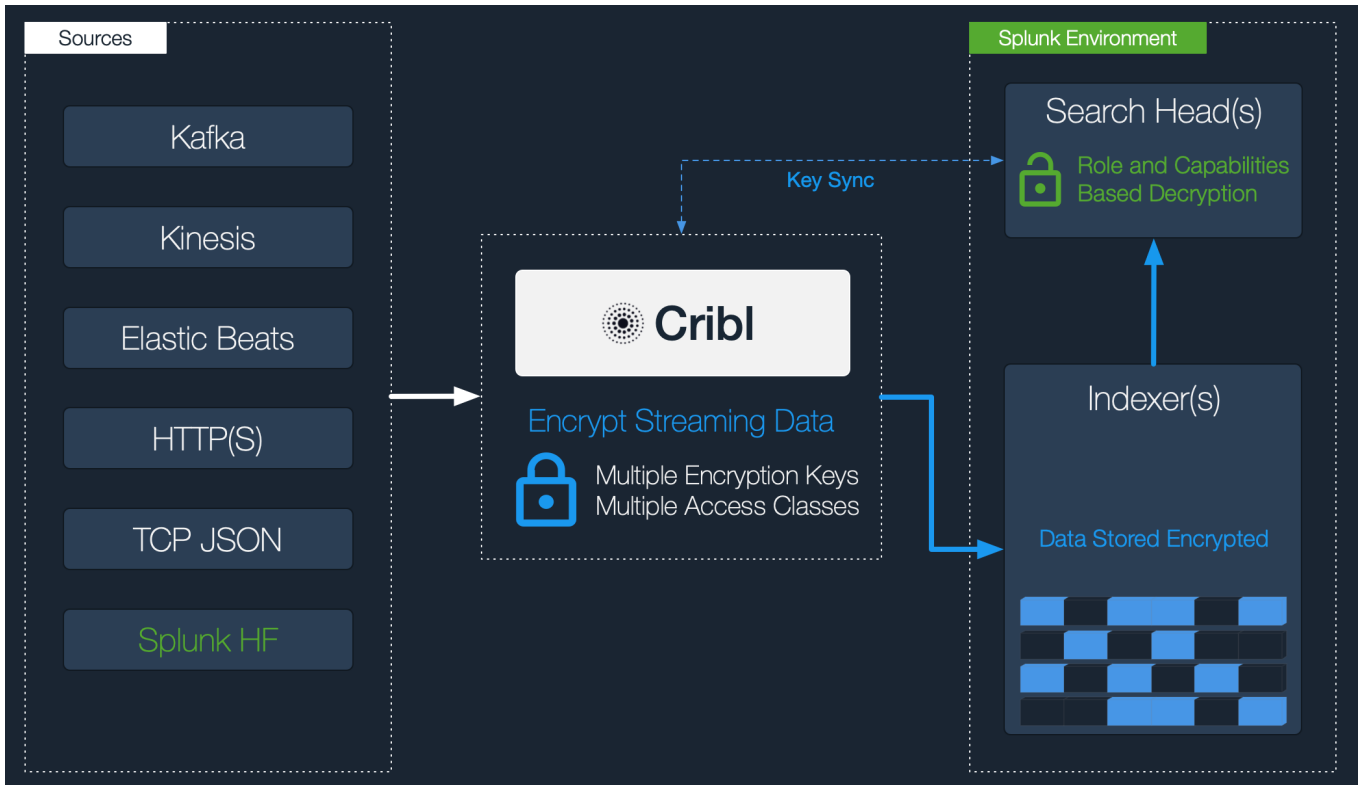
With Cribl you can encrypt your sensitive data in real-time before it's forwarded to and stored at a destination. Using the out-of-the-box **Mask** function you can define patterns to encrypt with specific key IDs or key classes.

Keys and Key Classes

Symmetric key encryption keys can be configured through the CLI or UI. They're used to encrypt the patterns and users are free to define as many keys as required. Key Classes are collection 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.

Encrypting in Cribl and decrypting in Splunk

1. Define one or more Keys and Key Classes on Cribl.
2. Sync `auth` with decryption side (Splunk Search Head)
3. Apply the Mask function with `C.Crypto.encrypt()` to patterns of interest
4. Decrypt on Splunk Search Head using Role Based Access Control on `decrypt` command.



Example

Encryption Side

- Generate one or more keys through via the CLI as such:

```
$CRIBL_HOME/bin/criblctl keys add -c 1 -i
```

...

```
$CRIBL_HOME/bin/criblctl keys add -c <N> -i
```

Add `-e <epoch>` if you'd like to set expiration for your keys.

Or via UI in Settings | Encryption Keys:

Encryption Keys

Get Key Bundle

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
v 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

Description

Encryption Algorithm*

KMS for this key*

Key Class*

Expiration time*

- Sync `auth/(cribl.secret|keys.json)` . To decrypt data the `decrypt` command will need access to these keys. `cribl.secret` and `keys.json` in `$(CRIBL_HOME)/local/cribl/auth` should be synced/copied over to the Search Head (decryption side).

Decryption Side

- Install Cribl App for Splunk on your Search Head. It will default in `mode-searchhead` .
- Assign permissions to the `decrypt` command per your requirements.
- Assign capabilities to your Roles per your requirements. Capability names should follow the format `cribl_keyclass_N` where `N` is the Cribl Key Class. For example, a role with capability `cribl_keyclass_1` has access to all key ids associated with key class `1` . You can use more capabilities as long as they follow this naming convention.

Capabilities

Select specific capabilities for this role.

Available capabilities add all »

- accelerate_datamodel
- accelerate_search
- admin_all_objects
- change_authentication
- change_own_password
- **cribl_keyclass_0**
- **cribl_keyclass_1**
- **cribl_keyclass_2**
- **cribl_keyclass_3**
- **cribl_keyclass_4**
- delete_by_keyword
- dispatch_rest_to_indexers
- edit_cmd
- edit_deployment_client
- edit_deployment_server
- edit_dist_peer
- edit_encryption_key_provider

→

Selected capabilities « clear all

- ← accelerate_datamodel
- ← admin_all_objects
- ← change_authentication
- ← **cribl_keyclass_2**
- ← **cribl_keyclass_3**
- ← dispatch_rest_to_indexers
- ← edit_cmd
- ← edit_deployment_client
- ← edit_deployment_server
- ← edit_dist_peer
- ← edit_encryption_key_provider
- ← edit_forwarders
- ← edit_health
- ← edit_httppaths
- ← edit_indexer_cluster
- ← edit_indexerdiscovery
- ← edit_input_defaults

Usage

Before Encryption: sample un-encrypted events. Notice values of `fieldA` and `fieldB`

index=main source=*wifi.log fieldA=* OR fieldB=* OR fieldC=* All time 🔍

✓ 4 events (before 1/5/19 1:11:25.000 PM) No Event Sampling

Events (4) Patterns Statistics Visualization

Format Timeline Zoom Out Zoom to Selection Deselect 1 hour per column

Show Fields List Format 20 Per Page

i	Time	Event
>	12/20/18 12:44:02.573 PM	Sat Dec 20 12:44:02.573 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:03.467 AM	Sat Dec 20 09:15:03.467 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:02.590 AM	Sat Dec 20 09:15:02.590 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 12:57:42.493 AM	Sat Dec 20 00:57:42.493 <kernel> fieldA=TOPSECRET fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267

Encrypting `fieldA` values with key class 1 and `fieldB` with key class 2

Filter ?

true

Description ?

Final ?



Capture field values

Encrypt

Masking Rules*

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 x

After Encryption: again, notice values of fieldA and fieldB

index=main source=wifi.log fieldA=* OR fieldB=* OR fieldC=* All time

4 events (before 1/5/19 1:19:21.000 PM) No Event Sampling

Events (4) Patterns Statistics Visualization

Format Timeline Zoom Out Zoom to Selection Deselect 1 hour per column

i	Time	Event
>	12/20/18 12:44:02.573 PM	Sat Dec 20 12:44:02.573 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::1Nbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:03.467 AM	Sat Dec 20 09:15:03.467 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::1Nbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:02.590 AM	Sat Dec 20 09:15:02.590 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::1Nbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 12:57:42.493 AM	Sat Dec 20 00:57:42.493 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=#3::1Nbbp0LA8LEH7nYwmFsFBg# fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267

Decrypting fieldB but not fieldA . Logged in user has been assigned capability cribl_keyclass_2

index=main source=wifi.log fieldA=* OR fieldB=* OR fieldC=* | decrypt All time

4 events (before 1/5/19 1:23:16.000 PM) No Event Sampling

Events (4) Patterns Statistics Visualization

Format Timeline Zoom Out Zoom to Selection Deselect 1 hour per column

i	Time	Event
>	12/20/18 12:44:02.573 PM	Sat Dec 20 12:44:02.573 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:03.467 AM	Sat Dec 20 09:15:03.467 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 9:15:02.590 AM	Sat Dec 20 09:15:02.590 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267
>	12/20/18 12:57:42.493 AM	Sat Dec 20 00:57:42.493 <kernel> fieldA=#2::IB76602//ZNKNrgyhQnJ0g# fieldB=SECRET fieldC=MODERATELYSECRET IOCTL (from pid 183) not recognized: 5 out of 267

KNOWN ISSUES

Known Issues

In-product upgrade issue on v2.0 (2019-10-22)

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.

In-product upgrade issue on v1.7. (2019-08-27)

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.

S3 stagePath issue on upgrade. (2019-03-21)

Problem: When upgrading from v1.2 with a S3 output configured `stagePath` was allowed to be undefined. In v1.4+ it is a required field and may causing schema violations on older configs when upgrading.

Workaround/Fix: Re-configure the output with a valid `stagePath` filesystem path.

THIRD PARTY SOFTWARE

Current List

azure/storage-blob: 10.3.0
ag-grid-community: 19.1.2
ag-grid-react: 19.1.2
ajv: 6.9.2
ajv-errors: 1.0.1
antd: 3.13.0
as-table: 1.0.36
avsc: 5.4.9
aws-sdk: 2.323.0
cidr-matcher: 1.0.5
classnames: 2.2.6
color-hash: 1.0.3
cookie-parser: 1.4.3
d3-time-format: 2.1.3
date-fns: 1.29.0
diff: 3.5.0
escodegen: 1.11.1
esprima: 4.0.1
express: 4.16.3
fast-bitset: 1.3.2
file-saver: 1.3.8
jwt-simple: 0.5.6
kafkajs: 1.4.5
lodash: 4.17.15
lz4js: 0.2.0
node-cache: 4.2.0
node-uuid: 1.4.8
numeral: 2.0.6
pako: 1.0.10
papaparse: 5.0.0-beta.0
query-string: 6.1.0

react: 16.7.0
react-dom: 16.7.0
react-jsonschema-form: 1.0.3
react-router-dom: 4.3.1
react-sortable-hoc: 0.8.3
react-split-pane: 0.1.82
regexp: 2.0.0
requirejs: 2.3.6
resize-observer-polyfill: 1.5.0
rxjs: 6.2.2
saxen: 8.1.0
streamcount: 1.0.1
tar-stream: 1.6.1
url: 0.11.0
winston: 3.0.0
xmlbuilder: 10.0.0
yaml: 1.3.2