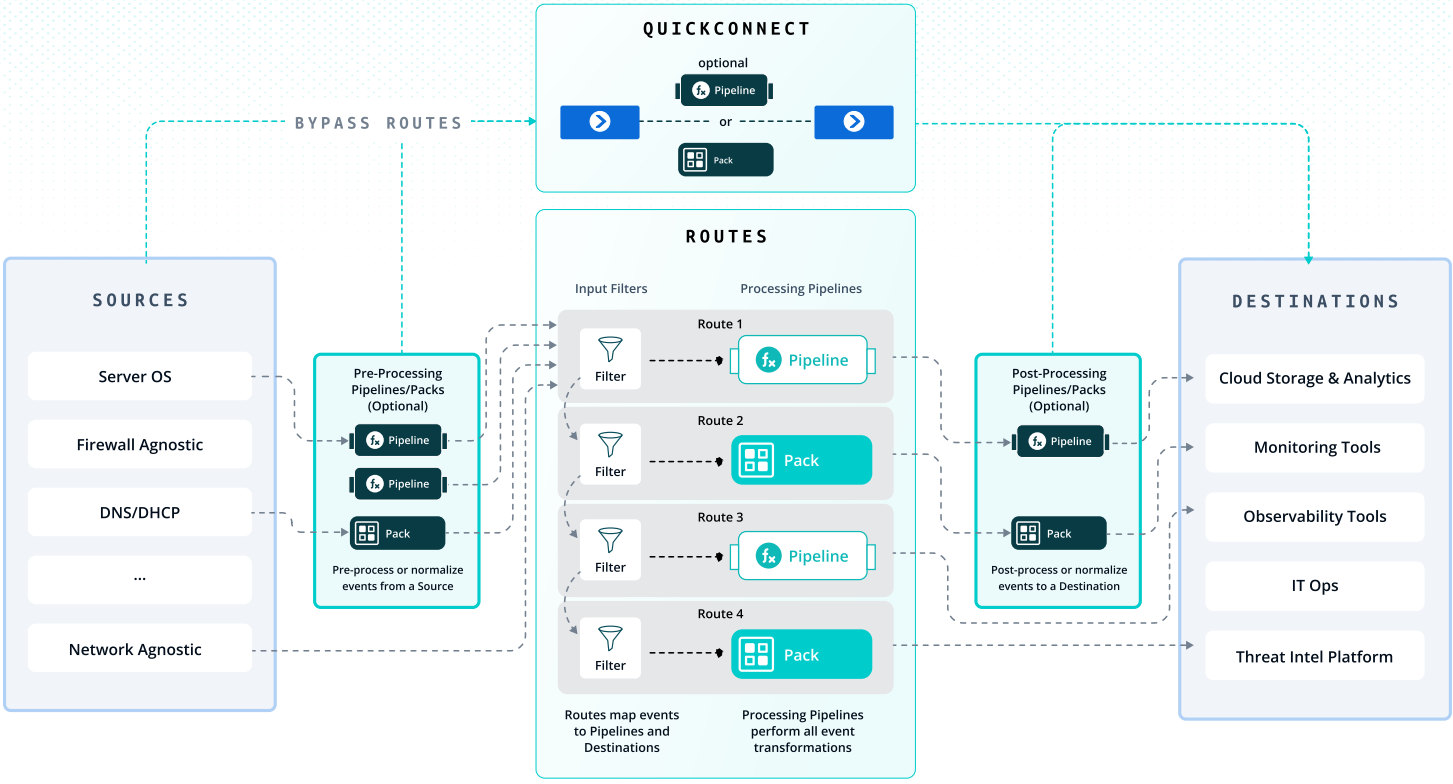
Tip Sheet: Cribl Stream™



Basic Concepts

SOURCES

Cribl Stream can process data from various metrics, logs, traces, and generic event sources, including Splunk, HTTP, Elastic Beats, Kinesis, Kafka, TCP JSON etc. Depending on the Source, both **push** and **pull** methods are supported.

DESTINATIONS

Cribl Stream can send data to various Destinations, including Exabeam, Splunk, SignalFx, Kafka, Elasticsearch, Kinesis, InfluxDB, Snowflake, Databricks, Honeycomb, Azure Blob Store, Azure EventHubs, TCP JSON, Wavefront, and many others. Destinations can be streaming (events are sent in real time) or non-streaming (events are sent in batches).

ROUTES

Routes evaluate incoming events against **filter** expressions to find the appropriate **Pipeline** to send them to. Routes are evaluated in order, and a Route can be associated only with one Pipeline and one Destination. However, one event may traverse multiple Routes.

PIPELINES

A series of **Functions** is called a Pipeline, and the order in which they are executed matters. Events are delivered to the beginning of a Pipeline by a Route, and as they're processed by a Function, they are passed onto the next Function down the line. Events only move forward, towards the end of the Pipeline and eventually out of the system.

Basic Concepts (cont.)

FUNCTIONS

A Function is a piece of **JavaScript** code that executes on an event, and it encapsulates the smallest amount of processing that can happen to that event. E.g., a Function can replace the term `foo` with `bar` on each event. Another one can hash bar, and yet another can add a field, say, `dc=jfk-42` to any event from host `us-nyc-42.cribl.io`.

PACKS

Packs enable Cribl Stream administrators and developers to pack up and share complex configurations and workflows across multiple Worker Groups, or across organizations. Packs can contain everything between a Source and a Destination: Routes (Pack-level), Pipelines (Pack-level), Functions (built-in and custom), Sample data files, Knowledge objects (Lookups, Parsers, Global Variables, Grok Patterns, and Schemas). Wherever you can reference a Pipeline, you can specify a Pack!

COLLECTOR SOURCES

Collector Sources are designed to ingest data intermittently, rather than continuously. You can use Collectors to dispatch on-demand (ad hoc) collection tasks, which fetch or "replay" (re-ingest) data from local or remote locations. Collectors also support scheduled periodic collection jobs that can make batch collection of stored data more like continual processing of streaming data. Supported Collectors include (but are not limited to): Filesystem / NFS, Azure Blob, Google Cloud Storage, 53 (including MinIO), REST, and Splunk Search.

QUICKCONNECT

QuickConnect is a visual rapid-development UI. With it, you can connect Cribl Stream inputs (Sources) to outputs (Destinations) through simple drag-and-drop. You can then insert Pipelines or Packs into the connections, to take advantage of Cribl Stream's full range of data-transformation Functions. Or you can omit these processing stages entirely, to send incoming data directly to Destinations - with minimal configuration fuss.

SCALING AND SIZING

Expected resource utilization will be proportional to how much overall processing is occurring. For instance, a Function that adds a static field will likely perform faster than one that applies a regex to finding and replacing a string. Cribl's current sizing guidance is **400GB thru/day/CPU**. For example:

Input:	**4TB/day**
Outputs:	**4TB/day to S3** **2TB/day to Splunk**
Total thru:	**10TB/day**
Est. CPUs:	**25 (10TB/400GB)**

CRIBL.CLOUD

The fast alternative to downloading and self-hosting Cribl Stream software is to launch Cribl.Cloud. This Saas version, whether free or paid, places the Leader and the Worker Nodes/Edge Nodes in Cribl.Cloud, where Cribl assumes responsibility for managing the infrastructure.

EVENT MODEL

All data processing is based on discrete data entities commonly known as **events**.

An event is generally defined as a collection of key/value pairs (fields). Some Sources deliver discrete events directly, while others might deliver bytestreams that need to be broken up by Event Breakers. The internal representation of a Cribl Stream event looks like this:

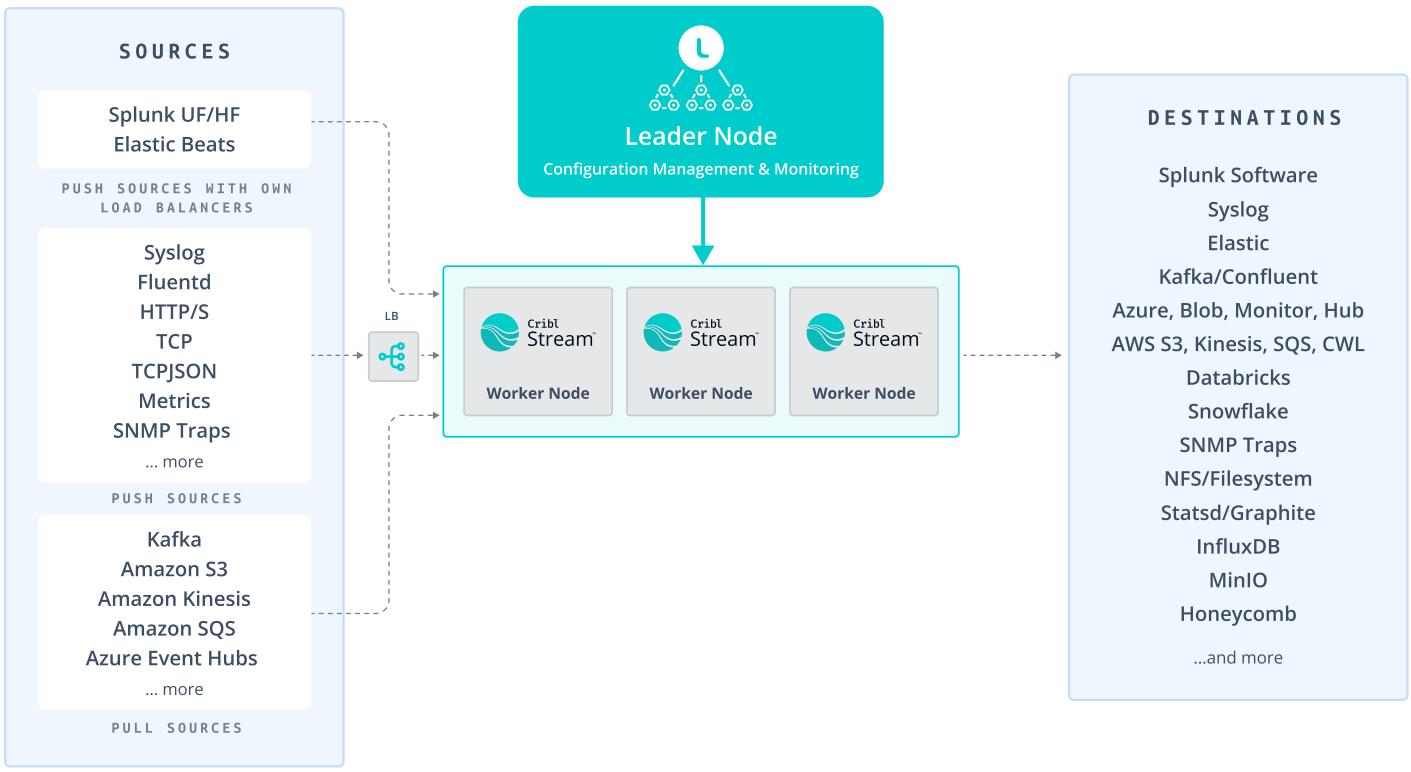
```
{
"_raw": "<body of non JSON parse-able event›",
"_time": "<timestamp in UNIX epoch format>",
"_inputId": "«Source of the event>",
"_other1": "«Internal field 1>",
"_other2": "«Internal field2>",
"_otherN": "<Internal fieldN›",
"key1": "<value1>",
"key2": "<value2>",
"key3": "<value3>",
"keyN": "<valueN>",
' ...":" ..."
}
```

Basic Concepts (cont.)

Fields that with start with a double underscore are internal to Stream. For example, syslog sources add both an `__inputId` and a `__srcIpPort` field to each event. Internal fields can be used in a Pipeline, but are not passed down to Destinations. If an event cannot be JSONparsed, all of its content will be assigned to the `_raw` field. If a timestamp is not configured, or cannot be extracted from an event, Stream will assign the current time (in UNIX epoch format) to `_time`.

DEPLOYMENTS

When data volume is low, and/or the amount of processing is light, a single-instance deployment may be sufficient. To accommodate higher volume, increased processing complexity, and increased availability, Cribl Stream can be scaled up and out across multiple instances. This is known as a **distributed deployment**. Distributed deployments are not limited to on-premises only or Cribl.Cloud only; Stream is also capable of **hybrid deployments**.



FILTERS AND VALUE EXPRESSIONS

You can use JavaScript filters and other value expressions to configure Stream's Routes and built-in Functions. Expressions are syntactically valid units of code that resolve to a value. Conceptually, Stream supports two types of expressions: Some assign a value to a field – e.g., `myAnswer=42`. Others evaluate to a value, – e.g., `(Math.random*42)`. Filters are expressions that must evaluate to either true (or truthy) or false (or falsy). You can use filters in Routes to select a subset of incoming data flow, and in Functions to scope or narrow down their applicability.

Some simple examples:

Filter: Check if incoming events are from host `foo` or filename ends in `.log`:

```
host=='foo' || source.endsWith('.log')
```

Expression: Assign field `sourcetype` the value of `cisco:asa` if string `%ASA` is in `_raw`, else leave it as is.

```
/%ASA/.test(_raw) ? 'cisco:asa' : sourcetype
```

Cribl Suite of Products

Cribl, the Data Engine for IT and Security, empowers organizations to transform their data management strategy. Powered by a data processing engine purpose-built for IT and Security, Cribl's product suite is a vendor-agnostic data management solution capable of collecting data from any source, processing billions of events per second, automatically routing data for optimized storage, and analyzing any data, at any time, in any location. With Cribl, IT and Security teams have the choice, control, and flexibility required to adapt to their ever-changing data needs. Cribl's offerings — Stream, Edge, Search, and Lake — are available either as discrete products or as a holistic solution.

CRIBL EDGE

Cribl Edge brings Cribl Stream's processing power to servers. This means all functionality, such as integrations with nearly all observability and security tools, is available directly. With built-in data capture and interactive preview, our rich, interactive, data-centric experience is now available where the data originates. We can now collect log data and system metrics directly from the Edge Nodes. Using one agent, we can feed a time-series database, SIEM, logging tool, and data lake.

Cribl Edge automatically discovers logs, metrics, application data, etc. – in real time – from your configured endpoints, and delivers them to Cribl Stream or any supported destination. Cribl Stream can then collect, reduce, enrich, transform, and route data from Cribl Edge to any destination. By using a Cribl HTTP/TCP Source, you can route data from Edge Nodes to Stream Worker Nodes connected to the same Leader without incurring additional costs.

CRIBL SEARCH

With Cribl Search, you can search, explore, and analyze machine data — logs, instrumentation, application, metrics, etc. — in place without moving it to specialized storage. The data can reside in Cribl Edge or other low-cost storage, like Cribl Lake or AWS S3.

By combining Cribl Search and Cribl Stream, administrators can create a data loop that allows them to:

1. Search for specific data in object stores using Cribl Search
2. Redirect the data to their data lake using Cribl Stream, creating a small, easier-to-sift dataset.

They can then send the resultant data into their SIEM without sending superfluous data with it.

CRIBL LAKE

Cribl Lake gives you a way to easily store, manage, enforce policy on, and access data. As mentioned above, you can set up Stream to send data to any data lake. Teams that choose Cribl Lake get a turnkey data lake solution that brings together large volumes of data from disparate sources, makes it easy to share, and possible to retrieve for future use. With Cribl Lakehouse, a groundbreaking evolution within Cribl Lake, IT and security teams can effortlessly store massive volumes of ever-changing telemetry data while enabling real-time, high-performance dashboards and analytics, all without requiring data engineering expertise. Check out our documentation to learn how to configure Cribl Stream to output to Cribl Lake.

CRIBL PRODUCT TIP SHEETS

Yes there is a Tip Sheet for that! Cribl Edge and Cribl Search also have Tip Sheets with myriad product specific knowledge. Check them out! For more info on Cribl Lake, our docs are the way.

Performance Tips

`Start On Boot (systemd)`

Cribl Stream can be configured to start at boot time with **systemd.** To do this, run the `boot-start` command. Make sure you first create a user you want to specify to run Cribl Stream.

Example: To run Cribl Stream on boot as existing user `cribl` use:

```
sudo $CRIBL_HOME/bin/cribl boot-start enable-m
systemd-u cribl
```

The above will install a unit file named `cribl.service`, and will start Cribl Stream at boot time as user `cribl`.

Note: A `-configDir` option can be used to specify where to install the unit file. If not specified, this location defaults to `/etc/systemd/system/.*`

It may be necessary to change ownership for the Cribl Stream installation (`[sudo] chown-R cribl $CRIBL_HOME`). Finally, enable Stream to ensure that the service starts on system boot (`[sudo] systemctl enable cribl`).

Available systemctl commands are:
`systemctl[start|stop|restart| status]cribl`

`JSON.parse(_raw)`

JSON events are received as strings in Cribl Stream. In order to improve performance (and reduce event size) try adding an Eval function to the beginning of your pipeline where **Name** is `_raw` and **Value Expression** is set to `JSON.parse(_raw)`. This will quickly convert the JSON back from a string (faster than using a parser) AND slightly shrink the overall event!

Regex Lookups

When using `C.LookupRegex` be wary of empty new lines. Regex lookups return true when a file has a dangling new line since these are treated as a wildcard. So remember: When making lookup files — don't leave an empty line at the end!

Filtering Function Performance

The filters in your Cribl Stream Routes and Pipelines are like the gasoline in your car's engine. The better the fuel, the better the engine runs. The better your Stream filters, the faster your Routes and Pipelines can process observability data. Observability at scale requires careful attention to minor performance items, such as the choice of the function in your filter. A good rule of thumb is: regex matching functions such as `match, test, search` will typically take more CPU to process than string matching functions such as `indexOf`, `includes`, and `startsWith`, etc.

_raw.split

Multi-line logs and multi-value arrays can be tough to deal with. Navigate the logs easier by adding an Eval to the beginning of your pipeline where **Name** is `_raw` and **Value Expression** is set to `_raw.split('\n').` This will quickly cut up the multiline log file for easy parsing later on. You can even reference specific lines in the log by using `_raw.split('\n')[i]` and substituting i with the desired line!

Using Built-in Functions

Stream ships with a growing collection of highly configurable, out-of-the-box Functions. Below, we list key built-in Functions by purpose, followed by available JavaScript methods and Cribl expressions.

- Create, remove, update, rename fields Functions: **Eval, Rename, Lookup, Regex Extract, Grok**
- Find & Replace, including basic sed-like, obfuscate, redact, hash etc.: **Mask, Eval**
- Add GeoIP information to events: **Lookup, GeoIP**
- Extract fields from structured and unstructured events: **Regex Extract, Parser**
- Extract and assign timestamps: **Auto Timestamp**
- Drop events: **Drop, Regex Filter, Sampling, Suppress, Dynamic Sampling**
- Sample events (e.g., high volume, low value data): **Sampling, Dynamic Sampling**
- Suppress events (e.g., remove duplicates etc.): **Suppress**
- Convert JSON arrays or XML elements into own events: **Unroll, JSON Unroll, XML Unroll**
- Serialize events to CEF format (send to various SIEMs): **CEF Serializer**
- Serialize / change format (e.g., convert JSON to CSV): **Serialize**
- Flatten nested structures (e.g., nested JSON): **Flatten**
- Aggregate events in real-time (i.e. statistical aggregations): **Aggregations**
- Convert events to metrics format: **Publish Metrics**

Commonly Used Functions

FUNCTION Examples

	DESCRIPTION	FIELD NAME	VALUE EXPRESSION
EVAL	Add a field called source and set it to **mySource**	`source`	`'mySource'`
	Change status to success if code is **200**	`status`	`code==200 ? 'success' : status`
	Set location field to **city, state**	`location`	`` `${city}, ${state}` ``
	Replace **.com** with **.net** in **url**	`url`	`url.replace(/\.com/, '.net')`
	Set **private** to **yes** if **myip** is on private range	`private`	`C.Net.isPrivate('10.10.2.0') ? 'yes' : 'no'`
	Create an array field called **myField**	`myField`	`['aa', 'bb', 'cc','dd']`
	Convert value of **classnam**e to lowercase	`classname`	`classname.toLowerCase()`

Commonly Used Functions (cont.)

FUNCTION Examples

	DESCRIPTION	MATCH REGEX	REPLACE EXPRESSION
MASK	Remove phone numbers from events	`phonenumber=\d+`	`''`
	Remove comments (lines that start with #)	`^#.*$` `/m`	`''`
	R eplace **sensitive** with **REDACTED**	`sensitive`	`'REDACTED'`
	Remove **description** from wineventlogs	`This event is generated[\s\S]+$`	`'DESCRIPTION REMOVED'`
	Redact last octet of an IPv4 address	`(\d+\.\d+\.\d+\.)\d+`	`` `${g1}xxx` ``
	MD5 hash a credit card number	`(creditcard=)(\d+)`	`` `${g1}${C.Mask.md5(g2)}` ``

FUNCTION Examples

SAMPLE EVENT: `2020-04-20 16:20:00 input=tcpjsonin rate=42 host=555.example.com region=us-east state=nj city=edgewater`

REGEX EXTRACT	Regex only **input** and rate **fields**	`'input=(?<input>\w+)\s+rate=(?<rate>\d+)`
	Regex Extract all KV pairs in the event	`(?<_NAME_0>[^\s]+)=(?<_VALUE_0>[^\s]+)`

Cribl Stream ships with a Regex Library that contains a set of pre-built common regex patterns.
Example: IPv4 Address - (?<!\d)(?:(?:[01]?\d\d?|2[0-4]\d|25[0-5])\.){3}(?:[01]?\d\d?|2[0-4]\d|25[0-5])(?!\d)

FUNCTION Examples

	SAMPLE EVENT:	FILTER EXPRESSION
DROP	Drop all **DEBUG** events	`_raw.includes('DEBUG')`
	Drop all DEBUG events from host **myhost**	`_raw.includes('DEBUG') && host=='myHost'`
	Drop 50% of all events (poor man's sampler)	`Math.random() > 0.5`
	Drop all events with length less than 42 bytes	`_raw.length < 42`