CRIBL NOTES on

U.S. $FREE

LOGSTREAM CHEAT SHEET

LogStream Cheat Sheet

Cribl

Basic Concepts

Sources

Cribl LogStream can process data from various metric, logs, 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.

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.

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.

Functions

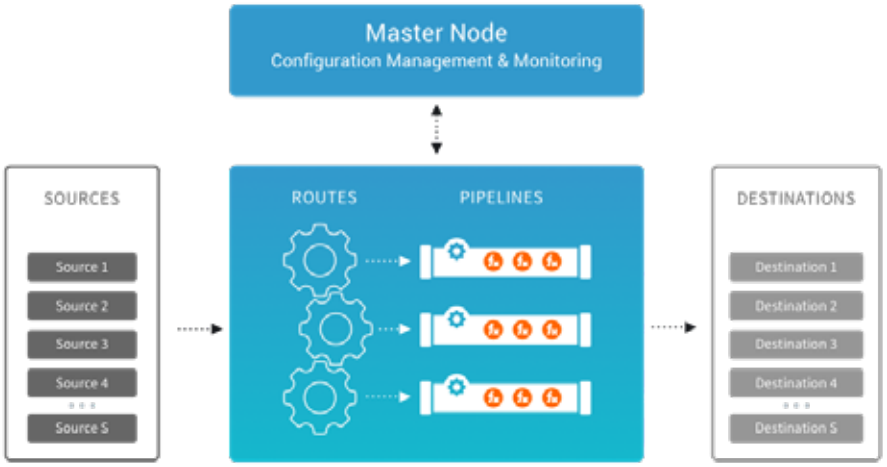
A Function is a piece of **JavaScript** code that executes on an event, and it encapsulates the smallest amount of processing that can happen to that event. E.g., a Function can replace the term `foo` with `bar` on each event. Another one can hash `bar`, and yet another can add a field, say, `dc=jfk-42`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

Destinations

Cribl LogStream can send data to various Destinations, including 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).

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 LogStream can be scaled up and out across multiple instances. This is known as a **distributed deployment**.



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)**

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 LogStream 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 field1>",
 "__other2": "<Internal field2>",
 "__otherN": "<Internal fieldN>",
 "key1": "<value1>",
 "key2": "<value2>",
 "key3": "<value3>",
 "keyN": "<valueN>",
 "...": "..."
}
```

Fields that with start with a double underscore are internal to LogStream. 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 JSON-parsed, all of its content will be assigned to the `_raw` field. If a timestamp is not configured, or cannot be extracted from an event, LogStream will assign the current time (in UNIX epoch format) to `_time`.

Filters and Value Expressions

You can use JavaScript filters and other value expressions to configure LogStream's Routes and built-in Functions. Expressions are syntactically valid units of code that resolve to a **value**. Conceptually, LogStream 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
```

LogStream Cheat Sheet

Cribl

Using Built-in Functions

LogStream 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, Prometheus Publisher**

Resolve hostname from IP address: **Reverse DNS**

Commonly Used Functions

Function | **Examples**

Eval

DESCRIPTION	FIELD NAME	VALUE EXPRESSION
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 **classname** to lowercase	`classname`	`classname.toLowerCase()`

Function | **Examples**

Mask

DESCRIPTION	MATCH REGEX	REPLACE EXPRESSION
Remove phone numbers from events	`phonenumber=\d+`	`''`
Remove comments (lines that start with **#**)	`^#.*$ /m`	`''`
Replace **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)}` ``

Commonly Used Functions (cont.)

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

Function	Examples	
	DESCRIPTION	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`

Useful JS methods:

String	`.startsWith()`, `.endsWith()`, `.trim()`, `.trimEnd()`, `.trimStart()`, `.substring()`, `.split()`, `.indexOf()`, `.length`, etc.
Number	`.isInteger()`, `.toFixed()`, `.parseFloat()`, `.toString()`, etc.
Math	`Math.E`, `Math.LN10`, `Math.abs()`, `Math.sin()`, `Math.log()`, `Math.max()`, `Math.pow()`, `Math.sqrt()`, etc.

Cribl Expressions:

Decode	`C.decode.base64()`, `C.decode.gzip()`, `C.decode.hex()`, `C.decode.uri()`
Encode	`C.encode.base64()`, `C.encode.gzip()`, `C.encode.hex()`, `C.encode.uri()`
Inline Lookup	`C.Lookup()`, `C.LookupCIDR()`, `C.LookupRegex()`
Mask	`C.Mask.isCC()`, `C.Mask.luhn()`, `C.Mask.md5()`, `C.Mask.sha1()`, `C.Mask.random()`, etc.
Net	`C.Net.cidrMatch()`, `C.Net.ipv6Normalize()`, `C.Net.isPrivate()`
Text	`C.Text.entropy()`, `C.Text.hashCode()`, `C.Text.isASCII()`, `C.Text.isUTF8()`, `C.Text.relativeEntropy()`
Time	`C.Time.strftime()`, `C.Time.strptime()`, `C.Time.timestampFinder()`
Others	`C.vars`, `C.env`, `C.Misc.xxx()`, `C.version` etc.