Collect Logs from Heroku

You can upload data from your Heroku app directly to Sumo Logic by pointing a Heroku log drain to the URL for an HTTP Source. The log drain forwards batches of data directly to the HTTP Source.

First perform the following prerequisites:

- Create an app in Heroku that generates logs.
- Install the Heroku Command Line Interface (CLI), formerly called the Heroku Toolbelt.
- Configure a Sumo Logic Hosted Collector with an HTTP Source.

Upload app data from Heroku

From the Heroku command line, go to the Heroku directory where your application is checked out, and type the following:

```
heroku drains:add https://collectors.sumologic.com/rec...ourceURLtoken]
```

If the set up is successful, you'll see this message:

```
Successfully added drain https://collectors.sumologic.com/rec...ourceURLtoken]
```

Log data will now be forwarded by Heroku as it is generated, per the behavior of Heroku’s LogPlex system.

(Optional) Set up FER for application name

This step is optional but recommended, as it makes it easier for you to query your Heroku application logs in Sumo.

When Sumo ingests Heroku application logs, it attaches the _sourceName metadata field to the data. The _sourceName Sumo assigns varies by application—it’s value is the unique identifier for the Logplex drain assigned to the application.

For ease of understanding the log data, you can use a field extraction rule (FER) to rename _sourceName from the drain UUID to the application name. For general information about FERs see Create a Field Extraction Rule.

You can determine the drain identifier by running the heroku drains command for your app. The identifier will look something like:
Then, define an FER in Sumo.

1. Go to Manage Data > Logs > Field Extraction Rules.
2. Rule Name. Enter a name for the FER.
3. Scope. Specify the _sourceCategory defined for the source to which you will be forwarding Heroku application logs. For example, if the _sourceCategory is "heroku", enter:
   _sourceCategory=heroku
4. Parse Expression. For each Heroku application reporting data to Sumo, enter a statement that renames the _sourceName from the drain ID to the application name. For example:

   if (_sourceName="Drain_ID", "Application_Name", _sourceName) as _sourceName

   The FER below changes the value of _sourceName for two applications. The first line changes _sourceName from d.98ee476d-d2d8-46bf-afc2-740f6f7e5b2a to CustApp. The second line changes _sourceName from d.00870f28-53f9-4680-b2ab-2287ec9d8637 to VendorApp:

   if (_sourceName="d.98ee476d-d2d8-46bf-afc2-740f6f7e5b2a", "CustApp", _sourceName) as _sourceName
   | if (_sourceName="d.00870f28-53f9-4680-b2ab-2287ec9d8637", "VendorApp", _sourceName) as _sourceName

---

**Known issues with Heroku Logplex**

HTTP based integration with Heroku Logplex works well for single-line, non-ordered log messages.

The Sumo Logic HTTP Source provides full access to multi-line message capabilities. However, the manner in which Logplex treats multi-line messages before they are sent to Sumo Logic significantly limits the effectiveness of our multi-line capabilities.

The most commonly affected use case is when stack traces are sent through Logplex. Stack traces span multiple lines, by contrast with single-line custom application logs. The messages are handled as follows:

- When a multi-line log message is sent to the Logplex system, each line is treated as a separate message.
- Each line is prefixed with a Heroku specific string. The string contains an integer indicating how many bytes are in the log line, followed by a syslog-style prefix that contains the timestamp, priority, and the Heroku component that generated the log line.
- Logplex does not guarantee that the ordering of messages remains intact, because the queue is unordered. When a multi-line stack trace enters Logplex, it is not guaranteed to come out in the same sequence.

Given this behavior, the following issues can arise.

- Log lines from multiple systems might get interleaved. Even if the deconstructed lines of a stack trace happen to stay in the correct order, it is possible that log lines from another component, or instance, could get mixed in with the log lines of the stack trace. Sumo customer experience suggests that this type of interleaving is infrequent, but can occur.
• When using the Logplex HTTP log drain, log lines are bundled up into a series of HTTPS POST bodies. It is possible that the multiple lines of a stack trace could get split across POST bodies, and the HTTP POST actions could get load balanced across different Sumo Logic API hosts.

Either of the following options can help address these ordering issues for multi-line messages.

• You can create multi-line RegEx boundaries that work around the prefixing of each line. However, when interleaving, reordering, or load balancing occurs, it is possible for the multi-line RegEx to fail.

• You can use an application logging library combined with a Sumo Logic appender, whereby bypassing the Logplex infrastructure. An example of such a library is Log4J with the Sumo Logic Log4J appender. When using this approach, the log lines are sent directly to Sumo Logic, retaining their original formatting. See the Sumo GitHub page for a list of Sumo Logic sponsored appenders.