Logging

Last updated 20 February 2019

Table of Contents

A Heroku app’s logs are aggregated from the output streams of all of its running processes, system components, and backing services. Heroku’s Logplex (https://devcenter.heroku.com/articles/logplex) routes log streams from all of these diverse sources into a single channel, providing a foundation for comprehensive logging.

Types of logs

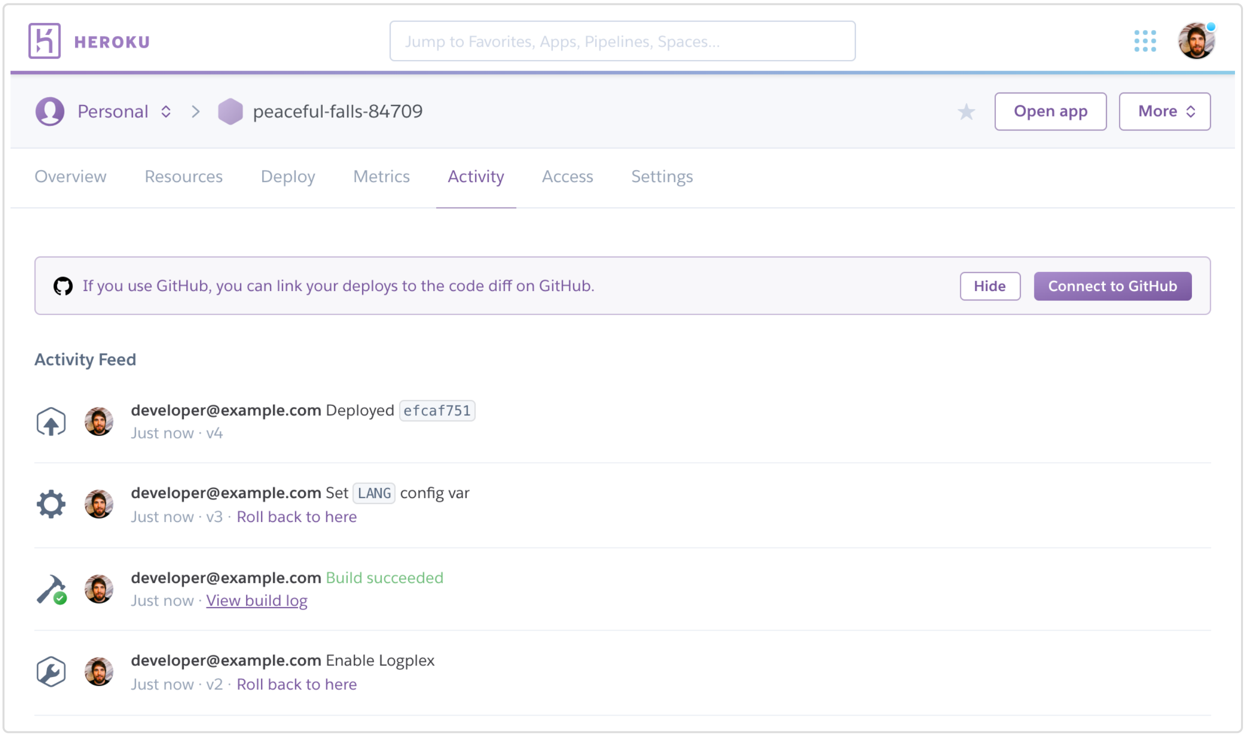
Runtime logs

Heroku aggregates the following categories of logs for a deployed app:

- **App logs** - Logging output from the application itself. This includes logs generated by your app’s code and dependencies. (Filter: `--source app`)
- **System logs** - Messages about actions taken by the Heroku platform infrastructure on behalf of your app, such as: restarting a crashed process, sleeping or waking a web dyno, or serving an error page due to a problem in your app. (Filter: `--source heroku`)
- **API logs** - Messages about administrative actions taken by you and other developers working on your app, such as: deploying new code, scaling the process formation, or toggling maintenance mode. (Filter: `--source app --dyno api`)
- **Add-on logs** - Messages from add-on services. See the add-on’s Dev Center article for details. (Filter varies by add-on)

Build logs

The logs that are generated while building and deploying your app are separate from the app's runtime logs. Logs for both successful and unsuccessful builds are available from your app's Activity tab in the Heroku Dashboard (https://dashboard.heroku.com):



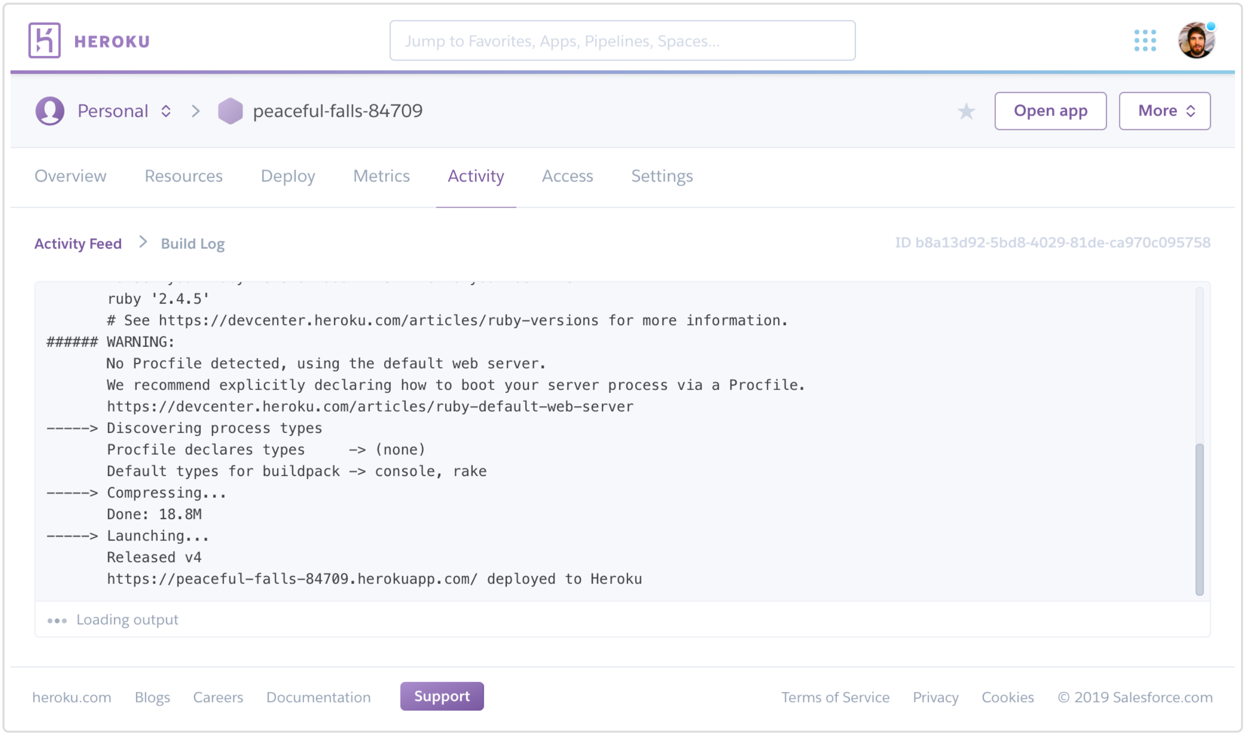
developer@example.com Deployed efcaf751
Just now · v4

developer@example.com Set LANG config var
Just now · v3 · Roll back to here

developer@example.com Build succeeded
Just now · View build log

developer@example.com Enable Logplex
Just now · v2 · Roll back to here

Click **View build log** for any build event in the Activity Feed to see its logs:



```
       ruby '2.4.5'
       # See https://devcenter.heroku.com/articles/ruby-versions for more information.
###### WARNING:
       No Procfile detected, using the default web server.
       We recommend explicitly declaring how to boot your server process via a Procfile.
       https://devcenter.heroku.com/articles/ruby-default-web-server
-----> Discovering process types
       Procfile declares types     -> (none)
       Default types for buildpack -> console, rake
-----> Compressing...
       Done: 18.8M
-----> Launching...
       Released v4
       https://peaceful-falls-84709.herokuapp.com/ deployed to Heroku
```

Log history limits

Logplex is designed for collating and routing log messages, not for storage. It retains the most recent 1,500 lines of your consolidated logs, which expire after 1 week.

For more production-ready persistence of logs, you can add one of the Heroku platform's available logging add-ons (https://elements.heroku.com/addons) to your app. Most of these add-ons offer a free plan to get started.

Alternatively, you can implement your own log drains (https://devcenter.heroku.com/articles/log-drains) for full control over what happens to your logs.

Writing to your log

Anything your app writes to standard out (`stdout`) or standard error (`stderr`) is captured into your logs. This means that you can log from anywhere in your application code with a simple output statement.

In Ruby, you could use something like:

```
puts "Hello, logs!"
```

In Java:

```
System.err.println("Hello, logs!");
System.out.println("Hello, logs!");
```

The same holds true for all other languages supported by Heroku.

To take advantage of real-time logging, you might need to disable any log buffering your application is performing. For example, in Ruby, add this to your `config.ru` file:

```
$stdout.sync = true
```

Some frameworks send log output somewhere other than `stdout` by default. These might require extra configuration. For example, when using the Ruby on Rails TaggedLogger by ActiveSupport, you should add the following into your app's configuration to get `stdout` logging:

```
config.logger = Logger.new(STDOUT)
```

Log retrieval

Log format

The output format of the `heroku logs` command is as follows:

```
timestamp source[dyno]: message
```

- **Timestamp** - The date and time recorded at the time the log line was produced by the dyno or component. The timestamp is in the format specified by RFC5424 (https://tools.ietf.org/html/rfc5424#section-6.2.3), and includes microsecond precision.

- **Source** - All of your app's dynos (web dynos, background workers, cron) have the source, `app`. All of Heroku's system components (HTTP router, dyno manager) have the source, `heroku`.
- **Dyno** - The name of the dyno or component that wrote the log line. For example, worker #3 appears as `worker.3`, and the Heroku HTTP router appears as `router`.
- **Message** - The content of the log line. Lines generated by dynos that exceed 10000 bytes are split into 10000 byte chunks without extra trailing newlines. Each chunk is submitted as a separate log line.

View logs

To fetch your app's most recent logs, use the `heroku logs` command:

```
$ heroku logs
2010-09-16T15:13:46.677020+00:00 app[web.1]: Processing PostController#list (for 208.39.138.12
2010-09-16T15:13:46.677023+00:00 app[web.1]: Rendering template within layouts/application
2010-09-16T15:13:46.677902+00:00 app[web.1]: Rendering post/list
2010-09-16T15:13:46.678990+00:00 app[web.1]: Rendered includes/_header (0.1ms)
2010-09-16T15:13:46.698234+00:00 app[web.1]: Completed in 74ms (View: 31, DB: 40) | 200 OK [htt
2010-09-16T15:13:46.723498+00:00 heroku[router]: at=info method=GET path="/posts" host=myapp.he
2010-09-16T15:13:47.893472+00:00 app[worker.1]: 2 jobs processed at 16.6761 j/s, 0 failed ...
```

In this example, the output includes log lines from one of the app's web dynos, the Heroku HTTP router, and one of the app's workers.

The `logs` command retrieves 100 log lines by default. You can specify the number of log lines to retrieve (up to a maximum of 1,500 lines) by using the `--num` (or `-n`) option.

```
$ heroku logs -n 200
```

Real-time tail

Similar to `tail -f`, real-time tail displays recent logs and leaves the session open for real-time logs to stream in. By viewing a live stream of logs from your app, you can gain insight into the behavior of your live application and debug current problems.

You can tail your logs using `--tail` (or `-t`).

```
$ heroku logs --tail
```

When you are done, press Ctrl+C to return to the prompt.

A real-time tail session is automatically terminated after one hour of inactivity.

Filtering

If you only want to fetch logs with a certain source, a certain dyno, or both, you can use the `--source` (or `-s`) and `--dyno` (or `-d`) filtering arguments:

```
$ heroku logs --dyno router
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path="/stylesheets/dev-cent
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path="/articles/bundler" ho

$ heroku logs --source app
2012-02-07T09:45:47.123456+00:00 app[web.1]: Rendered shared/_search.html.erb (1.0ms)
2012-02-07T09:45:47.123456+00:00 app[web.1]: Completed 200 OK in 83ms (Views: 48.7ms | ActiveRe
2012-02-07T09:45:47.123456+00:00 app[worker.1]: [Worker(host:465cf64e-61c8-46d3-b480-362bfd4ecf
2012-02-07T09:46:01.123456+00:00 app[web.6]: Started GET "/articles/buildpacks" for 4.1.81.209

$ heroku logs --source app --dyno worker
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ecf
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ecf
```

When filtering by dyno, either the base name (e.g., `--dyno web`) or the full name (e.g., `--dyno web.1`) can be used.

You can also combine the filtering switches with `--tail` to get a real-time stream of filtered output.

```
$ heroku logs --source app --tail
```

Log message ordering

When you retrieve logs, you might notice that they aren't always in exact chronological order, especially when multiple components are involved. Logs originate from many sources (router nodes, dynos, etc.) and are assembled into a single log stream by Logplex. Logplex itself uses a distributed architecture to ensure high availability, meaning that log messages might be collected across multiple Logplex nodes and therefore be delivered out of order.