Logs with Flask and Gunicorn

Pedro Henrique · Follow
5 min read · Aug 14, 2020

For any code running in production logs are vital. They are our way to know what is happening with the software while it is running. Web development is no exception from his rule, we need a way to keep a finger on the application's pulse.

When first starting on Python people rely heavily on the `print()` function, and it is very helpful on CLI apps, but not so much with Flask, a lightweight extensible and powerful framework for web development on Python.

Flask is great but it requires us to use Python's native logging functionalities and it can get messy when we are also using a WSGI (web server gateway interface) HTTP server. A common WSGI to be used with Flask is Gunicorn, accordingly to it's official website:

> Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.
>
> — gunicorn.org

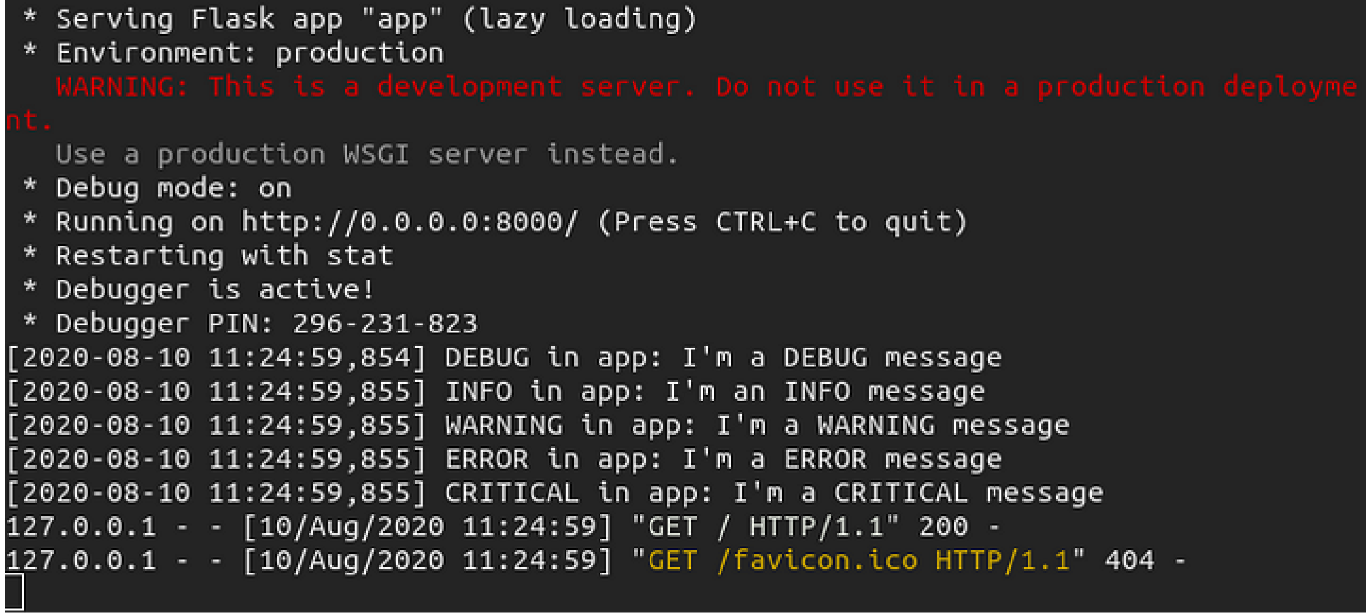
Simple Flask logs

Let's begin looking at Flask alone, below there is a small Flask application that we will be using as example on this post, you can save it to a file called app.py:

```
# app.py
import logging
```

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def default_route():
    """Default route"""
    app.logger.debug("I'm a DEBUG message")
    app.logger.info("I'm an INFO message")
    app.logger.warning("I'm a WARNING message")
    app.logger.error("I'm a ERROR message")
    app.logger.critical("I'm a CRITICAL message")

    return jsonify('a return message')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000, debug=True)
```

To run this application you just have to call `python app.py` on a terminal and it should start serving on port 8000 (I am assuming here that Flask is already installed on your machine, if not here is how). The output from calling localhost:8000 should look like this:



```
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 296-231-823
[2020-08-10 11:24:59,854] DEBUG in app: I'm a DEBUG message
[2020-08-10 11:24:59,855] INFO in app: I'm an INFO message
[2020-08-10 11:24:59,855] WARNING in app: I'm a WARNING message
[2020-08-10 11:24:59,855] ERROR in app: I'm a ERROR message
[2020-08-10 11:24:59,855] CRITICAL in app: I'm a CRITICAL message
127.0.0.1 - - [10/Aug/2020 11:24:59] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [10/Aug/2020 11:24:59] "GET /favicon.ico HTTP/1.1" 404 -
```

Flask output on terminal

As you can see there is a line in red telling us that we should not use this in production, and that is why we use Gunicorn. Following it there are all the logs that we put on our default_route and finally a message telling us it returned with status 200 (success!). Because I was running it from the browser, it tried to fetch a favicon but was not successful, hence the 404 (not found).

Adding Gunicorn

Flask's built-in WSGI is enough for development, but the framework itself says you shouldn't be using it in production. For heavier loads and a more dynamic environment we should be using a specialized WSGI, Gunicorn is a good option. It is lightweight, can be easily configured and is very reliable.

Particularly when it comes to logs, the problem we have with Gunicorn is that **it has it's own log handlers**. That means that logs are setup independently between Flask and Gunicorn. We need to make sure logs coming from Flask and Gunicorn are wired together in order for us to have a nice logging experience. Otherwise we could end up with different log levels and even different output location.

> In fact, the idea for this post came from something like this happening this week on a project I am working on. A colleague created a bug on Jira about the API logs not being displayed on our logging output tool, Graylog, and I got responsible for this task.
>
> While studying the problem I noticed that locally the only logs being outputted on my terminal were the ones from Flask or the critical ones from Gunicorn. Then I found this post about it by Thomas Stringer, and them it became clear, it was just a mater of wiring both outputs together.

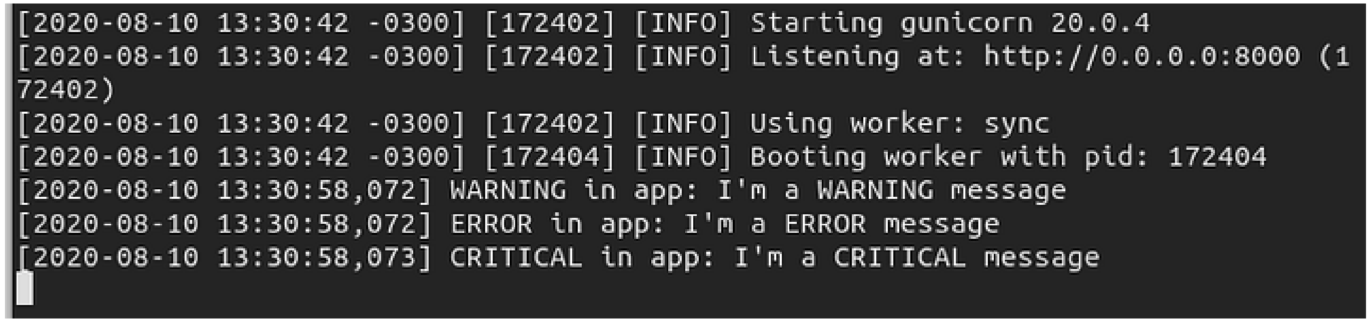
Continuing with the solution, we must configure Flask to make it output it’s logs using the Gunicorn’s handlers. We do that by adding these two lines of code to our app.py file:

```
gunicorn_logger = logging.getLogger('gunicorn.error')
app.logger.handlers = gunicorn_logger.handlers
```

Now we can stop our application if it is still running and start it agian with Gunicorn:

```
gunicorn --bind=0.0.0.0:8000 app:app
```

And the output after a call to localhost:8000 will be similar to this:



```
[2020-08-10 13:30:42 -0300] [172402] [INFO] Starting gunicorn 20.0.4
[2020-08-10 13:30:42 -0300] [172402] [INFO] Listening at: http://0.0.0.0:8000 (1
72402)
[2020-08-10 13:30:42 -0300] [172402] [INFO] Using worker: sync
[2020-08-10 13:30:42 -0300] [172404] [INFO] Booting worker with pid: 172404
[2020-08-10 13:30:58,072] WARNING in app: I'm a WARNING message
[2020-08-10 13:30:58,072] ERROR in app: I'm a ERROR message
[2020-08-10 13:30:58,073] CRITICAL in app: I'm a CRITICAL message
```

Gunicorn output on terminal

There is the problem I told you about, the only logs on the output are the warning, error and critical. What happened to all the others?

The main reason for this is that Gunicorn has it's own logger and it uses different log levels than Flask. The default log levels (the ones we are seeing right now) are warning and above.

A quick and dirty fix is to manually set the log levels for Gunicorn to be the same as the ones for Flask, just adding `app.logger.setLevel(logging.DEBUG)`. The problem with this approach is that the level is hardcoded on the application and even if we change that we would still have two log levels, one for Flask and one for Gunicorn. As good clean coders we would never allow such a thing to happen, right!?

A better approach

The following snippet allows us to use Gunicorn's log level as Flask's log level if we are running the application with Gunicorn.

```
if __name__ != '__main__':
    gunicorn_logger = logging.getLogger('gunicorn.error')

    app.logger.handlers = gunicorn_logger.handlers
    app.logger.setLevel(gunicorn_logger.level)
```

What it does is, first check if we are running the code directly. If `__name__` is equal to __main__, we have called it directly with `python app.py` if `__name__` is different from main, we are running it from Gunicorn and we should use it's log handlers.

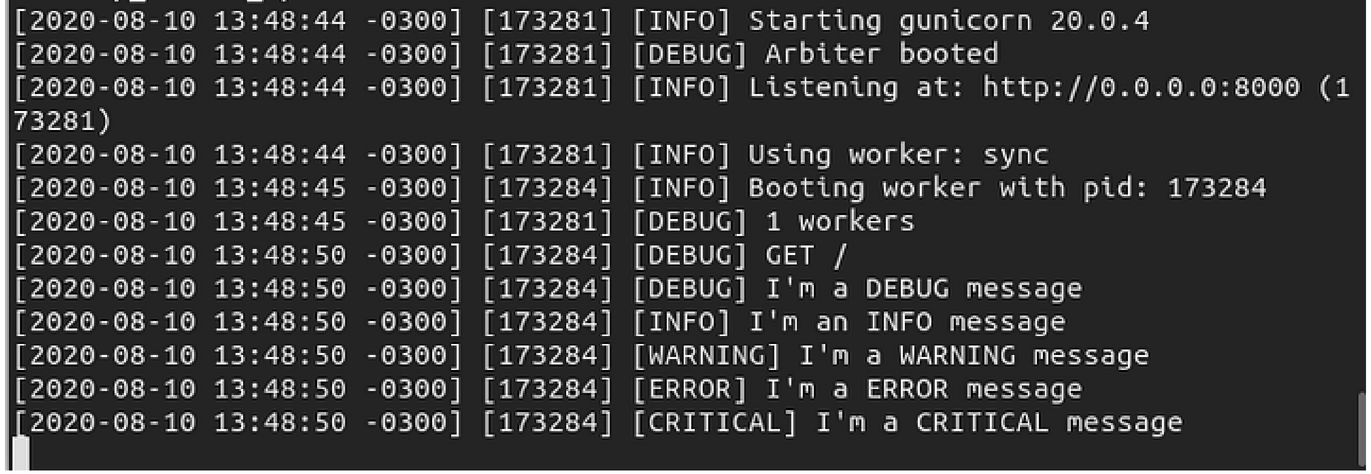
Line number 2 creates the gunicorn_logger that we will be using, on line 4 we add Gunicorn’s loggers as the applications loggers so Flask will use them too. Lastly but most important, we set the log levels for the app as the same ones used for Gunicorn.

Because we are setting the same log level for Flask as we have on Gunicorn we can set the log levels on the call to Gunicorn with the `--log-level` flag. When we do that we will have a single source of truth, much better.

An example call would be:

```
gunicorn --bind=0.0.0.0:8000 --log-level=debug app:app
```

And now the output is:


```
[2020-08-10 13:48:44 -0300] [173281] [INFO] Starting gunicorn 20.0.4
[2020-08-10 13:48:44 -0300] [173281] [DEBUG] Arbiter booted
[2020-08-10 13:48:44 -0300] [173281] [INFO] Listening at: http://0.0.0.0:8000 (1
73281)
[2020-08-10 13:48:44 -0300] [173281] [INFO] Using worker: sync
[2020-08-10 13:48:45 -0300] [173284] [INFO] Booting worker with pid: 173284
[2020-08-10 13:48:45 -0300] [173281] [DEBUG] 1 workers
[2020-08-10 13:48:50 -0300] [173284] [DEBUG] GET /
[2020-08-10 13:48:50 -0300] [173284] [DEBUG] I'm a DEBUG message
[2020-08-10 13:48:50 -0300] [173284] [INFO] I'm an INFO message
[2020-08-10 13:48:50 -0300] [173284] [WARNING] I'm a WARNING message
[2020-08-10 13:48:50 -0300] [173284] [ERROR] I'm a ERROR message
[2020-08-10 13:48:50 -0300] [173284] [CRITICAL] I'm a CRITICAL message
```

Output from Gunicorn with fixed logs

There are aditional logs coming from Gunicorn, but our logs from `default_route` are also being displayed in all their glory.

The complete code for our little application:

```
# app.py
import logging from flask
import Flask, jsonify

app = Flask(__name__) # We check if we are running directly or not

if __name__ != '__main__': # if we are not running directly, we set
the loggers
    gunicorn_logger = logging.getLogger('gunicorn.error')

    app.logger.handlers = gunicorn_logger.handlers
    app.logger.setLevel(gunicorn_logger.level)

@app.route('/')
def default_route():
    """Default route"""
    app.logger.debug("I'm a DEBUG message")
    app.logger.info("I'm an INFO message")
    app.logger.warning("I'm a WARNING message")
    app.logger.error("I'm a ERROR message")
    app.logger.critical("I'm a CRITICAL message")

    return jsonify('a return message')

if __name__ == '__main__':
   app.run(host='0.0.0.0', port=8000, debug=True)
```

Thanks for reading, hope this is useful for you. If you are still having problems configuring Flask with Gunicorn, leave a comment and I will do my best to help you!

. . .

Originally published at http://pedrofullstack.com on August 14, 2020.