Corndel DevOps Engineering Programme

in association with Softwire

Module 13: Project Exercise

Corndel Digital.

DB5H

Module 13

Project Exercise Brief

In this exercise we'll add some logging to the app, and send our logs to an external service.

Part 1: Adding logs

Step 1: View existing logs

Start up your app running locally, log in, and carry out a few operations on some TODOs. Now look in the console where you ran the app - you'll notice that the app is logging some information about the HTTP requests it receives, but not much else. We're going to enhance this information by adding our own log statements.

Step 2: Add log statements

Flask uses python's standard logging library. You can read the documentation for it here and here.

The information you'll need for now is that you can access a logger object from within your app using `app.logger`, which has methods corresponding to the log levels `debug`, `info`, `warning`, `error` and `critical`.

For example, to log the value of the variable `foo` at INFO level, you would call

```
app.logger.info("Value of foo is %s", foo)
```

Go ahead and add some logging statements to your app. You should consider the following:

- What does the app do where you might need logging? Either for auditing purposes, or for potential debugging?
- What information should be contained in the logs? You'll need enough to be useful, but not so much that the logs become overburdened, or reveal sensitive information.
- What level should the log statements be? Do they warn about a problem, contain information about normal running, or are they designed for debugging issues?

Some things you might want to consider logging include:

- Operations that manipulate TODOs
- User activity, like successfully (or unsuccessfully) logging in

Step 3: Configure log level

You'll notice that by default, when running locally, Flask logs everything at DEBUG level and above. This is because it's running in the development profile. When running in production, it will only log ERROR and above.

We'd like to make this configurable. Add an environment variable to the app config called `LOG_LEVEL` and in app.py, after creating the config object, add the line

```
app.logger.setLevel(app.config['LOG_LEVEL'])
```

Experiment with setting the value of this in your .env file (eg DEBUG, or ERROR) and check that the output from the app changes accordingly.

Part 2: Sending logs to Loggly

In this part, we're going to send the logs you've just added to a log management service. In this case we're going to use Loggly.

Step 1: Sign up to Loggly

Create a free trial account at Loggly. This will allow you to use all of Loggly's features for 30 days. (After 30 days you should still be able to use most of the features we set up here - the exception being the alerts set up in one of the stretch goals)

Step 2: Obtain a token from Loggly

Once you've created your account, log in and find the icon for "Logs" in the left-hand menu. Under this, select "Source Setup". Then, on the tabs along the top of the page, select "Customer Tokens". Add a new customer token. Copy the value of the token and make a note of it, remembering that this is a secret token so should be managed in the same way as other sensitive config values.

Add a new config parameter to the app called `LOGGLY_TOKEN` and set it in your `.env` file.

Step 3: Send logs to Loggly

We're going to send our logs to loggly using HTTPS, as documented here. Other methods are available, including syslog, but this is the easiest to set up.

Start by using poetry to install the package `loggly-python-handler`.

Then add the following import lines to app.py

```
from loggly.handlers import HTTPSHandler
from logging import Formatter
```

In the create_app function, after you set the logging level, add the lines

```
if app.config['LOGGLY_TOKEN'] is not None:
    handler = HTTPSHandler(f'https://logs-01.loggly.com/inputs/
{app.config["LOGGLY_TOKEN"]}/tag/todo-app')
    handler.setFormatter(
        Formatter("[%(asctime)s] %(levelname)s in %(module)s: %
(message)s")
    )
    app.logger.addHandler(handler)
```

These lines do the following:

- Create a new handler which sends logs to loggly via the URL given (which includes your own token)
- Tells this handler to format the logs in the same way as Flask's default formatting (so they appear the same in Loggly as they do in the console)
- Attaches the handler to the app logger

You can read more about loggers and handlers in the python documentation linked above.

Step 4: View logs in Loggly

Run the app and carry out some actions that will trigger log statements. You should still see the log statements in the console, since we've added a new handler, not replaced the existing one.

Now go to Loggly and navigate to Logs > Log Explorer. Set the time range from 1 day in the past until now, and click Search. You should see your logs.

Take a screenshot of the logs in Loggly and add it to the pull request when you submit your solution to the exercise.

Stretch goals

Send logs to loggly in JSON format

Currently you're sending the logs to Loggly as a text string, but log management tools usually come with powerful tools for analysing more structured logs. Try sending your logs to Loggly in JSON format by attaching a different formatter to your handler (the python library `python-json-logger` might come in handy). Now explore how you can search and filter these logs in Loggly based on the JSON fields.

Error handling and alerting

When run in a production environment (i.e. with `FLASK_ENV=production`) Flask will log any unhandled errors in your app at ERROR level.

Try setting up Loggly to send you an email alert when your app sends a log at ERROR level. The documentation here might help.

The easiest way to test this might be to add some temporary code to your app that throws an exception, and to run it locally using the production docker image.

HTTP logs to Loggly

You'll notice that in Loggly you can only see the log lines you've added, not the HTTP request logs that are being sent to the console. This is because these are logged to a different logger by an library called 'werkzeug', which is used by Flask.

We can fix this by adding a similar handler to this logger.

Add this import statement to app.py

```
from logging import getLogger
```

and, in the block where you add the Loggly handler, add this line

```
getLogger('werkzeug').addHandler(HTTPSHandler(f'https://
logs-01.loggly.com/inputs/{app.config["LOGGLY_TOKEN"]}/tag/todo-
app-requests'))
```

Try making some requests, and you should see additional logs showing up in Loggly. Note that Loggly has parsed the raw text message into a JSON structure, which allows easier searching and classification.

Note also that these logs have a different 'tag' in Loggly. This is set at the end of the URL we give the handler, and allows us to easily split out the request logs and our custom application logs.