

Load Testing & Bottlenecks

Software Architecture

April 19, 2023

Brae Webb



1 This Week

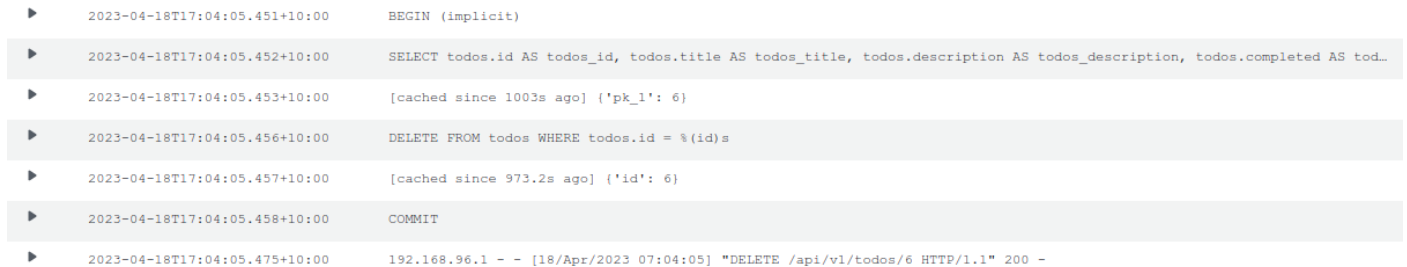
Our goal is to:

- Introduce structured logging to TaskOverflow.
- Deploy TaskOverflow to AWS.
- Write k6 tests to ensure TaskOverflow can handle given scenarios.
- Uncover and fix bottlenecks in the TaskOverflow application using the logs.

2 Watchtower

In this course we have repeatedly claimed that there is value in monitoring and logging. This week, we aim to prove it by using logging to help diagnose issues with a deployment of TaskOverflow. For this task, we have integrated [Watchtower](https://pypi.org/project/watchtower/)¹ into the project. Watchtower is a Python library that allows us to send logs to AWS CloudWatch, allowing us to monitor the applications performance in real time.

Currently the project is configured to log API accesses made with Flask and to log database queries created by SQL Alchemy. This results in an unstructured log stream as seen in Figure 1.



```
▶ 2023-04-18T17:04:05.451+10:00 BEGIN (implicit)
▶ 2023-04-18T17:04:05.452+10:00 SELECT todos.id AS todos_id, todos.title AS todos_title, todos.description AS todos_description, todos.completed AS tod...
▶ 2023-04-18T17:04:05.453+10:00 [cached since 1003s ago] {'pk_1': 6}
▶ 2023-04-18T17:04:05.456+10:00 DELETE FROM todos WHERE todos.id = %(id)s
▶ 2023-04-18T17:04:05.457+10:00 [cached since 973.2s ago] {'id': 6}
▶ 2023-04-18T17:04:05.458+10:00 COMMIT
▶ 2023-04-18T17:04:05.475+10:00 192.168.96.1 - - [18/Apr/2023 07:04:05] "DELETE /api/v1/todos/6 HTTP/1.1" 200 -
```

Figure 1: An example of logs made to AWS CloudWatch for a DELETE request in the TaskOverflow API.

Getting Started

1. Using the GitHub Classroom link for this practical provided by your tutor in Slack, create a repository to work within.
2. Install Terraform if not already installed, as it will be required again this week.
3. Start your learner lab and copy the AWS Learner Lab credentials into a credentials file in the root of the repository.

What's New We are returning to TaskOverflow roughly from the state at the end of the last practical. The following notable changes have been made:

- Watchtower has been installed as a dependency.
- In `docker-compose.yml`, we mount the credentials file to `/root/.aws/credentials`. This allows local testing of watchtower to log to AWS CloudWatch.
- Logging has been introduced for Flask and SQL Alchemy.

Our first task will be to replicate the above logs in Figure 1. Once you have copied `credentials` into the project root, start docker compose with:

```
$ docker-compose up
```

¹<https://pypi.org/project/watchtower/>

```

app_1 | * Serving Flask app 'todo'
app_1 | * Debug mode: on
app_1 | INFO:werkzeug:WARNING: This is a development server. Do not use it in a
      production deployment. Use a production WSGI server instead.
app_1 | * Running on all addresses (0.0.0.0)
app_1 | * Running on http://127.0.0.1:6400
app_1 | * Running on http://192.168.96.3:6400
app_1 | INFO:werkzeug:Press CTRL+C to quit
app_1 | INFO:werkzeug: * Restarting with stat
app_1 | INFO:botocore.credentials:Found credentials in shared credentials file: ~/.
      aws/credentials
app_1 | INFO:sqlalchemy.engine.Engine:select pg_catalog.version()
app_1 | INFO:sqlalchemy.engine.Engine:[raw sql] {}
app_1 | INFO:sqlalchemy.engine.Engine:select current_schema()
app_1 | INFO:sqlalchemy.engine.Engine:[raw sql] {}
app_1 | INFO:sqlalchemy.engine.Engine:show standard_conforming_strings
app_1 | INFO:sqlalchemy.engine.Engine:[raw sql] {}
app_1 | INFO:sqlalchemy.engine.Engine:BEGIN (implicit)
app_1 | INFO:sqlalchemy.engine.Engine:SELECT pg_catalog.pg_class.relname
app_1 | FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.
      pg_namespace.oid = pg_catalog.pg_class.relnamespace
app_1 | WHERE pg_catalog.pg_class.relname = %(table_name)s AND pg_catalog.pg_class.
      relkind = ANY (ARRAY[%(param_1)s, %(param_2)s, %(param_3)s, %(param_4)s, %(
      param_5)s]) AND pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) AND
      pg_catalog.pg_namespace.nspname != %(nspname_1)s
app_1 | INFO:sqlalchemy.engine.Engine:[generated in 0.00011s] {'table_name': 'todos',
      'param_1': 'r', 'param_2': 'p', 'param_3': 'f', 'param_4': 'v', 'param_5': 'm',
      'nspname_1': 'pg_catalog'}
app_1 | INFO:sqlalchemy.engine.Engine:COMMIT
app_1 | WARNING:werkzeug: * Debugger is active!

```

You should see logs similar to the above. Notice that information about Flask is prefixed with `INFO:werkzeug` and information about SQL Alchemy is prefixed with `INFO:sqlalchemy.engine.Engine`. This prefix indicates which log stream they are put into.

Open the AWS CloudWatch console and go to Log groups on the side panel. You should see a log group called `taskoverflow`, if you click on that group you can see the two log streams.



2.1 Structured Logging

Our first task will be to convert the current logging into a structured logging format. As we saw in last weeks tutorial, structured logging can be as simple as logging a JSON object. This allows logging services to quickly filter through logs based on criteria on the objects fields.

2.2 Correlation IDs

Correlation IDs are a mechanism to help understand the path of a request, event, message, etc. through a system. When logging it can be helpful to be able to trace the execution of a particular request.