Lab 2 Deploying Webapps on Containers on AWS

Written by:
Daníel Jóhannsson - 2311176
Vanessa Nguyen - 2313156
Danilo Vaz - 2313099
Sofia Canas - 2313066

November 11, 2023

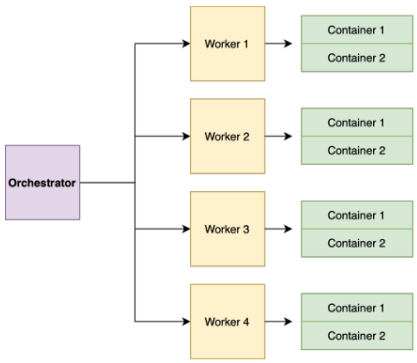


Figure 1: Shows how the orchestrator, workers and containers work together

1 Experiments with your docker containers

1.1 Receiving requests from users

The first experiment involves the orchestrator handling requests from users via port 80. We were tasked with distributing this data traffic among 4 workers using an orchestrator. Each worker has 2 similar containers that can handle requests from the user. The job of orchestrator instance is to keep track of which containers are available or not, using an internal status file. If none were to be available the orchestrator should hold the request in an internal queue until a container is available. We have been tasked with implementing this solution and should work similarly to 1.

1.2 Deploying an ML application on your workers using flask

The other part of this assignment is to deploy an ML-model on every container. The ML-model should be of simple nature and written using the flask framework. This application should be deployed on each container and be able to produce reasonable output generated by the model. Similarly to deploying the flask apps in the first assignment, here our flask application calls a function which is responsible for running inference on an ML model and returns a JSON response. By using the example code provided in the instructions, we were able to deploy the ML application using Flask.

2 How did you compose your docker containers

When we create the EC2 instances we pass along the bash script workerScript.sh in the UserData-field. In the script we create the docker-compose.yml file, specifying that we want to create two containers. One running on port 5000 and the other on 5001. We also create the dockerfile and server.py, containing the Flask application code. Lastly the docker compose command is runned, which launches the containers, making them ready for requests.

3 How the orchestrator manages the queue when all instances are busy

In our implementation we chose to have a queue with one spot for holding a request, in case all containers were to be full. Every time a new request is sent to the orchestrator a new thread is started where it either forwards the request if there is any available container. If not, this request is stored in a queue until a container is available, then it is popped from the queue and forwarded. The statuses of the containers can be seen in a json-file, which contains ip-addresses, port and status of each container.

4 How you deployed your flask application on the orchestrator

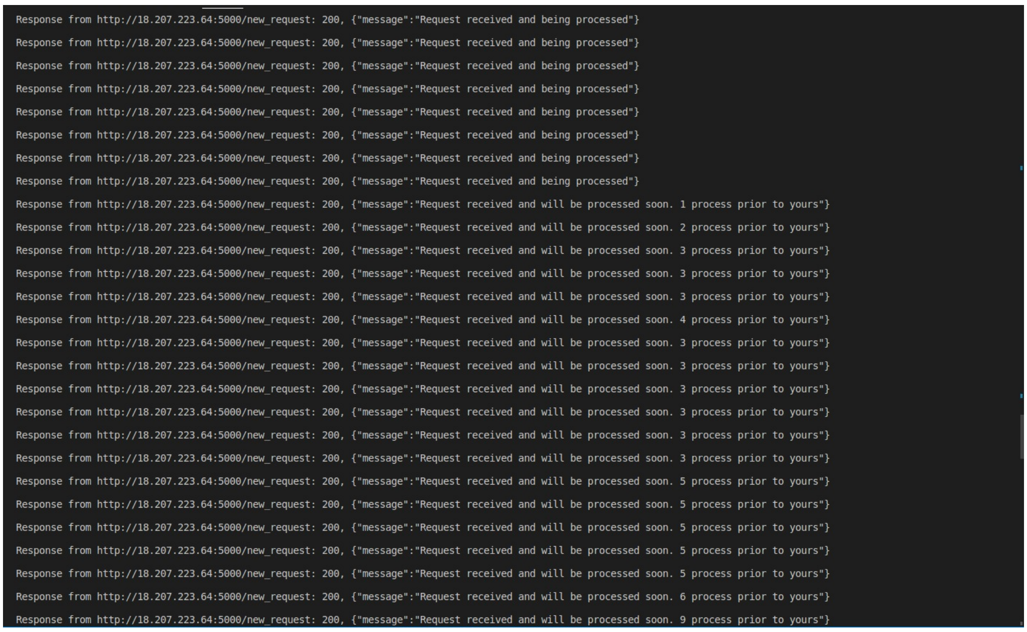
The orchestrator needs the DNS addresses for the workers and therefore we needed a script that reads the ip-addresses and creates a new local script which uses the ip-adresses for handling orchestration of data traffic. This local script is passed on as a parameter during the creation of the orchestrator instance. This way we are able to send the traffic received by the orchestrator to the right containers. Without this nested script it would not be possible to forward the requests.

5 How did you deploy your flask apps on your workers

This step is part composing the docker containers. When we create the worker instances, we load them with our workerScript which creates all the necessary docker-compose details, dockerfile and the python file containing the code for the flask app. In the dockerfile we copy the python file over to the container and also install all the dependencies needed for it. Lastly the containers are launched with docker-compose and have then succesfully deployed the flask application.

6 A sample of what your cluster returns for an incoming request

We sent 100 get requests with 5 threads to test the orchestrator. Figure 2 shows the responses for the incoming requests, when there's a free spot in the queue the responses will be processed and the message of the response yields "*Request received and being processed*". However if all containers are busy, the message will be "*Request received and will be processed soon. X process prior to yours*".



```
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and being processed"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 1 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 2 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 4 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 3 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 5 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 5 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 5 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 5 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 5 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 6 process prior to yours"}
Response from http://18.207.223.64:5000/new_request: 200, {"message":"Request received and will be processed soon. 9 process prior to yours"}
```

Figure 2: A sample of the responses from incoming requests

7 Summary of results and instructions to run your code

7.1 Instructions to run the code

- Clone repo from Github
- The following values need to be stored in ~/.aws/credentials:
  - AWS_ACCESS_KEY_ID
  - AWS_SECRET_ACCESS_KEY

  - AWS_SESSION_TOKEN

- cd to lab2-folder

Once all that is done we run the following command

```
# Runs the application
python Main.py
```

Listing 1: Python script to run the application

Runs all the functions and scripts to setup as well as launch all the worker instances, docker containers and the orchestrator instance.