POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

POLYTECHNIQUE MONTRÉAL

LOG8415E

ADVANCED CONCEPTS OF CLOUD COMPUTING

Lab 1

Cluster Benchmarking using EC2 Virtual Machines and Elastic Load Balancer (ELB)

Authors:
2009913 - Jordan Mimeault
2018968 - Antoine Lombardo
2020511 - Jacob Dupuis
2024785 - Alexandre Dufort

Lab Instructor:
Vahid Majdinasab

Instructor:
Amin Nikanjam

October 17, 2022

Contents

1 Flask Application Deployment Procedure

The deployment of our application can be done using the first option of our run.sh script. This script guides the user while configuring AWS, then run a Python script that uses Boto3 for creating Security Groups, Instances, Target Groups and Load Balancer. Boto 3 is a AWS SDK that can be used to create, configure, and manage AWS services. That script will automatically flush old instances and other configurations then configure everything correctly to make sure everything always work as expected.

The deployment of the Flask application in every instances is done with the use of a user_data script, which allows us to perform automated configuration tasks and run the Flask app instance starts. Our user_data script first installs git on each instance, then clone our git repository. It then install all the Python requirements then run the Flaks app on the instance. After that, each instance starts to respond to the port 80.

To make sure that our Flask app keeps running, we used nohup, which instructs the system to continue running it even if the session is disconnected. That way, we can be sure that the Flask app we started won't stop.

2 Cluster setup using Application Load Balancer.

We created two clusters (target groups) named cluster1 and cluster2. cluster1 targets every t2 instances, while cluster2 targets every m4 instances. These target groups allows us to redirect traffic to the desired type of instance, depending on the rules specified in the Load Balancer.

For the Load Balancer, we created an internet-facing Application Load Balancer. We then added a Listener that listens to the HTTP port 80. We specified rules to redirect traffic to the correct clusters, depending on the path that needs to be accessed. The first rule forwards the request to cluster1 if the path is /cluster1, the second rule forwards the request to cluster2 if the path is /cluster2, and the default route returns a fixed response which is a 404 NOT FOUND error, indicating that the routes leads to no specified cluster.

3 Benchmark results

We performed several measurements on the load balancer, cluster 1 (t2.large instances) and cluster 2 (m4.large instances). We have 6 diagrams representing all interesting data for the comparison of the 2 clusters. First, we calculated the number of requests made over a time interval. We did this for the load balancer as well as for the two clusters. Then, we have 3 diagrams representing the status codes, i.e. all the codes 2XX, 4XX and 5XX of the requests sent previously. Finally, the last diagram represents the average response time of each cluster for the same requests of diagram 1.

All the data was fetch using Boto3 CloudWatch API. Our script was made to automatically generate latex plots.

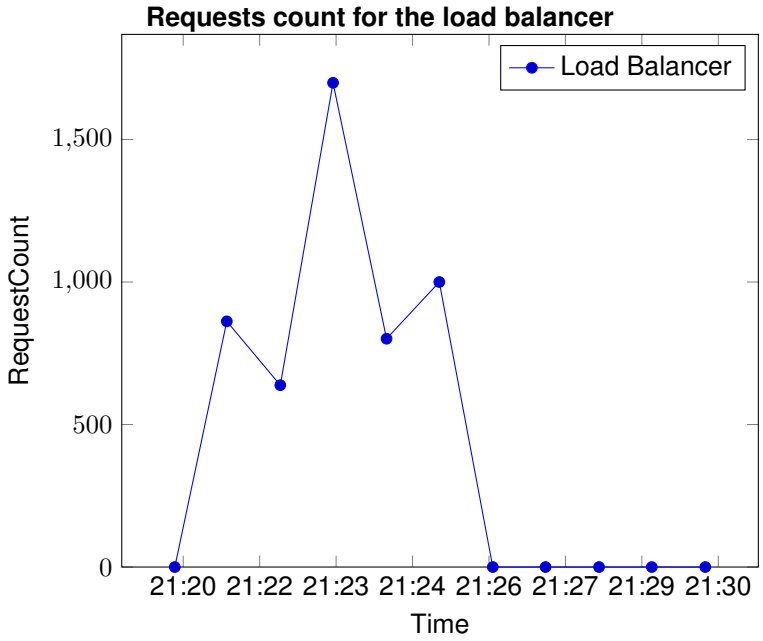

Figure 3.1 - Requests count for the load balancer

As we can see in the graph above (figure 3.1), the requests count of the load balancer totals around 5000 requests, which was expected since it includes all requests made to both clusters.

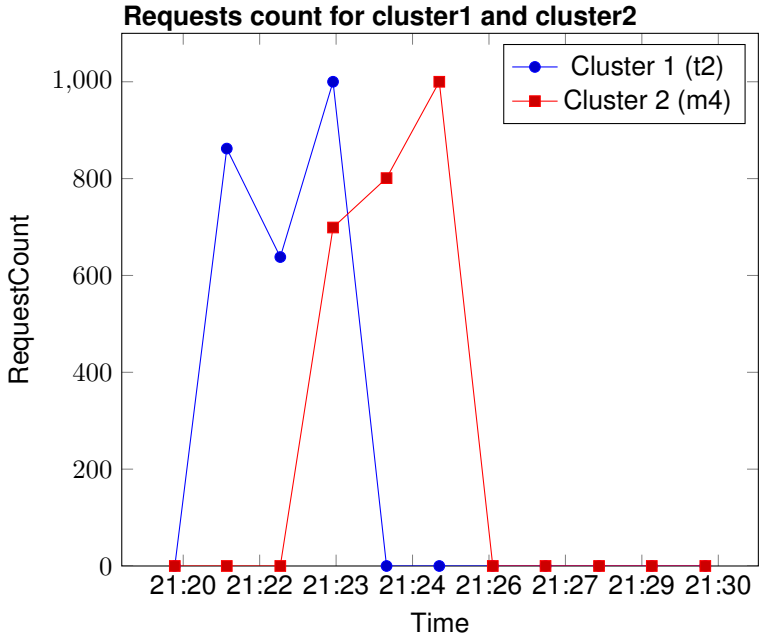

Figure 3.2 - Requests count for cluster1 and cluster2

This diagram (figure 3.2) shows that each cluster treats the requests similarly because the request count is similar over the time. Both clusters received a total of 2500 requests, which is what we expected since we sent 2500 requests to each route. This means that the load balancing is done effectively for both t2 and m4 instances.

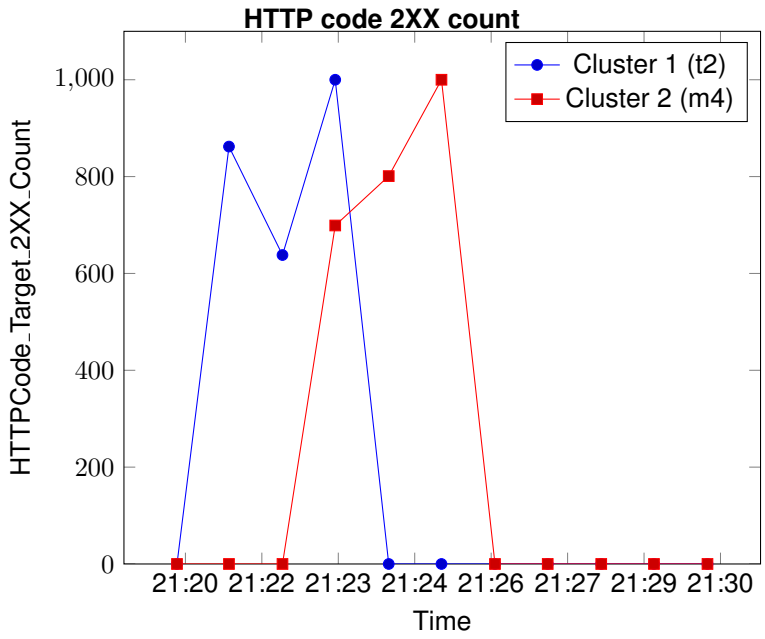

Figure 3.3 - HTTP code 2XX count

This diagram (figure 3.3) shows that there are as many HTTP status code 2XX as there are requests because the diagram is identical to the diagram of the requests count. This means that there were no errors in the load balancing. This affirmation is confirmed by analyzing the next two diagrams (figures 3.4 and 3.5).

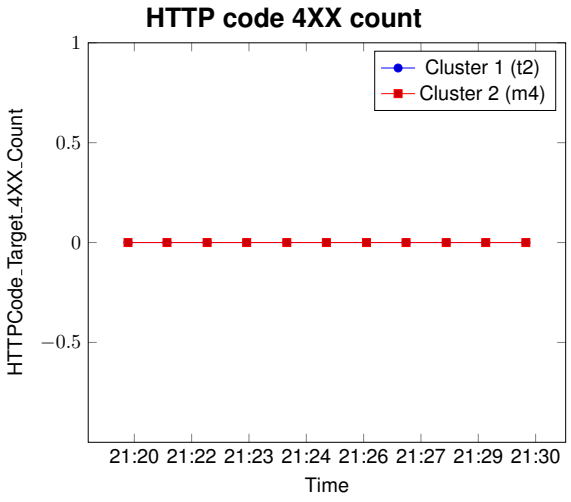

Figure 3.4 - HTTP code 4XX count

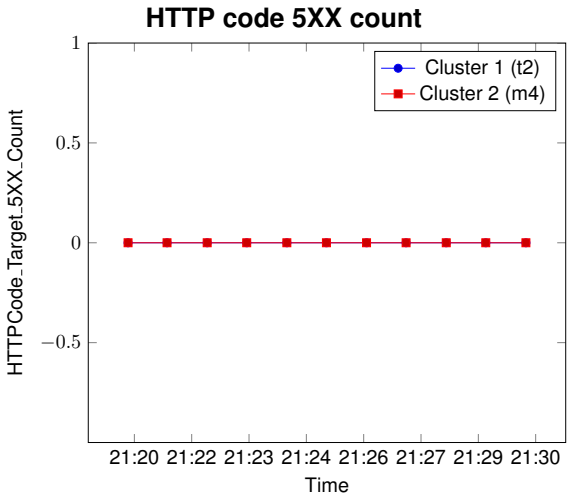

Figure 3.5 - HTTP code 5XX count

In above diagrams (figures 3.4 and 3.5), we can see that no error codes 4xx or 5xx were given. That means that no error occurred in any request, which allows us to say that no error has happen while doing our benchmarks.

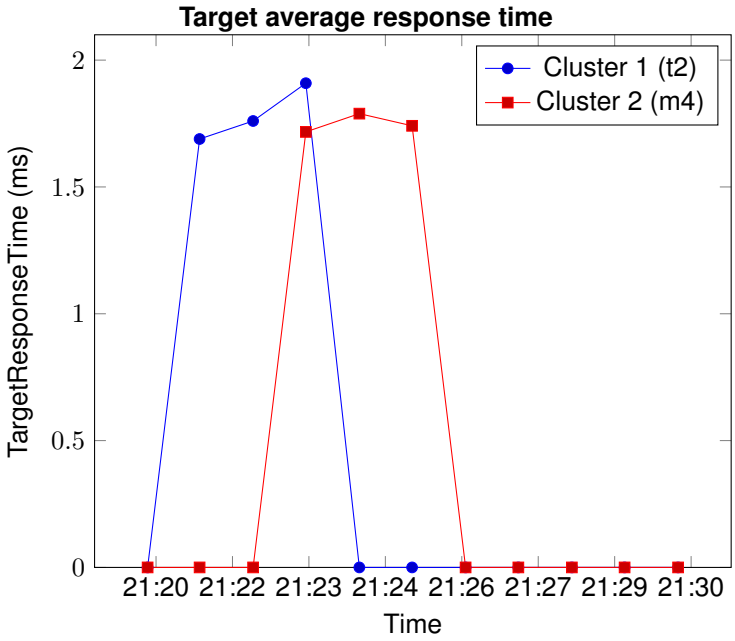

Figure 3.6 - Target average response time for each cluster

This diagram (figure 3.6) shows that the cluster 2 has a slightly better average time of response than the cluster 1. However, the difference is so small that we cannot conclude anything over this. Overall, both clusters seem to perform similarly.

4 Instructions to run the code.

The script needs to be run and should run correctly on any linux environment. However, it has only been tested on a Debian environment. The pre-requisite to run the script is to have git, pip3, python3, aws and docker installed. To make sure the script is executable, the command `sudo chmod +x script.sh` must be executed. Our bash script can then be run as root with the command `sudo .\script.sh`. Out git repository will then be automatically cloned and the `run.sh` script will be executed, which will automatically install all the necessary Python libraries.

Alternatively, the script can be run manually. To do so, the repository must be cloned using this command: `git clone https://github.com/JordMim/LOG8415E.git`. The script will then be located in the directory `LOG8415E/tp1`. The script must be set as executable using the command `sudo chmod +x run.sh`, and can then be run using the command `sudo ./run.sh`.

When running the script, you'll be asked to setup AWS credentials. If this is your first time using the script, you'll have to do this setup. After that, this step can be omitted, as it will fetch the default credentials saved in AWS.

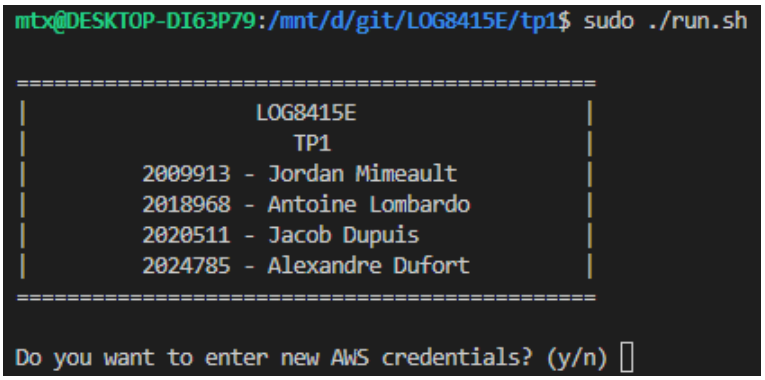

Figure 4.1 - Execution of command `sudo ./run.sh`

When AWS configuration is done, you'll be asked what actions needs to be done. There are five options:

- Configure AWS Load Balancer, which basically creates all the necessary AWS resources.
- Run requests sender, which runs the benchmark.
- Fetch metrics, which fetch CloudWatch metrics and export them into JSON and latex format.
- Only run benchmark and fetch metrics.
- Do everything.

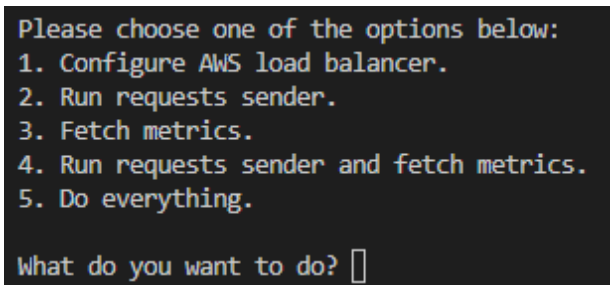

Figure 4.2 - Possible options of action

For each step of the script, the normal output should look like these:

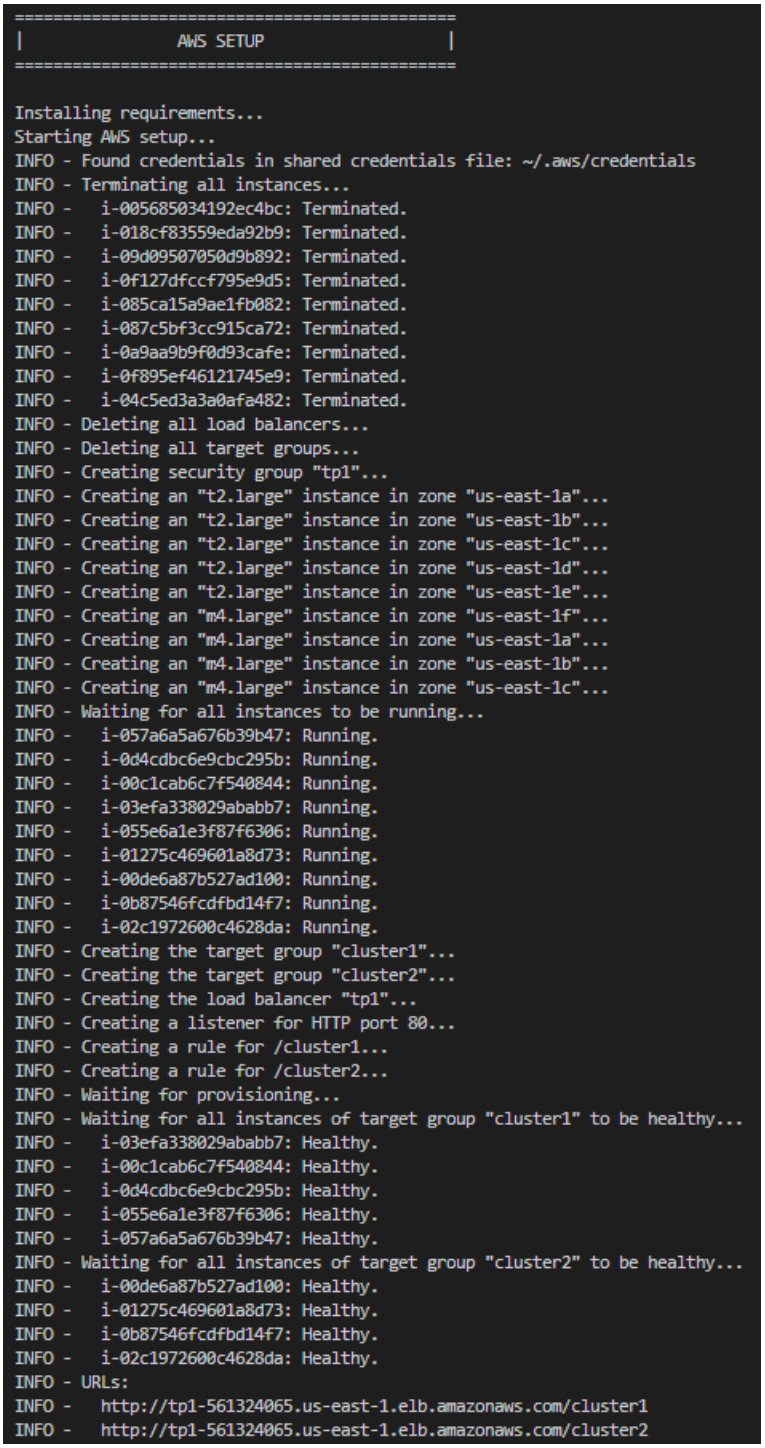

Figure 4.3 - Output of the AWS Setup step

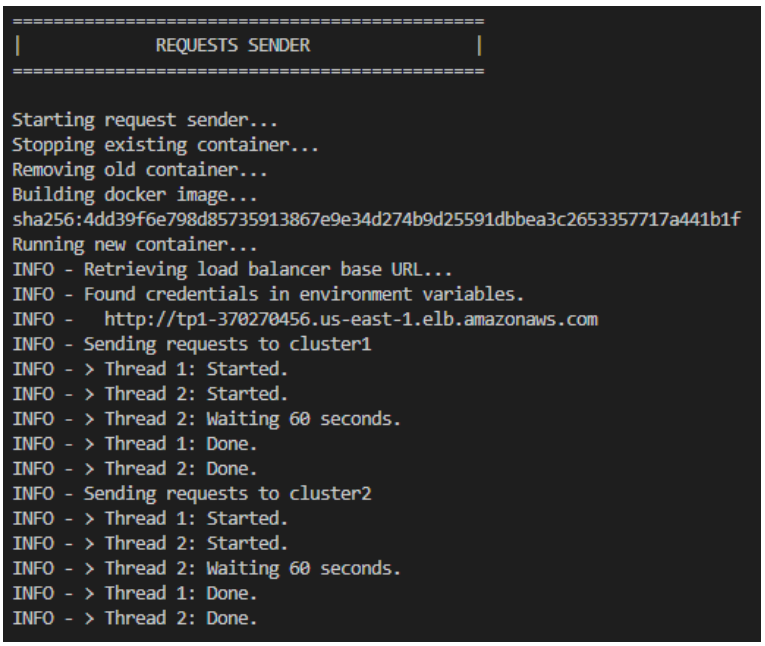

Figure 4.4 - Output of the Requests Sender step

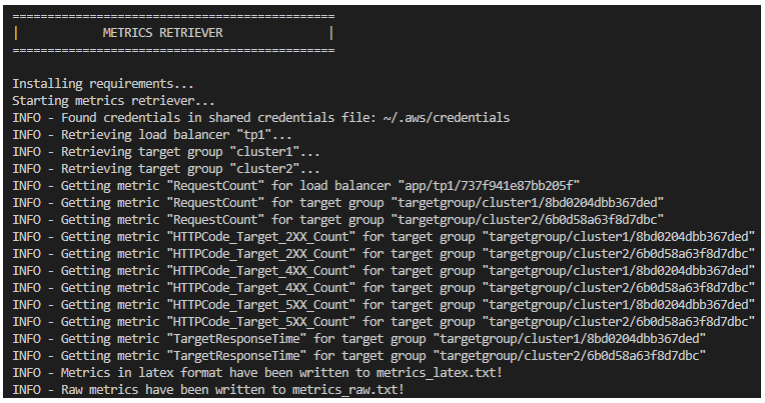

Figure 4.5 - Output of the Metrics Retriever step