Advanced Concepts of Cloud Computing
TP-1 Report

Rad Ali
Polytechnique Montréal
rad.ali@polymtl.com
1900574

Mathurin Chritin
Polytechnique Montréal
mathurin.chritin@polymtl.com
1883619

Adam Naoui
Polytechnique Montréal
adam.naoui@polymtl.com
18877197

Philipp Peron
Polytechnique Montréal
philipp.peron@polymtl.com
2230874

***Index Terms*—AWS EC2, ELB, Flask, Python**

I. Introduction

The cloud market has been growing rapidly over the last decade and is expected to continue growing. The largest provider nowadays is Amazon Web Services (AWS) with 34% market share, followed by Microsoft Azure and Google Cloud [1]. In this TP1 assignment, we created a cluster of several virtual machine instances running on AWS. To spread the load between them, we used an elastic load balancer. Each virtual machine hosts a simple web server. Finally, we also benchmarked the performance. The repository can be found under: https://github.com/PhilippPeron/advanced-cloud-log8415

II. Flask Application Deployment Procedure

To deploy the flask app on the machine instances, we pass a bash script during the instance creation with the 'UserData' argument. That bash script runs automatically on startup. It first installs python-flask as well as pip. With pip we then install ec2_metadata. As the next step, the script creates a python file that contains the flask application. The application shows different message depending on the URL path. For '/cluster1' and '/cluster2' it displays a message that tells the viewer that the instance is responding, together with the machine's instance ID. If no URL path is given, it tells the viewer to use '/cluster1' or '/cluster2' and gives the instance ID.
Finally, the bash script makes the flask-app executable, sets it as an environment variable and runs it on port 80.

III. Cluster setup using Application Load Balancer

We created the two clusters using two separate target groups that are used by the application load balancer. We registered all the instances of type M4.large to one target group and the instances of type T2.xlarge to the other one. Using path-patterns, we advised the load balancer to route traffic with the URL path '/cluster1' to the M4.large instance target group. For '/cluster2' it redirects traffic to an instance from the T2.xlarge instance target group. That way, we could benchmark the clusters separately by changing the URL of the HTTP requests.

IV. Results of Benchmarks

For the benchmarking step, we ran our script to set up the environment needed : nine instances across two target groups, and the load balancer with the appropriate rules. Since the cloudwatch metrics are updated every 5 minutes, we waited for 5 minutes to see an initial plot. We then ran our benchmarking script two times, with an interval of 5 minutes. The benchmarking script makes 1000 GET requests to /cluster1 and to /cluster2, and saves the response. 5 minutes after the second benchmark, we retrieved the CPU Utilization and Requests Across Target Group cloudwatch metrics using our third script. The results were:

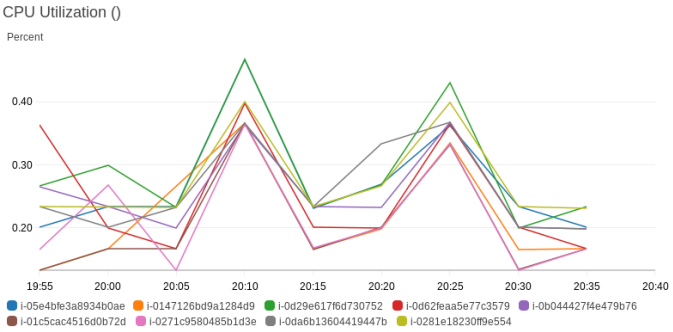

Fig. 1. CPU Utilization per instances (9 instances shown)

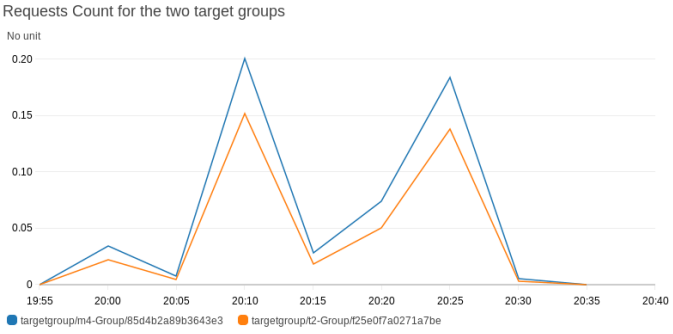

Fig. 2. Requests received per target group

We can see that the requests are equally balanced between our instances on figure 1 : the 9 curves are increasing and

decreasing simultaneously (except before 20:05, which is while our instances were still starting). The figure 2 confirms what we saw before : both of the target groups receive an equal amount of requests.

This was the output of our benchmarking script :

```
/cluster1 :
id-i-0a37ad4c6b5ab4f9c answered 251 requests
id-i-0a7aa5fdb8efcc920 answered 249 requests
id-i-0b1802e6e2fbff4bf answered 250 requests
id-i-0da4a6c92293ebafc answered 250 requests
Total successful requests : 1000

/cluster2 :
id-i-02a01f5ea43c7760a answered 202 requests
id-i-05b10c4195d3ca6f6 answered 199 requests
id-i-09a74529a662ac981 answered 200 requests
id-i-0cb3b39bb43e105ad answered 201 requests
id-i-0d402c8f961c0fe6d answered 198 requests
Total successful requests : 1000
```

We can read the exact amount of requests answered by the different instances across the two clusters.

V. Instructions to Run the Code

Assuming you have downloaded the Moodle archive, you can run our whole script like this :

```
$ chmod +x script.sh && ./script.sh
```

This will :

- setup the environment : create a new python virtual environment and install the required dependencies;
- setup the clusters : instances, security group, target groups, and load balancer;
- run the benchmark (1000 GET requests) ;
- download the cloudwatch metrics and save them as two figures : `requests_per_tg.png` and `cpu_usage_all_instances.png`;
- Delete the instances, target groups and load balancer.

Since this is automated, the cloudwatch metrics might not be as shown above, because the benchmark is run directly after the environment is setup and the metrics are retrieved directly after the benchmark is ran, thus not leaving enough time for the metrics to be updated.

References

[1] Statista, “Amazon leads 200-billion cloud market,” 2020.