POLYTECHNIQUE MONTRÉAL

LOG8415E – ADVANCED CLOUD COMPUTING

Assignment 1

Load Balancer From Scratch

Par :
Laurel, Tremblay (2141998)
Paul, Petibon (2002116)
Tristan, Lachance (2141919)

Submitted to :
Vahid, Majdinasab

September 23rd, 2024, Montreal

Table of Contents

Introduction

The purpose of this project was first to become familiar with the concept of Cloud Computing. The service we were assigned was Amazon AWS, which allowed us to learn more about it. Our objective was to create two clusters with 4 instances per cluster, as well as a Load Balancer that would manage the two clusters. Finally, we used a benchmark to analyze the performance of our clusters. This report will cover the deployment procedure, the cluster setup using an Application Load Balancer, the results of our benchmark, as well as the instructions to run our code.

FastAPI Deployment Procedure

The FastAPI deployment involved launching multiple EC2 instances in two distinct clusters, running a FastAPI application on each instance and setting up an Application Load Balancer to manage those clusters.

We used the AWS SDK for Python, which is boto3, to initialize an EC2 client and an ELBV2 client and be able to configure and manage our instances. We retrieved the latest Ubuntu AMI to ensure the use of an up-to-date, stable image for our instances.

Each instance was configured using a user data script, which was executed automatically at instance startup. The script was provided to us beforehand. It allowed us to install the different packages on our instances and run a simple FastAPI application that returns a unique identifier in response to a request.

A security group was created to allow inbound traffic on port 22 for SSH and port 8000 for the FastAPI service. This ensured that each instance was accessible for both administrative tasks and web traffic.

Cluster Setup

Since we wanted to test and use an Application Load Balancer, perform a benchmark, and compare t2.large with t2.micro instances, we made the following setup :

- **Cluster 1 :** Four t2.large instances
- **Cluster 2 :** Four t2.micro instances

Each instance ran the FastAPI application configured by the corresponding user data script.To ensure efficient traffic distribution across the two clusters, we created from scratch an Application Load Balancer (ALB) to manage incoming requests. This ALB is our ninth instance, which was a t2.large.

We created two target groups for each cluster (cluster1 and cluster2). Each target group was configured to route traffic to the instances running in that cluster on port 8000. The health check was set up on the ALB to continuously monitor the instances' availability and performance.

The Application Load Balancer was configured to distribute incoming traffic between the two target groups. This ensured balanced load distribution between Cluster1 and Cluster2, even if instances in one cluster became unhealthy or unresponsive. Our goal was also to compare a cluster that contains t2.micro instances and another that contains t2.large instances

After creating the target groups, the instances from Cluster1 and Cluster2 were registered with their respective groups. This allowed the ALB to route requests to the appropriate cluster based on its health check status.

Health checks were configured to monitor the status of the instances. These checks ensured that only healthy instances received traffic, thus improving the reliability and fault tolerance of the application.

Benchmark Results

Here are our average benchmark results :

Cluster 1 :

- Total time taken : 15.20 seconds
- Average time per request : 0.0061 seconds
- Cluster 1 distribution : '0' : 696, '2' : 690, '1' : 196, '3' : 918

Cluster 2 :

- Total time taken : 17.59 seconds
- Average time per request : 0.0070 seconds
- Cluster 2 distribution : '6' : 669, '5' : 1020, '7' : 344, '4' : 467

As can be observed, it is normal to see that Cluster 1 is faster since it contains t2.larges, while Cluster 2 contains t2.micros. However, it is important to note that the larger instances consume more resources than the micros, and therefore using this type of instance is more expensive. We can also see observe that the requests are spread between all the instances.

Instructions

Here are the steps to run the code :

- Copy and paste your AWS CLI Cloud Access credentials into /.aws/credentials
- Open a terminal in the 'LOG8415-TP1' folder and enter the following commands :

```
python -m pip install -r requirements.txt

./start.sh
```

Conclusion

In conclusion, this project not only provided us with a hands-on introduction to cloud computing through Amazon AWS, but also allowed us to practically implement and understand the workings of Application Load Balancers. Our experiments with two different clusters of instances (t2.large and t2.micro) showed significant differences in performance and cost-efficiency. The t2.large instances, while faster, highlighted the increased resource consumption and higher costs associated with more powerful hardware. On the other hand, t2.micro instances, though less performant, offered a more cost-effective solution for less resource-intensive applications.

The benchmark results clearly demonstrated the effectiveness of using an Application Load Balancer to manage traffic between differently specified clusters. This approach ensured optimal distribution of requests, maintaining system performance even when individual instances varied in health and response times.