LOG8415
Advanced Concepts of Cloud Computing
TP1

Thomas Caron - 1944066
Kevin Lam - 2024921
Ghazi Ben Achour - 1926009
Slimane Boussafeur - 2017001

Our repo can be found at: https://github.com/tomtom103/LOG8415E

1 Flask Application Deployment Procedure

In order to deploy our application, we have decided to use terraform to simplify deploying the code to every AWS EC2 instance. Terraform is an infrastructure as code tool that lets us define cloud resources using configuration files. For this assignment, we have chosen to deploy all EC2 instances, Security Groups, Application Load Balancers, and Target Groups for clusters using terraform.

The Flask application is deployed using a templated bash script, which is provided in the **user_data** part of the instance. This bash script allows us to set a new *iptables* rule, which allows us to redirect all TCP traffic coming from the port : 80 to our chosen port for the Flask application (in our case, port 5000). We then install *python* as well as all other required dependencies.

To run the flask application, we have chosen to use the *nohup* command, which means "no hang up" and allows us to execute a command and make sure that it ignores the HUP (hangup) signal when a user logs out. This makes sure that our application is constantly running, and that if a user decides to SSH inside the container, the application won't be interrupted.

2 Cluster setup using Application Load Balancer

We created two different clusters for our Application Load Balancer, respectively named $cluster1$ and $cluster2$. The first cluster points to 5 $m4.large$ instances, and the second cluster points to 4 $t2.large$ instances. Creating these clusters of instances allows us to easily balance the traffic received depending on whether the request was done to $/cluster1$ or $/cluster2$.

For the Application Load Balancer, we declare a non-internal ALB instance so that it will be internet-facing. We then declare a *listener* to listen for incoming HTTP requests.

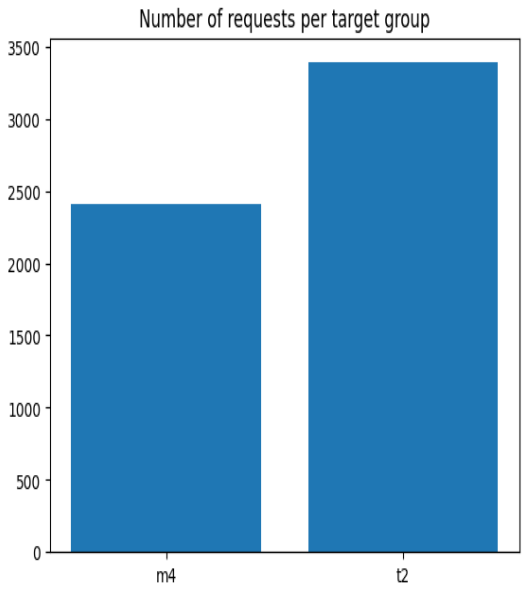
These are the declared rules for our HTTP listener:

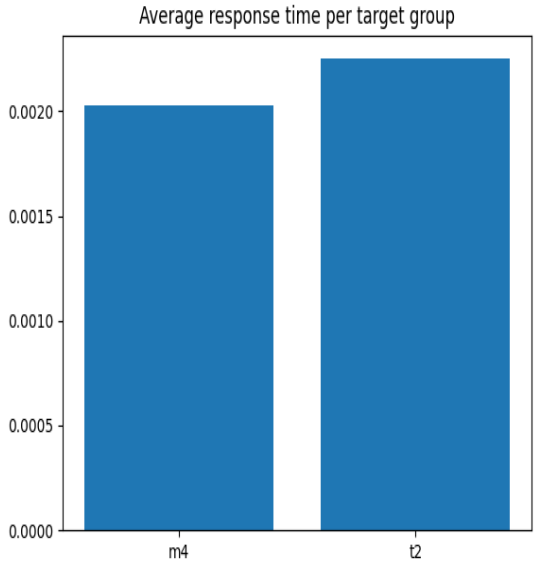
- The default action for requests coming directly to / is to return a 404, since requests should not be sent to that URL. We do this by declaring the default action of our listener to be a $fixed - response$, which allows us to return custom static content.
- We then add another rule which will forward all requests made following a certain pattern. In this case, all requests made that match the $/cluster1*$ pattern will be forwarded to the $cluster1$ cluster.
- The last rule is very similar to the previously declared rule, we forward all requests matching the $/cluster2*$ pattern to the second cluster.

We have also opted to create an $S3$ Bucket, to allow us to collect all of the ALB access logs. This bucket is created and destroyed at the same time as all other resources. This is, once again, all created and destroyed using our terraform configuration.

3 Result of our benchmark

We created two graphs to compare the number of requests and average response time. In addition, we created a table that displays all of the load balancer's metrics, such as the average active connection count, total bytes processed, total request count, and number of healthy hosts in M4 and T2.





According to the obtained bar chart of average response time per target group, M4 instances are slightly faster than T2 instances. But it's really similar still. It means that one target group is not more congested than the other. Which shows us that the balancing is done right.

According to the the bar chart , the load balancer properly distributes incoming application traffic across our target groups, since we can discern a good distribution between the two groups. There are more requests to the t2 target group which correlate with the increased average response time from that same target group.

Different elb metrics

Average active connection count	25.333333333333332
Total bytes processed	2413741.0
Total request count	5800.0
Number of healty hosts in M4	5.0
Number of healthy hosts in T2	4.0

From the load balancer metrics table, we can observe that the alb processes data properly for 5800 requests and all our instances appear to be healthy.

4 Instructions to run our code

In order to run our code, the only pre-requisite is to have terraform installed and to configure AWSCLIs. To find out how to install terraform, instructions are available at the following URL. To configure AWS credentials, the script *./scripts/login.sh* can be used to copy the access key and session token in the *./aws/credentials* file. Then, our bash script can be run using *./scripts.sh*, which will create all of the resources on AWS, run the benchmarks, fetch the results and tear-down all resources to avoid additional costs.