Lab 1 : Cluster Benchmarking using EC2 Virtual Machines and Elastic Load Balancer (ELB)

Presented to Professor Amin Nikanjam

By :
Jérémy Thai - 2016336
Jules Lefebvre - 1847158
Ismail Bakkouri - 1954157
Milen Bozhilov - 1899742

LOG8415E - Advanced Concepts of Cloud Computing

October 17th, 2022

1. Introduction

As the use of cloud computing continues to grow, it is important to have a thorough understanding of the usability of cloud computing platforms and the performance of the services they offer. This is why, in this assignment, we will analyze these points for the aws cloud-computing platform. To do this, two clusters using Load Balancers will be deployed, the nodes of which will contain Flask servers that can accept HTTP requests. Subsequently, a simulation of client requests to the Load Balancers will be launched. Finally, the aws data will all be retrieved and analyzed using the aws CloudWatch tool. In order to have a good idea of the usability of aws services, all stages of the process (deployment, simulation, data collection, termination) will be automated.

2. Flask Application Deployment Procedure

To be able to send requests and collect statistics on the load balancer, a server needs to be running to serve these requests. Since the performance of the server is not important given the use we make of it, a simple Flask server is more than enough to complete this assignment. The Flask server only needs to be able to serve requests and identify itself (in order to later know which instance of the cluster was polled). A simple get request function needs to be exposed from the server. Since we are polling two routes, one for each cluster, we need to link to routes to this get request. When querying this route, the client will receive the server instance identity as a response.

To deploy this server on an aws EC2 instance, the "UserData" field of each instance needs to be used to run the Flask code on creation. All the code in the "UserData" field will be run as shell script at the instance creation. This way, through this shell script, by simply installing python and Flask before running the Flask code with python, the Flask server can be deployed on any EC2 instance. The shell script code only needs to be adapted to the corresponding os.

The next section will explain the cluster setup and the necessary steps to expose the server instances to the internet.

3. Cluster setup using Application Load Balancer

To deploy the clusters with a load balancer, we had to:

- Create nine (9) EC2 instances - four (4) t2.large, and five (5) m4.large;
- Register the EC2 instances into two (2) different target groups to create distinct clusters;
- Create a load balancer;
- Bind the two target groups to the load balancer, with a listener;

- Create routes to access the clusters (/cluster1 and /cluster2).

To automate this process, we used a python 3 script, with the boto3 api.
To create the EC2 instances, we used **run_instances**() function, the parameters that we had to set up are:

- **BlockDeviceMappings**: allows tweaking the storage parameters of the instance.
- **ImageId:** allows selecting Ubuntu as the EC2 OS.
- **InstanceType**: allows selecting the instance type (t2.large, or m4.large).
- **Placement**: allows selecting the data center in which the instance will be hosted.
- **DisableApiTermination:** allows termination of the instance with boto3.
- **UserData**: allows selecting a script which is gonna be executed on the instance startup.

To create the TargetGroups, we used the function **create_target_group**() with these parameters:

- **Target_type:** allows selecting the target group type. In this case, it's set to 'instance', because we want to group EC2 instances.
- **Name:** allows setting up a name to identify the target group.
- **Port**: allows selecting the port on which the target group receives the traffic.

To register the instances to the target group, we used the function **register_targets**() with these parameters:

- **TargetGroupArn:** allows selecting the target group arn, used to identify the target group in which we want to link the instances.
- **Targets:** allows selecting a list of instances to link to the target group.

To create the load balancer, we used the function **create_load_balancer**() with these parameters:

- **Type:** allows selecting the Load balancer type, in this case, it's an application load balancer.
- **Name:** allows setting up a name to identify the load balancer.
- **Subnets:** allows selecting the IDs of the public s
- **IpAddressType:** allows selecting the ipType (ipv4 or ipv6) used by the subnets

To bind the target groups to the Load balancer and setup the routes to each one, we used the functions **create_listener**() and **create_rule**()

For the function **create_listener**(), we setup these parameters:

- **LoadBalancerArn:** allows selecting the load balancer arn, used to identify the load balancer in which we want to link the target group.
- **Port**: allows selecting the port on which the load balancer receives the traffic.
- **Protocol:** allows setting up the protocol to route the traffic.
- **DefaultActions:** Allows specifying the default mandatory route ('/') in this case, it's routed to the first cluster.

We used the function **create_rule**() to create the routes **/cluster1** and **/cluster2**. To do so, we used these parameters:

- **ListenerArn:** allows specifying the listener that we just created with create_listener
- **Actions:** allows selecting the action to apply when receiving a request to a specified route. In this case it's a forward to a target group.
- **Conditions:** allows to create a route to send the queries to the clusters.
- **Priority:** allows to set the priority of the new rule.

4. Benchmark results

After performing the test scenarios locally, a comparison was done between the results of benchmarking for EC2 instances from each cluster. This comparison was done using specific metrics related to the target groups and also the EC2 instances.

4.1 Table 1 - Target Group Metrics

Here is a description of the selected metrics related to the target group followed by an explanation of the results:

Metric	Description
Request Count Per Target	The average number of requests received by each target in a target group. [1]
Target Response Time (ms)	The time elapsed, in seconds, after the request leaves the load balancer until a response from the target is received. [1]
HTTP 2XXs	The number of HTTP response codes generated by the targets. This does not include any response codes generated by the load balancer. [1]

4.1.1 Request Count Per Target Comparison

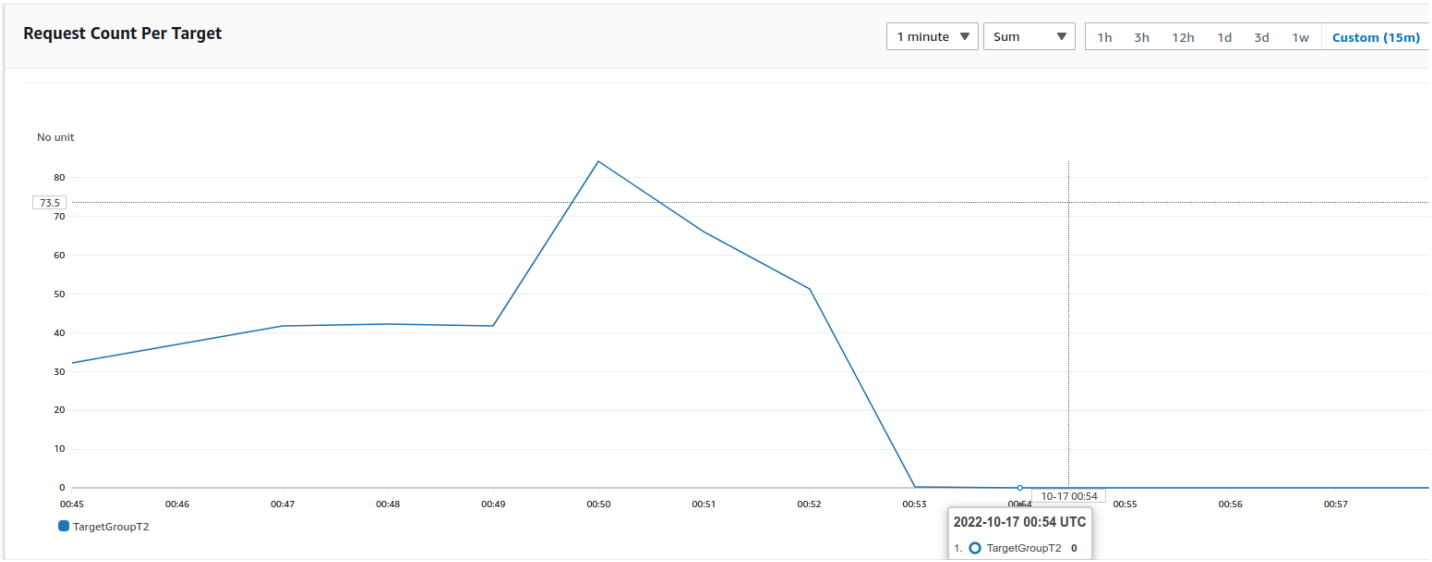

Figure 1 Request count per target plot for TargetGroupT2 where Y axis is request count per target and X axis is time.

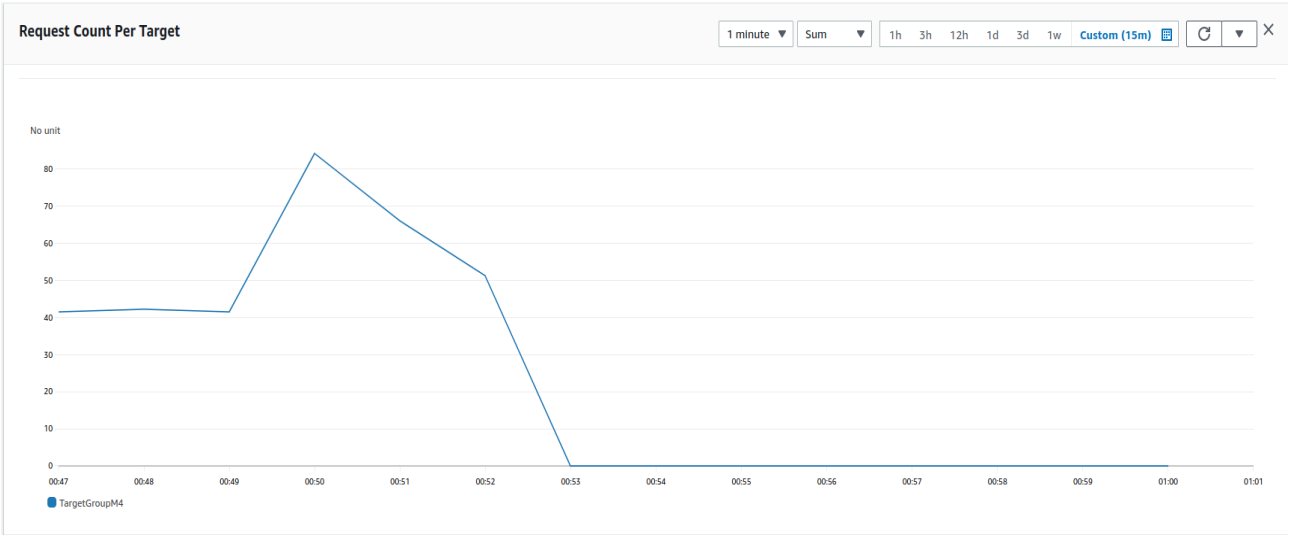

Figure 2 Request count per target plot for TargetGroupM4 where Y axis is request count per target and X axis is time.

As shown in figure 1 and 2, The number of request count per target looks to be exactly the same for both clusters. This means that the workload sent to both target groups was exactly the same due to the load balancer.

4.1.2 Target Response Time Comparison

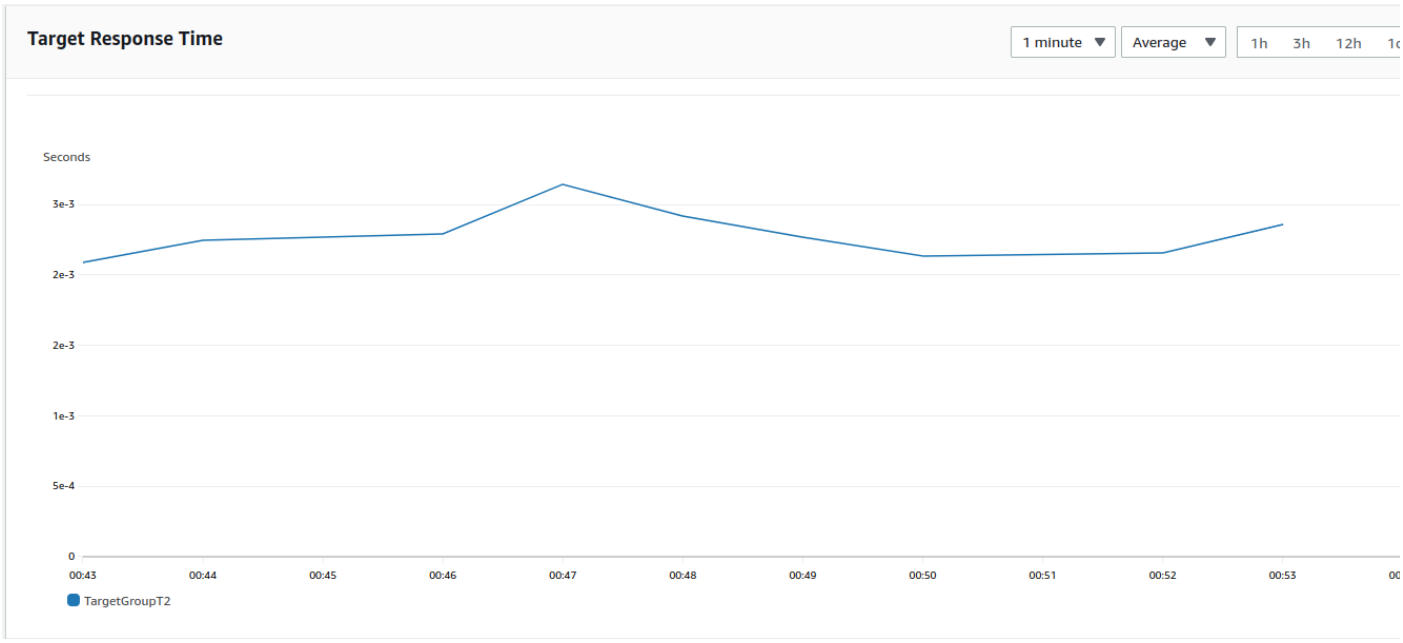

Figure 3 Target response time plot for TargetGroupT2 where Y axis is time elapsed until response from target and X axis is time.

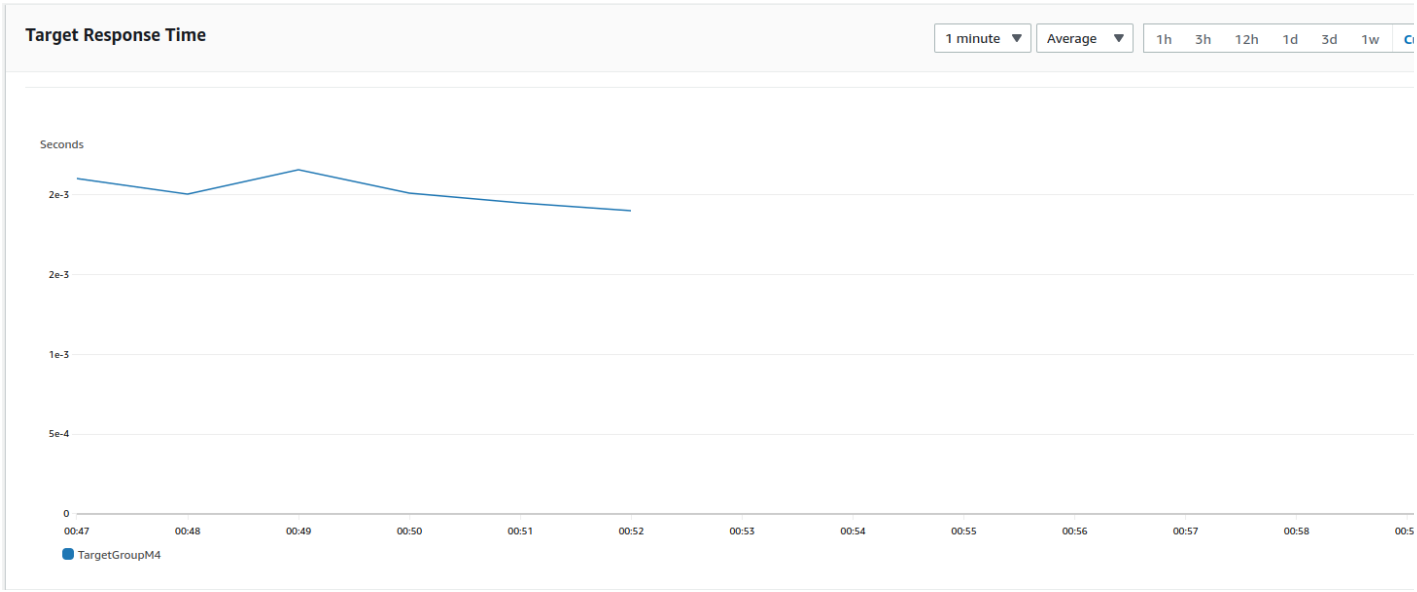

Figure 4 Target response time plot for TargetGroupM4 where Y axis is time elapsed until response from target and X axis is time.

Over a period of 1 minute, we can see in figure 4 that cluster #2 (TargetGroupM4) managed to keep a lower response time on average. In the case of cluster #1 (TargetGroupT2), figure 3 shows

that its response time was generally fluctuating between 0.0025 and 0.003 seconds, leading to a higher average response time. Moreover, a sudden spike can be observed towards the end of the period for cluster #1.

4.1.3 HTTP 2XXs Comparison

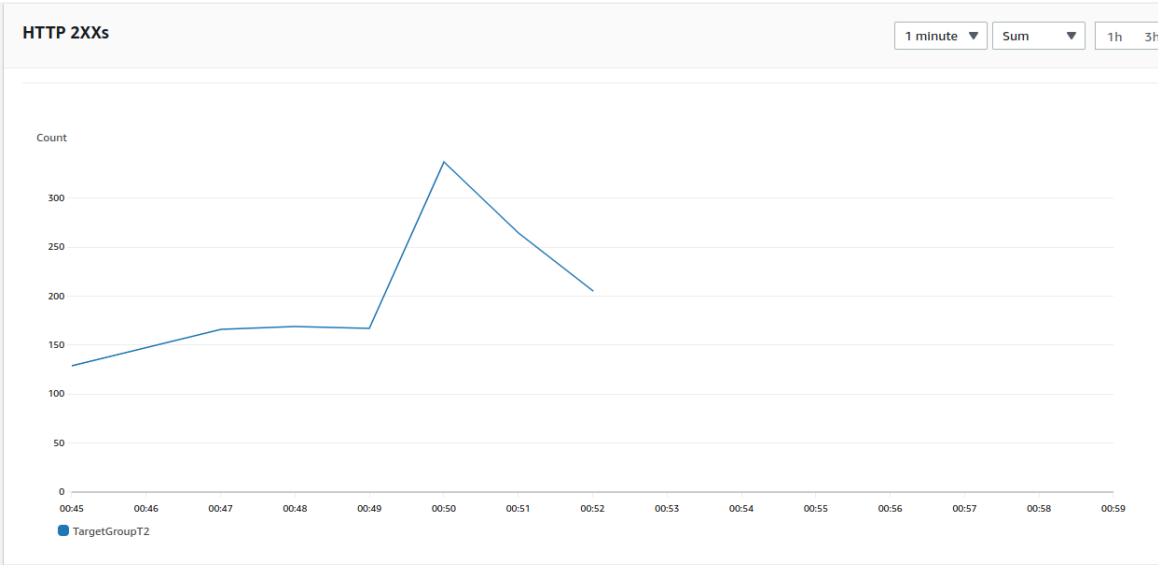

Figure 5 HTTP 2XXs plot for TargetGroupT2 where Y axis is the number of HTTP 2xx response codes generated by target and X axis is time.

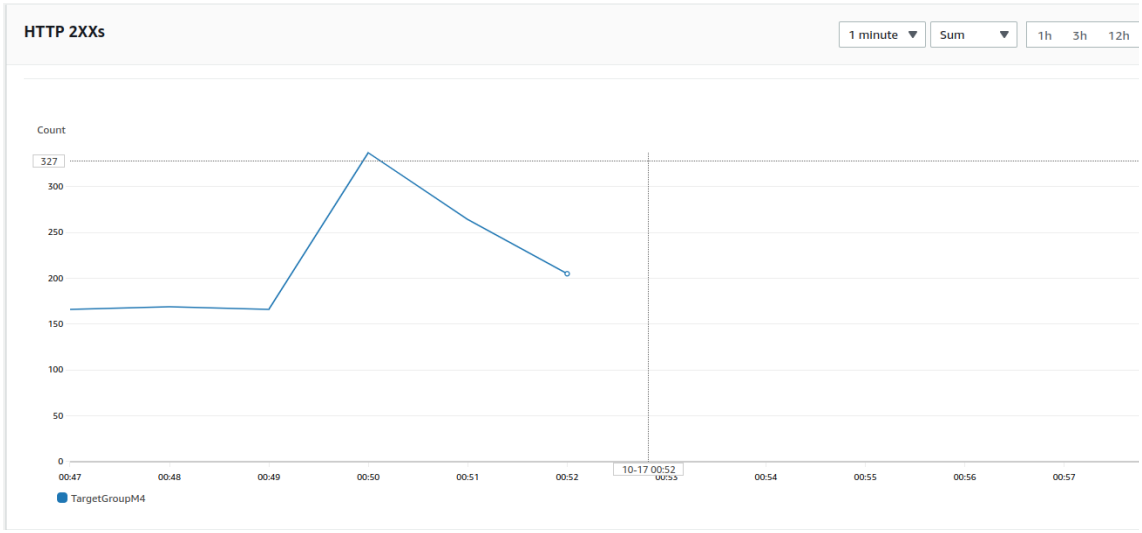

Figure 6 HTTP 2XXs plot for TargetGroupM4 where Y axis is the number of HTTP 2xx response codes generated by target and X axis is time.

As shown in figure 5 and 6, the graph is fluctuating in a similar manner for both clusters. A spike is observed at approximately 00:50s for both clusters, just like in the Request Count Per Target plots shown previously. Both clusters are behaving the same way, as most requests received are returned with a 2XX response code.

4.2 EC2 Instance Metrics

Here is a description of the selected metric related to the EC2 instances followed by an explanation of the results:

Metric	Description
CPU Utilization (%)	The percentage of allocated EC2 compute units that are currently in use on the instance. [2]

Using the output results in our console, the average CPU utilization per instance from each cluster was computed to give a better interpretable result. For cluster #1, we found an average of 2.17%. For cluster #2, we found an average of 10.03%. Therefore, it is clear that cluster #1 shows a much better use of compute resources overall.

5. Instructions to run the code

Before executing the main Python script, the following prerequisites must be met:

- The code must be located on the local machine
- Python 3 must be installed locally on the machine
- Docker must be installed locally on the machine
- User should have an AWS account

The next step is to set up the authentication credentials for the AWS account. If it does not already exist, a file named "credentials" should be created under "~/.aws/". Inside this file, both the aws access key id and secret access key should be specified. This will allow the boto3 client to have access to AWS. Another important file to have is the "config" file, also located under "~/.aws/". This file should specify a default region to use such as "region=us-east-1".

Now that configurations are done, the last step is to execute the "scripts.sh" bash script on instance creation, which will install python dependencies and run the main program. Log messages will be displayed in the console during each step of the execution. As for the results, they will be shown directly in the console or in the json folder.

The scripts are publicly available in the following GitHub repository: https://github.com/J-Lefebvre/LOG8415E.

References

[1]https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-cloudwatch-metrics.html

[2]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/viewing_metrics_with_cloudwatch.html