LOG8415
Advanced Concepts of Cloud Computing

William Bourret (2005931), Vlad Drelciuc (1941366),
Charles Fakih (2012512), Simon Tran (1961278)

Département Génie Informatique et Génie Logiciel
École Polytechnique de Montréal, Québec, Canada

1 Flask Application Deployment Procedure

We build a Flask application that simply listens on port 80 and returns "[instance_id] is responding now!". In order to deploy this app on each of our Amazon EC2 instances, we are using an AWS concept called UserData. UserData allows us to provide a bash script that will be executed automatically each time a new instance is deployed. Our script (available under tp1/flask_setup.sh) performs the following steps:

1. Update the apt packages
2. Install pip to manage our Python dependencies
3. Install Flask using pip
4. Create a flask_app directory
5. Create an app.py file
6. Grab the EC2 instance ID using ec2metadata
7. Inside app.py, initialize the Flask app with a route on "/ " and make the route return "[instance_id] is responding now! " when called on port 80
8. Execute the app

2 Cluster Setup

Out scripts performs the following cluster setup:

1. It creates an AWS Security Group (called custom-sec-group) that allows inbound access to ports 80 (HTTP) and 22 (SSH) and outbound access to all ports. All of our AWS EC2 Instances, Target Groups and Load Balancers will use this Security Group.
2. It launches four T2.large EC2 instances (2 vCPU, 8 GB RAM)

3. It launches five M4.large EC2 instances (2 vCPU, 8 GB RAM)
4. It creates one Target Group called: cluster1-tg
5. It attaches all T2 instances as targets of cluster1-tg
6. It creates one Target Group called: cluster2-tg
7. It attaches all M4 instances as targets of cluster2-tg
8. It creates one Application Load Balancer called: cluster1-elb
9. It attaches cluster1-tg as listener of cluster1-elb
10. It creates one Application Load Balancer called: cluster2-elb
11. It attaches cluster2-tg as listener of cluster2-elb

The Flask application will automatically be deployed as part of steps 2 and 3. Once the clusters are setup, we then run 2 independent workloads (workload 1 = 1000 GET requests sequentially; workload 2 = 500 GET requests, then one minute sleep, followed by 1000 GET requests) in parallel using threads and collect CloudWatch metrics (NewConnectionCount, ProcessedBytes, TargetResponseTime and RequestCountPerTarget) for: cluster1-tg, cluster2-tg, cluster1-elb and cluster2-elb.

3 Benchmark Results

As we can see in the NewConnectionCount metric's graph, both apps receive the same number of new TCP connections, which is expected since both receive the same load of requests in the benchmarking tests we perform through their respective LoadBalancer.

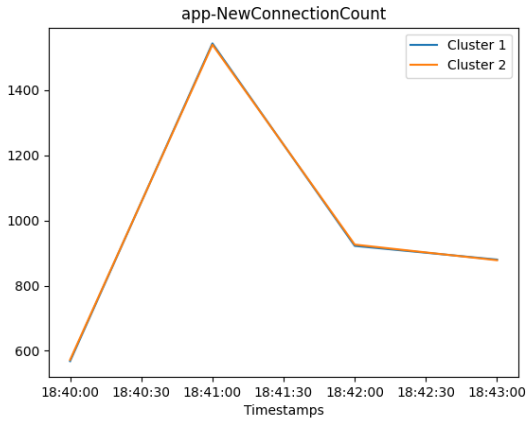

Figure 1: NewConnectionCount Metric

We can make the same observation for the ProcessedBytes metric's graph. Here, we measure the total number of bytes processed by each LoadBalancer. Since we send the same traffic to both

the LoadBalancer for the M4's as we do for the LoadBalancer with the T2's, the processed bytes are the same in both. Thus, in both graphs, the two lines are superposed.

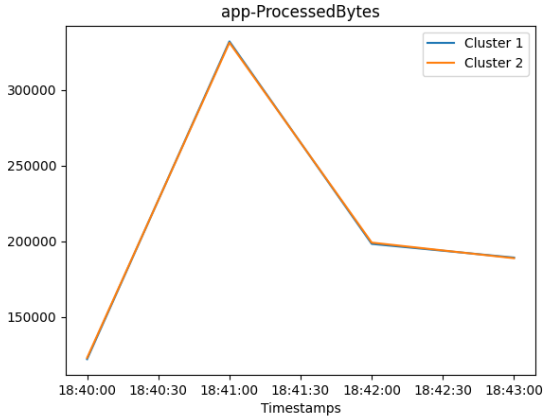

Figure 2: ProcessedBytes Metric

The TargetResponseTime metric's graph shows that the second cluster's target group which contains the M4's has a response time that is roughly 25 % higher than the first cluster's target group which involves the T2 instances. This is explained by the fact that we have 5 M4 instances whereas we have 4 T2 instances.

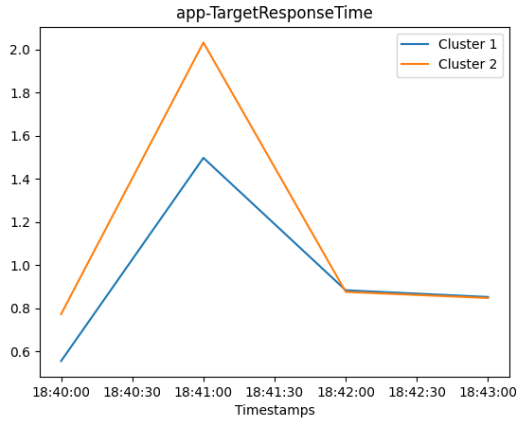

Figure 3: TargetResponseTime Metric

The same can be said about the RequestCountPerTarget graph which follows the metric counting the average number of requests received by each one of the target groups. As we can see, the second cluster has a consistently 25 % higher number of request counts owing to a bigger number of instances.

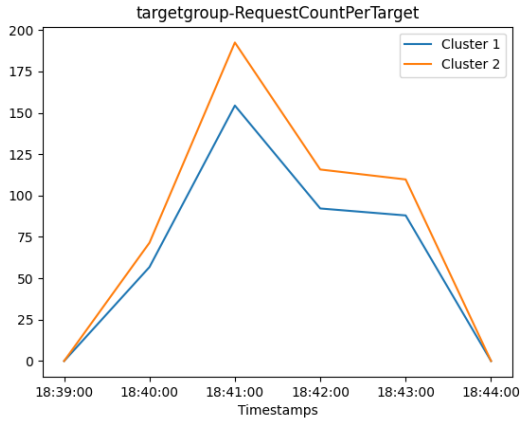

Figure 4: RequestCountPerTarget Metric

4 How to Run the Code

To run the code, you will need to:

1. Install and run Docker Engine on your local machine
2. Navigate to the project folder: **cd tp1**
3. Update the environment variables inside **.env** with the correct values for AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY and AWS_SESSION_TOKEN
4. Make **scripts.sh** executable: **chmod +x scripts.sh**
5. Then simply execute: **./scripts.sh**