LOG8415
Advanced Concepts of Cloud Computing
TP1

Marwane Adala
Vahid Majdinasab
Chun-An Bau
{marwane.adala, vahid.majdinasab, chun-an.bau}@polymtl.ca

We introduced how to set up the system, including the instances, the target groups, and load balancer in section 1, and recorded what should be noticed while launching the Flask application in section 2. In section 3, we described what metrics did we use to evaluate our system. Finally, we showed the procedures of running all the scenarios and benchmarks in section 4. The setup procedures and corresponding scripts could be found in our GitHub repositories [4] and [3].

1 Environment Setup

We set up the system in the order of **instances**, **target groups**, and the **load balancer**.

1.1 Instance

The setup procedures were relatively smooth, but we kept on failing to launch the 10th instance, which might result from resource limitations. The **security group** of the instances should allow the specific inbound traffic. In this assignment, we should allow inbound traffic with TCP port 80 to reach the instances. (In our implementation, we permitted all the TCP inbound traffic.)

1.2 Target groups

We registered the instances into two target groups, which named *cluster1* and *cluster2*, respectively. We used */cluster1* and */cluster2* as the **health check path** while building up the target groups.

1.3 Load Balancer

The balancer type we used is **Application Load Balancer**, which was a useful tool to manage the HTTP/HTTPS traffic. While building up the balancer, all availability zones that the instances stayed should be selected. We added 2 listener's rules:

- Forward the traffic to *cluster1* if the request's postfix is */cluster1*
- Forward the traffic to *cluster2* if the request's postfix is */cluster2*

After that, the requests sent to the DNS of the load balancer with specific postfix would be forwarded to the defined destinations.

2 Flask Application Setup

Since we ran the flask application on port 80, a restricted port is only authorized for the root user. We had to launch the flask program with superuser. If the instance received a request that was forwarded by the load balancer, it would return a short clip of json object, which contained the instance ID of the responding instance [5].

3 Benchmark

We got the benchmark information using *AWS CLI* by following the instructions on the websites [6][2].

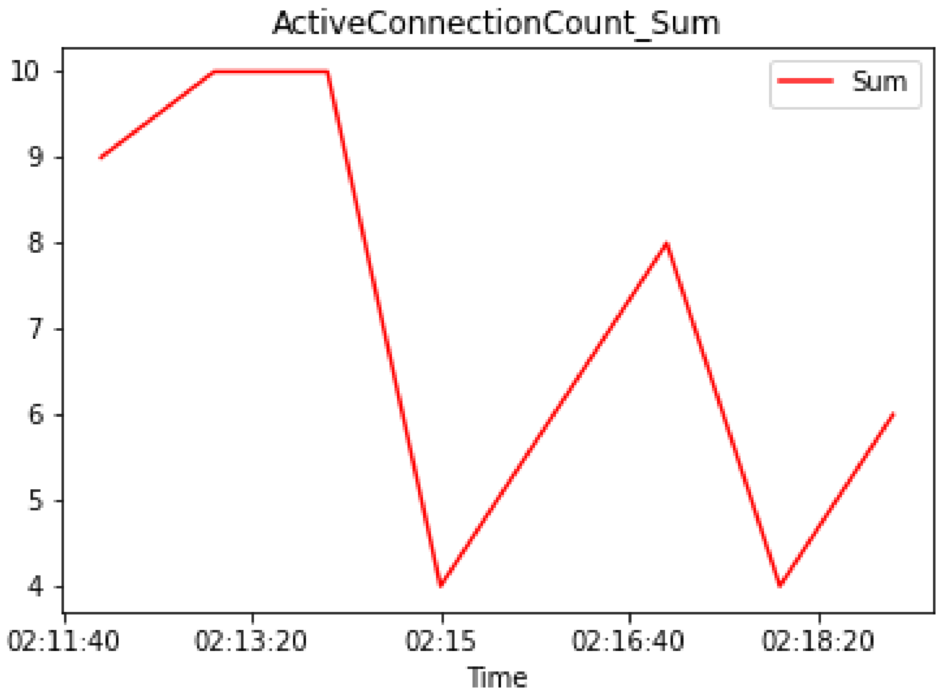

Figure 1: ActiveConnectionCount Metric

For this benchmarking step, we wrote a script to build up evaluation environment, to send, as a first scenario, 1000 GET requests sequentially and to send, as a second scenario, 500 GET requests, then one minute sleep, followed by 1000 GET requests. Our script run four scenarios in total (two for each cluster). After executing these scenarios, we provide a bash script (see metrics.sh) that provides two functions: one to get all the metrics possible from the load balancer and the other to get all the metrics possible from target groups (from the two clusters specifically). To perform benchmarking and evaluate our system, we chose to highlight the following metrics (the first two metrics are for the two clusters and the remaining four metrics are for load balancer):

- RequestCountPerTarget

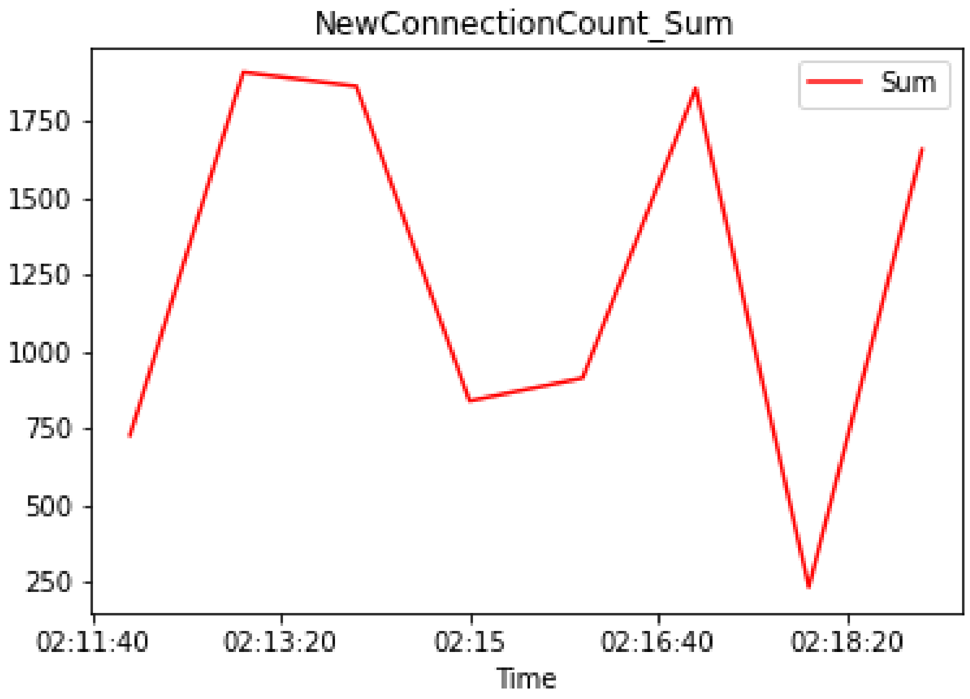

Figure 2: NewConnectionCount Metric

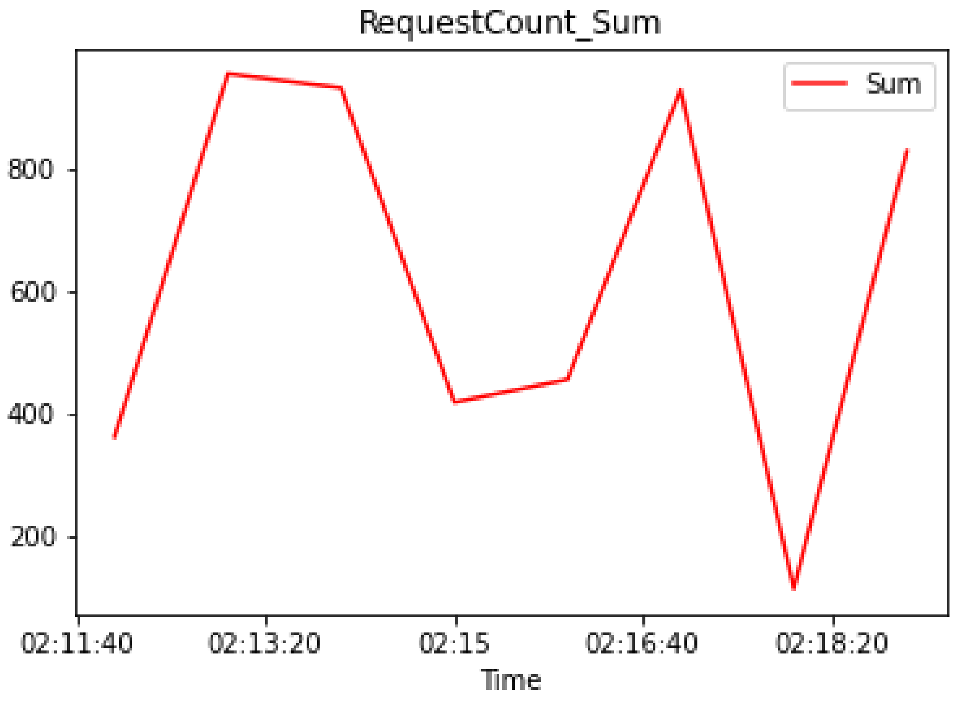

Figure 3: RequestCount Metric

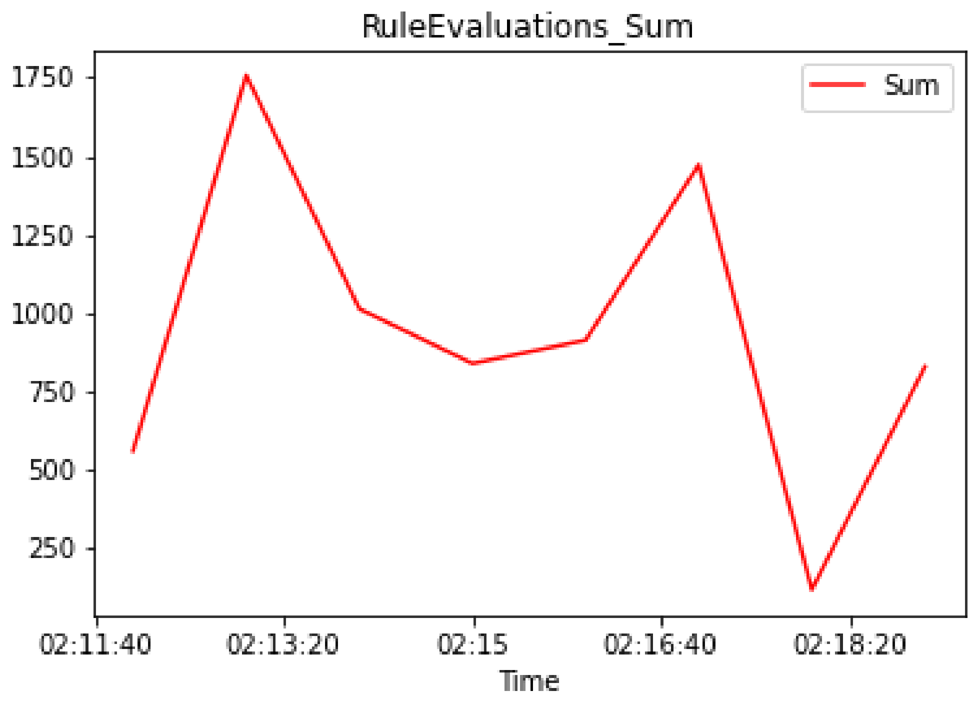

Figure 4: RuleEvaluations Metric

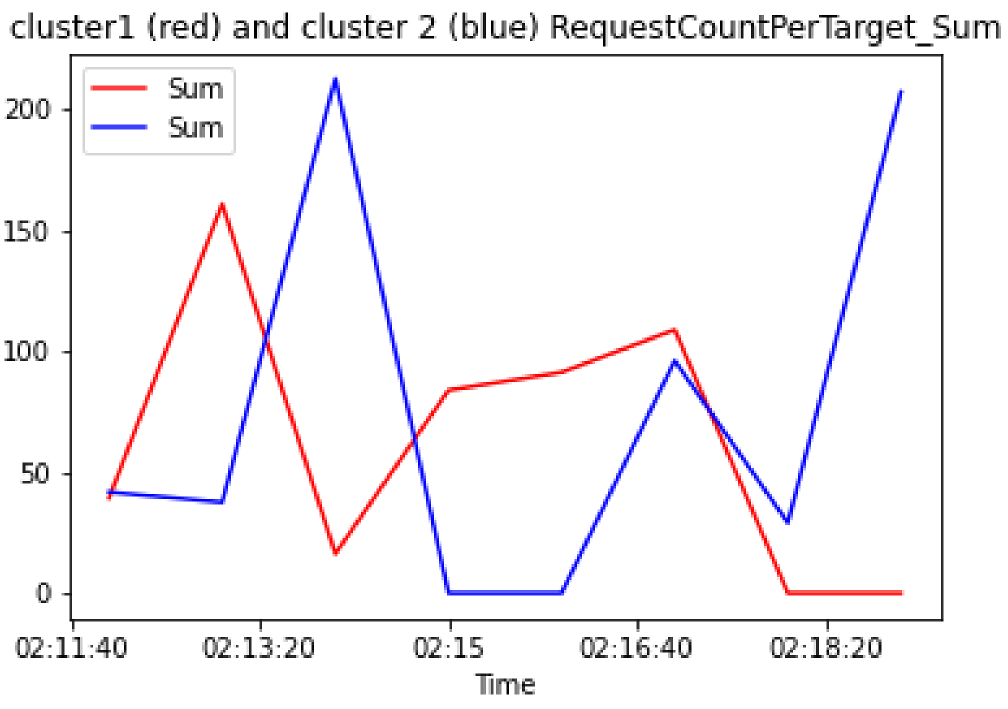

Figure 5: Clusters RequestCountPerTarget Comparison

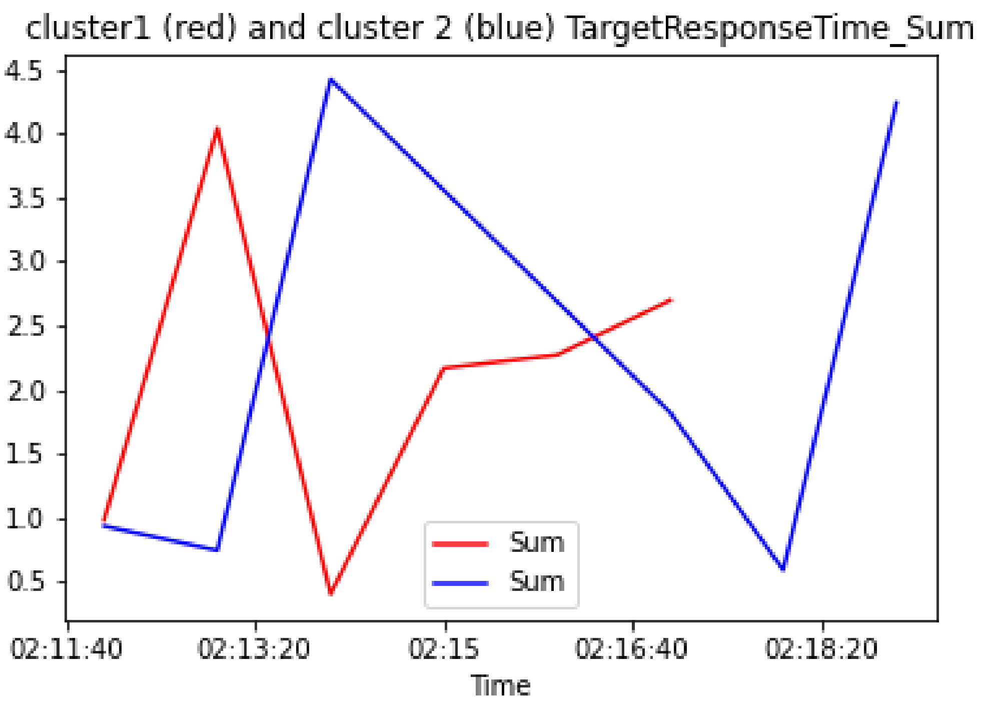

Figure 6: Clusters TargetResponseTime Comparison

- TargetResponseTime
- ActiveConnectionCount
- NewConnectionCount
- RequestCount
- RuleEvaluations

We refer the reader to the Figures 1, 2, 3, 4, 5 and 6 to get a clear idea about the results we have obtained for these six metrics. We should note first that all these results were obtained on 2021-10-06 (we only specified the time and remove the date from the x-axis of each metric). For ActiveConnectionCount metric, we should first say that it corresponds to "the total number of concurrent TCP connections active from clients to the load balancer and from the load balancer to the targets"[1]. What we notice for this first metric that it is maximum at the beginning then it starts to decrease and to oscillate in a variable way. NewConnectionCount metric correponds to "the total number of new TCP connections established from clients to the load balancer and from the load balancer to targets"[1]. This second metric looks and behaves similarly to the first, in the variation rate. As for RequestCount metric, it corresponds to "the total number of data requested by the load balancer over IPv6"[1]. The fourth metric RuleEvaluations correponds to "the number of rules processed by the load balancer given a request rate averaged over an hour"[1]. These last two metrics have the same variations as NewConnectionCount metric. We should admit that these

four metrics give us an idea about the traffic inbound and outbound for our load balancer and the two clusters.
Furthermore, the RequestCountPerTarget corresponds to "the average number of requests received by each target in a target group"[1] and the TargetResponseTime corresponds to "the time elapsed, in seconds, after the request leaves the load balancer until a response from the target is received". These two last metrics enable us to compare between the two clusters of instances. In particular, we note that after applying the two scenarios (the two streams of GET requests), we find that cluster 2 managed to receive more requests in average than cluster 1 especially after examining the graph for RequestCountPerTarget. In addition and based on the TargetResponseTime, we find that cluster 1 is better than cluster 2 in responding (shorter response time in average). This different percentage of request per cluster can be due to different instances number (the first cluster contains 5 m4.large instances and the second cluster contains 4 t2.xlarge instances). At the end, we should underline the fact that we have similar partition of inbound/outbound traffic in each cluster.

4 Execution Method

We wrote scripts to build the whole system up as well as run the benchmark and provided metrics and figures. All these steps can be done using scripts.sh (see scripts.sh) which build and run a single docker container (docker-metrics:latest). This docker container creates instances (5 m4.large instances for cluster1 and 4 t2.xlarge instances for cluster2). It installs also Flask and deploys a simple script on each instance. It creates then our load balancer and two target groups (for the two clusters). It creates also a listener and binds it to the given target group and load balancer. It binds given instances to given target group. After that, the same docker container performs the two scenarios (required and explained in the assignment) of testing on each cluster. Finally, our docker container provides us with all possible metrics from load balancer and target groups, and, it plots the graphs for these metrics. The docker terminates all instances at the end. To successfully run the scripts, the reader should download the github repo [3] which contains the corresponding Dockerfile and all the required scripts. We should also note that some values must be changed to run the scripts successfully (the location for the ec2-keypair, the subnet networks for the load balancer, the ImageID, and the information for Load balancer, cluster1 and cluster2,...etc). After that, we should make scripts.sh executable and execute it to build and run the docker image.

References

[1] *AWS CloudWatch Metrics Documentation.* URL: https://docs.sysdig.com/en/docs/sysdig-monitor/metrics-dictionary/cloud-provider/aws/elastic-application-load-balancing-alb/.

[2] *CloudWatch metrics for your Application Load Balancer.* URL: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-cloudwatch-metrics.html.

[3] *GitHub repo for Docker.* URL: https://github.com/marsup13/Lab1LOG8415DockerVersion/.

[4] *GitHub repo for LAB1.* URL: https://github.com/marsup13/lab1LOG8415.

[5] *How to get the instance id from within an ec2 instance?* URL: https://stackoverflow.com/questions/625644/how-to-get-the-instance-id-from-within-an-ec2-instance.

[6] *Load-balancer-cloudwatch-metrics*. URL: https://github.com/awsdocs/elb-application-load-balancers-user-guide/blob/master/doc_source/load-balancer-cloudwatch-metrics.md.