Load Balancing using Round Robin

Group 6 : Akhil Karrothu (ak8367) Sri Rachana Achyuthuni (sa9547)

ABSTRACT

For our term project, we are planning to implement Round Robin Approach for Load Balancing in Cloud Computing. We would also be comparing this Load Balancing approach with two other known Load Balancing Approaches - AWS Classic Load Balancer and HAProxy load balancer. This report presents a detailed description of our implementation and comparative analysis.

1. INTRODUCTION

Load Balancing is a popular technique introduced commonly in Cloud Computing to ensure that the load on each server in a cluster is distributed equally. Load Balancing ensures high performance and throughput. In cloud computing, it facilitates distribution amongst servers placed anywhere in the world, thus, decreasing the response time. In case of failures, further requests are transmitted to other alive servers in the cluster maintaining reliability of the application.

There are multiple ways of implementing Load Balancing. The most common and effective way is to have a proxy server which stands as the gateway or the door for all incoming requests. This proxy server will then redirect requests to the web servers that stand behind the proxy server. The incoming server is usually unfamiliar with which web server it is interacting as the IP address that reflects is of the proxy server.

In our project, we first implemented Round Robin[9] in Java. Next, we deployed Nginx [6] containers using Docker [4] that use the inbuilt Round Robin algorithm to redirect requests to Apache 1 containers, the details of which have been described in section 3. Then, we implemented HaProxy [5] and AWS Load Balancer [1] (section 4). The motivation behind choosing Docker and these particular Load Balancing systems can be seen in section 2. We present an evaluation and comparative analysis in section 5. Finally, we present our analysis and limitations of the Round Robin approach in section 6, the Discussions section.

2. MOTIVATION

The reason behind choosing Docker containers for the initial stage of implementation along with Nginx is that containers are light-weight and cost-free. The initial stage of implementation, setup and testing requires a lot of debugging. Multiple requests would incur costs when implemented on cloud. Also, it would be time taking and incur a lot of costs if multiple servers need to be accessed remotely. Using virtual machines locally would prove to be machine intensive. Thus, for the initial stage of implementation, we chose to use Docker. Through this, we could also explore networking amongst containers.

```
FROM ubuntu:16.04
RUN apt-get -y update && apt-get -y install apache2 && rm -rf /var/lib/apt/lists/*
CMD apachectl -DFOREGROUND
```

Figure 1: Dockerfile for deploying Apache Web Server

For the comparative analysis, since cloud computing is being used extensively today, we decided to launch AWS Classic Load Balancer and study it closely. This particular Load Balancer was the foremost Load Balancer launched. Even though it is being deprecated now, it is very popular. HaProxy is the other load balancer that we have used for the comparative study. It is similar to Nginx - open-source, freely available and extremely fast and reliable. Considering these similarities, we thought it would be interesting to compare and find any existing differences amongst them.

3. IMPLEMENTATION

3.1 Docker implementation

There are three main components in the implementation, the details of which are described in detail in this section.

3.1.1 Deploying Apache Container

Apache is one very well known web server which is used frequently for hosting web sites. To launch the container with Apache in it, we have created a Dockerfile which has base image - Ubuntu16.04 Then, in the Dockerfile, we have specified the installation of Apache modules and the command to start Apache service in the foreground. The image of the Dockerfile can be found in Figure 1. Next, we have manually added an index.html file in */var/www/html* within the container which is the directory from which the files are hosted in Apache. The index.html file added can be seen in Figure 2

3.1.2 Round Robin in Nginx

We have implemented a Round Robin algorithm in Java using interfaces and classes. However, we discovered that integration of custom Load Balancing algorithm in Nginx needs some heavy-lifting and will also compromise on the efficiency. Thus, we chose to use the in-built modules of Nginx to implement Round Robin Algorithm. This includes

```
<!DOCTYPE html>
<html>
<head>
<title>Hello</title>
</head>
<body>

<h1>Hello!</h1>
<p>This is from container 1</p>

</body>
</html>
```

Figure 2: index.html file being hosted in Web Server

```
FROM nginx
COPY ./nginx.conf /etc/nginx/nginx.conf
```

Figure 3: Dockerfile used for hosting nginx

setting up an upstream block of servers and then routing the traffic to this upstream in the nginx.conf file. The complete file has been provided.

3.1.3 Deploying Nginx container

After the modification of the nginx.conf file, we have created a Dockerfile for deploying nginx with the modified nginx.conf file by copying the nginx.conf file from the local directory to the */etc/nginx/* folder. For this, the base image used is the enterprise image of nginx which is automatically started as a foreground process, once deployed. The content of the file can be seen in Figure 3. We have created a bind between the localhost's 8080 port and the container's port 80 on which Nginx is listening.

3.2 Deployment onto EC2

In order to verify the usage of Round Robin approach on cloud, we have deployed 3 EC2 servers - 1 for Nginx and 2 for Apache Web Servers. We opened up port 80 on the Nginx server and gave it public access. For the Apache Web Servers, we set up security groups [7] such that the access onto port 80 was allowed only for the Nginx server. This can be achieved by adding a rule in the security group of EC2 server mentioning that the allowed incoming traffic is from the Nginx Server.

4. HAPROXY AND AWS LOAD BALANCER

HaProxy is a fast, reliable and open-source Load Balancer. We have launched an EC2 server, installed HaProxy using yum [8]. Then, we implemented Round Robin in this HaProxy by modifying the config file and adding an upstream of the backend servers, as shown in Figure 4.

The same backend Apache Servers used earlier for the implementation and testing of Nginx on EC2 were used as the backend servers with the modification in the security groups such that incoming traffic is allowed only on port 80 from the HaProxy server.

AWS itself offers different types of Load Balancers such as Classic, Application, Network and Gateway Load Balancer

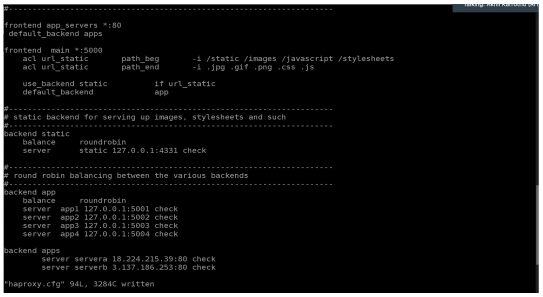

Figure 4: HaProxy config file

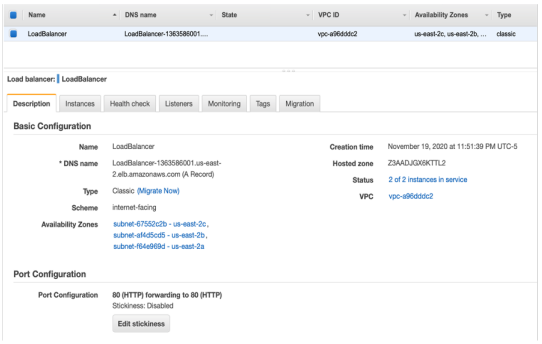

Figure 5: AWS Classic Load Balancer

[2]. For the sake of our evaluation, we have launched a Classic Load Balancer, as shown in Figure 5 and added the 2 Apache server instances to it.

5. EVALUATION AND RESULTS

In this section, we will be looking into some basic tests conducted and the results obtained in each of the load balancers.

5.1 Docker and Nginx System

In order to test the effectiveness of the Docker system that we have built, we have deployed 5 Apache web server containers whose IP addresses have been added to the upstream of the Nginx container. Then, we have repeatedly hit on the running Nginx container by visiting - *http://127.0.0.1:8080*

We found that every time the site is hit, in a round robin fashion, each container's index.html page was being displayed. An example can be seen in Figure 6.

Then, we killed one of the containers. Without modify-

Figure 6: index.html file being rendered from container by Nginx

Figure 7: Blank page from AWS Classic Load Balancer

Figure 8: 502 Bad Gateway response from Nginx

ing the upstream configuration in Nginx, we have repeated tried to reach the server. We found that the requests do not get dropped. Once the container whose IP has been configured in the upstream is not found, the request is routed to the next available container in the round robin sequence itself. This takes a couple more seconds for the very first time. However, from the next time on wards, the round robin sequence continues without any additional time on all the available containers. We have also generated multiple simultaneous requests from multiple browsers and found that the round robin sequence is withheld quite well. In 9/10 browser sessions, the round robin sequence took place as expected. In one session, due to the concurrency perhaps, it took a couple of seconds for the round robin to executed as expected.

5.2 Comparative Analysis

5.2.1 Fault Tolerance

When all the web server containers were killed, as expected, the load balancing Nginx server rendered a 502 Bad Gateway error as indicated in Figure 8 which means that the backend server which the proxy is trying to connect to has sent a bad response. This message is not very obvious but it does give the user a fair overview. In case of HaProxy, we can see from Figure 9 that the error message is more clear, indicating that there seem to be no active servers to handle requests. In case of AWS Classic Load Balancer, a blank page is rendered which can be seen in Figure 7 which is very unclear and is certainly a drawback.

When a couple of servers were down, Nginx took a couple of seconds to acknowledge the failure of the systems while both HaProxy and AWS Classic Load Balancer were quick to acknowledge and start routing the requests to other systems in the cluster.

5.2.2 Time taken

The time taken for creating the entire Docker system was about 3 days. However, once the steps have been provided

Figure 9: 503 Service Unavailable from HaProxy

with the specific commands and the files have been provided, the reconstruction of the system took about 20 minutes. The time taken for the load balancer to render the page was quite quick - about 10 milliseconds - since the docker network is being used and the proxy container is local itself.

All three systems on the cloud were pretty fast in redirecting requests to the backend systems, accounting to justa. few milliseconds. However, AWS Classic Load Balancer is definitely the fastest in implementation and is very clearly documented as it can be launched using GUI. It is considerably slower to implement Round Robin in Nginx and HAProxy as the config file needs to be manually edited.

5.2.3 Cost

AWS Classic Load Balancer costs $0.025 per hour [3] while both Nginx and HaProxy are open-source and are thus, free of cost. However, standard costs for the EC2 server used to run Nginx and HaProxy will be applicable. We did not incur any cost as we used free-tier instances.

5.2.4 Performance

All three load balancers perform considerable well. So far, we haven't encountered any issues were some requests were not redirected correctly or were dropped. We have tested using multiple browsers and multiple sessions and found that Round Robin worked efficiently in case of all the three systems.

6. DISCUSSION

Round Robin approach is an easy to implement and useful approach. All the resources and the links in the network will be used equally, thus, making sure that there won't be any resource crunch. However, there are certain limitations to this approach. The load balancer does not stop to see whether any of the server is under utilized. The number of packets being dropped in this approach is comparatively more than in the other approaches.

In order to overcome these limitations, one popular approach is the Weighted Fair Queuing Load Balancing. In this approach, each link is assigned a particular weight based on several factors such as the capacity of the server, the number of requests it is processing, etc. The requests are sent in the order of the weight of each link. This ensures that each server is utilized in a fair manner.

7. CONCLUSION & FUTURE WORK

We have successfully implemented and compared different types of Load Balancers that make use of Round Robin approach on AWS Cloud. Based on several tests and a close observation, we can conclude that there is no one particular approach that would suite every use-case. While AWS Load Balancing is easy to implement and use, it proves to be more expensive than the other open-source approaches

such as Nginx and HaProxy. However, these approaches need manual changes in the config files which requires a little technical knowledge. Thus, it is not as easy to implement as AWS Load Balancer.

In the future, we would like to implement HAProxy in containers as well to perform better comparative analysis. Also, we would like to experiment with other AWS Load Balancing techniques in order to study the technique of load balancing on cloud even further. We would also like to look into a Weighted Round Robin approach which should be able to overcome some of the limitations discussed.

8. REFERENCES

[1] Aws classic load balancer. `https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/introduction.html`. Accessed: 2020-09-25.

[2] Aws load balancers. `https://aws.amazon.com/elasticloadbalancing/?elb-whats-new.sort-by=item.additionalFields.postDateTime&elb-whats-new.sort-order=desc`. Accessed: 2020-11-18.

[3] Cost of aws load balancers. `https://aws.amazon.com/elasticloadbalancing/pricing/`. Accessed: 2020-11-18.

[4] Docker. `https://en.wikipedia.org/wiki/Docker_(software)`. Accessed: 2020-10-23.

[5] Haproxy. `http://www.haproxy.org`. Accessed: 2020-09-25.

[6] Nginx. `https://www.nginx.com`. Accessed: 2020-10-23.

[7] Security groups. `https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html`. Accessed: 2020-11-18.

[8] Yum. `https://en.wikipedia.org/wiki/Yum_(software)`. Accessed: 2020-11-18.

[9] S. Kaur, K. Kumar, J. Singh, and Navtej Singh Ghumman. Round-robin based load balancing in software defined networking. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 2136–2139, 2015.