LOG8415
Advanced Concepts of Cloud Computing

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

Lab 1

Fall 2023

By:
Samy Cheklat
Ming Xiao Yuan
Cassy Charles

Presented to:
Vahid Majdinasab

Saturday, October 7th 2023

1 Introduction

Amazon Web Services (AWS) stands out as a leading provider of Infrastructure as a Service (IaaS) in the cloud computing domain. Among their offerings, Amazon Elastic Compute Cloud (Amazon EC2) is a web-based service that empowers users to execute application programs within the AWS public cloud infrastructure. This service enables developers to quickly provision virtual machines (VMs) across AWS data centers globally, delivering essential compute capacity for various IT projects and cloud workloads.
The primary objectives of this laboratory assignment encompass the creation of two clusters, each consisting of five EC2 instances, and the implementation of an Application Load Balancer (ALB) for each cluster. Within each individual EC2 instance, a Flask application is to be deployed, and Docker will be employed to conduct test scenarios locally. The ensuing report will comprehensively detail the experimental procedure, including the deployment of the Flask Application on each EC2 instance, the configuration of the cluster setup with ALB, the presentation of benchmark results, and a conclusive description of how to execute the script.

2 Deployment of Flask application

Flask, a compact Python web framework, offers valuable tools and features that simplify the development of web applications in Python. In the process of deploying our Flask application using Docker, we followed a systematic procedure to ensure a smooth and efficient deployment. We used a Dockerfile. Initially, we ensured that our Flask application and its dependencies were enumerated in a requirements.txt file with their respective versions. With the "pip install" command, it's possible to fetch these dependencies and install them.
Upon the successful creation of our EC2 virtual machines, where we granted traffic access on port 80 within the security group, we proceeded to connect to the instance for deploying Flask. Upon launching our Docker image on the instance, a command facilitates the automatic execution of our app.py program, enabling its exposure on port 5000.

3 Cluster setup using Application Load Balancer

3.1 Terraform file

A load balancer acts as the sole contact point for clients, evenly distributing incoming application traffic among various targets, such as EC2 instances, located across multiple Availability Zones. This enhances the overall availability of your application.
An Application Load Balancer (ALB) executes routing decisions at the application layer, specifically dealing with HTTP/HTTPS protocols at layer 7 of the OSI model. ALBs support intricate path-based routing, enabling the distribution of requests to one or more ports on each container instance within your cluster. Dynamic host port mapping is a notable feature, allowing traffic redirection based on request content. ALB creation involves three core elements: the Listener, the Target group, and the Rules.
The Listener, functioning as an arbiter for connection requests, scrutinizes clients utilizing configured protocols and ports. The Target group constitutes instances adept at handling incoming

requests. Rules, intrinsic to a listener, govern the load balancer's approach to routing requests. Each rule, characterized by priority, multiple actions, and conditions, triggers its actions when met. A mandatory default rule for each listener is stipulated, with the option for supplementary rules.
Target groups orchestrate the routing of requests to one or more enlisted targets, such as EC2 instances, via designated protocols and port numbers. Notably, a target can be affiliated with multiple target groups, introducing a layer of complexity. Configuration of health checks, executed on all targets aligned with a specified target group within a listener rule, is customizable on a per-target group basis.
Terraform was utilized to construct the AWS infrastructure. In our Terraform script, "resource" blocks were employed to create a total of nine instances. Specifically, we designated five of these instances as m4.large using the instance type, while the remaining four were specified as t2.large. The allocation of instances across two distinct physical regions, namely us-east-1c and us-east-1d, was achieved through the utilization of the availability zone attribute

```
resource "aws_instance" "instances_m4" {
  ami                    = "ami-03a6eaae9938c858c"
  instance_type          = "m4.large"
  key_name               = var.key_pair_name_m4
  vpc_security_group_ids = [aws_security_group.security_gp.id]
  availability_zone      = "us-east-1c"
  user_data              = file("./user_data.sh")
  count = 5
  tags = {
    Name = "M4"
  }
}
```

Figure 1: Setup for our instances of type m4.large

```
resource "aws_instance" "instances_t2" {
  ami                    = "ami-03a6eaae9938c858c"
  instance_type          = "t2.large"
  key_name               = var.key_pair_name_t2
  vpc_security_group_ids = [aws_security_group.security_gp.id]
  availability_zone      = "us-east-1d"
  user_data              = file("./user_data.sh")
  count = 4
  tags = {
    Name = "T2"
  }
```

Figure 2: Setup for our instances of type t2.large

The creation of the Application Load Balancer (ALB) is also accomplished using the Terraform "resource" block. We refer to the ID of the previously generated security group and define the subnets to which the ALB should be associated. To implement this, we dynamically retrieve all the subnets within our VPC through a "data" block.
Following this sequence, we proceed to set up two distinct target groups using a "resource" block—one dedicated to our T4 instances and another to our M4 instances. Additionally, we specify port 80 during this step. Consequently, when a request reaches the load balancer, it is directed to port 80 on a specific instance.
After configuring our ALB and creating the target groups, the subsequent step entails the establishment of our ALB listener resource. This listener is intricately connected to the previously

```
resource "aws_alb_target_group_attachment" "M4_attachments" {
  count            = length(aws_instance.instances_m4)
  target_group_arn = aws_alb_target_group.M4.arn
  target_id        = aws_instance.instances_m4[count.index].id
  port             = 80
}

resource "aws_alb_target_group_attachment" "T2_attachments" {
  count            = length(aws_instance.instances_t2)
  target_group_arn = aws_alb_target_group.T2.arn
  target_id        = aws_instance.instances_t2[count.index].id
  port             = 80
}
```

Figure 3: Attachments of the M4 and t2 target groups

established ALB resource. Initially, this listener inherently directs incoming requests received by the ALB to port 80 of the M4 instances. Nevertheless, we proceed to generate two resources that function as rules—one tailored for the M4 instances and another for the T2 instances.
The M4 rule guides the ALB to direct requests arriving on routes to the M4 target group, while the T2 rule instructs the ALB to route requests received on routes to the T2 target group.
Subsequent to this, we initiate the creation of target group attachments, enabling us to associate each of our five M4 instances and four T2 instances with their respective target groups through unique identifiers known as ARNs.

4 Benchmark results

Two threads (cluster_1 and cluster_2) are created to simulate concurrent generation of HTTP requests to different clusters. cluster_1 generates 1000 requests to Cluster 1. cluster_2 generates 500 initial requests to Cluster 2, sleeps for 60 seconds, and then generates an additional 1000 requests. The script assumes the existence of a load balancer accessible through the provided URL. Requests are sent to specific endpoints (/cluster1 and /cluster2) indicating different clusters through the load balancer. The intentional sleep in cluster_2 introduces a change in load over time, allowing observation of the system's ability to adapt to varying workloads.

	Cluster 1	Cluster 2
Instance Type	m4.large	t2.large
Security group	Protocol on port 80	Protocol on port 80

Table 1: Specifications of the two clusters

5 Instructions to run the code

Prerequisite:

- Ensure that the source code is prepared on the local machine.
- Ensure that Docker is installed on the local machine.
- Ensure that terraform is installed on the local machine

To execute the code, launch a bash command using the following command: bash run.sh