Load Balancer From Scratch

Vahid Majdinasab

2024

Abstract

The aim of this assignment is to familiarize yourself with a cloud computing provider's services and the deployment of your own services. You will be introduced to the concept of Infrastructure as Code and use APIs to create, and deploy a cluster of virtual machines. You will then write and deploy your very own load-balancer, and use it to manage your cluster. Finally, you will use the metrics provided by the cloud provider to analyze the performance of your cluster.

1 Introduction

Amazon Web Services (AWS) is a leading infrastructure as a service cloud provider. Amazon Elastic Compute Cloud (Amazon EC2) is a web-based service that allows businesses to run application programs in the Amazon Web Services (AWS) public cloud. Amazon EC2 allows a developer to spin up virtual machines (VMs), which provide compute capacity for IT projects and cloud workloads that run with global AWS data centers. An AWS user can increase or decrease instance capacity as needed within minutes using the Amazon EC2 web interface or an application programming interface (API). A developer can also define an auto-scaling policy and group to manage multiple instances at once.

During the first laboratory session, the Lab instructor will present an introduction to Cloud Computing and AWS EC2. He will provide the necessary guidelines to create instances, how to set them up how to use SSH to connect to them. Once you get familiar with the Cloud provider, you should create a cluster over instances and start benchmarking them with some scenarios.

It is important to stop EC2 instances when you are not working on them. You will have 50 CAD credits provided by AWS for three assignments of this course. You should manage your expenses carefully. The overall goals of this lab assignment are:

- Create clusters using EC2 and Elastic Load Balancer (ELB).
- Benchmarking clusters to compare their performances.
- Develop your own load balancer.
- Automate your solution.

- Report your findings in a handover documentation format.
- Show a demo of your work.

2 What is an EC2?

An Amazon EC2 (Elastic Compute Cloud) instance is a virtual server in Amazon's cloud computing platform. It provides scalable computing capacity, allowing users to run applications and services without the need for physical hardware. EC2 instances come in various types optimized for different use cases, such as compute-intensive, memory-intensive, or storage-optimized workloads. Users can quickly launch, configure, and terminate these instances as needed, paying only for the compute time used. This flexibility makes EC2 a cornerstone of cloud infrastructure, enabling businesses to efficiently manage their computing resources and adapt to changing demands.

3 What is a Load Balancer

Elastic Load Balancing (ELB) is a load-balancing service for AWS deployments. ELB automatically distributes incoming application traffic and scales resources to meet traffic demands. ELB helps an IT team adjust capacity according to incoming application and network traffic.

- An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports on each container instance in your cluster. Application Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance. When the task is launched, the NGINX container is registered with the Application Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance.
- A Network Load Balancer makes routing decisions at the transport layer (TCP/SSL). It can handle millions of requests per second. After the load balancer receives a connection, it selects a target from the target group for the default rule using a flow hash routing algorithm. It attempts to open a TCP connection to the selected target on the port specified in the listener configuration. It forwards the request without modifying the headers. Network Load Balancers support dynamic host port mapping. For example, if your task's container definition specifies port 80 for an NGINX container port, and port 0 for the host port, then the host port is dynamically chosen from the ephemeral port range of the container instance. When the task is launched, the NGINX container is registered with the Network Load Balancer as an instance ID and port combination, and traffic is distributed to the instance ID and port corresponding

to that container. This dynamic mapping allows you to have multiple tasks from a single service on the same container instance.

- A Classic Load Balancer makes routing decisions at either the transport layer (TCP/SSL) or the application layer (HTTP/HTTPS). Classic Load Balancers currently require a fixed relationship between the load balancer port and the container instance port. For example, it is possible to map the load balancer port 80 to the container instance port 3030 and the load balancer port 4040 to the container in- stance port 4040. However, it is not possible to map the load balancer port 80 to port 3030 on one container instance and port 4040 on another container instance. This static mapping requires that your cluster has at least as many container instances as the desired count of a single service that uses a Classic Load Balancer. The Classic ELB operates at Layer 4. Layer 4 represents the transport layer and is controlled by the protocol being used to transmit the request. For web applications, this will most commonly be the TCP/IP protocol, although UDP may also be used. A network device, of which the Classic ELB is an example, reads the protocol and port of the incoming request, and then routes it to one or more back-end servers. In contrast, the ALB operates at Layer 7. Layer 7 represents the application layer, and as such allows for the redirection of traffic based on the content of the request. Whereas a request to a specific URL backed by a Classic ELB would only enable routing to a particular pool of homogeneous servers, the ALB can route based on the content of the URL, and direct to a specific subgroup of backing servers existing in a heterogeneous collection registered with the load balancer.

For this assignment, you will focus on creating a simple application load balancer. Given a target group, your load balancer will periodically check the load each instance is in, by sending requests and measuring the response time. Based on the results, it will then keep track of which instance is more responsive and redirects the requests to the more responsive instance.

4 AWS Cloud Watch

Elastic Load Balancing publishes data points to Amazon CloudWatch for your load balancers and your targets. CloudWatch enables you to retrieve statistics about those data points as an ordered set of time-series data, known as metrics. Think of a metric as a variable to monitor and the data points as the values of that variable over time. For example, you can monitor the total number of healthy targets for a load balancer over a specified time period. Each data point has an associated time stamp and an optional unit of measurement. You can use metrics to verify that your system is performing as expected. For example, you can create a CloudWatch alarm to monitor a specified metric and initiate an action (such as sending a notification to an email address) if the metric goes outside what you consider an acceptable range.
Take a look at some of the metrics you can use: AWS CloudWatch

5 Benchmarking

5.1 FastAPI App

In this assignment, you are required to deploy a simple FastAPI application in all your EC2 instances. For more information about FastAPI, feel free to check this URL: FastAPI
Your EC2 instances should be identified using a unique ID. For example, instance 1 is required to return a simple message showing: `Instance number 1 is responding now!`. We tend to keep the implementation of the web services as simple as possible.

5.2 Load Balancer Components

There are two ways to implement your application load balancer.

- Your load balancer can send requests to the EC2 instances that it is responsible for, measure the response time for each instance, and keep track of which instance is more responsive. It then redirects the requests to the target instance until another instance shows more responsiveness.
- You can use CloudWatch metrics to monitor the performance of each instance. You are free to choose whichever metric you believe is more suitable. Once the chosen metrics (e.g. CPU utilization) of an instance go over a set limit, your load balancer can redirect new requests to the other instances in your target group.

6 Setup

Before you begin:

- Setup 9 EC2 instance. 5 `t2.micro` and 4 `t2.large` instances.
- Deploy your FastAPI app on each instance.

6.1 Clustering and ELB Setup

The aim is to create two separate clusters, one using t2.large instances and another one using t2.micro instances. Then we would like to send some requests to each cluster and measure the performance of the responses using CloudWatch metrics. Ensure that your instances in each cluster listen to `/cluster1` and `/cluster2` routes respectively and return the instance ID, which is going to be unique for each EC2 instance. This means that if we send the traffic to the /cluster1 endpoint, only the first cluster with t2.large EC2 instances will respond, and for sending requests to the /cluster2 endpoint, we expect to see IDs from the second cluster formed using t2.large instances.
You have to create two target groups to route traffic to the target cluster1. Follow the steps in this document to create an Application Load Balancer (ELB): create an application load balancer.

In our setup, we need to make sure that we have two rules to target each EC2 cluster.

6.2 Deploying your FastAPI app on each instance

- On each instance, first make sure hat your packages are updated:

```
sudo yum update -y # for Amazon Linux 2
sudo apt-get update && sudo apt-get upgrade -y # for Ubuntu
```

Listing 1: Update commands for different Linux distributions

- Install Python and pip:

```
sudo yum install python3 python3-pip -y    # for Amazon Linux 2
sudo apt-get install python3 python3-pip -y    # for Ubuntu
```

Listing 2: Installing Python and Pip for different Linux distributions

- Transfer your python app to your EC2 instance:

```
scp -i /path/to/private_key /path/to/local/file ec2-user@<
   instance_public_ip>:/path/to/destination
```

Listing 3: Transferring python files to the EC2 instance

  The command above uses `scp (aka, secure copy)` to transfer local files to your aws instance. It has multiple arguments:

  – path to private key: which is the local path to your key-pair.
  – path to local file: full (or relative) path to the python file you wish to transfer to your instance.
  – ec2-user: the username of the instance you are running. **If you are running an Ubuntu instance, it'll be** `ubuntu`.
  – instance public-ip: the public IP of your ec2 instance.
  – path to destination: the full path of the desired destination on your ec2 instance.

- The FastAPI script will be available on Moodle and you can use it as a template.

- Install the required dependencies:

```
pip3 install fastapi uvicorn
```

Listing 4: Install the packages

- Run the FastAPI app:

```
uvicorn main:app --host 0.0.0.0 --port 8000
```

Listing 5: Run the FastAPI app

6.3 Benchmarking

You should send one thousand simultaneous requests to each target group and measure their response times. You can use the following `Python` script:

```python
import asyncio
import aiohttp
import time

async def call_endpoint_http(session, request_num):
    url = "your load balancer url"
    headers = {’content-type’: ’application/json’}
    try:
        async with session.get(url, headers=headers) as response:
            status_code = response.status
            response_json = await response.json()
            print(f"Request {request_num}: Status Code: {status_code}")
            return status_code, response_json
    except Exception as e:
        print(f"Request {request_num}: Failed - {str(e)}")
        return None, str(e)

async def main():
    num_requests = 1000
    start_time = time.time()

    async with aiohttp.ClientSession() as session:
        tasks = [call_endpoint_http(session, i) for i in range(num_requests)]
        await asyncio.gather(*tasks)

    end_time = time.time()
    print(f"\nTotal time taken: {end_time - start_time:.2f} seconds")
    print(f"Average time per request: {(end_time - start_time) / num_requests:.4f} seconds")

if __name__ == "__main__":
    asyncio.run(main())
```

Listing 6: Benchmarking

7 Automation and Infrastructure as Code

Your solution should be end-to-end automated. This means that during your demo, the creation of the instances, the load balancer, the target groups, transferring and running the fast API app, and running the benchmarks **SHOULD** be done by running a simple bash script. As such, you will need to develop your solutions using AWS' SDKs depending on the programming language that you prefer: AWS SDKs

8 Working in Groups

You should work in groups of two, three, or four.

9 Report

A submission per group is required for this assignment. To present your findings, you must write a lab report on your benchmarks using **LATEX** template. This lab description document is an example of a LATEX document. In your report, answer the following questions:

- FastAPI Deployment Procedure.
- Cluster setup using Application Load Balancer.
- Results of your benchmark.
- Instructions to run your code.

One submission per group is required for this assignment. You must submit only one ZIPPED file named assignment1.zip which includes:

- report.pdf: Contains the name of the team members and all the material of the report. You can design the format and sections at your convenience.
- Your code: This will include all the necessary commands to execute and show the results of your benchmarking. It should have enough comments and descriptions to understand every single section of it.