Polytechnique Montreal
LOG8415: Lab 1
Cluster Benchmarking using EC2 Virtual Machines and Elastic Load Balancer (ELB)

Abstract

In this assignment, first you will learn about a Cloud provider computing service. Then you will create a cluster of virtual machines and perform extensive benchmarking on several instance types in your cluster. The aim of this assignment is to get familiar with Cloud-based solutions and auto scaling based on different metrics. You are asked to launch virtual machines, connect them as a cluster, use load balancer to distribute workloads, and perform analysis over their performance. Finally, you are asked to prepare and submit a report on Moodle. Your report should contain the results of the benchmarking and you need to provide an automated solution to deploy it.

Objectives

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 necessary guidelines to create instances, how to set them up and 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 100 CAD credit 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.
- Automate your solution.
- Report your findings in a handover documentation format
- Show a demo of your work.

Elastic Load Balancer (ELB)

Elastic Load Balancing (ELB) is a load-balancing service for Amazon Web Services (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 al- lows 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 ap- plications, 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.

AWS CloudWatch

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 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.

Look at some of the metrics you can use:

https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-monitoring.html

Benchmarking

Flask Web Service

In this assignment, you are required to deploy Flask application in all your EC2 instances. For more information about Flask, feel free to check this URL:

https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-using-flask-and-python-3

Your EC2 instances should be identified using a unique ID. For example, instance 1 is required to print out a simple message showing:" Instance number 1 is responding now!". We tend to keep the implementation of the web services as simple as possible.

Application Load Balancer components

A load balancer serves as the single point of contact for clients. The load balancer distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones. This increases the availability of your application. You add one or more listeners to your load balancer.

A listener checks for connection requests from clients, using the protocol and port that you configure. The rules you define for a listener determine how the load balancer routes requests to its registered targets. Each rule consists of a priority, one or more actions, and one or more conditions. When the conditions for a rule are met, then its actions are performed. You must define a default rule for each listener, and you can optionally define additional rules.

Each target group routes requests to one or more registered targets, such as EC2 instances, using the protocol and port number that you specify. You can register a target with multiple target groups. You can configure health checks on a per target group basis. Health checks are performed on all targets registered to a target group specified in a listener rule for your load balancer.

Amazon EC2 Auto Scaling — Ensures that you are running your desired number of instances, even if an instance fails, and enables you to automatically increase or decrease the number of instances as the demand on your instances' changes. If you enable Auto Scaling with Elastic Load Balancing, instances that are launched by Auto Scaling are automatically registered with the load balancer, and instances that are terminated by Auto Scaling are automatically de-registered from the load balancer.

Setup

Before you begin:

1. Launch overall 10 EC2 instances (in different Availability Zones) using the types specified below:

Type	Number of Instances
M4.Large	5
T2.Large	4

2. Deploy Flask Application on EC2 Instances: Please follow the instructions below to deploy Flask web app on each EC2 instance: https://linuxhint.com/ultimate-guide-to-install-flask-on-ubuntu/
3. Deploy your Flask apps in port 80 (HTTP).

Clustering and ELB setup

The aim is to create two separate clusters, one using m4.large instances and another one using t2.large instance. Then we would like to send some requests to each cluster and measure the performance of the responses using CloudWatch metrics.

Ensure that your five 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 /cluster1, only the first cluster with m4.large EC2 instances will respond, and for sending requests to /cluster2, we expect to see IDs from the second cluster formed using t2.xlarge instances. You must create two target groups to route traffic to the target cluster.

Follow the steps in this document to create an Application Load Balancer (ELB):

https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html

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

Analysis

You should provide a fair comparison between your results of benchmarking for EC2 instances. Run test scenarios as described below and report the performance using CloudWatch metrics. You can use the code snippet as the application to test the performance of your cluster by sending HTTP GET requests using Python (this is just a sample code):

```
import json
import requests

# def consumeGETRequestSync():
#     client_crt='cert.pem'
#     client_key='key.pem'
#     url=''
#     certserver='cacert.pem'
#     headers={'content-type':'application/json'}
#     r=requests.get(url,verify=certserver,headers=headers,cert=(client_crt,client_key))
#     print(r.status_code)
#     print(r.json())

def consumeGETRequestSync():
    url='http://LoadBalancerOne-1001411125.us-east-1.elb.amazonaws.com/cluster1'
    r=requests.get(url)
    print(r.status_code)
    print(r.json(),end=' status ')

consumeGETRequestSync()
```

We will be using Docker to perform our test scenarios locally. A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized

software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences, for instance between development and staging.

You must create a Docker container that hosts the necessary script to send GET requests. To understand how to create a container, please follow this link:

https://www.docker.com/blog/how-to-dockerize-your-python-applications/

The container runs locally on your laptop and sends two separate threads:

- 1000 GET requests sequentially.
- 500 GET requests, then one minute sleep, followed by 1000 GET requests.
- Using AWS CloudWatch metrics, you are asked to provide results of each, and every available metric related to your work in your final report. Make sure that you organize the results including tables, diagrams, and present them with relevant information.
- Find out about all available metrics here: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-cloudwatch-metrics.html
- **The above scenarios must be automated and launched using a bash script.** It is important that you set up the environment prior to the demo of your work, so that we can only test the benchmarking section.

Paper

A submission per group is required for this assignment. To present your findings, you must write a lab report on your benchmarks.

In your report, answer the following questions:

1. Flask Application Deployment Procedure.
2. Cluster setup using Application Load Balancer.
3. Results of your benchmark.
4. Instructions to run your code.

One submission per group is required for this assignment. You must submit only one ZIPPED file named tp1.zip which includes:

- tp1.pdf. Containing the name of the team members and all the material of the report. You can design the format and sections at your convenience.
- scripts.sh. Which will include all the necessary commands to execute and show results of your benchmarking. It should have enough comments and descriptions to understand every single section of it.