AWS LAB 1

October 7, 2022

Aura Kiiskinen - aura.kiiskinen@polymtl.ca
Cedric Helewaut - cedric.helewaut@polymtl.ca
Nick Reiter - nick.reiter@polymtl.ca
Victor Mesterton - victor.mesterton@polymtl.ca

Abstract

This paper presents the written elaboration of the first assignment of the Advanced Cloud Computing lecture of École Polytechnique de Montréal.

The basic idea consists of creating two clusters, each consisting of five AWS EC2 Ubuntu instances distributed within different time zones in order to provide availability and outage-free communication. Furthermore, it is required to connect these instances with an Elastic Load Balancer (ELB). ELBs support the adjustment of resource capacity by taking incoming application and network traffic into account. Secondly, the two different clusters must be benchmarked against each other with the help of test scenarios. These test scenarios are evaluated by Amazon's CloudWatch and the creation of a Flask server which is publishing the data. Lastly, it is aimed to automate the previous steps and to build the set-up with a short instruction-based command.

1 Introduction

The assignment was to create a single script that would setup and run jobs on Amazon Web Service (AWS) EC2 instances automatically. This was done using python and AWS Boto3 library. Boto3 has it own commands and functions and is used to forward commands to the AWS CLI, which then setups our instances on AWS learner lab.
In the assignment were we also supposed to install Flask on the instances, which was done using a shell script. Then the instances were assigned to two target groups, which work as clusters. These target groups should then be connected to a Application Load Balancer, which would assign and balance incoming `http` requests. The Performance of the clusters and instances were then supposed to be analysed using Cloud Watch.
The following sections of the documentation will explain each step in more detail. Firstly, the deployment and configuration of Flask on Ubuntu instances is given. Followed by the setup of an Application Load Balancer architecture. Third, the results of the CloudWatch benchmarking metrics are evaluated. Lastly, the point by point instruction for running the code is explained. At the moments is the automated script only possible to run on either MacOS or Linux based systems. We have not created support for Windows machines at the moment.

2 Flask Application Deployment Procedure

To be able to install Flask on an EC2 Ubuntu instance, it is required to make use of scripting language commands that come in the form of a Unix shell (see Listing 1).
TODO: Description of First to lines
The first Unix command is updating all existing packages of the instance. By default, the Ubuntu machines are not equipped with Python. Thus, the Ubuntu distribution of Python including its most eminent packages must be installed. Also, it is required to install the virtual environment package of Python to be able to create a separate, independent developing environment. Afterwards, in the root of the instance, a project directory for the Flask application is created and navigated towards. Thereupon, a virtual environment is compiled and activated. Flask is not comprised within the default stack of Python packages and, hence, must be installed manually. To install a Flask server, a basic python script is coded that returns *Hello World from "IP address of the instance"*. Note, the Flask server is hosted on the IP address `0.0.0.0` and Port `8080`. At last, the Python script is executed.

Listing 1: Flask Deployment on an EC2 Ubuntu instance

```
ssh -o "StrictHostKeyChecking no" $1
ssh -i labsuser.pem ubuntu@$1 << REALEND
# ---Do the necessary installations for flask---
sudo apt-get update
# Install python, pip, flask, nginx and gunicorn3
sudo apt-get install python3
# sudo apt-get install python3 -V
yes | sudo apt-get install python3-venv
mkdir flask_app && cd flask_app
python3 -m venv venv
source venv/bin/activate
pip install flask
python -m flask --version

# Make flask application - it is opened automatically
cat <<EOF >flask_app.py
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return 'Hello, World from ' + str($1)
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
EOF

# To execute above code
# export flask_app=flask_app.py
# flask run
python3 flask_app.pyecho yes |
REALEND
```

3 Cluster setup using Application Load Balancer

In this project report, a bottom up approach is used to create the architecture in figure 1. First and foremost, Ubuntu instances are created. These instances are assigned to a target group, which is subsequently linked to an Application Load Balancer.
To start, a security group is created that controls the traffic allowing to reach and leave the latter associated instances. In the given use case, a TCP connection with the following parameter is allowed TODO.
Furthermore, all availability zones, which are accessible from the current location, are read. The two previously listed components, security groups and availability zones are, thereupon, used to create five Ubuntu `T2.large` and four `M4.large` instances, which are equally distributed through the three AWS availability zones `us-east-1a, us-east-1b, us-east-1c`. The instances are used to create two clusters by assigning them to target groups, one for the T2 and one for the M4 instances. The target groups route requests to one or more registered targets. In the use case, the target groups are initiated with TCP access on port 80.
Application Load Balancers automatically distribute the incoming request across the target groups and availability zones. First, the load balancer is compiled and then assigned to the three subnets `us-east-1a, us-east-1b, us-east-1c`. Second, a listener of the load balancer is instantiated to assign the target groups to the load balancer, to route requests depending on rules. In the prevalent case, one listener working on port 80 is forwarding HTTP queries to the two target groups.
Lastly, Flask is installed with the already explained Unix shell file is carried out on the nine instances in a parallel manner.

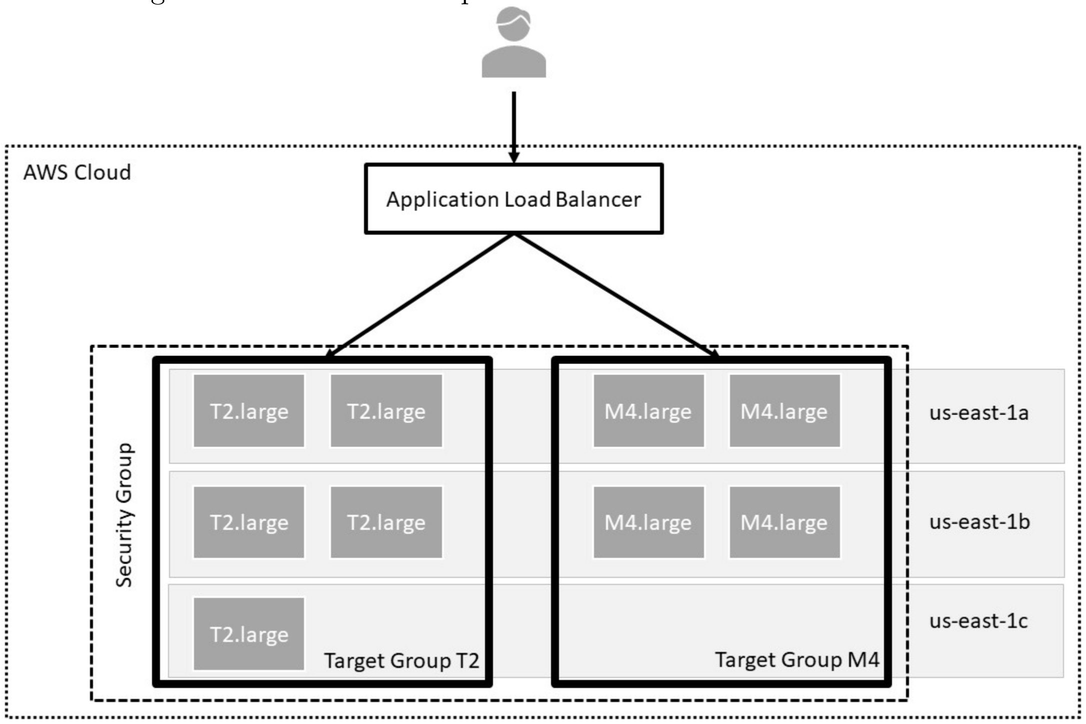
4 Results of benchmark

5 Instructions to run code

This section provides a step-by-step instruction to run the given implementation with ease.

1. Download the AWS Command Line Interface and install it
2. Learner Lap ≫ AWS Details ≫ AWS CLI Show
3. Execute `aws configure` in the command line
4. Copy and paste the suiting credentials from step 2 and *region=us-east-1; output=json*
5. Unzip the compressed project file
6. Navigate with your command line to the location where you stored the project folder
7. Run the Python script with `python3 lab1_script.py` in your command line

Figure 1: Theoretical Setup of the Cloud Architecture


AWS Cloud
Application Load Balancer
Security Group
T2.large
T2.large
M4.large
M4.large
us-east-1a
T2.large
T2.large
M4.large
M4.large
us-east-1b
T2.large
us-east-1c
Target Group T2
Target Group M4