University of Western Australia
CITS5503 - Cloud Computing
Labs 5-9 Report

Student Name: Tao Hu
Student Id: 23805764

Table of Contents

Introduction

This file contains the instructions I took and the explanations for completing labs 5-9 of the unit CITS5503 - Cloud Computing during 2023 semester 2.

The device I have used to complete all the labs is a M1 chip Macbook air. The virtual machine software used is UTM and the linux OS virtualised is Kali Linux.

Lab 5 - Networking

This lab involves managing CIDR addresses, NAT and setting up ELB in AWS.

[2] Setting up an Application Load Balancer

[1] Create 2 EC2 instances in two different availability zones of a specific region

The first step is to create and run two ec2 instances. Both ec2 instances will have the same settings, but each will be created at a different available zone in the same region. A security group that allows both ssh and http is created and attached to both of the two ec2 instances.

First I will create a python file that creates a security group that allows both ssh and http.

```
import boto3
import json

# security group is part of ec2 commands
ec2 = boto3.client("ec2")

sg_GroupName = "23805764-lab5-sg"

# create security group
csg_response = ec2.create_security_group(
   GroupName=sg_GroupName,
   Description="Security group for lab 5 net working.")

sg_id = csg_response["GroupId"]
print(f"Created security group: id={sg_id}")

# configure security group for ssh
asgi_response = ec2.authorize_security_group_ingress(
   GroupName=sg_GroupName,
   IpProtocol="tcp",
```

```
    FromPort=22,
    ToPort=22,
    CidrIp="0.0.0.0/0"
)

print("configured security group inbound traffic for ssh")
print(json.dumps(asgi_response["SecurityGroupRules"], indent=4))

# configure security group for http
asgi_response = ec2.authorize_security_group_ingress(
    GroupName=sg_GroupName,
    IpProtocol="tcp",
    FromPort=80,
    ToPort=80,
    CidrIp="0.0.0.0/0"
)

print("configured security group inbound traffic for http")
print(json.dumps(asgi_response["SecurityGroupRules"], indent=4))
```

The above is the code of the python file called createSg.py. It creates a security group first, then configures the inbound traffic of the new security group. Traffic from port 22 (SSH) and 80 (HTTP) are allowed.

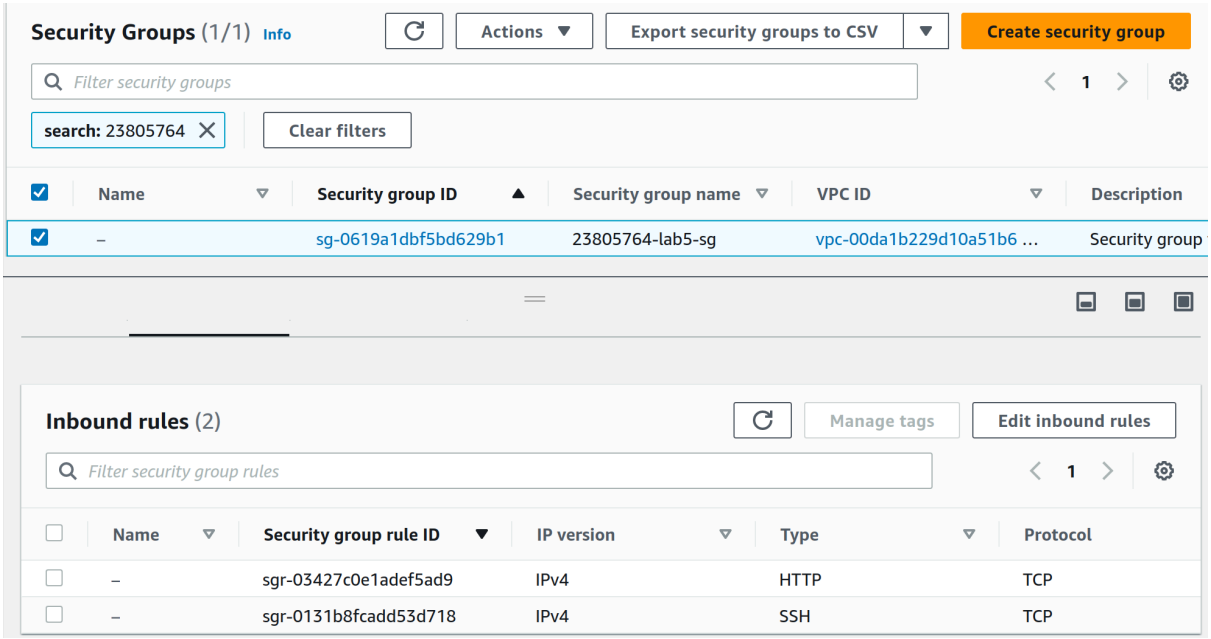
```
python3 createSg.py
```

Run the python file to create and configure the security group.

```
Created security group: id=sg-0619a1dbf5bd629b1
configured security group inbound traffic for ssh
[
      {
      "SecurityGroupRuleId": "sgr-0131b8fcadd53d718",
      "GroupId": "sg-0619a1dbf5bd629b1",
      "GroupOwnerId": "489389878001",
      "IsEgress": false,
      "IpProtocol": "tcp",
      "FromPort": 22,
      "ToPort": 22,
      "CidrIpv4": "0.0.0.0/0"
      }
```

```
]
configured security group inbound traffic for http
[
      {
      "SecurityGroupRuleId": "sgr-03427c0e1adef5ad9",
      "GroupId": "sg-0619a1dbf5bd629b1",
      "GroupOwnerId": "489389878001",
      "IsEgress": false,
      "IpProtocol": "tcp",
      "FromPort": 80,
      "ToPort": 80,
      "CidrIpv4": "0.0.0.0/0"
      }
]
```

The above response shows that all the commands are successful. From the Json output, the information is simply the information about the rules that have been created. The useful information in the output to storei the groupId that is genereated.



The above screenshot taken from the console verifies that my security group is created and has the correct inbound traffic rule configured. The bottom left right of the screenshot shows that HTTP and SHH is a value of the attribute Type.

The next step is to create a security key that can be bound to the ec2 instance when accessing it through ssh.

```
import boto3
import json
import os
```

```python
import stat

# security group is part of ec2 commands
ec2 = boto3.client("ec2")

myKeyName = "23805764-lab5-key"

# create the key pair
ckp_response = ec2.create_key_pair(KeyName=myKeyName)
key_id = ckp_response["KeyPairId"]
print(f"Successfully created key pair: KeyPairId={key_id}")

key_out_file = "23805764-lab5-key.pem"
keyFile = open(key_out_file, "w")
keyFile.write(ckp_response["KeyMaterial"])
print(f"Wrote the key content to file: {key_out_file}")
keyFile.close()

# key file permission can only be read by user
os.chmod(key_out_file, stat.S_IREAD)
print("Changed key file permission")
```

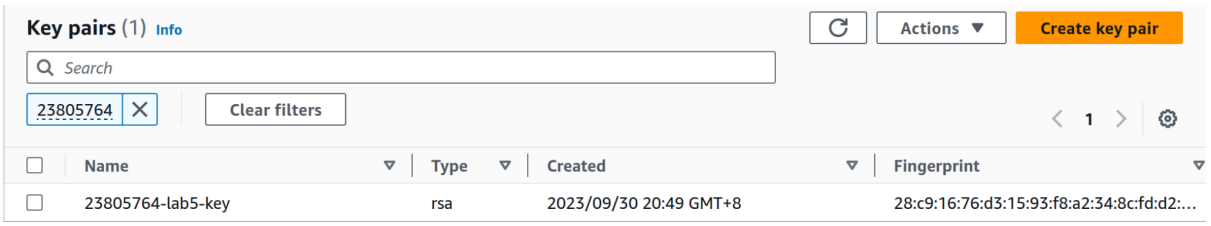
The above is the code of python file createSk.py. A new security key will be created and the key content is stored locally.

```
python3 createSK.py
```

Run the program.

```
Successfully created key pair: KeyPairId=key-06d4877e7547be537
Wrote the key content to file: 23805764-lab5-key.pem
Changed key file permission
```

Above output shows the key is created.



Above screenshot from the console verifies the key exists.

With both the security group and key created, now I would want to create the ec2 instances. Two ec2 instances will be created and use the security group and key from the above. However, the available zone that is set will be different for the two in the same region. The ap-southeast-2 region will be used.

```
aws ec2 describe-availability-zones
```

The above AWS cli command is run in the terminal to find out all the available zones in my current region.


```
{
      "AvailabilityZones": [
      {
            "State": "available",
            "OptInStatus": "opt-in-not-required",
            "Messages": [],
            "RegionName": "ap-southeast-2",
            "ZoneName": "ap-southeast-2a",
            "ZoneId": "apse2-az3",
            "GroupName": "ap-southeast-2",
            "NetworkBorderGroup": "ap-southeast-2",
            "ZoneType": "availability-zone"
      },
      {
            "State": "available",
            "OptInStatus": "opt-in-not-required",
            "Messages": [],
            "RegionName": "ap-southeast-2",
            "ZoneName": "ap-southeast-2b",
            "ZoneId": "apse2-az1",
            "GroupName": "ap-southeast-2",
            "NetworkBorderGroup": "ap-southeast-2",
            "ZoneType": "availability-zone"
      },
      {
            "State": "available",
            "OptInStatus": "opt-in-not-required",
            "Messages": [],
            "RegionName": "ap-southeast-2",
            "ZoneName": "ap-southeast-2c",
            "ZoneId": "apse2-az2",
            "GroupName": "ap-southeast-2",
            "NetworkBorderGroup": "ap-southeast-2",
            "ZoneType": "availability-zone"
      },
```

```
        {
                "State": "available",
                "OptInStatus": "opted-in",
                "Messages": [],
                "RegionName": "ap-southeast-2",
                "ZoneName": "ap-southeast-2-akl-1a",
                "ZoneId": "apse2-akl1-az1",
                "GroupName": "ap-southeast-2-akl-1",
                "NetworkBorderGroup": "ap-southeast-2-akl-1",
                "ZoneType": "local-zone",
                "ParentZoneName": "ap-southeast-2c",
                "ParentZoneId": "apse2-az2"
        },
        {
                "State": "available",
                "OptInStatus": "opted-in",
                "Messages": [],
                "RegionName": "ap-southeast-2",
                "ZoneName": "ap-southeast-2-per-1a",
                "ZoneId": "apse2-per1-az1",
                "GroupName": "ap-southeast-2-per-1",
                "NetworkBorderGroup": "ap-southeast-2-per-1",
                "ZoneType": "local-zone",
                "ParentZoneName": "ap-southeast-2b",
                "ParentZoneId": "apse2-az1"
        }
        ]
}
```


Above is the output show data of the available zones in the same region, which is ap-southeast-2 The top two zones ap-southeast-2a and ap-southeast-2b will be chosen for the two ec2 instances later on to be run in.

Now I need to create a python file that creates two ec2 instances when run.

```
import boto3
import json

# security group is part of ec2 commands
ec2 = boto3.client("ec2")

sg_GroupName = "23805764-lab5-sg"
myKeyName = "23805764-lab5-key"

zone_1 = "ap-southeast-2a"
```

```
zone_2 = "ap-southeast-2b"

rs_response = ec2.run_instances(
   ImageId="ami-d38a4ab1",
   SecurityGroups=[sg_GroupName],
   MinCount=1,
   MaxCount=1,
   InstanceType="t2.micro",
   KeyName=myKeyName,
   Placement={
       "AvailabilityZone": zone_1
   }
)

instance_1_id = rs_response["Instances"][0]["InstanceId"]
print(f"Created ec2 instance: id={instance_1_id}, zone={zone_1}")

instance_1_name = "23805764-lab5-a"
ec2.create_tags(
   Resources=[instance_1_id],
   Tags=[{
       "Key": "Name",
       "Value": instance_1_name
   }]
)
print(f"Assigned tags to instance {instance_1_id}
Name={instance_1_name}")

rs_response = ec2.run_instances(
   ImageId="ami-d38a4ab1",
   SecurityGroups=[sg_GroupName],
   MinCount=1,
   MaxCount=1,
   InstanceType="t2.micro",
   KeyName=myKeyName,
   Placement={
       "AvailabilityZone": zone_2
   }
)

instance_1_id = rs_response["Instances"][0]["InstanceId"]
print(f"Created ec2 instance: id={instance_1_id}, zone={zone_2}")
```

```
instance_2_name = "23805764-lab5-b"
ec2.create_tags(
   Resources=[instance_1_id],
   Tags=[{
        "Key": "Name",
        "Value": instance_2_name
   }]
)
print(f"Assigned tags to instance {instance_1_id}
Name={instance_2_name}")
```

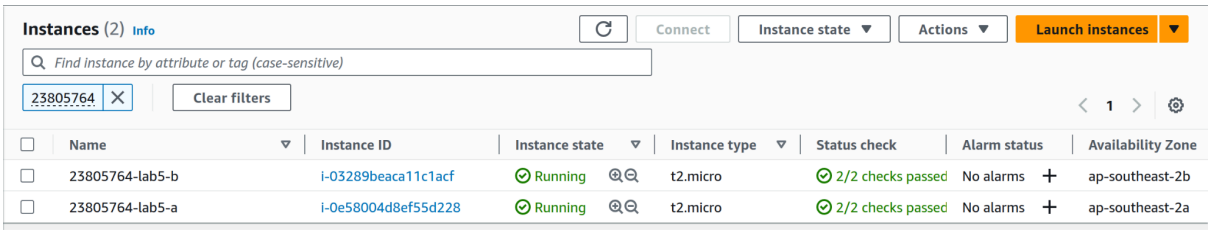
The above is the content of file createEc2.py, two different ec2 instances are created in two different zones.

```
python3 createEc2.py
```

Run the program.

```
Created ec2 instance: id=i-0e58004d8ef55d228, zone=ap-southeast-2a
Assigned tags to instance i-0e58004d8ef55d228 Name=23805764-lab5-a
Created ec2 instance: id=i-03289beaca11c1acf, zone=ap-southeast-2b
Assigned tags to instance i-03289beaca11c1acf Name=23805764-lab5-b
```

The above output shows that I have successfully created two instances with one in zone ap-southeast-2a and one in zone ap-southeast-2b.



Above screenshot verifies that two instances are created in two different zones.

[2] Create the Application Load Balancer

[a] Create the load balancer

A python file is written to use boto3 to create a load balancer.

```
import boto3
```

```
import json

instance_a_id = "i-0e58004d8ef55d228"
instance_b_id = "i-03289beaca11c1acf"
security_group_id = "sg-0619a1dbf5bd629b1"

ec2 = boto3.client("ec2")
elbv2 = boto3.client("elbv2")

di_response = ec2.describe_instances(
   InstanceIds = [
       instance_a_id,
       instance_b_id
   ]
)

instance_a_subnet_id =
di_response["Reservations"][0]["Instances"][0]["NetworkInterfaces"][0][
"SubnetId"]
instance_b_subnet_id =
di_response["Reservations"][1]["Instances"][0]["NetworkInterfaces"][0][
"SubnetId"]

print(f"Queried instance {instance_a_id}, subnet
id={instance_a_subnet_id}")
print(f"Queried instance {instance_b_id}, subnet
id={instance_b_subnet_id}")

load_balancer_name = "23805764-lab5-lb"

# default load balancer is application
# configure the securtiy groups and subnet id
clb_response = elbv2.create_load_balancer(
   Name=load_balancer_name,
   SecurityGroups=[
       security_group_id
   ],
   Subnets=[
       instance_a_subnet_id,
       instance_b_subnet_id
   ]
)
```

```
load_balancer_arn = clb_response["LoadBalancers"][0]["LoadBalancerArn"]

print(f"Created Loadbalancer with name={load_balancer_name},
arn={load_balancer_arn}")
```

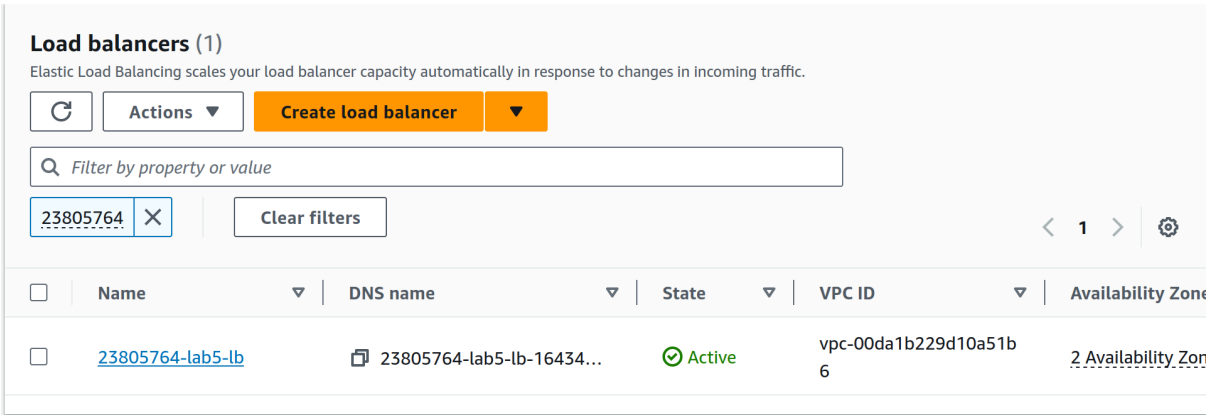
The above is the content of file createElb.py. The security group id and instance ids was saved from the outputs earlier when creating one. The code then uses the ec2 client to retrieve the subnet id of both of the instances. Then a load balancer is created and the arn is outputted.

```
python3 createElb.py
```

Run the program.

```
Queried instance i-0e58004d8ef55d228, subnet id=subnet-0c1878c6a739707b7
Queried instance i-03289beaca11c1acf, subnet id=subnet-0a7d8e51199753df1
Created Loadbalancer with name=23805764-lab5-lb,
arn=arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:loadbalance
r/app/23805764-lab5-lb/1853f3a1f41e1d97
```

Above output shows the arn of the created load balancer.



The above screenshot verifies that the load balancer exists in the AWS console.

[b] Create a target group

The next step is to create a target group using the same VPC id that both of the instances is part of.

```
import boto3
import json

instance_a_id = "i-0e58004d8ef55d228"
instnace_b_id = "i-03289beaca11c1acf"
```

```
ec2 = boto3.client("ec2")
elbv2 = boto3.client("elbv2")

vpc_id = ec2.describe_instances(
   InstanceIds = [
       instance_a_id
   ]
)["Reservations"][0]["Instances"][0]["NetworkInterfaces"][0]["VpcId"]

print(f"Queried instance {instance_a_id}, vpc id={vpc_id}")

target_group_name = "23805764-lab5-tg"

ctg_response = elbv2.create_target_group(
   Name=target_group_name,
   Port=80,
   Protocol="HTTP",
   VpcId=vpc_id
)

target_group_arn = ctg_response["TargetGroups"][0]["TargetGroupArn"]

print(f"Created target group with name={target_group_name},
arn={target_group_arn}")
```

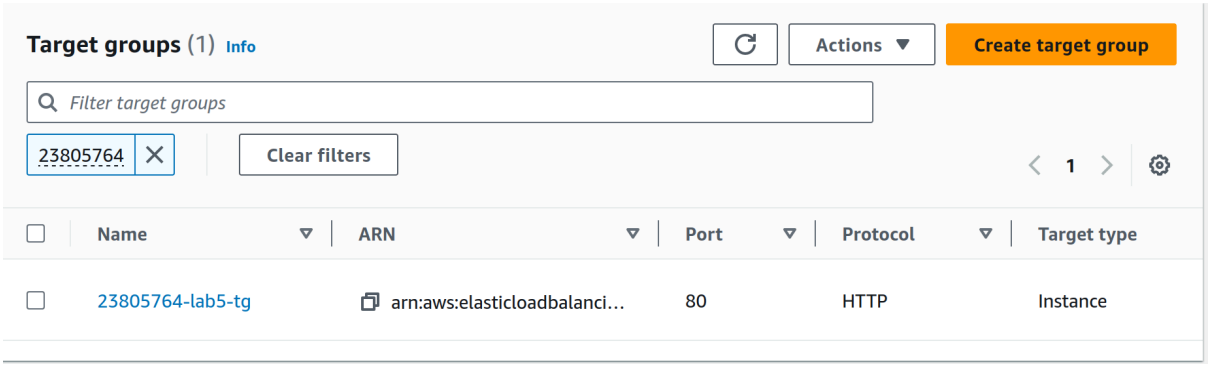
The above is the content of file craeteTg.py. The program first retrieves the VPC id of one of the ec2 instances. Then the VPC id is used to create a target group.

```
python3 createTg.py
```

Run the program.

```
Queried instance i-0e58004d8ef55d228, vpc id=vpc-00da1b229d10a51b6
Created target group with name=23805764-lab5-tg,
arn=arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:targetgroup
/23805764-lab5-tg/05248de6cfd45258
```

The above output shows the arn of the created target group.



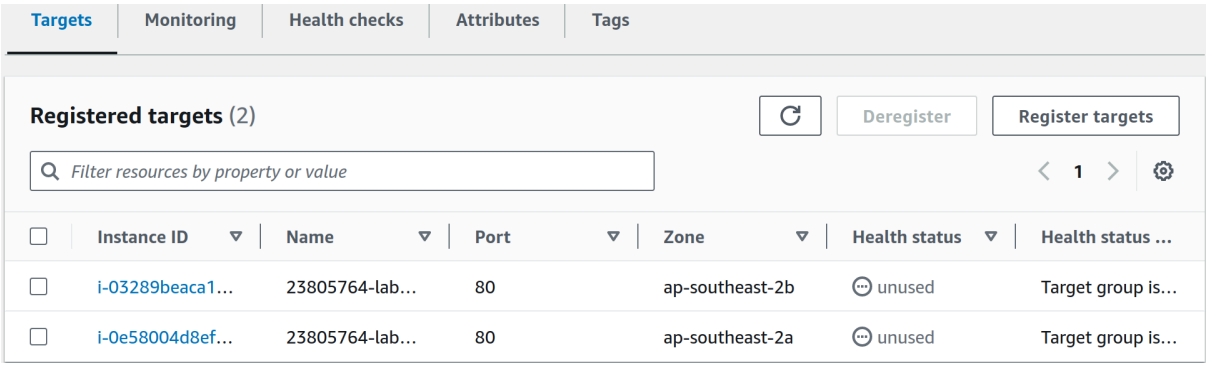
Above screenshot verifies that the target group exists in the AWS console.

[c] Register targets in the target group

When the target group is created it is not registered with any instances. This step will register the two ec2 instances to the target group created.

```
import boto3
import json

instance_a_id = "i-0e58004d8ef55d228"
instance_b_id = "i-03289beaca11c1acf"
target_group_arn="arn:aws:elasticloadbalancing:ap-southeast-2:489389878
001:targetgroup/23805764-lab5-tg/05248de6cfd45258"

elbv2 = boto3.client("elbv2")

rt_response = elbv2.register_targets(
   TargetGroupArn=target_group_arn,
   Targets=[
       {
           "Id": instance_a_id
       },
       {
           "Id": instance_b_id
       }
   ]
)

print(f"Registered targets {instance_a_id} and {instance_b_id} to
target group with arn={target_group_arn}")
```

The above is the content of the file registerT.py. It registers the two ec2 instances to the created target group.



Above screenshot verifies that the two instances are registered as targets under the target group.

[d] Create a listener

Both the target group and load balancer are created and configured correctly, but not linked together for a purpose. This step will attempt to create a listener for the two resources, which will route requests from the load balancer into the target group.

```
import boto3
import json

target_group_arn =
"arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:targetgroup/2
3805764-lab5-tg/05248de6cfd45258"
load_balancer_arn =
"arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:loadbalancer/
app/23805764-lab5-lb/1853f3a1f41e1d97"

elbv2 = boto3.client("elbv2")

cl_response = elbv2.create_listener(
   DefaultActions=[
       {
           "TargetGroupArn": target_group_arn,
           "Type": "forward"
       }
   ],
   LoadBalancerArn=load_balancer_arn,
   Port=80,
   Protocol="HTTP"
)

listener_arn = cl_response["Listeners"][0]["ListenerArn"]
```

```
print(f"Created listener, arn={listener_arn}")
```

The above is the content of file createL.py. It creates a listener that listens to port 80 on the specified load balancer and target group.

```
python3 createL.py
```

Run the program.

```
Created listener,
arn=arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:listener/ap
p/23805764-lab5-lb/1853f3a1f41e1d97/4ca3f4664570496b
```

Above output shows the listener arn.

[e] Install apache2

To allow web access of the two instances, apache2 is to be installed. Both instances will be accessed through ssh using the key created earlier and then apache2 is installed using the apt command.

The Ip of both of the instances can be found in the aws console. It is copied and used when connecting through ssh.

```
ssh -i 23805764-lab5-key.pem ubuntu@54.252.214.62
ssh -i 23805764-lab5-key.pem ubuntu@13.210.229.112
```

The above is run with each command at a different terminal.

```
sudo apt-get update; sudo apt install apache2
```

The above command ran on both terminals to first update apt-get and then install apache2.

```
sudo vim /var/www/html/index.html
```

Edit the index.html file in both terminals and change the heading of the html page.

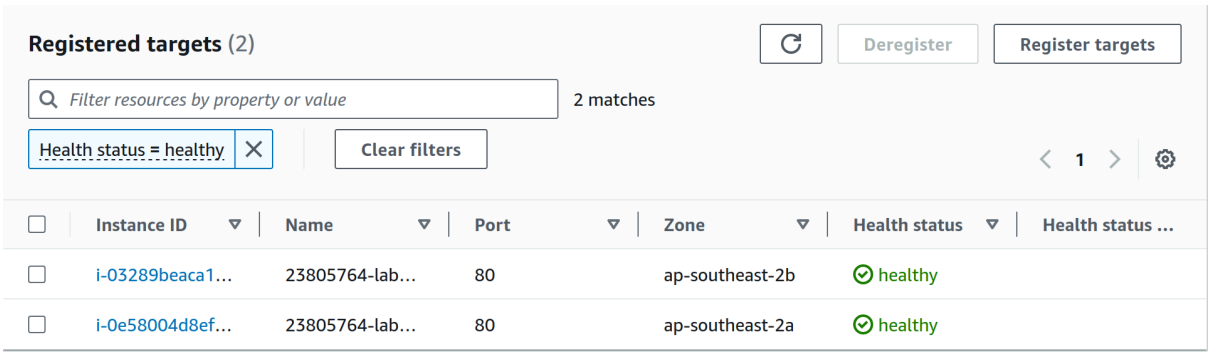
```
      <span class="floating_element">
            This is from the first instance.
      </span>
```

The above is the change in the first terminal.

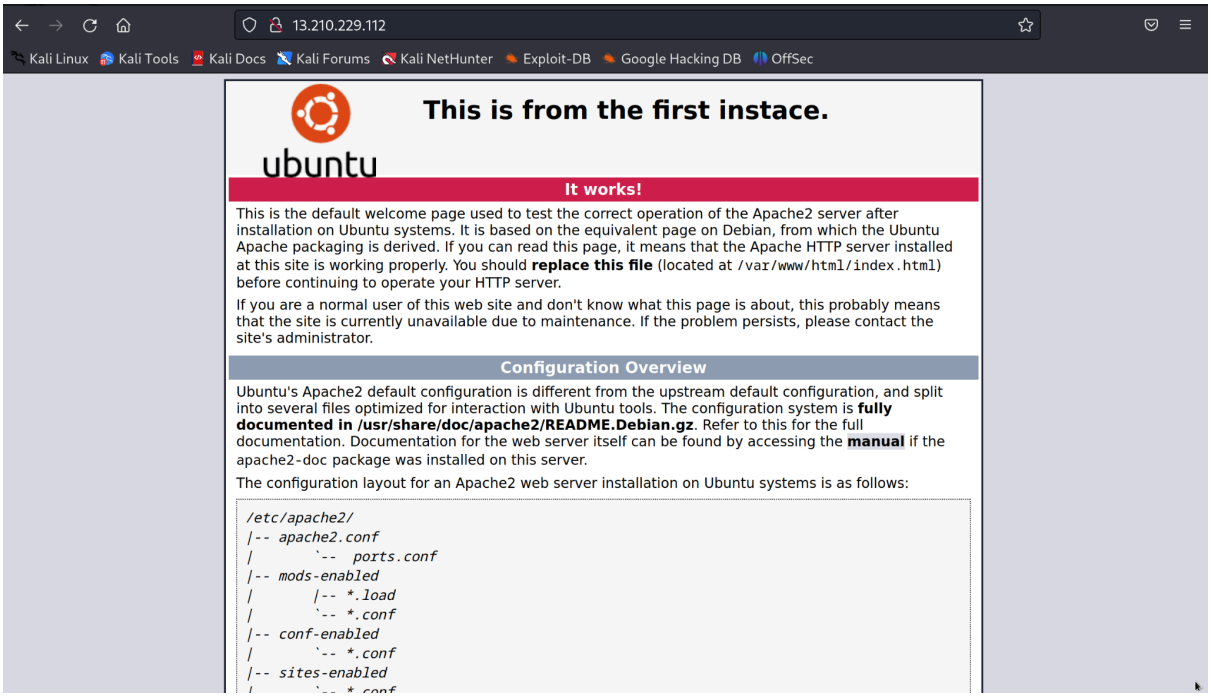
```
<span class="floating_element">
      This is from the second instance.
</span>
```

The above is the change in the second terminal.

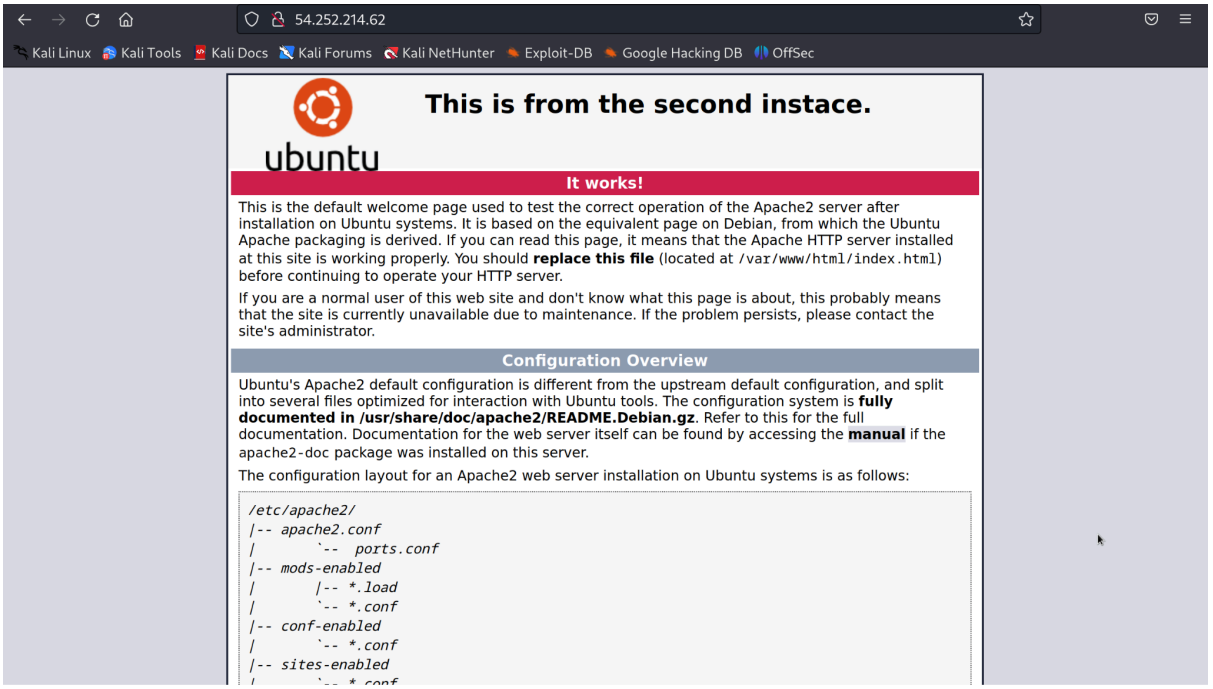
[f] Verify HTTP access instances



Above screenshots shows that the two instances are healthy after installing apache2.



Above screenshot shows the html page received from accessing the first instance's ip address.



Above screenshot shows the html page received from accessing the second instnace's ip address.

Lab 6 - Web Application

This lab involves creating a web app using Django, implementing nginx and load balance requests to it and retrieving data from DynamoDb to display in the app.

[1] Create an Ec2 instance

[1] Create Ec2 micro instance and SSH into it

To create and configure a Ec2 instance, a large portion of the code from lab 5 is copied and altered. After some trial and errors, I figured out that using the ami ami-d38a4ab, which is version 16.04 from lab 1 will not work for this lab. In this lab a higher version of Ubuntu OS is required. From the AWS console, ami-0310483fb2b488153 is the latest Ubuntu 22.04, which will be used.

```
import boto3
import json
import os
import stat

ec2 = boto3.client("ec2")

sg_GroupName = "23805764-lab6-sg"
myKeyName = "23805764-lab6-key"
```

```
# create security group
csg_response = ec2.create_security_group(
   GroupName=sg_GroupName,
   Description="Security group for lab 6 net working.")

sg_id = csg_response["GroupId"]
print(f"Created security group: id={sg_id}")

# configure security group for ssh
asgi_response = ec2.authorize_security_group_ingress(
   GroupName=sg_GroupName,
   IpProtocol="tcp",
   FromPort=22,
   ToPort=22,
   CidrIp="0.0.0.0/0"
)

print("configured security group inbound traffic for ssh")
print(json.dumps(asgi_response["SecurityGroupRules"], indent=4))

# configure security group for http
asgi_response = ec2.authorize_security_group_ingress(
   GroupName=sg_GroupName,
   IpProtocol="tcp",
   FromPort=80,
   ToPort=80,
   CidrIp="0.0.0.0/0"
)


print("configured security group inbound traffic for http")
print(json.dumps(asgi_response["SecurityGroupRules"], indent=4))

# create the key pair
ckp_response = ec2.create_key_pair(KeyName=myKeyName)
key_id = ckp_response["KeyPairId"]
print(f"Successfully created key pair: KeyPairId={key_id}")

key_out_file = "23805764-lab6-key.pem"
keyFile = open(key_out_file, "w")
keyFile.write(ckp_response["KeyMaterial"])
print(f"Wrote the key content to file: {key_out_file}")
keyFile.close()
```

```
# key file permission can only be read by user
os.chmod(key_out_file, stat.S_IREAD)
print("Changed key file permission")


rs_response = ec2.run_instances(
   ImageId="ami-0310483fb2b488153",
   SecurityGroups=[sg_GroupName],
   MinCount=1,
   MaxCount=1,
   InstanceType="t2.micro",
   KeyName=myKeyName
)

instance_id = rs_response["Instances"][0]["InstanceId"]
print(f"Created ec2 instance: id={instance_id}")

instance_name = "23805764-lab6-3-ec2"
ec2.create_tags(
   Resources=[instance_id],
   Tags=[{
       "Key": "Name",
       "Value": instance_name
   }]
)
print(f"Assigned tags to instance {instance_id} Name={instance_name}")
```

The above is the content of file createEc2.py. A security group is created then configured for allowing inbound traffic to the port 20 and 80. Then a key pair is created and stored locally. Finally, a instance is created using the security group and key pair.

```
python3 createEc2.py
```

Run the program.

```
Created security group: id=sg-07a33a2e5db14cc40
configured security group inbound traffic for ssh
[
      {
      "SecurityGroupRuleId": "sgr-0cc3b1397fd42a75f",
```

```
        "GroupId": "sg-07a33a2e5db14cc40",
        "GroupOwnerId": "489389878001",
        "IsEgress": false,
        "IpProtocol": "tcp",
        "FromPort": 22,
        "ToPort": 22,
        "CidrIpv4": "0.0.0.0/0"
        }
]
configured security group inbound traffic for http
[
        {
        "SecurityGroupRuleId": "sgr-046e364d9eb59763d",
        "GroupId": "sg-07a33a2e5db14cc40",
        "GroupOwnerId": "489389878001",
        "IsEgress": false,
        "IpProtocol": "tcp",
        "FromPort": 80,
        "ToPort": 80,
        "CidrIpv4": "0.0.0.0/0"
        }
]
Successfully created key pair: KeyPairId=key-0ea65603182c1d956
Wrote the key content to file: 23805764-lab6-key.pem
Changed key file permission
Created ec2 instance: id=i-018843a24d80f879e
Assigned tags to instance i-018843a24d80f879e Name=23805764-lab6-3-ec2
```

Above is the output showing the ids of different resources.

Now I need to ssh into the ec2 instance and install python.

```
ssh -i 23805764-lab6-key.pem ubuntu@3.27.157.242
```

The above command to SSH into the ec2 instance, ip found in AWS console

```
sudo apt-get update; sudo apt-get upgrade; sudo apt-get install
python3-venv.
```

Run the above command to upgrade and install python3-venv.

[2] Create a directory and set up the virtual environment

```
sudo bash
```

Get root permission in the terminal.

```
mkdir -p /opt/wwc/mysites; cd /opt/wwc/mysites; python3 -m venv myvenv
```

The above command will create a directory in path /opt/wwc/mysites and then set up a virtual environment.

[3] Activate virtual environment

```
source myvenv/bin/.activate
```

Above is called to go to the python virtual environment.

```
pip install django
```

Above to install django

```
Collecting django
  Downloading Django-4.2.5-py3-none-any.whl (8.0 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━ 8.0/8.0 MB 39.7 MB/s eta 0:00:00
Collecting sqlparse>=0.3.1
  Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━ 41.2/41.2 KB 7.0 MB/s eta 0:00:00
Collecting asgiref<4,>=3.6.0
  Downloading asgiref-3.7.2-py3-none-any.whl (24 kB)
Collecting typing-extensions>=4
  Downloading typing_extensions-4.8.0-py3-none-any.whl (31 kB)
Installing collected packages: typing-extensions, sqlparse, asgiref,
django
Successfully installed asgiref-3.7.2 django-4.2.5 sqlparse-0.4.4
typing-extensions-4.8.0
```

Above output from installing django.

```
django-admin startproject lab; cd lab; python3 manage.py startapp polls
```

Above to create a new project and

[2] Install and configure nginx

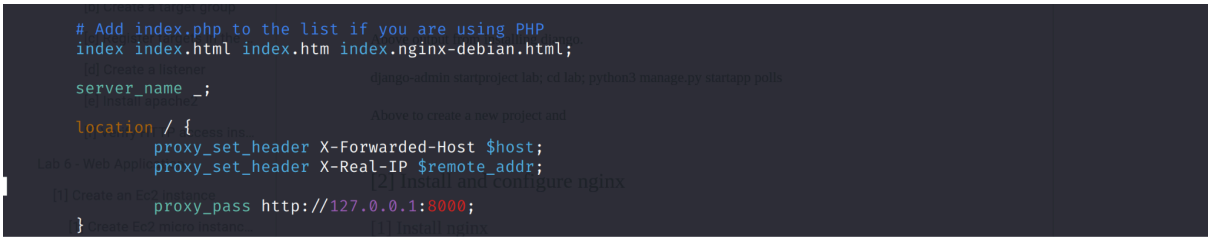
[1] Install nginx

```
apt install nginx
```

Install nginx

```
vim /etc/nginx/sites-enabled/default
```

Use vim to edit the location part.



Above screenshot shows the new location section in the file.

[2] Start nginx

```
service nginx restart
```

To start nginx.

[3] Run server

```
python3 manage.py runserver 8000
```

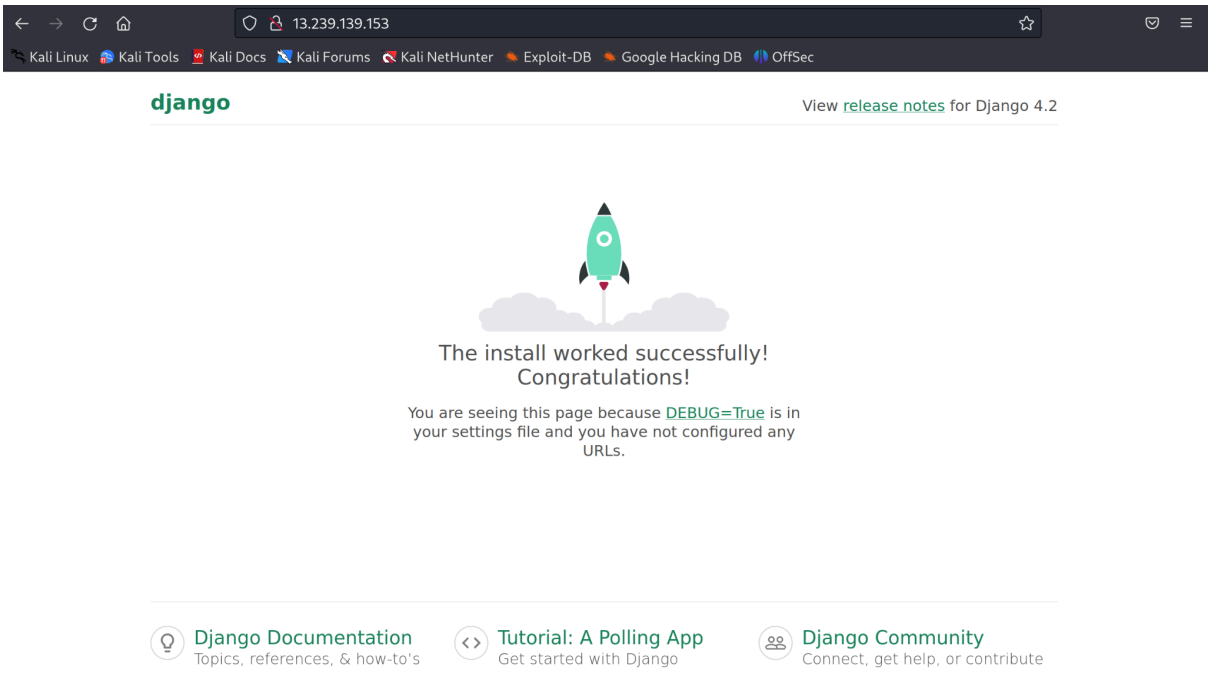
Start the server

```
Watching for file changes with StatReloader
Performing system checks...
```

```
System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly
until you apply the migrations for app(s): admin, auth, contenttypes,
sessions.

Run 'python manage.py migrate' to apply them.
October 01, 2023 - 08:13:15
Django version 4.2.5, using settings 'lab.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[01/Oct/2023 08:13:32] "GET / HTTP/1.0" 200 10664
Not Found: /favicon.ico
[01/Oct/2023 08:13:33] "GET /favicon.ico HTTP/1.0" 404 2107
```

Above is the output.

[4] Verify server is working in browser



Above screenshot shows that the server is indeed accessible.

[3] Change the code

[1] Update files

Since the Ec2 instance is connected using SSH, vim is used to edit the different python files.

vim polls/views.py

The above is called to open the vim editor on the first file views.py that will be edited.

Edit polls/views.py

```
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def index(request):
      return HttpResponse("Hello, world.")
```

Above is the new polls/views.py

```
from django.urls import path
from . import views

urlpatterns = [
      path('', views.index, name='index'),
]
```

Above is the new polls/urls.py

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
      path('admin/', admin.site.urls),
      path('polls/', include('polls.urls')),
]
```

Above is the new lab/urls.py

[2] Run server

```
python3 manage.py runserver 8000
```

Run the server again.

[3] Verify change made

Visit http://13.239.139.153/polls/ to verify the change

Above screenshot shows that changes are made.

[4] Add an application load balancer

[1] Create application load balancer

The ELB created in lab 5 has been deleted, so a new ELB needs to be created.

```
import boto3
import json

instance_id = "i-018843a24d80f879e"
security_group_id = "sg-07a33a2e5db14cc40"

ec2 = boto3.client("ec2")
elbv2 = boto3.client("elbv2")
```

```
# get instnace information
di_response = ec2.describe_instances(
    InstanceIds = [
        instance_id,
    ]
)

instance_subnet_id =
di_response["Reservations"][0]["Instances"][0]["NetworkInterfaces"][0][
"SubnetId"]
instance_vpc_id =
di_response["Reservations"][0]["Instances"][0]["NetworkInterfaces"][0][
"VpcId"]

print(f"Queried instance {instance_id}, subnet id={instance_subnet_id},
vpc id={instance_vpc_id}")

load_balancer_name = "23805764-lab5-lb"

# default load balancer is application
# configure the securtiy groups and subnet id
clb_response = elbv2.create_load_balancer(
    Name=load_balancer_name,
    SecurityGroups=[
        security_group_id
    ],
    Subnets=[
        instance_subnet_id,
        "subnet-0a7d8e51199753df1"
    ]
)

load_balancer_arn = clb_response["LoadBalancers"][0]["LoadBalancerArn"]

print(f"Created Loadbalancer with name={load_balancer_name},
arn={load_balancer_arn}")

# create target group
target_group_name = "23805764-lab6-tg"

# the default healthcheck path is /, here is /polls
ctg_response = elbv2.create_target_group(
    Name=target_group_name,
```

```
    Port=80,
    Protocol="HTTP",
    VpcId=instance_vpc_id,
    HealthCheckPath="/polls"
)

target_group_arn = ctg_response["TargetGroups"][0]["TargetGroupArn"]

print(f"Created target group with name={target_group_name},
arn={target_group_arn}")

# register target
rt_response = elbv2.register_targets(
    TargetGroupArn=target_group_arn,
    Targets=[
        {
            "Id": instance_id
        }
    ]
)

print(f"Registered instnace {instance_id} to target group with
arn={target_group_arn}")

# create listener
cl_response = elbv2.create_listener(
    DefaultActions=[
        {
            "TargetGroupArn": target_group_arn,
            "Type": "forward"
        }
    ],
    LoadBalancerArn=load_balancer_arn,
    Port=80,
    Protocol="HTTP"
)

listener_arn = cl_response["Listeners"][0]["ListenerArn"]

print(f"Created listener, arn={listener_arn}")
```

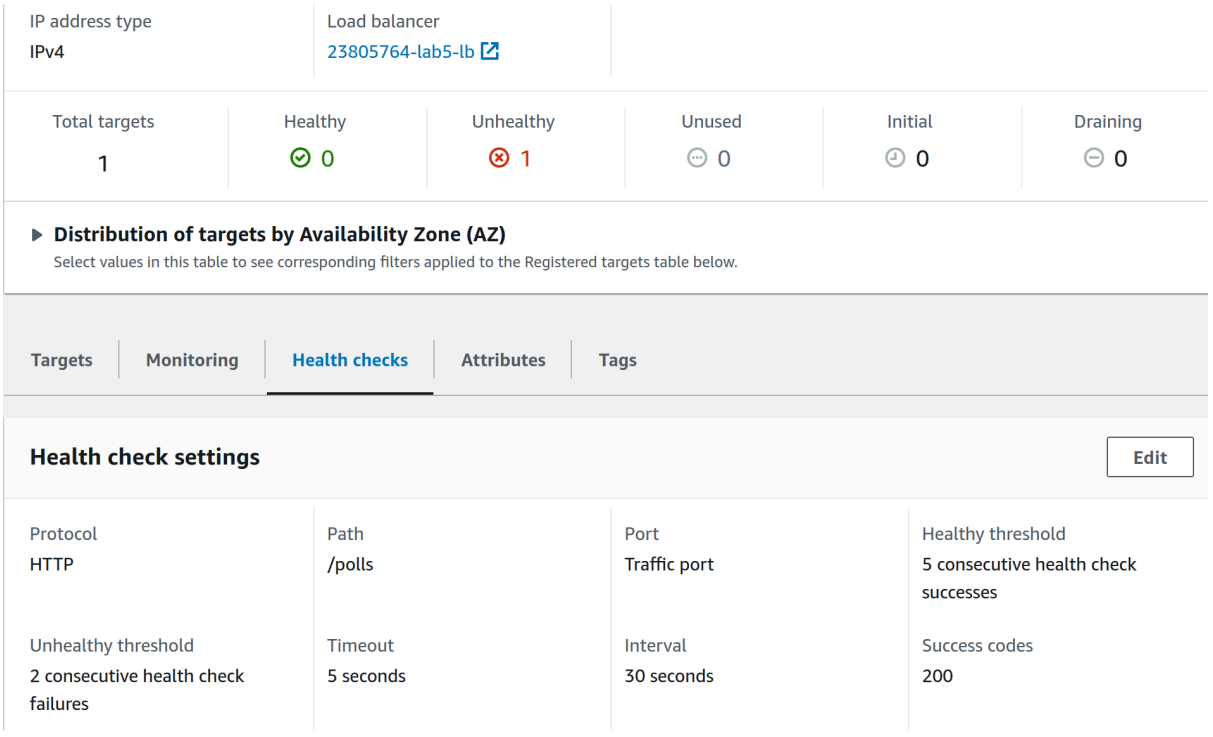
The above is the content of file createElb.py. It first queries the subnet id and VPC id of the instance I created. Then, a load balancer is created with two subnet ids as required, the second subnet id is found in the AWS console. A target group is created and configured to listen to path /polls.

```
python3 createElb.py
```

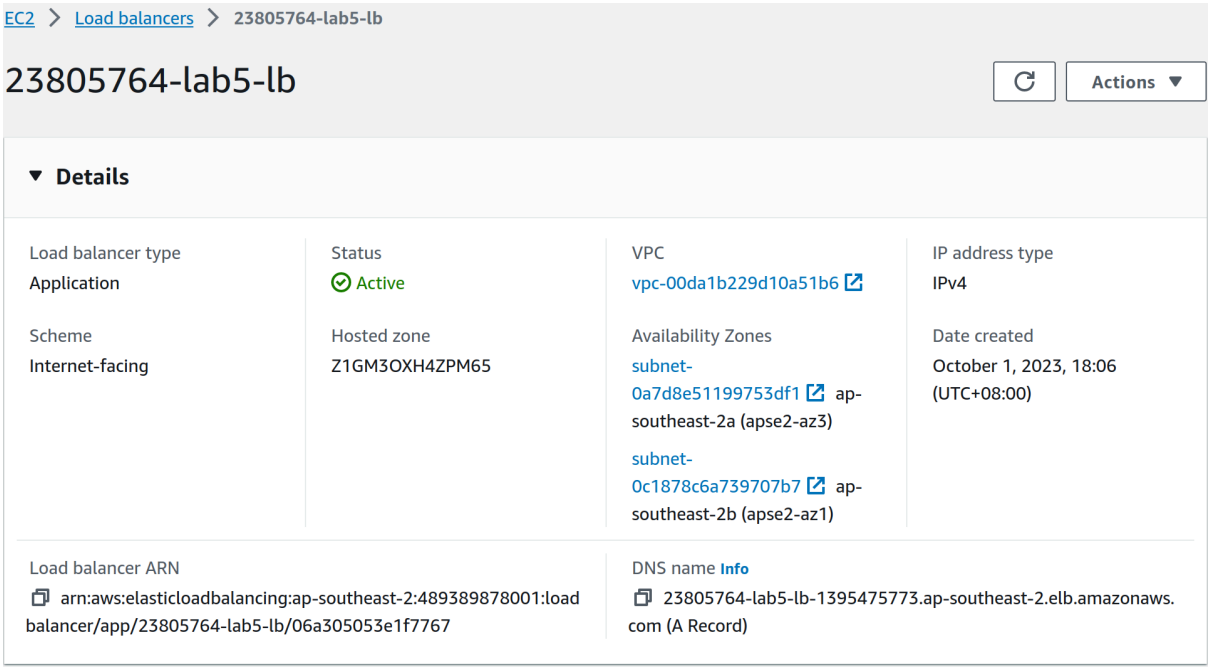
Run the program.

```
Queried instance i-018843a24d80f879e, subnet
id=subnet-0c1878c6a739707b7, vpc id=vpc-00da1b229d10a51b6
Created Loadbalancer with name=23805764-lab5-lb,
arn=arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:loadbalance
r/app/23805764-lab5-lb/06a305053e1f7767
Created target group with name=23805764-lab6-tg,
arn=arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:targetgroup
/23805764-lab6-tg/bceb64a73d2f3529
Registered instnace i-018843a24d80f879e to target group with
arn=arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:targetgroup
/23805764-lab6-tg/bceb64a73d2f3529
Created listener,
arn=arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:listener/ap
p/23805764-lab5-lb/06a305053e1f7767/df2744f942f1689d
```

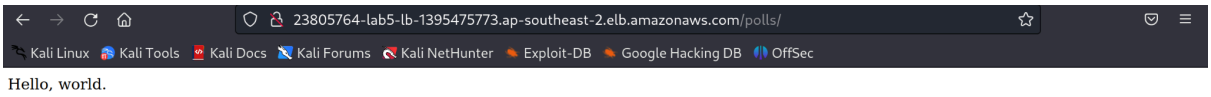
Above output shows some of the id and arn of different resources.



Above screen shot shows that the target group created is checking the path /polls as required.



Above screenshot shows my load balancer's DNS name. This url should be visited in the browser to check if the application is accessible.



Above screen shot shows that I am accessing the /poll path with the load balancer's DNS name, and the hello world page is displayed.

Lab7 - DevOps

This lab involves writing python programs using the fabric library. The program should be able to configure a Ec2 instance to install nginx and create a Django web application.

[1] Create an Ec2 instance

Code from previous labs is copied and altered to create an Ec2 instance.

```
import boto3
import json
import os
import stat

ec2 = boto3.client("ec2")

sg_GroupName = "23805764-lab7-sg"
myKeyName = "23805764-lab7-key"

# create security group
csg_response = ec2.create_security_group(
   GroupName=sg_GroupName,
   Description="Security group for lab 6 net working.")

sg_id = csg_response["GroupId"]
print(f"Created security group: id={sg_id}")

# configure security group for ssh
asgi_response = ec2.authorize_security_group_ingress(
   GroupName=sg_GroupName,
   IpProtocol="tcp",
   FromPort=22,
   ToPort=22,
   CidrIp="0.0.0.0/0"
)

print("configured security group inbound traffic for ssh")
print(json.dumps(asgi_response["SecurityGroupRules"], indent=4))

# configure security group for http
asgi_response = ec2.authorize_security_group_ingress(
   GroupName=sg_GroupName,
   IpProtocol="tcp",
```

```
    FromPort=80,
    ToPort=80,
    CidrIp="0.0.0.0/0"
)


print("configured security group inbound traffic for http")
print(json.dumps(asgi_response["SecurityGroupRules"], indent=4))

# create the key pair
ckp_response = ec2.create_key_pair(KeyName=myKeyName)
key_id = ckp_response["KeyPairId"]
print(f"Successfully created key pair: KeyPairId={key_id}")

ssh_dir = f"{os.getenv('HOME')}/ssh"
key_out_file = "23805764-lab7-key.pem"
key_out_ssh_file = os.path.join(ssh_dir, key_out_file)

# make sure ~/ssh is created
if not os.path.isdir(ssh_dir):
    os.mkdir(ssh_dir)

keyFile = open(key_out_file, "w")
keyFile2 = open(key_out_ssh_file, "w")
keyFile.write(ckp_response["KeyMaterial"])
keyFile2.write(ckp_response["KeyMaterial"])
print(f"Wrote the key content to file: {key_out_file} and
{key_out_ssh_file}")
keyFile.close()
keyFile2.close()

# key file permission can only be read by user
os.chmod(key_out_file, stat.S_IREAD)
os.chmod(key_out_ssh_file, stat.S_IREAD)
print("Changed key file permission")


rs_response = ec2.run_instances(
    ImageId="ami-0310483fb2b488153",
    SecurityGroups=[sg_GroupName],
    MinCount=1,
    MaxCount=1,
    InstanceType="t2.micro",
```

```
    KeyName=myKeyName
)

instance_id = rs_response["Instances"][0]["InstanceId"]
print(f"Created ec2 instance: id={instance_id}")

instance_name = "23805764-lab7-3-ec2"
ec2.create_tags(
    Resources=[instance_id],
    Tags=[{
        "Key": "Name",
        "Value": instance_name
    }]
)
print(f"Assigned tags to instance {instance_id} Name={instance_name}")
```

The above is the content of file createEc2.py, which is a combination of code used in the previous lab for creating an Ec2 instance together. The changes made include saving the key to ~/ssh folder for use in the next step.

[2] Install and configure Fabric on your VM

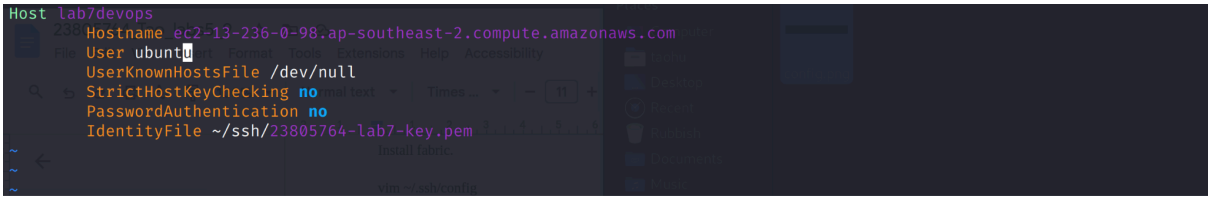
The first step is to install fabric on my VM running from UTM.

```
pip install fabric
```

Install fabric.

```
vim ~/.ssh/config
```

Edit the config file in ~/.ssh.



The above is a screenshot of the contents in the config file.

Now I should test that I can set up a connection in the python interactive environment.

```
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from fabric import Connection
>>> c = Connection("lab7devops")
>>> res = c.run("uname -s")
Linux
>>>
```

Above screen shows that I can connect to the Ec2 instance in python.

[3] Write a python script to automate the installation of nginx

```
from fabric import Connection

con = Connection("lab7devops")

# upgrade
con.sudo("apt-get update")
con.sudo("apt-get upgrade")
con.sudo("apt-get install python3-venv")

# install nginx
con.sudo("apt install nginx")

default_content = """
server {
 listen 80 default_server;
 listen [::]:80 default_server;

 location / {
   proxy_set_header X-Forwarded-Host \$host;
   proxy_set_header X-Real-IP \$remote_addr;

   proxy_pass http://127.0.0.1:8000;
 }
}
"""

# change /etc/nginx/site-enabled/default
con.sudo(f"echo \"{default_content}\" | sudo tee
/etc/nginx/sites-enabled/default")

con.sudo("service nginx restart")
```

Above is the content of the file initNginx.py. The code will use fabric to update and upgrade the Ec2 instance. Then nginx is installed and the site's default file is modified.

```
python3 installNginx.py
```

Running the above to install Nginx

[4] Update the python script to install your Django app

The instructions for this lab are not clear, so some assumptions are made. Assuming this part is to simulate a DevOps process, where code is implemented locally and then deployed to a Ec2 instance.

The first step is to create a Django app locally.

```
pip install django;
django-admin startproject lab;
cd lab;
python3 manage.py startapp polls
```

The above commands are run in my VM that runs on UTM. First django is installed, then a new project called lab is created and a polls folder is created inside.

Then the next step is to edit the project and update the files according to the previous lab so that the /polls path displays a hellworld html page. I used VSCode to edit and make all the modifications.

Now a python program needs to be written to simulate the DevOps process. This involves configuring the remote Ec2 instance by creating a python virtual environment. Then uploading the local app to the Ec2 instance and then running the app in the background.

```
from fabric import Connection
import subprocess

con = Connection("lab7devops")

# remote target directory
site_path = "/opt/wwc/mysites"

con.sudo(f"mkdir -p {site_path}")

zip_name = "lab.zip"
# use tar to zip the app
subprocess.run(["tar", "-cvzf", zip_name, "lab"])
```

```
with con.cd(site_path):
    # sudo bash cannot be run, meaning django cannot be installed
    # because of permission, so sudo chmod to change permission of
    # /opt recursively.
    con.run("sudo chmod -R 777 /opt")
    con.run("if [[ ! -d myvenv ]]; then python3 -m venv myvenv; fi")
    # use put to transfer file and then unzip in the Ec2 instance
    with con.prefix("source myvenv/bin/activate"):
        con.run("pip install django")
        con.put(zip_name, f"{site_path}/")
        con.run("tar -xvzf lab.zip")
        # run in the background
        con.run("python3 lab/manage.py runserver 8000 &")
```

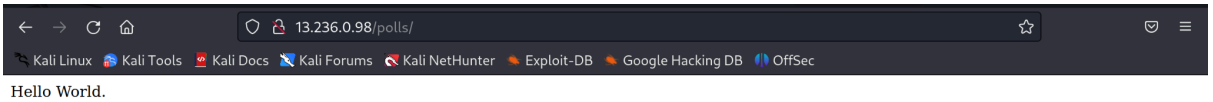
The above is the content of file initApp.py. It first zips the django project locally and then transfers to the Ec2 instance. Then the app is started in the background.

```
python3 initApp.py
```

Run the program to perform a transfer project and run on the Ec2 instance.

```
ubuntu@ip-172-31-5-239:/opt/wwc/mysites$ ps aux | grep runserver
ubuntu     33334  0.0  0.1   7760  1948 ?        S    05:01   0:00 bash -c cd /opt/wwc/mysites && source myvenv/bin/activat
e && python3 lab/manage.py runserver 8000 &
ubuntu     33335  0.2  3.8  48260 38336 ?        S    05:01   0:00 python3 lab/manage.py runserver 8000
ubuntu     33336  0.9  4.1 197868 41512 ?        Sl   05:01   0:00 /opt/wwc/mysites/myvenv/bin/python3 lab/manage.py runser
ver 8000
ubuntu     33364  0.0  0.2   7004  2252 pts/0    S+   05:03   0:00 grep --color=auto runserver
ubuntu@ip-172-31-5-239:/opt/wwc/mysites$
```

The above screenshot shows that the project is running in the background after initApp.py completed.



The above also verifies that visiting the /polls path will display the hello world html page.

Lab8 - AI

The purpose of this lab is to use jupyter notebook and pandas to create a dataset. Boto3 and sagemaker will also be used to create training and hyperparameter optimisation jobs.

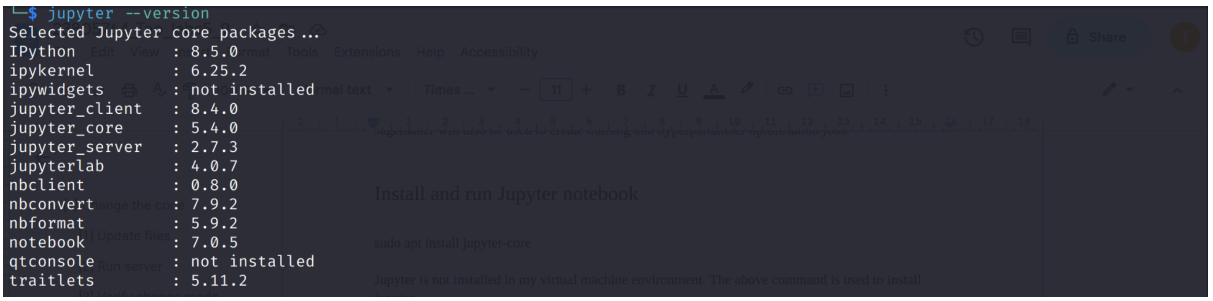
Install and run Jupyter notebook

```
sudo apt install jupyter-core
```

Jupyter is not installed in my virtual machine environment. The above command is used to install jupyter.

```
jupyter --version
```

Above command to show the jupyter version.



Above screenshot verifies that jupyter is installed and the notebook version is 7.0.5.

```
pip3 install notebook
jupyter notebook
```

The above command uses pip3 to install a notebook for python. The second command is used to start a jupyter notebook application.

```
Jupyter command `jupyter-notebook` not found.
```

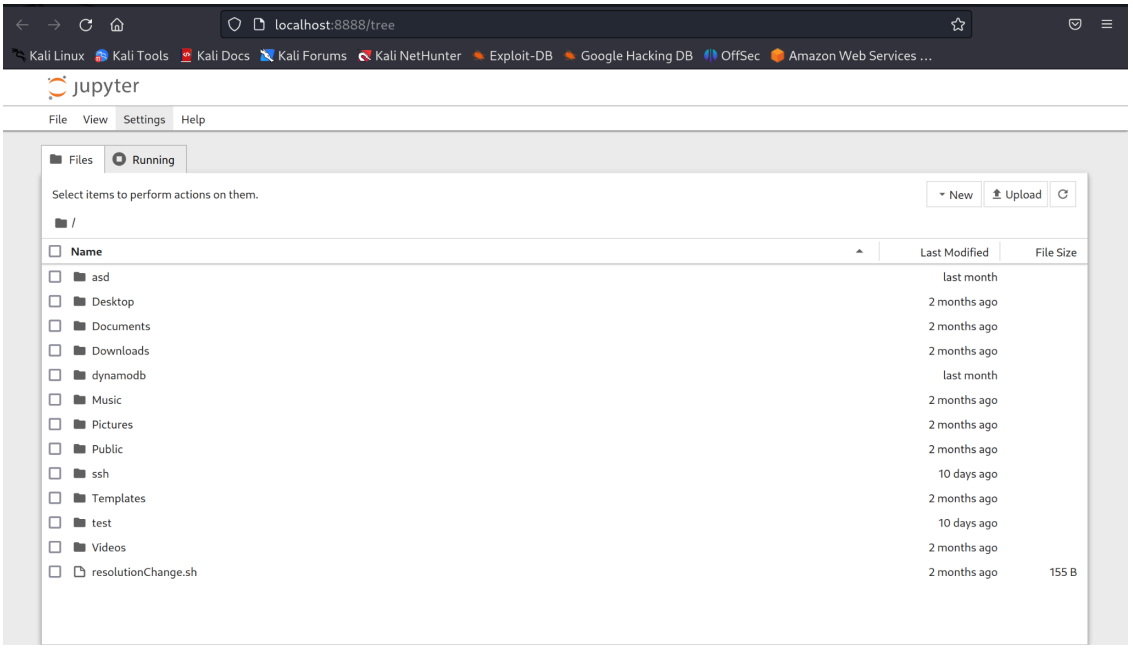
The above error output comes from the jupyter notebook command.

```
sudo -H pip3 install notebook
```

After some research, the solution is to use the above command to install a notebook with pip.

jupyter notebook

The above command is run to start up the jupyter notebook server.



Above screenshot shows the output of the above command. The above page is automatically opened in the browser.

Set Up Python Environment

```
pip3 install sagemaker pandas ipykernel
```

Use the above command to install sagemaker, pandas and ipykernel.

```
pip3 freeze | grep "sagemaker\|pandas\|ipykernel"
```

Use the above command to verify the above are installed and the verison.

```
ipykernel==6.25.2
pandas==1.5.3
sagemaker==2.192.0
```

The above output shows the version of the three libraries.

Run Hyperparameter Tuning jobs

The instructions for this part of the lab are stored in a jupyter notebook file.

```
wget
https://github.com/zhangzhics/CITS5503_Sem2_2023/raw/master/Labs/src/Lab
AI.ipynb
```

Use the above command to download the jupyter notebook file.

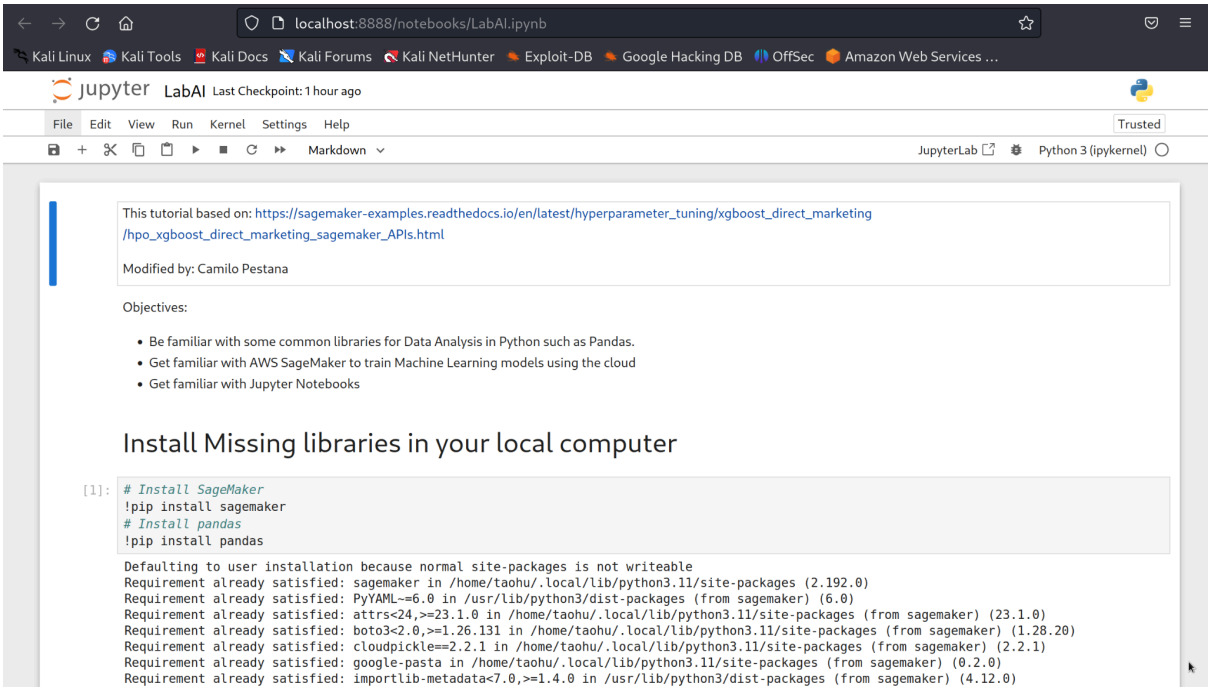
```
ls -l
```

Show files in the current directory.

```
total 24
-rw-r--r-- 1 taohu taohu 18055 Oct 12 17:54 LabAI.ipynb
drwxr-xr-x 2 taohu taohu  4096 Oct 12 17:08 ss
```

The above output shows the file LabAi.ipynb is in the current directory.

Then I open the downloaded file in the browser where the jupyter is running.



The above screenshot shows the content of the notebook.

After reading the prepared sagemaker session section, I need to create a s3 bucket.

```
import boto3
import json

s3Client = boto3.client("s3")

cb_response = s3Client.create_bucket(
   Bucket="23805764-s3",
   CreateBucketConfiguration={
        'LocationConstraint': 'ap-southeast-2',
   },
)

print(json.dumps(cb_response, indent=4))
```

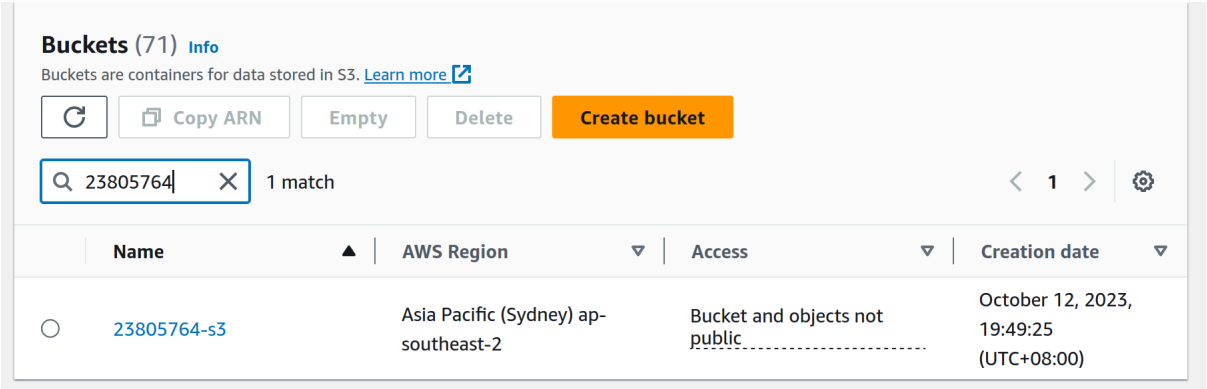
The above is the content of the file createS3.py. The program creates a S3 bucket in the ap-southeast-2 region.

```
python3 createS3.py
```

Run the python program to create the S3 bucket.

```
{
      "ResponseMetadata": {
      "RequestId": "B7YPJVCAAV8EV8NJ",
      "HostId":
"RTU9MYzrI/DLp2YHPTN3e5jp2AiCFzGrSy0gojCV18uCsSBb8tRSMR0jideb8PsiQpn+ucT
R0QQ=",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
            "x-amz-id-2":
"RTU9MYzrI/DLp2YHPTN3e5jp2AiCFzGrSy0gojCV18uCsSBb8tRSMR0jideb8PsiQpn+ucT
R0QQ=",
            "x-amz-request-id": "B7YPJVCAAV8EV8NJ",
            "date": "Thu, 12 Oct 2023 11:49:24 GMT",
            "location": "http://23805764-s3.s3.amazonaws.com/",
            "server": "AmazonS3",
            "content-length": "0"
      },
      "RetryAttempts": 0
      },
      "Location": "http://23805764-s3.s3.amazonaws.com/"
}
```


The above is the output showing the response. The important information in the above json output is the Location attribute.



The above screenshot shows that my bucket has been created.

The next step is to modify the configurations in the notebook. This includes replacing the placeholders with my student id.

Prepare SageMaker session

```
[5]: import sagemaker
import boto3

import numpy as np  # For matrix operations and numerical processing
import pandas as pd  # For munging tabular data
from time import gmtime, strftime
import os

region = 'ap-southeast-2'
smclient = boto3.Session().client("sagemaker")

iam = boto3.client('iam')
sagemaker_role = iam.get_role(RoleName='SageMakerRole')['Role']['Arn']

student_id = "23805764"
bucket = '23805764-s3'
prefix = f"sagemaker/{student_id}-hpo-xgboost-dm"
```

The above is a screenshot showing the modified prepare sagemaker session section. The existing RoleName is outdated and the new rolename is found in the aws console.

The next step in the jupyter notebook is to download the dataset

Please take some time to read about the data with more detail here Let's start by downloading the direct marketing dataset from UCI's ML Repository.

You can download the dataset manually or use the commands below. These commands should work for Linux and MacOS users.

```
[2]: !wget -N https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip
!unzip -o bank-additional.zip
```

```
--2023-10-13 13:57:04--  https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: 'bank-additional.zip'

bank-additional.zip     [      <=>           ] 434.15K   370KB/s    in 1.2s

Last-modified header missing -- time-stamps turned off.
2023-10-13 13:57:07 (370 KB/s) - 'bank-additional.zip' saved [444572]

Archive:  bank-additional.zip
   creating: bank-additional/
  inflating: bank-additional/.DS_Store
   creating: __MACOSX/
   creating: __MACOSX/bank-additional/
  inflating: __MACOSX/bank-additional/._.DS_Store
  inflating: bank-additional/.Rhistory
  inflating: bank-additional/bank-additional-full.csv
  inflating: bank-additional/bank-additional-names.txt
  inflating: bank-additional/bank-additional.csv
  inflating: __MACOSX/._bank-additional
```

The above code block is run. The data in the form a zip is downloaded and then unzipped

```
ls -l
```

Run the above in the terminal in the same directory as the notebook is in.

```
total 480
drwx------ 2 taohu taohu   4096 Mar 26  2014 bank-additional
-rw-r--r-- 1 taohu taohu 444572 Oct 13 13:57 bank-additional.zip
-rw-r--r-- 1 taohu taohu      251 Oct 12 19:49 createS3.py
-rw-r--r-- 1 taohu taohu  25369 Oct 13 13:58 LabAI.ipynb
drwxrwxr-x 3 taohu taohu   4096 Mar 26  2014 __MACOSX
drwxr-xr-x 2 taohu taohu   4096 Oct 13 13:57 ss
```

Above output shows some of the files downloaded from running the above code block.

[3]:
```
data = pd.read_csv("./bank-additional/bank-additional-full.csv", sep=";")
pd.set_option("display.max_columns", 500)  # Make sure we can see all of the columns
pd.set_option("display.max_rows", 50)  # Keep the output on one page
data
```

[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.var.r
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0	nonexistent	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0	nonexistent	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0	nonexistent	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0	nonexistent	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	0	nonexistent	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	334	1	999	0	nonexistent	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	383	1	999	0	nonexistent	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	189	2	999	0	nonexistent	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	442	1	999	0	nonexistent	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	239	3	999	1	failure	

41188 rows × 21 columns

The above shows the next code block and the output, which shows the data in a table.

Answer the following questions: - Which variables are categorical? - Which ones are numerical?

Variables such as age and duration are numerical.

Variables such as martial and month are categorical

[4]:
```
data["no_previous_contact"] = np.where(
    data["pdays"] == 999, 1, 0
)  # Indicator variable to capture when pdays takes a value of 999
data["not_working"] = np.where(
    np.inld(data["job"], ["student", "retired", "unemployed"]), 1, 0
)  # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)  # Convert categorical variables to sets of indicators
model_data
```

[4]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	no_previous_contact	not_working	job_admin.	job_bl co
0	56	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	0	0	
1	57	149	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	0	0	
2	37	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	0	0	
3	40	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	0	1	
4	56	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0	1	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	334	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	1	0	
41184	46	383	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	0	0	
41185	56	189	2	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	1	0	
41186	44	442	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1	0	0	
41187	74	239	3	999	1	-1.1	94.767	-50.8	1.028	4963.6	1	1	0	

41188 rows × 67 columns

The above code block creates some new categorical variables based on some other categorical variables. Then all categorical variables are converted into a number of boolean variables based on the number of unique values.

```
[10]: model_data = model_data.drop(
          ["duration", "emp.var.rate", "cons.price.idx", "cons.conf.idx", "euribor3m", "nr.employed"],
          axis=1,
      )
```

```
[7]: model_data
```

[7]:

	age	campaign	pdays	previous	no_previous_contact	not_working	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self employe
0	56	1	999	0	1	0	0	0	0	1	0	0	(
1	57	1	999	0	1	0	0	0	0	0	0	0	(
2	37	1	999	0	1	0	0	0	0	0	0	0	(
3	40	1	999	0	1	0	1	0	0	0	0	0	(
4	56	1	999	0	1	0	0	0	0	0	0	0	(
...	...	...	...	...	...	...	...	...	...	...	...	...	..
41183	73	1	999	0	1	1	0	0	0	0	0	1	(
41184	46	1	999	0	1	0	0	1	0	0	0	0	(
41185	56	2	999	0	1	1	0	0	0	0	0	1	(
41186	44	1	999	0	1	0	0	0	0	0	0	0	(
41187	74	3	999	1	1	1	0	0	0	0	0	1	(

41188 rows × 61 columns

The next code block is run to remove some of the variables.

```
[11]: train_data, validation_data, test_data = np.split(
          model_data.sample(frac=1, random_state=1729),
          [int(0.7 * len(model_data)), int(0.9 * len(model_data))],
      )

      pd.concat([train_data["y_yes"], train_data.drop(["y_no", "y_yes"], axis=1)], axis=1).to_csv(
          "train.csv", index=False, header=False
      )
      pd.concat(
          [validation_data["y_yes"], validation_data.drop(["y_no", "y_yes"], axis=1)], axis=1
      ).to_csv("validation.csv", index=False, header=False)
      pd.concat([test_data["y_yes"], test_data.drop(["y_no", "y_yes"], axis=1)], axis=1).to_csv(
          "test.csv", index=False, header=False
      )
```

Above code block is run next, which will split the dataset into training, validation and test. Then, these data are stored in the corresponding .csv files.

```
ls -l *.csv
```

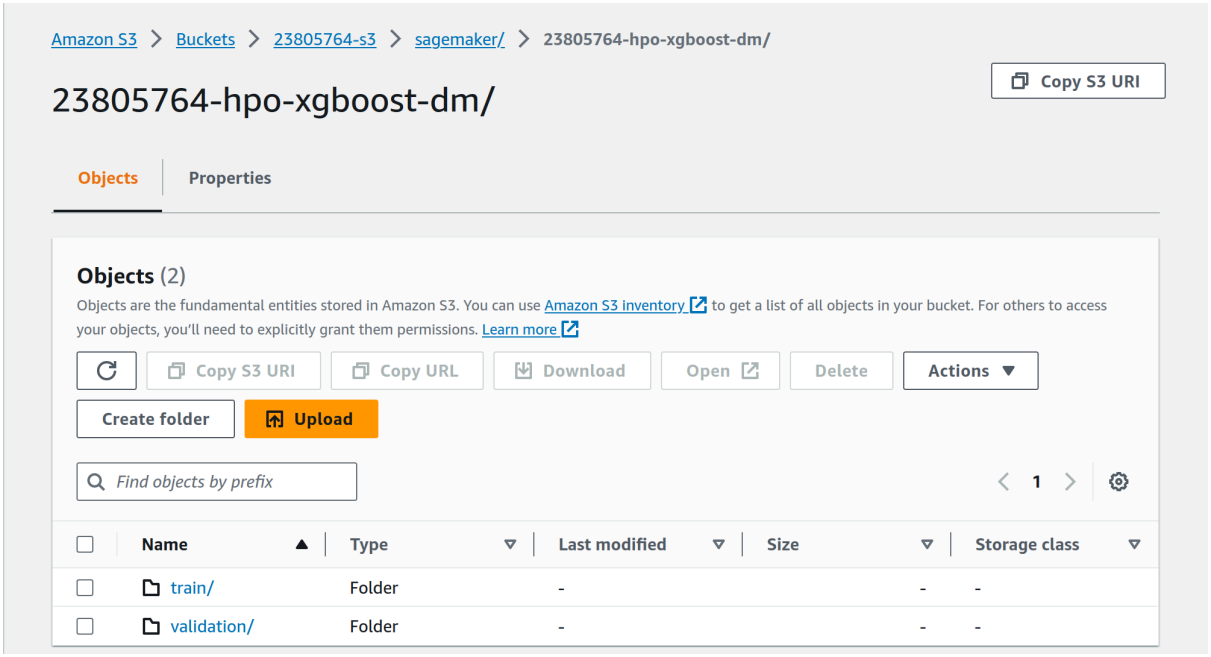
Run the above command in the terminal to verify the .csv is created.

```
-rw-r--r-- 1 taohu taohu  506492 Oct 13 14:26 test.csv
-rw-r--r-- 1 taohu taohu 3544984 Oct 13 14:26 train.csv
-rw-r--r-- 1 taohu taohu 1012968 Oct 13 14:26 validation.csv
```

The above is the output showing the 3 splitted data sets.

```
[12]: boto3.Session().resource("s3").Bucket(bucket).Object(
          os.path.join(prefix, "train/train.csv")
      ).upload_file("train.csv")
      boto3.Session().resource("s3").Bucket(bucket).Object(
          os.path.join(prefix, "validation/validation.csv")
      ).upload_file("validation.csv")
```

The above code block is run next. The train.csv and validation.csv created previously are uploaded to my S3 bucket.



Above screenshot shows that both .csv files are stored in my S3 bucket.

Above screenshot shows some part of the next code block that is run and the output. A new hyper tuning config is created, but not yet uploaded to AWS.

```
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "1",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 43200},
}
```

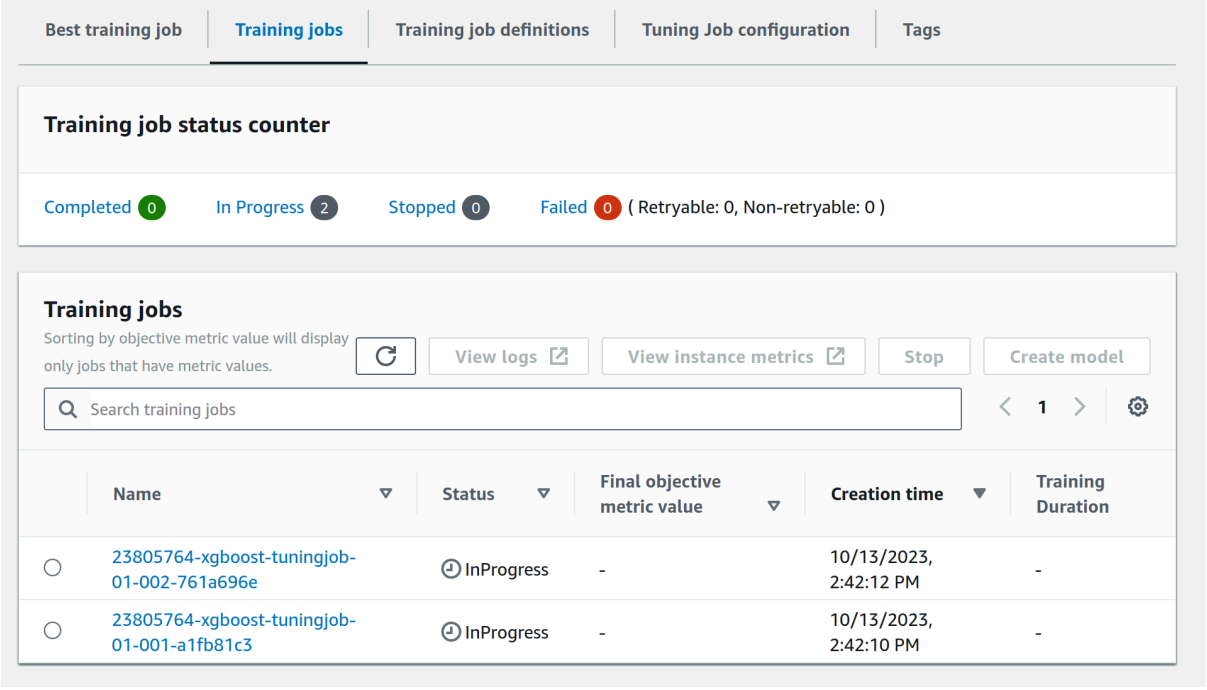
Now we can launch a hyperparameter tuning job by calling create_hyper_parameter_tuning_job API. After the hyperparameter tuning job is created, we can go to SageMaker console to track the progress of the hyperparameter tuning job until it is completed.

While Training you can take screenshots of the jobs you just launched on SageMaker-> Training -> Hyperparameter tuning jobs

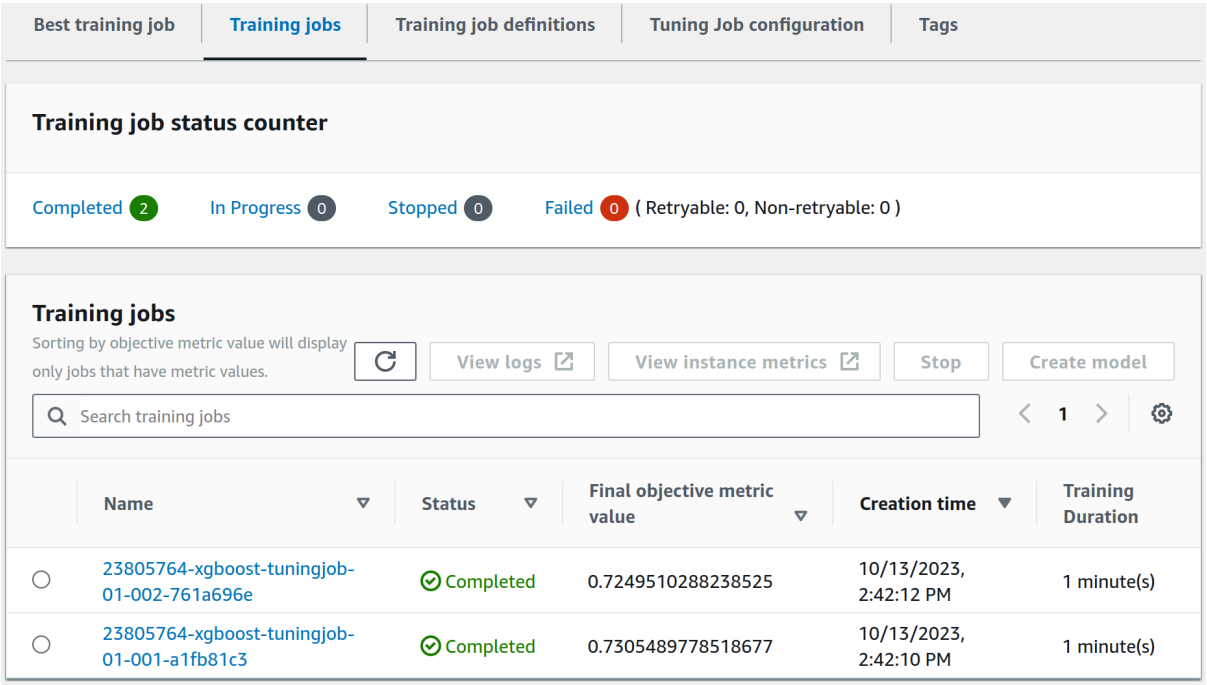
[15]:
```
#Launch Hyperparameter Tuning Job
smclient.create_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name,
    HyperParameterTuningJobConfig=tuning_job_config,
    TrainingJobDefinition=training_job_definition,
)
```

[15]:
```
{'HyperParameterTuningJobArn': 'arn:aws:sagemaker:ap-southeast-2:489389878001:hyper-parameter-tuning-job/23805764-xgboost-tuningjob-01',
 'ResponseMetadata': {'RequestId': '0cda5906-d2f9-4eb6-9f6e-d06142cbb5f6',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '0cda5906-d2f9-4eb6-9f6e-d06142cbb5f6',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '135',
   'date': 'Fri, 13 Oct 2023 06:42:06 GMT'},
  'RetryAttempts': 0}}
```

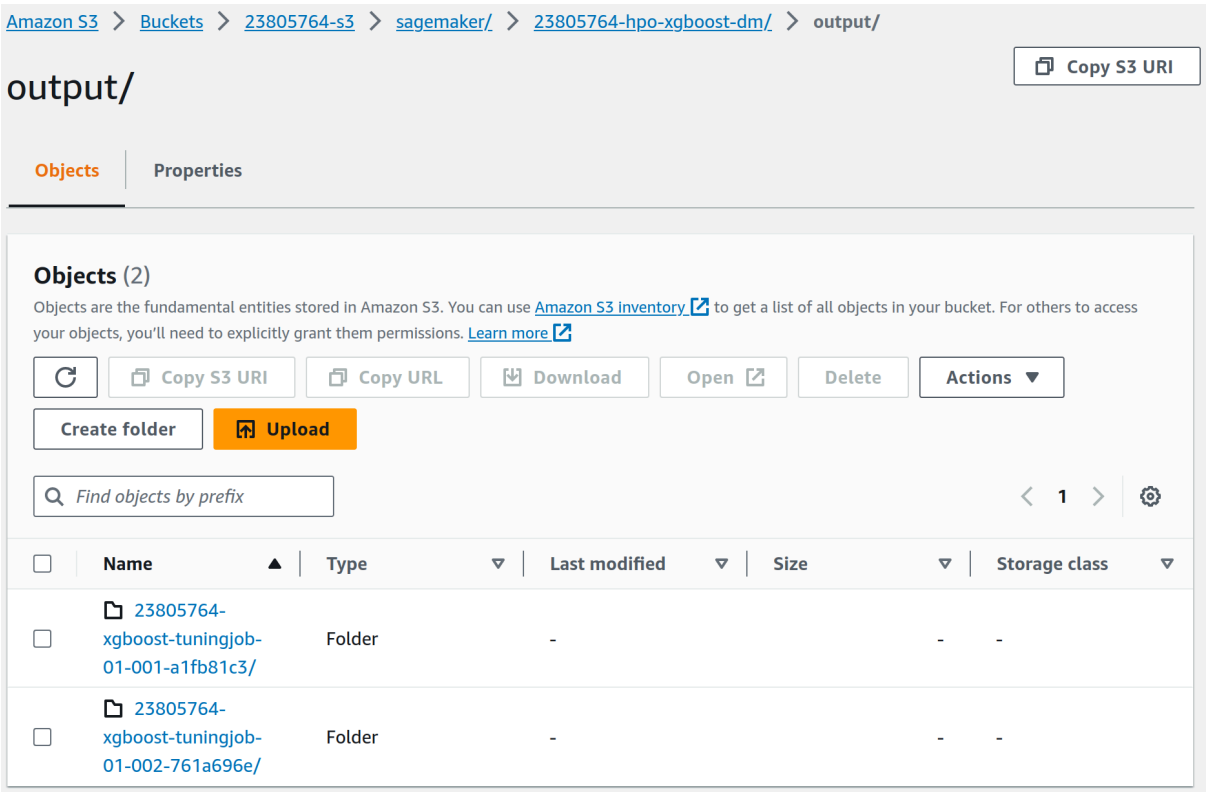
The above two code blocks are run next. The first code block creates a training configuration. Then the second code block uses the previously created tuning configuration and training configuration to create a hyper parameter tuning job in the AWS sagemaker.



Above screenshot shows the two training jobs that are still in progress in my hyper parameter section.



Above screen shot shows the two tuning jobs after some minutes. It can be seen that the status is completed.



The above screenshot shows the output from the completed tuning job.

Lab9 - More AI

The purpose of this lab is to get familiar with some of the AI services provided by AWS. This will involve creating scripts to use features from AWS Comprehend and AWS Rekognition.

AWS Comprehend

[1] Detecting languages from text

[1.1] Modify code

After doing some research, I found out that there is a library called iso639 that can be used to convert the language codes to the language name.

```
pip3 install iso-639
```

The above command is run in the terminal to install the iso639 library.

```
code .. > /dev/null &
```

Use the above to open vscode editor for writing python scripts.

```
import boto3
from iso639 import languages

client = boto3.client('comprehend')

# Detect Entities
response = client.detect_dominant_language(
   Text="The French Revolution was a period of social and political
upheaval in France and its colonies beginning in 1789 and ending in
1799.",
)

language_out = response['Languages'][0]

print(f"{languages.get(alpha2=language_out['LanguageCode']).name}
detected with {int(language_out['Score'] * 100)} confidence")
```

The above code of file detectLanguage.py. The iso639 library is used to convert the language code into the language name.

```
python3 detectLanguage.py
```

```
┌──(taohu㉿192-168-064-3)-[~/Documents/CC/labs/lab9]
└─$ python3 detectLangauge.py
English detected with 99 confidence
```

Above output is as expected.

[1.2] Test your code with other languages

There are 3 different sets of text from different languages to test in this section. In order to perform the test, the detectLanguage.py file from earlier has to be modified to take the text from the program arguments.

```
import boto3
import sys
from iso639 import languages

client = boto3.client('comprehend')

inputText = "The French Revolution was a period of social and political
upheaval in France and its colonies beginning in 1789 and ending in
1799."

if len(sys.argv) > 1:
   inputText = sys.argv[1]

# Detect Entities
response = client.detect_dominant_language(
   Text=inputText,
)

language_out = response['Languages'][0]

print(f"{languages.get(alpha2=language_out['LanguageCode']).name}
detected with {int(language_out['Score'] * 100)} confidence")
```

The above is the content of the modified detectLanguage.py. If text is found from the first command line argument then it is replaced and tested. If no arguments are found the original english text is used.

The next step would then be to run the program three times with the three different texts. However, another python script can be created to do this instead.

```
import subprocess
```

```
spanish = """\"El Quijote es la obra más conocida de Miguel de
Cervantes Saavedra. Publicada su primera parte con el título de El
ingenioso hidalgo don Quijote de la Mancha a comienzos de 1605, es una
de las obras más destacadas de la literatura española y la literatura
universal, y una de las más traducidas. En 1615 aparecería la segunda
parte del Quijote de Cervantes con el título de El ingenioso caballero
don Quijote de la Mancha.\""""
french = """Moi je n'étais rien Et voilà qu'aujourd'hui Je suis le
gardien Du sommeil de ses nuits Je l'aime à mourir Vous pouvez détruire
Tout ce qu'il vous plaira Elle n'a qu'à ouvrir L'espace de ses bras
Pour tout reconstruire Pour tout reconstruire Je l'aime à mourir" [From
the Song: "Je l'Aime a Mourir" - Francis Cabrel ]"""
Italian = """L'amor che move il sole e l'altre stelle." [Quote from
"Divine Comedy" - Dante Alighieri]"""

if __name__ == "__main__":
   subprocess.run(["python3", "detectLanguage.py", spanish])
   subprocess.run(["python3", "detectLanguage.py", french])
   subprocess.run(["python3", "detectLanguage.py", Italian])
```

The above is the content of the file called testThreeLanguage.py. The subprocess module is used to run the detection process 3 times.

```
┌──(taohu㉿192-168-064-3)-[~/Documents/CC/labs/lab9]
└─$ python3 testThreeLanguage.py
Spanish detected with 99 confidence
French detected with 99 confidence
Italian detected with 76 confidence
```

The above output shows that the first two languages had a high confidence while the last one is a bit lower.

[2] Sentiment Analysis

A python script can be created to perform the detection of the sentiment on the three different phrases.

```
import boto3
import json
from testThreeLanguage import *

client = boto3.client('comprehend')

def getLanguageCode(text):
   response = client.detect_dominant_language(
```

```
        Text=text,
    )

    language_out = response['Languages'][0]

    return language_out["LanguageCode"]

def getSentimentOutput(text):
    return client.detect_sentiment(
        Text=text,
        LanguageCode=getLanguageCode(text)
    )

if __name__ == "__main__":
    print(json.dumps(getSentimentOutput(spanish), indent=4))
    print(json.dumps(getSentimentOutput(french), indent=4))
    print(json.dumps(getSentimentOutput(Italian), indent=4))
```

The above is the content of the file detectSentiment.py. The process of finding the sentiment is to first retrieve the language code.

```
python3 detectSentiment.py
```

Run the program.


```
{
        "Sentiment": "NEUTRAL",
        "SentimentScore": {
        "Positive": 0.19532117247581482,
        "Negative": 0.0014884761767461896,
        "Neutral": 0.8015366196632385,
        "Mixed": 0.0016537061892449856
        },
        "ResponseMetadata": {
        "RequestId": "1415a55d-64a0-413f-8746-bb5de9962ab4",
        "HTTPStatusCode": 200,
        "HTTPHeaders": {
              "x-amzn-requestid": "1415a55d-64a0-413f-8746-bb5de9962ab4",
              "content-type": "application/x-amz-json-1.1",
              "content-length": "165",
              "date": "Fri, 13 Oct 2023 12:14:00 GMT"
        },
        "RetryAttempts": 0
        }
```

```
}
{
      "Sentiment": "POSITIVE",
      "SentimentScore": {
      "Positive": 0.6583537459373474,
      "Negative": 0.14182481169700623,
      "Neutral": 0.19934231042861938,
      "Mixed": 0.00047905041719786823
      },
      "ResponseMetadata": {
      "RequestId": "66106414-3cfd-4f50-967e-ad33c884e4ef",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
            "x-amzn-requestid": "66106414-3cfd-4f50-967e-ad33c884e4ef",
            "content-type": "application/x-amz-json-1.1",
            "content-length": "164",
            "date": "Fri, 13 Oct 2023 12:14:00 GMT"
      },
      "RetryAttempts": 0
      }
}
{
      "Sentiment": "POSITIVE",
      "SentimentScore": {
      "Positive": 0.5614008903503418,
      "Negative": 0.00035198486875742674,
      "Neutral": 0.43817514181137085,
      "Mixed": 7.199830724857748e-05
      },
      "ResponseMetadata": {
      "RequestId": "dae22274-52b3-4a2c-b7ca-9d6d7f9725ef",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
            "x-amzn-requestid": "dae22274-52b3-4a2c-b7ca-9d6d7f9725ef",
            "content-type": "application/x-amz-json-1.1",
            "content-length": "165",
            "date": "Fri, 13 Oct 2023 12:14:01 GMT"
      },
      "RetryAttempts": 0
      }
}
```


The above is the output, the first phrase is neutral and the rest are positive.

[3] Detect entities

```
import json
import boto3
from detectSentiment import *

def getEntitiesOutput(text):
    return client.detect_entities(
        Text=text,
        LanguageCode=getLanguageCode(text)
    )

def printFilteredEntities(response):
    for entity in response["Entities"]:
        print(f"Text: {entity['Text']}, Score: {entity['Score']}")

if __name__ == "__main__":
    print("\nspanish")
    printFilteredEntities(getEntitiesOutput(spanish))
    print("\nfrench")
    printFilteredEntities(getEntitiesOutput(french))
    print("\nitalian")
    printFilteredEntities(getEntitiesOutput(Italian))
```

The above is the content of the file detectEntities.py. The same procedure is used, language code is retrieved then entities are retrieved. The entities are filtered to only print the text and score.

```
spanish
Text: El Quijote, Score: 0.9781462550163269
Text: Miguel de Cervantes Saavedra, Score: 0.9994053244590759
Text: primera parte, Score: 0.8793331980705261
Text: El ingenioso hidalgo don Quijote de la Mancha, Score:
0.8935216069221497
Text: 1605, Score: 0.8937330842018127
Text: una, Score: 0.7286107540130615
Text: española, Score: 0.9838613867759705
Text: una de las más, Score: 0.6204593181610107
Text: 1615, Score: 0.9873249530792236
Text: segunda parte, Score: 0.8961632251739502
Text: Quijote de Cervantes, Score: 0.6453461647033691
Text: El ingenioso caballero don Quijote de la Mancha, Score:
0.8628606796264648
```

```
french
Text: aujourd'hui, Score: 0.98050457239151
Text: Tout ce qu'il, Score: 0.7125235795974731
Text: tout, Score: 0.518359899520874
Text: Je, Score: 0.5280687212944031
Text: Francis Cabrel, Score: 0.9993712902069092

italian
Text: Divine Comedy, Score: 0.9266974925994873
Text: Dante Alighieri, Score: 0.9992790222167969
```

The above is the output showing information for the entities detected.

Question 1: In your words describe what entities are.

Entities can be understood as a word or set of words that can be pointed or referred to. This could be the nouns in english, person, car and school are entities.

[4] Detect keyphrases

```python
import json
import boto3
from detectSentiment import *

def getKeyPhrasesOutput(text):
    return client.detect_key_phrases(
        Text=text,
        LanguageCode=getLanguageCode(text)
    )

def printFilteredKeyPhrases(response):
    for keyPhrase in response["KeyPhrases"]:
        print(f"Text: {keyPhrase['Text']}, Score: {keyPhrase['Score']}")

if __name__ == "__main__":
    print("\nspanish")
    printFilteredKeyPhrases(getKeyPhrasesOutput(spanish))
    print("\nfrench")
    printFilteredKeyPhrases(getKeyPhrasesOutput(french))
    print("\nitalian")
    printFilteredKeyPhrases(getKeyPhrasesOutput(Italian))
```

The above is the content of the file detectKeyPhrases.py. The responses are filtered to only print the text and score.

```
python3 detectKeyPhrases.py
```

Run the program.

```
spanish
Text: El Quijote, Score: 0.9990371465682983
Text: la obra, Score: 0.9991592764854431
Text: más conocida, Score: 0.9984182119369507
Text: Miguel de Cervantes Saavedra, Score: 0.999431312084198
Text: su primera parte, Score: 0.9976469874382019
Text: el título, Score: 0.9904770851135254
Text: El ingenioso hidalgo don Quijote de la Mancha, Score:
0.8760415315628052
Text: comienzos, Score: 0.99972003698349
Text: 1605, Score: 0.9883131384849548
Text: las obras, Score: 0.9977561235427856
Text: más destacadas, Score: 0.9996249079704285
Text: la literatura española, Score: 0.999392032623291
Text: la literatura universal, Score: 0.9996477961540222
Text: las más traducidas, Score: 0.9990267753601074

french
Text: Moi, Score: 0.9994874000549316
Text: je, Score: 0.6980675458908081
Text: n'étais rien, Score: 0.9264425039291382
Text: aujourd'hui Je suis le gardien Du sommeil de ses nuits, Score:
0.9319865107536316
Text: Je, Score: 0.9998171925544739
Text: l', Score: 0.9996682405471802
Text: Vous, Score: 0.9989306330680847
Text: Tout ce, Score: 0.9805358648300171
Text: qu', Score: 0.9602937698364258
Text: il, Score: 0.9985939860343933
Text: vous, Score: 0.9994726777076721
Text: Elle, Score: 0.99965500831604
Text: L'espace de ses bras, Score: 0.9922744035720825
Text: tout, Score: 0.9700436592102051
Text: tout, Score: 0.9929914474487305
Text: Je, Score: 0.9998824596405029
Text: l', Score: 0.9998278617858887
Text: the Song, Score: 0.7904514074325562
Text: Je, Score: 0.999923586845398
Text: l', Score: 0.9994863271713257
Text: Francis Cabrel, Score: 0.9674522280693054
```

```
italian
Text: L'amor, Score: 0.9999387860298157
Text: che, Score: 0.9992818236351013
Text: il sole, Score: 0.9999328255653381
Text: l'altre stelle, Score: 0.9998706579208374
Text: Quote, Score: 0.6211094260215759
Text: Divine Comedy, Score: 0.9043272137641907
Text: Dante Alighieri, Score: 0.9998005628585815
```

The above is the output. There is much more metadata information of the actual request about the text, but is filtered out. The interesting data to text and the confidence for predicting the text as key phrases.

Question 1: In your words describe what keyphrases are.

> Key phrases are a subset of the whole phrases that decorates a noun. It would contain some words that surround an entity word and describe this particular word. For example, the car is green, the entity car as being decorated to be green.

[5] Detect syntax

```
import json
import boto3
from detectSentiment import *

def getSyntaxOutput(text):
    return client.detect_syntax(
        Text=text,
        LanguageCode=getLanguageCode(text)
    )

def printFilteredResponse(response):
    for token in response["SyntaxTokens"]:
        part_of_speech = token["PartOfSpeech"]
        print(f"Text: {token['Text']}, Tag: {part_of_speech['Tag']}
Score: {part_of_speech['Score']}")

if __name__ == "__main__":
    print("\nspanish")
    printFilteredResponse(getSyntaxOutput(spanish))
    print("\nfrench")
    printFilteredResponse(getSyntaxOutput(french))
    print("\nItalian")
```

```
  printFilteredResponse(getSyntaxOutput(Italian))
```

The above is the content of the file detectSyntax.py. The response json is filtered to only contain the text, tag and confidence.

```
python3 detectSyntax.py
```

Run the program.

```
spanish
Text: ", Tag: PUNCT Score: 0.999998927116394
Text: El, Tag: DET Score: 0.9995042085647583
Text: Quijote, Tag: PROPN Score: 0.8593556880950928
Text: es, Tag: VERB Score: 0.9999241828918457
Text: la, Tag: DET Score: 0.9999356269836426
Text: obra, Tag: NOUN Score: 0.9992451667785645
Text: más, Tag: ADV Score: 0.9999443292617798
Text: conocida, Tag: ADJ Score: 0.8230065107345581
Text: de, Tag: ADP Score: 0.999983549118042
Text: Miguel, Tag: PROPN Score: 0.9803635478019714
Text: de, Tag: ADP Score: 0.9997972846031189
Text: Cervantes, Tag: PROPN Score: 0.994879961013794
Text: Saavedra, Tag: PROPN Score: 0.9998025298118591
Text: ., Tag: PUNCT Score: 0.9999887943267822
Text: Publicada, Tag: VERB Score: 0.9990255832672119
Text: su, Tag: DET Score: 0.9999998807907104
Text: primera, Tag: ADJ Score: 0.9999592304229736
Text: parte, Tag: NOUN Score: 0.9999293088912964
Text: con, Tag: ADP Score: 0.9999922513961792
Text: el, Tag: DET Score: 0.9999998807907104
Text: título, Tag: NOUN Score: 0.997194766998291
Text: de, Tag: ADP Score: 0.9999880790710449
Text: El, Tag: DET Score: 0.999948263168335
Text: ingenioso, Tag: ADJ Score: 0.6049794554710388
Text: hidalgo, Tag: NOUN Score: 0.874273955821991
Text: don, Tag: PROPN Score: 0.4859069287776947
Text: Quijote, Tag: PROPN Score: 0.972184419631958
Text: de, Tag: ADP Score: 0.9999661445617676
Text: la, Tag: DET Score: 0.9991268515586853
Text: Mancha, Tag: PROPN Score: 0.9599988460540771
Text: a, Tag: ADP Score: 0.9998328685760498
Text: comienzos, Tag: PROPN Score: 0.5019738078117371
Text: de, Tag: ADP Score: 0.9999890327453613
Text: 1605, Tag: NUM Score: 0.999769389629364
```

```
Text: ,, Tag: PUNCT Score: 0.9999908208847046
Text: es, Tag: VERB Score: 0.9999984502792358
Text: una, Tag: PRON Score: 0.9992654919624329
Text: de, Tag: ADP Score: 0.9999909400939941
Text: las, Tag: DET Score: 0.9999719858169556
Text: obras, Tag: NOUN Score: 0.9999707937240601
Text: más, Tag: ADV Score: 0.999987006187439
Text: destacadas, Tag: ADJ Score: 0.9668636918067932
Text: de, Tag: ADP Score: 0.999997615814209
Text: la, Tag: DET Score: 0.9999908208847046
Text: literatura, Tag: NOUN Score: 0.9983854293823242
Text: española, Tag: ADJ Score: 0.9995923638343811
Text: y, Tag: CCONJ Score: 1.0
Text: la, Tag: DET Score: 0.9999812841415405
Text: literatura, Tag: NOUN Score: 0.9997329115867615
Text: universal, Tag: ADJ Score: 0.9998001456260681
Text: ,, Tag: PUNCT Score: 0.9999995231628418
Text: y, Tag: CCONJ Score: 1.0
Text: una, Tag: PRON Score: 0.9945530891418457
Text: de, Tag: ADP Score: 0.9999747276306152
Text: las, Tag: DET Score: 0.9361013174057007
Text: más, Tag: ADV Score: 0.9998936653137207
Text: traducidas, Tag: ADJ Score: 0.7704160809516907
Text: ., Tag: PUNCT Score: 0.9999884366989136
Text: En, Tag: ADP Score: 0.9998667240142822
Text: 1615, Tag: NUM Score: 0.9997568726539612
Text: aparecería, Tag: VERB Score: 0.9999510049819946
Text: la, Tag: DET Score: 0.9999918937683105
Text: segunda, Tag: ADJ Score: 0.999798595905304
Text: parte, Tag: NOUN Score: 0.9998125433921814
Text: del, Tag: ADP Score: 1.0
Text: Quijote, Tag: PROPN Score: 0.7023084759712219
Text: de, Tag: ADP Score: 0.9999327659606934
Text: Cervantes, Tag: PROPN Score: 0.9905615448951721
Text: con, Tag: ADP Score: 0.9999775886535645
Text: el, Tag: DET Score: 1.0
Text: título, Tag: NOUN Score: 0.9948986172676086
Text: de, Tag: ADP Score: 0.9999887943267822
Text: El, Tag: DET Score: 0.9999467134475708
Text: ingenioso, Tag: ADJ Score: 0.6844601631164551
Text: caballero, Tag: NOUN Score: 0.9124211072921753
Text: don, Tag: PROPN Score: 0.7402583360671997
Text: Quijote, Tag: PROPN Score: 0.9801114201545715
Text: de, Tag: ADP Score: 0.9999586343765259
Text: la, Tag: DET Score: 0.9994983673095703
Text: Mancha, Tag: PROPN Score: 0.9703096151351929
```

```
Text: ., Tag: PUNCT Score: 0.9884323477745056
Text: ", Tag: PUNCT Score: 0.9999995231628418

french
Text: ", Tag: PUNCT Score: 0.9998039603233337
Text: Moi, Tag: PRON Score: 0.9927421808242798
Text: je, Tag: PRON Score: 0.9999997615814209
Text: n', Tag: ADV Score: 0.953050971031189
Text: étais, Tag: AUX Score: 0.925997257232666
Text: rien, Tag: PRON Score: 0.9460894465446472
Text: Et, Tag: CCONJ Score: 0.9998199343681335
Text: voilà, Tag: VERB Score: 0.9998373985290527
Text: qu', Tag: SCONJ Score: 0.6309455633163452
Text: aujourd'hui, Tag: ADV Score: 0.9999371767044067
Text: Je, Tag: PRON Score: 0.9998631477355957
Text: suis, Tag: AUX Score: 0.8938817977905273
Text: le, Tag: DET Score: 0.9999561309814453
Text: gardien, Tag: NOUN Score: 0.9991658926010132
Text: Du, Tag: ADP Score: 0.7481499314308167
Text: sommeil, Tag: NOUN Score: 0.9985960125923157
Text: de, Tag: ADP Score: 0.9999814033508301
Text: ses, Tag: DET Score: 0.9996718168258667
Text: nuits, Tag: NOUN Score: 0.9992406368255615
Text: Je, Tag: PRON Score: 0.9995236396789551
Text: l', Tag: PRON Score: 0.9981119632720947
Text: aime, Tag: VERB Score: 0.9999455213546753
Text: à, Tag: ADP Score: 0.9999827146530151
Text: mourir, Tag: VERB Score: 0.9997568726539612
Text: Vous, Tag: PRON Score: 0.9900070428848267
Text: pouvez, Tag: AUX Score: 0.603804349899292
Text: détruire, Tag: VERB Score: 0.9999974966049194
Text: Tout, Tag: DET Score: 0.9437078833580017
Text: ce, Tag: PRON Score: 0.9953692555427551
Text: qu', Tag: PRON Score: 0.9307742714881897
Text: il, Tag: PRON Score: 0.9999969005584717
Text: vous, Tag: PRON Score: 0.9999879598617554
Text: plaira, Tag: VERB Score: 0.9995384216308594
Text: Elle, Tag: PRON Score: 0.9962259531021118
Text: n', Tag: ADV Score: 0.9456425905227661
Text: a, Tag: VERB Score: 0.9944429993629456
Text: qu', Tag: ADV Score: 0.8714373111724854
Text: à, Tag: ADP Score: 0.999915599822998
Text: ouvrir, Tag: VERB Score: 0.9999995231628418
Text: L', Tag: DET Score: 0.7943806648254395
Text: espace, Tag: NOUN Score: 0.9983248114585876
Text: de, Tag: ADP Score: 0.999991774559021
```

```
Text: ses, Tag: DET Score: 0.9999270439147949
Text: bras, Tag: NOUN Score: 0.9991507530212402
Text: Pour, Tag: ADP Score: 0.9851304888725281
Text: tout, Tag: PRON Score: 0.9497367739677429
Text: reconstruire, Tag: VERB Score: 0.9999784231185913
Text: Pour, Tag: ADP Score: 0.9934188723564148
Text: tout, Tag: PRON Score: 0.9504051208496094
Text: reconstruire, Tag: VERB Score: 0.9999599456787109
Text: Je, Tag: PRON Score: 0.9919244647026062
Text: l', Tag: PRON Score: 0.9929136633872986
Text: aime, Tag: VERB Score: 0.9996553659439087
Text: à, Tag: ADP Score: 0.9999797344207764
Text: mourir, Tag: VERB Score: 0.9998212456703186
Text: ", Tag: PUNCT Score: 0.9998500347137451
Text: [, Tag: PUNCT Score: 0.9990584254264832
Text: From, Tag: PROPN Score: 0.852135181427002
Text: the, Tag: DET Score: 0.9714577794075012
Text: Song, Tag: PROPN Score: 0.9267997145652771
Text: :, Tag: PUNCT Score: 0.9985225796699524
Text: ", Tag: PUNCT Score: 0.9996064305305481
Text: Je, Tag: PRON Score: 0.9986742734909058
Text: l', Tag: PRON Score: 0.88944011926651
Text: Aime, Tag: PROPN Score: 0.9122504591941833
Text: a, Tag: AUX Score: 0.8670608401298523
Text: Mourir, Tag: VERB Score: 0.5136366486549377
Text: ", Tag: PUNCT Score: 0.9996033310890198
Text: -, Tag: PUNCT Score: 0.9998313188552856
Text: Francis, Tag: PROPN Score: 0.998817503452301
Text: Cabrel, Tag: PROPN Score: 0.9971158504486084
Text: ], Tag: PUNCT Score: 0.999731719493866

Italian
Text: ", Tag: PUNCT Score: 0.9999940395355225
Text: L', Tag: DET Score: 0.9964624047279358
Text: amor, Tag: NOUN Score: 0.9934152364730835
Text: che, Tag: PRON Score: 0.9955034852027893
Text: move, Tag: VERB Score: 0.9971327781677246
Text: il, Tag: DET Score: 0.9999856948852539
Text: sole, Tag: NOUN Score: 0.9965366125106812
Text: e, Tag: CCONJ Score: 0.9998295307159424
Text: l', Tag: DET Score: 0.999956488609314
Text: altre, Tag: ADJ Score: 0.7081868648529053
Text: stelle, Tag: NOUN Score: 0.9895918965339661
Text: ., Tag: PUNCT Score: 0.9992202520370483
Text: ", Tag: PUNCT Score: 0.9999991655349731
Text: [, Tag: PUNCT Score: 0.9999961853027344
```

```
Text: Quote, Tag: NOUN Score: 0.7765853404998779
Text: from, Tag: NOUN Score: 0.7075539827346802
Text: ", Tag: PUNCT Score: 0.9999961853027344
Text: Divine, Tag: PROPN Score: 0.5333973169326782
Text: Comedy, Tag: PROPN Score: 0.9428843259811401
Text: ", Tag: PUNCT Score: 0.9999252557754517
Text: -, Tag: PUNCT Score: 0.9996337890625
Text: Dante, Tag: PROPN Score: 0.9838617444038391
Text: Alighieri, Tag: PROPN Score: 0.9993308782577515
Text: ], Tag: PUNCT Score: 0.9999929666519165
```

The above is the output.

Question 1: In your words describe what syntaxes are.

> Syntax should be the individual keywords, symbols or match an element rule of a language. Each syntax element should be defined by the language or else it would be an error. Syntax errors for example in programming languages happen when no match is found.

AWS Rekognition

This section will explore some of the available image recognition services that AWS provides.

[6] Add images to a S3 bucket

The first step is to create a S3 bucket to store the images that will be used for testing.

```
cp ../lab8/createS3.py .
```

The current working directory is always named as the current lab. The above code will copy the createS3 python script from the previous. This will be used to create the S3 bucket.

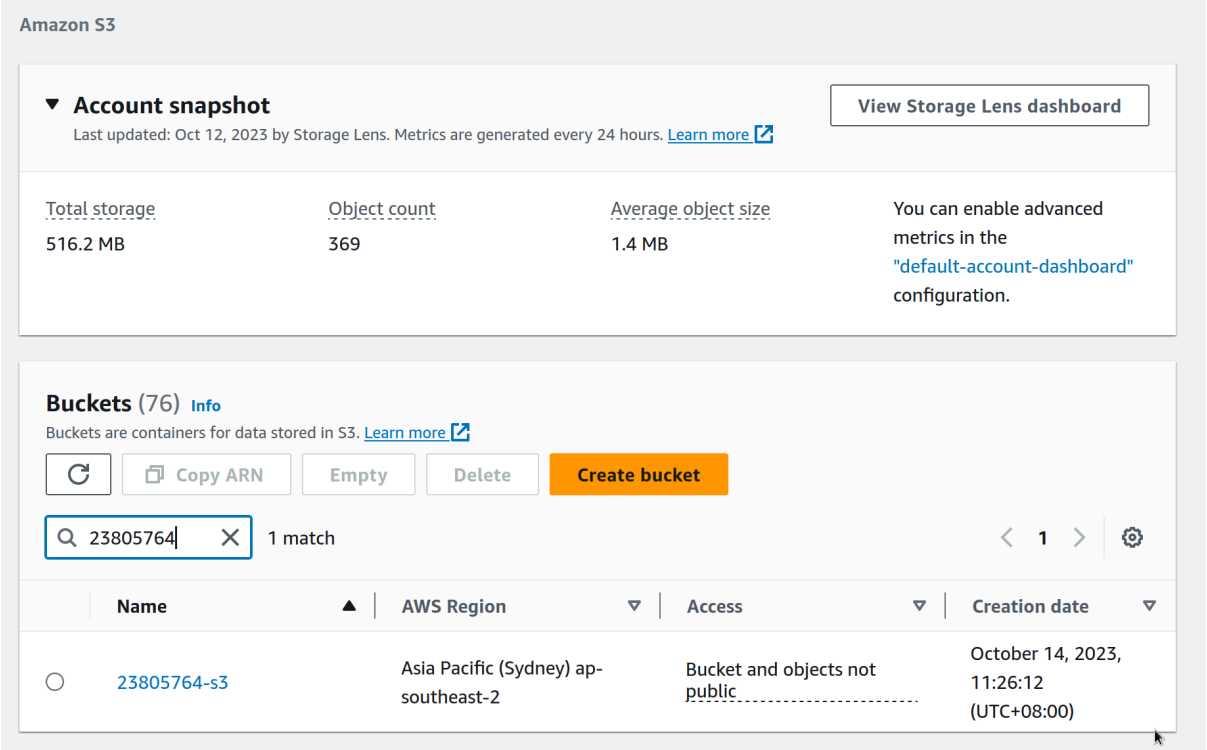
```
python3 createS3.py
```

Run the program.

```
{
      "ResponseMetadata": {
      "RequestId": "XZ32SVX1J5PF0MR9",
      "HostId":
"b1M5fjViHsYxAxPCIkA1m+fIHv+PNgsCaQPr5brDoIkxW0swojPsc1hU976Ub+lJTCsFJWy
iI8Q=",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
```

```
            "x-amz-id-2":
"b1M5fjViHsYxAxPCIkA1m+fIHv+PNgsCaQPr5brDoIkxW0swojPsc1hU976Ub+lJTCsFJWy
iI8Q=",
            "x-amz-request-id": "XZ32SVX1J5PF0MR9",
            "date": "Sat, 14 Oct 2023 03:26:11 GMT",
            "location": "http://23805764-s3.s3.amazonaws.com/",
            "server": "AmazonS3",
            "content-length": "0"
      },
      "RetryAttempts": 0
      },
      "Location": "http://23805764-s3.s3.amazonaws.com/"
}
```


The above output shows the S3 bucket information.



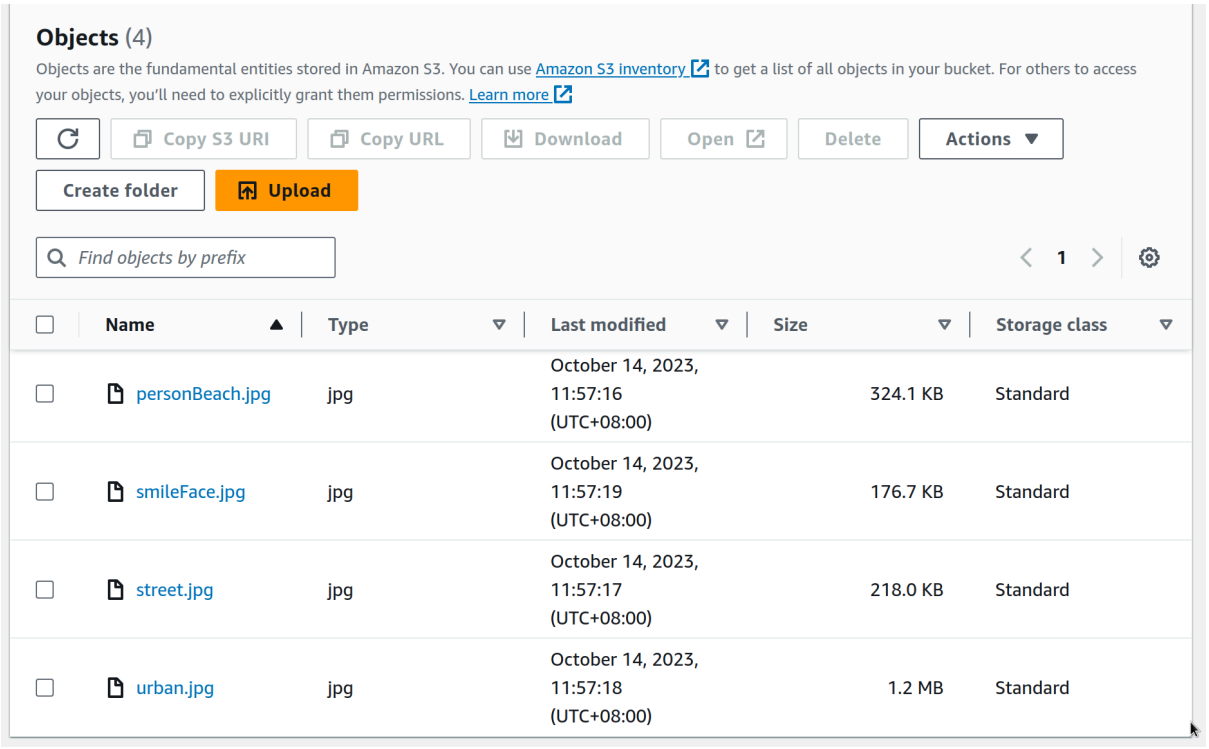
Above screenshot shows the newly created S3 bucket in the AWS console.

Four different types of images have to be selected and uploaded to the S3 bucket as per the lab specification. I will google the keywords and randomly pick some images. A script will be written to download the images locally then upload them to the S3 bucket.

```
import boto3
import subprocess
import os
```

```
dirName = "images"
s3_client = boto3.client("s3")
bucket_name = "23805764-s3"

if not os.path.isdir(dirName):
   os.mkdir(dirName)

subprocess.run(["wget", "-O", os.path.join(dirName, "urban.jpg"),
"https://archive.unews.utah.edu/wp-content/uploads/2021/04/Ewing-City-C
reek_edit.jpg"])
subprocess.run(["wget", "-O", os.path.join(dirName, "personBeach.jpg"),
"https://cdn.pixabay.com/photo/2015/08/12/10/20/person-885698_1280.jpg"
])
subprocess.run(["wget", "-O", os.path.join(dirName, "smileFace.jpg"),
"https://as1.ftcdn.net/v2/jpg/00/84/58/86/1000_F_84588657_bWFCviijaLzBA
Q5Yah2QkhMdBL8ueic5.jpg"])
subprocess.run(["wget", "-O", os.path.join(dirName, "street.jpg"),
"https://www.serviceobjects.com/blog/wp-content/uploads/2022/11/street-
name-route-66-in-typical-green-street-plates-in-flagstaff-picture-id141
5870181.jpg"])

for f in os.listdir(dirName):
   key_name = os.path.join(dirName, f)
   s3_client.upload_file(key_name, bucket_name, key_name)
```

The above is the content of a file called uploadFiles.py. The subprocess module and the shell command wget is used to first retrieve the 4 select images. Then all 4 images are uploaded to the s3 bucket using the upload_file function



The above screenshot shows that the 4 images are stored in the S3 bucket.

[7] Create script to perform corresponding Rekognition functions on the images

The Rekognition client from boto3 will be used to access these functionalities. There will be 4 different functions used on 4 different images. Therefore, 4 scripts are written where each performs a single function on a single image.

```
import boto3

bucket_name = "23805764-s3"
r_client = boto3.client("rekognition")

dl_response = r_client.detect_labels(
    Image={
        "S3Object": {
            "Bucket": bucket_name,
            "Name": "images/urban.jpg"
        }
    }
)

for label in dl_response["Labels"]:
    print(label["Name"])
```

Above is the content of the file called detectLabels.py. Using the urban image stored in the S3 bucket, item detection is performed. The labels detected will be printed in the end, the response it self is too large of a json object to view.

```
python3 detectLabel.py
```

Run the program.

```
City
Accessories
Bag
Handbag
Path
Architecture
Building
Person
Sidewalk
Helmet
Metropolis
Urban
Car
Transportation
Vehicle
Walkway
Bicycle
Clothing
Footwear
Shoe
Machine
Wheel
Plant
Jeans
Pants
Road
Glasses
Sandal
Cycling
Sport
Hat
Street
Intersection
Neighborhood
Flagstone
Pedestrian
```

The above is the output showing the labels detected in the image.

```
└─$ python3 detectLabels.py
City
Accessories
Bag
Handbag
Path
Architecture
Building
Person
```

The above is a screenshot of the first few labels.

The next step is to create a script to use the moderation functionality.

```
import boto3
import json

bucket_name = "23805764-s3"
r_client = boto3.client("rekognition")

dml_response = r_client.detect_moderation_labels(
    Image={
        "S3Object": {
            "Bucket": bucket_name,
            "Name": "images/personBeach.jpg"
        }
    }
)

print(json.dumps(dml_response, indent=4))
```

The above is the content of the file called detectModeration.py.

```
python3 detectModeration.py
```

Run the program.


```
{
      "ModerationLabels": [
      {
            "Confidence": 99.3395004272461,
            "Name": "Female Swimwear Or Underwear",
            "ParentName": "Suggestive"
      },
      {
```

```
            "Confidence": 99.3395004272461,
            "Name": "Suggestive",
            "ParentName": ""
      },
      {
            "Confidence": 92.88905334472656,
            "Name": "Revealing Clothes",
            "ParentName": "Suggestive"
      }
      ],
      "ModerationModelVersion": "6.1",
      "ResponseMetadata": {
      "RequestId": "0d2a450c-2d69-4a46-945d-2dd8dfaca339",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
            "x-amzn-requestid": "0d2a450c-2d69-4a46-945d-2dd8dfaca339",
            "content-type": "application/x-amz-json-1.1",
            "content-length": "303",
            "date": "Sat, 14 Oct 2023 04:17:40 GMT"
      },
      "RetryAttempts": 0
      }
}
```


The above is the output. It can be seen that few moderation labels are detected. Female underwear and revealing clothes are some of the moderation labels that are accurate on the image.

The next section is to write a script for facial analysis.

```
import boto3
import json

bucket_name = "23805764-s3"
r_client = boto3.client("rekognition")

df_response = r_client.detect_faces(
   Image={
       "S3Object": {
           "Bucket": bucket_name,
           "Name": "images/smileFace.jpg"
       }
   },
   Attributes=[
       "ALL"
   ]
```

```
)

print(json.dumps(df_response, indent=4))
```

The above is the content of the file called detectFaces.py.

```
python3 detectFaces.py
```

Run the program.


```
{
      "FaceDetails": [
      {
            "BoundingBox": {
                  "Width": 0.3619709610939026,
                  "Height": 0.5541731119155884,
                  "Left": 0.3205263018608093,
                  "Top": 0.13333918154239655
            },
            "AgeRange": {
                  "Low": 21,
                  "High": 29
            },
            "Smile": {
                  "Value": true,
                  "Confidence": 96.16567993164062
            },
            "Eyeglasses": {
                  "Value": false,
                  "Confidence": 92.82384490966797
            },
            "Sunglasses": {
                  "Value": false,
                  "Confidence": 99.99415588378906
            },
            "Gender": {
                  "Value": "Female",
                  "Confidence": 99.9963607788086
            },
            "Beard": {
                  "Value": false,
                  "Confidence": 95.17671966552734
            },
            "Mustache": {
                  "Value": false,
```

```
                "Confidence": 98.32809448242188
            },
            "EyesOpen": {
                "Value": true,
                "Confidence": 98.67451477050781
            },
            "MouthOpen": {
                "Value": true,
                "Confidence": 95.50548553466797
            },
            "Emotions": [
                {
                "Type": "HAPPY",
                "Confidence": 99.10562133789062
                },
                {
                "Type": "SURPRISED",
                "Confidence": 6.389232158660889
                },
                {
                "Type": "FEAR",
                "Confidence": 5.895319938659668
                },
                {
                "Type": "SAD",
                "Confidence": 2.1557252407073975
                },
                {
                "Type": "CONFUSED",
                "Confidence": 0.2557298541069031
                },
                {
                "Type": "ANGRY",
                "Confidence": 0.1286841183900833
                },
                {
                "Type": "DISGUSTED",
                "Confidence": 0.11922607570886612
                },
                {
                "Type": "CALM",
                "Confidence": 0.03622562810778618
                }
            ],
            "Landmarks": [
                {
```

```
"Type": "eyeLeft",
"X": 0.419589638710022,
"Y": 0.3599396049976349
},
{
"Type": "eyeRight",
"X": 0.5819530487060547,
"Y": 0.3702886402606964
},
{
"Type": "mouthLeft",
"X": 0.4250343143939972,
"Y": 0.536641001701355
},
{
"Type": "mouthRight",
"X": 0.5603702664375305,
"Y": 0.5452019572257996
},
{
"Type": "nose",
"X": 0.4968009889125824,
"Y": 0.45629096031188965
},
{
"Type": "leftEyeBrowLeft",
"X": 0.36098331212997437,
"Y": 0.3147438168525696
},
{
"Type": "leftEyeBrowRight",
"X": 0.4561255872249603,
"Y": 0.30759742856025696
},
{
"Type": "leftEyeBrowUp",
"X": 0.4095728397369385,
"Y": 0.29571041464805603
},
{
"Type": "rightEyeBrowLeft",
"X": 0.5486693382263184,
"Y": 0.3133572041988373
},
{
"Type": "rightEyeBrowRight",
```

```
"X": 0.6425445079803467,
"Y": 0.33234715461730957
},
{
"Type": "rightEyeBrowUp",
"X": 0.5960034728050232,
"Y": 0.30734094977378845
},
{
"Type": "leftEyeLeft",
"X": 0.3906918168067932,
"Y": 0.3573477864265442
},
{
"Type": "leftEyeRight",
"X": 0.45150676369667053,
"Y": 0.3632071614265442
},
{
"Type": "leftEyeUp",
"X": 0.41953808069229126,
"Y": 0.350669264793396
},
{
"Type": "leftEyeDown",
"X": 0.4199140965938568,
"Y": 0.3674773573875427
},
{
"Type": "rightEyeLeft",
"X": 0.549483597278595,
"Y": 0.369432270526886
},
{
"Type": "rightEyeRight",
"X": 0.6103456616401672,
"Y": 0.37123575806617737
},
{
"Type": "rightEyeUp",
"X": 0.5823659896850586,
"Y": 0.36097750067710876
},
{
"Type": "rightEyeDown",
"X": 0.5805153846740723,
```

```
"Y": 0.37763872742652893
},
{
"Type": "noseLeft",
"X": 0.4649542272090912,
"Y": 0.4752369225025177
},
{
"Type": "noseRight",
"X": 0.5246492028236389,
"Y": 0.47894537448883057
},
{
"Type": "mouthUp",
"X": 0.4929519593715668,
"Y": 0.5188669562339783
},
{
"Type": "mouthDown",
"X": 0.4904269576072693,
"Y": 0.5722848773002625
},
{
"Type": "leftPupil",
"X": 0.419589638710022,
"Y": 0.3599396049976349
},
{
"Type": "rightPupil",
"X": 0.5819530487060547,
"Y": 0.3702886402606964
},
{
"Type": "upperJawlineLeft",
"X": 0.32299086451530457,
"Y": 0.3583352565765381
},
{
"Type": "midJawlineLeft",
"X": 0.3477744460105896,
"Y": 0.5524060130119324
},
{
"Type": "chinBottom",
"X": 0.4860256016254425,
"Y": 0.6653945446014404
```

```
                  },
                  {
                  "Type": "midJawlineRight",
                  "X": 0.6333051323890686,
                  "Y": 0.5701171159744263
                  },
                  {
                  "Type": "upperJawlineRight",
                  "X": 0.6756014823913574,
                  "Y": 0.3802776634693146
                  }
            ],
            "Pose": {
                  "Roll": 2.2371764183044434,
                  "Yaw": -0.08805951476097107,
                  "Pitch": 2.7487003803253174
            },
            "Quality": {
                  "Brightness": 87.18608856201172,
                  "Sharpness": 96.61495208740234
            },
            "Confidence": 99.99940490722656,
            "FaceOccluded": {
                  "Value": false,
                  "Confidence": 99.57190704345703
            },
            "EyeDirection": {
                  "Yaw": -0.20080675184726715,
                  "Pitch": -7.404569625854492,
                  "Confidence": 99.96868133544922
            }
      }
      ],
      "ResponseMetadata": {
      "RequestId": "381ff75e-e68d-4e0e-a11c-82984d7c01dd",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
            "x-amzn-requestid": "381ff75e-e68d-4e0e-a11c-82984d7c01dd",
            "content-type": "application/x-amz-json-1.1",
            "content-length": "3506",
            "date": "Sat, 14 Oct 2023 04:27:48 GMT"
      },
      "RetryAttempts": 0
      }
}
```

The above is the output. From the above landmarks part of the json output, it can be seen that the nose, eye and mouth are detected. The person in the image is also identified as smiling, female and of the age 21 - 29, which is very accurate.

The last section is to make use of the text detection functionalities.

```
import boto3
import json

bucket_name = "23805764-s3"
r_client = boto3.client("rekognition")

dt_response = r_client.detect_text(
   Image={
       "S3Object": {
           "Bucket": bucket_name,
           "Name": "images/street.jpg"
       }
   }
)

for text in dt_response["TextDetections"]:
   print(f"DetectedText: {text['DetectedText']}, Confidence:
{text['Confidence']}")
```

The above is the content of the file detectText.py. The response is filtered to only print the text and the confidence.

```
python3 detectText.py
```

Run the program.

```
DetectedText: Route 66, Confidence: 98.86072540283203
DetectedText: Rt, Confidence: 98.80632781982422
DetectedText: St, Confidence: 99.40856170654297
DetectedText: ON, Confidence: 99.41295623779297
DetectedText: San Francisco, Confidence: 98.48822021484375
DetectedText: Route, Confidence: 98.28768920898438
DetectedText: 66, Confidence: 99.43375396728516
DetectedText: Rt, Confidence: 98.80632781982422
DetectedText: St, Confidence: 99.40856170654297
DetectedText: ON, Confidence: 99.41295623779297
DetectedText: San, Confidence: 99.5833969116211
DetectedText: Francisco, Confidence: 97.3930435180664
```

The above is the output. It can be seen that there is a repetition of the texts detected. “Route 66” is detected but there is also a detection of the two words separately.