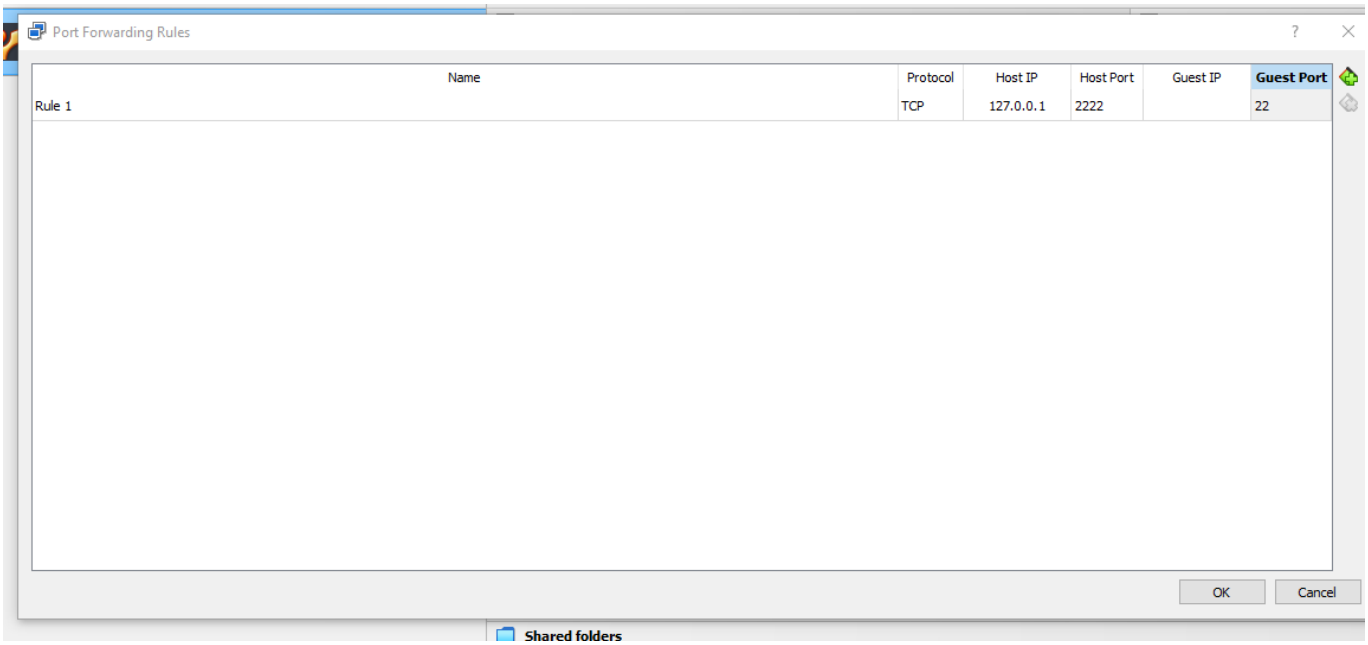
Lab Report 5 - 9

Author: Joo Kai Tay (22489437)

Lab 5: Networking

Section 1: Configure inbound IP on VM

1. Configure the network adapted in VirtualBox Manager using the rule: host IP 127.0.0.1 and host port 2222 mapped to Guest Port 22



2. Install tasksel and openssh-server

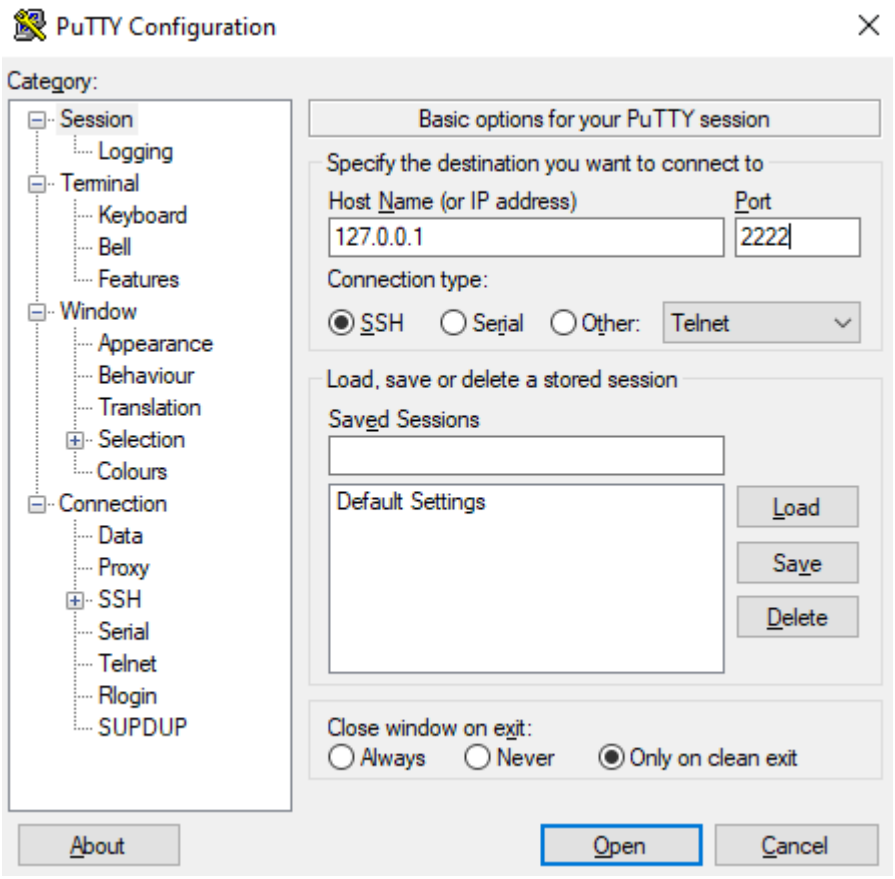
```
jookai@jookai:~$ sudo apt install tasksel
```

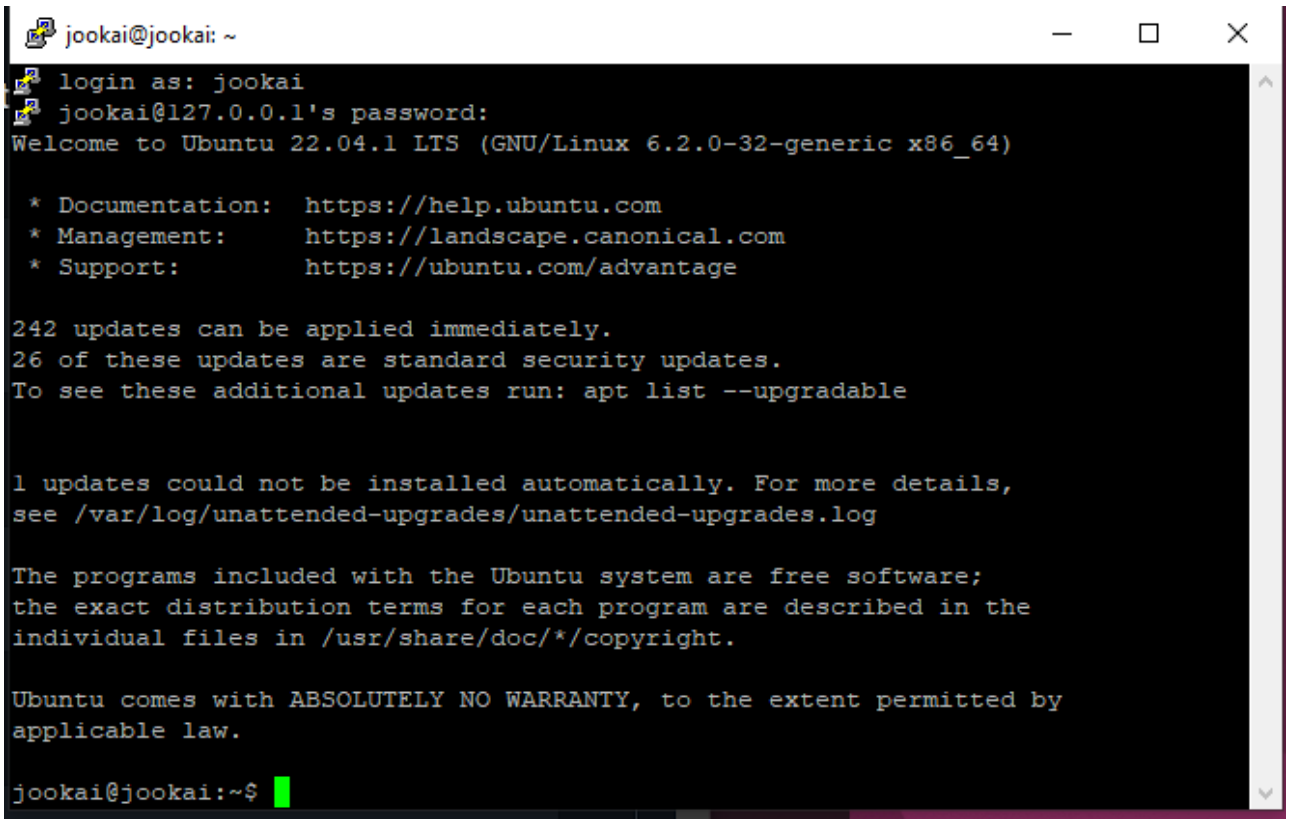
```
jookai@jookai:~$ sudo tasksel install openssh-server
```

3. Starting the ssh service on the ubuntu VM

```
jookai@jookai:~$ sudo service ssh start
```

4. SSH into the Ubuntu VM from the hostOS using Putty:





5. Terminate the SSH service:

```
jookai@jookai:~$ sudo service ssh stop
```

Section 2: Setting up an Application Load Balancer

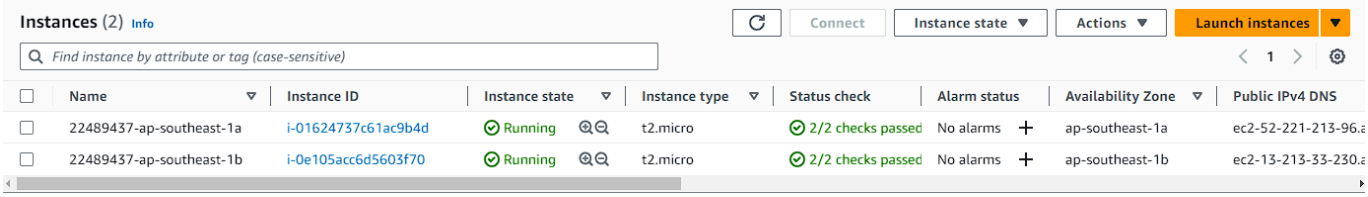
1. The following function is used to create 2 EC2 instances in two different availability zones of ap-southeast-1. The reason ap-southeast-1 was used instead of ap-southeast-2 was due to the limit in VPCUs on ap-southeast-2 which did not allow for any new EC2 instances to be created on the region at the time of attempting this lab.

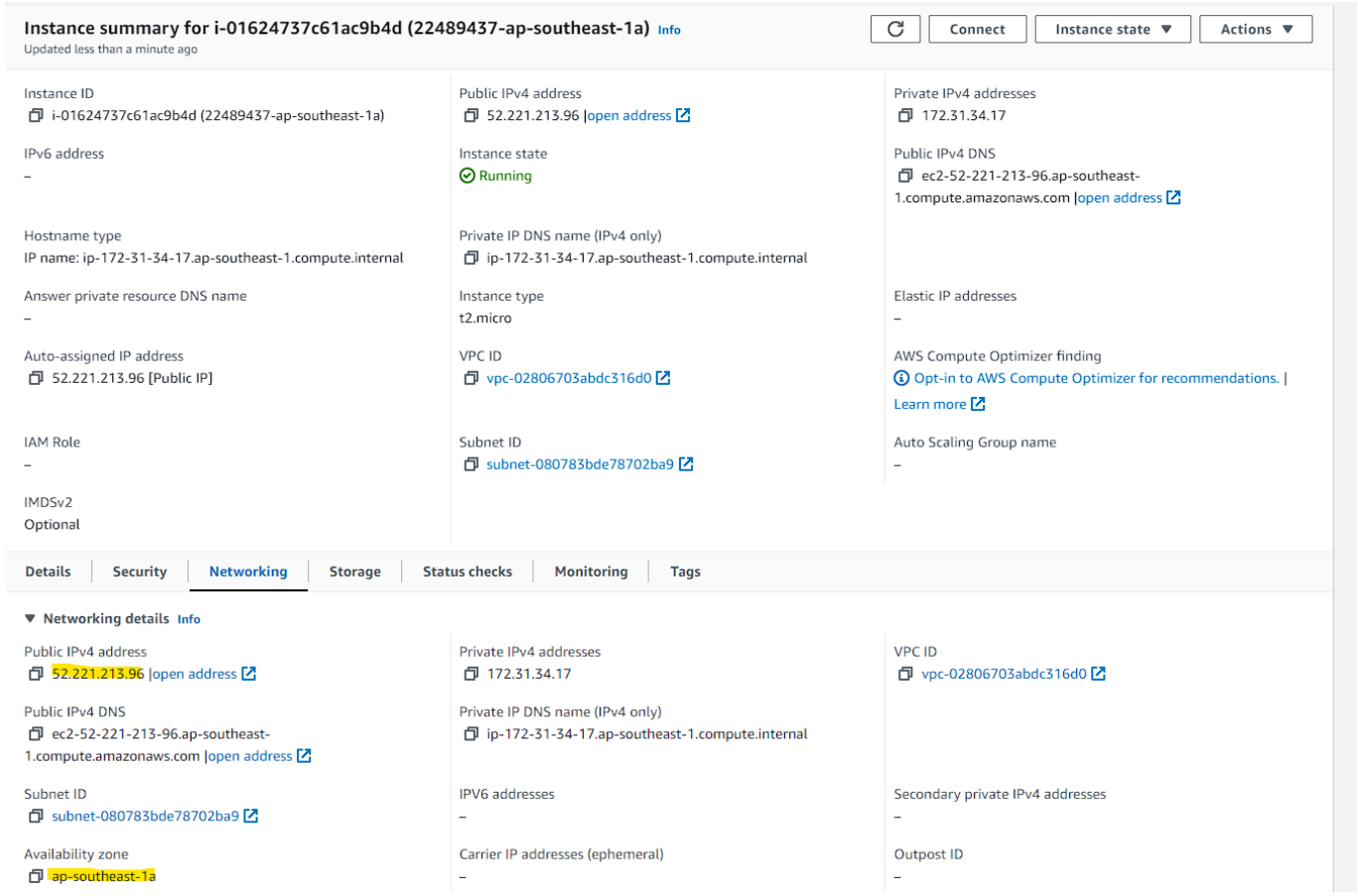
```
def launch_ec2_instances():
    # Create a security group
    response = ec2.create_security_group(
        GroupName=f"{student_number}-sg",
        Description="security group for development environment"
    )
    security_group_id = response['GroupId']

    # Authorize inbound SSH traffic for the security group
    ec2.authorize_security_group_ingress(
        GroupId=security_group_id,
        IpProtocol="tcp",
        FromPort=22,
        ToPort=22,
        CidrIp="0.0.0.0/0"
    )

    # Create a key pair and save the private key to a file
    response = ec2.create_key_pair(KeyName=f"{student_number}-key")
    private_key = response['KeyMaterial']
    private_key_file = f"{student_number}-key.pem"

    # Allow writing to the private key file
    os.chmod(private_key_file, 0o666)
    with open(private_key_file, 'w') as key_file:
        key_file.write(private_key)
    # Set the correct permissions for the private key file
    os.chmod(private_key_file, 0o400)
    # Copy the private key file to ~/.ssh directory
    ssh_directory = os.path.expanduser("~/.ssh")
    if not os.path.exists(ssh_directory):
        os.makedirs(ssh_directory)

    shutil.copy(private_key_file, ssh_directory)

    availability_zones = ["ap-southeast-1a", "ap-southeast-1b"]

    for i, az in enumerate(availability_zones):
        instance_name = f"{student_number}-{az}"

        instance_params = {
            'ImageId': 'ami-0df7a207adb9748c7',
            'InstanceType': 't2.micro',
            'KeyName': f"{student_number}-key",
            'SecurityGroupIds' : [security_group_id],
            'MinCount': 1,
            'MaxCount': 1,
```

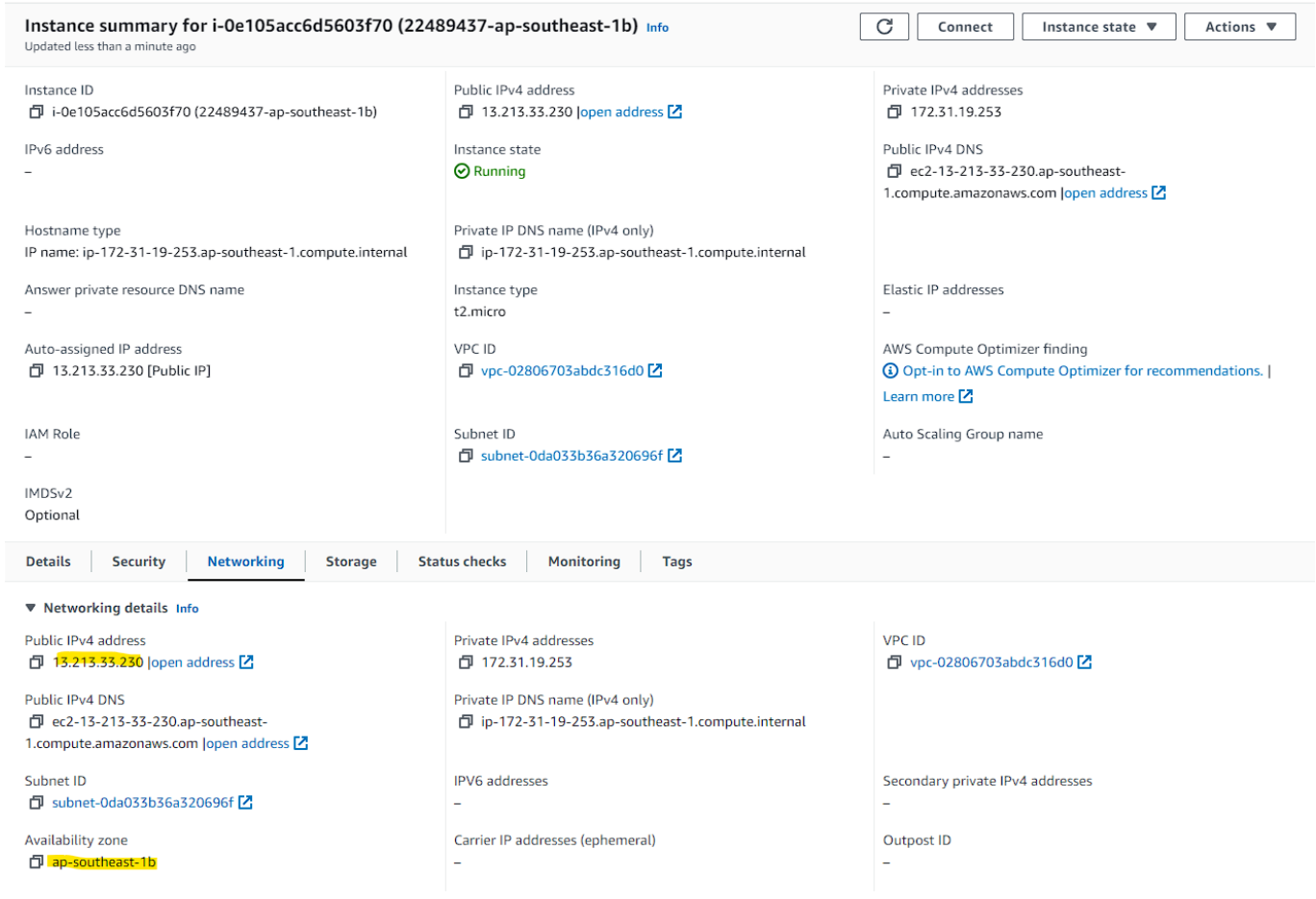
```
            'Placement': {'AvailabilityZone': az},
            'TagSpecifications': [
                {
                    'ResourceType': 'instance',
                    'Tags': [{'Key': 'Name', 'Value': instance_name}]
                }
            ]
        }

        # Launch an EC2 instance
        response = ec2.run_instances(**instance_params)

        instance_id = response['Instances'][0]['InstanceId']

        # Wait for the instance to be up and running
        ec2.get_waiter('instance_running').wait(InstanceIds=[instance_id])

        # Describe the instance to get its public IP address
        response = ec2.describe_instances(InstanceIds=[instance_id])
        public_ip_address = response['Reservations'][0]['Instances'][0]
['PublicIpAddress']

        print(f"Instance {i+1} created successfully in Availability Zone {az} with
Public IP: {public_ip_address}")
```

The created EC2 instances can be observed below. Note that the highlighted public IP addresses and availability zones in the AWS console correspond to the terminal output.

```
jookai@jookai:~/Desktop/cits5503/lab5$ python3 lab5.py -i
Instance 1 created successfully in Availability Zone ap-southeast-1a with Public
 IP: 52.221.213.96
Instance 2 created successfully in Availability Zone ap-southeast-1b with Public
 IP: 13.213.33.230
```







2. The code below creates an application load balancer. a. The code creates the load balancer and specifies the two region subnets retreived from step 1. b. The code creates a listener with a default rule

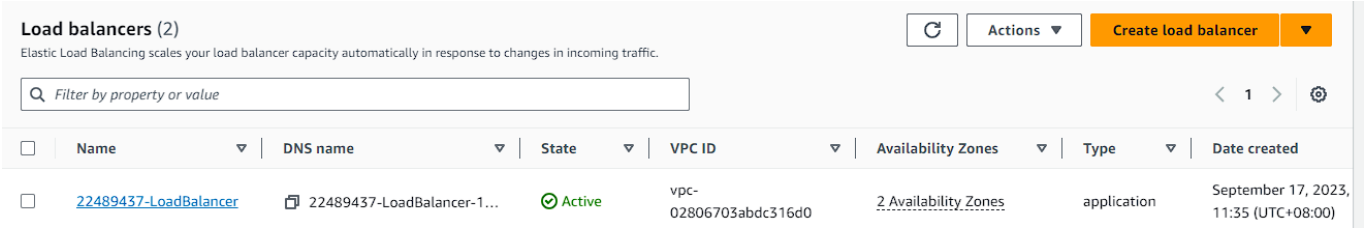
Protocol: HTTP and Port 80 forwarding on to the target group c. The code creates a target group using the VPC from step 1 d. The code registers the two EC2 instances from step 1 as targets

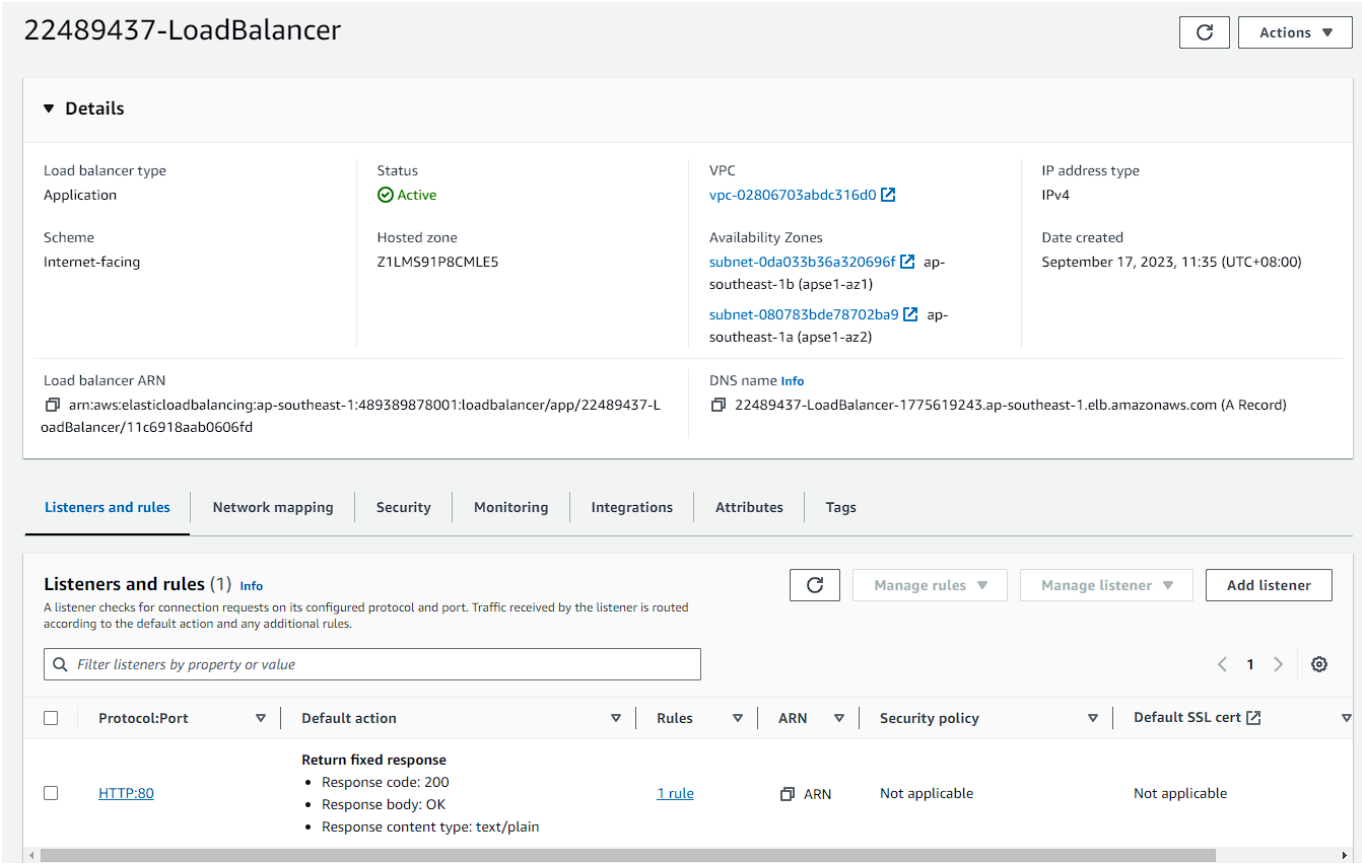
```
def create_load_balancer():
    vpc_id = 'vpc-02806703abdc316d0'
    security_group_id = 'sg-0021774194b407020'
    subnet_ids = ['subnet-080783bde78702ba9', 'subnet-0da033b36a320696f']

    response = elb.create_load_balancer(
        Name='22489437-LoadBalancer',
        Subnets=subnet_ids,
        SecurityGroups=[security_group_id],
        Scheme='internet-facing',
        Tags=[
            {
                'Key': 'Name',
                'Value': '22489437-LoadBalancer'
            },
        ]
    )

    load_balancer_arn = response['LoadBalancers'][0]['LoadBalancerArn']
    print(f"Load Balancer ARN: {load_balancer_arn}")

    # Create a target group
    response = elb.create_target_group(
        Name='22489437-target-group',
        Protocol='HTTP',
        Port=80,
        VpcId=vpc_id,
        TargetType='instance'
    )

    # Get the ARN of the target group
    target_group_arn = response['TargetGroups'][0]['TargetGroupArn']
    print(f"Target Group ARN: {target_group_arn}")

    # Create a listener for HTTP traffic (Port 80)
    response = elb.create_listener(
        DefaultActions=[
            {
                'Type': 'forward',
                'TargetGroupArn': target_group_arn,
            },
        ],
        LoadBalancerArn=load_balancer_arn,
        Port=80,
        Protocol='HTTP',
    )

    listener_arn = response['Listeners'][0]['ListenerArn']
    print(f"Listener ARN: {listener_arn}")
```

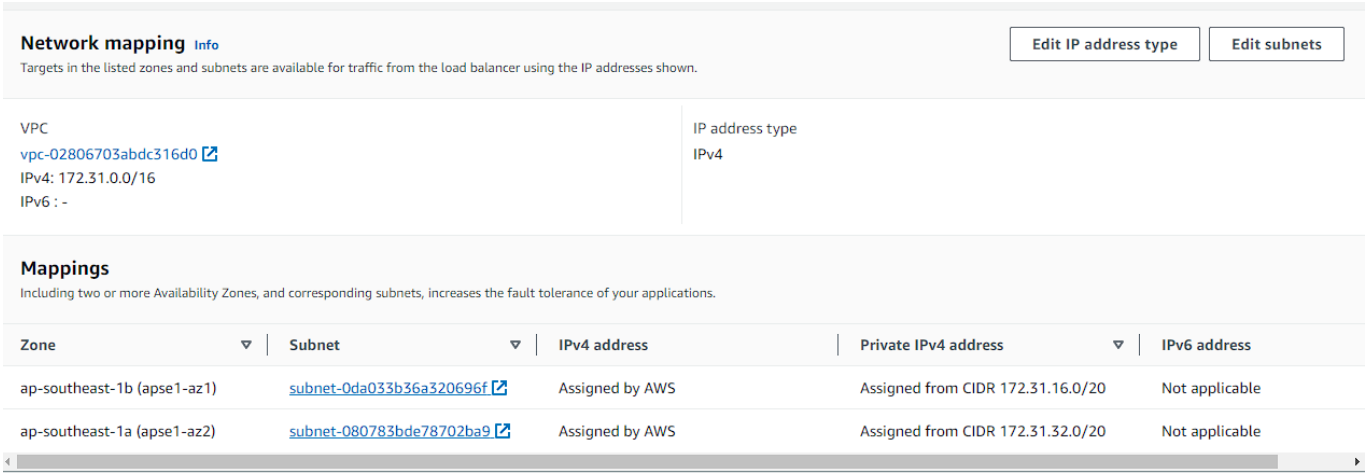
```
    instance_1_id = 'i-01624737c61ac9b4d'
    instance_2_id = 'i-0e105acc6d5603f70'

    # Register the instances in the target group
    elb.register_targets(
        TargetGroupArn=target_group_arn,
        Targets=[
            {'Id': instance_1_id},
            {'Id': instance_2_id},
        ]
    )

    # Print registration status
    print("Targets registered successfully.")
```

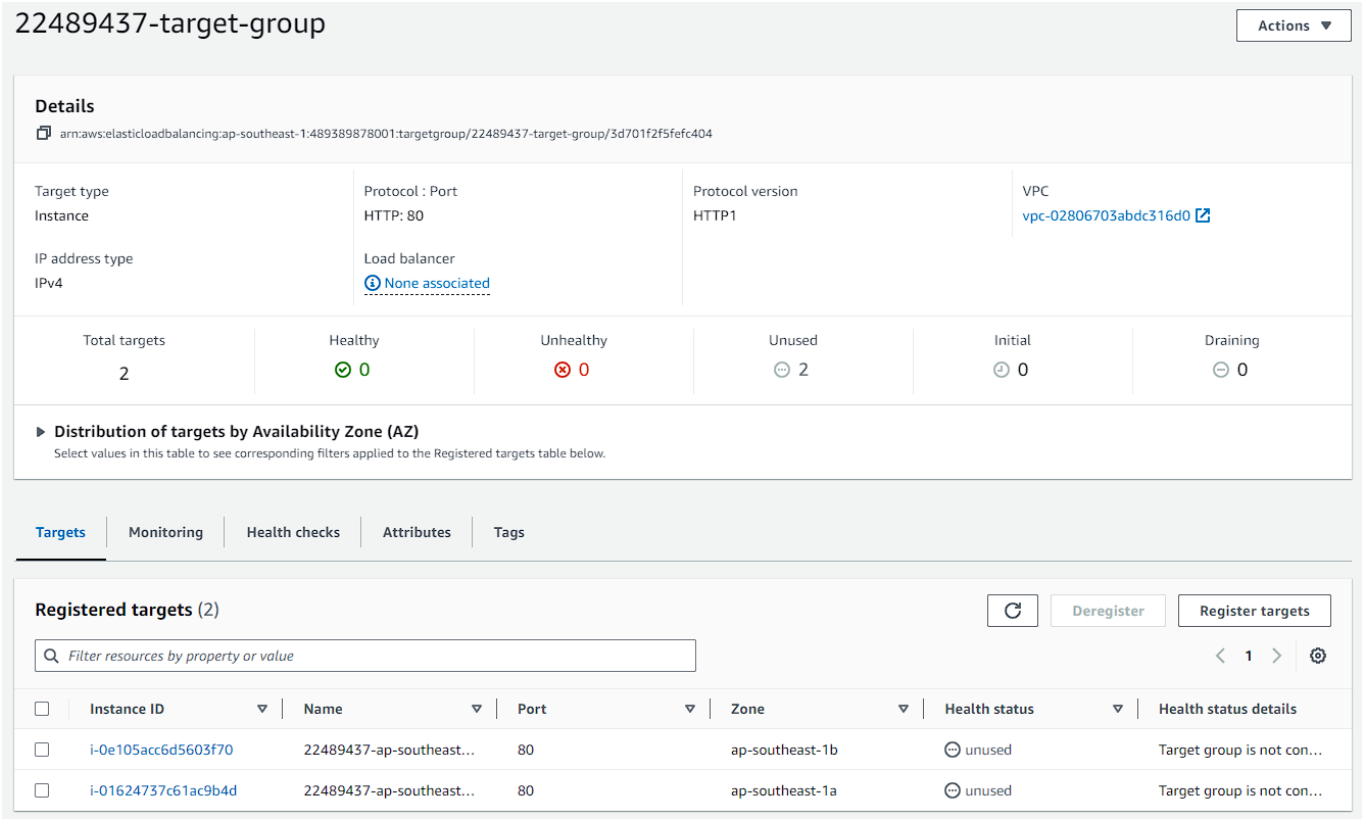
The following screenshots show the output of running the code as well as the results in the AWS terminal.

```
jookai@jookai:~/Desktop/cits5503/lab5$ python3 lab5.py -lb
Load Balancer ARN: arn:aws:elasticloadbalancing:ap-southeast-1:489389878001:load
balancer/app/22489437-LoadBalancer/11c6918aab0606fd
Listener ARN: arn:aws:elasticloadbalancing:ap-southeast-1:489389878001:listener/
app/22489437-LoadBalancer/11c6918aab0606fd/913934794eb2c296
Target Group ARN: arn:aws:elasticloadbalancing:ap-southeast-1:489389878001:targe
tgroup/22489437-target-group/3d701f2f5fefc404
Targets registered successfully.
```




22489437-LoadBalancer
Actions
Details
Load balancer type
Application
Status
Active
VPC
vpc-02806703abdc316d0
IP address type
IPv4
Scheme
Internet-facing
Hosted zone
Z1LMS91P8CMLE5
Availability Zones
subnet-0da033b36a320696f ap-southeast-1b (apse1-az1)
subnet-080783bde78702ba9 ap-southeast-1a (apse1-az2)
Date created
September 17, 2023, 11:35 (UTC+08:00)
Load balancer ARN
arn:aws:elasticloadbalancing:ap-southeast-1:489389878001:loadbalancer/app/22489437-LoadBalancer/11c6918aab0606fd
DNS name Info
22489437-LoadBalancer-1775619243.ap-southeast-1.elb.amazonaws.com (A Record)
Listeners and rules
Network mapping
Security
Monitoring
Integrations
Attributes
Tags
Listeners and rules (1) Info
A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.
Manage rules
Manage listener
Add listener
Filter listeners by property or value
Protocol:Port
Default action
Rules
ARN
Security policy
Default SSL cert
HTTP:80
Return fixed response
Response code: 200
Response body: OK
Response content type: text/plain
1 rule
ARN
Not applicable
Not applicable

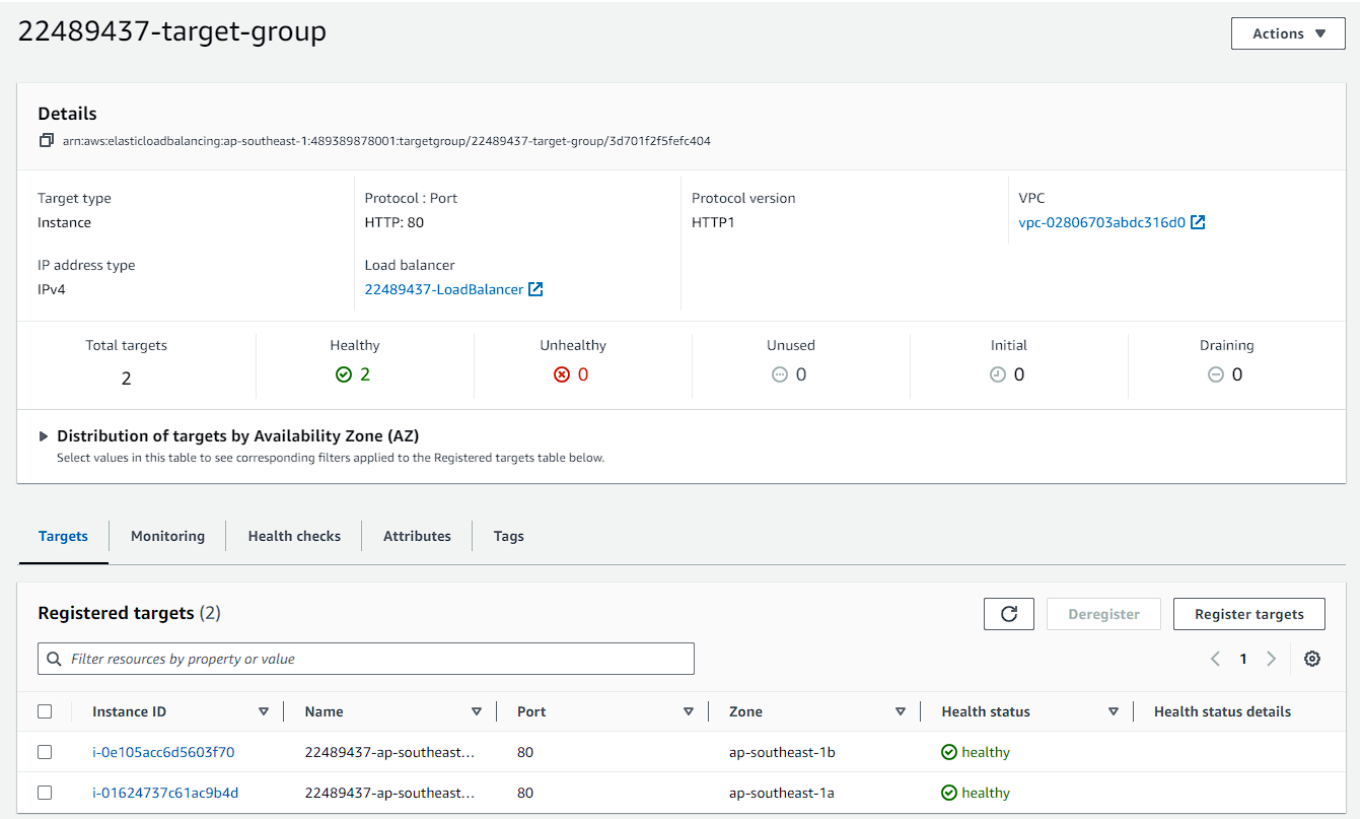

Network mapping Info
Targets in the listed zones and subnets are available for traffic from the load balancer using the IP addresses shown.
Edit IP address type
Edit subnets
VPC
vpc-02806703abdc316d0
IPv4: 172.31.0.0/16
IPv6 : -
IP address type
IPv4
Mappings
Including two or more Availability Zones, and corresponding subnets, increases the fault tolerance of your applications.
Zone
Subnet
IPv4 address
Private IPv4 address
IPv6 address
ap-southeast-1b (apse1-az1)
subnet-0da033b36a320696f
Assigned by AWS
Assigned from CIDR 172.31.16.0/20
Not applicable
ap-southeast-1a (apse1-az2)
subnet-080783bde78702ba9
Assigned by AWS
Assigned from CIDR 172.31.32.0/20
Not applicable



3. In this step, we will SSH into each of the instances created in step 1 and install Apache2. Screenshots showing this process for one of the EC2 instances have been attached:

```
jookai@jookai:~/.ssh$ ssh -i 22489437-key.pem ubuntu@52.221.213.96
The authenticity of host '52.221.213.96 (52.221.213.96)' can't be established.
ED25519 key fingerprint is SHA256:PHQL/z7oZ1klTmFoiao+rOjS708r4bLdfADQwiAlBIg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.221.213.96' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-1025-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Sep 17 03:50:30 UTC 2023

  System load:  0.0               Processes:             96
  Usage of /:   20.6% of 7.57GB   Users logged in:       0
  Memory usage: 24%               IPv4 address for eth0: 172.31.34.17
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-34-17:~$ S
```

```
ubuntu@ip-172-31-34-17:~$ sudo apt install apache2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils bzip2 libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.3-0 mailcap mime-support
  ssl-cert
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom www-browser
  bzip2-doc
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils bzip2 libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap liblua5.3-0 mailcap mime-support
  ssl-cert
0 upgraded, 13 newly installed, 0 to remove and 127 not upgraded.
Need to get 2137 kB of archives.
After this operation, 8505 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```



4. In this step we will edit the `/var/www/html/index.html` file to report the instance name and availability zone.

```
<!DOCTYPE html>
<html>
<body>
<h1>This is Instance 1 from availability zone ap-southeast-1a</h1>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>

<h1>This is instance 2 from availability zone ap-southeast-1b</h1>

</body>
</html>
```

5. By refreshing the page repeatedly, we can access both EC2 instances

Lab 6: Networking

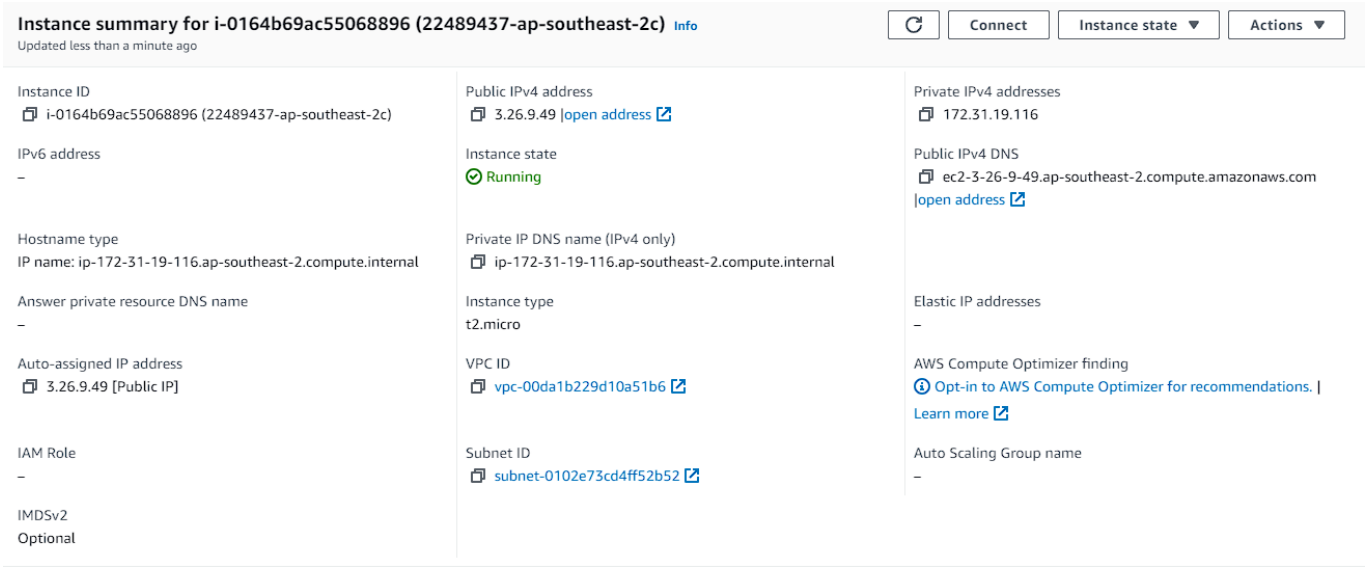
Section 1: Create an EC2 Instance

1. The code below was used to create an EC2 instance on ap-southeast-2c. For the lab this week, there was available capacity on ap-southeast-2 so there was no need to create the instance on another region. The AMI provided in lab 2 `ami-d38a4ab1` had a heavily outdated version of python and other utilities that are not compatible with modern programs, therefore an updated AMI was selected instead `ami-0310483fb2b488153`.

```
jookai@jookai:~/Desktop/cits5503/lab6$ python3 lab6.py -i
Instance 1 created successfully in Availability Zone ap-southeast-2c with Public
 IP: 3.26.9.49
```

```
def launch_ec2_instances():
    # Create a security group
    response = ec2.create_security_group(
        GroupName=f"{student_number}-sg",
        Description="security group for development environment"
    )
    security_group_id = response['GroupId']

    # Authorize inbound SSH traffic for the security group
    ec2.authorize_security_group_ingress(
        GroupId=security_group_id,
        IpProtocol="tcp",
        FromPort=22,
        ToPort=22,
        CidrIp="0.0.0.0/0"
    )

    ec2.authorize_security_group_ingress(
        GroupId=security_group_id,
        IpProtocol="tcp",
        FromPort=80,
        ToPort=80,
        CidrIp="0.0.0.0/0"
    )

    # Create a key pair and save the private key to a file
    response = ec2.create_key_pair(KeyName=f"{student_number}-key")
    private_key = response['KeyMaterial']
    private_key_file = f"{student_number}-key.pem"

    # Allow writing to the private key file
    os.chmod(private_key_file, 0o666)
    with open(private_key_file, 'w') as key_file:
        key_file.write(private_key)
    # Set the correct permissions for the private key file
    os.chmod(private_key_file, 0o400)
    # Copy the private key file to ~/.ssh directory
    ssh_directory = os.path.expanduser("~/.ssh")
    if not os.path.exists(ssh_directory):
        os.makedirs(ssh_directory)
```

```
    shutil.copy(private_key_file, ssh_directory)

    availability_zones = ["ap-southeast-2b", "ap-southeast-2c"]

    for i, az in enumerate(availability_zones):
        instance_name = f"{student_number}-{az}"

        instance_params = {
            'ImageId': 'ami-0310483fb2b488153',
            'InstanceType': 't2.micro',
            'KeyName': f"{student_number}-key",
            'SecurityGroupIds' : [security_group_id],
            'MinCount': 1,
            'MaxCount': 1,
            'Placement': {'AvailabilityZone': az},
            'TagSpecifications': [
                {
                    'ResourceType': 'instance',
                    'Tags': [{'Key': 'Name', 'Value': instance_name}]
                }
            ]
        }

        # Launch an EC2 instance
        response = ec2.run_instances(**instance_params)

        instance_id = response['Instances'][0]['InstanceId']

        # Wait for the instance to be up and running
        ec2.get_waiter('instance_running').wait(InstanceIds=[instance_id])

        # Describe the instance to get its public IP address
        response = ec2.describe_instances(InstanceIds=[instance_id])
        public_ip_address = response['Reservations'][0]['Instances'][0]
['PublicIpAddress']

        print(f"Instance {i+1} created successfully in Availability Zone {az} with
Public IP: {public_ip_address}")
```



2. Using the private key obtained and public IP address obtained from step 1, SSH into the EC2 instance and install the Python 3 virtual environment package.

```
jookai@jookai:~/.ssh$ ssh -i 22489437-key.pem ubuntu@3.26.9.49
The authenticity of host '3.26.9.49 (3.26.9.49)' can't be established.
ED25519 key fingerprint is SHA256:U83QWvm3/tMOx/A5pm0k9WqrPlr0CgYxUyQjKTAFdys.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.26.9.49' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-1025-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sat Sep 23 00:47:17 UTC 2023

  System load:  0.20166015625     Processes:             99
  Usage of /:   20.6% of 7.57GB   Users logged in:       0
  Memory usage: 24%               IPv4 address for eth0: 172.31.19.116
  Swap usage:   0%
```

```
ubuntu@ip-172-31-19-116:~$ sudo apt-get update
```

```
ubuntu@ip-172-31-19-116:~$ sudo apt-get upgrade
```

```
ubuntu@ip-172-31-19-116:~$ sudo apt-get install python3-venv
```

3. Creating a directory with path `/opt/wwc/mysites` and setting up the virtual environment.

```
root@ip-172-31-19-116:/home/ubuntu# sudo mkdir -p /opt/wwc/mysites
```

```
root@ip-172-31-19-116:/home/ubuntu# cd /opt/wwc/mysites
root@ip-172-31-19-116:/opt/wwc/mysites# python3 -m venv myvenv
root@ip-172-31-19-116:/opt/wwc/mysites# source myvenv/bin/activate
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites#
```

4. A Django project is a collection of configurations and apps for a particular website. In this step we install Django and create a new Django app named polls.
    - `lab`: The configuration directory
    - `polls`: The directory containing the app
    - `manage.py`: The command line utility that lets us interact with the new app

```
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites# pip install django
Collecting django
```

```
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites# django-admin startproject lab
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites# cd lab
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# python3 manage.py startapp
polls
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# ls
lab  manage.py  polls
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab#
```

Section 2: Install and Congigure Nginx

1. Installing and configuring nginx:
    - Nginx is a popular open-source web server software that can also be used as a reverse proxy, load balancer, mail proxy, and HTTP cache.
    - The configuration file is edited to tell Nginx to pass requests to the backend server running on the same machine 127.0.0.1 at port 8000.

```
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# apt install nginx
```

```
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# vi /etc/nginx/sites-enabled
/default
```

```
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;
  }
}
```

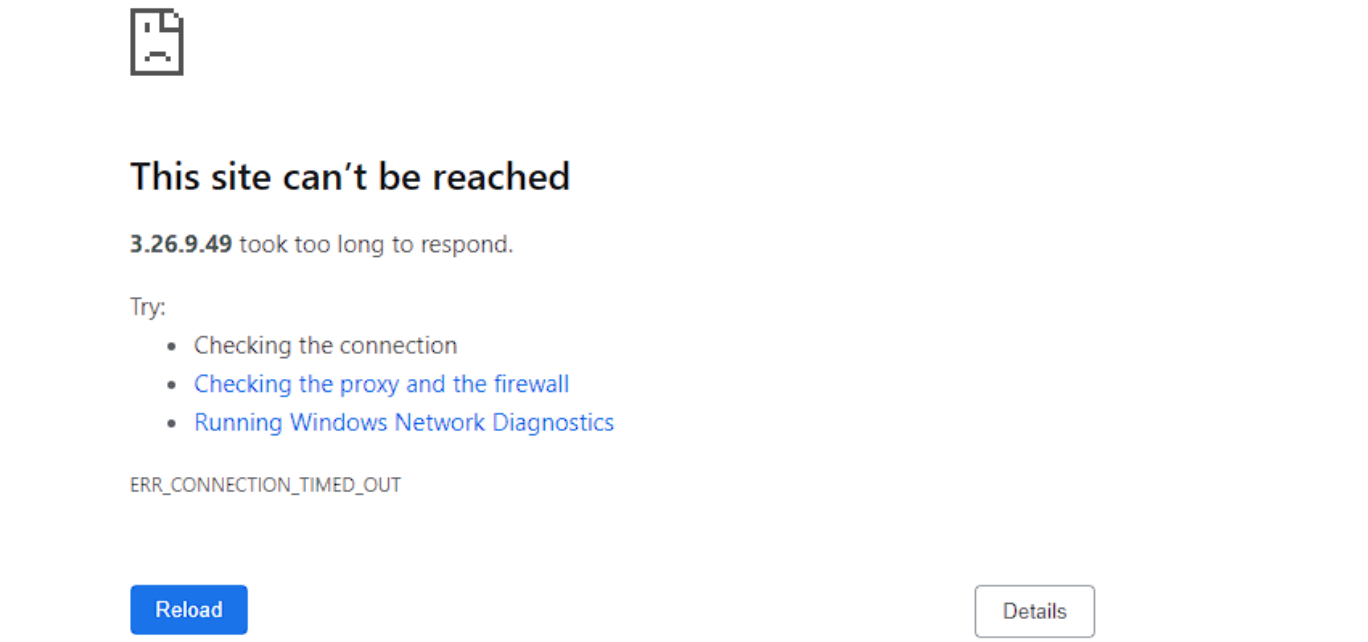
2. Restarting Nginx so that the changes from step 1 take effect:

```
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# service nginx restart
```

3. Using the command `python3 manage.py runserver 8000` to start Django's development web server at port 8000.

```
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# python3 manage.py runserver
 8000
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.
September 23, 2023 - 01:07:45
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.
```

4. Trying the access the public IP address of the EC2 instance results in an error:




```
Not Found: /polls/
[23/Sep/2023 08:14:39] "GET /polls/ HTTP/1.0" 404 2092
Not Found: /polls/
[23/Sep/2023 08:14:56] "GET /polls/ HTTP/1.0" 404 2092
```

Section 3: Change the code

1. Editing `polls/views.py`
    - Thsi code creates a simple view that returns an HTTP response with the text "Hello, world." when it's called.

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

def index(request):
    return HttpResponse("Hello, world.")
~
```

2. Edit `polls/urls.py`
    - This code defines a URL pattern for this view in the urls.py file, so that Django knows which view to call for a given URL.

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

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

3. Edit `lab/urls.py`
    - The code configures the URL patterns for the Django project.

```
URL configuration for lab project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import include, path
from django.contrib import admin

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
-- INSERT --                                                    24,2          Bot
```

4. Running the application and getting `Hello, world`

```
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# vi polls/views.py
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# vi polls/urls.py
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# vi lab/urls.py
(myvenv) root@ip-172-31-19-116:/opt/wwc/mysites/lab# python3 manage.py runserver
 8000
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.
September 23, 2023 - 01:31:53
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.
```

Hello, world.

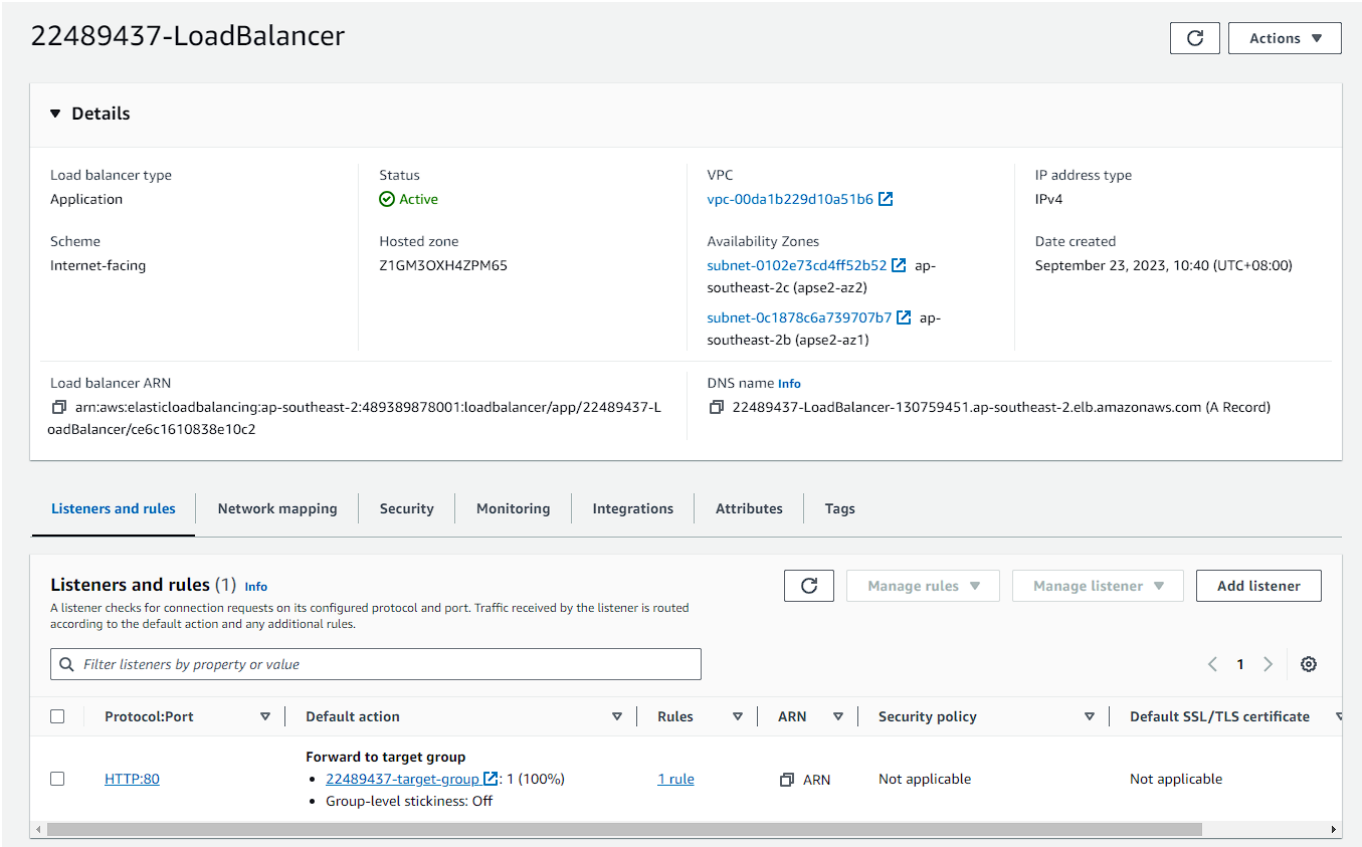
Section 4: Adding an application load balance (ALB)

1. The code below creates an application load balancer, specifies the region subnet where the EC2 instance resides, creates a listener with a default rule Protocol: HTTP and Port 80 forwarding.

```
def create_load_balancer():
    vpc_id = 'vpc-00da1b229d10a51b6'
    security_group_id = 'sg-0fb8992bd2473b7bc'
    subnet_ids = ['subnet-0c1878c6a739707b7', 'subnet-0102e73cd4ff52b52']

    response = elb.create_load_balancer(
        Name='22489437-LoadBalancer',
        Subnets=subnet_ids,
        SecurityGroups=[security_group_id],
        Scheme='internet-facing',
        Tags=[
            {
                'Key': 'Name',
                'Value': '22489437-LoadBalancer'
            },
        ]
    )

    load_balancer_arn = response['LoadBalancers'][0]['LoadBalancerArn']
    print(f"Load Balancer ARN: {load_balancer_arn}")

    # Create a target group
    response = elb.create_target_group(
        Name='22489437-target-group',
        Protocol='HTTP',
        Port=80,
        VpcId=vpc_id,
        HealthCheckProtocol='HTTP',
```

```
        HealthCheckPort='80',
        HealthCheckPath='/polls/',
        HealthCheckIntervalSeconds=30,
        HealthCheckTimeoutSeconds=5,
        HealthyThresholdCount=5,
        UnhealthyThresholdCount=2,
        Matcher={
            'HttpCode': '200'
        },
        TargetType='instance'
    )

    # Get the ARN of the target group
    target_group_arn = response['TargetGroups'][0]['TargetGroupArn']
    print(f"Target Group ARN: {target_group_arn}")

    # Create a listener for HTTP traffic (Port 80)
    response = elb.create_listener(
        DefaultActions=[
            {
                'Type': 'forward',
                'TargetGroupArn': target_group_arn,
            },
        ],
        LoadBalancerArn=load_balancer_arn,
        Port=80,
        Protocol='HTTP',
    )

    listener_arn = response['Listeners'][0]['ListenerArn']
    print(f"Listener ARN: {listener_arn}")


    instance_1_id = 'i-0164b69ac55068896'

    # Register the instances in the target group
    elb.register_targets(
        TargetGroupArn=target_group_arn,
        Targets=[
            {'Id': instance_1_id},
        ]
    )

    # Print registration status
    print("Targets registered successfully.")
```

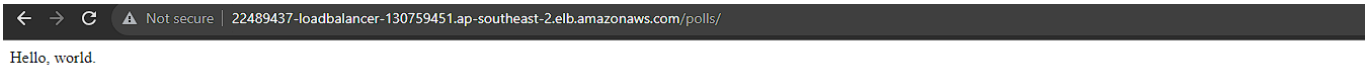
```
jookai@jookai:~/Desktop/cits5503/lab6$ python3 lab6.py -lb
Load Balancer ARN: arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:load
balancer/app/22489437-LoadBalancer/ce6c1610838e10c2
Target Group ARN: arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:targe
tgroup/22489437-target-group/1448f7d6495f1fac
Listener ARN: arn:aws:elasticloadbalancing:ap-southeast-2:489389878001:listener/
app/22489437-LoadBalancer/ce6c1610838e10c2/e43f09b70777fe3d
Targets registered successfully.
```



2. Viewing the health check on the `/polls/` page

```
[23/Sep/2023 08:09:55] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:10:08] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:10:25] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:10:39] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:10:55] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:11:09] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:11:26] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:11:39] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:11:56] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:12:09] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:12:26] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:12:39] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:12:56] "GET /polls/ HTTP/1.0" 200 13
[23/Sep/2023 08:13:09] "GET /polls/ HTTP/1.0" 200 13
```

3. Accessing the site using the url:



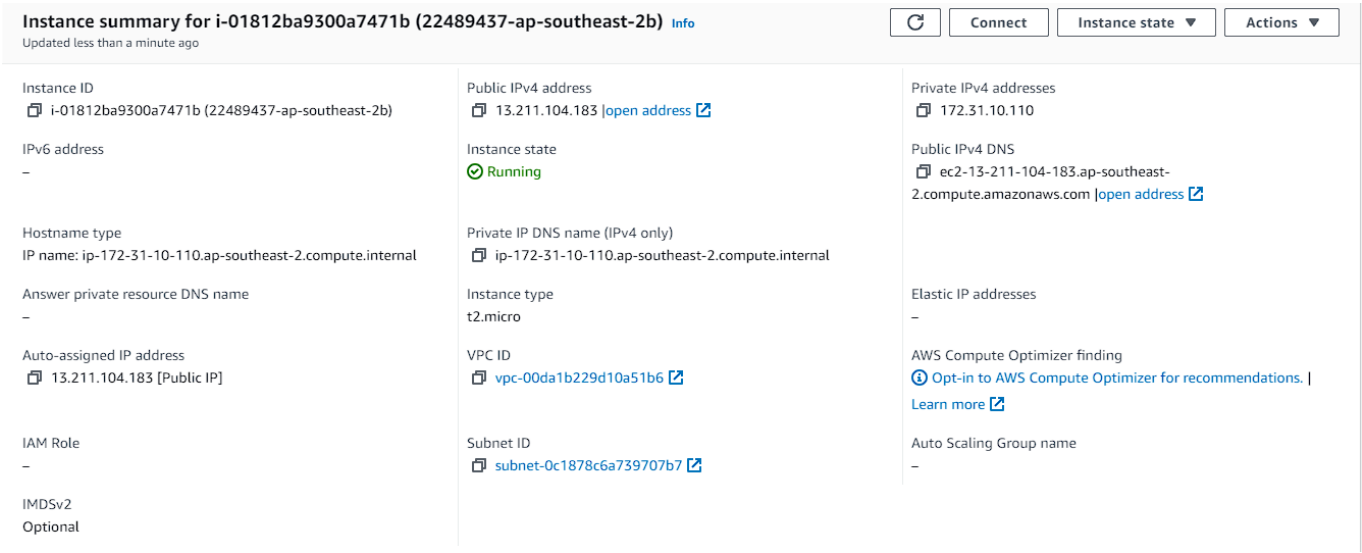
Hello, world.

Lab 7: DevOps

Step 1: Create an EC2 Instance

1. The code used in Labs 5 and 6 was used to create an EC2 instance in availability zone ap-southeast-2b

```
jookai@jookai:~/Desktop/cits5503/lab7$ python3 lab7.py -i
Instance 1 created successfully in Availability Zone ap-southeast-2b with Public
 IP: 13.211.104.183
```



Step 2: Install Fabric

1. Fabric is a Python library used to perform Linux shell commands remotely over SSH. It is a command-line tool that uses SSH for application deployment or for administration tasks. In this lab we will be using fabric to automate the deployment of a django app on the remote EC2 instance.

```
jookai@jookai:~/Desktop/cits5503/lab7$ pip install fabric
Defaulting to user installation because normal site-packages is not writeable
Collecting fabric
  Downloading fabric-3.2.2-py3-none-any.whl (59 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.4/59.4 KB 2.0 MB/s eta 0:00:00
Requirement already satisfied: paramiko>=2.4 in /usr/lib/python3/dist-packages (
from fabric) (2.9.3)
Collecting invoke>=2.0
  Downloading invoke-2.2.0-py3-none-any.whl (160 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 160.3/160.3 KB 5.0 MB/s eta 0:00:00
Collecting deprecated>=1.2
  Downloading Deprecated-1.2.14-py2.py3-none-any.whl (9.6 kB)
Collecting decorator>=5
  Downloading decorator-5.1.1-py3-none-any.whl (9.1 kB)
Collecting wrapt<2,>=1.10
  Downloading wrapt-1.15.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.ma
nylinux_2_17_x86_64.manylinux2014_x86_64.whl (78 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.4/78.4 KB 5.2 MB/s eta 0:00:00
Installing collected packages: wrapt, invoke, decorator, deprecated, fabric
Successfully installed decorator-5.1.1 deprecated-1.2.14 fabric-3.2.2 invoke-2.2
.0 wrapt-1.15.0
```

The configuration file located in `~/.ssh` contains the IPV4 DNS of the EC2 instance and the path of the private key file on the local VM to allow SSH access into the EC2 instance.

```
Host <ec2instance>
        Hostname ec2-13-211-104-183.ap-southeast-2.compute.amazonaws.com
        User ubuntu
        UserKnownHostsFile /dev/null
        StrictHostKeyChecking no
        PasswordAuthentication no
        IdentityFile ~/.ssh/22489437-key.pem
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"config" [New File]
```

2. Testing Fabric

```
jookai@jookai:~/.ssh$ python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from fabric import Connection
>>> c = Connection('<ec2instance>')
>>> result = c.run('uname -s')
Linux
>>>
```

Step 3: Python Script to automate installation of nginx

1. The following script is used to automate the instllation of nginx on the EC2 instance.

```
def create_nginx():
    with Connection('<ec2instance>') as c:
        c.sudo('apt update -y')
        c.sudo('apt upgrade -y')
        c.sudo('apt install nginx -y')
        c.sudo('rm /etc/nginx/sites-enabled/default')
        os.system('scp -i ~/.ssh/22489437-key.pem default ubuntu@ec2-13-211-104-
183.ap-southeast-2.compute.amazonaws.com:/home/ubuntu')
        c.sudo('mv /home/ubuntu/default /etc/nginx/sites-enabled/')
        c.sudo('service nginx restart')
```

The script is run on the local VM:

```
jookai@jookai:~/Desktop/cits5503/lab7$ python3 lab7.py -n

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

Checking if nginx is installed on the EC2 instance:

```
ubuntu@ip-172-31-10-110:~$ nginx -v
nginx version: nginx/1.18.0 (Ubuntu)
```

Step 4: Python script to install Django app

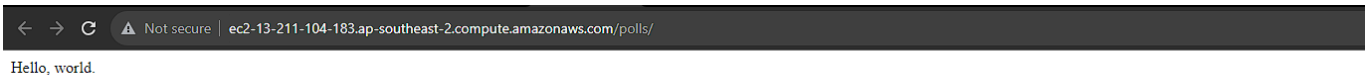
1. The following script is used to automate the installation of the Django app on the EC2 instanc:

```
def create_django_app():
    with Connection('<ec2instance>') as c:
        c.sudo('mkdir -p /opt/wwc/mysites')
        c.sudo('apt-get install python3-pip -y')
        c.sudo('apt install python3-django -y')
        c.sudo('pip3 install django')
        c.run('django-admin startproject lab')
        c.sudo('chmod 777 ./lab')
        c.run('cd ./lab && python3 manage.py startapp polls')
        c.sudo('mv ./lab /opt/wwc/mysites')
        c.sudo('rm /opt/wwc/mysites/lab/polls/views.py')
        c.sudo('rm /opt/wwc/mysites/lab/lab/urls.py')
        os.system('scp -i ~/.ssh/22489437-key.pem ./polls/views.py ubuntu@ec2-13-
211-104-183.ap-southeast-2.compute.amazonaws.com:/opt/wwc/mysites/lab/polls')
        os.system('scp -i ~/.ssh/22489437-key.pem ./polls/urls.py ubuntu@ec2-13-
211-104-183.ap-southeast-2.compute.amazonaws.com:/opt/wwc/mysites/lab/polls')
        os.system('scp -i ~/.ssh/22489437-key.pem ./lab/urls.py ubuntu@ec2-13-211-
104-183.ap-southeast-2.compute.amazonaws.com:/opt/wwc/mysites/lab/lab')
        c.run('cd /opt/wwc/mysites/lab && python3 manage.py runserver 8000')
```

The script is run on the local VM:

```
jookai@jookai:~/Desktop/cits5503/lab7$ python3 lab7.py -d
views.py                                      100%   98     1.8KB/s   00:00
urls.py                                       100%  108     2.0KB/s   00:00
urls.py                                       100%  169     2.8KB/s   00:00
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.
September 29, 2023 - 14:07:35
Django version 3.2.12, using settings 'lab.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Checking to see if `Hello world` is displayed when visiting the Public IPv4 DNS of the EC2 instance:



Lab 8: AI

Step 1: Install and run jupyter notebooks

1. Jupyter notebook was already installed on this machine for CITS5508 machine learning. Therefore, only the command to launch it was needed.

```
jookai@jookai:~/Desktop/cits5503/lab8$ jupyter notebook
```

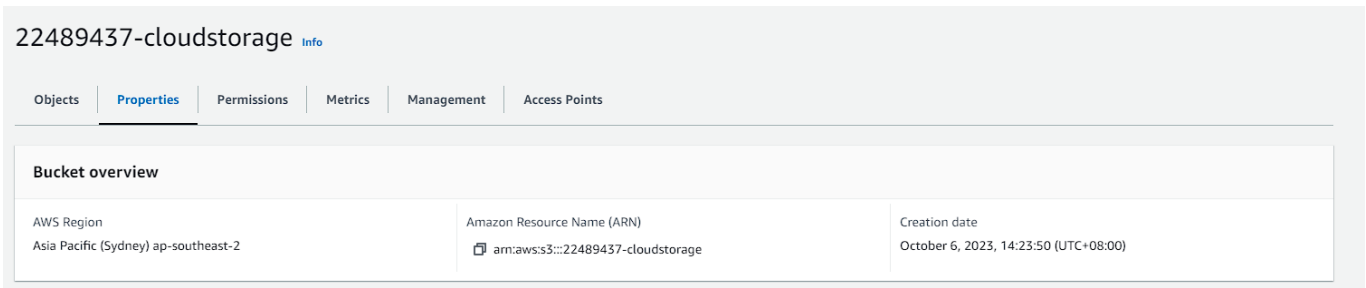
2. This brings up Jupyter notebook in the browser where we can create a notebook for this lab named `lab8.ipynb` which is an Interactive Python Notebook.

Step 2: Setup Python Environment

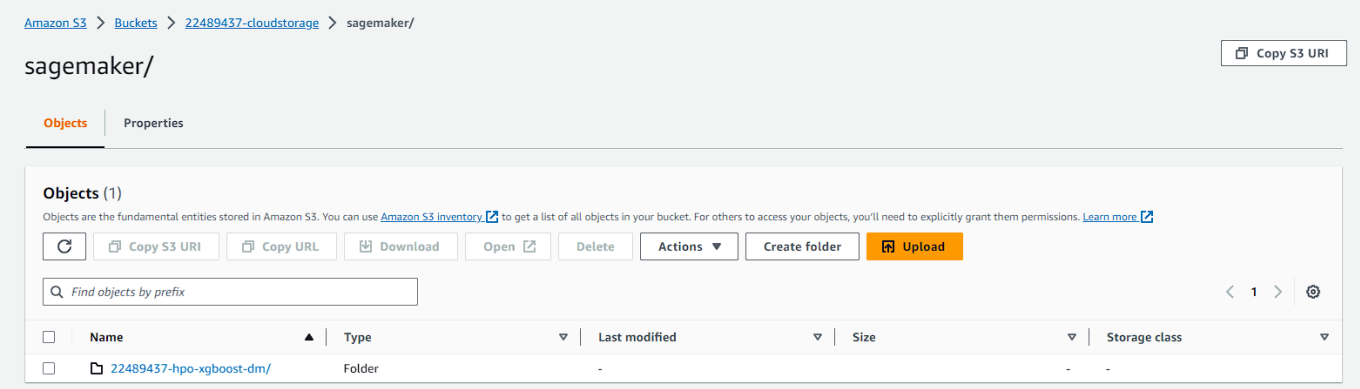
1. The following command is run `pip3 install sagemaker pandas ipykernel`.
    - This installs sagemaker, which is the Python SDK for Amazon SageMaker, that allows us to build, train, and deploy machine learning models. By installing this SDK, we can interact with the SageMaker service directly from the Python environment on the virtual machine.
    - Pandas is a popular data manipulation and analysis library for Python.
    - Ipykernel allows the python environment to interface with Jupyrer.

Step 3: Create Role and Bucket

1. Creating a S3 bucket to store the files generated in the later parts of the machine learning process:



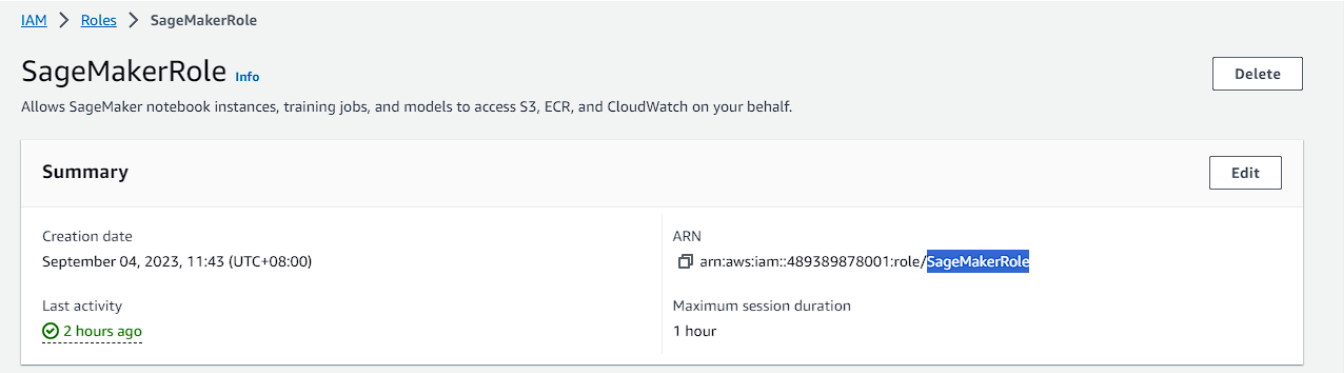
2. Creating the folders `sagemaker/22489437-hpo-xgboost-dm`



3. Due to security reasons, students are not allowed to directly create IAM roles. Therefore, we will use the code provided in the labsheet to use the already created IAM role.
    - `smclient = boto3.Session().client("sagemaker")`: This initializes a low-level client representing Amazon SageMaker. So, smclient will be used to make requests to the SageMaker service.
    - `sagemaker_role = iam.get_role(RoleName='Role_AWS_SageMaker')['Role']['Arn']`: This uses the IAM client to retrieve the ARN of a specific IAM role named 'Role_AWS_SageMaker'. This ARN can later be used to grant permissions or interact with the role in various ways within AWS.

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

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

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

4. The code provided fails to execute as the role `Role_AWS_SageMaker` does not exist. Therefore, we log into the IAM system in the AWS console and find that the actual name of the created roles is `SageMakerRole`. When using the actual name of the role, the code executes and we can proceed to the next step.



Step 4: Download Dataset

1. We download and extract the direct marketing dataset from UCI's ML Repository.

```
jookai@jookai:~/Desktop/cits5503/lab8$ wget -N https://archive.ics.uci.edu/ml/mac
hine-learning-databases/00222/bank-additional.zip
--2023-10-06 14:54:17--  https://archive.ics.uci.edu/ml/machine-learning-database
s/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... co
nnected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘bank-additional.zip’

bank-additional.zip     [      <=>           ] 434.15K   419KB/s    in 1.0s

Last-modified header missing -- time-stamps turned off.
2023-10-06 14:54:19 (419 KB/s) - ‘bank-additional.zip’ saved [444572]

jookai@jookai:~/Desktop/cits5503/lab8$ unzip -o bank-additional.zip
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 dataset is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed.

2. We read the dataset into a Pandas dataframe and inspect the contents:
    - We see that the data fame has 41188 rows and 21 columns

```
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
```

```
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
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp.v
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

3. In order to answer which variables are categorical and numerical, the following function was used:
    - This makes use of the `select_dtypes` method to filter out the categorical and numeric columns

```
# numeric columns
numeric_cols = data.select_dtypes(include=['number']).columns.tolist()
print("Numeric columns:", numeric_cols)

# categorical columns
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
print("Categorical columns:", categorical_cols)
```

```
# numeric columns
numeric_cols = data.select_dtypes(include=['number']).columns.tolist()
print("Numeric columns:", numeric_cols)

# categorical columns
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
print("Categorical columns:", categorical_cols)
```

```
Numeric columns: ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.emp
loyed']
Categorical columns: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome', 'y']
```

4. The following code performs some transformations on the data:
    - `data["no_previous_contact"] = np.where(data["pdays"] == 999, 1, 0)`: This line creates a new column in the DataFrame data named "no_previous_contact". The values in this column are determined by the values in the "pdays" column. If a value in "pdays" is 999, the new column gets a value of 1; otherwise, it gets 0.
    - `data["not_working"] = np.where(np.in1d(data["job"], ["student", "retired", "unemployed"]), 1, 0)`: This line creates another new column named "not_working". It checks the "job" column of the DataFrame. If the job description is either "student", "retired", or "unemployed", the new column gets a value of 1, indicating the person is not actively employed. Otherwise, it gets 0.
    - `model_data = pd.get_dummies(data)`: This line creates a new DataFrame, model_data, by converting the categorical variables in data into "indicator" variables. For each level in a categorical variable, a new column will be created to indicate if it is present or not.

```
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.in1d(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
```

This creates the `model_data` dataframe that has 67 columns.

[40]:

	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
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

5. The following code removes the the economic features and duration from the data as they would need to be forecasted with high precision to use as inputs in future predictions.

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

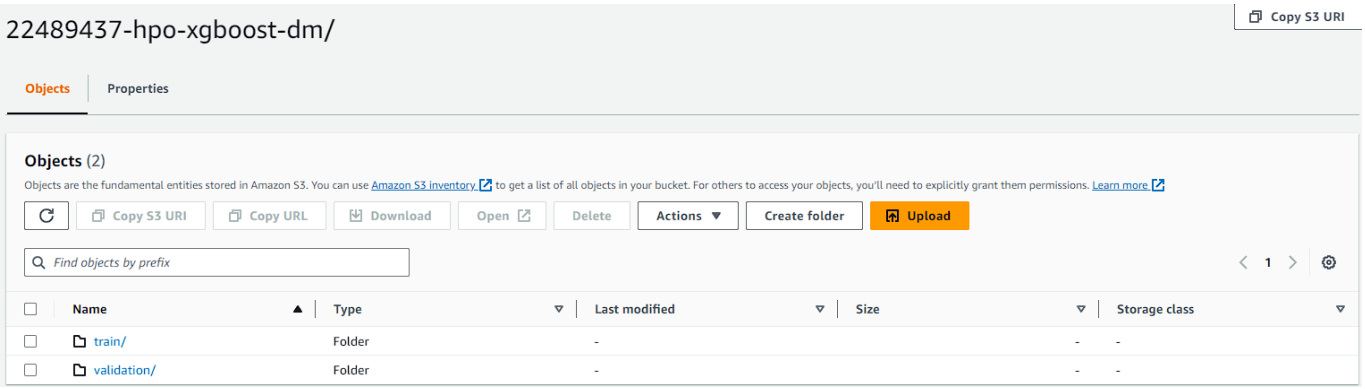
Step 5: Split Data into training, validation and test

1. The following code splits the dataset into training (70%), validation (20%), and test (10%).
    - The first block of code shuffles model_data using the sample method with frac=1 (which means to sample the whole dataset) and a random seed of 1729. The shuffled data is then split into three segments.
    - For each of the datasets (train, validation, and test), the code reorders the columns so that the target column y_yes is the first column, and then it drops the y_no and y_yes columns from the rest. It then saves the reordered data to a CSV file. This is done for compatibility reasons as Amazon SageMaker's XGBoost algorithm expects the first column to be the target variable and the CSV should not include headers.

```
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
)
```

2. The newly created data is the uploaded to the S3 bucket we created earlier.

```
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")
```



Step 6: Setup Hyperparameter Optimization

1. The following code sets up a configuration for a hyperparameter tuning job:
    - ParameterRanges: This specifies the hyperparameters you want to tune and their possible values.
    - CategoricalParameterRanges: This would contain parameters that have categorical values, but it's empty as we have transformed all the categorical variables into indicator variables in the previous steps.
    - ContinuousParameterRanges: This contains hyperparameters that can take any real value within a range.

- ResourceLimits: This part specifies how many training jobs can be created in total (MaxNumberOfTrainingJobs) and how many of them can run in parallel (MaxParallelTrainingJobs).
- Strategy: The strategy used for the tuning job. In this case, the "Bayesian" strategy is used, which is a popular method for hyperparameter optimization. It builds a probability model of the objective function and uses it to select the most promising hyperparameters to evaluate in the true objective function.
- HyperParameterTuningJobObjective: This is where you define the metric you want to optimize. The goal here is to maximize the "validation:auc" (Area Under the Curve for the validation set).

```
tuning_job_name = f"{student_id}-xgboost-tuningjob-01"

print(tuning_job_name)

tuning_job_config = {
    "ParameterRanges": {
        "CategoricalParameterRanges": [],
        "ContinuousParameterRanges": [
            {
                "MaxValue": "1",
                "MinValue": "0",
                "Name": "eta",
            },
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "min_child_weight",
            },
            {
                "MaxValue": "2",
                "MinValue": "0",
                "Name": "alpha",
            },
        ],
        "IntegerParameterRanges": [
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "max_depth",
            }
        ],
    },
    "ResourceLimits": {"MaxNumberOfTrainingJobs": 2, "MaxParallelTrainingJobs":
2},
    "Strategy": "Bayesian",
    "HyperParameterTuningJobObjective": {"MetricName": "validation:auc", "Type":
"Maximize"},
}
```

2. The hyperparameter tuning job will launch training jobs to find an optimal configuration of hyperparameters. These training jobs should be configured using the SageMaker

CreateHyperParameterTuningJob API. To configure the training jobs, we define a JSON object and pass it as the value of the TrainingJobDefinition parameter inside CreateHyperParameterTuningJob.

- AlgorithmSpecification – The registry path of the Docker image containing the training algorithm and related metadata. We fetch the XGboost Docker image for our specic usecase.
- InputDataConfig: This specifies where SageMaker should fetch the training and validation data from. For both data sources, the content type is CSV and data distribution is "FullyReplicated", which means each training instance will get the full dataset.
- OutputDataConfig: This defines where SageMaker should store the output of the training job (e.g., model artifacts). The output will be stored in the specified S3 bucket under an "output" prefix.
- ResourceConfig: Specifies the infrastructure to be used for the training job. In this case, a single ml.m5.xlarge instance with a volume size of 10GB.
- RoleArn: This is the Amazon Resource Name (ARN) of the IAM role that SageMaker can assume to perform tasks on your behalf. We make use of the ARN of the SageMakerRole that we retreived earlier.
- StaticHyperParameters: These are hyperparameters specific to the XGBoost algorithm that will remain fixed during training. For instance, objective is set to "binary:logistic", which is suitable for binary classification problems.
- StoppingCondition: This specifies that the training job should be terminated if it runs for more than 43200 seconds (or 12 hours).

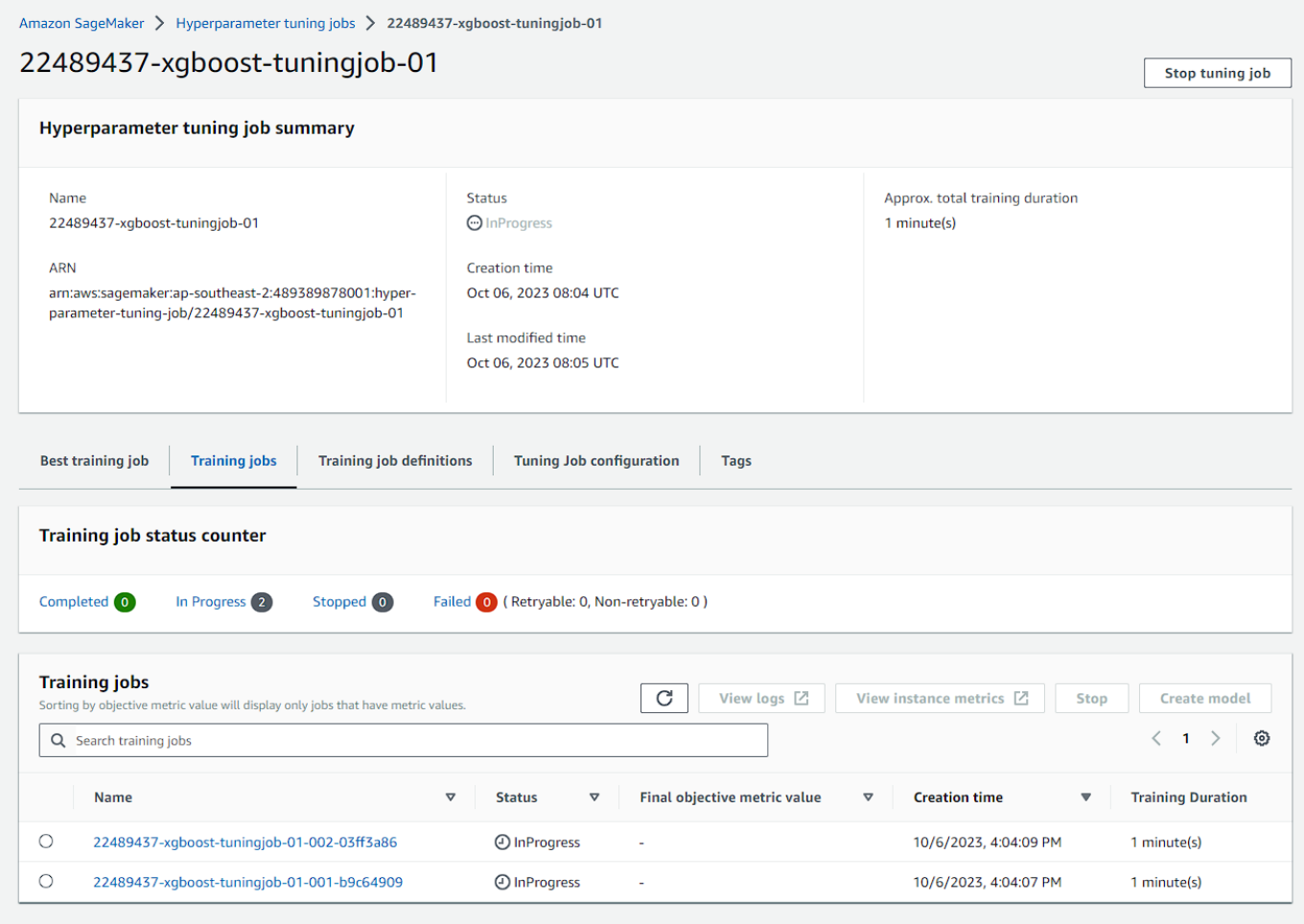
```
from sagemaker.image_uris import retrieve
# Use XGBoost algorithm for training
training_image = retrieve(framework="xgboost", region=region, version="latest")

s3_input_train = "s3://{}/{}/train".format(bucket, prefix)
s3_input_validation = "s3://{}/{}/validation/".format(bucket, prefix)

training_job_definition = {
    "AlgorithmSpecification": {"TrainingImage": training_image,
"TrainingInputMode": "File"},
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train,
                }
            },
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
```

```
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation,
                }
            },
        },
    ],
    "OutputDataConfig": {"S3OutputPath": "s3://{}/{}/output".format(bucket,
prefix)},
    "ResourceConfig": {"InstanceCount": 1, "InstanceType": "ml.m5.xlarge",
"VolumeSizeInGB": 10},
    "RoleArn": sagemaker_role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "1",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4",
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 43200},
}
```

3. This code snippet is launching a hyperparameter tuning job on Amazon SageMaker.

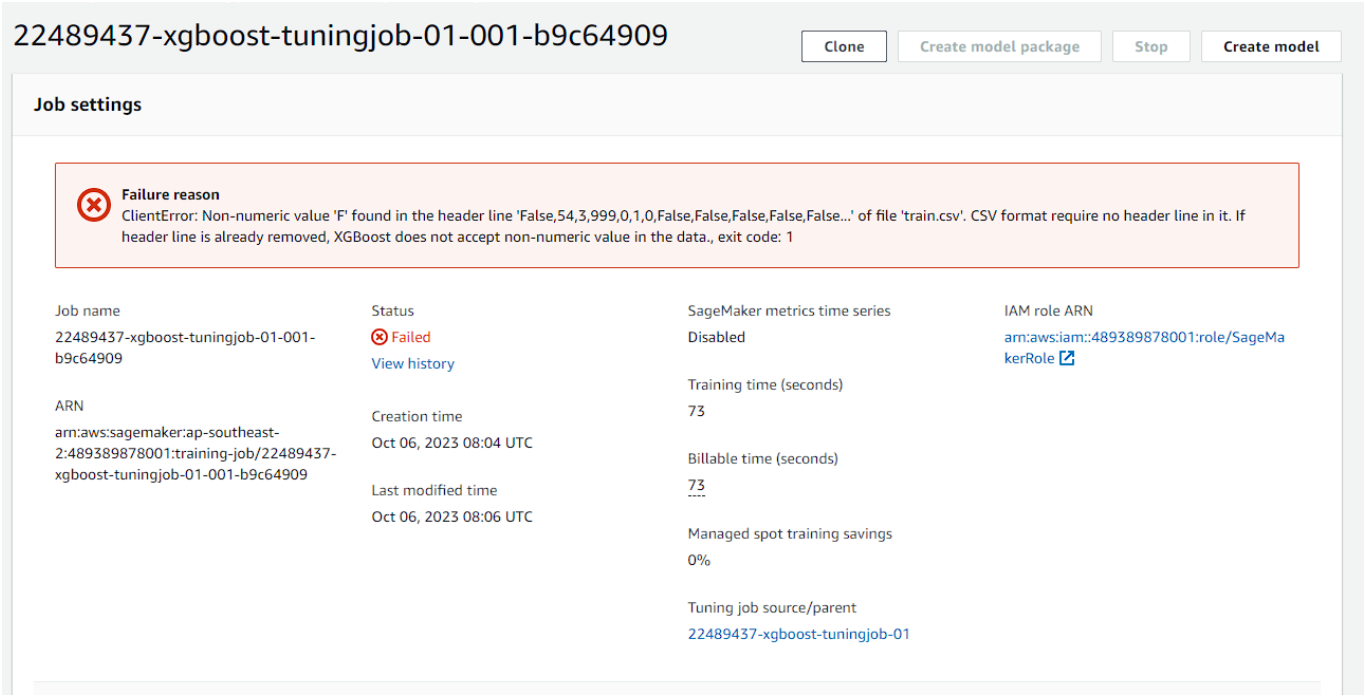
```
#Launch Hyperparameter Tuning Job
smclient.create_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuning_job_name,
    HyperParameterTuningJobConfig=tuning_job_config,
    TrainingJobDefinition=training_job_definition,
)
```

4. We can log into the sagemaker service on the AWS console. Here we can see all the running training instances.

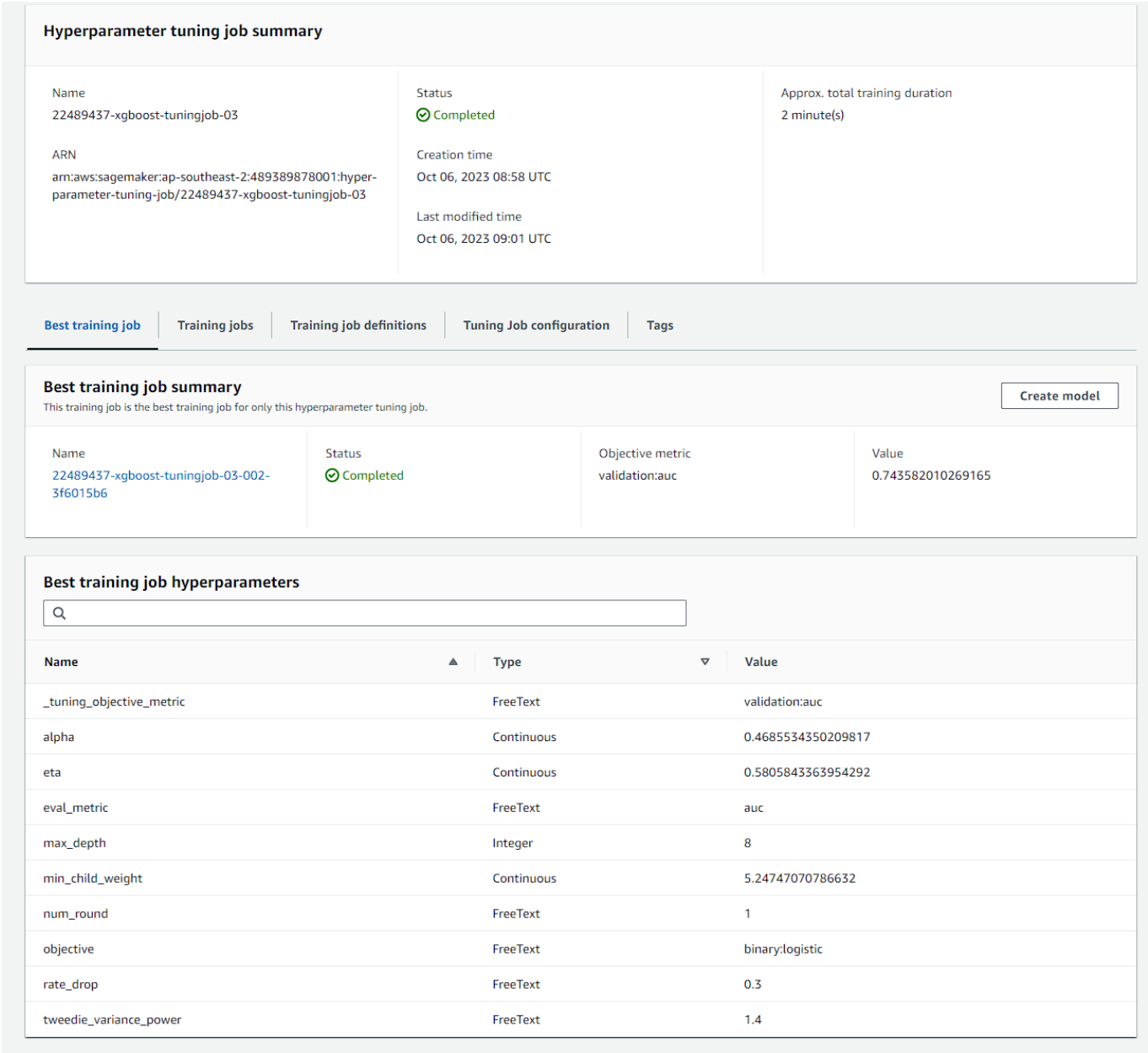


5. The hyperparameter optimization failed and we got the message "ClientError: Non-numeric value 'F' found in the header line 'False,54,3,999,0,1,0,False,False,False,False,False...' of file 'train.csv'". In order to resolve this, we convert all the True/False in the dataset to 1/0 using the following code.

```
bool_cols = model_data.select_dtypes(include=['bool']).columns
model_data[bool_cols] = model_data[bool_cols].astype(int)
```



6. Running the code again, we succeed this time and we can view the best training job hyperparaneters either in the AWS console or in the s3://YOUR_BUCKET/sagemaker/22489437-hpo-xgboost-dm/output.



Hyperparameter tuning job summary

Name 22489437-xgboost-tuningjob-03	Status Completed	Approx. total training duration 2 minute(s)
ARN arn:aws:sagemaker:ap-southeast-2:489389878001:hyper-parameter-tuning-job/22489437-xgboost-tuningjob-03	Creation time Oct 06, 2023 08:58 UTC	
	Last modified time Oct 06, 2023 09:01 UTC	

Best training job | Training jobs | Training job definitions | Tuning Job configuration | Tags

Best training job summary

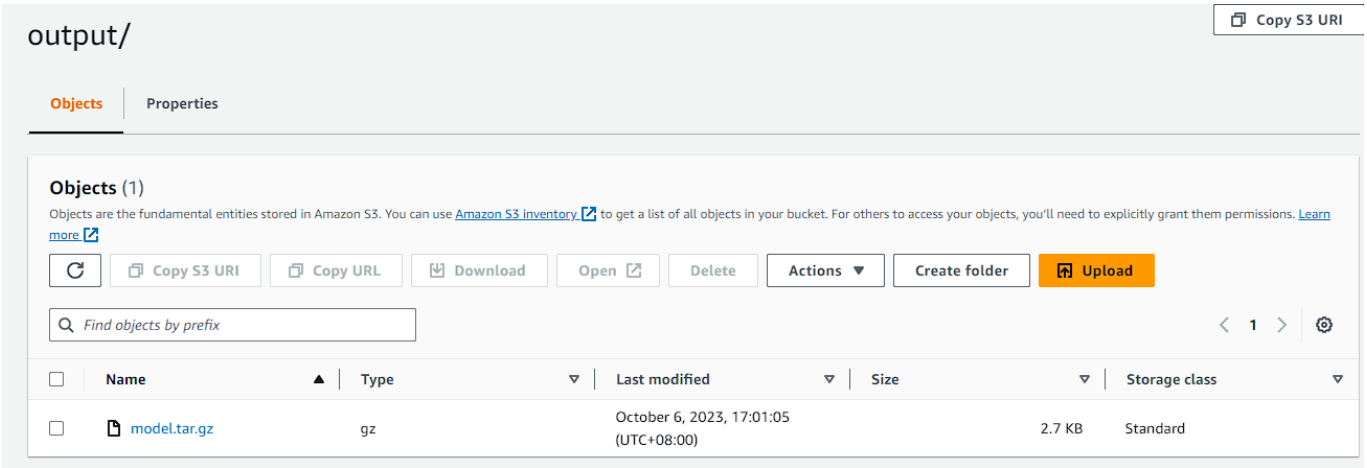
This training job is the best training job for only this hyperparameter tuning job.

Create model

Name	Status	Objective metric	Value
22489437-xgboost-tuningjob-03-002-3f6015b6	Completed	validation:auc	0.743582010269165

Best training job hyperparameters

Name	Type	Value
_tuning_objective_metric	FreeText	validation:auc
alpha	Continuous	0.4685534350209817
eta	Continuous	0.5805843363954292
eval_metric	FreeText	auc
max_depth	Integer	8
min_child_weight	Continuous	5.24747070786632
num_round	FreeText	1
objective	FreeText	binary:logistic
rate_drop	FreeText	0.3
tweedie_variance_power	FreeText	1.4



output/

Copy S3 URI

Objects | Properties

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix

1

Name	Type	Last modified	Size	Storage class
model.tar.gz	gz	October 6, 2023, 17:01:05 (UTC+08:00)	2.7 KB	Standard

Lab 9: More AI

Section 1: Detect Languages from text

1. We run the example code and confirm the output:

```
import boto3
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.",
)

print(response['Languages'])
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 lab9.py
[{'LanguageCode': 'en', 'Score': 0.9983963966369629}]
jookai@jookai:~/Desktop/cits5503/lab9$
```

2. We modify the code to use a script to recognize different languages and returnt he message in the format: "<predicted_language> detected with xx% confidence"
    - `client = boto3.client('comprehend')`: This initializes a client to interact witht he Amazon Comprehend service
    - `argParser = argparse.ArgumentParser()`: The first section of the main function uses the argparse library to allow the program to accept command line input of plaintext which we will process.
    - `response = client.detect_dominant_language`: We call the detect_dominant_language method of the Comprehend client to determine the dominant language of the plaintext. The result is stored in the response variable.
    - `lang_code = response['Languages'][0]['LanguageCode']`: This line extracts the language code (e.g., "en" for English) of the detected dominant language from the response.
    - `lang = pycountry.languages.get(alpha_2=lang_code).name`: Since we want the language name in full we use the pycountry library to convert the language into a full language name.
    - `conf = str(int(response['Languages'][0]['Score'] * 100))`: Here, the confidence score (a float between 0 and 1) returned by the Comprehend service is multiplied by 100 to convert it into a percentage and then rounded to an integer.

```
client = boto3.client('comprehend')

def main(argv):
    argParser = argparse.ArgumentParser()
    argParser.add_argument("-t", "--plaintext", type=str)

    args = argParser.parse_args()
    plaintext = args.plaintext

    response = client.detect_dominant_language(
        Text=plaintext
    )

    lang_code = response['Languages'][0]['LanguageCode']
    lang = pycountry.languages.get(alpha_2=lang_code).name
```

```
    conf = str(int(response['Languages'][0]['Score'] * 100))

    print(lang +" detected with "+ conf + "%"+ " confidence")

    return 0
```

3. The results of the script on the 4 provided samples can be seen below:

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_lang.py -t "The French Rev
olution was a period of social and political upheaval in France and its colonies
 beginning in 1789 and ending in 1799."
English detected with 99% confidence
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_lang.py -t "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 1
605, es una de las obras más destacadas de la literatura española y la literatur
a universal, y una de las más traducidas. En 1615 aparecería la segunda parte de
l Quijote de Cervantes con el título de El ingenioso caballero don Quijote de la
 Mancha."
Spanish detected with 99% confidence
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_lang.py -t "Moi je n'étais
 rien Et voilà qu'aujourd'hui Je suis le gardien Du sommeil de ses nuits Je l'ai
me 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 m
ourir"
French detected with 99% confidence
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_lang.py -t "L'amor che mov
e il sole e l'altre stelle."
Italian detected with 99% confidence
```

Section 2: Sentiment Analysis

1. The code below performs sentiment analysis on the supplied plaintext:
    - `client = boto3.client('comprehend')`: This initializes a client to interact witht he Amazon Comprehend service
    - `argParser = argparse.ArgumentParser()`: The first section of the main function uses the argparse library to allow the program to accept command line input of plaintext which we will process.
    - `response = client.detect_dominant_language`: We call the detect_dominant_language method of the Comprehend client to determine the dominant language of the plaintext. The result is stored in the response variable.
    - `lang_code = response['Languages'][0]['LanguageCode']`: This line extracts the language code (e.g., "en" for English) of the detected dominant language from the response. This is required for the following function call.
    - `sen = client.detect_sentiment(Text=plaintext, LanguageCode=lang_code)`: We use the detect_sentiment method of the Comprehend client to determine the sentiment of the text. This method requires both the text and its language code. The sentiment result (e.g., "POSITIVE", "NEGATIVE", "NEUTRAL", "MIXED") is then extracted from the response.

```
client = boto3.client('comprehend')

def main(argv):
    argParser = argparse.ArgumentParser()
    argParser.add_argument("-t", "--plaintext", type=str)

    args = argParser.parse_args()
    plaintext = args.plaintext

    response = client.detect_dominant_language(
        Text=plaintext
    )

    lang_code = response['Languages'][0]['LanguageCode']
    lang = pycountry.languages.get(alpha_2=lang_code).name

    sen = client.detect_sentiment(Text=plaintext, LanguageCode = lang_code)
    sentiment = sen['Sentiment']
    print("The sentiment of the %s text is %s" % (lang, sentiment))

    return 0
```

2. The results of the script on the 4 provided samples can be seen below:

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 sentiment_analysis.py -t "The Fre
nch Revolution was a period of social and political upheaval in France and its c
olonies beginning in 1789 and ending in 1799."
The sentiment of the English text is NEUTRAL
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 sentiment_analysis.py -t "El Quij
ote es la obra más conocida de Miguel de Cervantes Saavedra. Publicada su primer
a parte con el título de El ingenioso hidalgo don Quijote de la Mancha a comienz
os de 1605, es una de las obras más destacadas de la literatura española y la li
teratura universal, y una de las más traducidas. En 1615 aparecería la segunda p
arte del Quijote de Cervantes con el título de El ingenioso caballero don Quijot
e de la Mancha."
The sentiment of the Spanish text is NEUTRAL
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 sentiment_analysis.py -t "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'a
ime à mourir"
The sentiment of the French text is NEGATIVE
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 sentiment_analysis.py -t "L'amor
che move il sole e l'altre stelle."
The sentiment of the Italian text is POSITIVE
```

Section 3: Entity detection

1. The code below performs key phrase on the supplied plaintext:

- `client = boto3.client('comprehend')`: This initializes a client to interact witht he Amazon Comprehend service
- `argParser = argparse.ArgumentParser()`: The first section of the main function uses the argparse library to allow the program to accept command line input of plaintext which we will process.
- `response = client.detect_dominant_language`: We call the detect_dominant_language method of the Comprehend client to determine the dominant language of the plaintext. The result is stored in the response variable.
- `lang_code = response['Languages'][0]['LanguageCode']`: This line extracts the language code (e.g., "en" for English) of the detected dominant language from the response. This is required for the following function call.
- `ent = client.detect_entities(Text=plaintext, LanguageCode = lang_code)`: With the detect_entities method of the Comprehend client, the script determines the entities present in the text.

```
client = boto3.client('comprehend')

def main(argv):
    argParser = argparse.ArgumentParser()
    argParser.add_argument("-t", "--plaintext", type=str)

    args = argParser.parse_args()
    plaintext = args.plaintext

    response = client.detect_dominant_language(
        Text=plaintext
    )

    lang_code = response['Languages'][0]['LanguageCode']

    ent = client.detect_entities(Text=plaintext, LanguageCode = lang_code)
    entities = ent['Entities']

    if not entities:
        print('There are no entities in the text')
        return
    print("The entities in the text are:")
    for ent in entities:
       print(ent['Text'] + ' is ' + ent['Type'])

    return 0
```

2. The results of the script on the 4 provided samples can be seen below:

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_entities.py -t "The French
 Revolution was a period of social and political upheaval in France and its colo
nies beginning in 1789 and ending in 1799."
The entities in the text are:
French Revolution is EVENT
France is LOCATION
1789 is DATE
1799 is DATE
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_entities.py -t "El Quijote
 es la obra más conocida de Miguel de Cervantes Saavedra. Publicada su primera p
arte 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 liter
atura universal, y una de las más traducidas. En 1615 aparecería la segunda part
e del Quijote de Cervantes con el título de El ingenioso caballero don Quijote d
e la Mancha."
The entities in the text are:
El Quijote is TITLE
Miguel de Cervantes Saavedra is PERSON
primera parte is QUANTITY
El ingenioso hidalgo don Quijote de la Mancha is TITLE
1605 is DATE
una de is QUANTITY
española is OTHER
una de las más is QUANTITY
1615 is DATE
segunda parte is QUANTITY
Quijote de Cervantes is TITLE
El ingenioso caballero don Quijote de la Mancha is TITLE
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_entities.py -t "Moi je n'é
tais 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 ouv
rir L'espace de ses bras Pour tout reconstruire Pour tout reconstruire Je l'aime
 à mourir"
The entities in the text are:
aujourd'hui is DATE
Tout ce qu' is QUANTITY
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_entities.py -t "L'amor che
 move il sole e l'altre stelle."
There are no entities in the text
```

3. Entities refer to chunks of text that can be classified into groups based on their semantic meaning. These often represent real-world objects such as people, places, organizations, dates, and other named objects or concepts. In the context of Natural Language Processing (NLP), identifying entities can help in understanding the content and context of a document.

Section 4: Keyphrase detection

1. The code below performs key phrase on the supplied plaintext:
    - `client = boto3.client('comprehend')`: This initializes a client to interact witht he Amazon Comprehend service
    - `argParser = argparse.ArgumentParser()`: The first section of the main function uses the argparse library to allow the program to accept command line input of plaintext which we will

process.

- `response = client.detect_dominant_language`: We call the detect_dominant_language method of the Comprehend client to determine the dominant language of the plaintext. The result is stored in the response variable.
- `lang_code = response['Languages'][0]['LanguageCode']`: This line extracts the language code (e.g., "en" for English) of the detected dominant language from the response. This is required for the following function call.
- `key_phrases = client.detect_key_phrases(Text=plaintext, LanguageCode = lang_code)`: The detect_key_phrases method of the Comprehend client is called to find the key phrases in the text. These key phrases are then stored in the key_phrase variable.

```
client = boto3.client('comprehend')

def main(argv):
    argParser = argparse.ArgumentParser()
    argParser.add_argument("-t", "--plaintext", type=str)

    args = argParser.parse_args()
    plaintext = args.plaintext

    response = client.detect_dominant_language(
        Text=plaintext
    )

    lang_code = response['Languages'][0]['LanguageCode']

    key_phrases = client.detect_key_phrases(Text=plaintext, LanguageCode =
lang_code)
    key_phrase = key_phrases['KeyPhrases']
    if not key_phrase:
        print('There are no key phrases in the text')
        return
    print("The key phrases in the text are:")
    for kp in key_phrase:
       print(kp['Text'])

    return 0
```

2. The results of the script on the 4 provided samples can be seen below:

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_key_phrases.py -t "The Fre
nch Revolution was a period of social and political upheaval in France and its c
olonies beginning in 1789 and ending in 1799."
The key phrases in the text are:
The French Revolution
a period
social and political upheaval
France
its colonies
1789
1799
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_key_phrases.py -t "El Quijote e
s la obra más conocida de Miguel de Cervantes Saavedra. Publicada su primera parte co
n 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 Cerva
ntes con el título de El ingenioso caballero don Quijote de la Mancha."
The key phrases in the text are:
El Quijote
la obra
más conocida
Miguel de Cervantes Saavedra
su primera parte
el título
El ingenioso hidalgo don Quijote de la Mancha
comienzos
1605
las obras
más destacadas
la literatura española
la literatura universal
las más traducidas
la segunda parte
Quijote de Cervantes
el título
ingenioso caballero don Quijote de la Mancha
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_key_phrases.py -t "Moi je n'éta
is 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"
The key phrases in the text are:
Moi
je
n'étais rien
aujourd'hui
Je suis le gardien Du sommeil de ses nuits
Je
l'
Vous
Tout ce
qu'
il
vous
Elle
L'espace de ses bras
tout
tout
Je
l'
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_key_phrases.py -t "L'amor che m
ove il sole e l'altre stelle."
The key phrases in the text are:
L'amor
che
il sole
l'altre stelle
```

3. Keyphrases, often termed as "keywords" or "key terms", refer to words or groups of words within a text that are deemed significant or representative of the primary content and topics of that text. The

identification of keyphrases is crucial in various fields of text analysis, as these phrases help in summarizing, categorizing, and understanding the primary themes or topics of the text.

Section 5: Detecting Syntax

1. The code below performs key phrase on the supplied plaintext:
    - `client = boto3.client('comprehend')`: This initializes a client to interact witht he Amazon Comprehend service
    - `argParser = argparse.ArgumentParser()`: The first section of the main function uses the argparse library to allow the program to accept command line input of plaintext which we will process.
    - `response = client.detect_dominant_language`: We call the detect_dominant_language method of the Comprehend client to determine the dominant language of the plaintext. The result is stored in the response variable.
    - `lang_code = response['Languages'][0]['LanguageCode']`: This line extracts the language code (e.g., "en" for English) of the detected dominant language from the response. This is required for the following function call.
    - `syn = client.detect_syntax(Text=plaintext, LanguageCode = lang_code)`: The detect_syntax method of the Comprehend client identifies the syntactic elements (parts of speech) in the text, and the results are stored in the syntax variable.

```
client = boto3.client('comprehend')

def main(argv):
    argParser = argparse.ArgumentParser()
    argParser.add_argument("-t", "--plaintext", type=str)

    args = argParser.parse_args()
    plaintext = args.plaintext

    response = client.detect_dominant_language(
        Text=plaintext
    )

    lang_code = response['Languages'][0]['LanguageCode']

    syn = client.detect_syntax(Text=plaintext, LanguageCode = lang_code)
    syntax = syn['SyntaxTokens']
    print("The syntax in the text are:")
    for s in syntax:
       print(s['Text'] +' is '+ s['PartOfSpeech']['Tag'] + '\t')

    return 0
```

2. The results of the script on the 4 provided samples can be seen below:

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_syntax.py -t "The French Revolu
tion was a period of social and political upheaval in France and its colonies beginni
ng in 1789 and ending in 1799."
The syntax in the text are:
The is DET
French is PROPN
Revolution is PROPN
was is VERB
a is DET
period is NOUN
of is ADP
social is ADJ
and is CONJ
political is ADJ
upheaval is NOUN
in is ADP
France is PROPN
and is CONJ
its is PRON
colonies is NOUN
beginning is VERB
in is ADP
1789 is NUM
and is CONJ
ending is VERB
in is ADP
1799 is NUM
. is PUNCT
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_syntax.py -t "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 d
e 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."
The syntax in the text are:
El is DET
Quijote is PROPN
es is VERB
la is DET
obra is NOUN
más is ADV
conocida is ADJ
de is ADP
Miguel is PROPN
de is ADP
Cervantes is PROPN
Saavedra is PROPN
. is PUNCT
Publicada is VERB
su is DET
primera is ADJ
parte is NOUN
con is ADP
el is DET
título is NOUN
de is ADP
El is DET
ingenioso is ADJ
hidalgo is NOUN
don is PROPN
Quijote is PROPN
de is ADP
la is DET
Mancha is PROPN
a is ADP
comienzos is PROPN
de is ADP
1605 is NUM
```

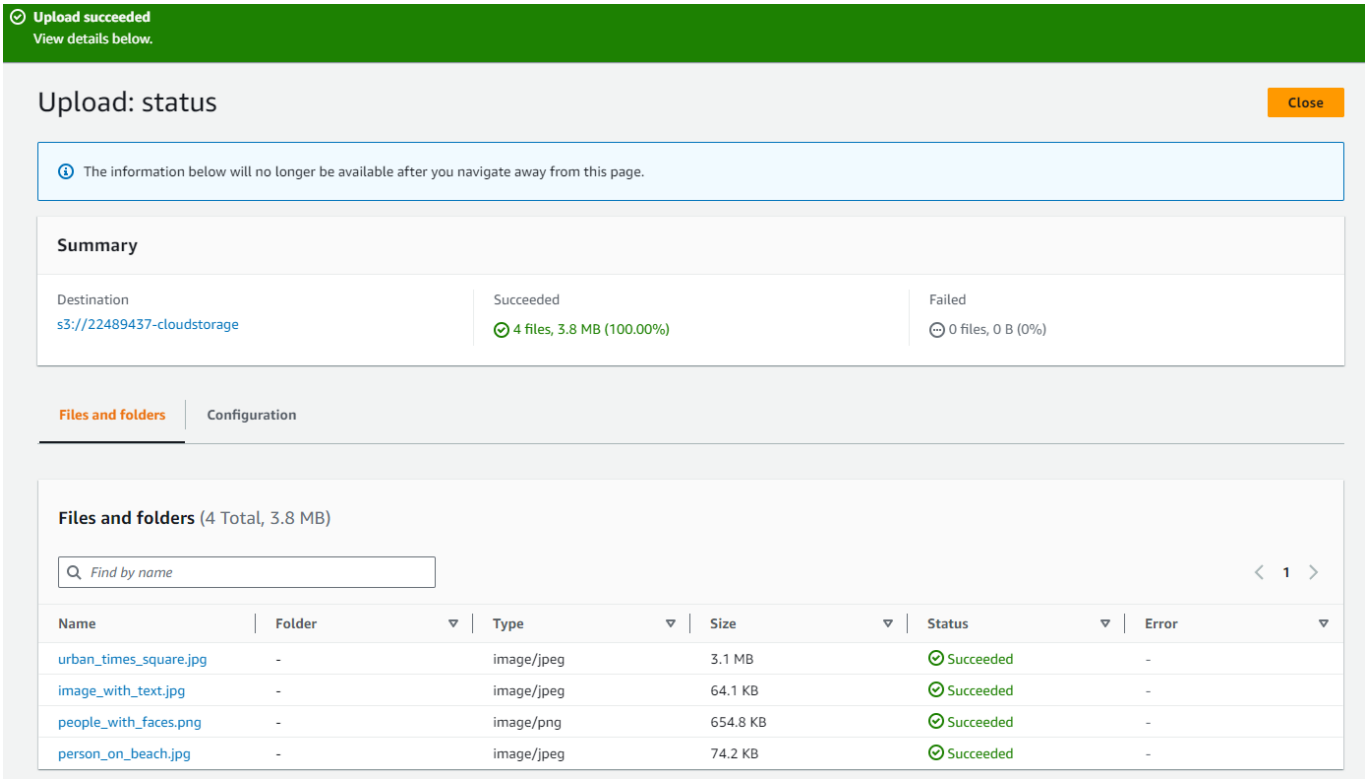
```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_syntax.py -t "Moi je n'étais ri
en Et voilà qu'aujourd'hui Je suis le gardien Du sommeil de ses nuits Je l'aime à mou
rir Vous pouvez détruire Tout ce qu'il vous plaira Elle n'a qu'à ouvrir L'espace de s
es bras Pour tout reconstruire Pour tout reconstruire Je l'aime à mourir"
The syntax in the text are:
Moi is PRON
je is PRON
n' is ADV
étais is AUX
rien is PRON
Et is CCONJ
voilà is VERB
qu' is SCONJ
aujourd'hui is ADV
Je is PRON
suis is AUX
le is DET
gardien is NOUN
Du is ADP
sommeil is NOUN
de is ADP
ses is DET
nuits is NOUN
Je is PRON
l' is PRON
aime is VERB
à is ADP
mourir is VERB
Vous is PRON
pouvez is AUX
détruire is VERB
Tout is DET
ce is PRON
qu' is PRON
il is PRON
vous is PRON
plaira is VERB
Elle is PRON
n' is ADV
a is VERB
```

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 detect_syntax.py -t "L'amor che move i
l sole e l'altre stelle."
The syntax in the text are:
L' is DET
amor is NOUN
che is PRON
move is VERB
il is DET
sole is NOUN
e is CCONJ
l' is DET
altre is ADJ
stelle is NOUN
. is PUNCT
```

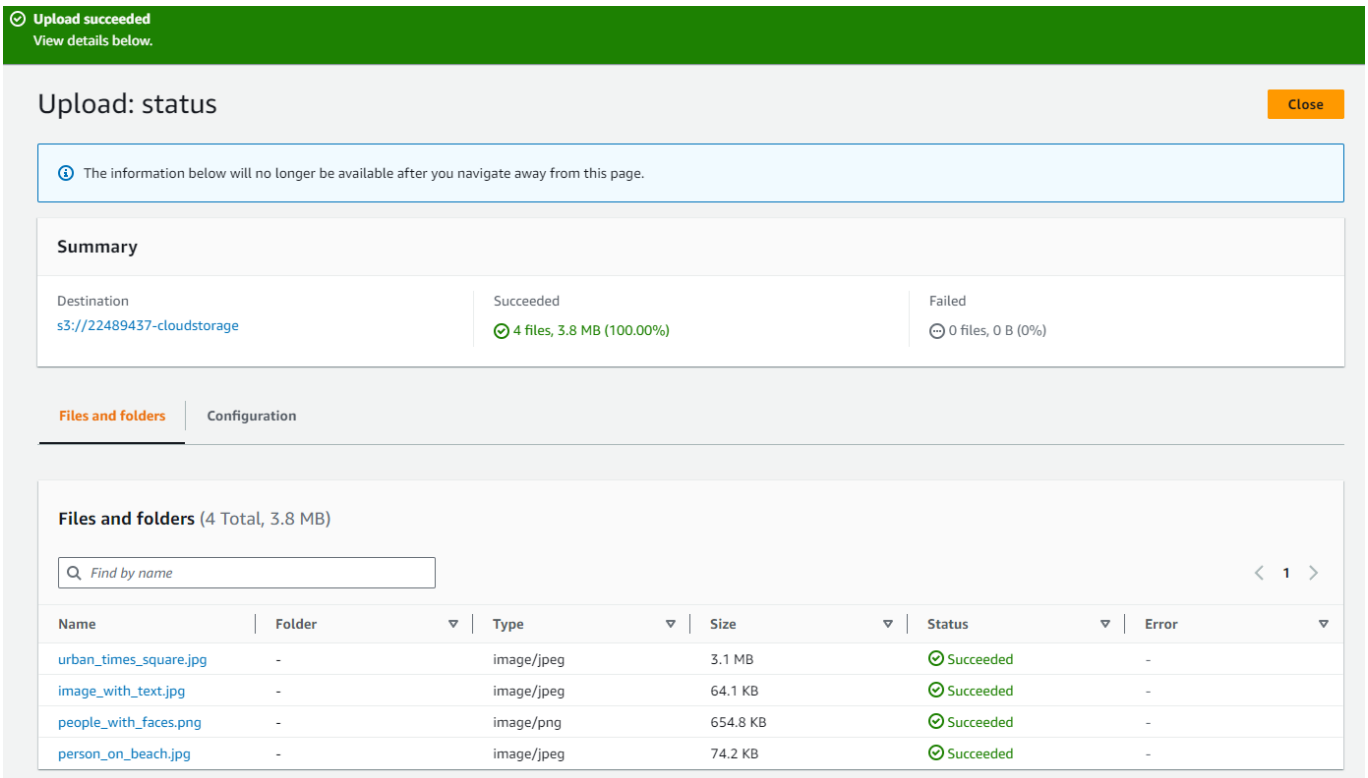
3. Syntax refers to the arrangement of words and phrases in sentences to create well-formed and meaningful statements in a particular language. In the context of linguistics and natural language processing (NLP), syntax is the study of the rules, principles, and processes that govern the structure of sentences. Essentially, it's about how words come together to form coherent and grammatically correct sentences.

Section 6: Creating S3 bucket

1. We create an S3 bucket named `22489437-cloudstorage` using the aws console



2. We upload 4 files to the newly created S3 bucket.



`urban_times_square.jpg`: This simulates an image of an urban setting

`person_on_beach.jpeg`: This is a group of people on the beach wearing swimsuits. This will be used to test detection of explicit and suggestive adult content.

`people_with_faces.png`: This is 5 people with different races to test the analysis of facial attributes.

`image_with_text.jpg`: This is an image with text to test extraction of text from images.

3. The paths to these images will be stored in a dictionary:

```
bucket_images=[
    {
        'S3Object': {
            'Bucket': '22489437-cloudstorage',
            'Name': 'urban_times_square.jpg'
        },
    },
    {
        'S3Object': {
            'Bucket': '22489437-cloudstorage',
            'Name': 'person_on_beach.jpeg'
        },
    },
    {
```

```
        'S3Object': {
            'Bucket': '22489437-cloudstorage',
            'Name': 'people_with_faces.png'
        },
    },
    {
        'S3Object': {
            'Bucket': '22489437-cloudstorage',
            'Name': 'image_with_text.jpg'
        },
    },
]
```

Section 7: Label Recognition

1. The code below performs label recognition on the images listed in section 6:
    - A rekognition client object is created using the boto3 library. This object is used to make calls to the Amazon Rekognition service.
    - For each image, the code tries to detect labels using the Amazon Rekognition's detect_labels method. This method identifies objects, people, text, scenes, etc., in the image.
    - The detected labels are printed

```
client = boto3.client('rekognition')

def main(argv):
    for image in bucket_images:
        image_name = image['S3Object']['Name']
        try:
            response = client.detect_labels(Image=image, MaxLabels=3,
MinConfidence=0.95)
            print('Labels detected in %s:' % image_name)
            print(response['Labels'])
        except Exception as e:
            print(f"Error processing {image_name}: {e}")

    return 0
```

2. The output of the code is shown below, each of the detected labels as well as their confidence score is shown below:

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 label_recognition.py
Labels detected in urban_times_square.jpg:
[{'Name': 'City', 'Confidence': 100.0, 'Instances': [], 'Parents': [], 'Aliases': [{'
Name': 'Town'}], 'Categories': [{'Name': 'Buildings and Architecture'}]}, {'Name': 'M
etropolis', 'Confidence': 100.0, 'Instances': [], 'Parents': [{'Name': 'City'}, {'Nam
e': 'Urban'}], 'Aliases': [], 'Categories': [{'Name': 'Buildings and Architecture'}]}
, {'Name': 'Urban', 'Confidence': 100.0, 'Instances': [], 'Parents': [], 'Aliases': [
], 'Categories': [{'Name': 'Colors and Visual Composition'}]}]
Labels detected in person_on_beach.jpeg:
[{'Name': 'Person', 'Confidence': 99.99945068359375, 'Instances': [{'BoundingBox': {'
Width': 0.1709585338830948, 'Height': 0.7989519238471985, 'Left': 0.5553748607635498,
 'Top': 0.09726513177156448}, 'Confidence': 98.67424774169922}], 'Parents': [], 'Alia
ses': [{'Name': 'Human'}], 'Categories': [{'Name': 'Person Description'}]}, {'Name':
'Walking', 'Confidence': 99.99945068359375, 'Instances': [], 'Parents': [{'Name': 'Pe
rson'}], 'Aliases': [], 'Categories': [{'Name': 'Actions'}]}, {'Name': 'Man', 'Confid
ence': 98.67424774169922, 'Instances': [{'BoundingBox': {'Width': 0.1709585338830948,
 'Height': 0.7989519238471985, 'Left': 0.5553748607635498, 'Top': 0.09726513177156448
}, 'Confidence': 98.67424774169922}], 'Parents': [{'Name': 'Adult'}, {'Name': 'Male'}
, {'Name': 'Person'}], 'Aliases': [], 'Categories': [{'Name': 'Person Description'}]}
]
Labels detected in people_with_faces.png:
[{'Name': 'Head', 'Confidence': 99.99775695800781, 'Instances': [], 'Parents': [{'Nam
e': 'Person'}], 'Aliases': [], 'Categories': [{'Name': 'Person Description'}]}, {'Nam
e': 'Face', 'Confidence': 99.997314453125, 'Instances': [], 'Parents': [{'Name': 'Hea
d'}, {'Name': 'Person'}], 'Aliases': [], 'Categories': [{'Name': 'Person Description'
}]}, {'Name': 'People', 'Confidence': 99.99517059326172, 'Instances': [], 'Parents':
[{'Name': 'Person'}], 'Aliases': [], 'Categories': [{'Name': 'Person Description'}]}]
Labels detected in image_with_text.jpg:
[{'Name': 'Advertisement', 'Confidence': 87.20674133300781, 'Instances': [], 'Parents
': [], 'Aliases': [], 'Categories': [{'Name': 'Text and Documents'}]}, {'Name': 'Pers
on', 'Confidence': 74.20307159423828, 'Instances': [{'BoundingBox': {'Width': 0.07917
390018701553, 'Height': 0.16412995755672455, 'Left': 0.5184438824653625, 'Top': 0.835
8700275421143}, 'Confidence': 74.20307159423828}, {'BoundingBox': {'Width': 0.0694150
4776477814, 'Height': 0.1563015580177307, 'Left': 0.5728679895401001, 'Top': 0.843698
4419822693}, 'Confidence': 11.670639038085938}], 'Parents': [], 'Aliases': [{'Name':
'Human'}], 'Categories': [{'Name': 'Person Description'}]}, {'Name': 'Poster', 'Confi
dence': 60.965911865234375, 'Instances': [], 'Parents': [{'Name': 'Advertisement'}],
'Aliases': [{'Name': 'Brochure'}, {'Name': 'Flyer'}], 'Categories': [{'Name': 'Furnit
ure and Furnishings'}]}]
jookai@jookai:~/Desktop/cits5503/lab9$
```

Section 8: Image Moderation

1. The code below performs image moderation on the images listed in section 6:
    - A rekognition client object is created using the boto3 library. This object is used to make calls to the Amazon Rekognition service.
    - For each image, this code detects moderation labels for the image. Amazon Rekognition's detect_moderation_labels method is used to identify potentially unsafe content in the image. This can help filter out content that violates certain guidelines or standards.

```
client = boto3.client('rekognition')

def main(argv):
    for image in bucket_images:
        image_name = image['S3Object']['Name']
        try:
            response = client.detect_moderation_labels(Image=image)
            print('Moderated for %s:' % image_name)
```

```
            print(response['ModerationLabels'])
            print('\n')
        except Exception as e:
            print(f"Error processing {image_name}: {e}")

    return 0
```

2. The output of the code is shown below. As seen, only `person_on_beach.jpeg` was flagged due to suggestive female swimwear. The other images did not trigger the moderation tool.

```
jookai@jookai:~/Desktop/cits5503/lab9$ python3 image_moderation.py
Moderated for urban_times_square.jpg:
[]


Moderated for person_on_beach.jpeg:
[{'Confidence': 87.66188049316406, 'Name': 'Revealing Clothes', 'ParentName': 'Sugges
tive'}, {'Confidence': 87.66188049316406, 'Name': 'Suggestive', 'ParentName': ''}, {'
Confidence': 63.533164978027344, 'Name': 'Female Swimwear Or Underwear', 'ParentName'
: 'Suggestive'}]


Moderated for people_with_faces.png:
[]


Moderated for image_with_text.jpg:
[]
```

Section 9: Facial Analysis

1. The code below performs facial analysis on the images listed in section 6:
    - A rekognition client object is created using the boto3 library. This object is used to make calls to the Amazon Rekognition service.
    - For each image, this code uses the detect_faces method of the Rekognition client to identify faces in the image.
    - Using the detect_faces method, we can detect faces in an image, and for each detected face, we can get details about facial attributes, landmarks, pose, quality, and more. This can be useful for a range of applications, from simple face detection to more advanced use cases like emotion analysis.

```
client = boto3.client('rekognition')

def main(argv):
    for image in bucket_images:
        image_name = image['S3Object']['Name']
        try:
            response = client.detect_faces(Image=image)
            print('facial analysis for %s:' % image_name)
            print(response['FaceDetails'])
            print('\n')
        except Exception as e:
```

```
            print(f"Error processing {image_name}: {e}")

    return 0
```

2. The output of the code is shown below. For each detected feature, the coordinates of the bounding box is provided.


```
facial analysis for people_with_faces.png:
[{'BoundingBox': {'Width': 0.18083573877811432, 'Height': 0.4653489589691162, 'Left':
 0.23514479398727417, 'Top': 0.20818035304546356}, 'Landmarks': [{'Type': 'eyeLeft',
'X': 0.29482248425483704, 'Y': 0.388049840927124}, {'Type': 'eyeRight', 'X': 0.374631
22606277466, 'Y': 0.3974248468875885}, {'Type': 'mouthLeft', 'X': 0.29231223464012146
, 'Y': 0.5370317101478577}, {'Type': 'mouthRight', 'X': 0.35929590463638306, 'Y': 0.5
447272062301636}, {'Type': 'nose', 'X': 0.33712297677993774, 'Y': 0.49112316966056824
}], 'Pose': {'Roll': 3.905557870864868, 'Yaw': 7.637889385223389, 'Pitch': -19.255167
00744629}, 'Quality': {'Brightness': 61.46929931640625, 'Sharpness': 78.6435012817382
8}, 'Confidence': 99.99952697753906}, {'BoundingBox': {'Width': 0.1695878803730011, '
Height': 0.43042850494384766, 'Left': 0.7369900941848755, 'Top': 0.26204851269721985}
, 'Landmarks': [{'Type': 'eyeLeft', 'X': 0.7608891129493713, 'Y': 0.4412580132484436}
, {'Type': 'eyeRight', 'X': 0.8304799199104309, 'Y': 0.4309614896774292}, {'Type': 'm
outhLeft', 'X': 0.7761893272399902, 'Y': 0.5804926156997681}, {'Type': 'mouthRight',
'X': 0.8343128561973572, 'Y': 0.5725774168968201}, {'Type': 'nose', 'X': 0.7818950414
657593, 'Y': 0.5175009369850159}], 'Pose': {'Roll': -6.425829887390137, 'Yaw': -21.25
846290588379, 'Pitch': -0.17397046089172363}, 'Quality': {'Brightness': 82.9406585693
3594, 'Sharpness': 73.32209777832031}, 'Confidence': 99.99952697753906}, {'BoundingBo
x': {'Width': 0.16280005872249603, 'Height': 0.4405725598335266, 'Left': 0.0874923989
1767502, 'Top': 0.2222963124513626}, 'Landmarks': [{'Type': 'eyeLeft', 'X': 0.1566646
844148636, 'Y': 0.4024817645549774}, {'Type': 'eyeRight', 'X': 0.22591157257556915, '
Y': 0.39458730816841125}, {'Type': 'mouthLeft', 'X': 0.16228783130645752, 'Y': 0.5500
428676605225}, {'Type': 'mouthRight', 'X': 0.21997620165348053, 'Y': 0.54232132434844
97}, {'Type': 'nose', 'X': 0.2160029262304306, 'Y': 0.47530367970466614}], 'Pose': {'
Roll': -0.8922376036643982, 'Yaw': 26.049856185913086, 'Pitch': -0.10571860522031784}
, 'Quality': {'Brightness': 71.64167785644531, 'Sharpness': 67.22731018066406}, 'Conf
idence': 99.99971008300781}, {'BoundingBox': {'Width': 0.15335309505462646, 'Height':
 0.43857285380363464, 'Left': 0.5824416279792786, 'Top': 0.22930672764778137}, 'Landm
arks': [{'Type': 'eyeLeft', 'X': 0.6128487586975098, 'Y': 0.3855222761631012}, {'Type
': 'eyeRight', 'X': 0.6837011575698853, 'Y': 0.39103224873542786}, {'Type': 'mouthLef
t', 'X': 0.6199813485145569, 'Y': 0.5292211174964905}, {'Type': 'mouthRight', 'X': 0.
6792996525764465, 'Y': 0.5344579815864563}, {'Type': 'nose', 'X': 0.6395815014839172,
 'Y': 0.4736301004886627}], 'Pose': {'Roll': -0.03951301798224449, 'Yaw': -12.4487447
73864746, 'Pitch': -2.851287841796875}, 'Quality': {'Brightness': 72.50347900390625,
'Sharpness': 78.64350128173828}, 'Confidence': 99.99987030029297}, {'BoundingBox': {'
Width': 0.15609972178936005, 'Height': 0.4269851744174957, 'Left': 0.4176094830036163
3, 'Top': 0.15845444798469543}, 'Landmarks': [{'Type': 'eyeLeft', 'X': 0.465683728456
4972, 'Y': 0.32279446721076965}, {'Type': 'eyeRight', 'X': 0.5358383655548096, 'Y': 0
.3253035545349121}, {'Type': 'mouthLeft', 'X': 0.4680750370025635, 'Y': 0.45728719234
46655}, {'Type': 'mouthRight', 'X': 0.5267676115036011, 'Y': 0.4594101011753082}, {'T
ype': 'nose', 'X': 0.5017674565315247, 'Y': 0.40200287103652954}], 'Pose': {'Roll': 2
.317734956741333, 'Yaw': 3.2928566932678223, 'Pitch': -0.36501815915107727}, 'Quality
': {'Brightness': 84.46928405761719, 'Sharpness': 78.64350128173828}, 'Confidence': 9
9.99991607666016}]
```


Section 10: Text Extraction

1. The code below performs text extraction on the images listed in section 6:
    - A rekognition client object is created using the boto3 library. This object is used to make calls to the Amazon Rekognition service.

- For each image, the code uses the detect_text method of the Rekognition client to identify and extract text present in the image. This function can recognize and extract both printed and handwritten text.

```
client = boto3.client('rekognition')
def main(argv):
    for image in bucket_images:
        image_name = image['S3Object']['Name']
        try:
            response = client.detect_text(Image=image)
            print('text extraction for %s:' % image_name)
            print(response['TextDetections'])
            print('\n')
        except Exception as e:
            print(f"Error processing {image_name}: {e}")

    return 0
```

2. The output of the code is shown below. As seen in the screenshot, the first line of the text `How To` was detected.

```
text extraction for image_with_text.jpg:
[{'DetectedText': 'HOW TO', 'Type': 'LINE', 'Id': 0, 'Confidence': 99.43235778808594,
 'Geometry': {'BoundingBox': {'Width': 0.11992187798023224, 'Height': 0.0427083335816
8602, 'Left': 0.4390624463558197, 'Top': 0.17499998211860657}, 'Polygon': [{'X': 0.43
90624463558197, 'Y': 0.17499998211860657}, {'X': 0.5589843392372131, 'Y': 0.174999982
11860657}, {'X': 0.5589843392372131, 'Y': 0.21770831942558289}, {'X': 0.4390624463558
197, 'Y': 0.21770831942558289}]}}, {'DetectedText': 'ADD TEXT', 'Type': 'LINE', 'Id':
 1, 'Confidence': 99.35179138183594, 'Geometry': {'BoundingBox': {'Width': 0.42929688
09604645, 'Height': 0.20624999701976776, 'Left': 0.28515625, 'Top': 0.269791662693023
7}, 'Polygon': [{'X': 0.28515625, 'Y': 0.2697916626930237}, {'X': 0.7144531011581421,
```