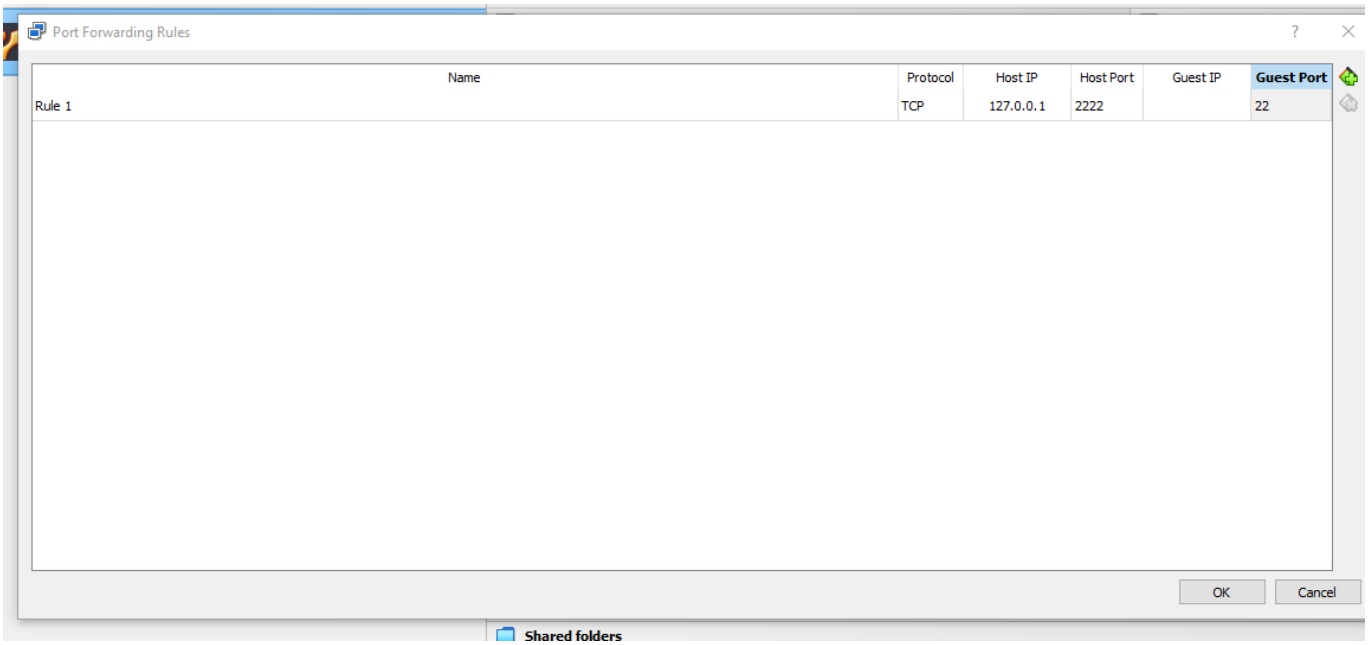
Lab Report 5 - 9

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

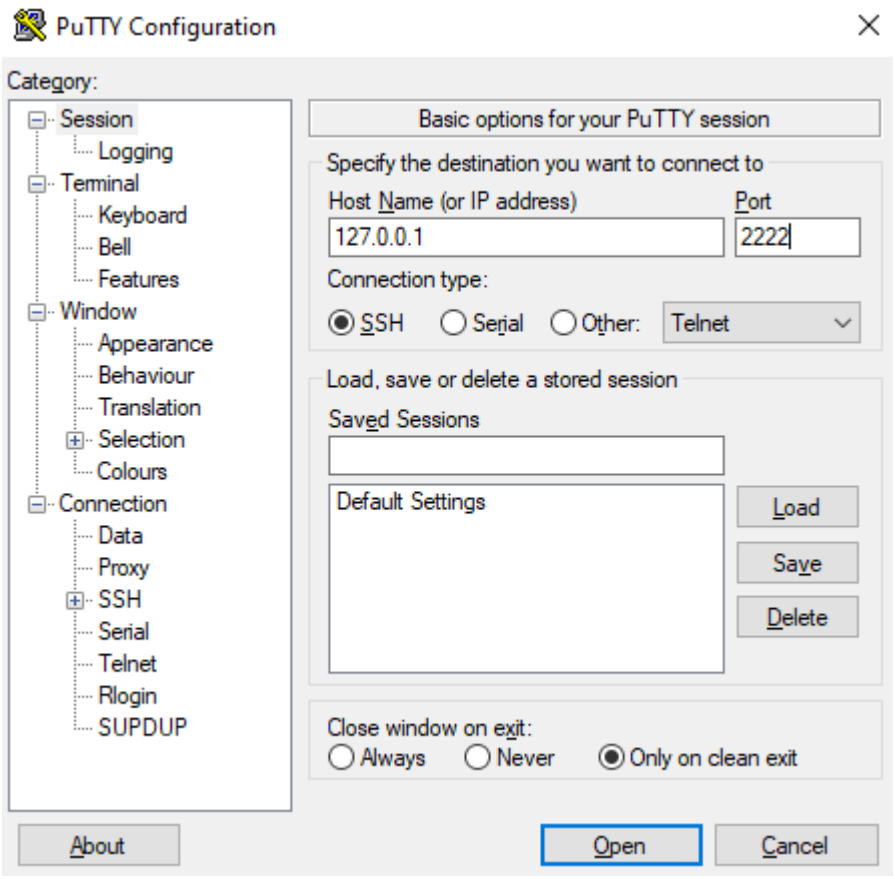
```
jookai@jookai:~$ sudo apt install tasksel
```

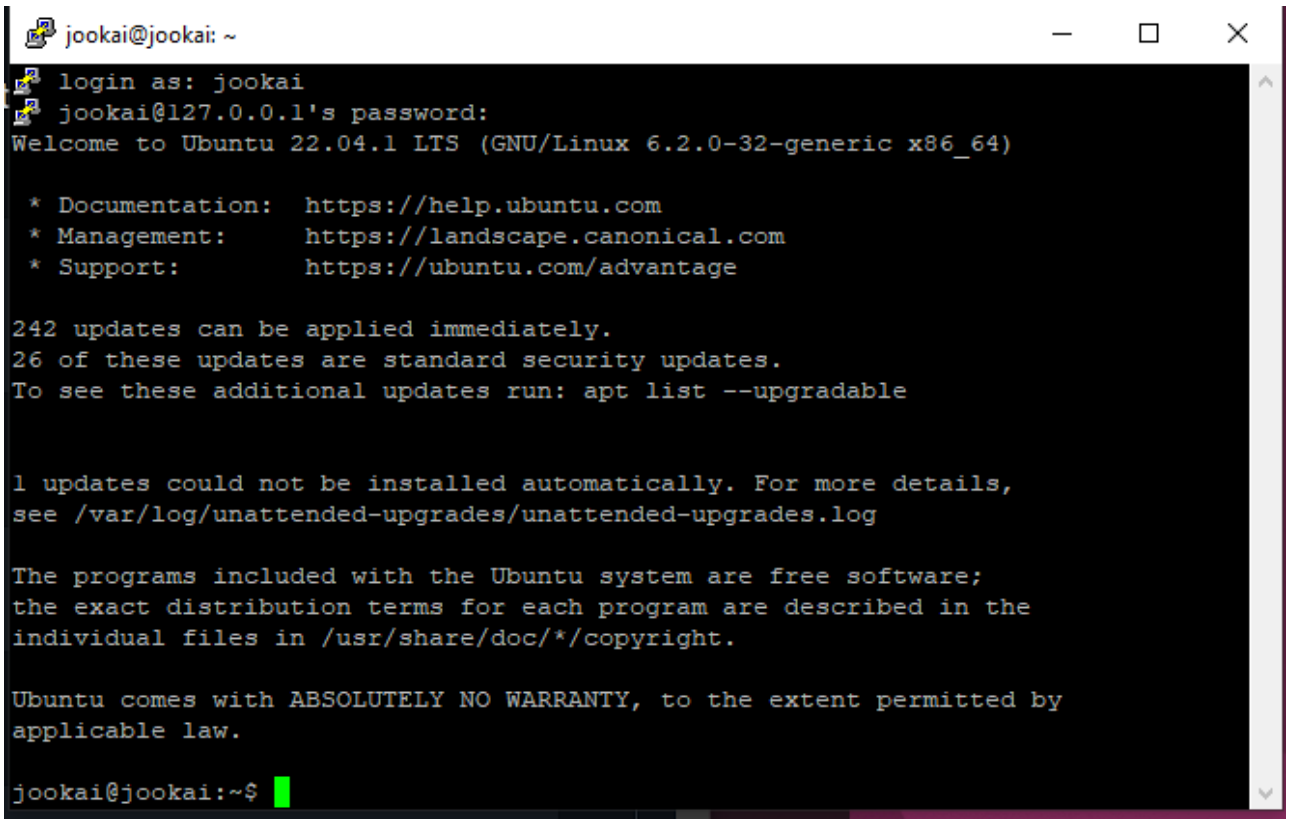
```
jookai@jookai:~$ sudo tasksel install openssh-server
```

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

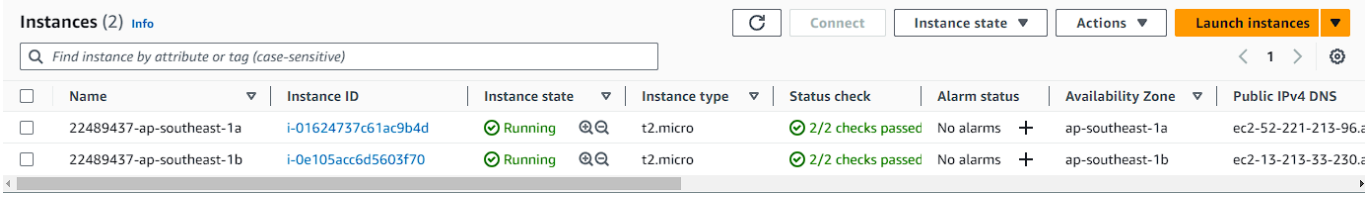
1. The following function is used to create 2 EC2 instances in two different availability zones of ap-southeast-1. The reason ap-southeast-1 was used instead of ap-southeast-2 was due to the limit in VPCUs on ap-southeast-2 which did not allow for any new EC2 instances to be created on the region at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

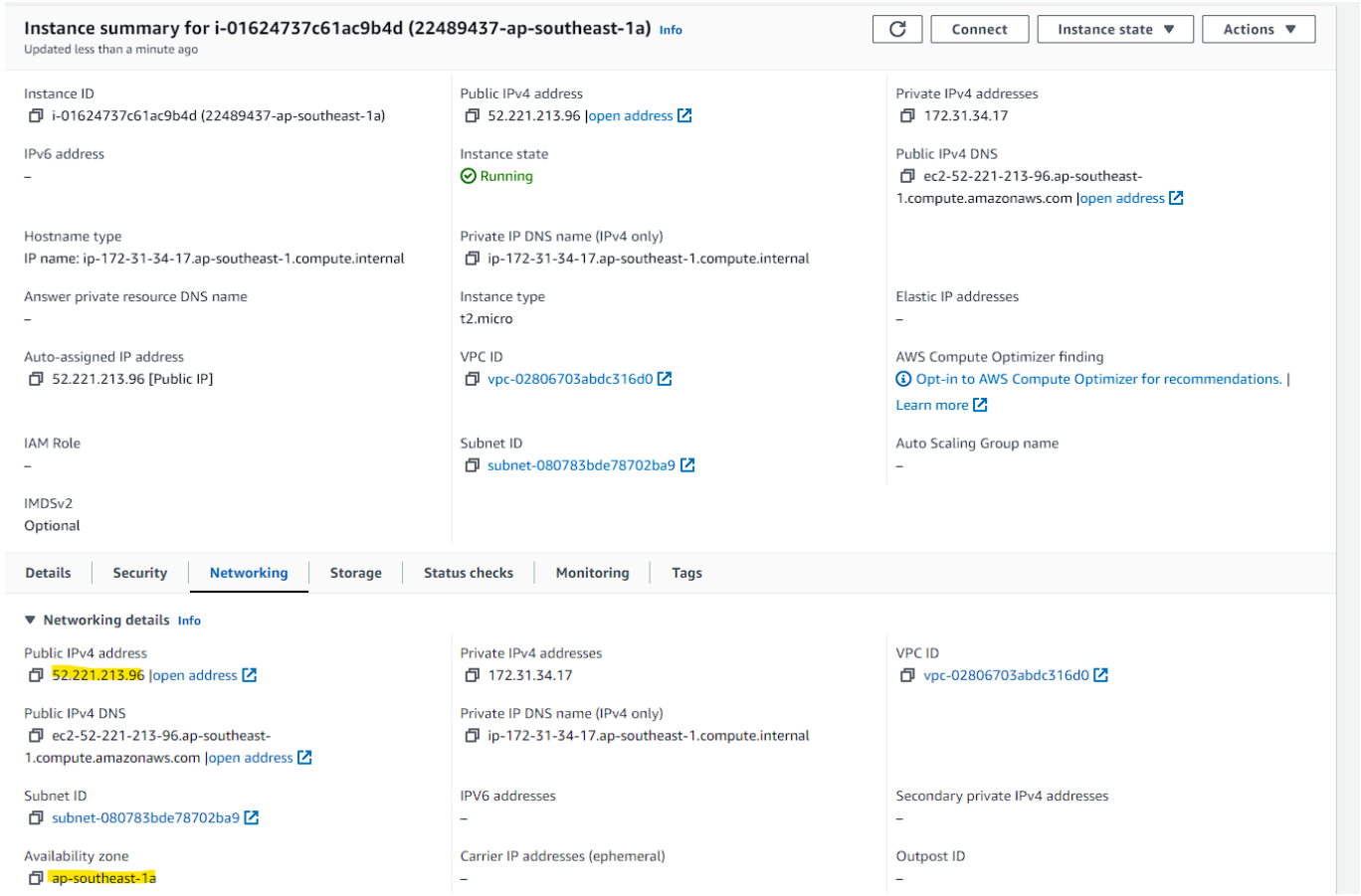
```
def launch_ec2_instances():
    # Create a security group
    response = ec2.create_security_group(
        GroupName=f"{student_number}-sg",
        Description="security group for development environment"
    )
    security_group_id = response['GroupId']

    # 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,
```

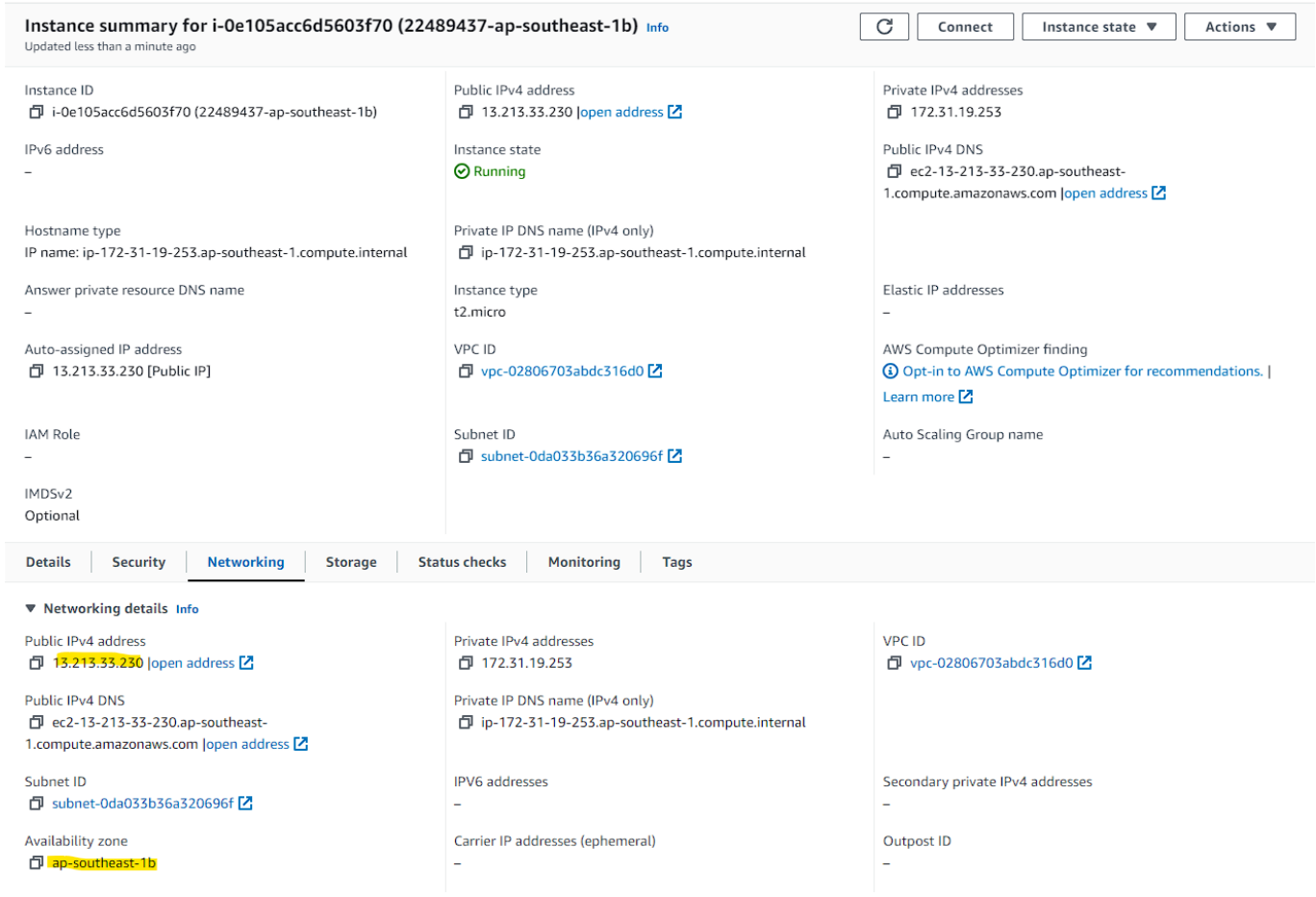
```
            'Placement': {'AvailabilityZone': az},
            'TagSpecifications': [
                {
                    'ResourceType': 'instance',
                    'Tags': [{'Key': 'Name', 'Value': instance_name}]
                }
            ]
        }

        # 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

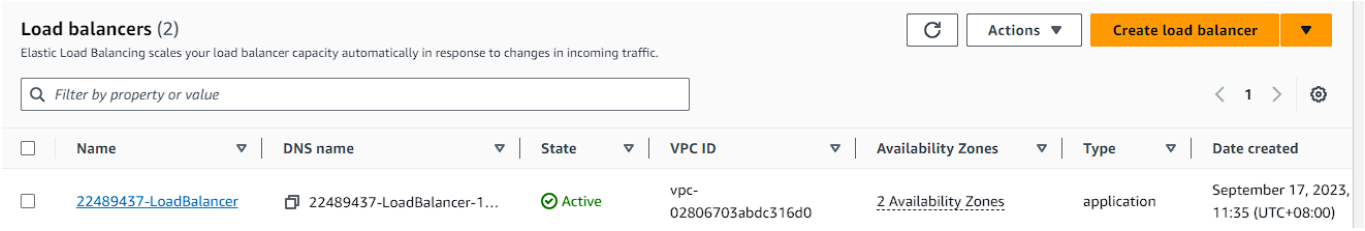
Protocol: HTTP and Port 80 forwarding on to the target group c. The code creates a target group using the VPC from step 1 d. The code registers the two EC2 instances from step 1 as targets

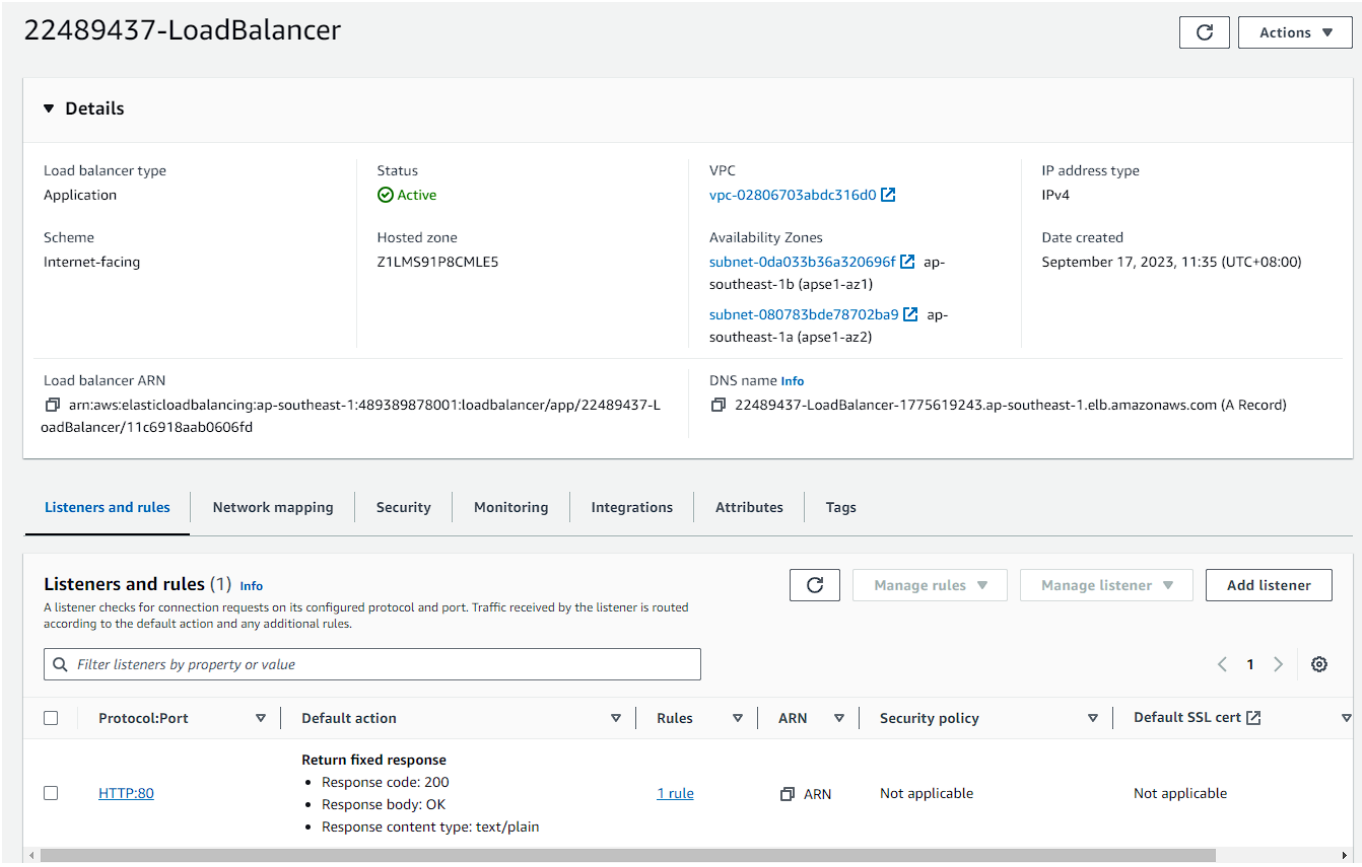
```
def create_load_balancer():
    vpc_id = 'vpc-02806703abdc316d0'
    security_group_id = 'sg-0021774194b407020'
    subnet_ids = ['subnet-080783bde78702ba9', 'subnet-0da033b36a320696f']

    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}")
```

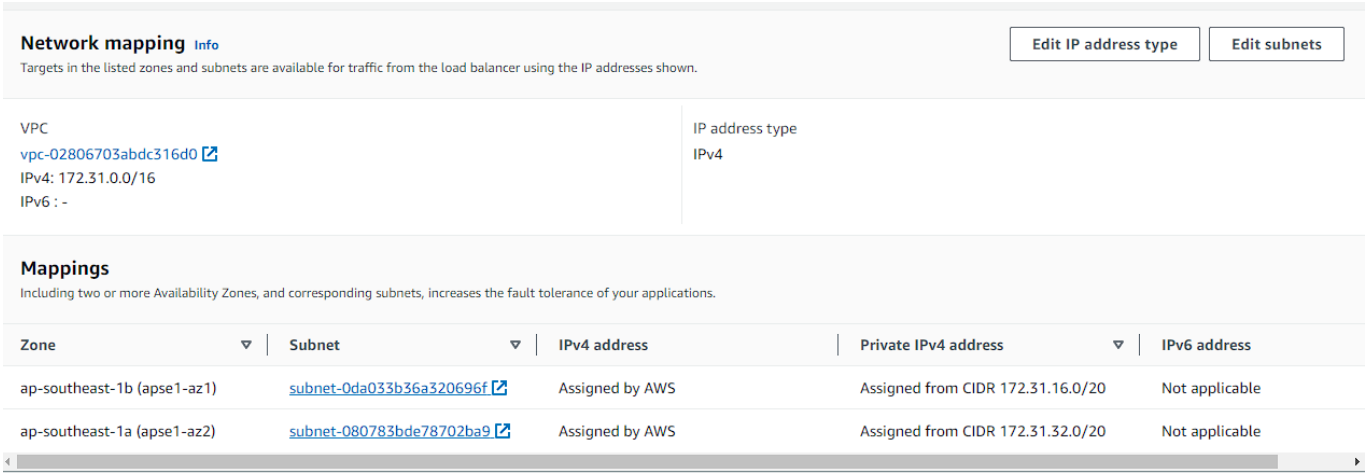
```
    instance_1_id = 'i-01624737c61ac9b4d'
    instance_2_id = 'i-0e105acc6d5603f70'

    # 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.")
```

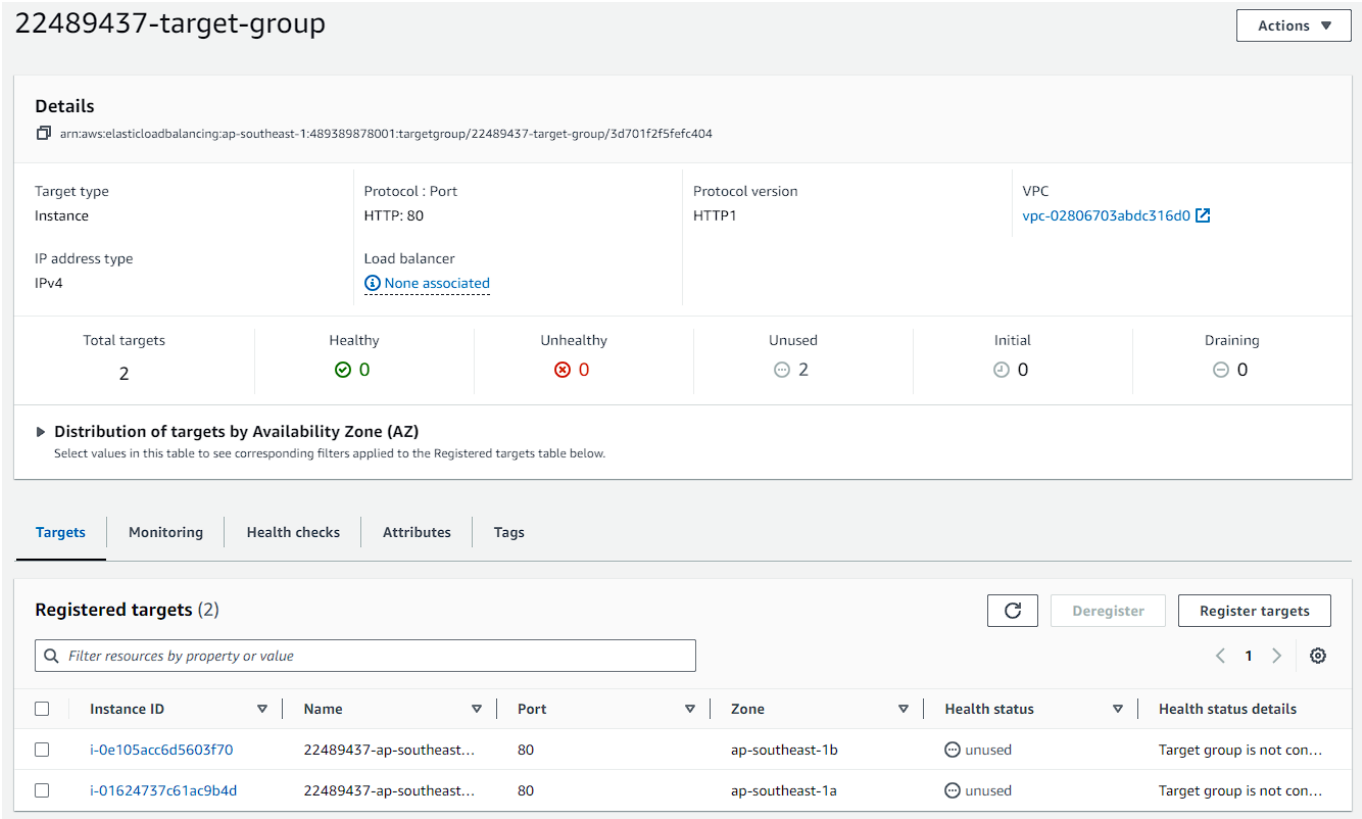
The following screenshots show the output of running the code as well as the results in the AWS terminal.

```
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

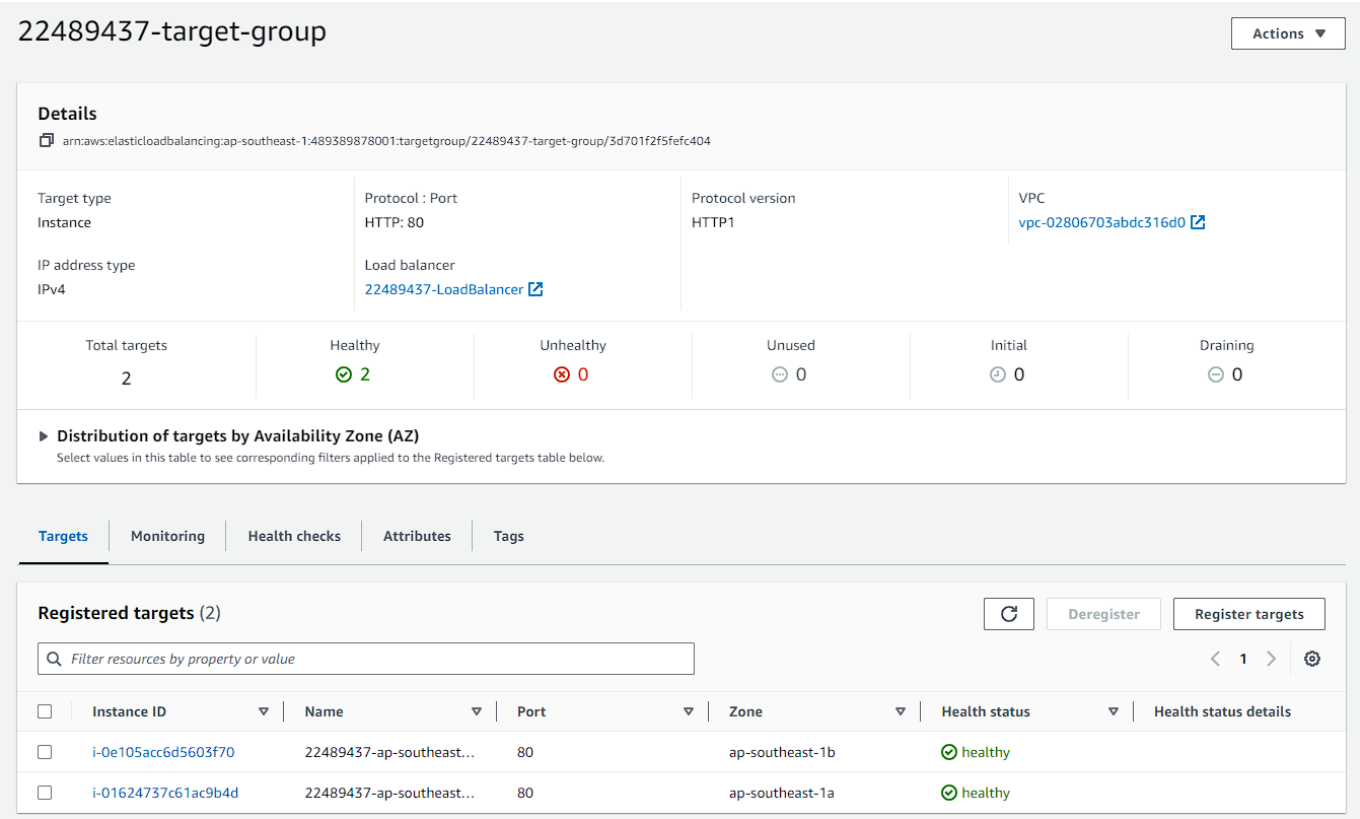

Network mapping Info
Targets in the listed zones and subnets are available for traffic from the load balancer using the IP addresses shown.
Edit IP address type
Edit subnets
VPC
vpc-02806703abdc316d0
IPv4: 172.31.0.0/16
IPv6 : -
IP address type
IPv4
Mappings
Including two or more Availability Zones, 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

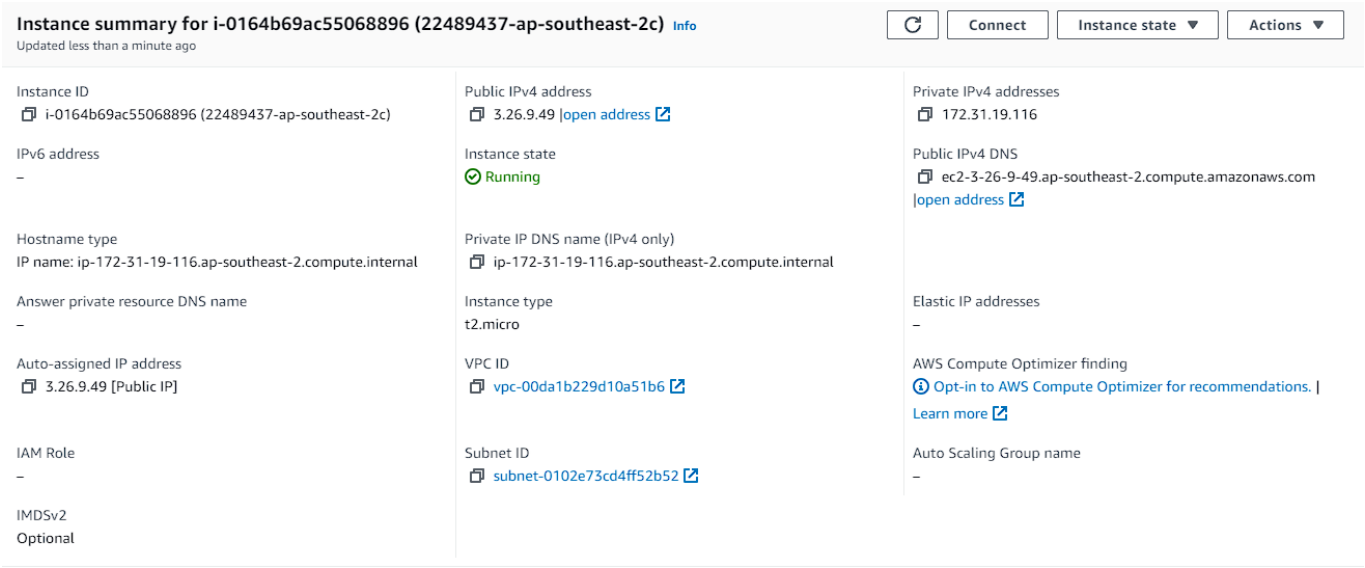
Section 1: Create an EC2 Instance

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;
  }
}
```

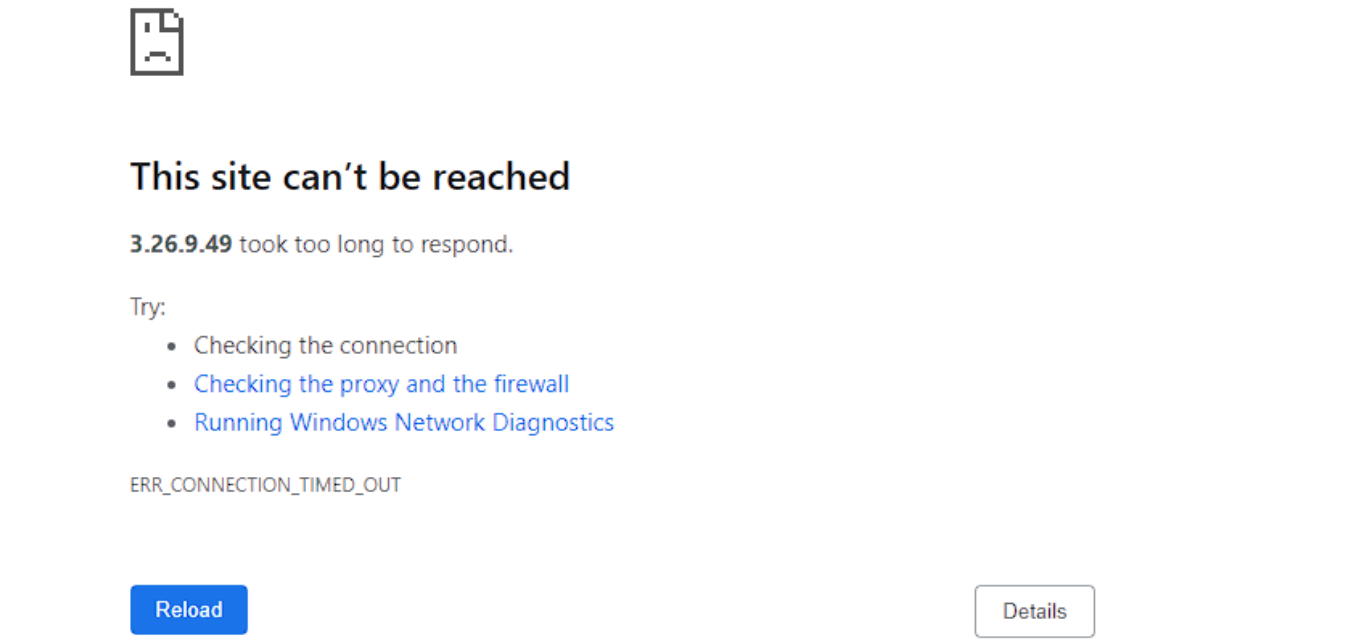
2. Restarting Nginx so that the changes from step 1 take effect:

```
(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.