CAPSTONE PROJECT – Migrating to Cloud

Project Objective

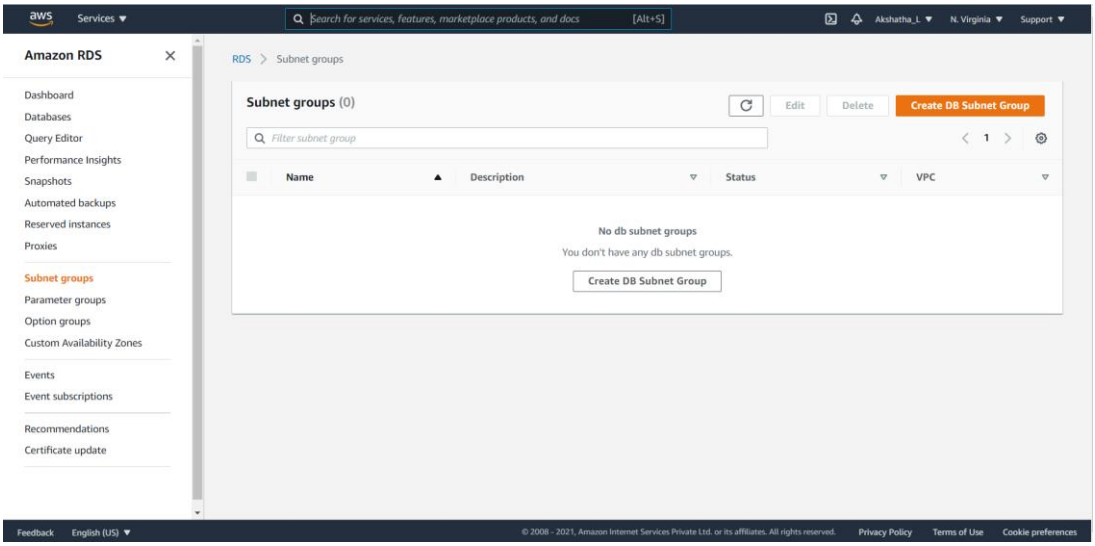
- 'UrbanSloth' is a unicorn food delivery apps in India which has unexpected increase in the load on their application.
- The objective of the project is to set-up the POC application , which is part of the application hosted on the third party rented infrastructure , in AWS.
- The migration of the application must happen without any loss of data.
- The application hosted on AWS should be efficient, reliable and secure.
- Appropriate AWS services must be selected for the migration based on the requirements.
- After migration of the POC application, end-to-end testing must be done to ensure the application works as expected.

Project Requirements

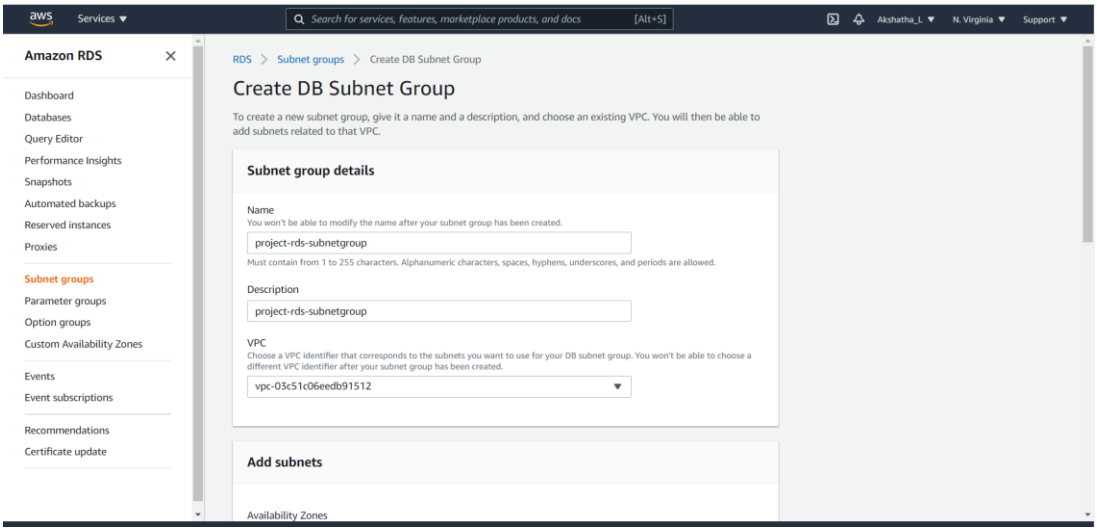
- Migrate the on-premise database onto the database server running on the cloud making sure of secure access to it. That is, for instance it need not be accessible from outside the VPC and should be accessible only from the EC2 instance where the Python application is installed.
- On an EC2 instance, replicate all the dependencies and application libraries and configuration from the on-premise server where the Python application is set up.
- Configure the application to run on normal http port instead of testing & development port 5000.
- Set up auto-scaling group with application load balancer for the above application configuration.

Project Implementation

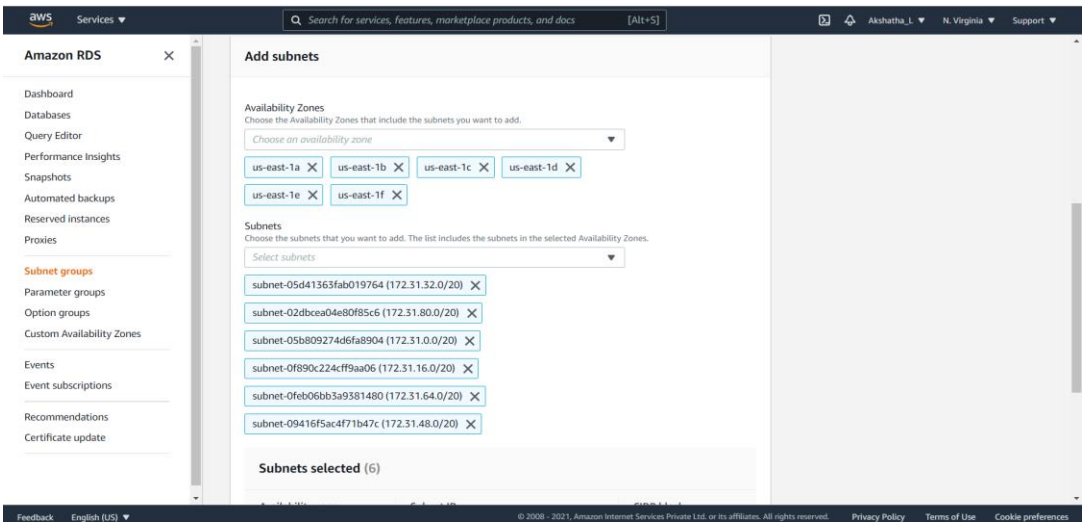
- ### Setting up the database server (RDS) on AWS.
    1. Creating DB Subnet group.
        a. Navigate to the 'RDS' service in AWS Account. Click on the 'subnet groups', and 'Create DB subnet group'.



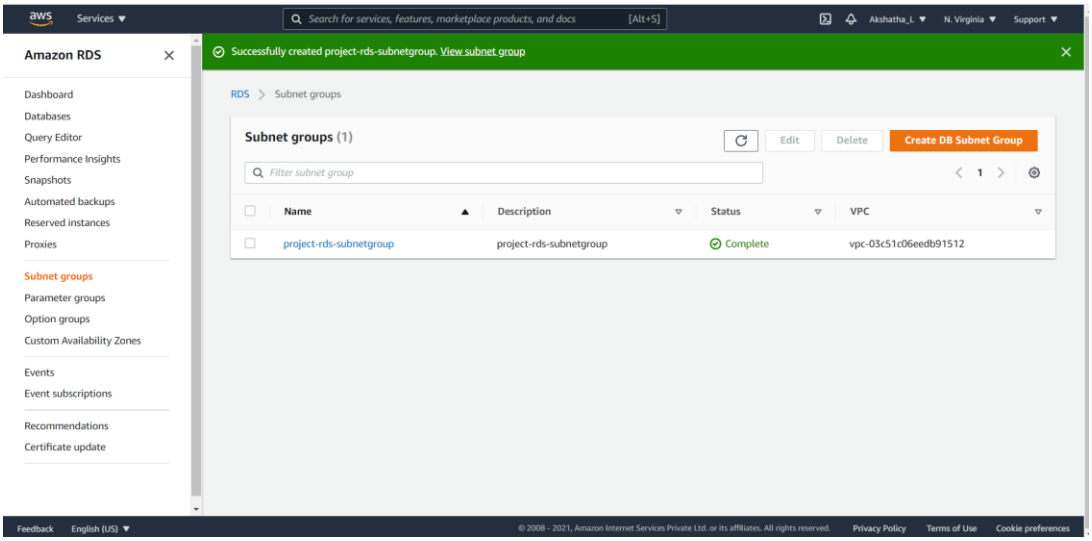
b. Name of the subnet group – ‘project-rds-subnetgroup’. The VPC in which the subnet group must be created should also be specified. (Choosing default VPC)



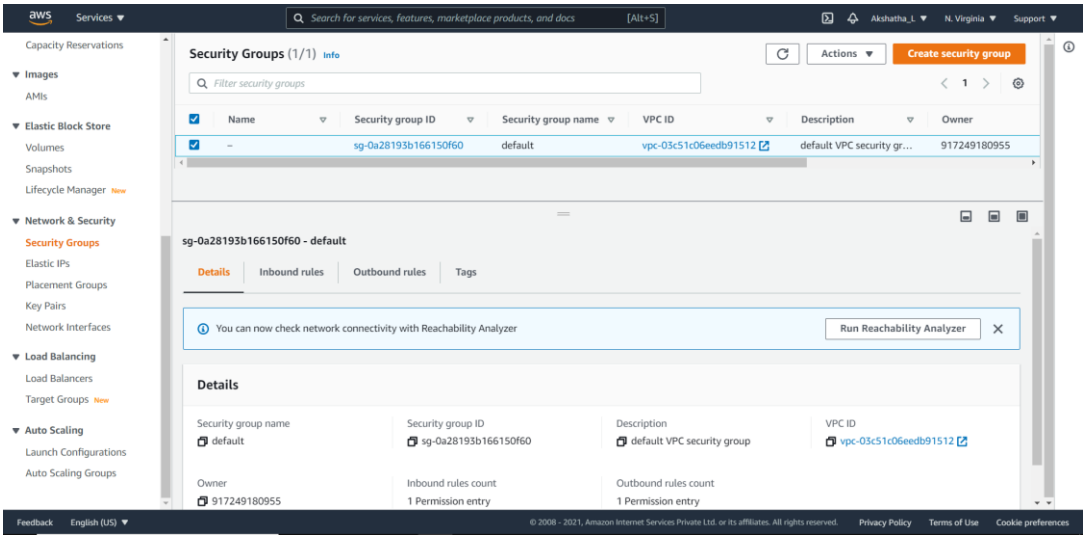
c. Choose all the availability zones and subnets in the region to ensure availability of the database.



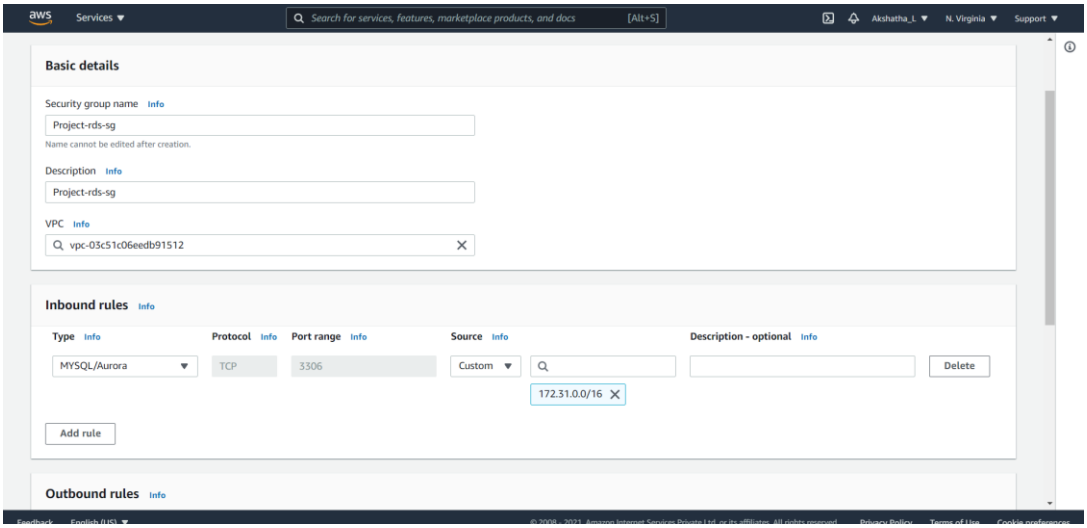
d. The DB Subnet group is created successfully.



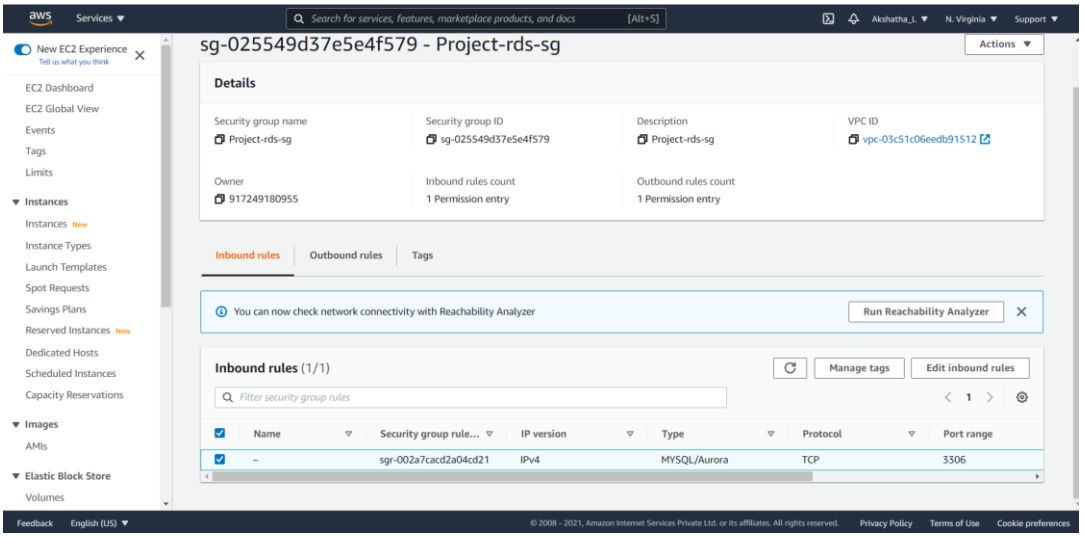
2. Create security group for RDS.
   a. Navigate to ‘ec2’ service in the AWS Dashboard and go to ‘Security group’. A Default Security group will already be present. Create a new security group.



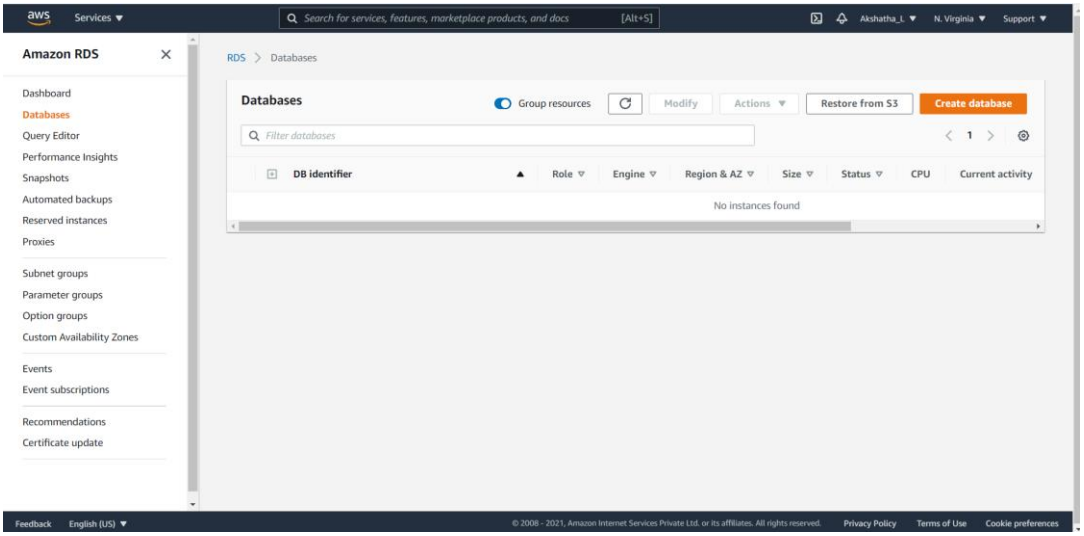
   b. Appropriate name and description must be given while creating the security group(‘Project-rds-sg’). Add an inbound rule to group , where the Type is ‘MYSQL’ , port is 3306 and the source is the CIDR block of the VPC in which the RDS will be created.(Default VPC)



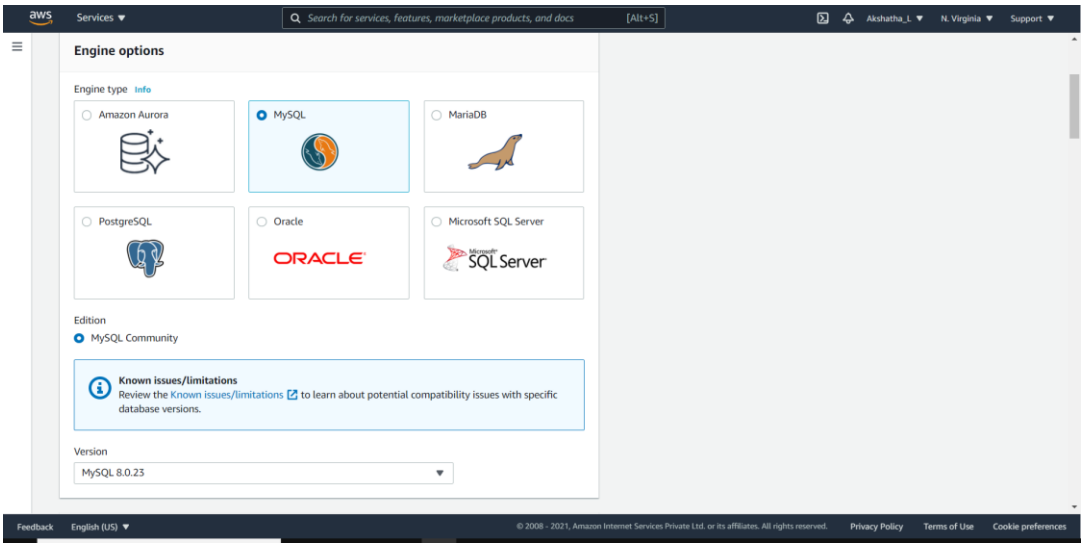
c. Security group is created successfully.



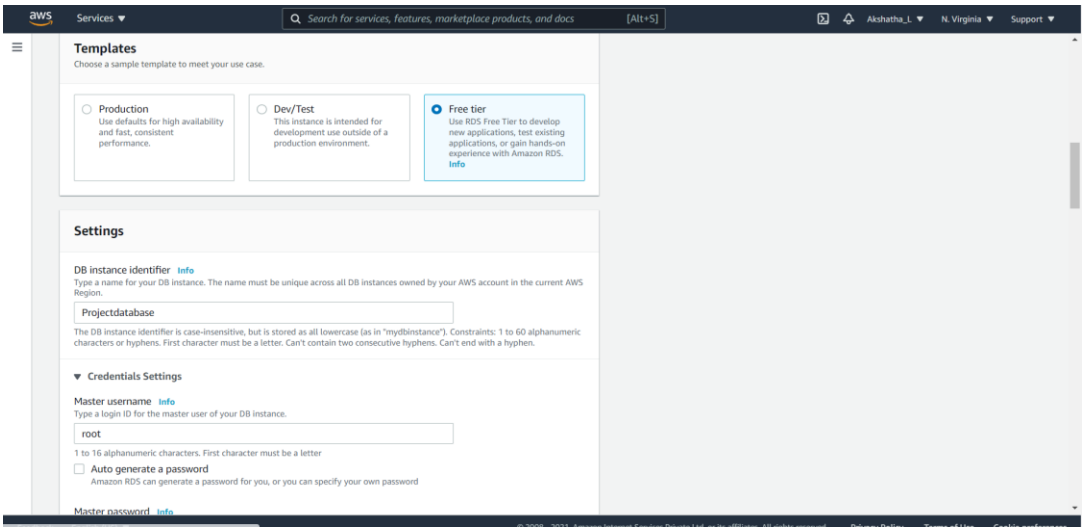
3. Create an RDS instance.
    a. Navigate to the ‘RDS’ service in the AWS Dashboard, and create a new database.



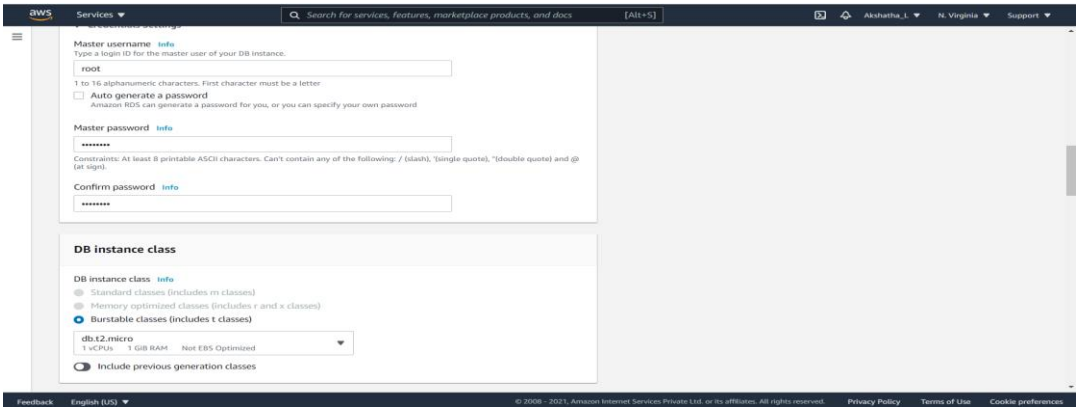
b. Choose Engine Type as ‘MySQL’ and select appropriate edition and version.



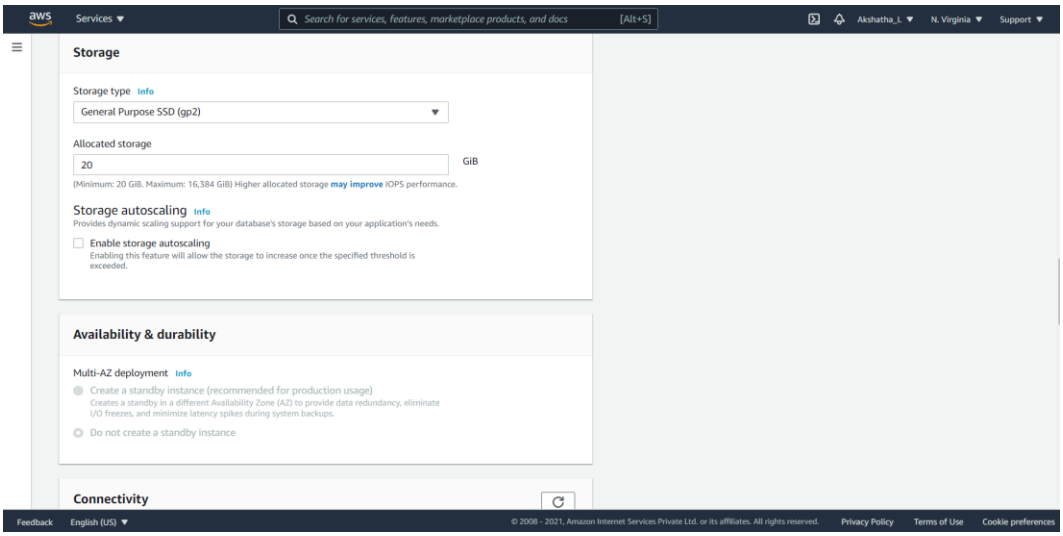
c. Choose ‘Free Tier’ as template. Also, an appropriate name for the RDS instance.



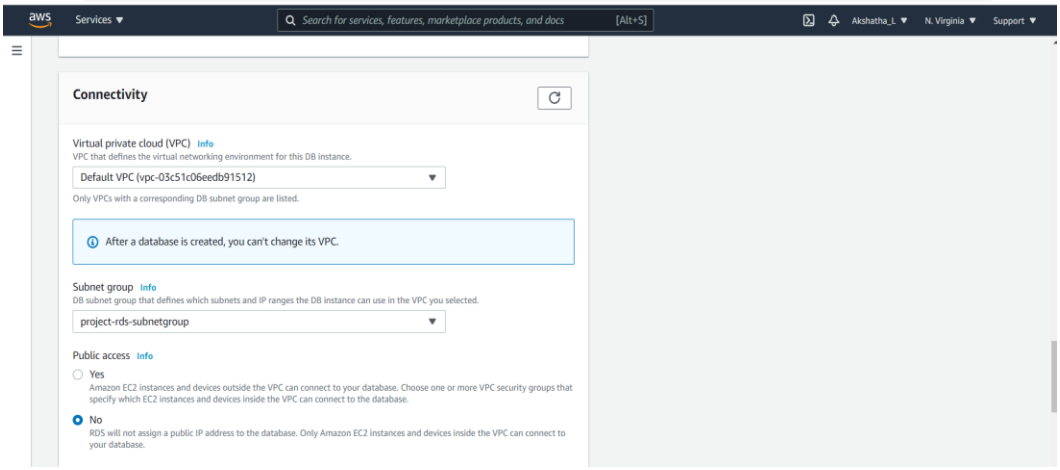
d. The RDS must have a username and password. Password Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), "(double quote) and @ (at sign). The DB Instance class is – ‘db.t2.micro’.



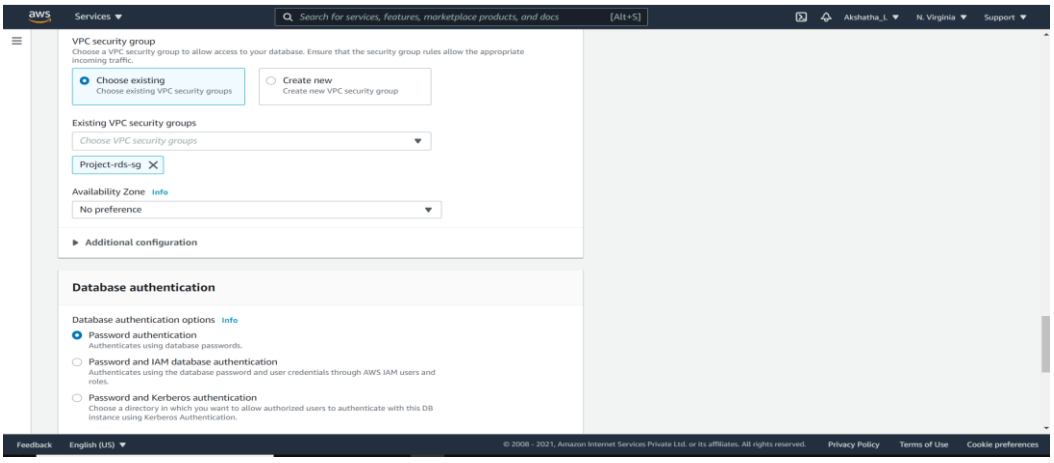
e. Allocate 20 GB storage for the database and disable storage autoscaling.



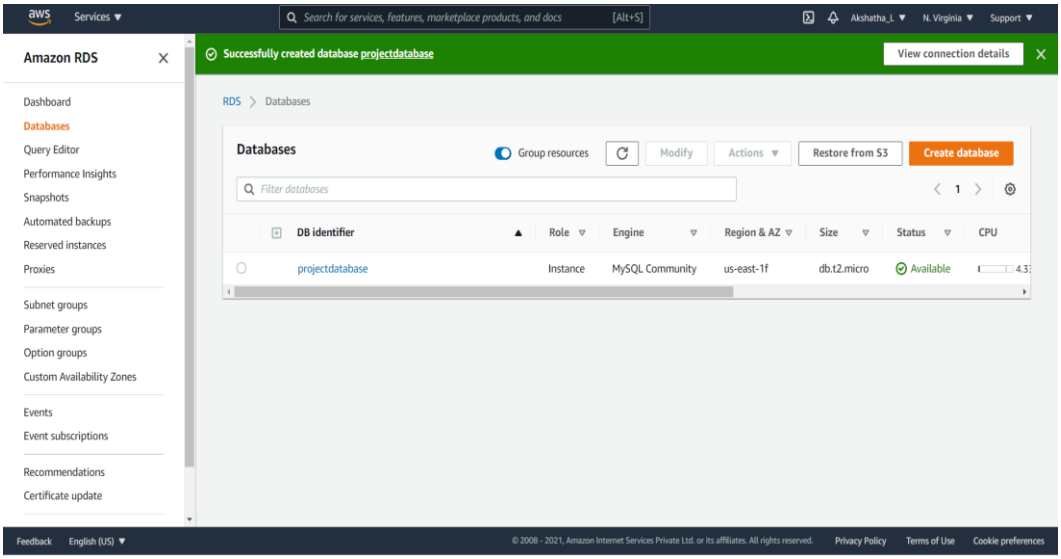
f. Choose the default VPC for the RDS and choose the subnet group which was created previously. The RDS must not have public access.



g. Choose the security group which was created previously. Choose Password authentication for database authentication.



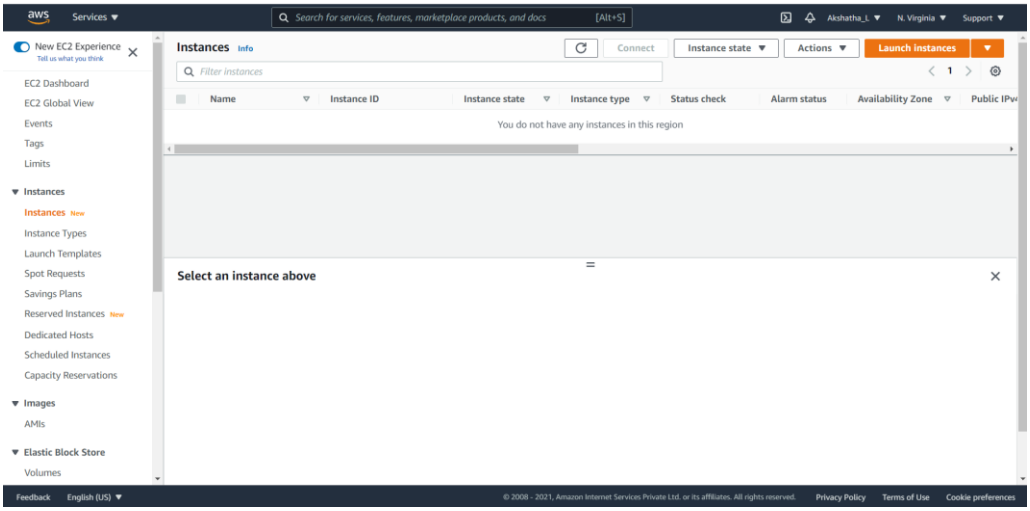
h. The RDS Instance is created successfully. (projectdatabase)



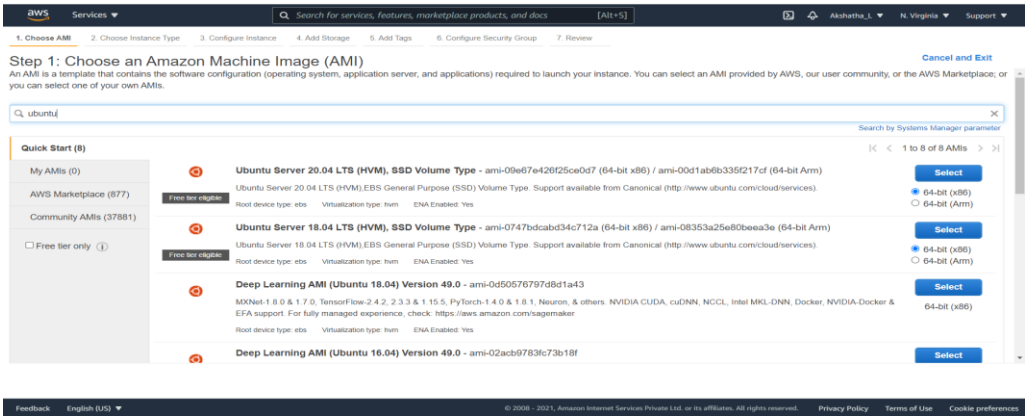
- Launching and setting up EC2 Instance.

1. Launch an Ubuntu EC2 instance.

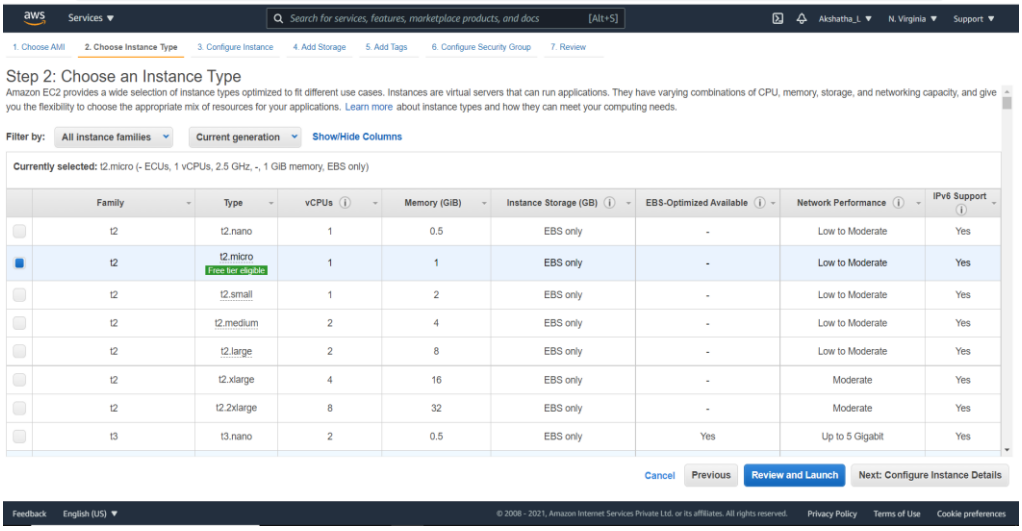
a. Navigate to the ‘EC2’ service in the AWS Dashboard. Click on ‘Launch instance’.



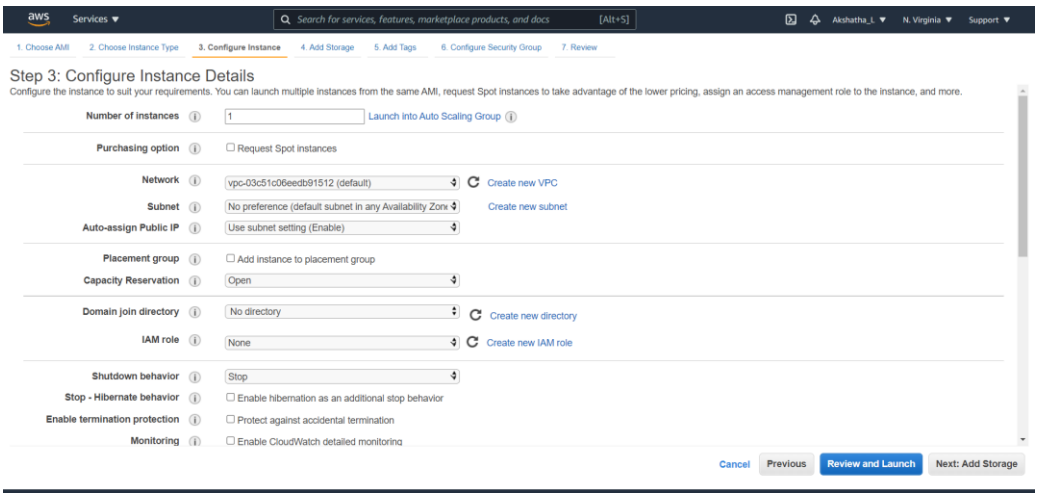
b. Select the ‘ubuntu-20.04’ AMI



c. The instance type is “t2.micro”.W



d. While configuring the instance, select the Default VPC and enable the public IP for the instance.

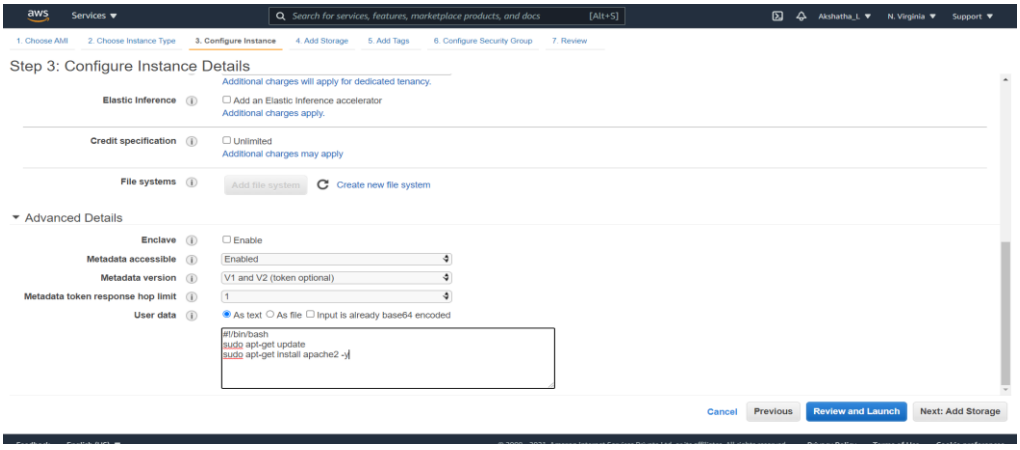


e. The below bash script must be added in the userdata section:

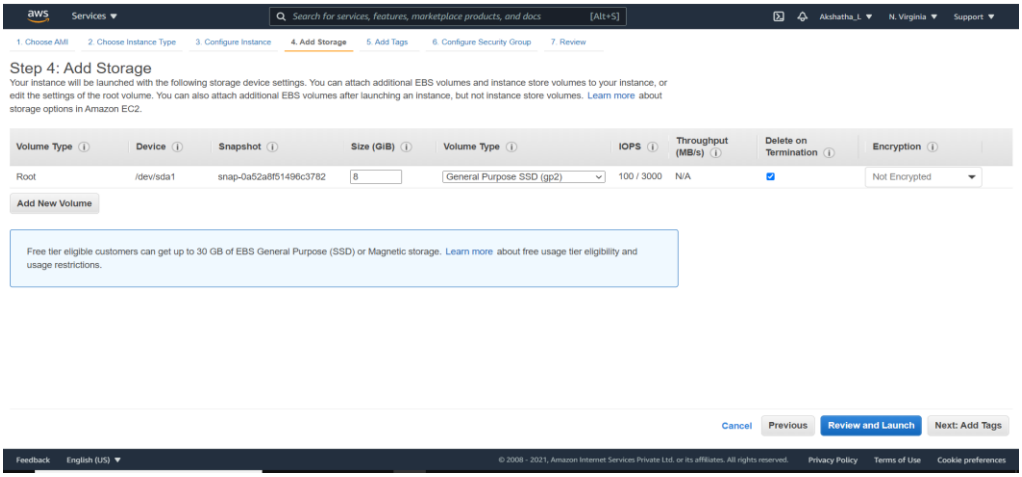
#!/bin/bash

sudo apt-get update

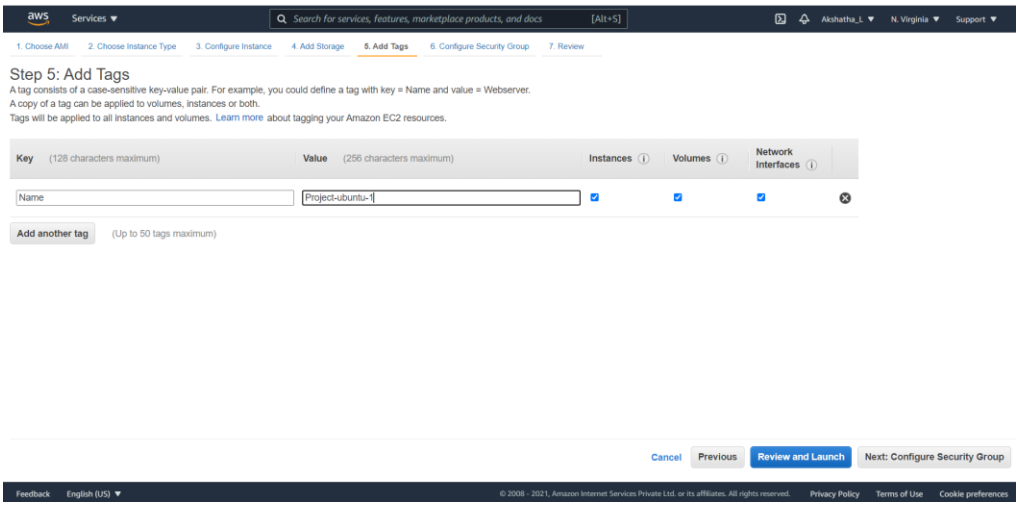
sudo apt-get install apache2 -y



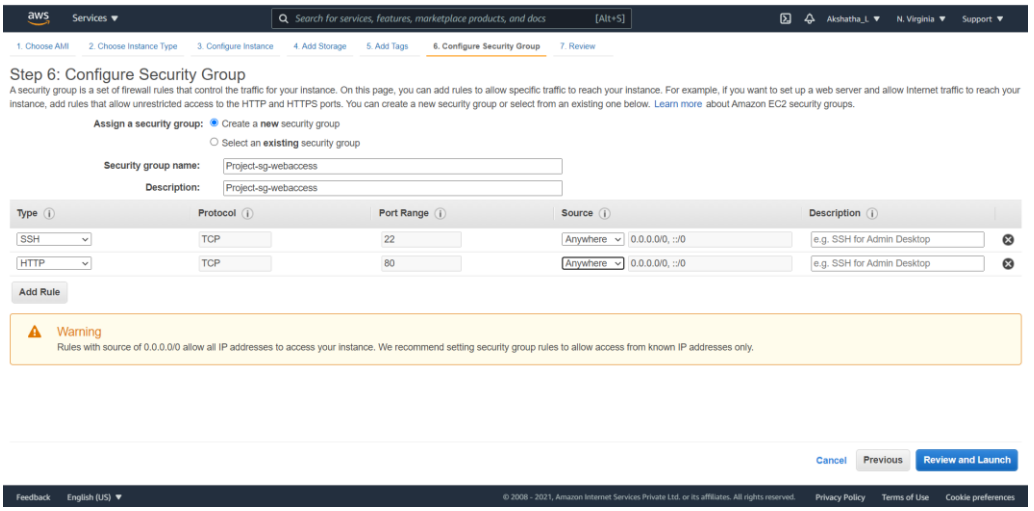
f. Allocate 8GB storage for the instance.



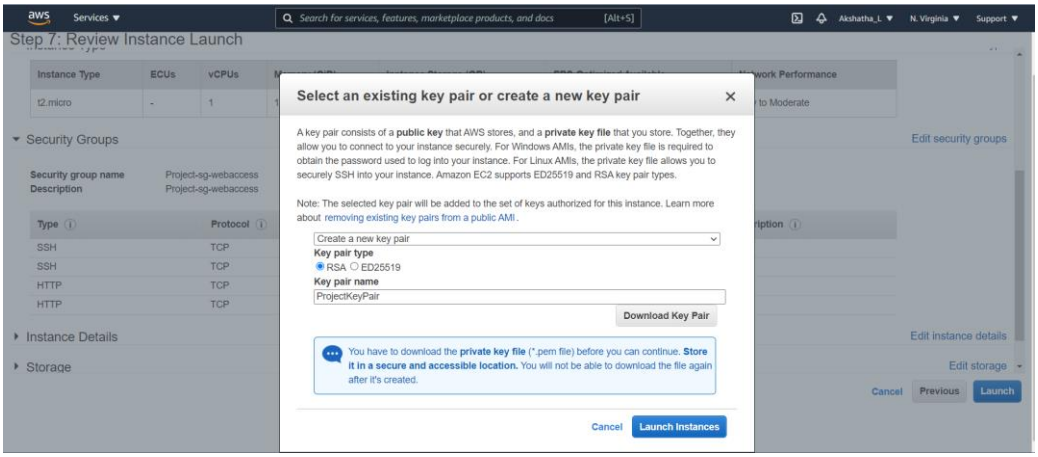
g. Appropriate name tag must be given for the instance(“Project-ubuntu-1”)



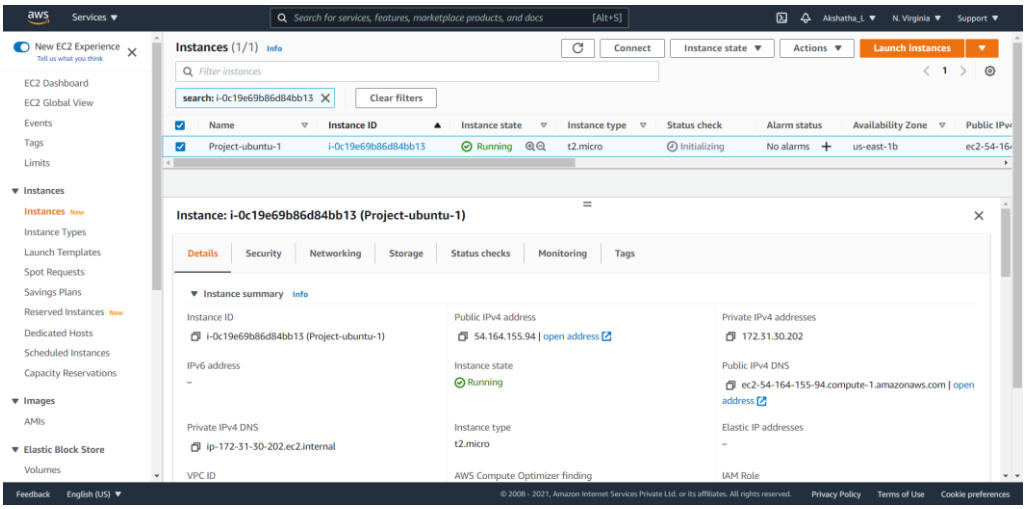
h. Create a new security group for the EC2 instance , and add inbound rules for the instance for “HTTP” type and “SSH” type and source as “Anywhere”



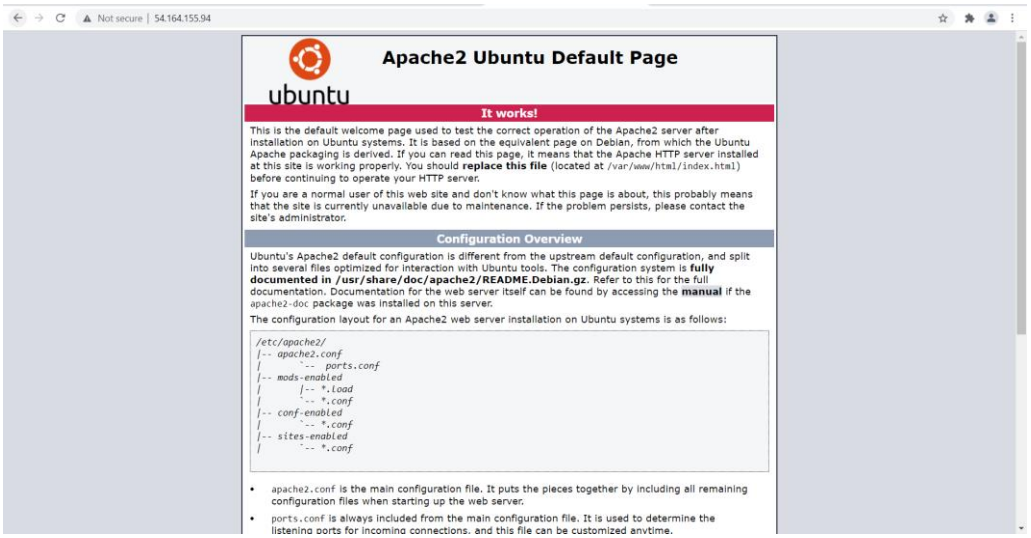
i. Launch the instance with existing or new keypair.



j. The instance is launched successfully.

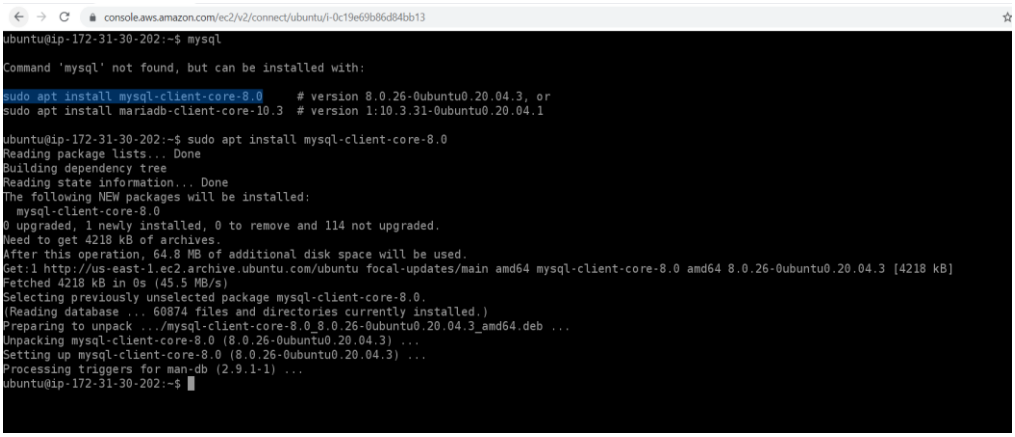


k. In a chrome tab, go to – http://<Public-IP>:80 , if the apache server is installed , the below page must be visible.

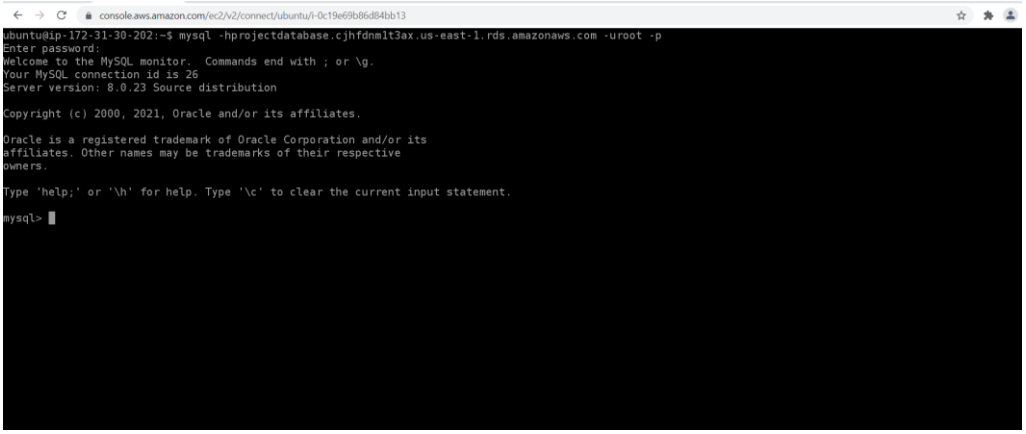


2. Install MySQL client in the EC2 instance.
    a. SSH into the instance with putty , and execute the following command –

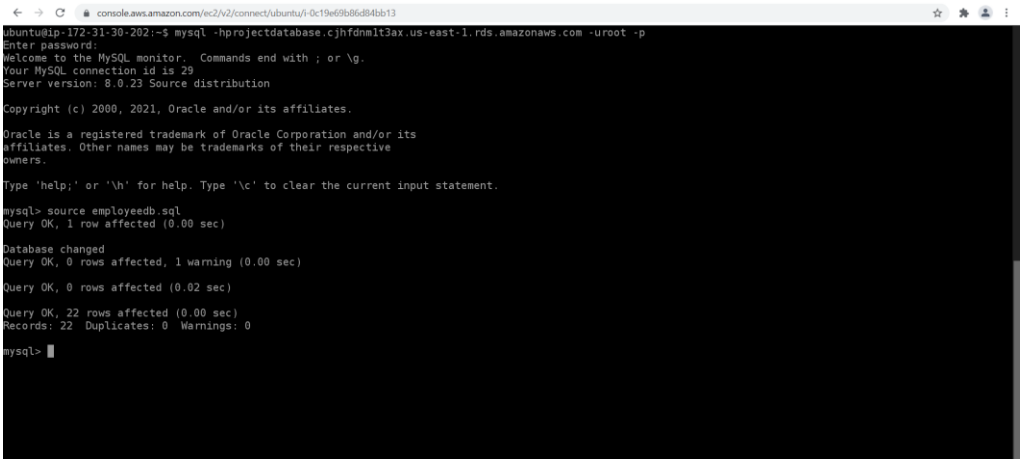
sudo apt-get update
sudo apt-get install mysql-client-core-8.0



b. Connect to the RDS Database with the command –
'mysql -h<Endpoint of RDS> -u<username> -p'



c. After copying the 'employeedb.sql' in the EC2, execute –
'source employeedb.sql'

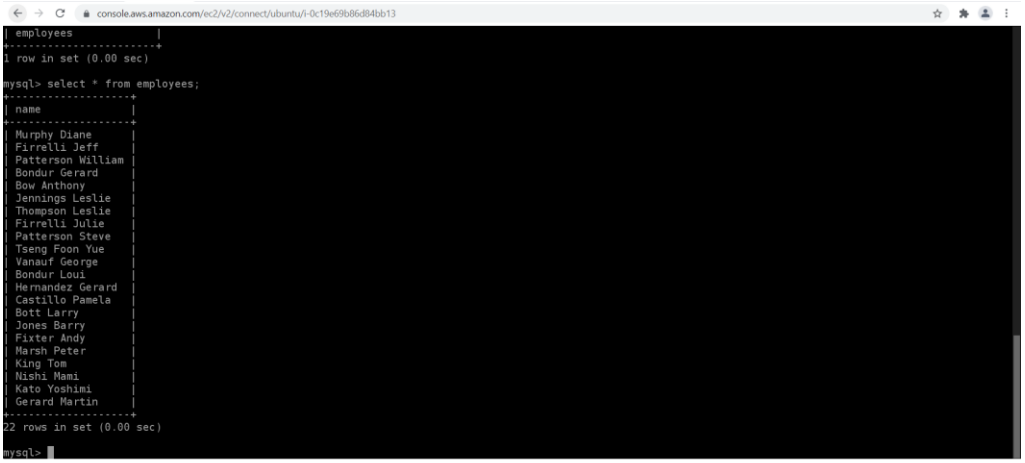


d. To ensure that the data has been restored properly , check if the database and tables are created .
show databases;
use employee_db;
*select * from employees;*

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| employee_db        |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use employee_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------------+
| Tables_in_employee_db |
+----------------------+
| employees            |
+----------------------+
1 row in set (0.00 sec)
```



3. Configuring and testing Python-Flask application.
   a. Install python, pip3 and flask

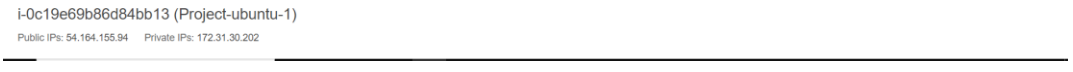
      python3 –version

      udo apt-get install python3-pip

      pip3 install flask

      pip3 install flask-mysql

```
ubuntu@ip-172-31-30-202:~$ python3 --version
Python 3.8.5
ubuntu@ip-172-31-30-202:~$ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential cpp cpp-9 dpkg-dev fakeroot g++ g++-9 gcc gcc-10-base gcc-9 gcc-9-base
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan5 libatomic1 libbinutils libc-dev-bin libc6-dev libcc1-0 libcrypt-dev
  libctf-nobfd0 libctf0 libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libgcc-9-dev libgcc-s1 libgomp1 libisl22 libitm1 liblsan0 libmpc3
  libpython3-dev libpython3.8 libpython3.8-dev libpython3.8-minimal libpython3.8-stdlib libquadmath0 libstdc++-9-dev libstdc++6 libtsan0 libubsan1
  linux-libc-dev make manpages-dev python-pip-whl python3-dev python3-wheel python3.8 python3.8-dev python3.8-minimal zlib1g-dev
Suggested packages:
  binutils-doc cpp-doc gcc-9-locales debian-keyring g++-multilib g++-9-multilib gcc-9-doc gcc-multilib autoconf automake libtool flex bison gdb gcc-doc
  gcc-9-multilib glibc-doc bzr libstdc++-9-doc make-doc python3.8-venv python3.8-doc binfmt-support
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu build-essential cpp cpp-9 dpkg-dev fakeroot g++ g++-9 gcc gcc-9 gcc-9-base libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan5 libatomic1 libbinutils libc-dev-bin libc6-dev libcc1-0 libcrypt-dev libctf-nobfd0 libctf0
  libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libgcc-9-dev libgomp1 libisl22 libitm1 liblsan0 libmpc3 libpython3-dev libpython3.8-dev
  libquadmath0 libstdc++-9-dev libtsan0 libubsan1 linux-libc-dev make manpages-dev python-pip-whl python3-dev python3-pip python3-wheel python3.8-dev
  zlib1g-dev
The following packages will be upgraded:
  gcc-10-base libgcc-s1 libpython3.8 libpython3.8-minimal libpython3.8-stdlib libstdc++6 python3.8 python3.8-minimal
8 upgraded, 50 newly installed, 0 to remove and 106 not upgraded.
Need to get 56.6 MB of archives.
After this operation, 214 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 python3.8 amd64 3.8.10-0ubuntu1~20.04 [387 kB]
```



```
ubuntu@ip-172-31-30-202:~$ pip3 install flask-mysql
Collecting flask-mysql
  Downloading Flask_MySQL-1.5.2-py2.py3-none-any.whl (3.8 kB)
Requirement already satisfied: Flask in /usr/lib/python3/dist-packages (from flask-mysql) (1.1.1)
Collecting PyMySQL
  Downloading PyMySQL-1.0.2-py3-none-any.whl (43 kB)
     |████████████████████████████████| 43 kB 3.5 MB/s
Installing collected packages: PyMySQL, flask-mysql
Successfully installed PyMySQL-1.0.2 flask-mysql-1.5.2
ubuntu@ip-172-31-30-202:~$
```

i-0c19e69b86d84bb13 (Project-ubuntu-1)

Public IPs: 54.164.155.94 Private IPs: 172.31.30.202

b. Copy the app.py code into the EC2 instance , and change the following in the code-

app.config['MYSQL_DATABASE_USER'] = '<your admin username>'

app.config['MYSQL_DATABASE_PASSWORD'] = '<your admin user password>'

app.config['MYSQL_DATABASE_DB'] = 'employee_db'

app.config['MYSQL_DATABASE_HOST'] = '<your RDS instance end point>'

console.aws.amazon.com/ec2/v2/connect/ubuntu/i-0c19e69b86d84bb13

```
import os
from flask import Flask
from flaskext.mysql import MySQL      # For newer versions of flask-mysql
# from flask.ext.mysql import MySQL   # For older versions of flask-mysql
app = Flask(__name__)

mysql = MySQL()
mysql_database_host = 'MYSQL_DATABASE_HOST' in os.environ and os.environ['MYSQL_DATABASE_HOST'] or  'localhost'
# MySQL configurations
app.config['MYSQL_DATABASE_USER'] = 'root'
app.config['MYSQL_DATABASE_PASSWORD'] = 'Root123$'
app.config['MYSQL_DATABASE_DB'] = 'employee_db'
app.config['MYSQL_DATABASE_HOST'] = 'projectdatabase.cjhfdnm1t3ax.us-east-1.rds.amazonaws.co'
mysql.init_app(app)

conn = mysql.connect()

cursor = conn.cursor()

@app.route("/")
def main():
    return "Welcome!"

@app.route('/how are you')
def hello():
    return 'I am good, how about you?'
@app.route('/read from database')
def read():
    cursor.execute("SELECT * FROM employees")
    row = cursor.fetchone()
    result = []
    while row is not None:
        result.append(row[0])
"app.py" 36L, 1047C                                          13,93         Top
```

i-0c19e69b86d84bb13 (Project-ubuntu-1)

Public IPs: 54.164.155.94 Private IPs: 172.31.30.202

c. Run the python flask application , with the command – *FLASK_APP=app.py flask run --host=0.0.0.0*

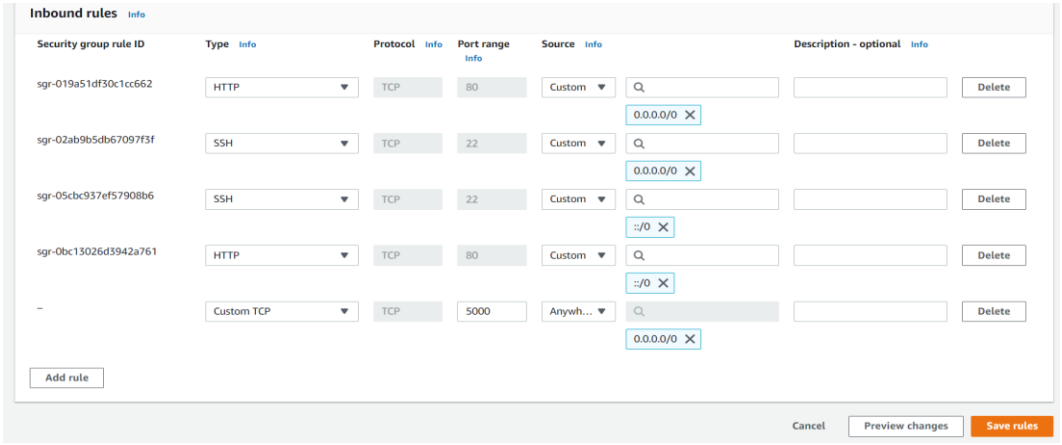
console.aws.amazon.com/ec2/v2/connect/ubuntu/i-0c19e69b86d84bb13

```
ubuntu@ip-172-31-30-202:~$ FLASK_APP=app.py flask run --host=0.0.0.0
 * Serving Flask app "app.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

i-0c19e69b86d84bb13 (Project-ubuntu-1)

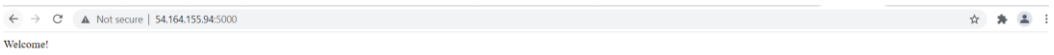
Public IPs: 54.164.155.94 Private IPs: 172.31.30.202

d. As the python application is running in port 5000, add the inbound rule in the security group of the instance.

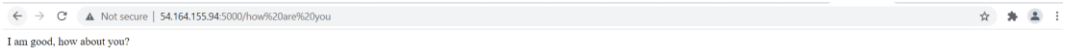


e. Open a browser and enter the URL

http://<IP address of your EC2 Instance>:5000 -> Welcome

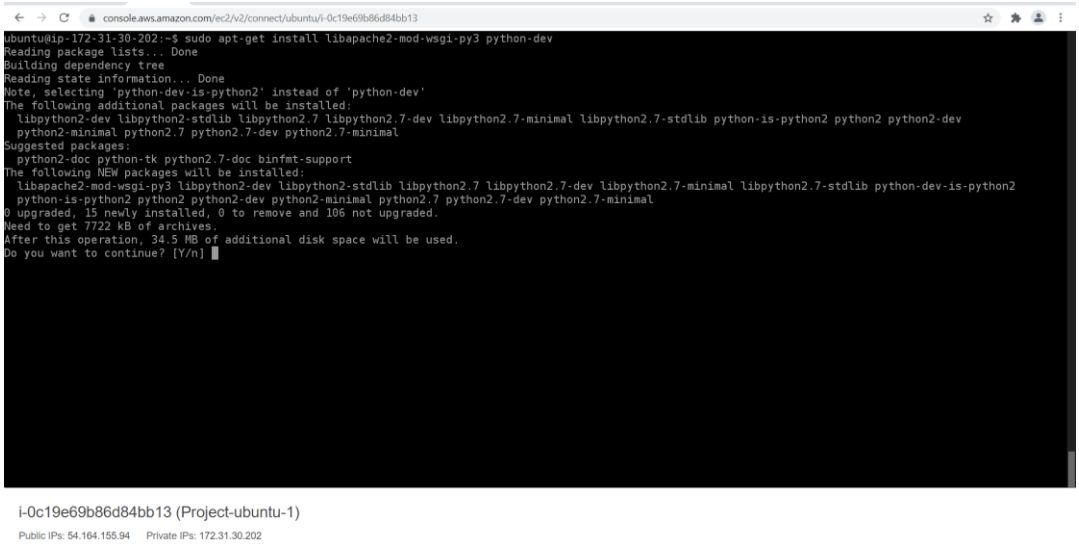


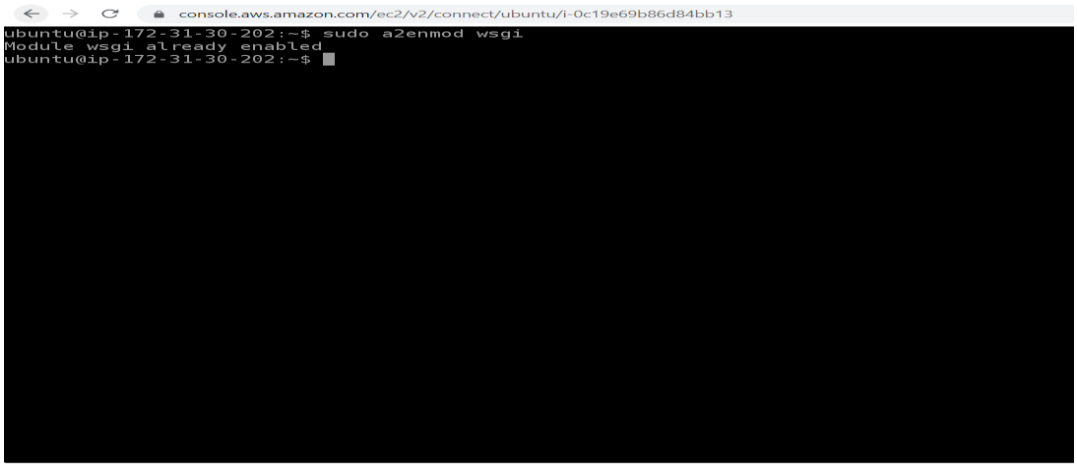
http://<IP address of your EC2 Instance>:5000/how%20are%20you -> I am good, how about you?



http://<IP address of your EC2 Instance>:5000/read%20from%20database-><List of employee names>

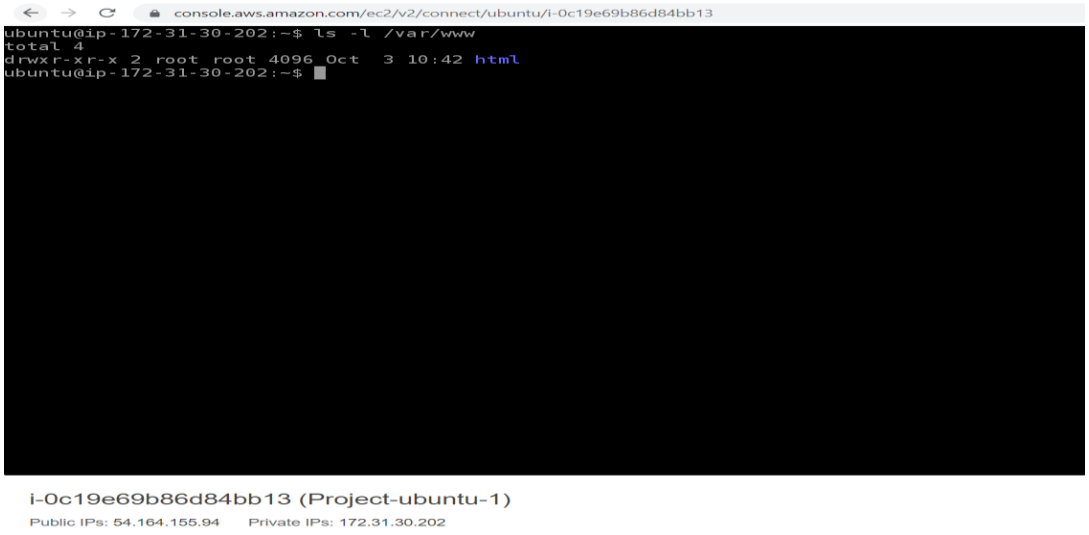
- Change Port of Flask application from 5000 to 80.
    1. Install and enable mod_wsgi(Web server Gateway Interface module)
       $ sudo apt-get install libapache2-mod-wsgi-py3 python-dev
       $ sudo a2enmod wsgi





List the contents of the default web pages directory.

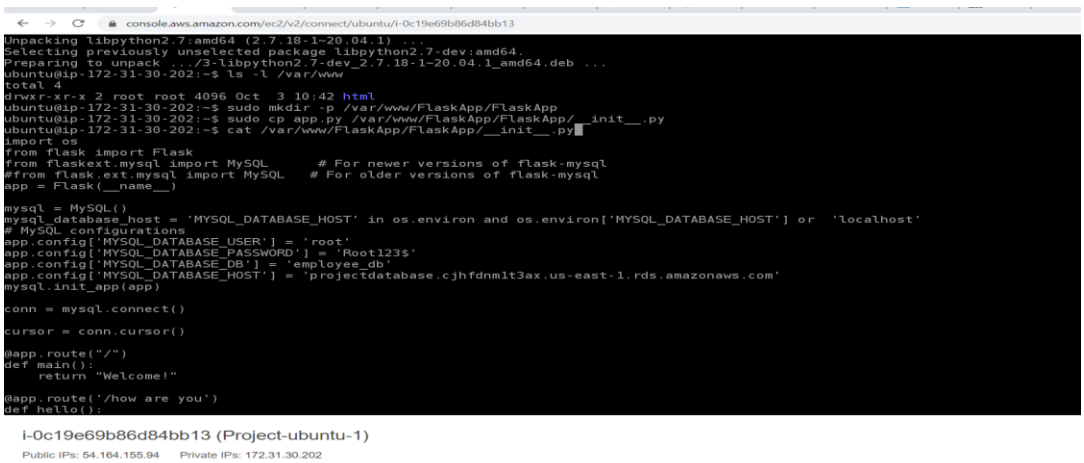
$ ls -l /var/www



2. Create the required sub-directories in the above and copy the existing flask application into the subdirectory.

   $ sudo mkdir -p /var/www/FlaskApp/FlaskApp

   $ sudo cp app.py /var/www/FlaskApp/FlaskApp/__init__.py



3. Configure and Enable a New Virtual Host at port number 80.
    a) Create the configuration file for the FlaskApp.

    $ sudo nano /etc/apache2/sites-available/FlaskApp.conf

    And copy paste the following lines:

    Copy paste the follwing lines.

    -->

    *<VirtualHost *:80>*

    # Add Public DNS name or Public IP address of your EC2 Ubuntu Instance

    ServerName ec2-54-164-155-94.compute-1.amazonaws.com

    ServerAdmin anyEMailId@example.com

    # Give an alias to to start your website url with

    WSGIScriptAlias / /var/www/FlaskApp/flaskapp.wsgi

    <Directory /var/www/FlaskApp/FlaskApp/>

    Order allow,deny

    Allow from all

    </Directory>

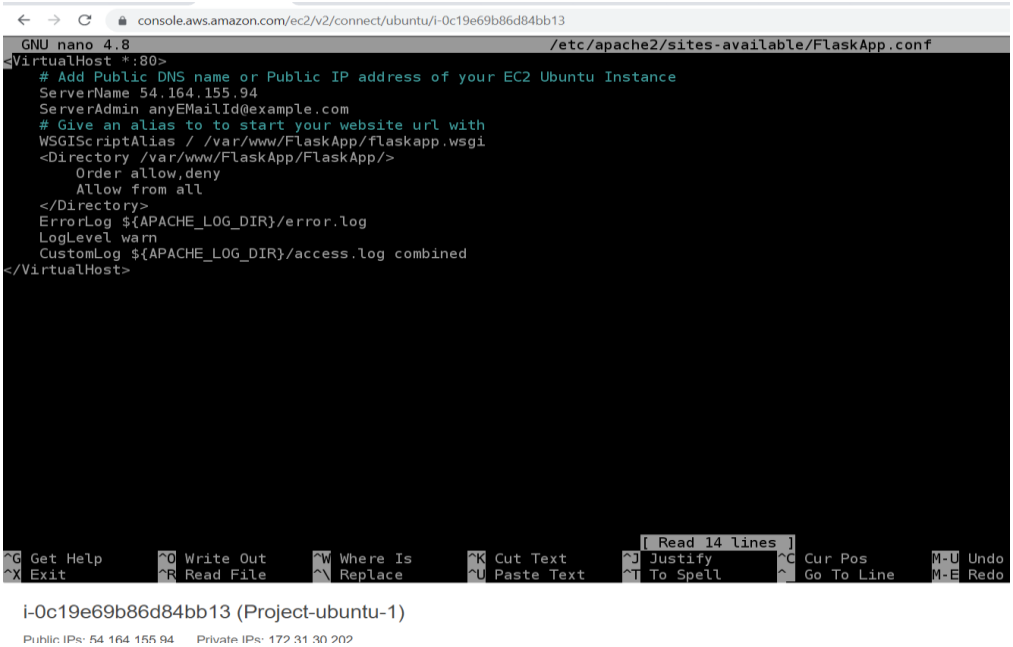
    ErrorLog ${APACHE_LOG_DIR}/error.log

    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/access.log combined

    </VirtualHost>

    <--



    b) Enable the virtual host with the command below.

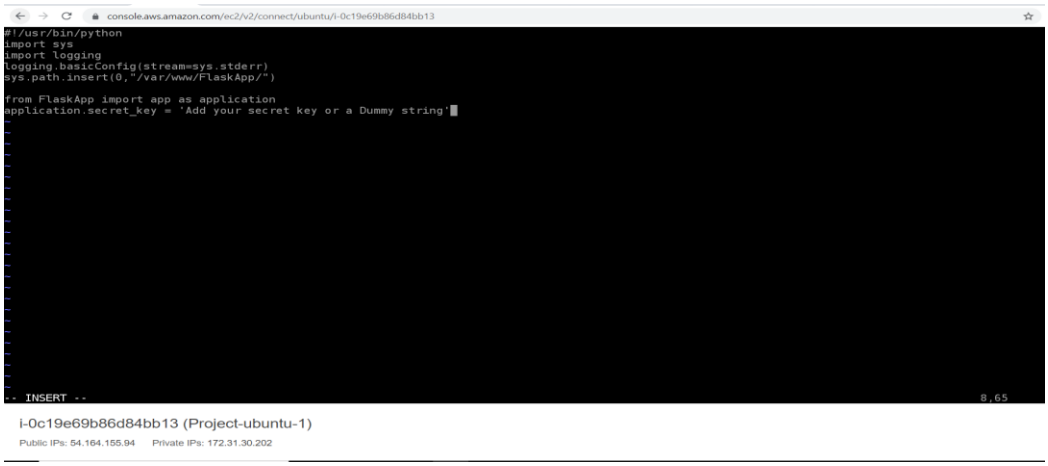
    $ sudo a2ensite FlaskApp

    $sudo systemctl reload apache2

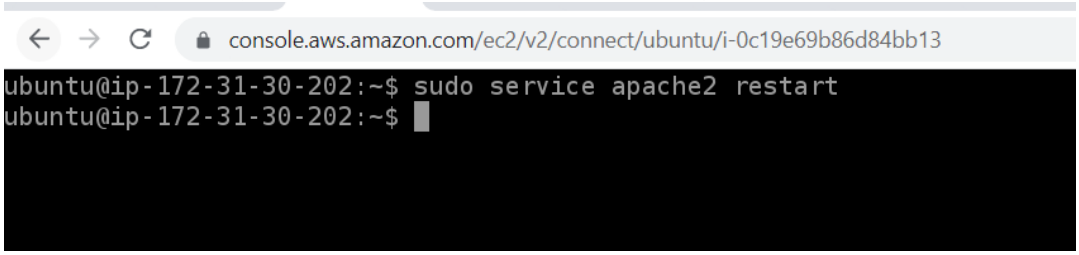
4. Create the .wsgi File
   Run the following command and create a configuration file flaskapp.wsgi in the FlaskApp directory
   $ sudo nano /var/www/FlaskApp/flaskapp.wsgi

   -->
   #!/usr/bin/python
   import sys
   import logging
   logging.basicConfig(stream=sys.stderr)
   sys.path.insert(0,"/var/www/FlaskApp/")

   from FlaskApp import app as application
   application.secret_key = 'Add your secret key or a Dummy string'
   <--



5. Finally restart apache2 server*($ sudo service apache2 restart)*



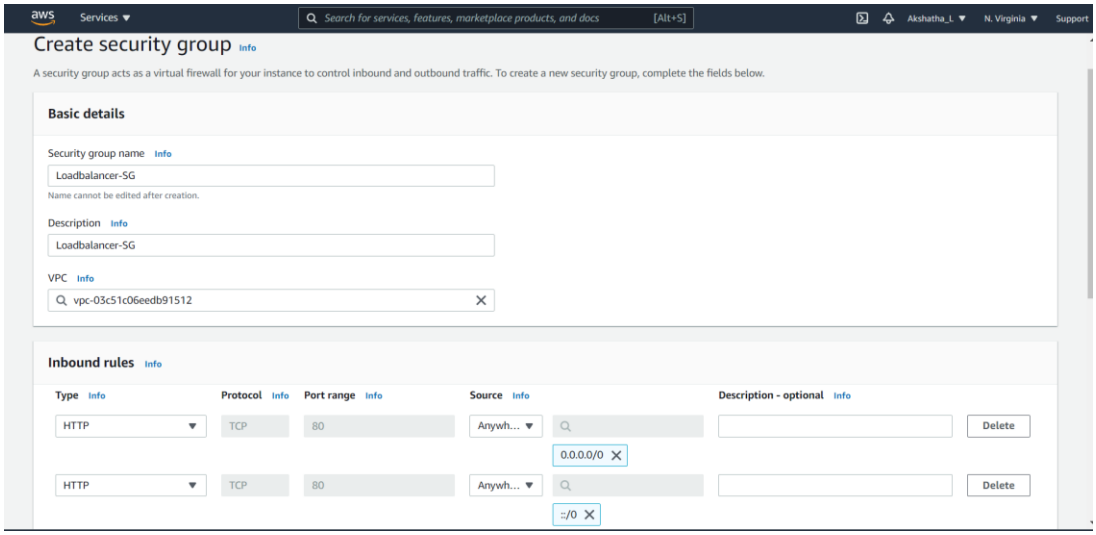
6. Now open a browser on your laptop and in the address bar give the Public IP address or Public DNS name of your EC2 instance.
http://<IP address of your EC2 Instance>/ => "Welcome"

http://<IP address of your EC2 Instance>/how%20are%20you
=> I am good, how about you?

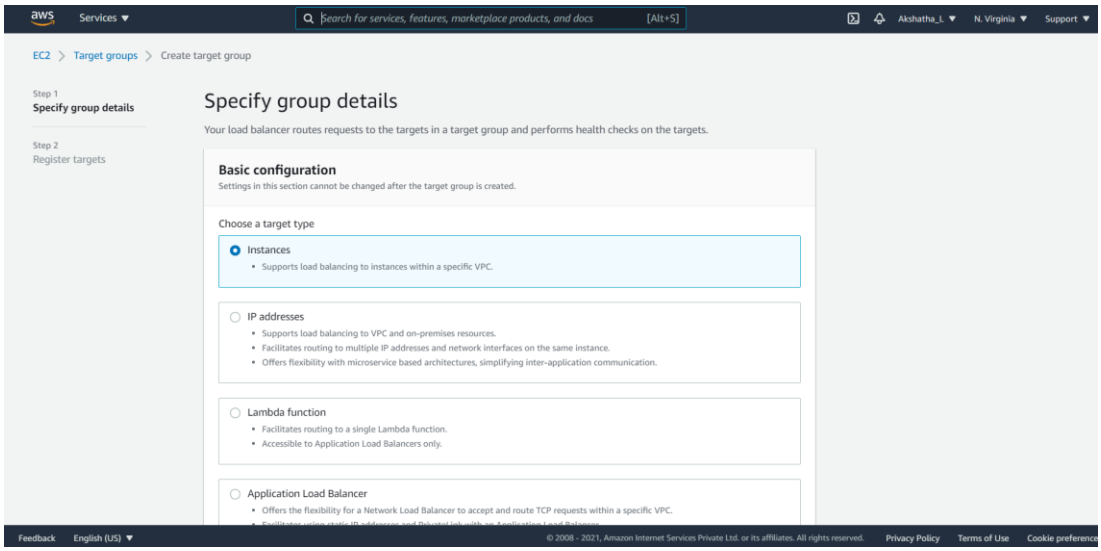
http://<IP address of your EC2 instance/read%20from%20database => <list of employee names>

This indicates that you have installed Flask and configured a sample test Flask application to use port number 80 properly.

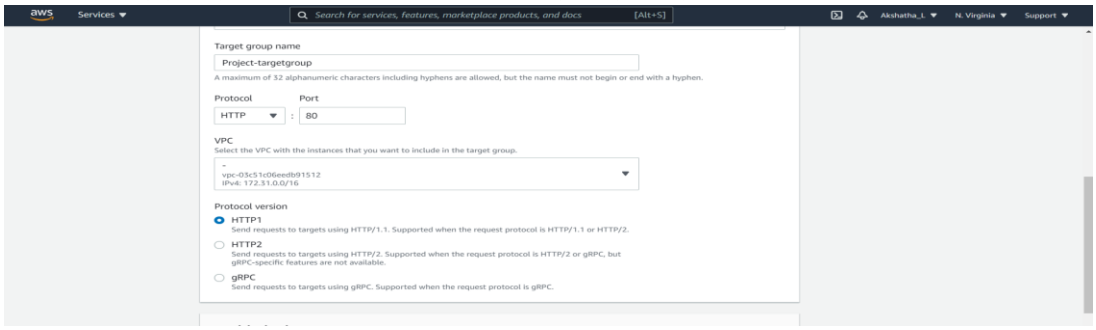
- Set up auto-scaling group with application load balancer for the above application configuration.
    1. Create Security group for load balancer.
        a. Navigate to the “EC2 service” in AWS Dashboard and go to “Security groups” option to create a security group.



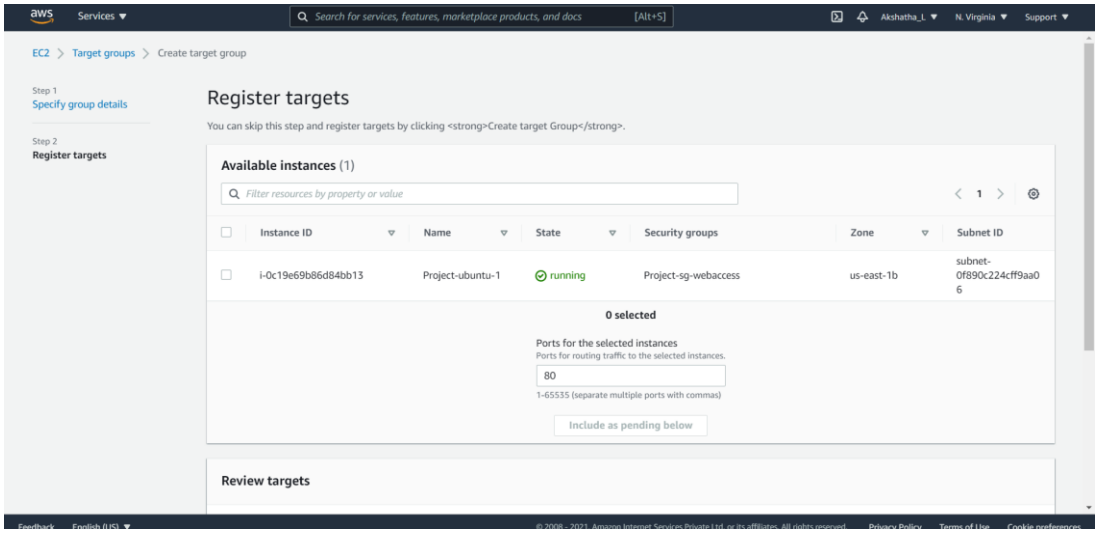
2. Create target group for load balancer.
   a. Move to “Target groups” and click on “Create target group”. Select target type as Instance.



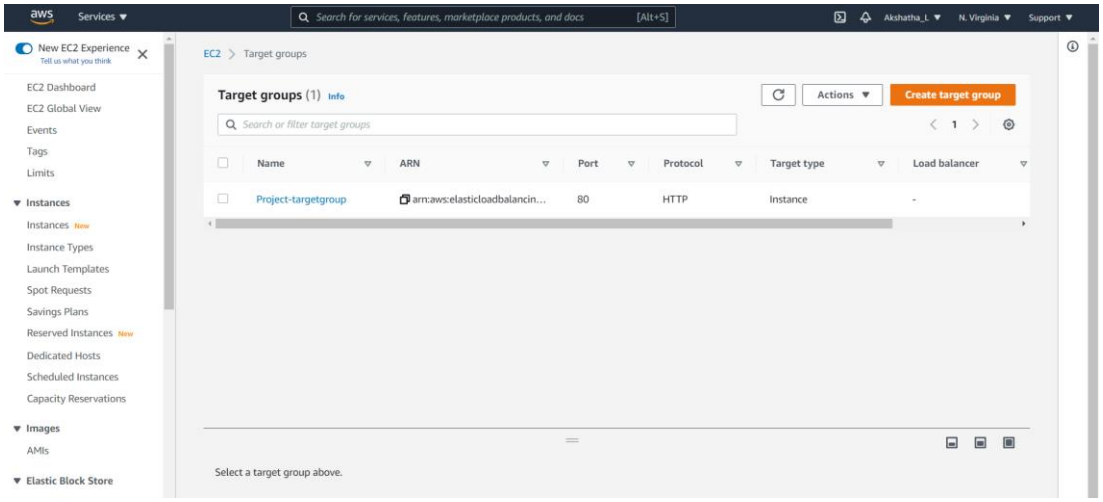
   b. Give an appropriate name to the target group.



   c. Do not register any instance with the target group now.

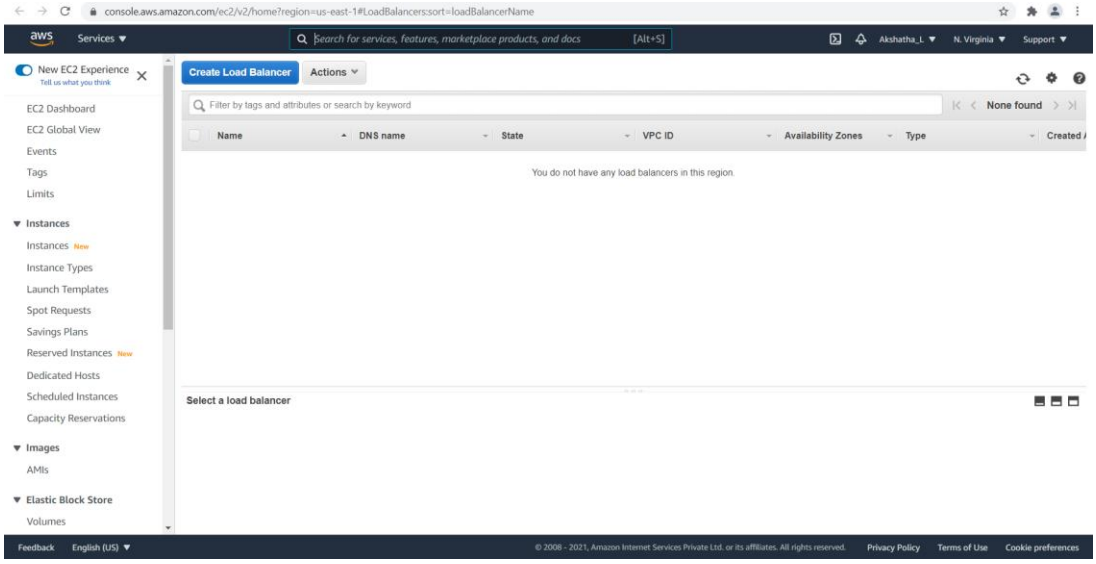


   d. Target group is created successfully.

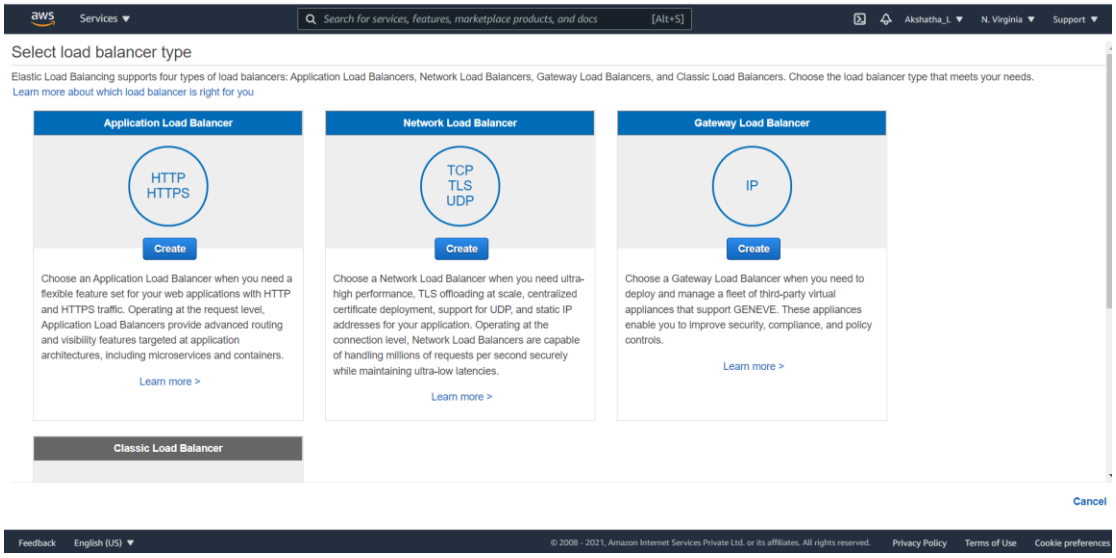


3. Create application load balancer.

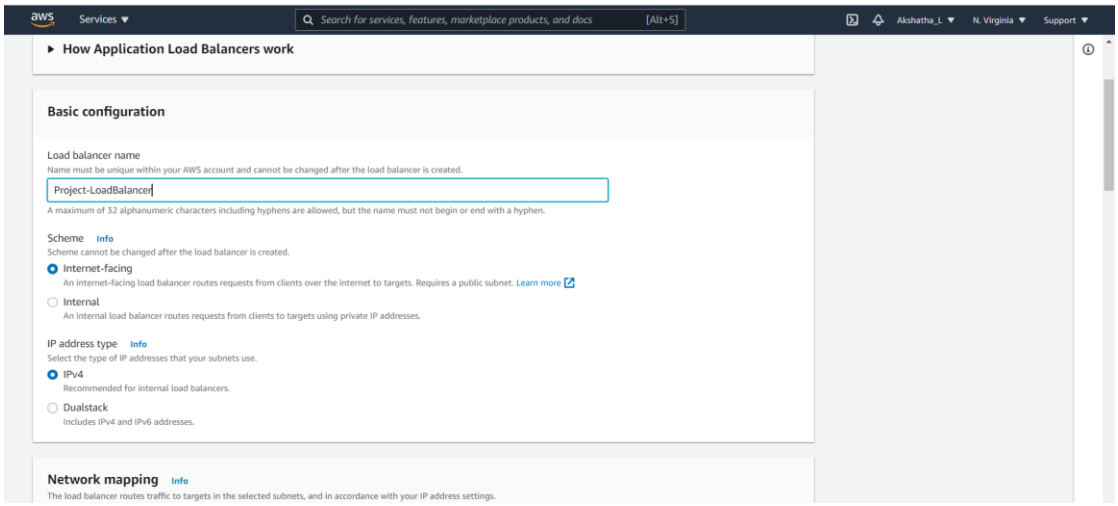
a. Go to “Load balancer” page in AWS Dashboard and click on “Create Load balancer”.



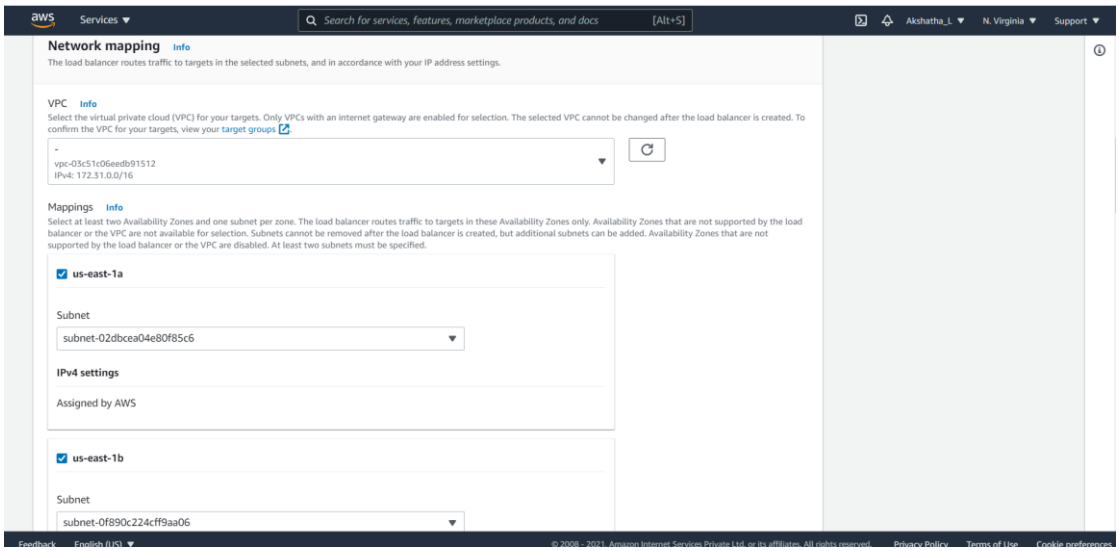
b. Select Load balancer type as “Application Load balancer.



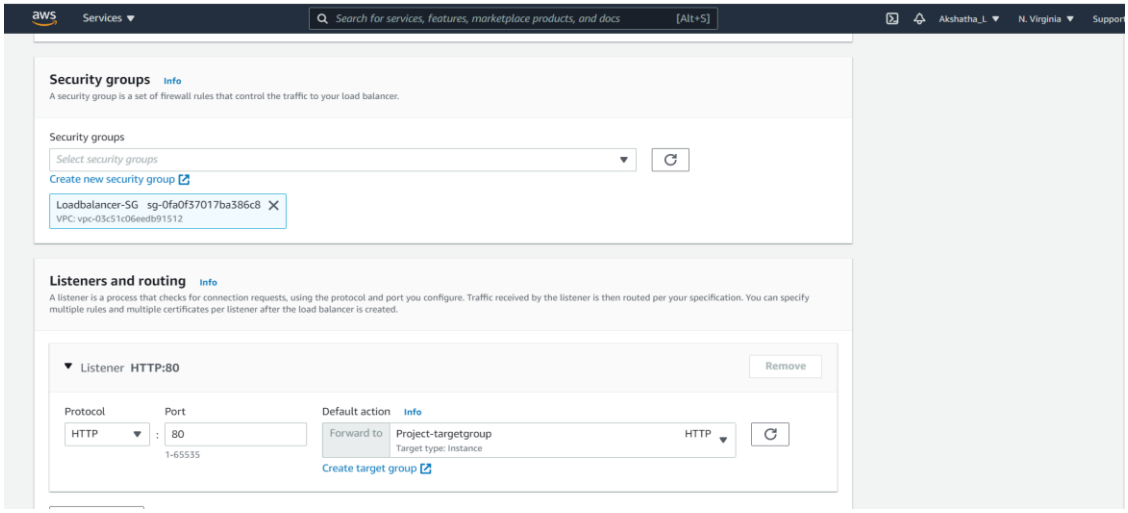
c. Let the load balancer be internet facing , and give a name to the load balancer.



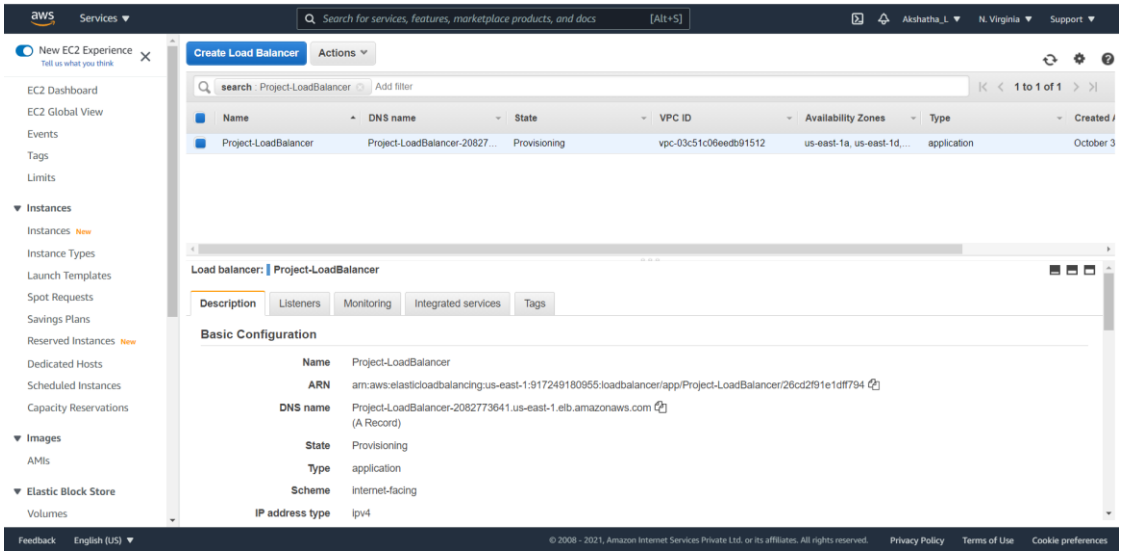
d. Place the Load balancer in default VPC and select all available subnets in the VPC.



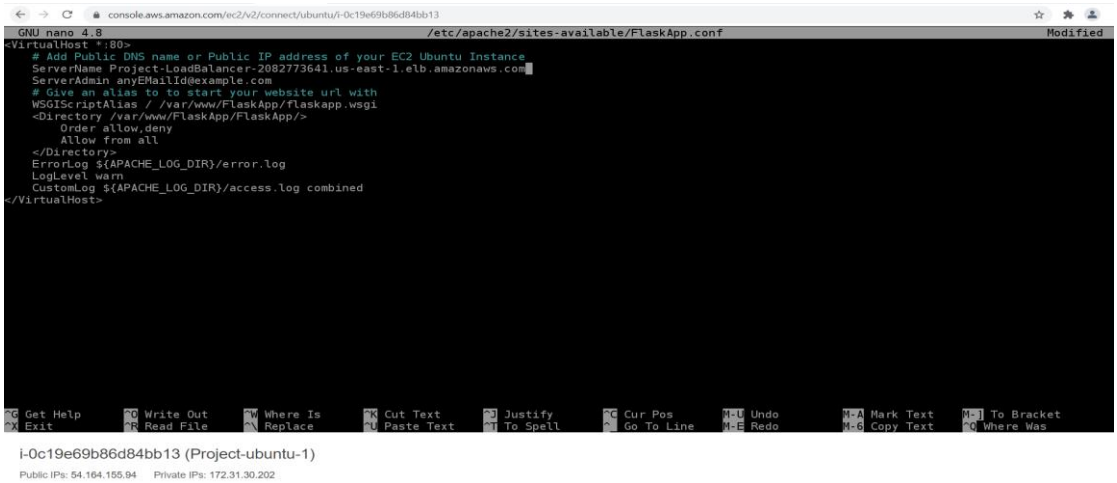
e. Select the security group and target group that were previously created and attach it to the load balancer.



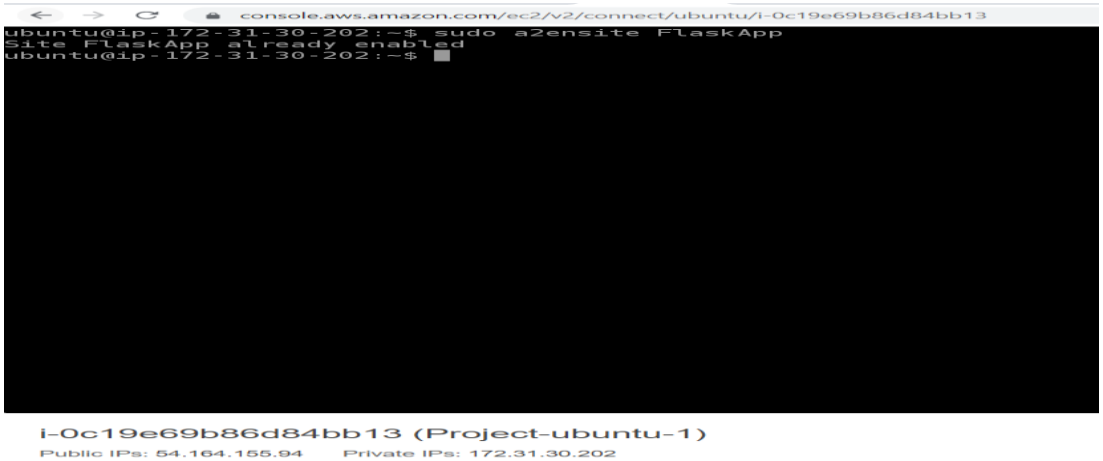
f. The load balancer is created successfully.



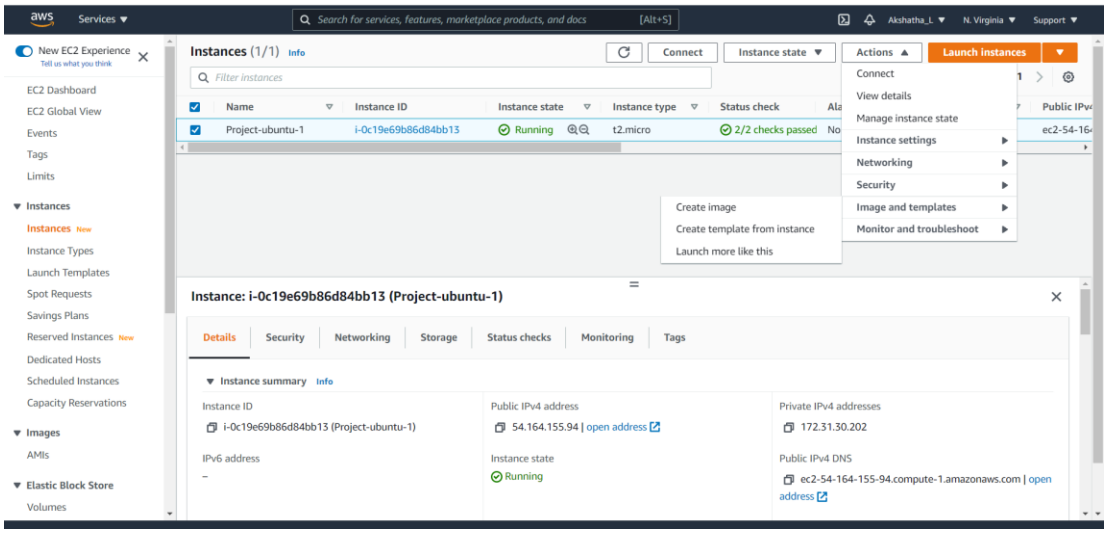
4. Create an AMI from the existing EC2 instance.
   a. SSH into the EC2 instance , and edit the “FlaskApp.conf” . Replace the Ec2 DNS with Load balancer DNS.



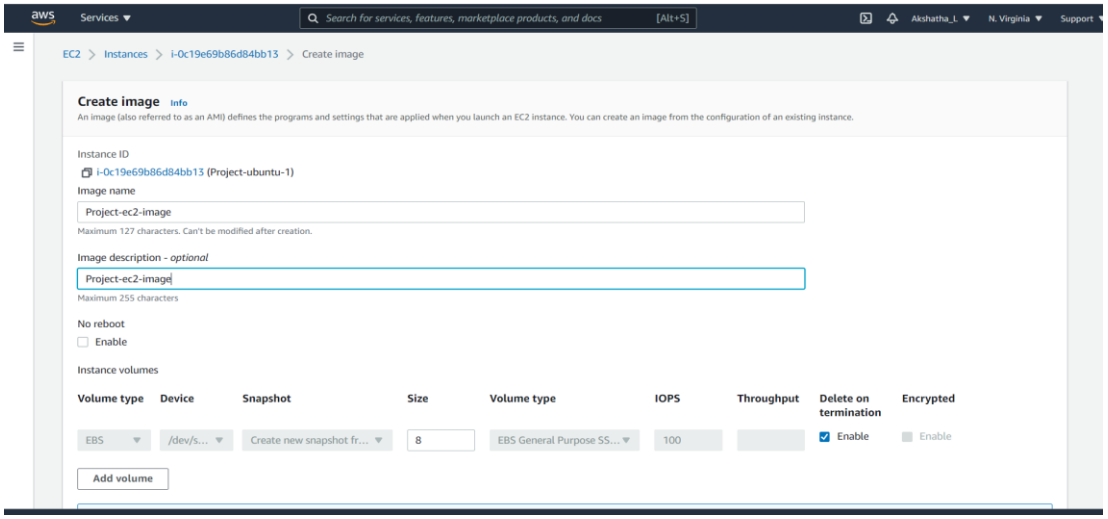
   b. Enable the virtual host with the command below-
   $ sudo a2ensite FlaskApp
   $ sudo service apache2 restart.



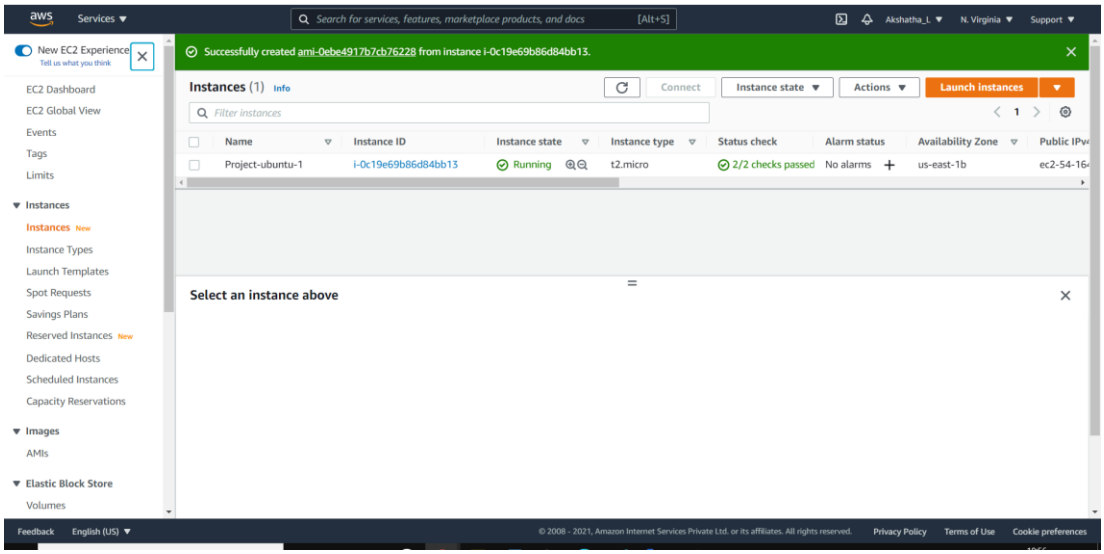
c. Click on the instance in AWS dashboard and , Create an AMI from it.



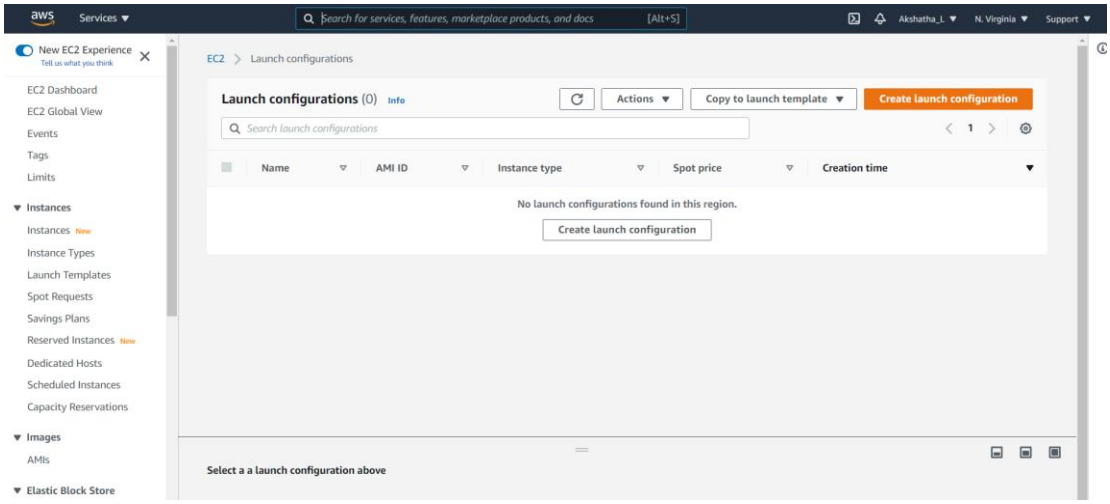
d. Give a name to the image .



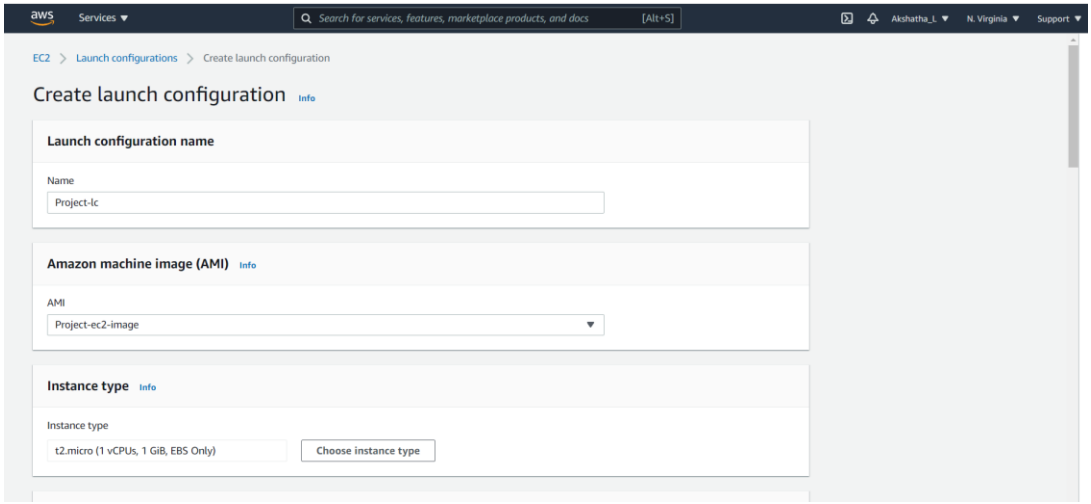
e. The image is created successfully.



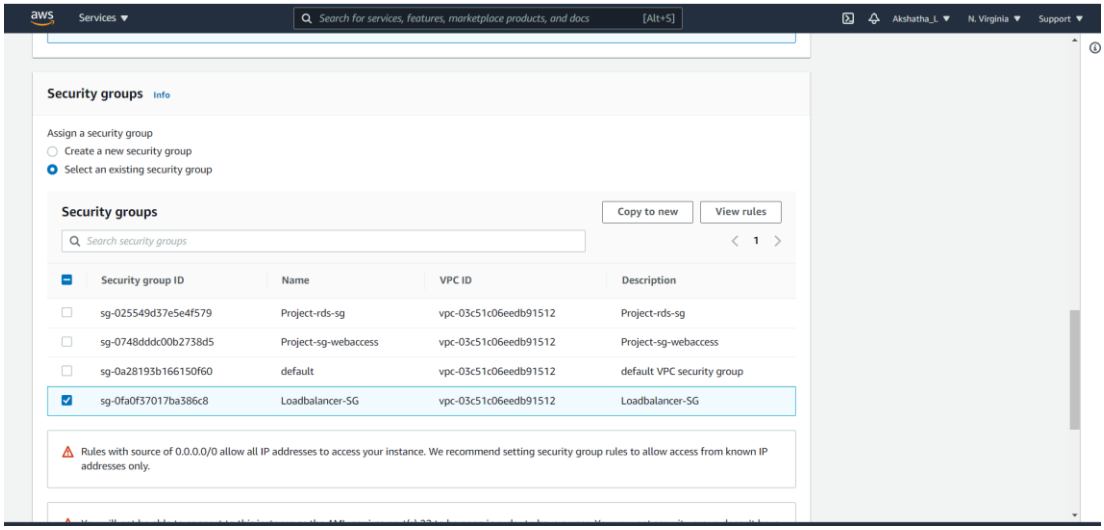
5. Create Launch configuration from the AMI.
   a. On the navigation pane, under AUTO SCALING, choose Launch Configurations. Choose Create launch configuration, and enter a name for the launch configuration such as my-lc-01.



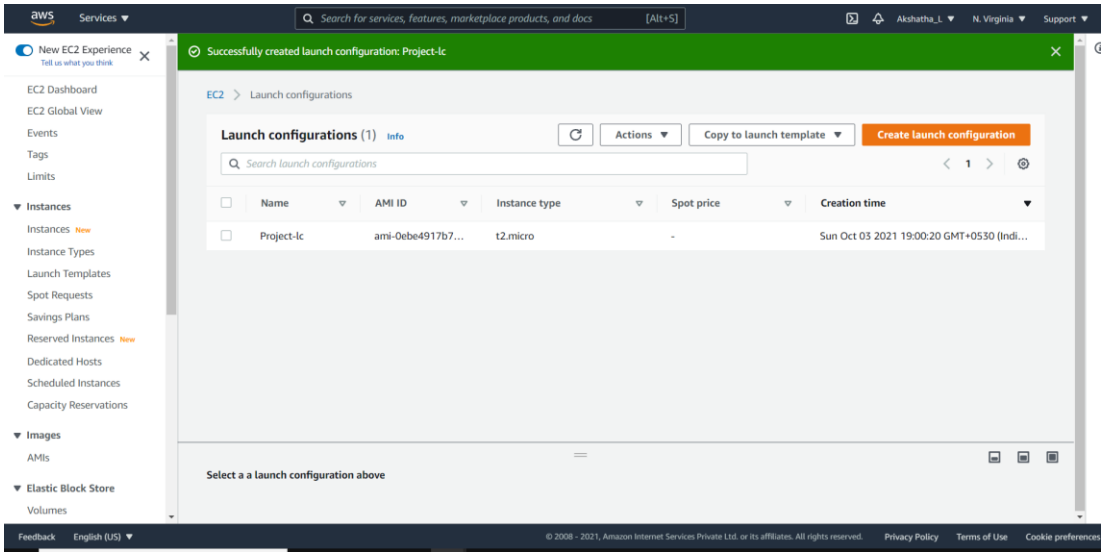
   b. Select the AMI which was created from the Ec2 instance.



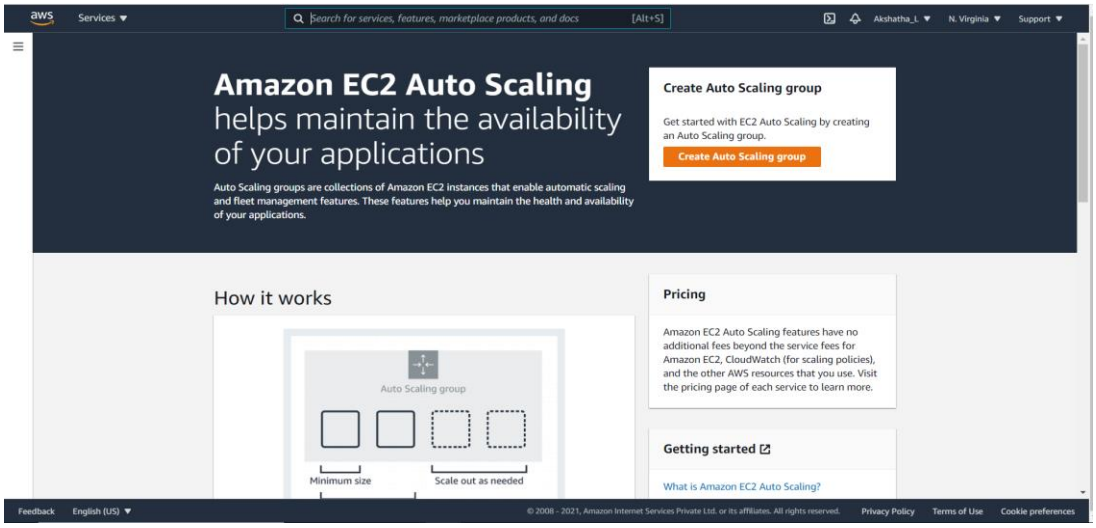
   c. Attach the security group with the launch configuration.



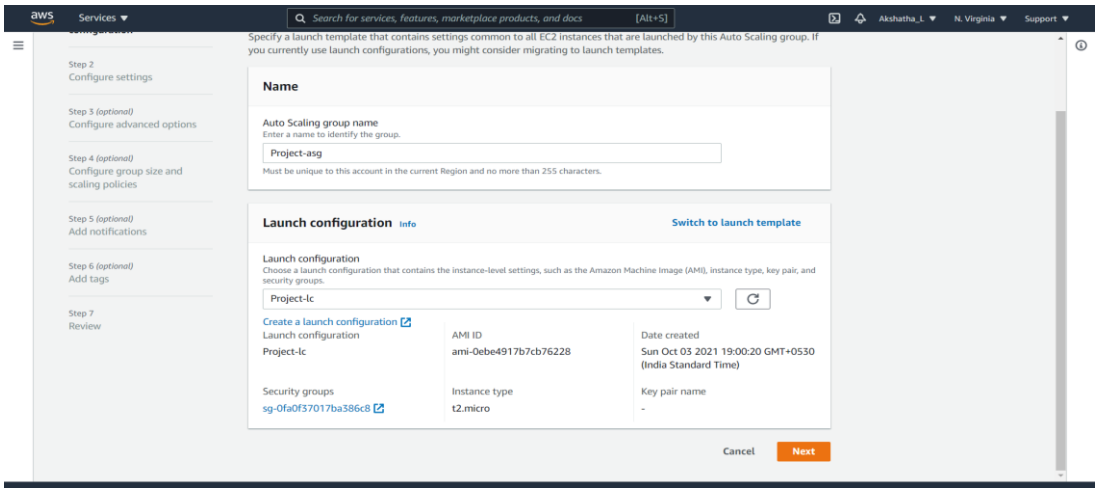
d. The Launch configuration is created successfully.



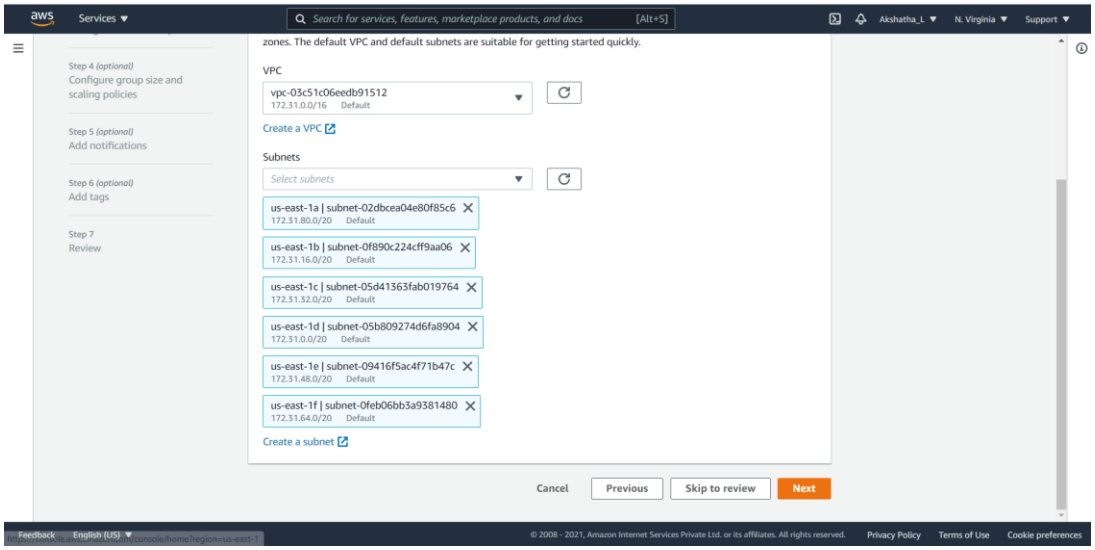
6. Create Autoscaling group.
   a. On EC2 services page - Choose Create an Auto Scaling group on left navigation panel.



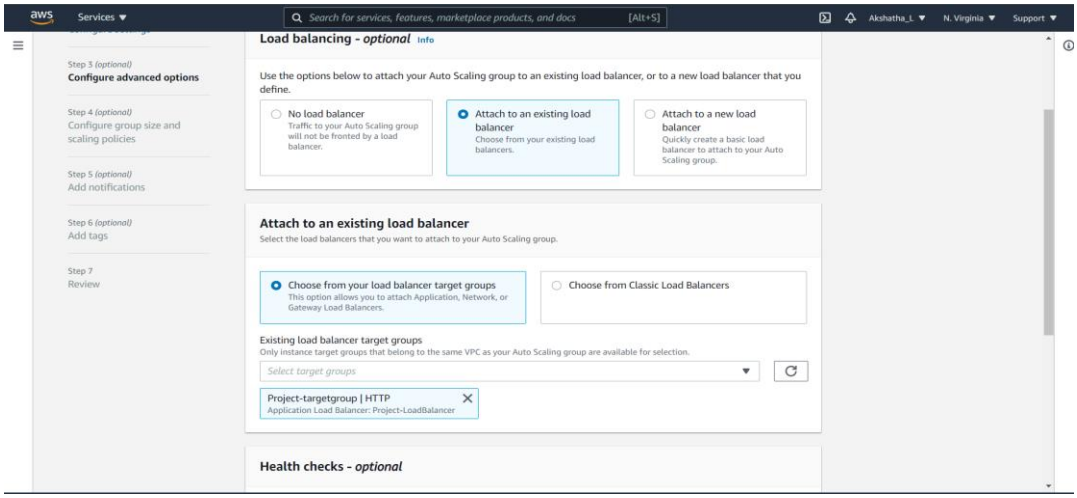
   b. Give a name to the auto scaling group and select the launch configuration that was created in the previous group.



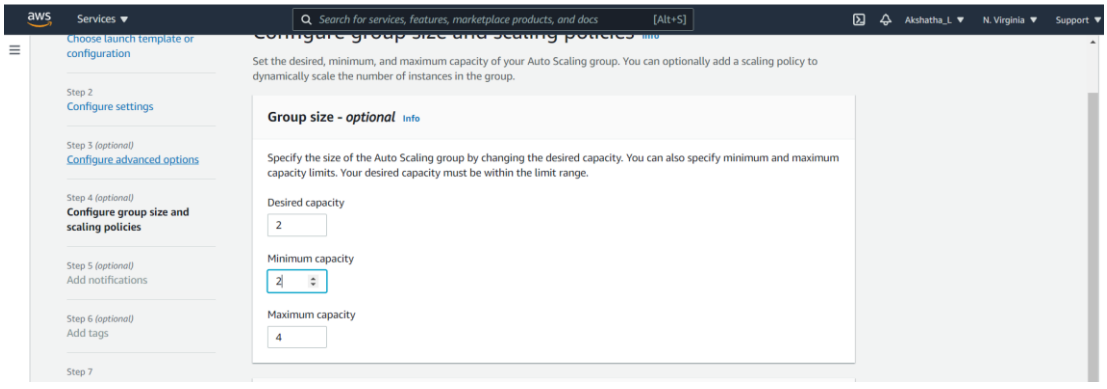
c. Place the auto scaling group in the default VPC and select all the subnets available in the VPC.



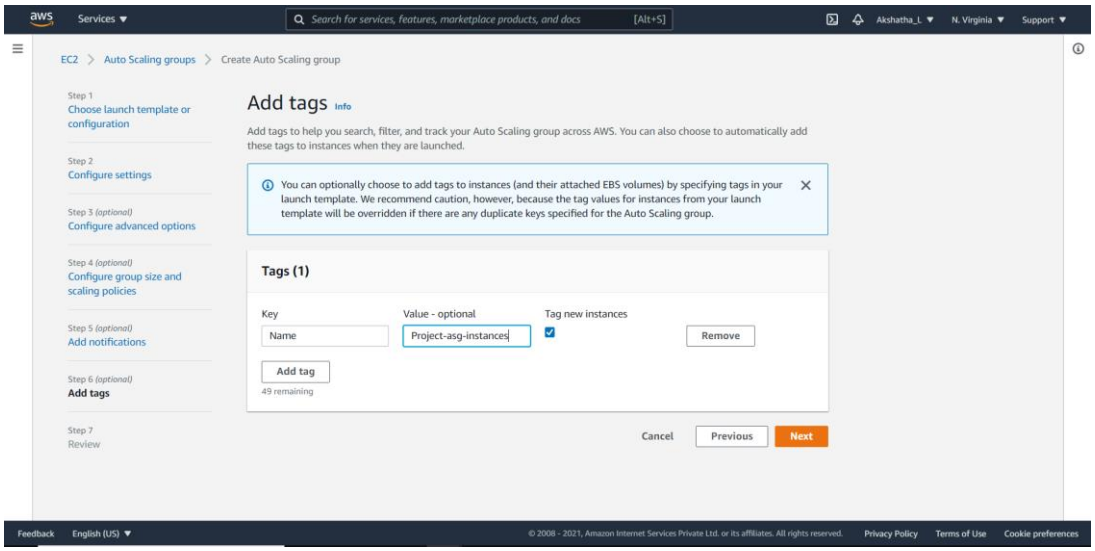
d. Attach the auto scaling group with the load balancer and target group that was created.



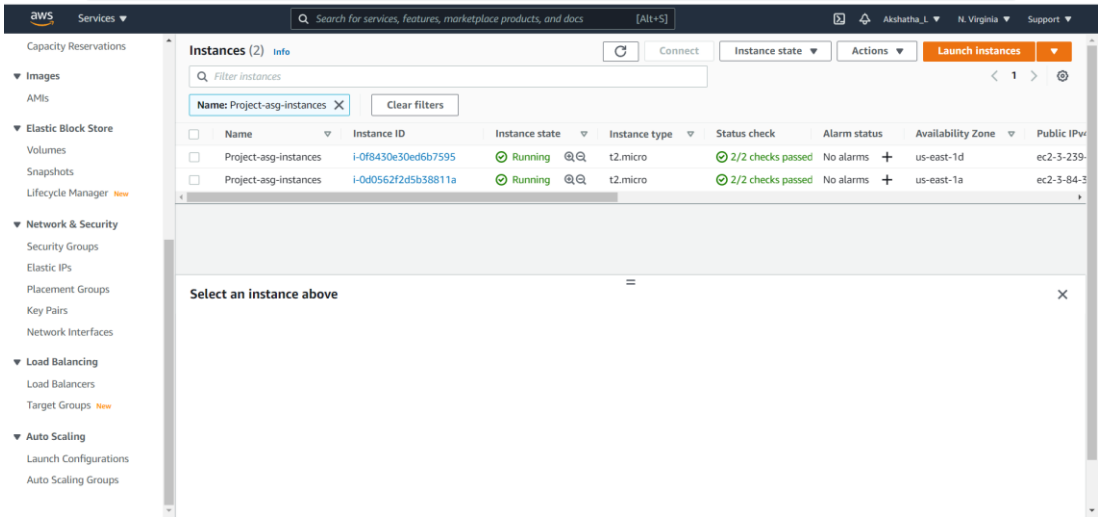
e. Choose the group size to be – Min -2 , max -2, desired -2



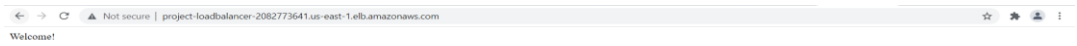
f. Add tags to the auto scaling group and create the group.



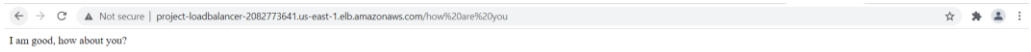
g. After the auto scaling group is created successfully, we can see two instances with the same tag running.



7. Testing.
   a. Open a browser and enter the URL – *http://<DNS name of your loadbalancer>/ => Welcome.*



http://<DNS name of your loadbalancer>/how%20are%20you => I am good, how about you?



http://<DNS name of your loadbalancer>/read%20from%20database => <List of employee names>