edureka! LEARNING CENTER

ELC Cloud and DevOps Program

Application Hosting on AWS

Problem Solution

First Step of solution is to create the VPC for secure design where we are going to launch our database and python application server.

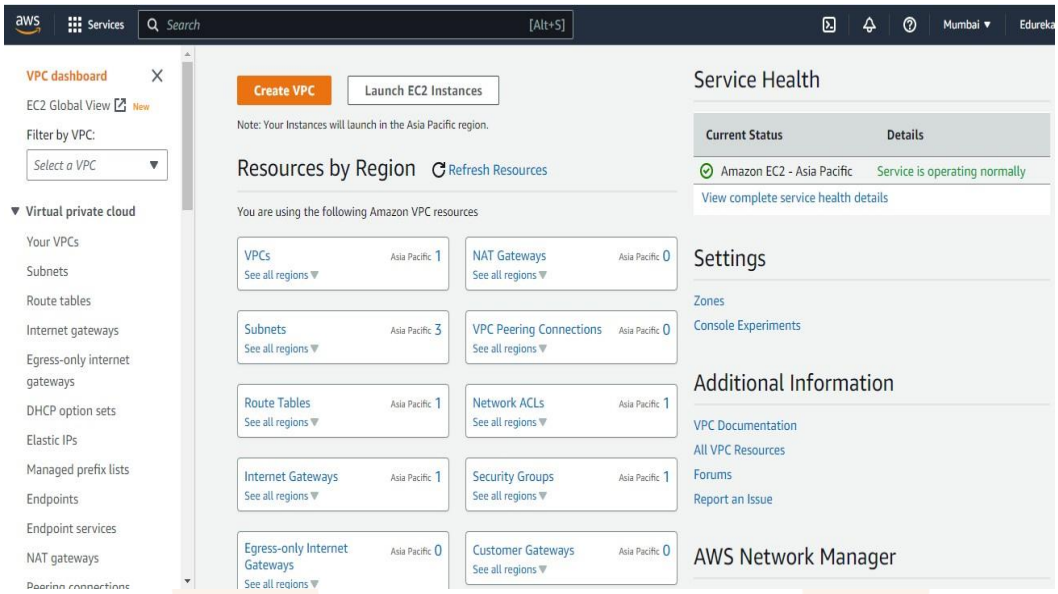

Figure 4.1: VPC dashboard

Create our own VPC provide the name and CIDR value as 0.0.0.0/16 with default tenancy

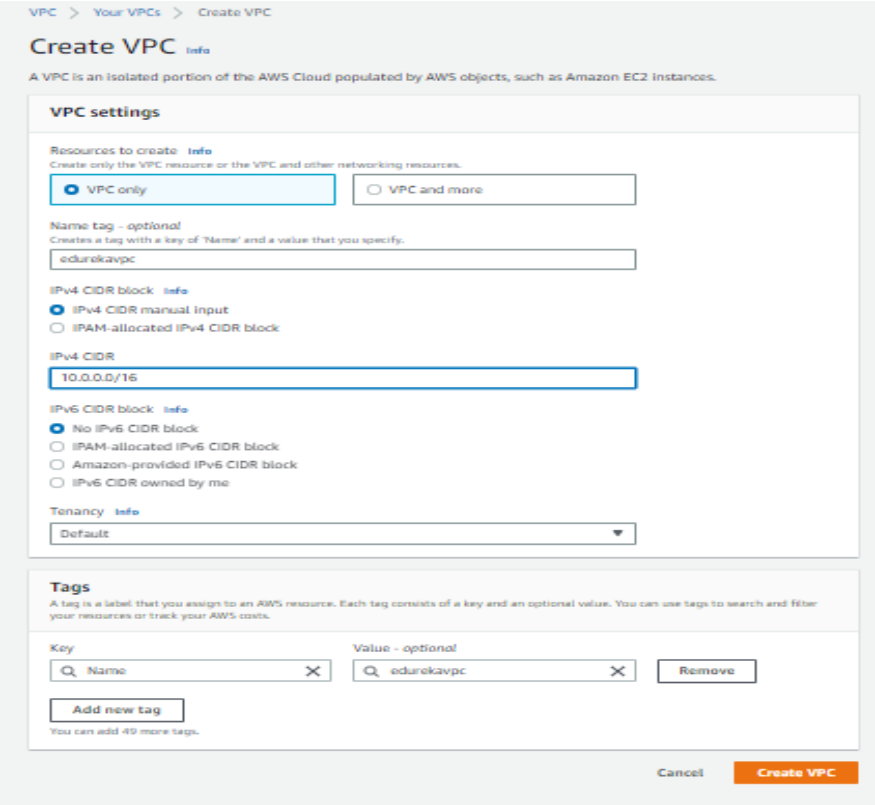

Figure 4.2: Create VPC

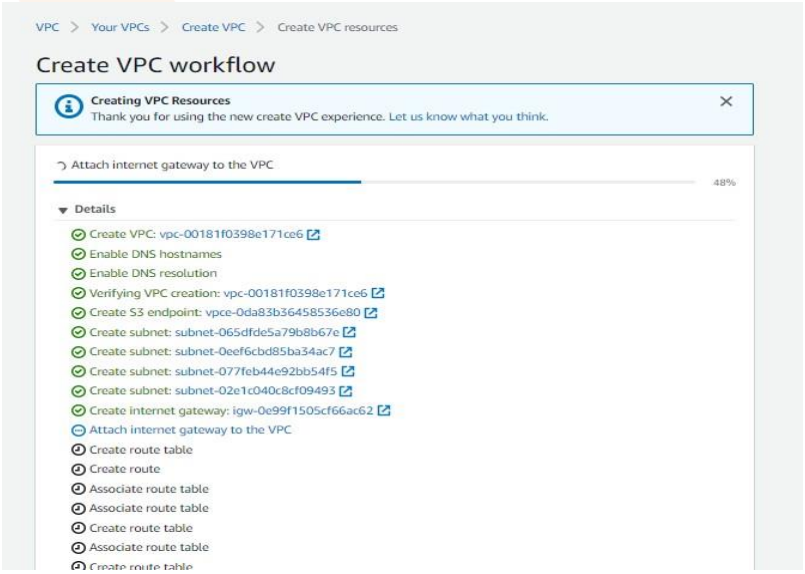

Figure 4.3: VPC creating resources

Next step in creating vpc is to launch the subnet and we need two subnet on private for Database and Public for Backend API.

In the public subnet we have to launch under our own created VPC in with CIDR as 10.0.1.0/24 And Private subnet with CIDR as 10.0.2.0/24

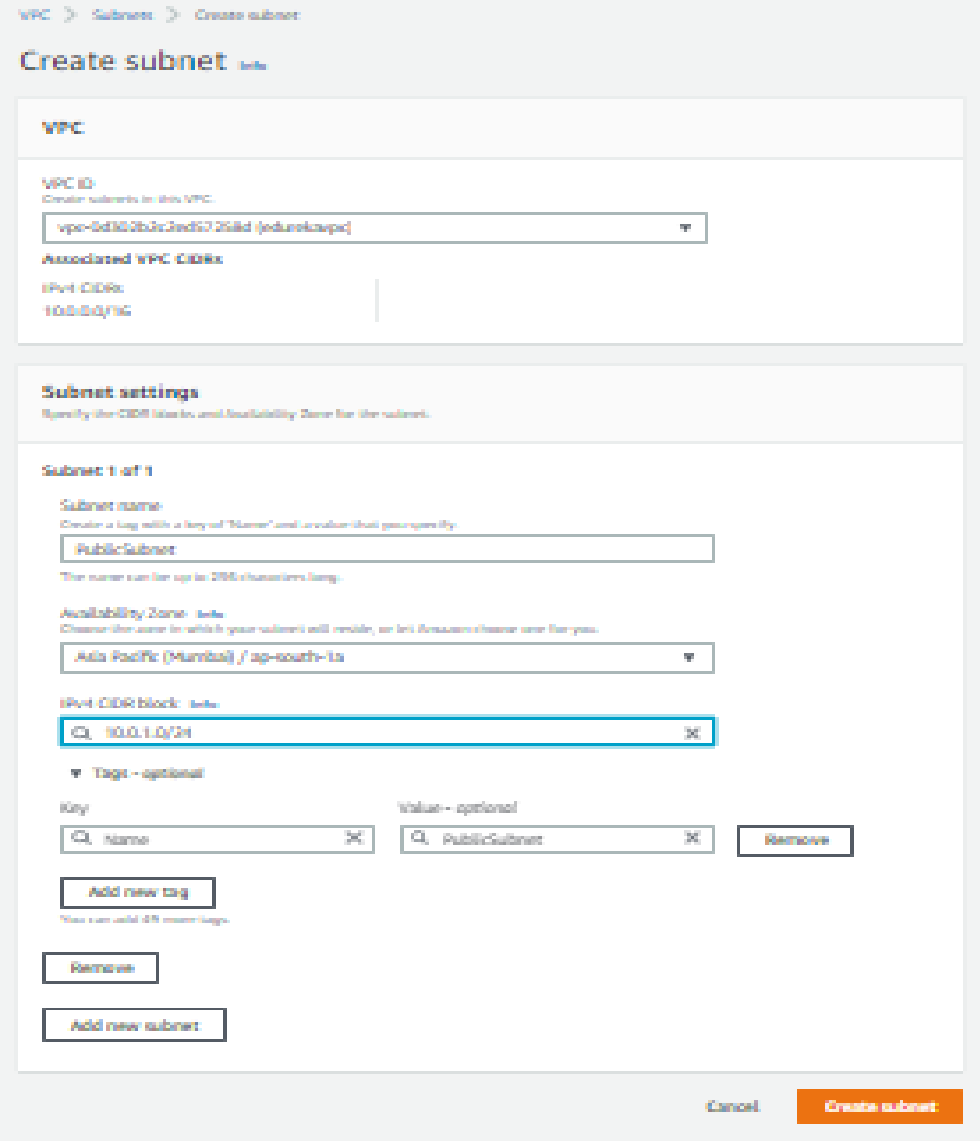

Figure 4.4: Create Public Subnet

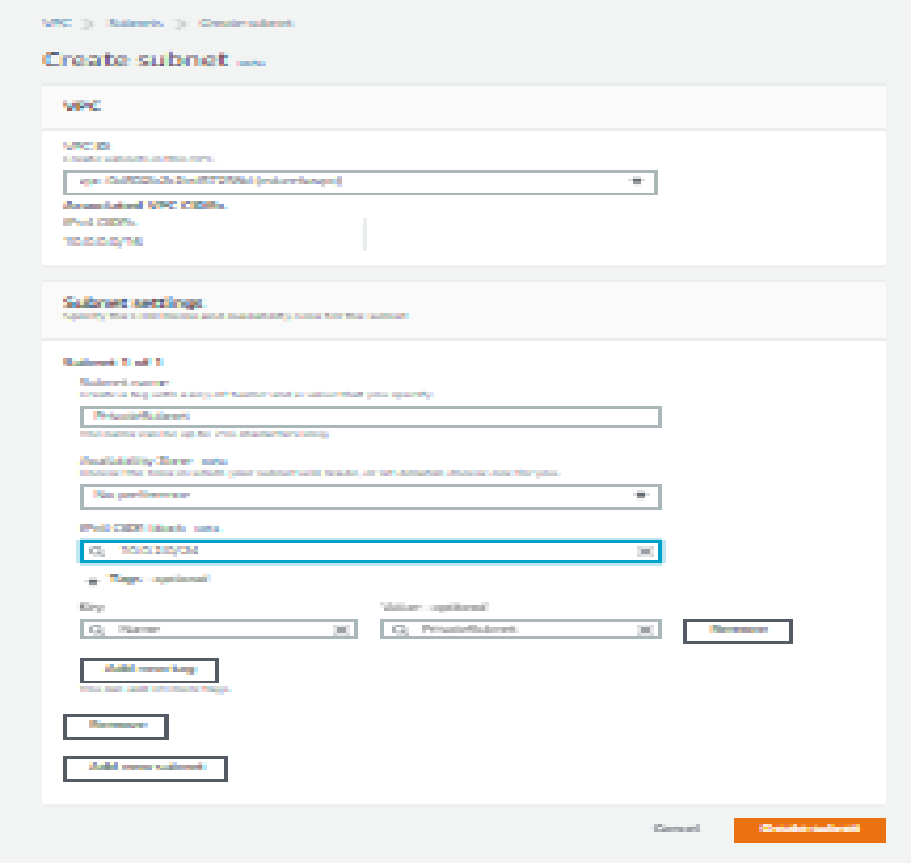

Figure 4.5: Create Private Subnet

Create two route tables for securing our subnet to connect over internet. We have one public route table and one private route table inside our own VPC.

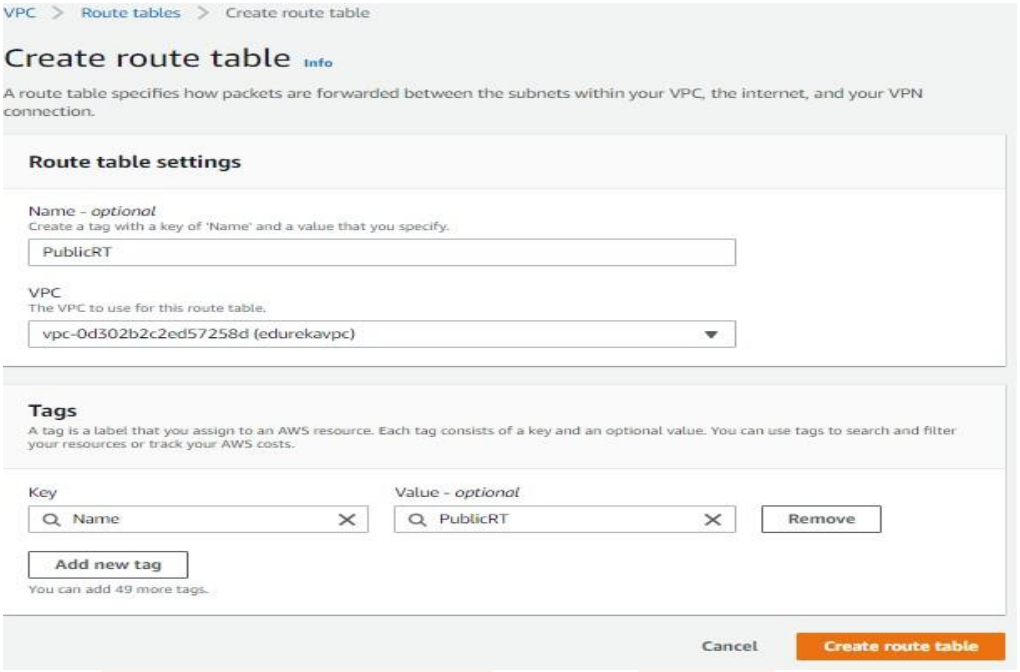

Figure 4.6: Create route table for Public

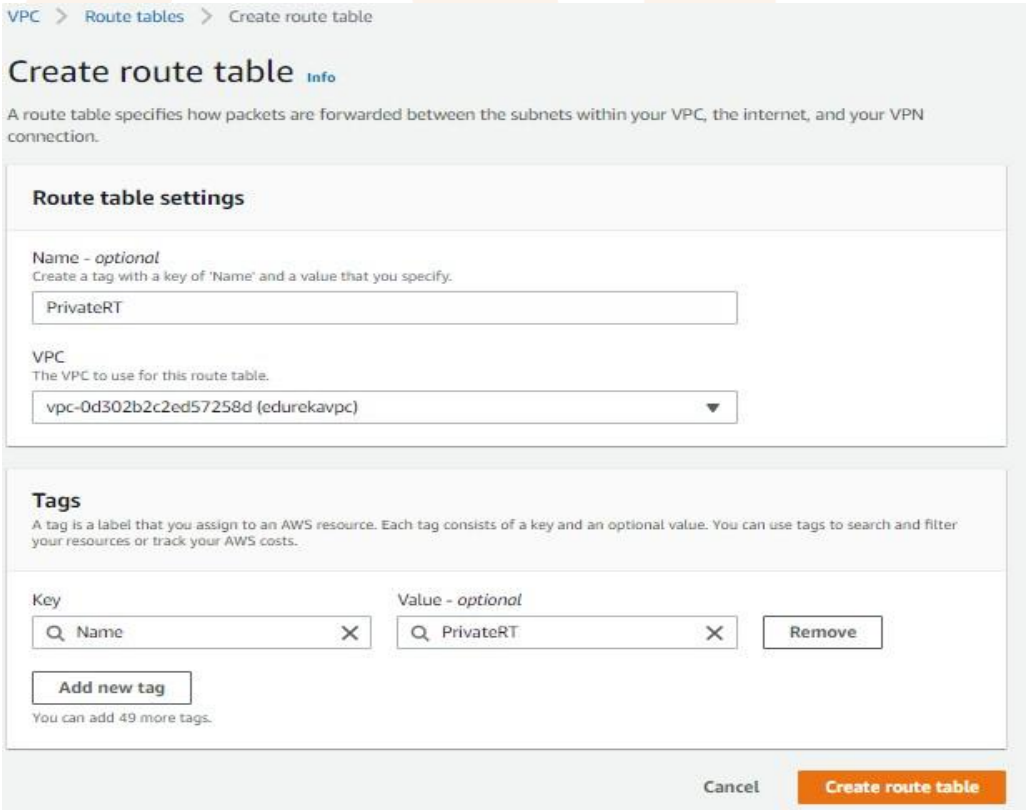

Figure 4.7: Create route table for Private

We have to work on now subnet association with both public route table and private route table.

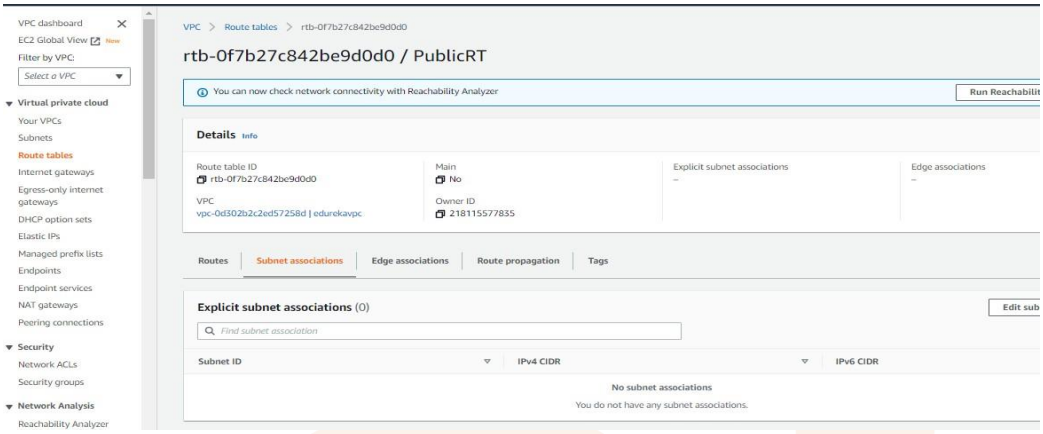

Figure 4.8: Explicit Subnet association

Link public subnet with public route table in the select list we need to select our subnet and click checkbox for association.

Similarly associate private subnet with private route table.

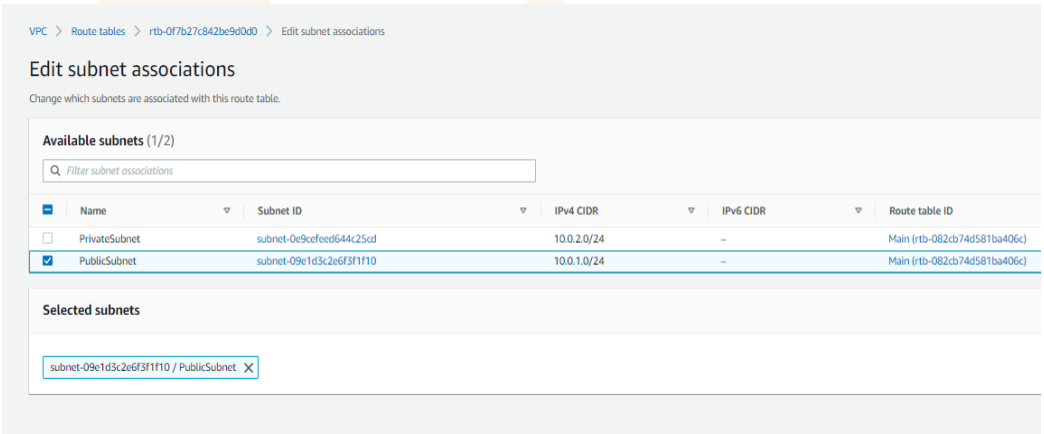

Figure: 4.9: Subnet associations

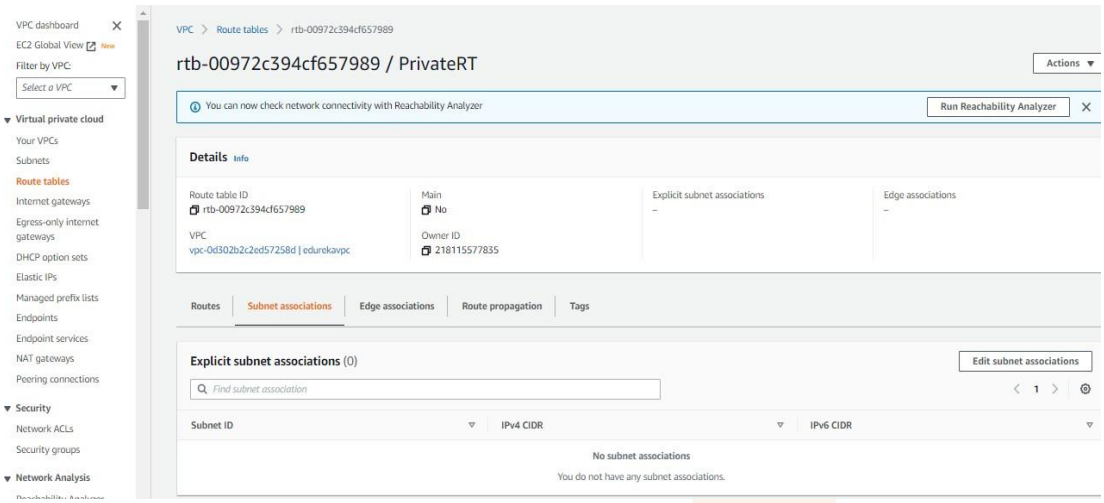

Figure: 4.10: Explicit Subnet associations

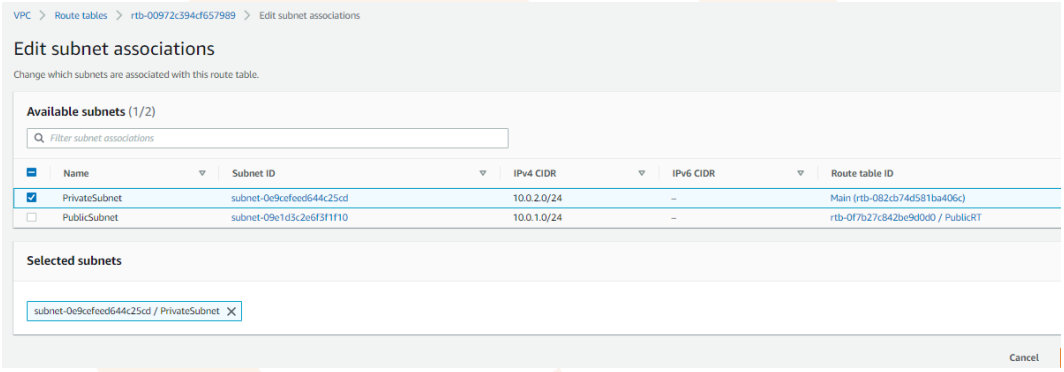

Figure: 4.11: Subnet associations

Now we need to add one internet gateway to the VPC to make it available outside as the public URL.

Select internet gateway and click on action and attach to VPC which you have created.

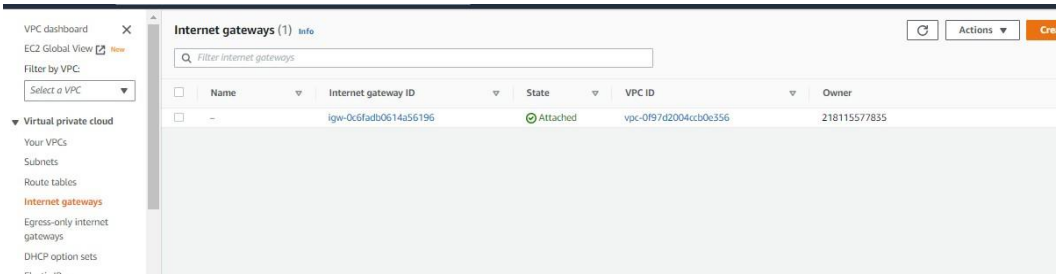

Figure 4.12 Attach Internet Gateway

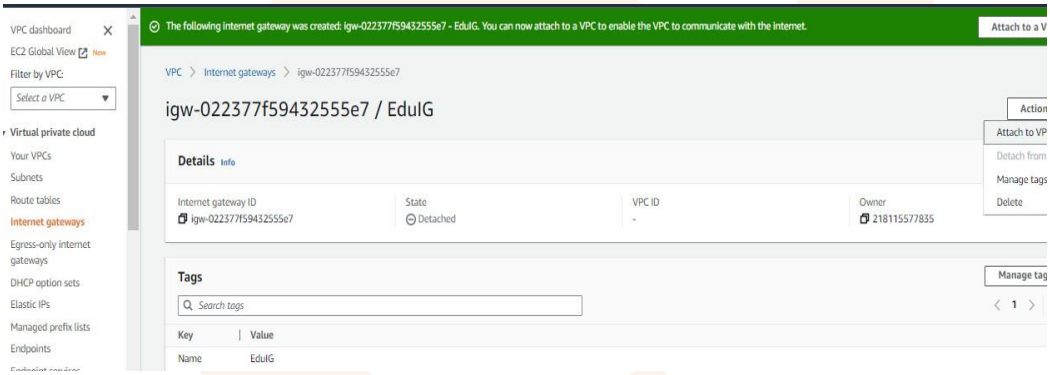

Figure 4.13: Attaching Internet gateway to VPC

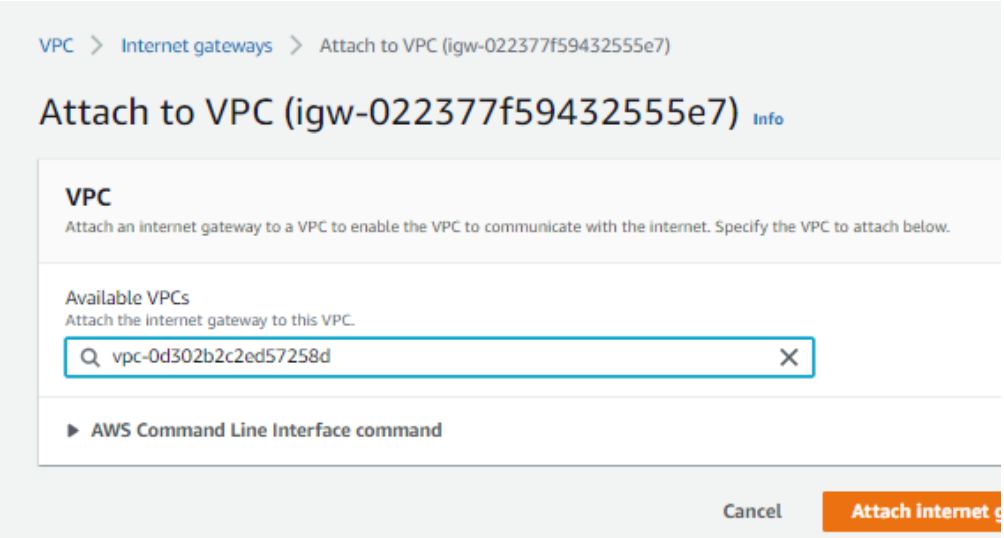

Figure 4.14: Attaching Internet gateway to VPC

By this we are ready for with our VPC configuration with on public and private subnet Now we need one server to deploy the application for our frontend part of application.

So, we are going to launch the elastic beanstalk container where we will deploy our app with continuous deployment mode

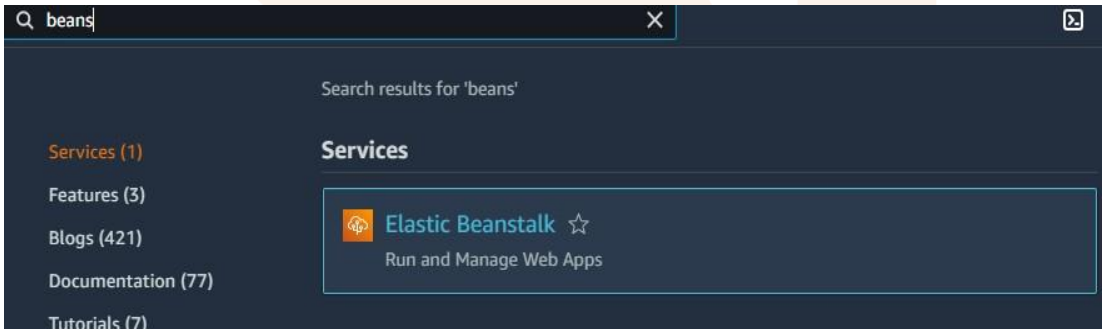

Figure 4.15: Search Elastic Beanstalk

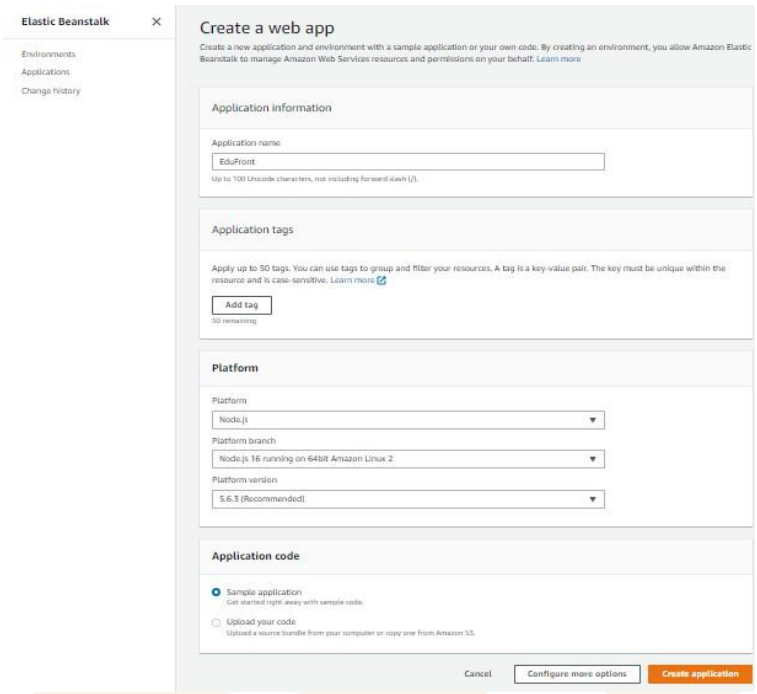

Figure 4.16: Create a Web app in Elastic Beanstalk

This will launch one instance over Ec2 as pass service where we can deploy our frontend

Now create a pipe for continuous integration i.e. CI/CD using Codepipeline

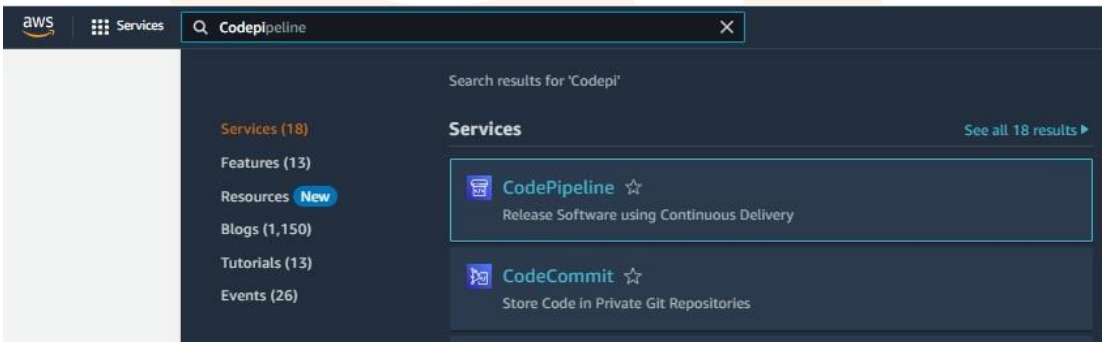

Figure 4.17: Search a CodePipeline

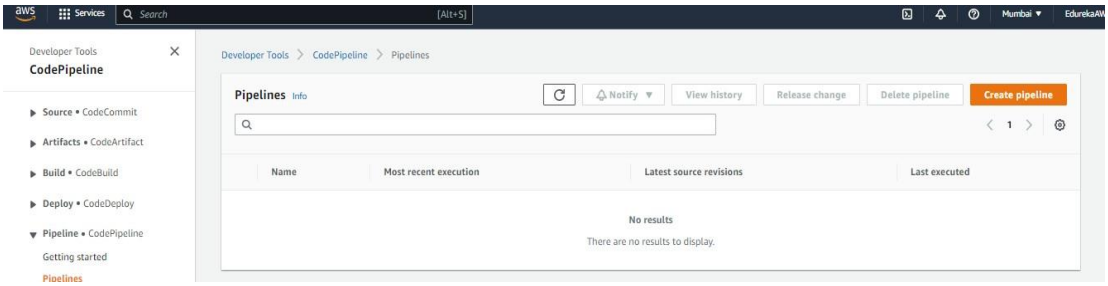

Figure 4.18: Create a CodePipeline

Provide the name to code pipeline for frontend app

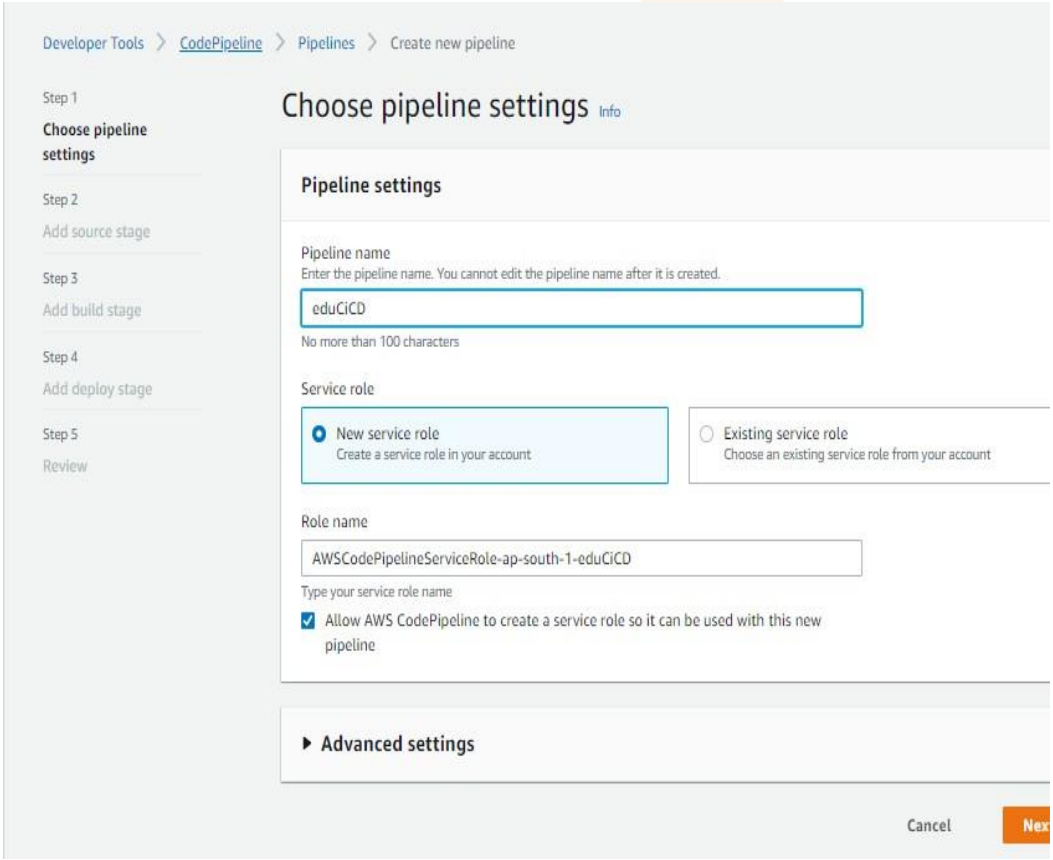

Figure 4.19: Configure the CodePipeline

Now in next step we will select the source from where we have to pick the code and deploy.

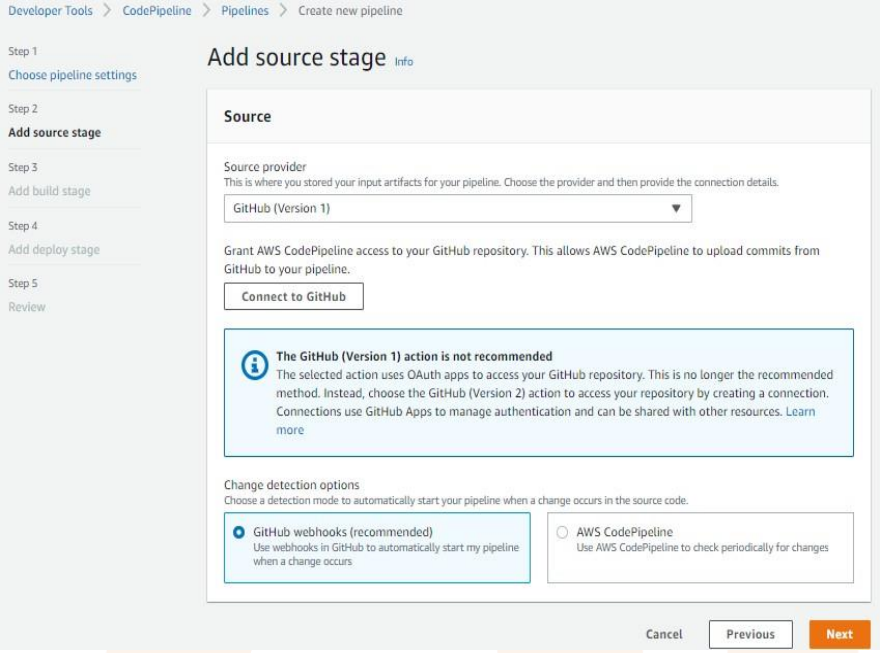

Figure 4.20: Add the Source Provider

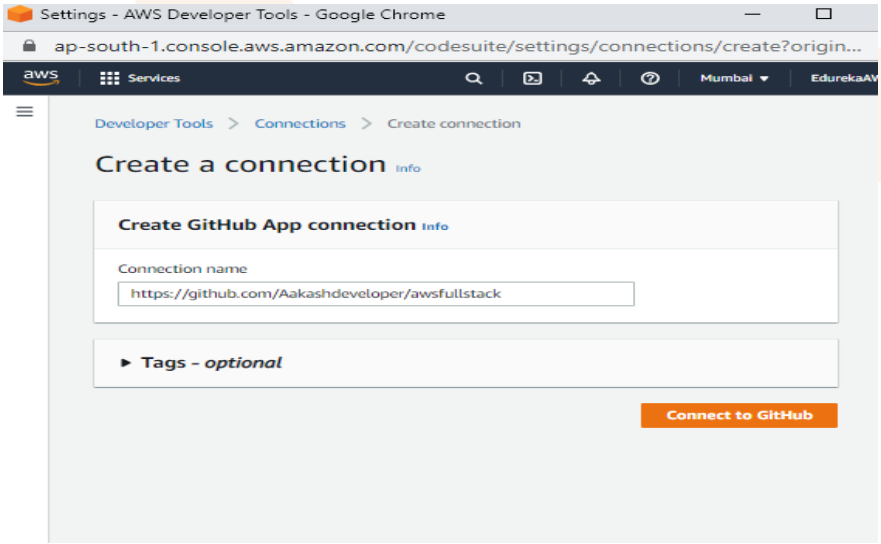

Figure 4.21: Create GitHub connection

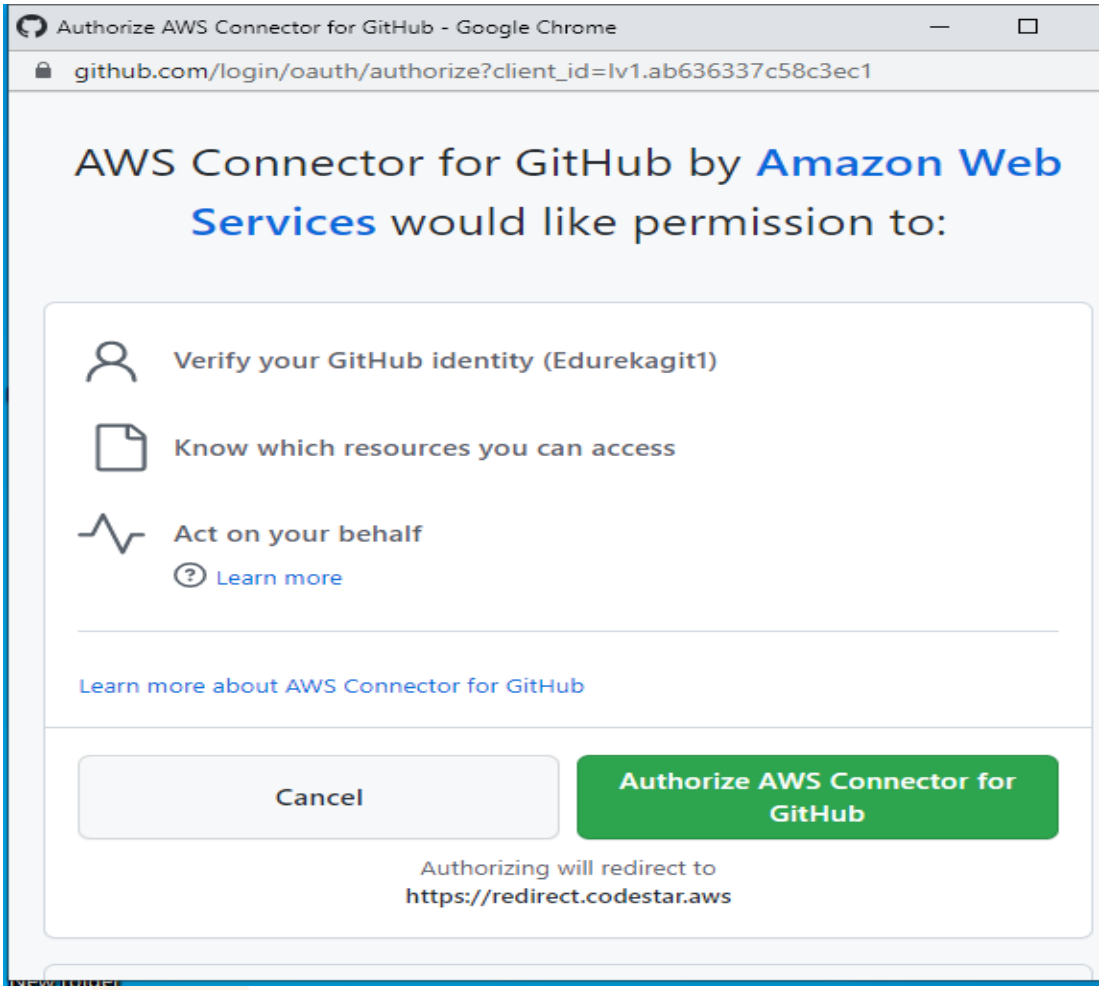

Figure 4.22: Authorize AWS connector for GitHub

Select the repo which you want to link for code pipeline and select the branch.

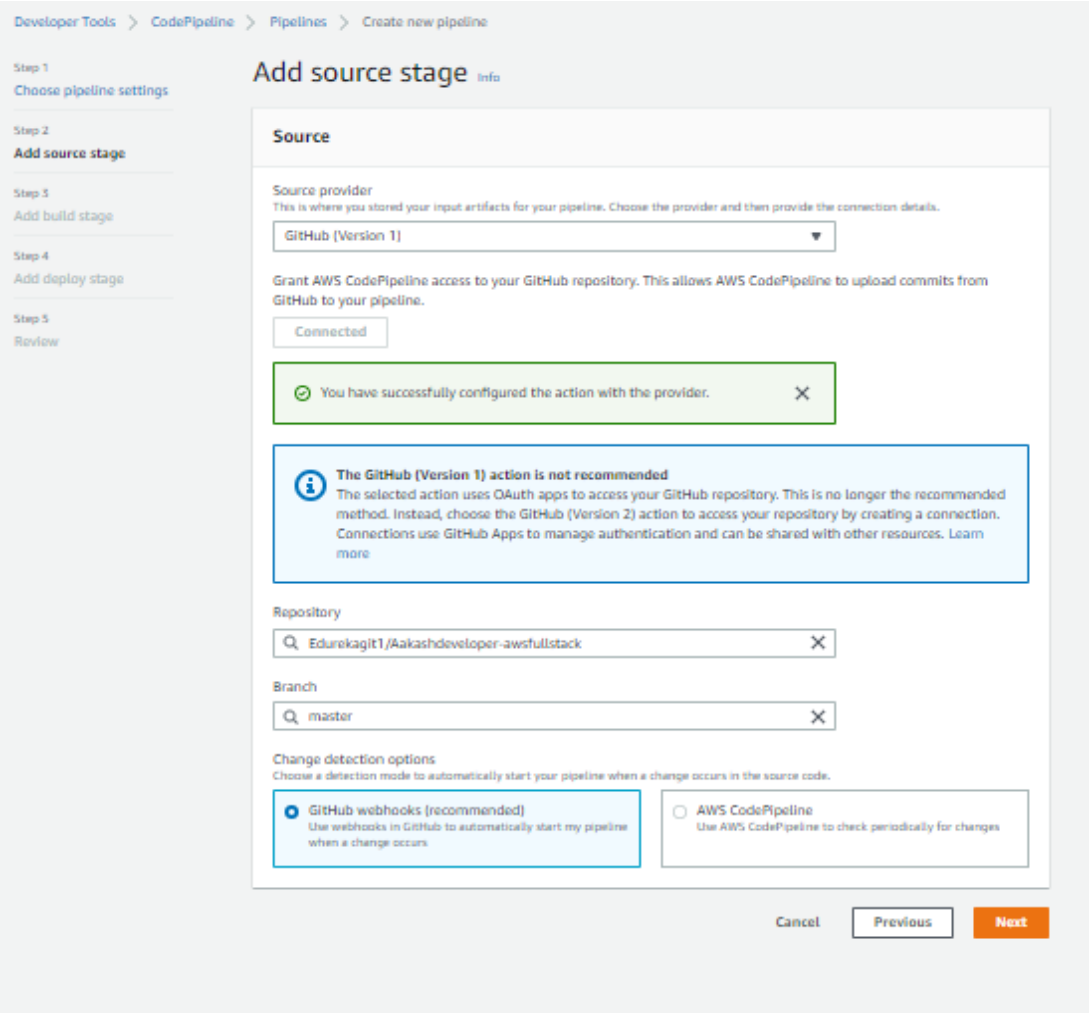

Figure 4.23: Select the repository and branch

Frontend code is available over the repo: https://github.com/Edurekagit1/Aakashdeveloper-awsfullstack This is the react frontend app we need to change the URL of backend python app.

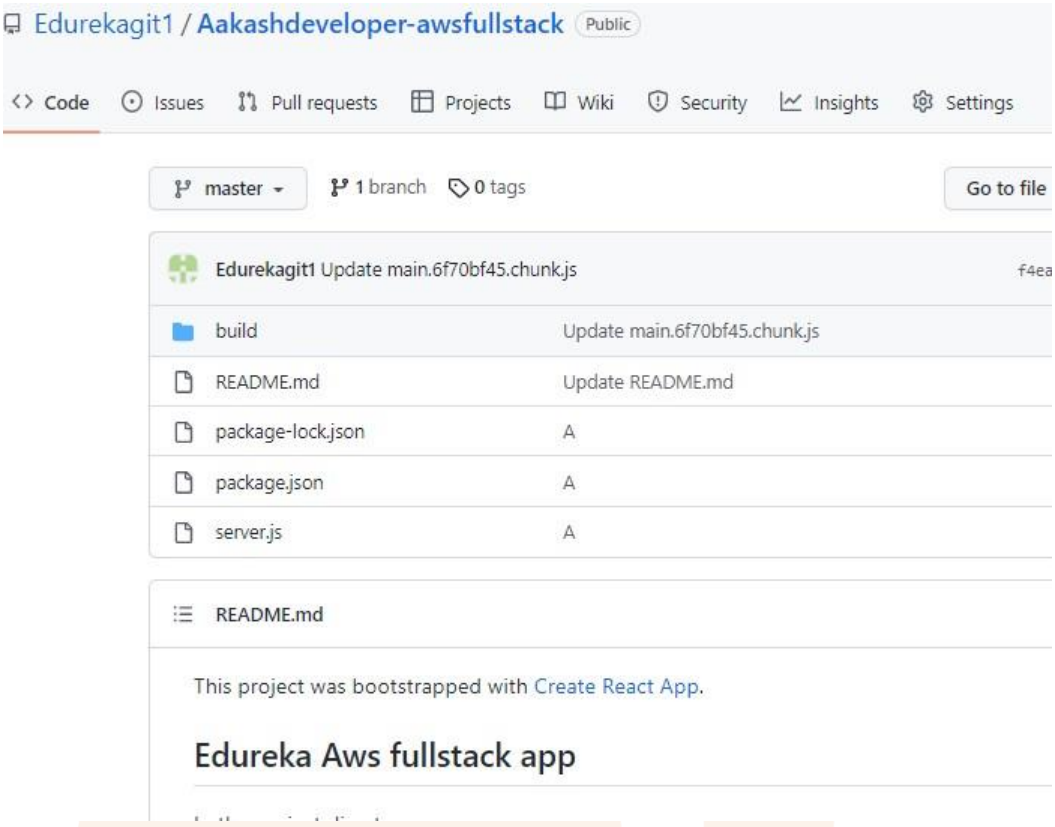

Figure 4.24: Source code in repository

For this demo we can skip the build stage of the application as it more like core script need to build app.

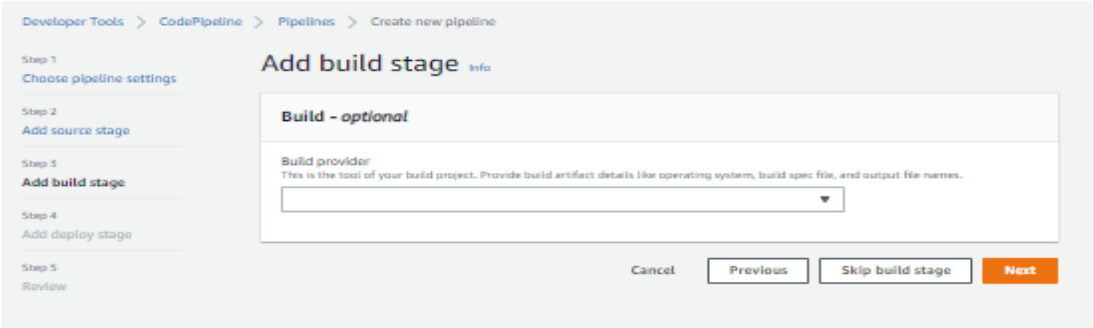

Figure 4.25: Add build stage

Now select the server where we want to deploy the application and i.e. beanstalk that we have launched in earlier step

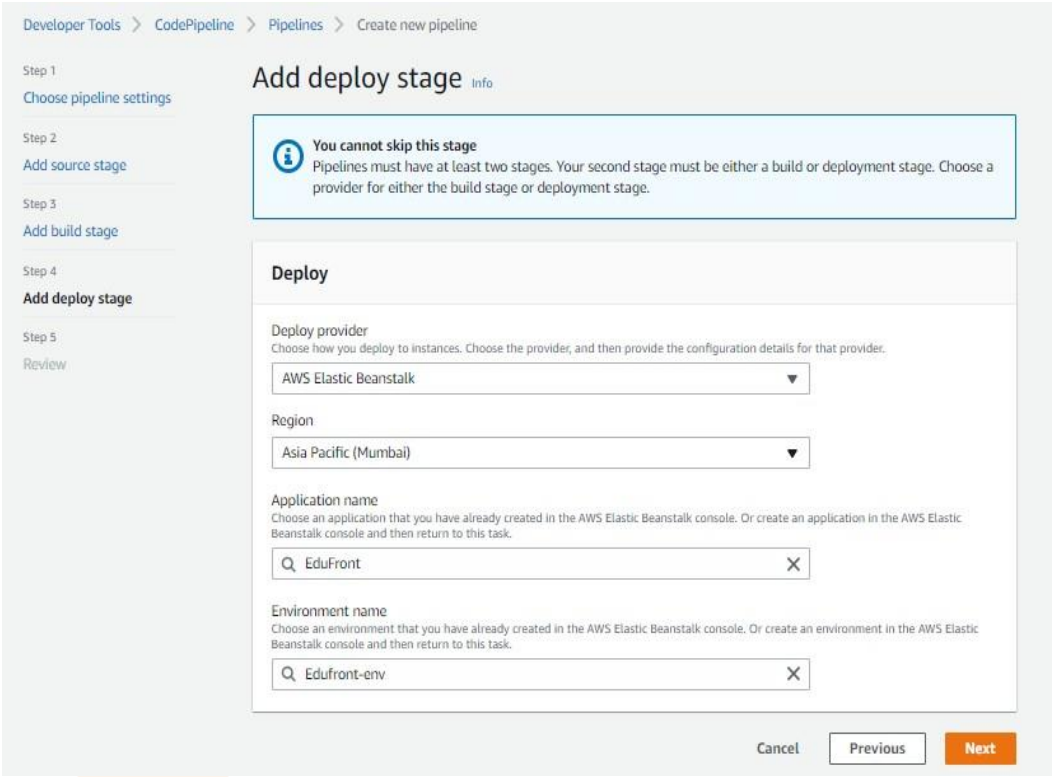

Figure 4.26: Add deploy stage

Once ready our code pipeline is ready to deploy our frontend on the bean stalk.

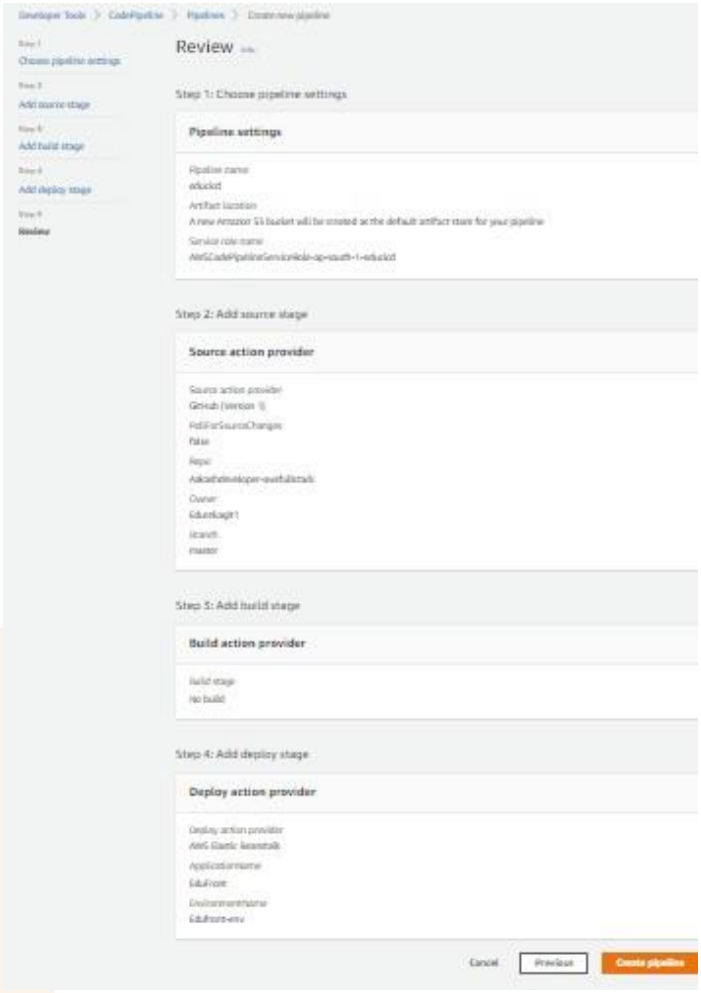

Figure 4.27: Review the Pipeline

As soon as the code commit on the master branch this pipeline will execute and deploy code on the server.

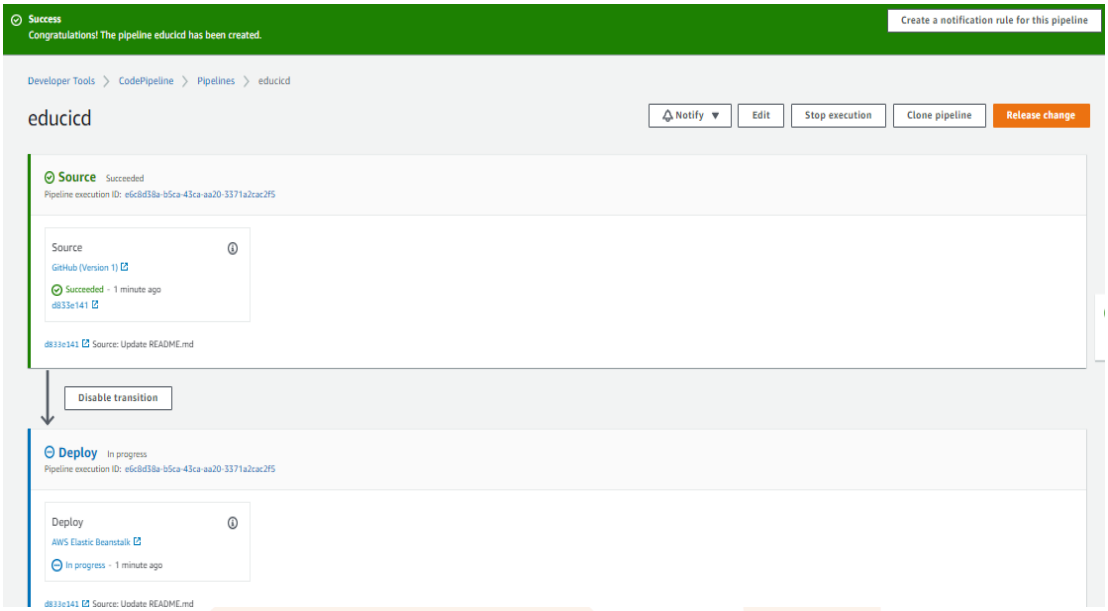

Figure 4.28: CodePipeline source to deploy transition

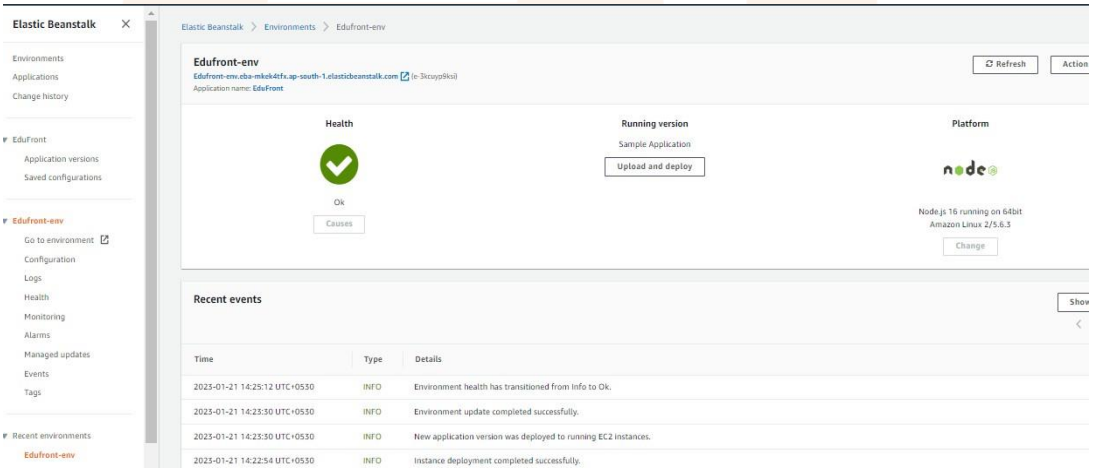

Figure 4.29: Triggered in the Beanstalk

It’s time to start launch the Ec2 into our subnet for backend application inside our VPC

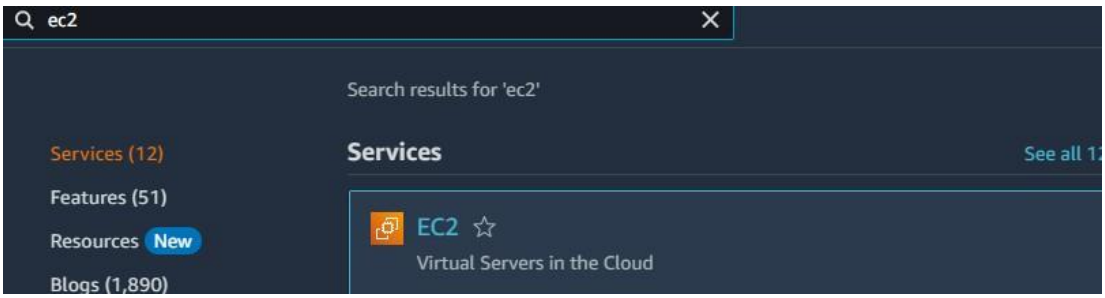

Figure 4.30: Search an EC2

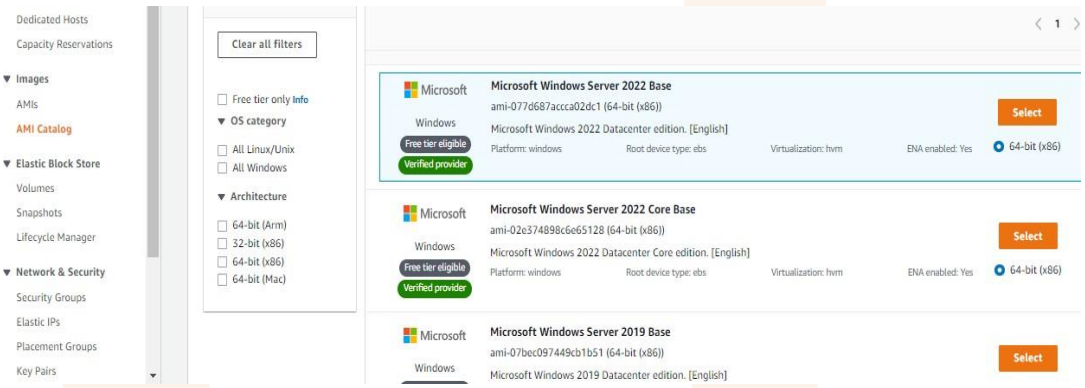

Figure 4.31: Select the AMI

In the configuration tab make sure to check the VPC select should be our VPC and subnet should be public.

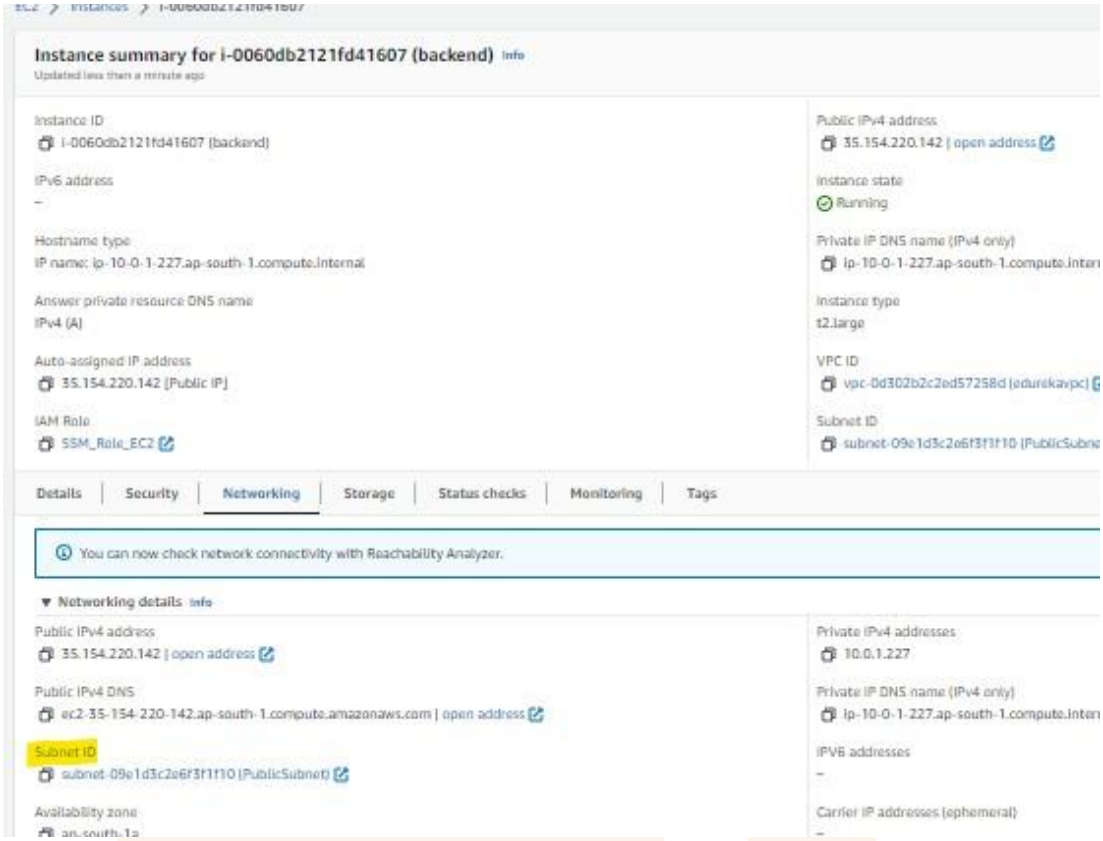

Figure 4.32: Backend instance configuration

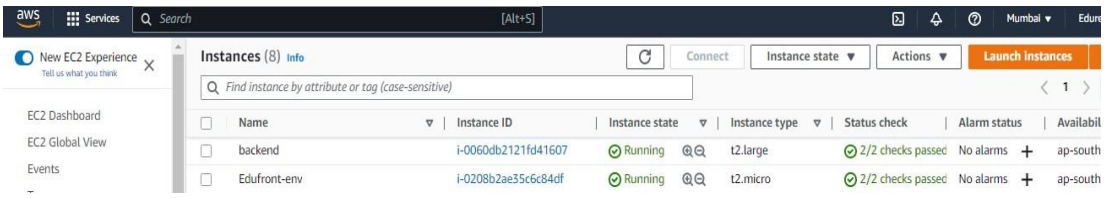

Figure 4.33: Backend instance configuration

Connect to backend instance using the RDP client through fleet manager remote desktop

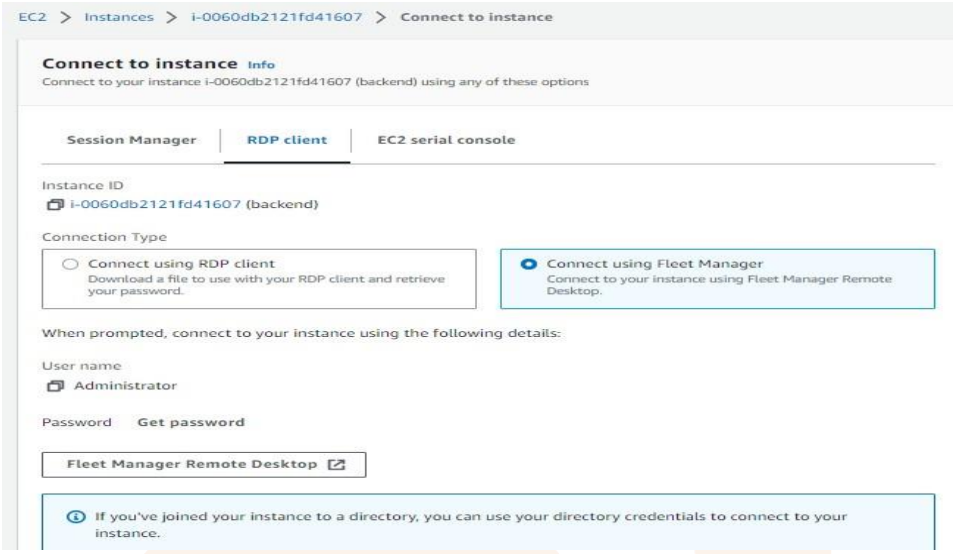

Figure 4.34: Connect using Fleet manager

Once by frontend and backend is ready, we have to launch our database (RDS) in private subnet

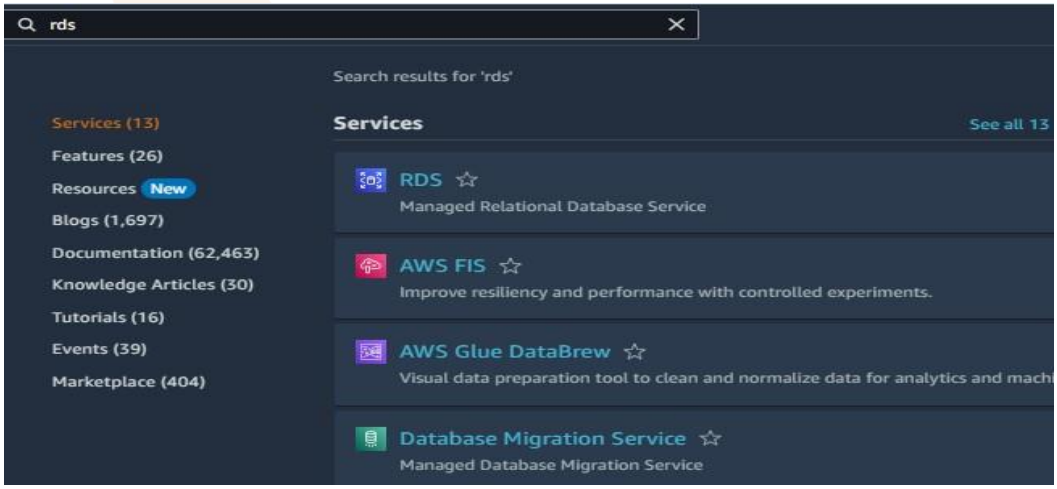

Figure 4.35: Search the RDS

Select the database engine and database creation method for our application.

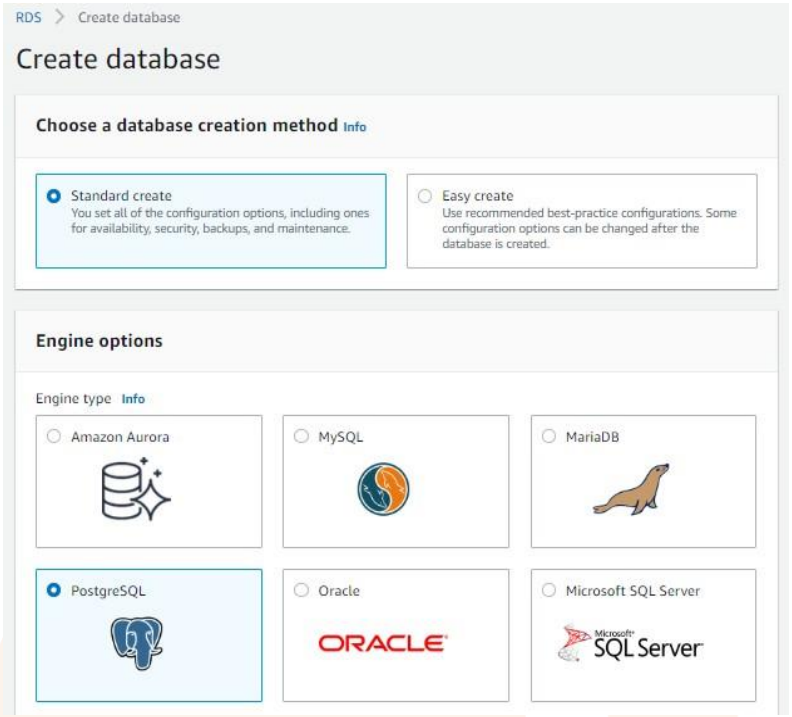

Figure 4.36: Create a database

While launching make sure that we have select our own created VPC for security of DB Server

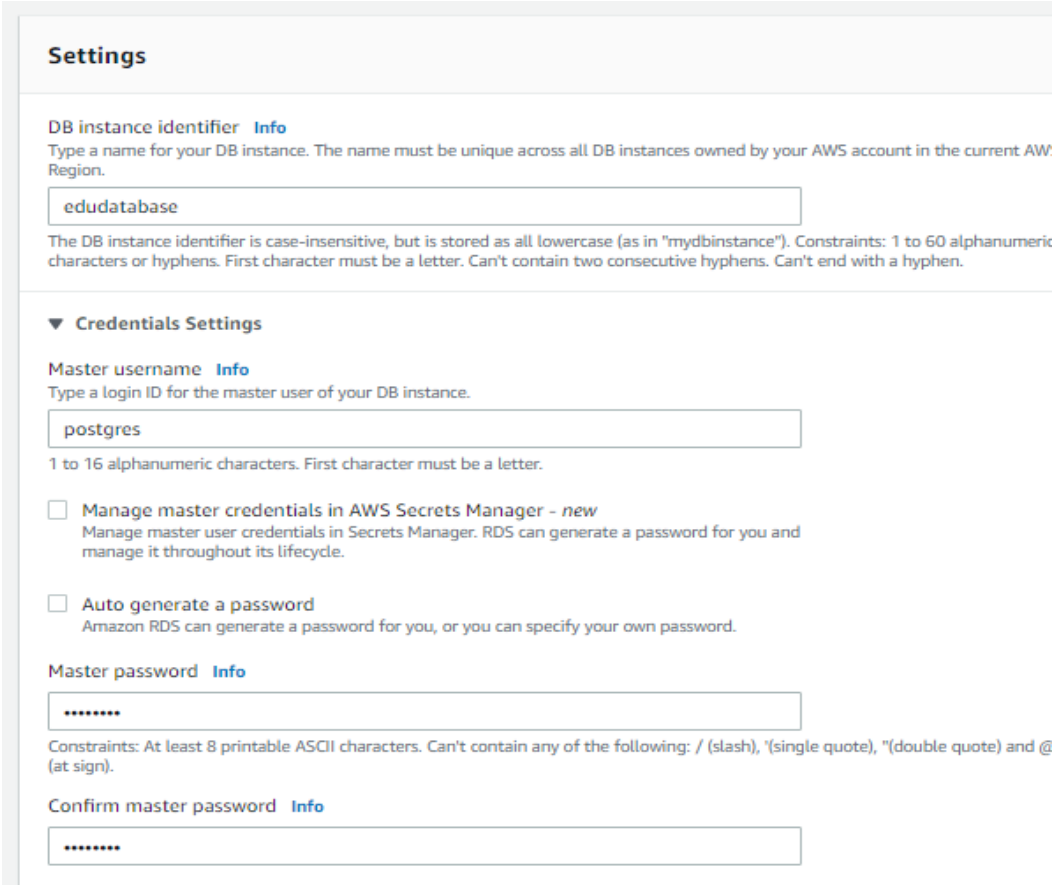

Figure 4.37: Configure the Database

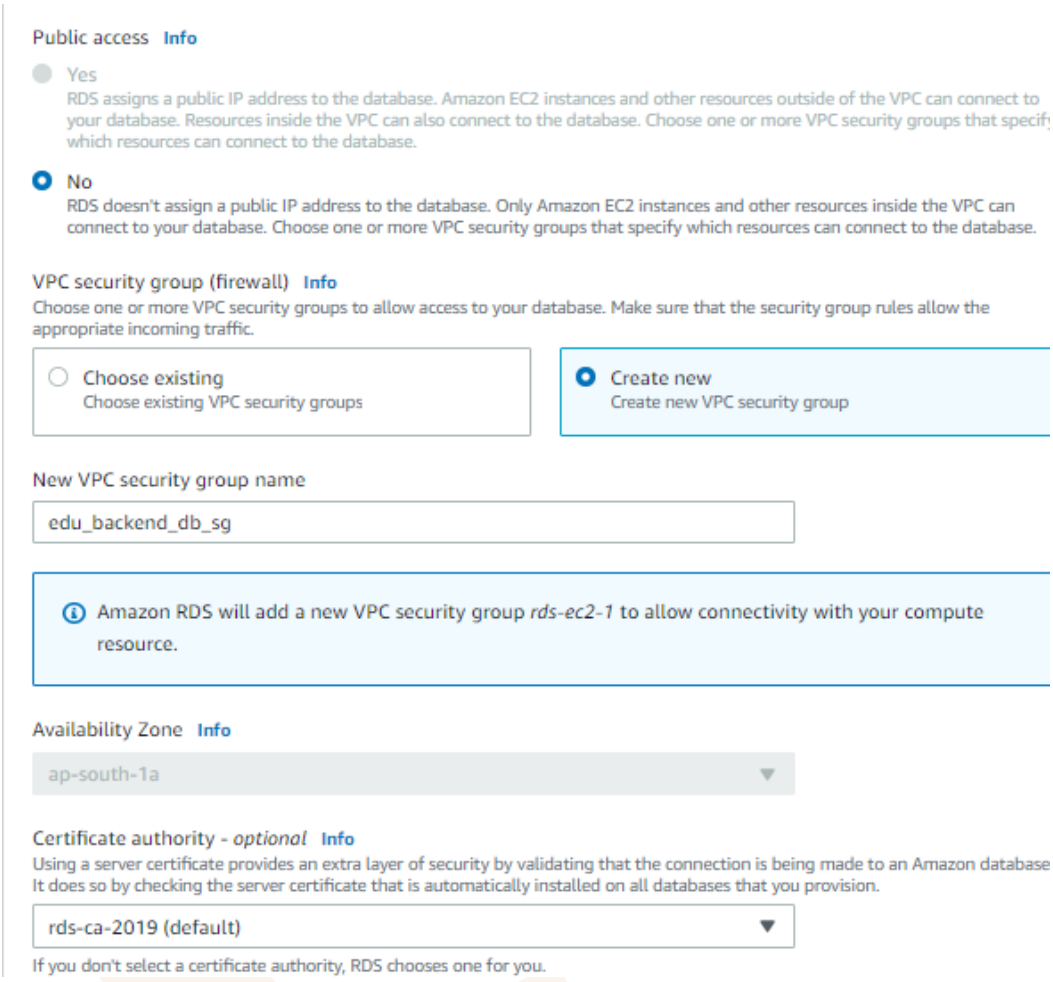

Figure 4.38: Create a new Group Name

RDS will create a new sg for backend ec2 instance to allow connectivity with db instance we created.

Connect to backend instance using fleet manager by installing SSM agent through IAM role.

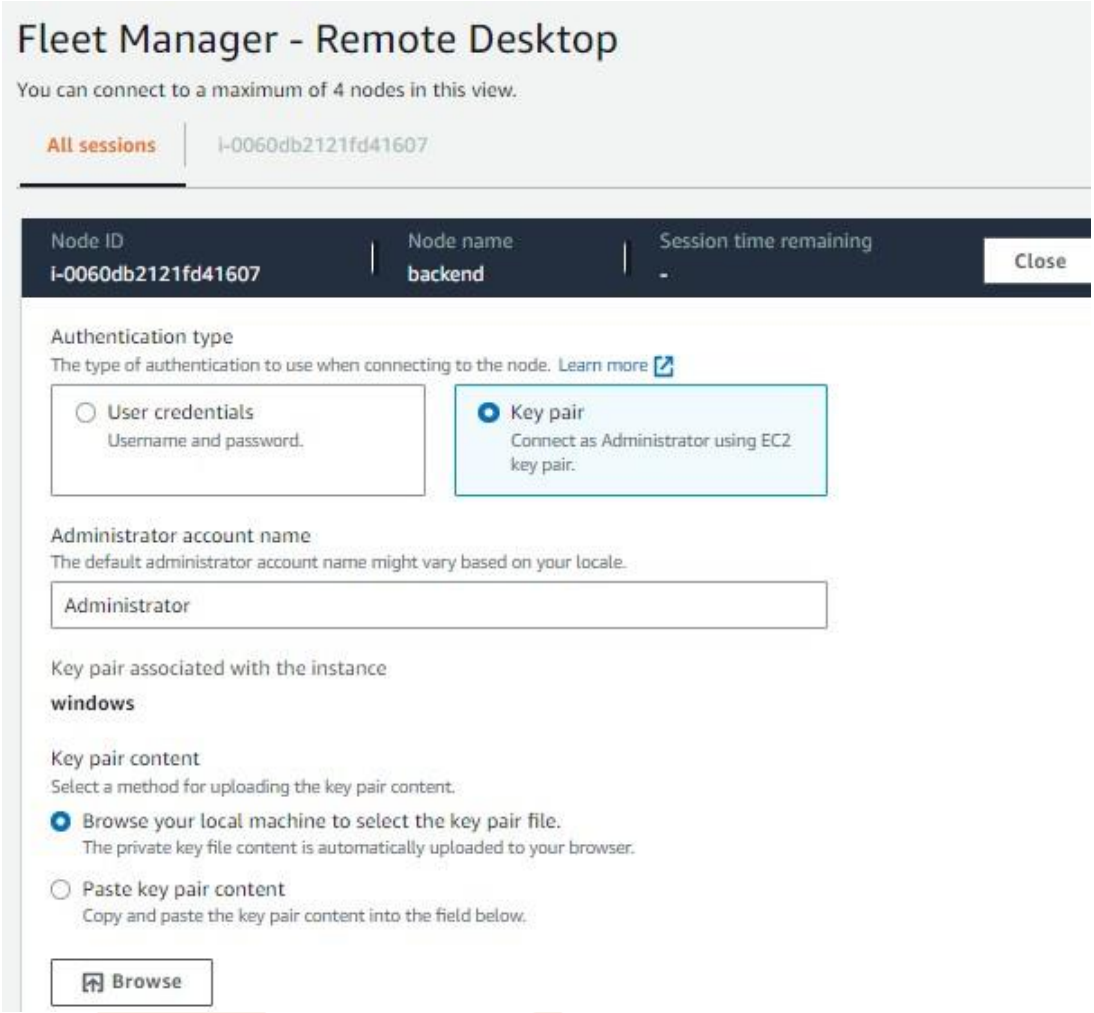

Figure 4.39: Set-up the Authentication Type

Figure 4.40: Windows Remote Desktop- Fleet Manager

Over the window instance install python for running our backend application

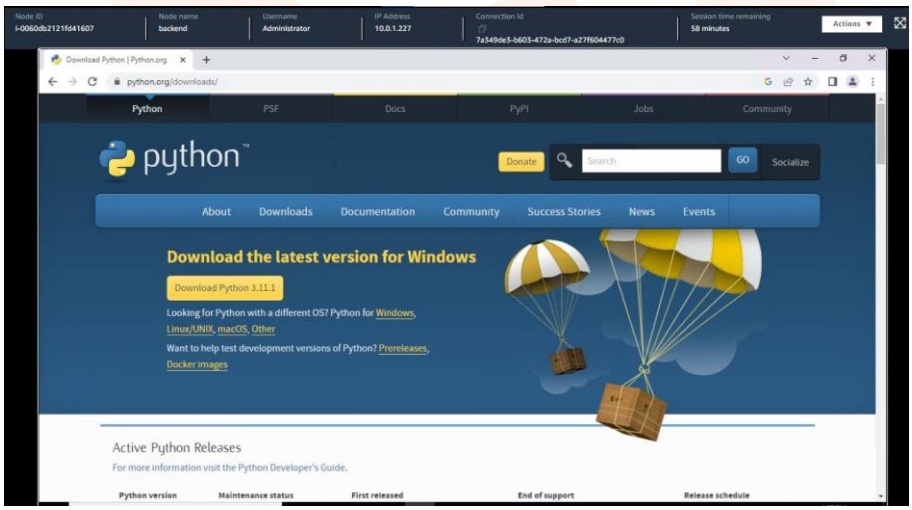

Figure 4.41: Download the Python

Now navigate to command prompt inside instance and type python to make sure python is running.

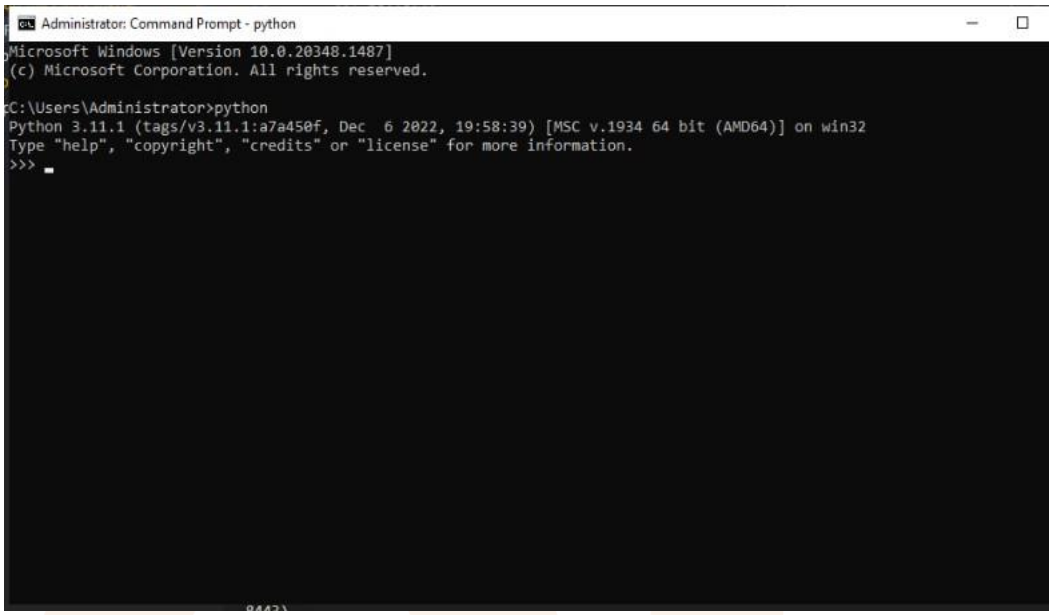

Figure 4.42: Command Prompt- python

Now navigate to https://github.com/Aakashdeveloper And open Python-aws-rds repo and clone the code

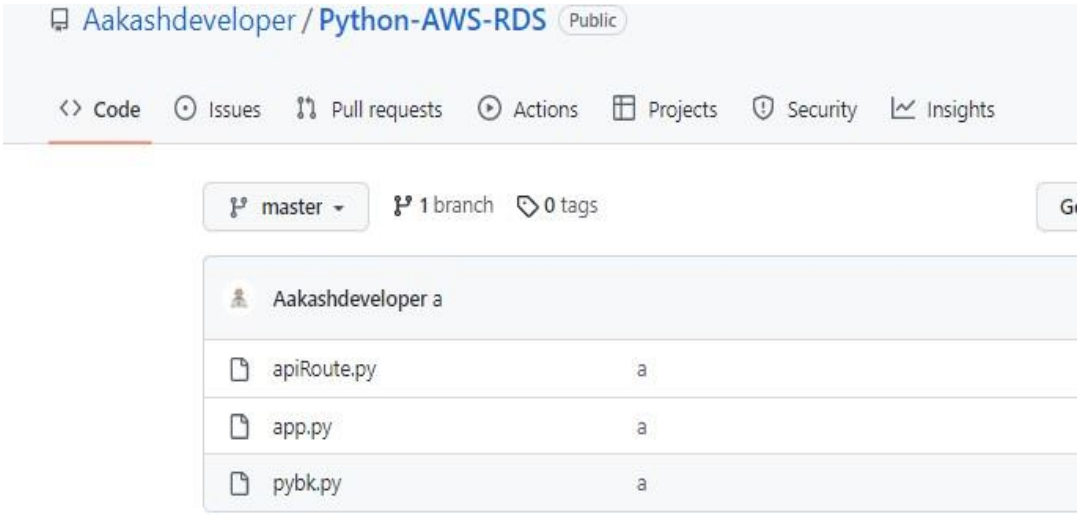

Figure 4.43: Command Prompt- python

Now as we have to connect to DB instance from the window server. For postgres we have to install pgadmin on the window server. https://www.pgadmin.org/download/

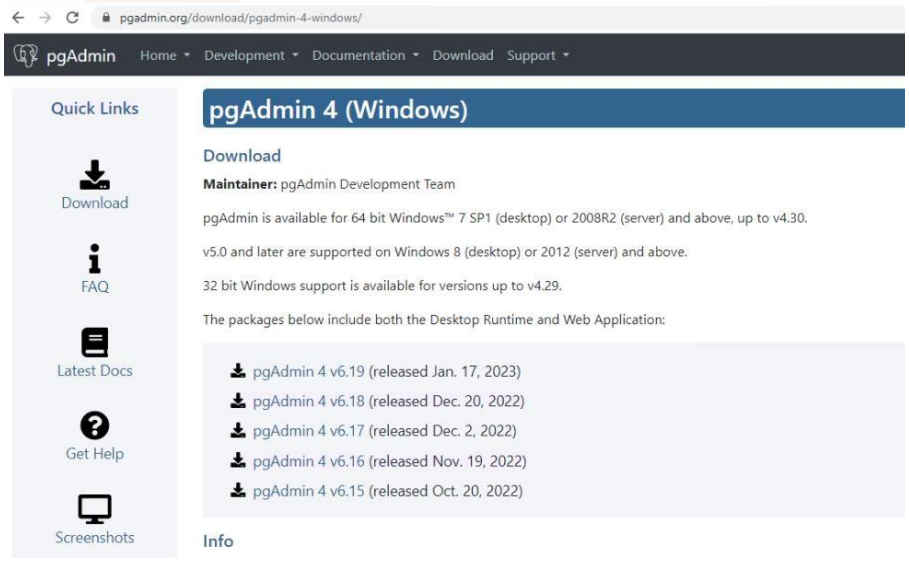

Figure 4.44: Download pg admin

Visual studio code is also required for code editing and checking python code.

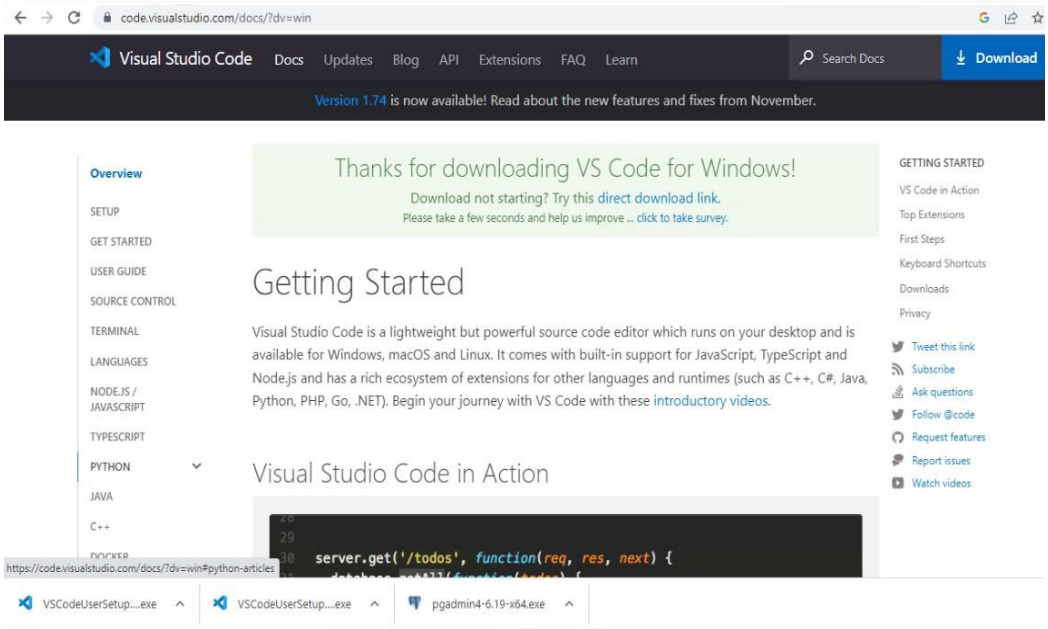

Figure 4.45: Download Visual Studio

On Line number 26 in apiRoute.py file we need to add credentials for our DB which we have launched change host, port ,password and dbname.

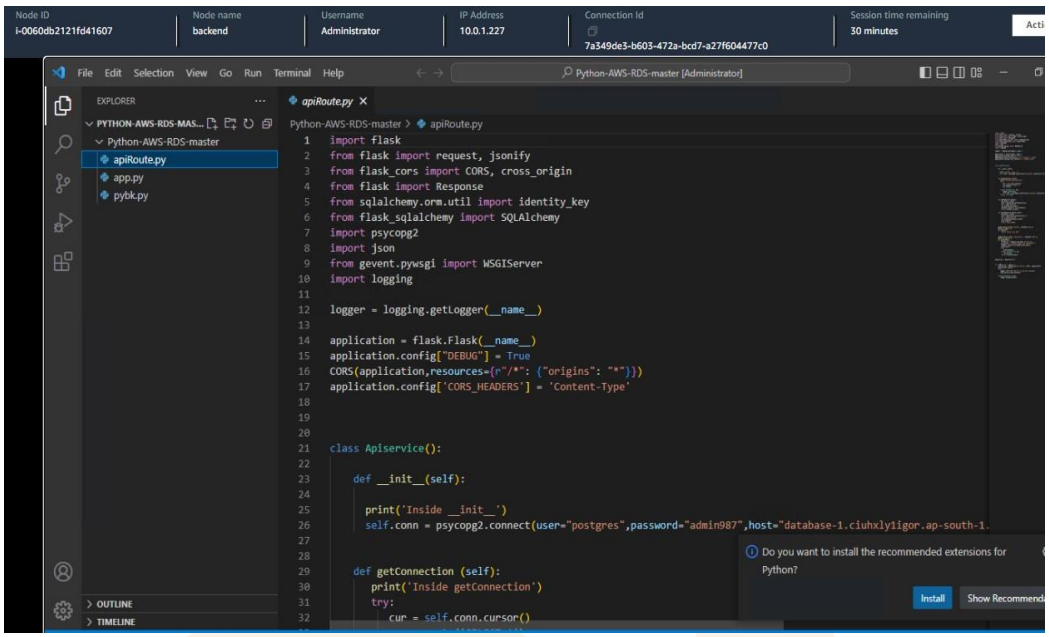

Figure 4.46: Visual Studio – Python AWS RDS Master

Add Select * from restaurant in line 69

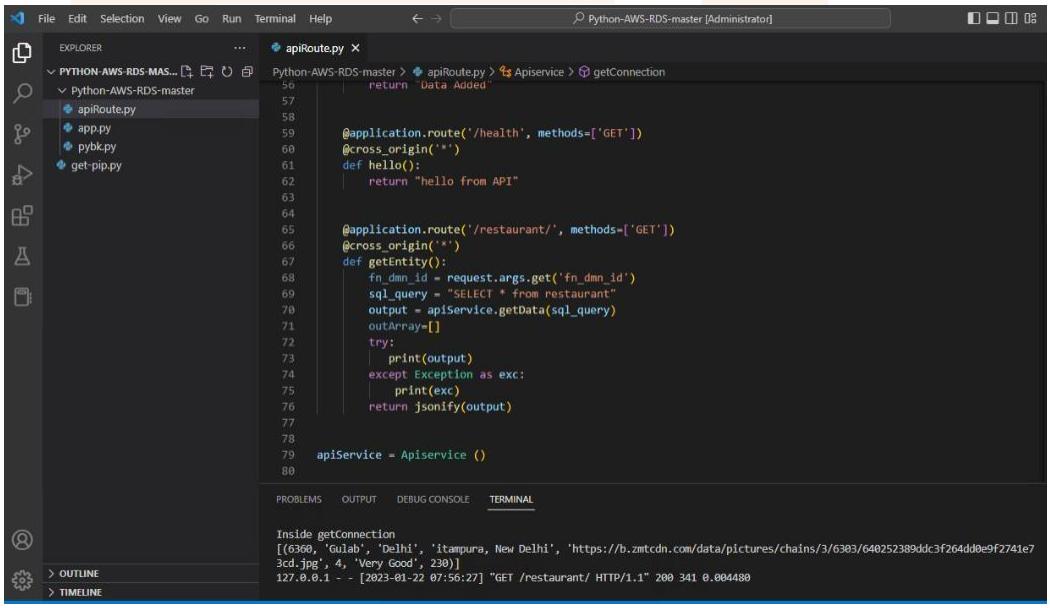

Figure 4.47: Check the terminal

We can get the credential for DB connection on the database page in connectivity section.

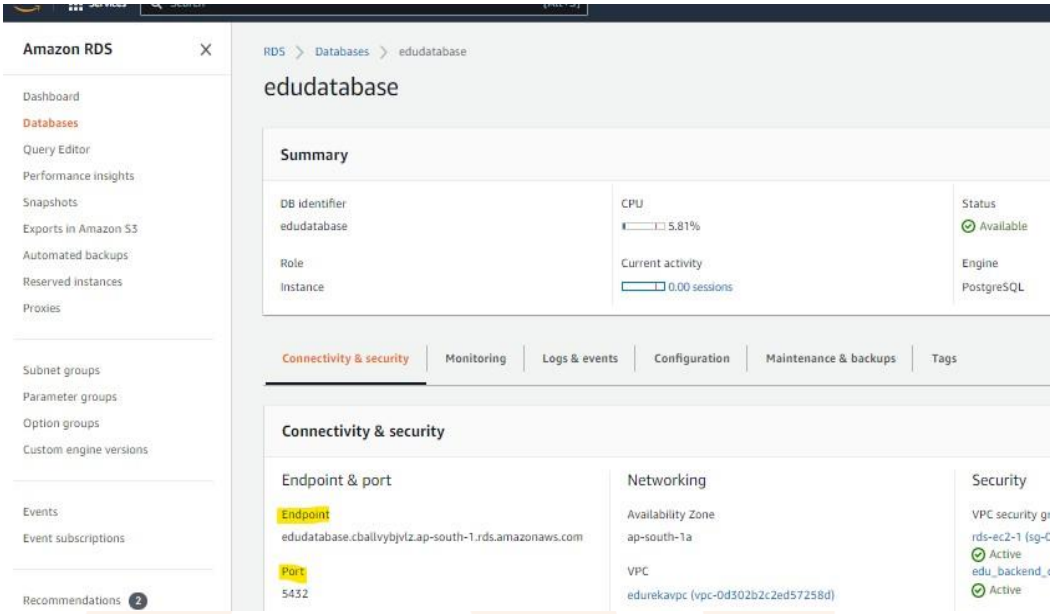

Figure 4.48: Note the RDS Endpoint and Port

Now on public Ec2 machine with pg admin login to postgres and create table and add column to the database.

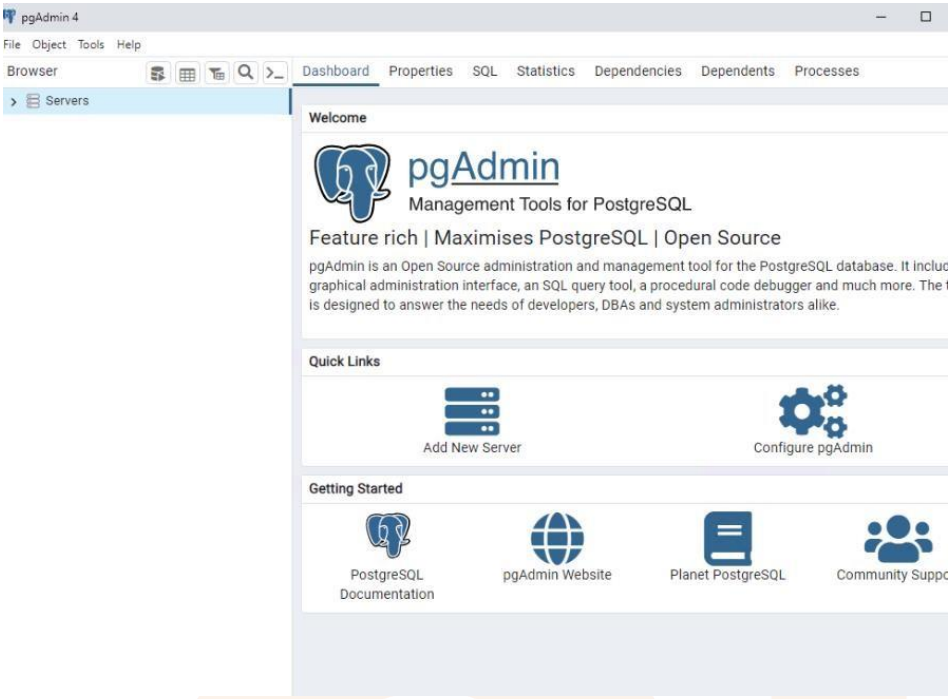

Figure 4.49: pgAdmin: Add New Server

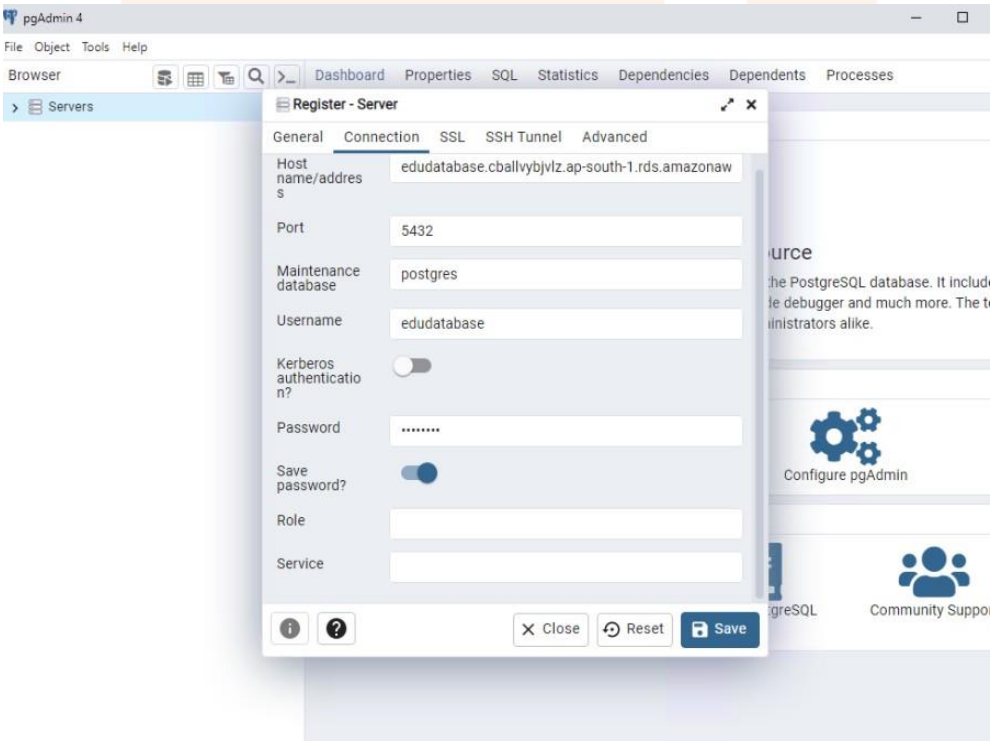

Figure 4.50: Register-Server Configure Connection

Add db credentials and details

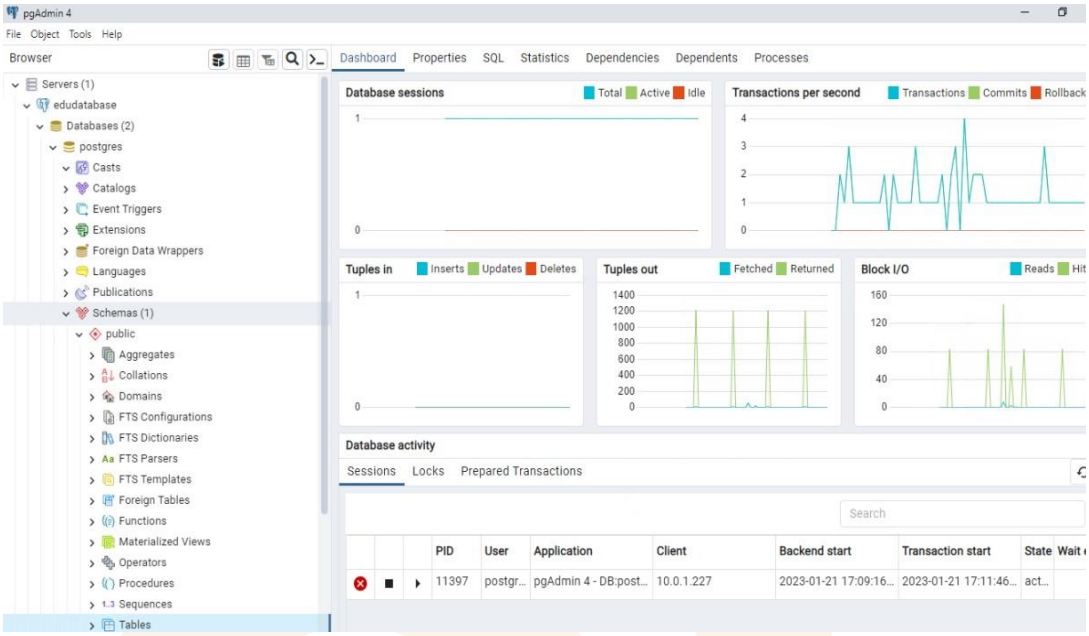

Figure 4.51: Select the Tables

Now we will create a table in the edudatabase

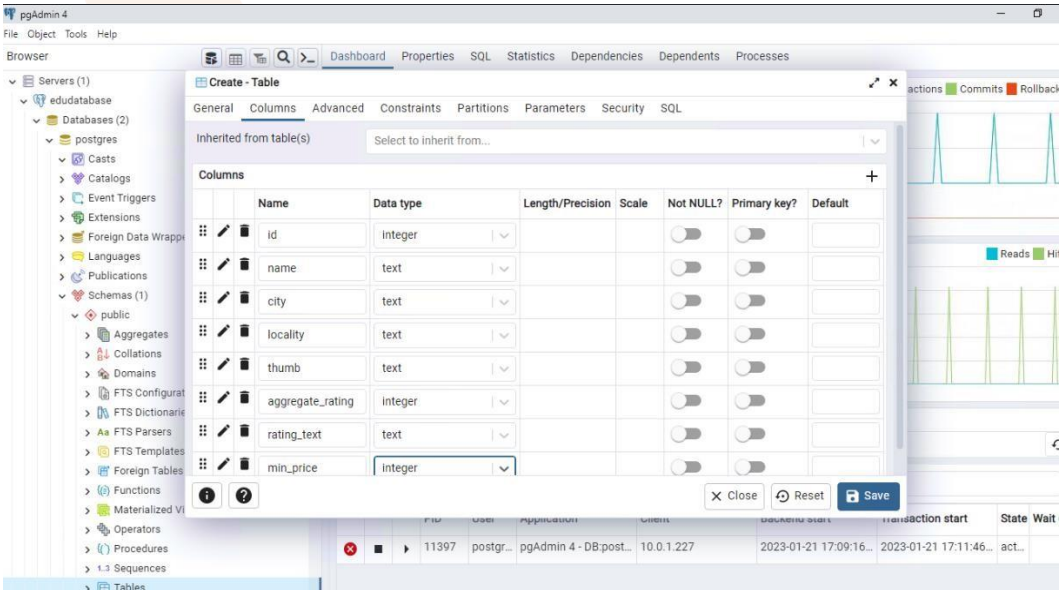

Figure 4.52: Create- Table

Insert the record into the table using pg admin console and verify the insertion with select query.

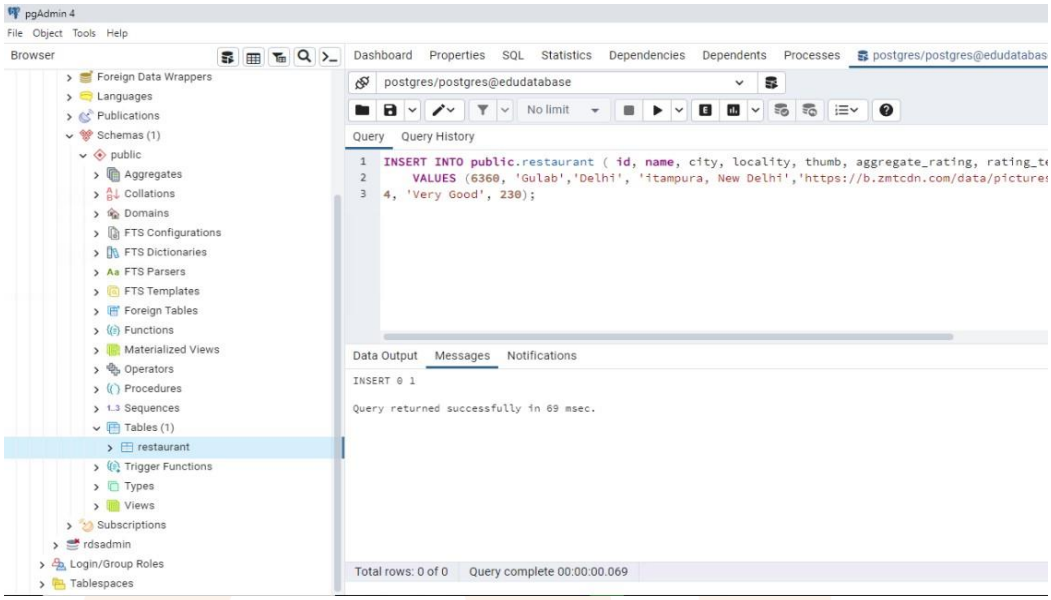

Figure 4.53: Run the Server

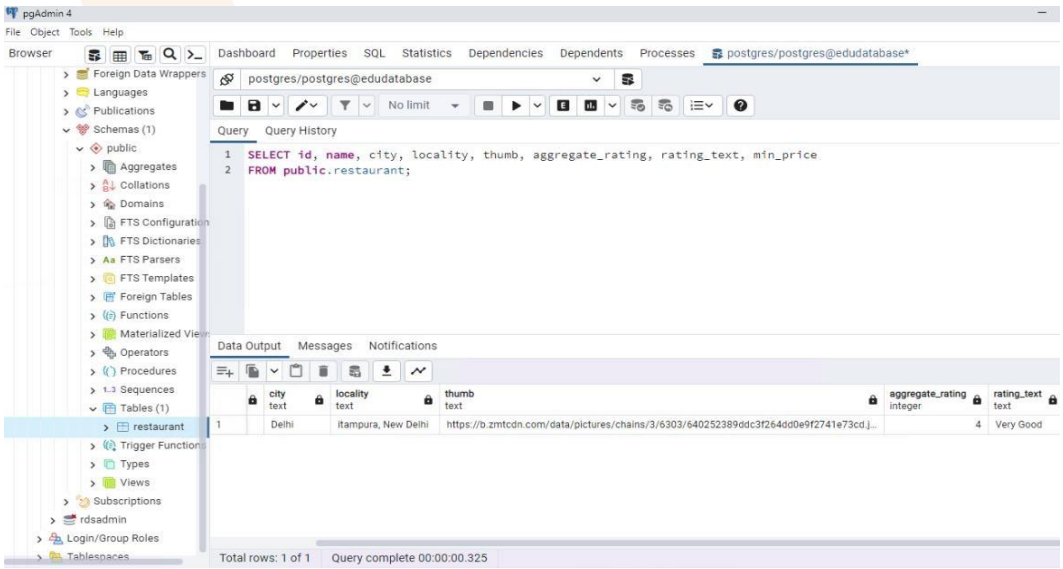

Figure 4.54: Run the query

Now we check our api code we can see the data inserted in private subnet into RDS is now accessible from the public Ec2.

Run this apiRoute.py code and check for **http://35.154.220.142:8443/restaurant/** in browser

Figure 4.55: Run the localhost

As our API is now ready we can connect backend to frontend and access data from the frontend

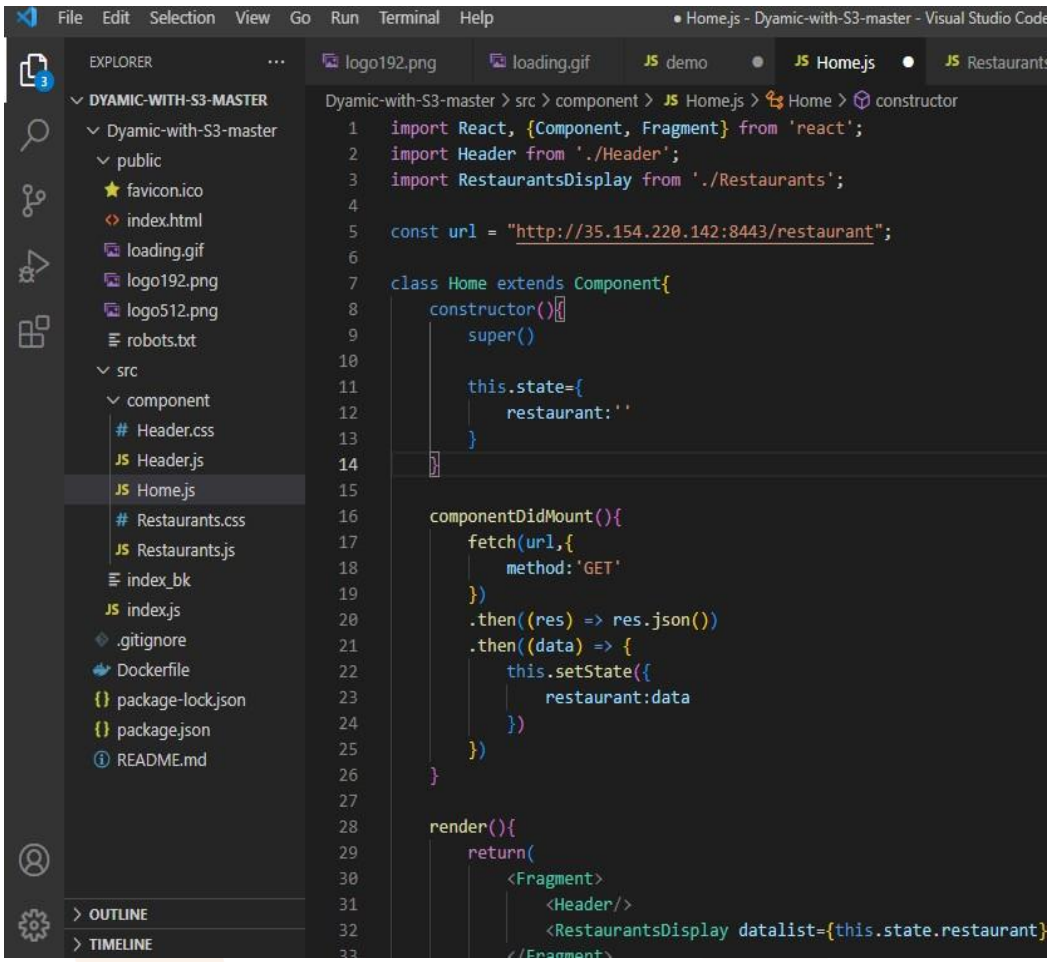

Figure 4.56: Change the const url in the code

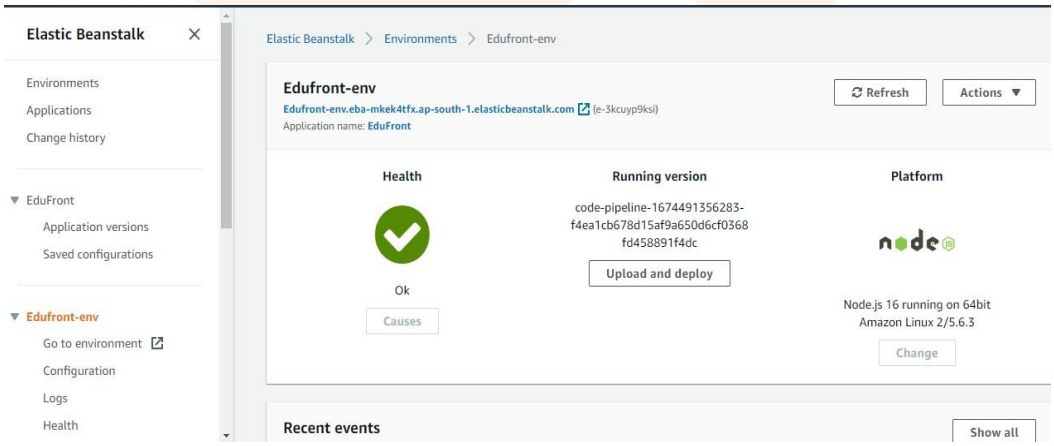

Figure 4.57: Click on the Edufront-env link

We can see the final output of the application on the screen.

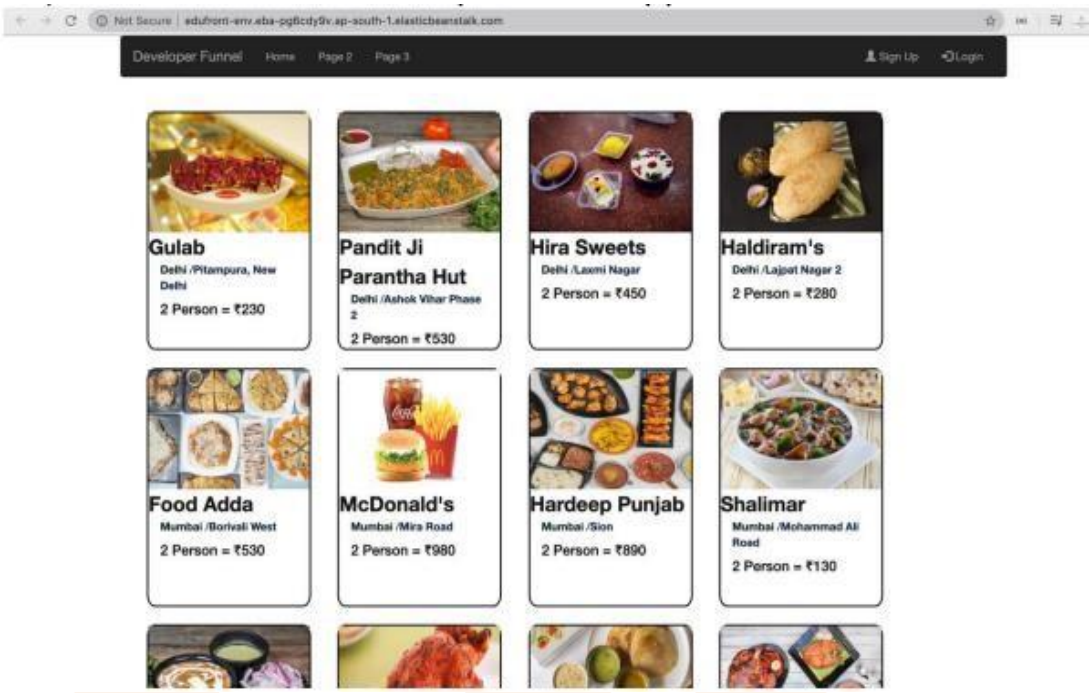

Figure 4.58: Final Output