# **Three-Tier Web Architecture on AWS**

## 1. Objective

To design and deploy a secure, scalable, and modular 3-tier architecture on AWS, separating the application logic, presentation, and database layers using EC2, RDS, S3, IAM, Load Balancers, and other AWS services.

## 2. Architecture Overview

### Tier 1: Presentation Layer (Web Tier)

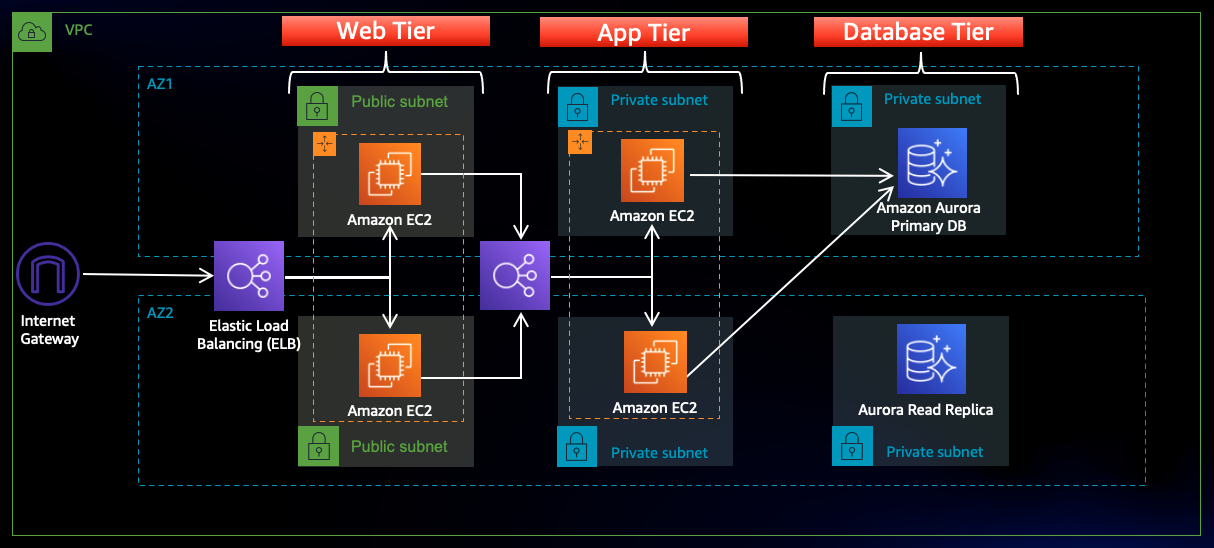
* Hosted on EC2 instance
* Uses **Nginx** to serve the frontend application
* Publicly accessible
* Communicates with the internal load balancer to route requests to the App Tier

### Tier 2: Application Layer (App Tier)

* Backend logic using **Node.js**
* Deployed on private EC2 instance
* Managed using **PM2** for process handling
* Communicates with the database securely

### Tier 3: Data Layer (Database Tier)

* **MySQL RDS instance** hosted in private subnets
* No public access for enhanced security

****

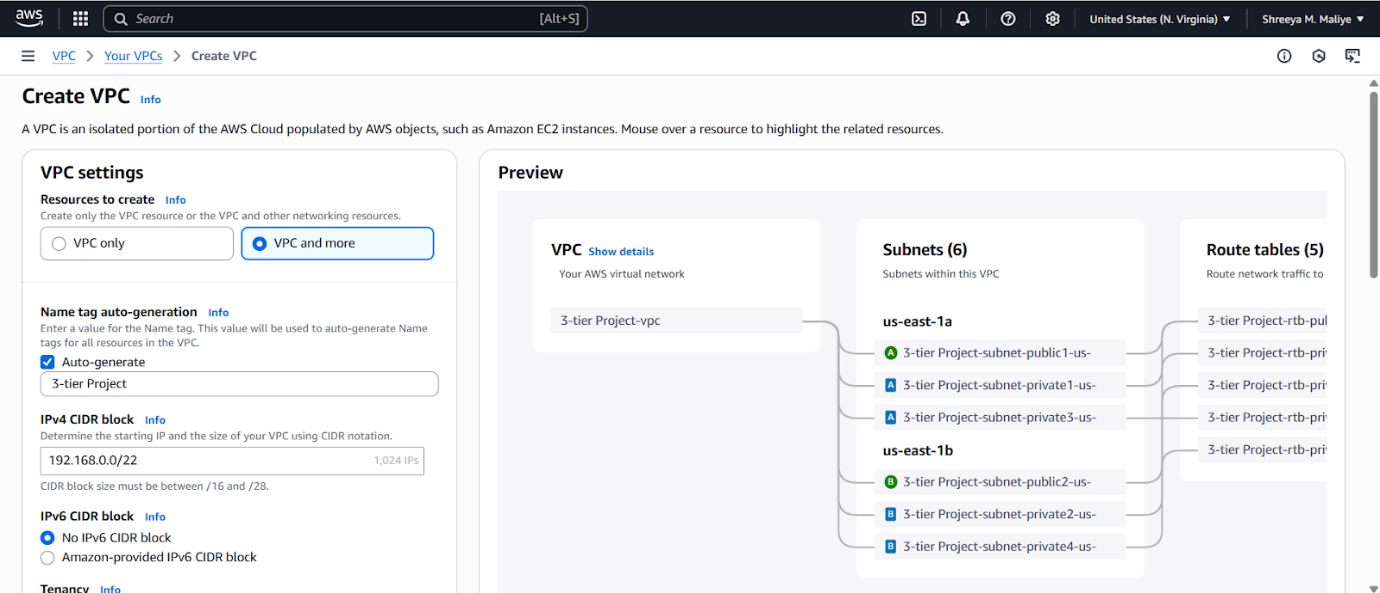
## 3. AWS Services Used

| **Service** | **Purpose** |
| --- | --- |
| VPC | Isolated network space |
| Subnets | Logical separation of resources (public & private) |
| EC2 | Hosts Web Tier and App Tier applications |
| RDS | Managed MySQL database for Data Tier |
| S3 | Stores source code, configs (nginx, dbconfig.js) |
| IAM | Secure permission control |
| Load Balancers | Distribute traffic between App/Web tiers |
| Target Groups | Attach EC2s to Load Balancers |
| Security Groups | Control traffic flow |
| Session Manager | Secure EC2 connection without SSH |

## 4. Step-by-Step Implementation

### Step 1: VPC and Subnet Setup

* Create custom VPC
* Add Public & Private Subnets in different Availability Zones

****

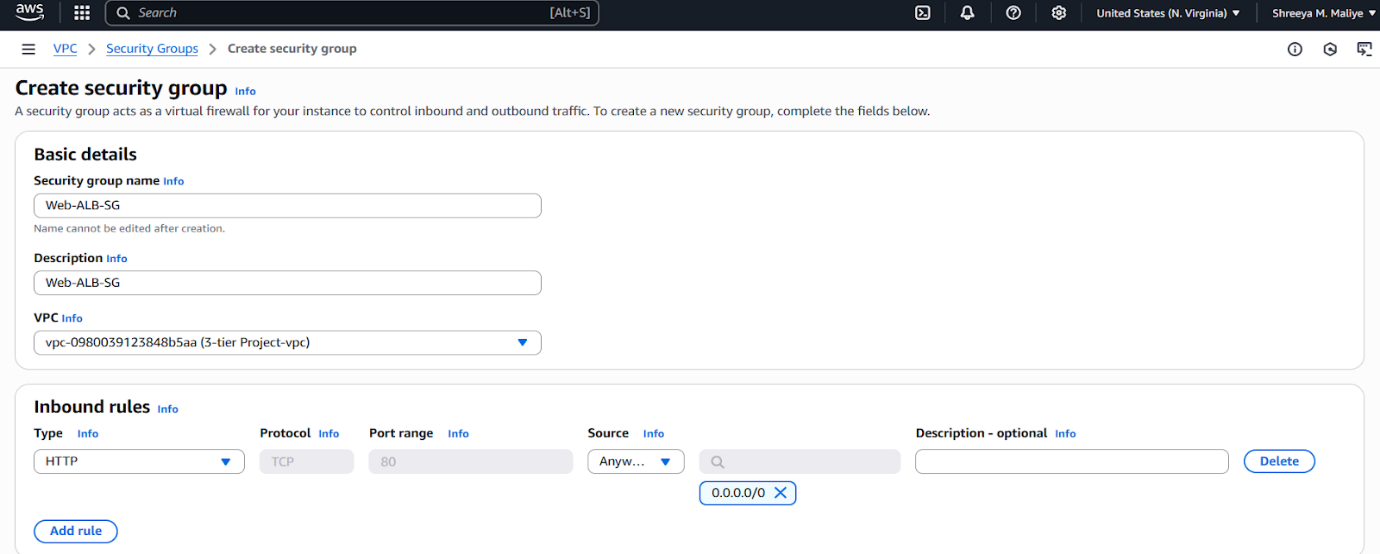
 Resource map diagram

****

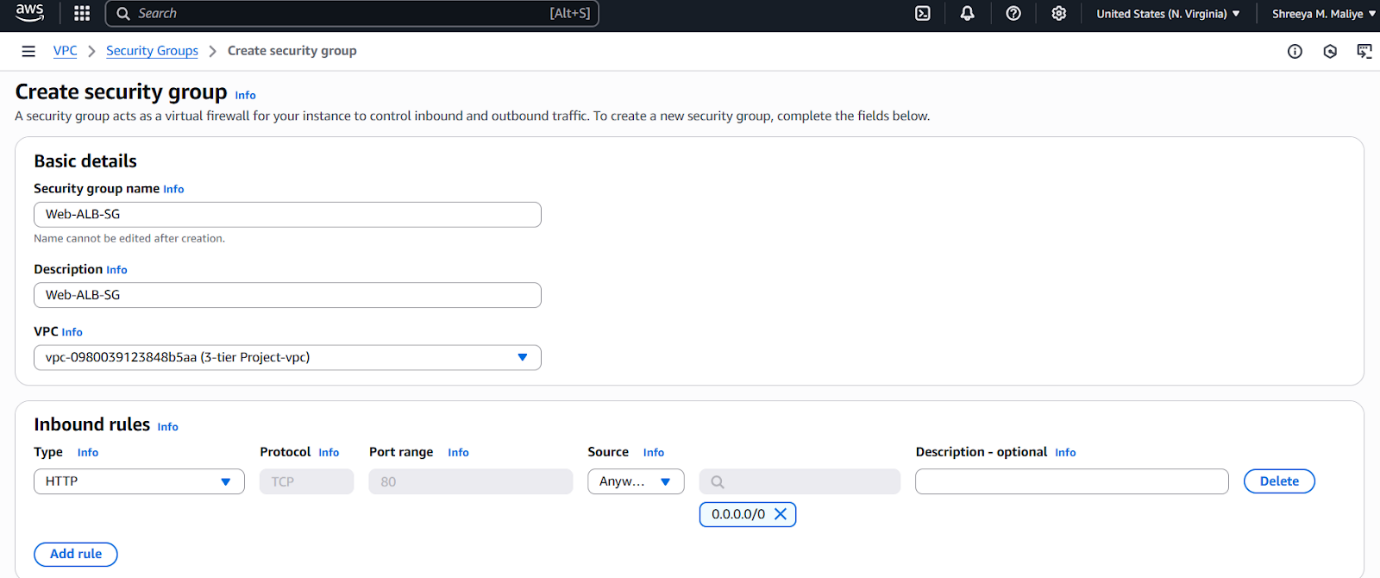
### Step 2: Create Security Groups

Create the following:

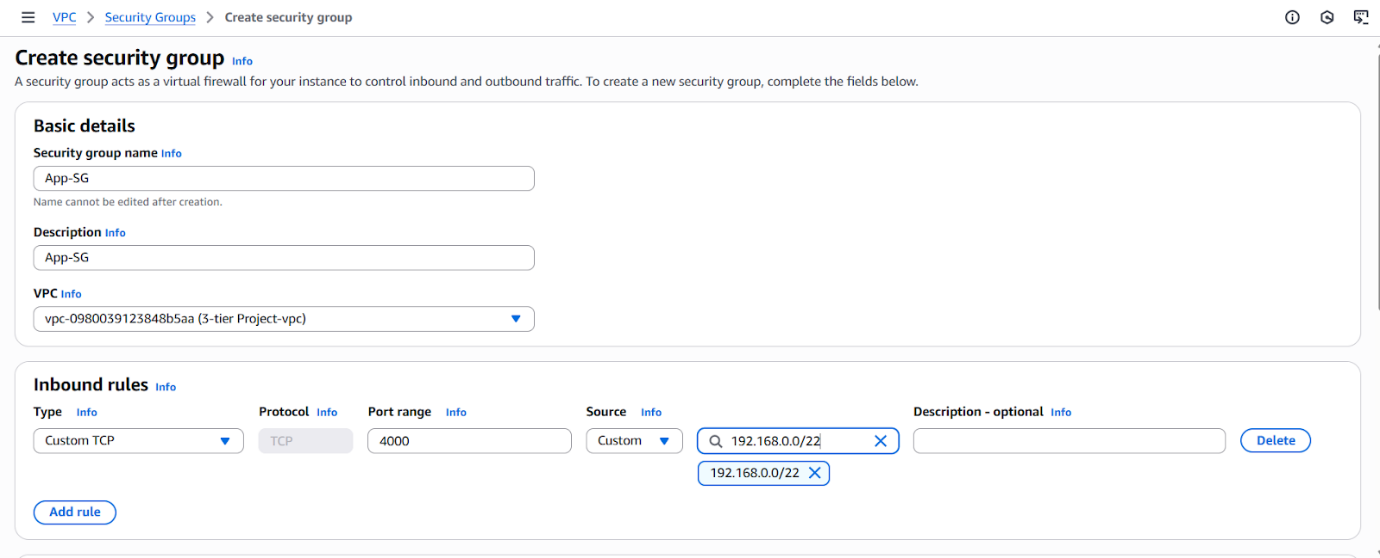
* Security Group for External Load Balancer

****

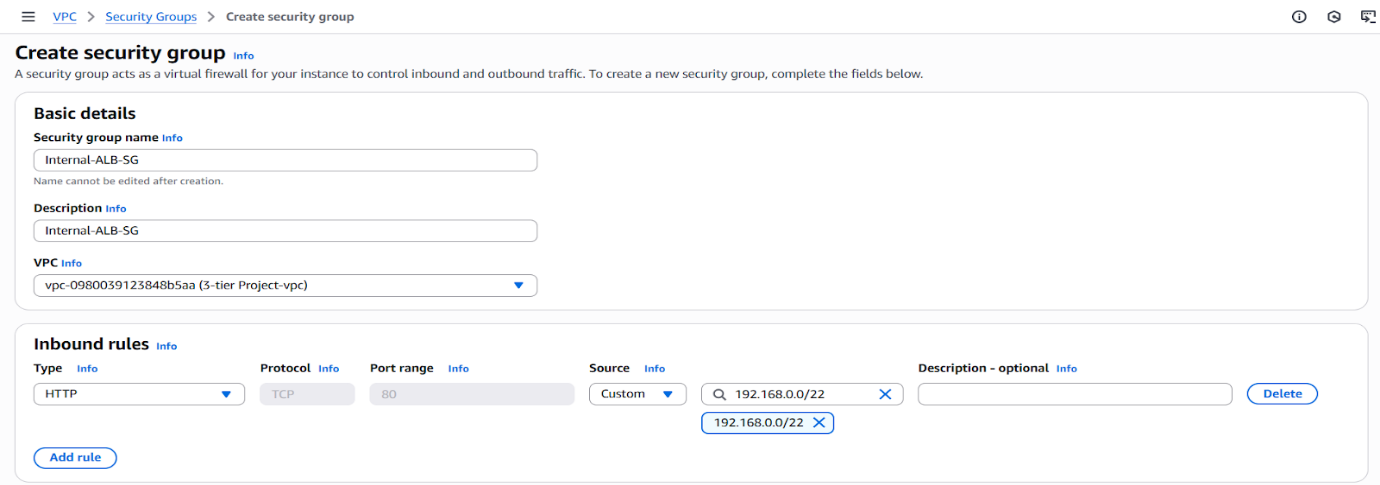
* Web Tier Security Group

****

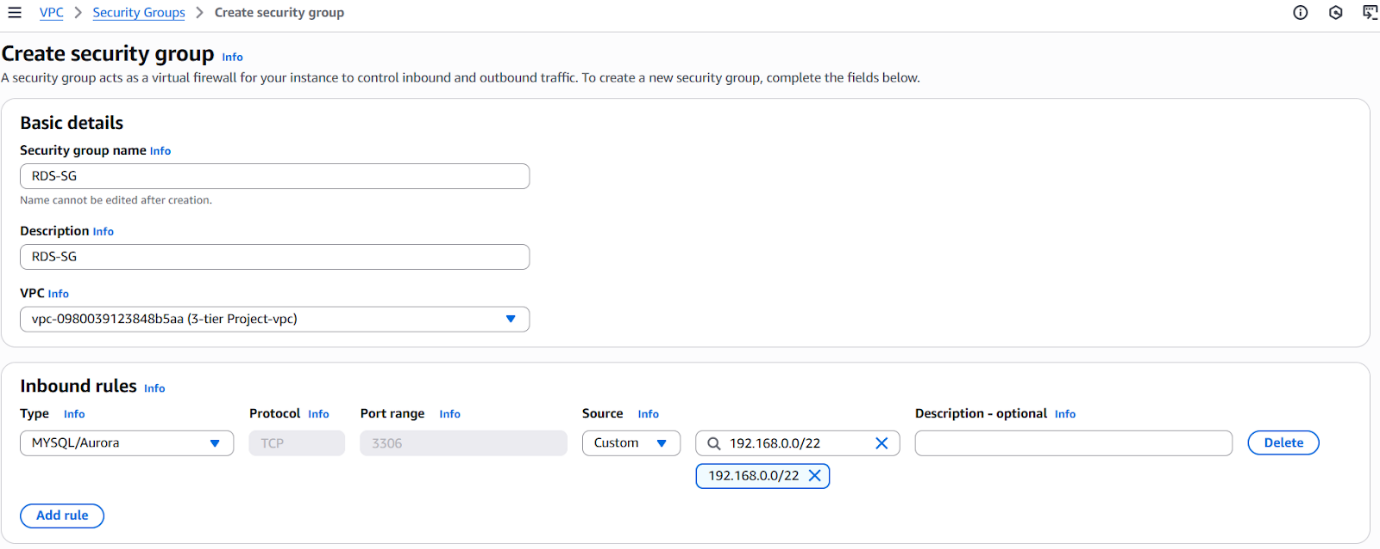
* App Tier Security Group

****

* Internal Load Balancer Security Group

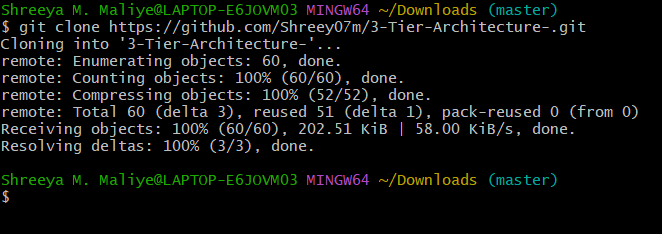
****

* Database Security Group

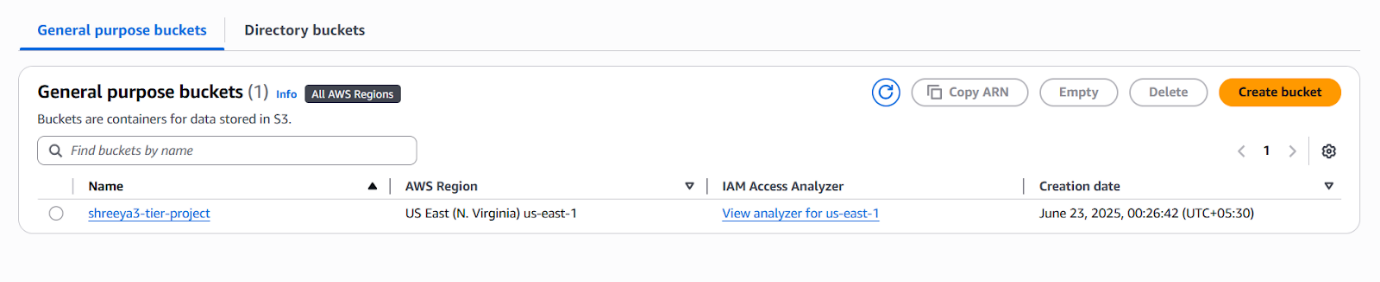
****

### Step 3: Prepare Source Code & Upload to S3

* Clone the GitHub repo locally

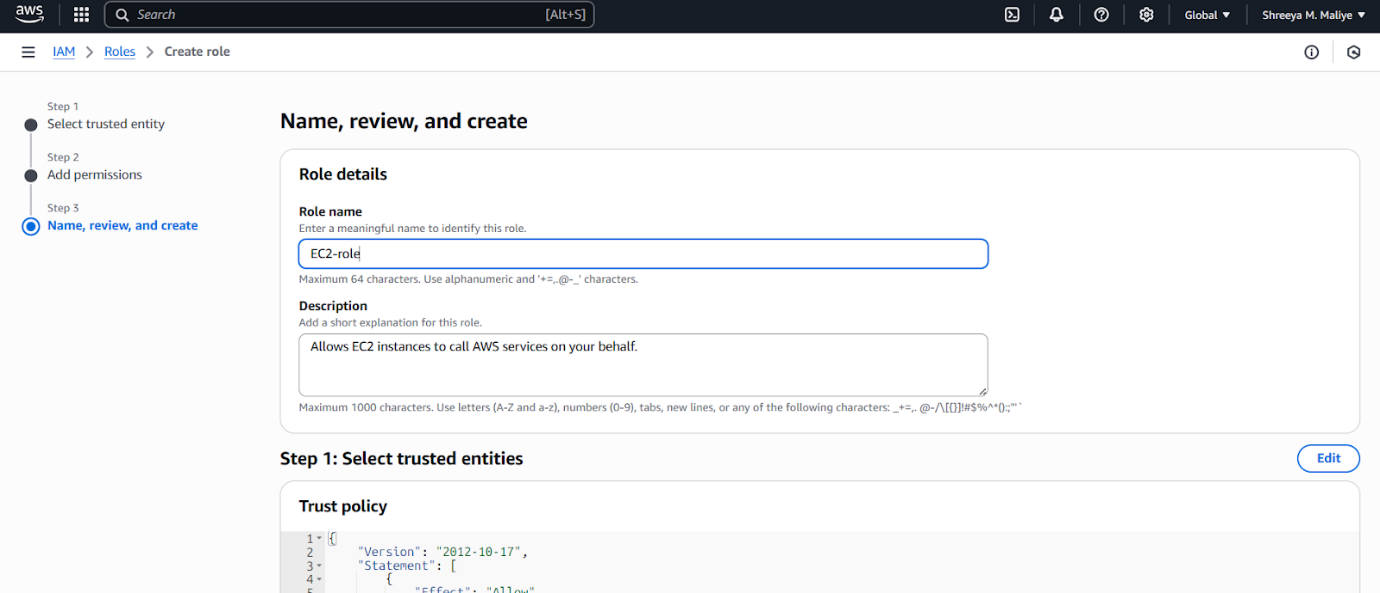
****

* Create S3 bucket and upload application files and nginx.conf

****

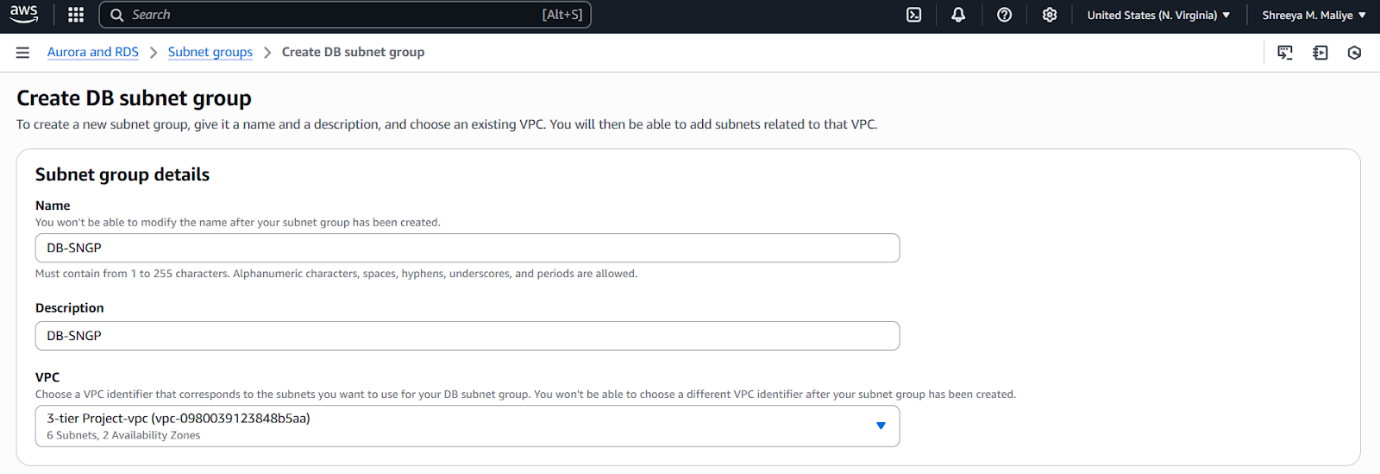
### Step 4: IAM Role

* Create IAM role with EC2 instance to call AWS services
* Attach to EC2 instances (App and Web tiers)

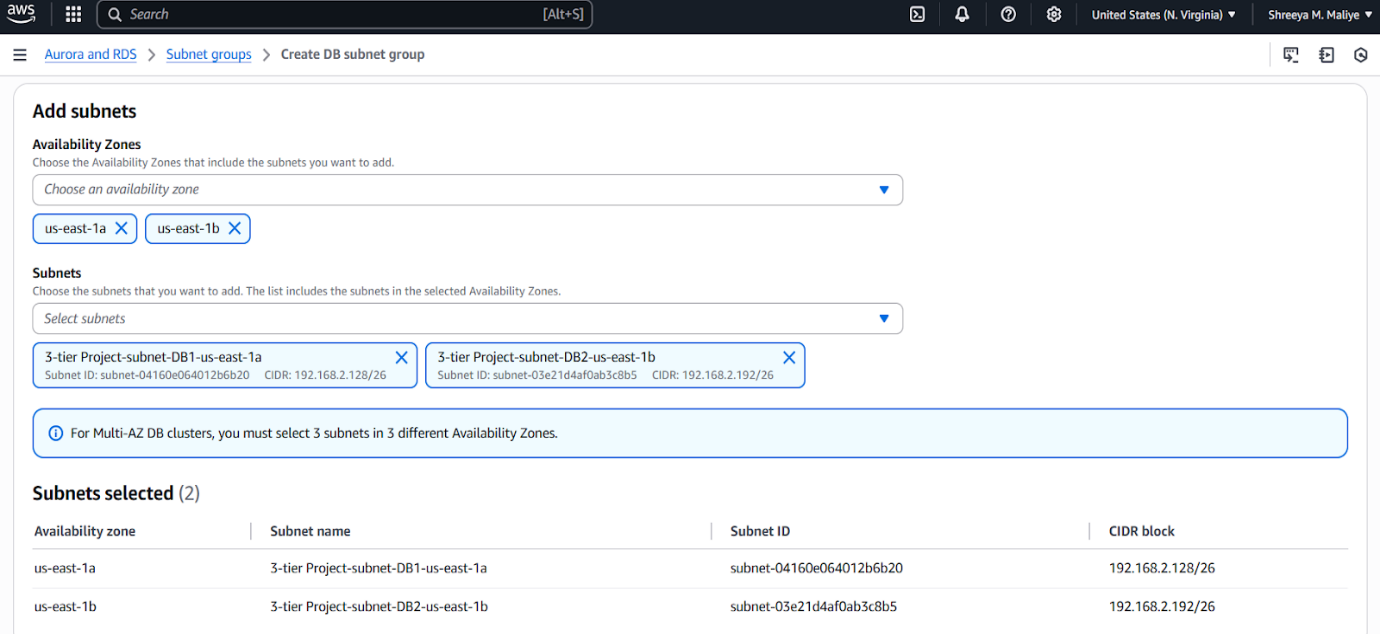
****

### Step 5: Launch RDS (MySQL)

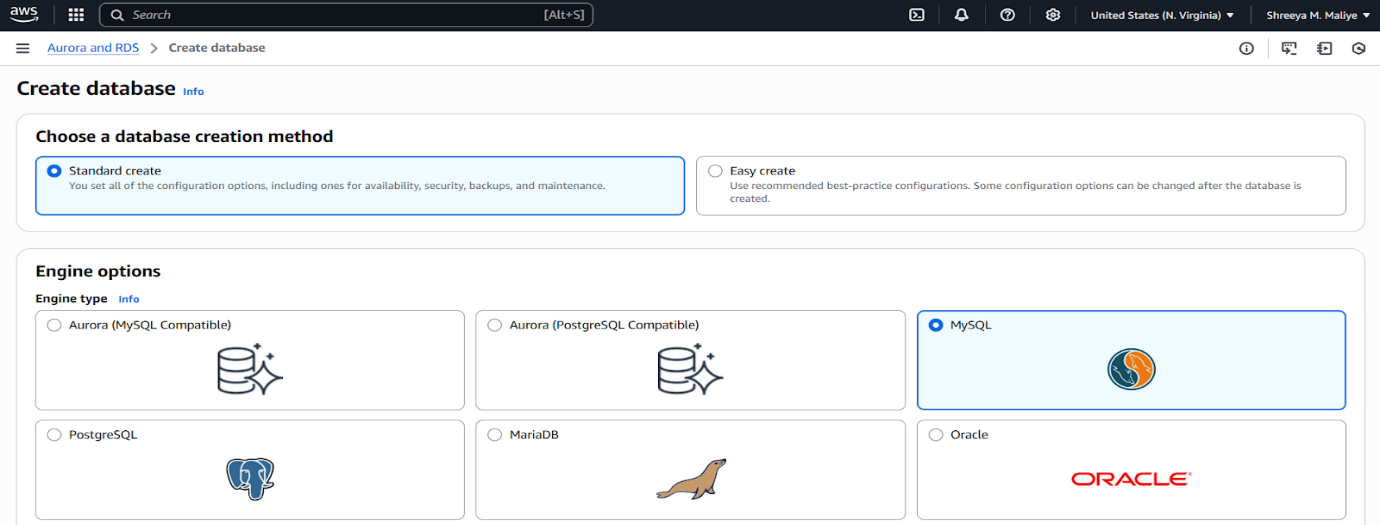
* Create a DB Subnet Group

****

* Use private subnets for RDS

****

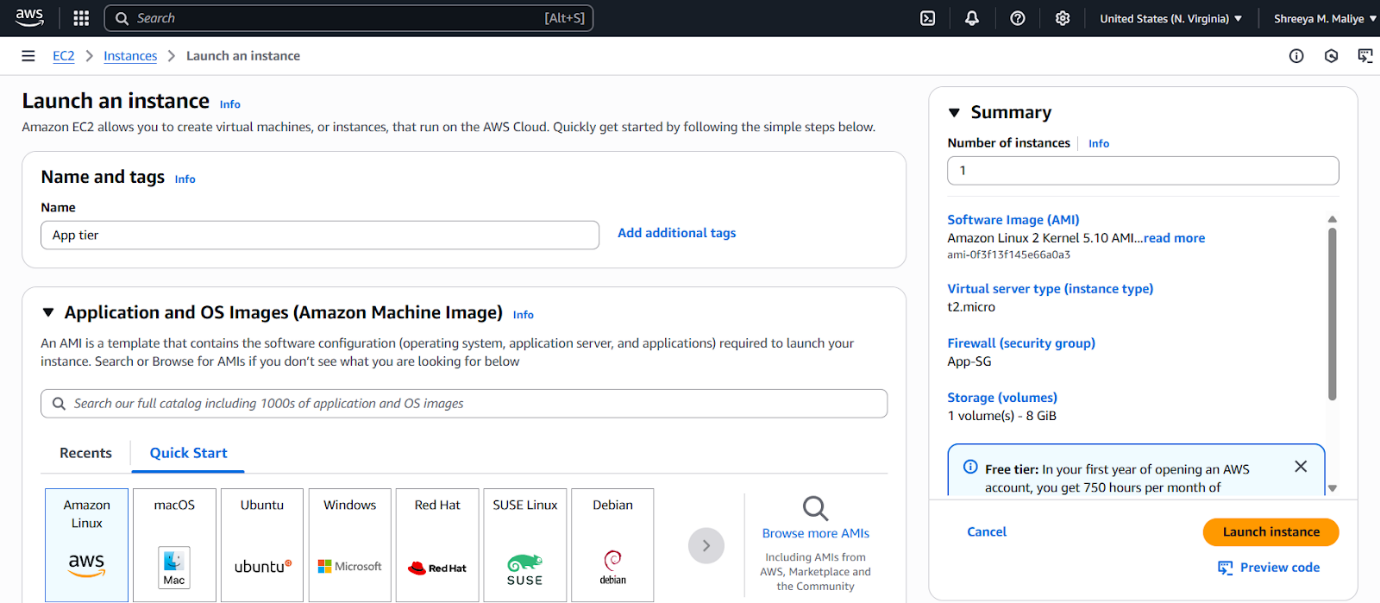
* Setup of RDS Database

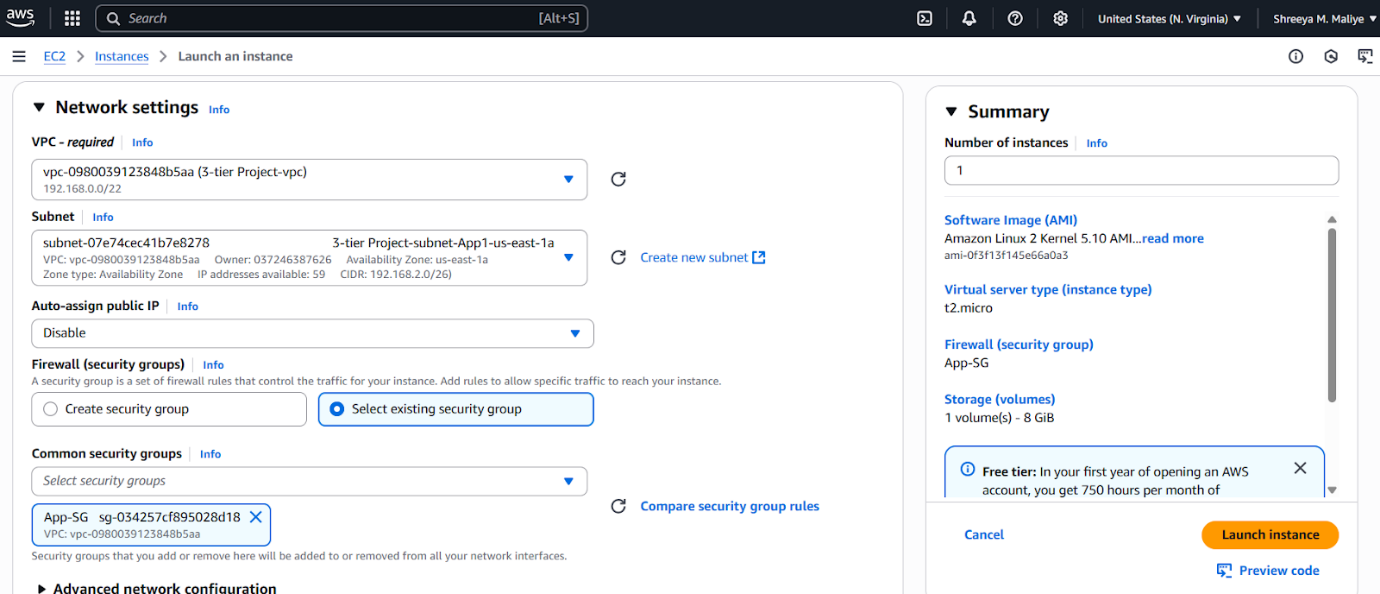
****

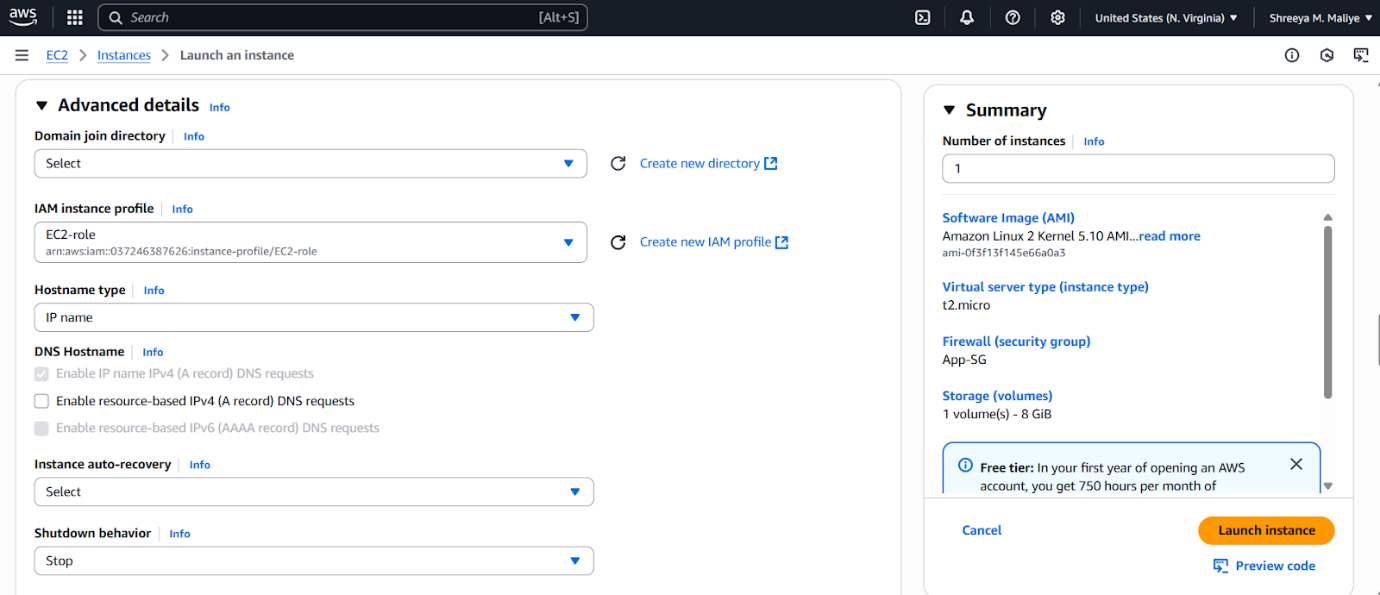
* + DB Name: three-tierdb
  + Master User: admin
  + Password: \*\*\*\*\*\*\*

### Step 6: Launch App Tier EC2 & Connect

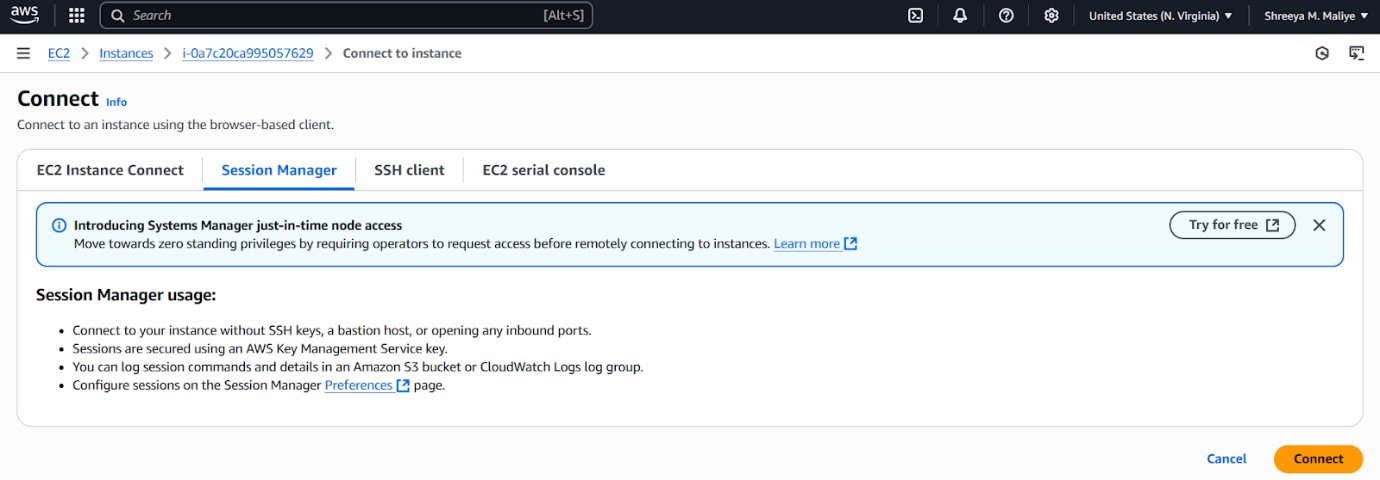
* Launch EC2 instance in private subnet

****

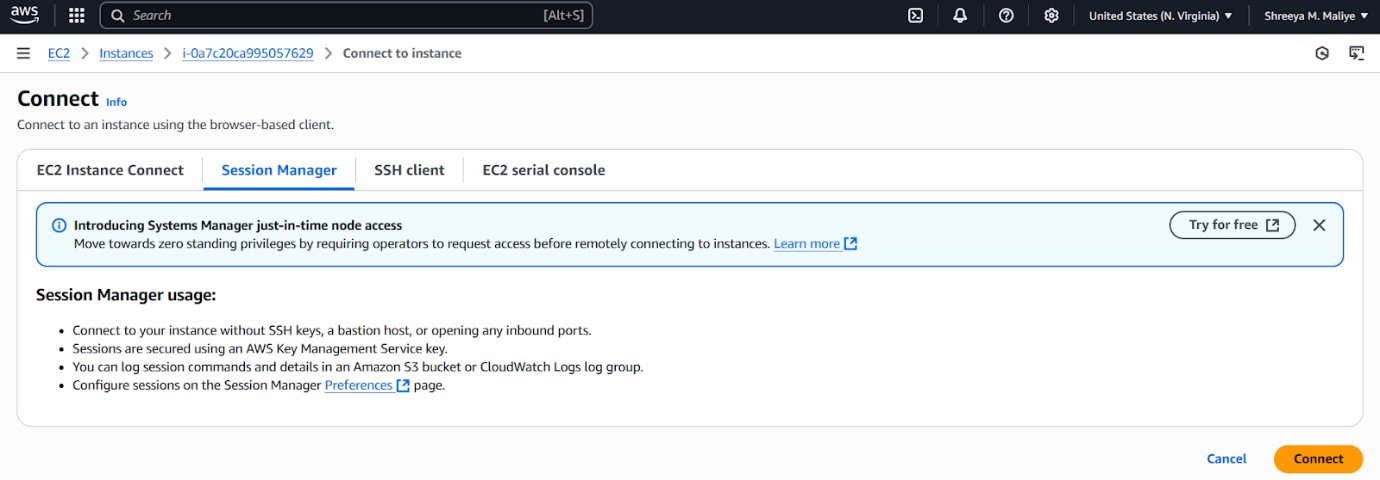
****

****

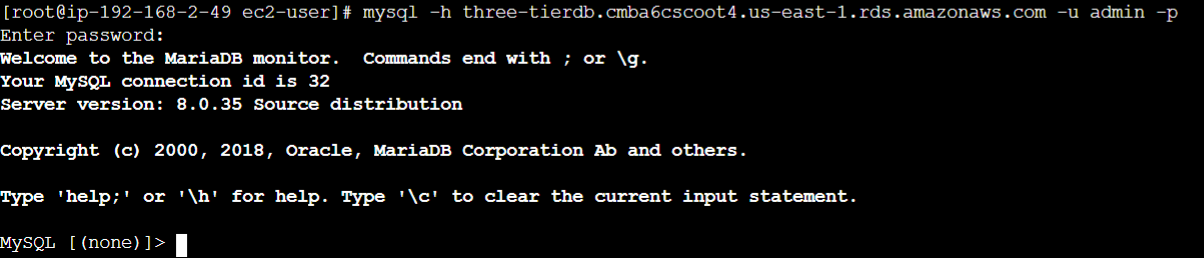
* Connect EC2 using **Session Manager**

****

* Install MySQL client: sudo yum install mysql -y

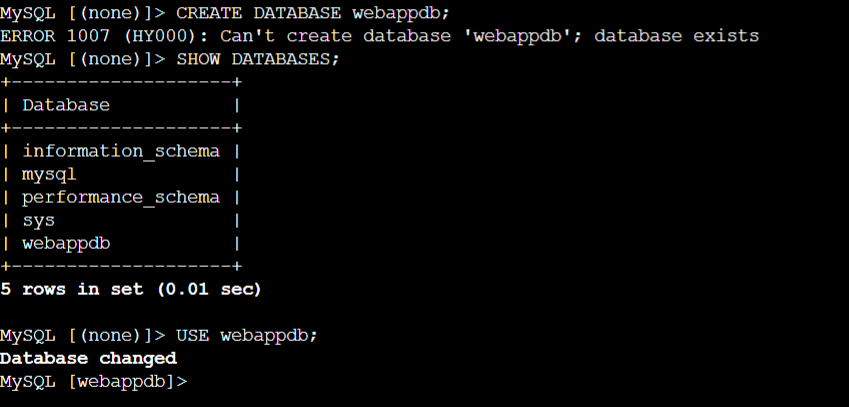
****

* Connect to DB: mysql -h <DB Endpoint> -u admin -p

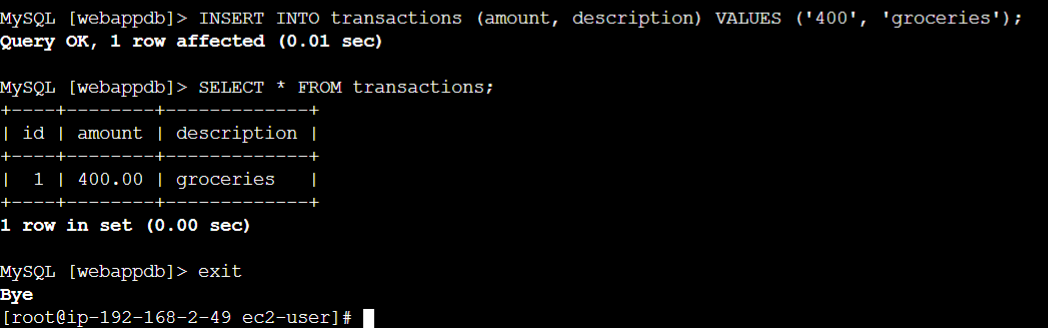
****

* Create Database:

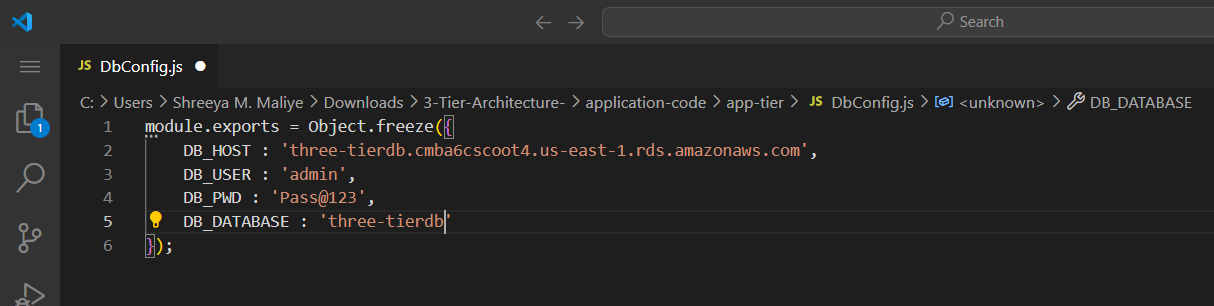
CREATE DATABASE webappdb;  
USE webappdb;  
CREATE TABLE transactions (  
 id INT NOT NULL AUTO\_INCREMENT,  
 amount DECIMAL(10,2),  
 description VARCHAR(100),  
 PRIMARY KEY(id)  
);

****

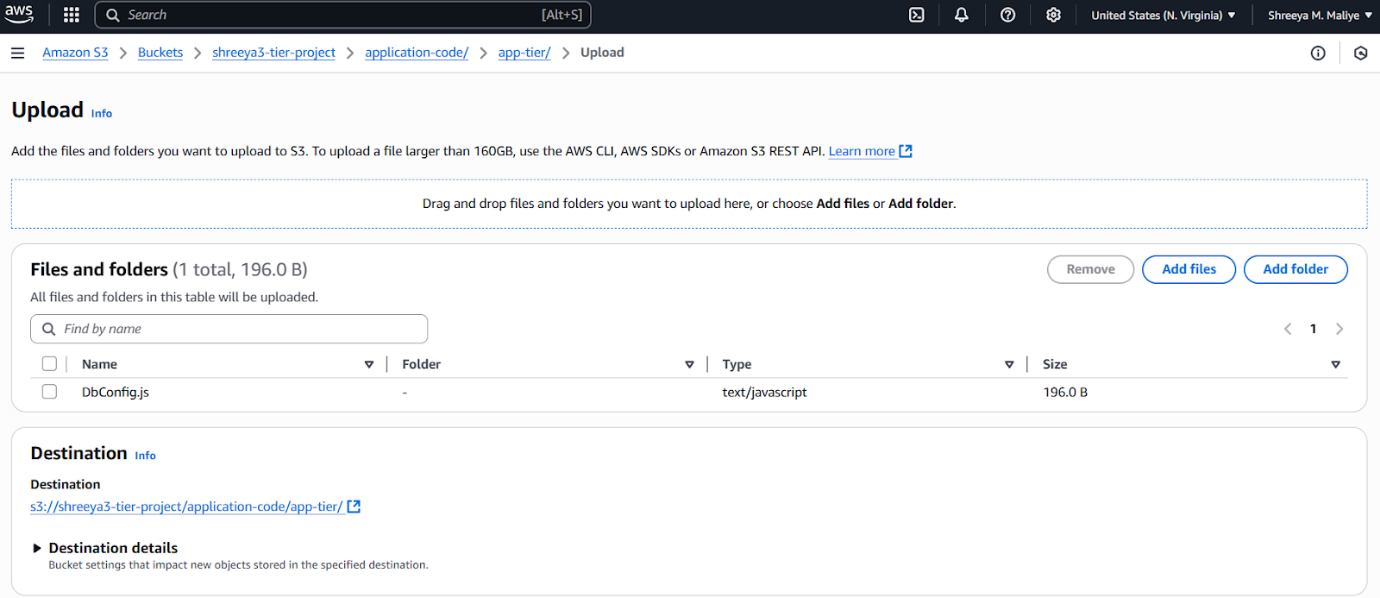
Insert information into the table and verify the entry is created or not

****

Change the [DbConfig.js](http://dbconfig.js)

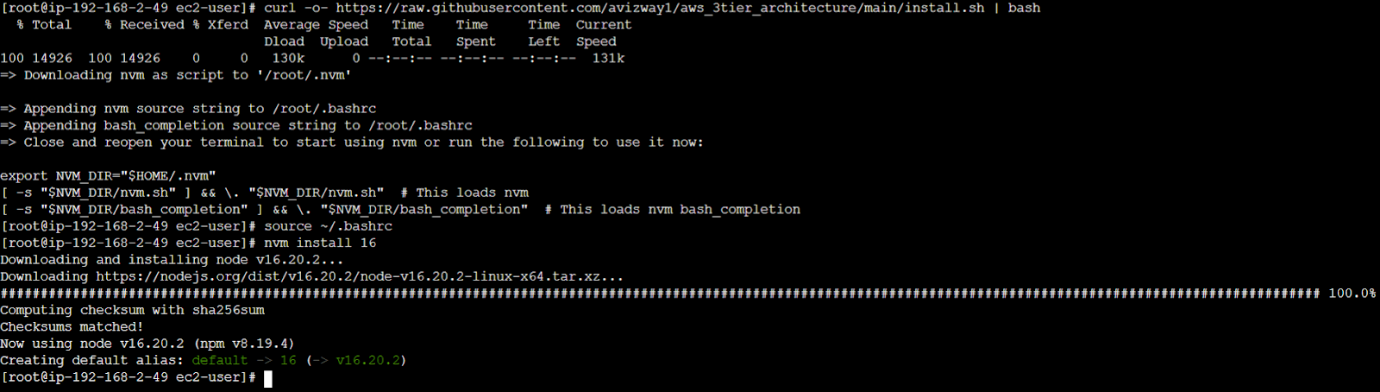
****

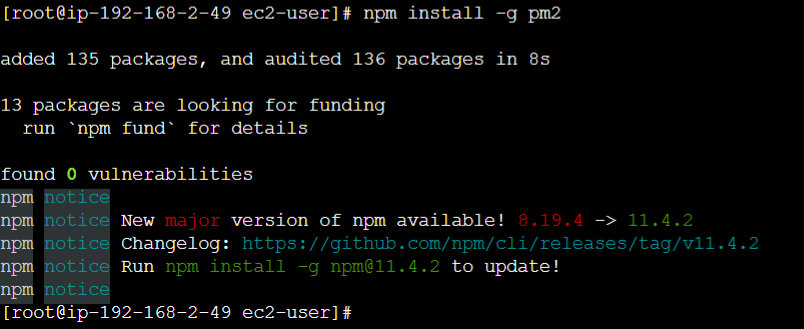
Update the [Dbconfig.js](http://dbconfig.js) file into S3 bucket

****

* Install Node.js and PM2:

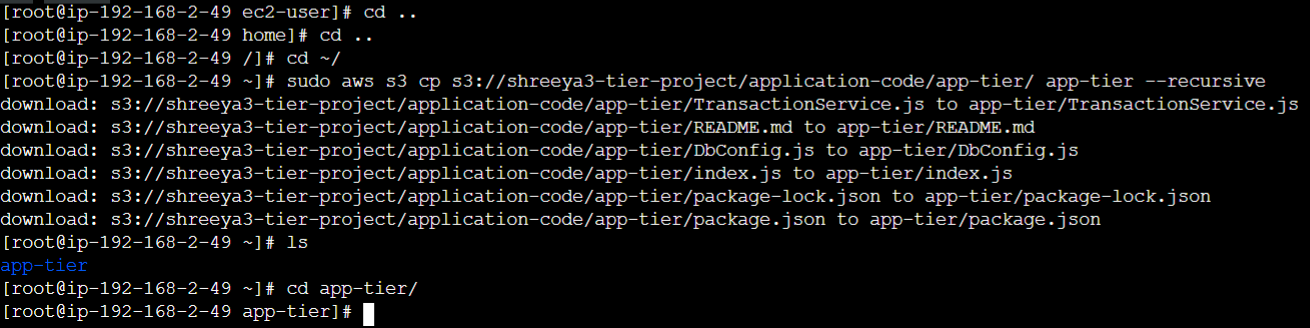
nvm install 16  
npm install -g pm2

****

****

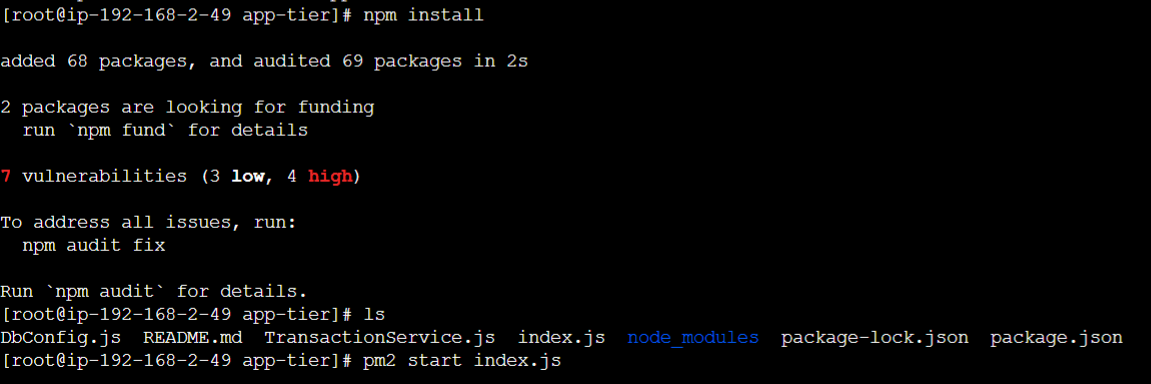
* Copy backend code from S3:

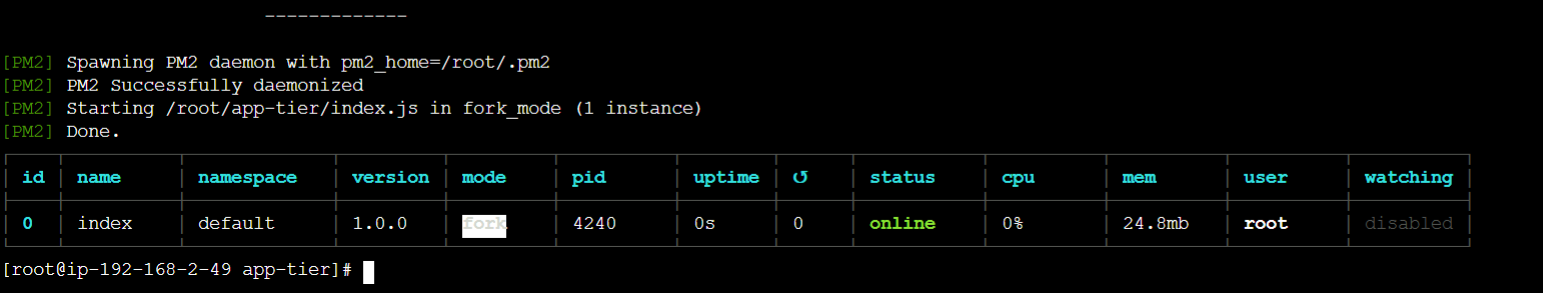
aws s3 cp s3://<bucket-name>/application-code/app-tier/ app-tier –recursive

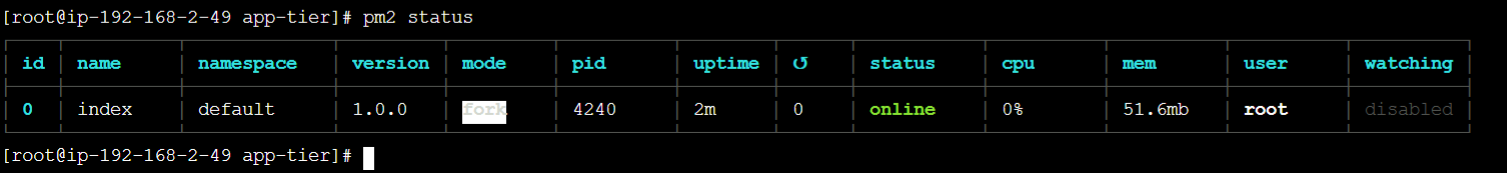
****

* Run app using pm2:

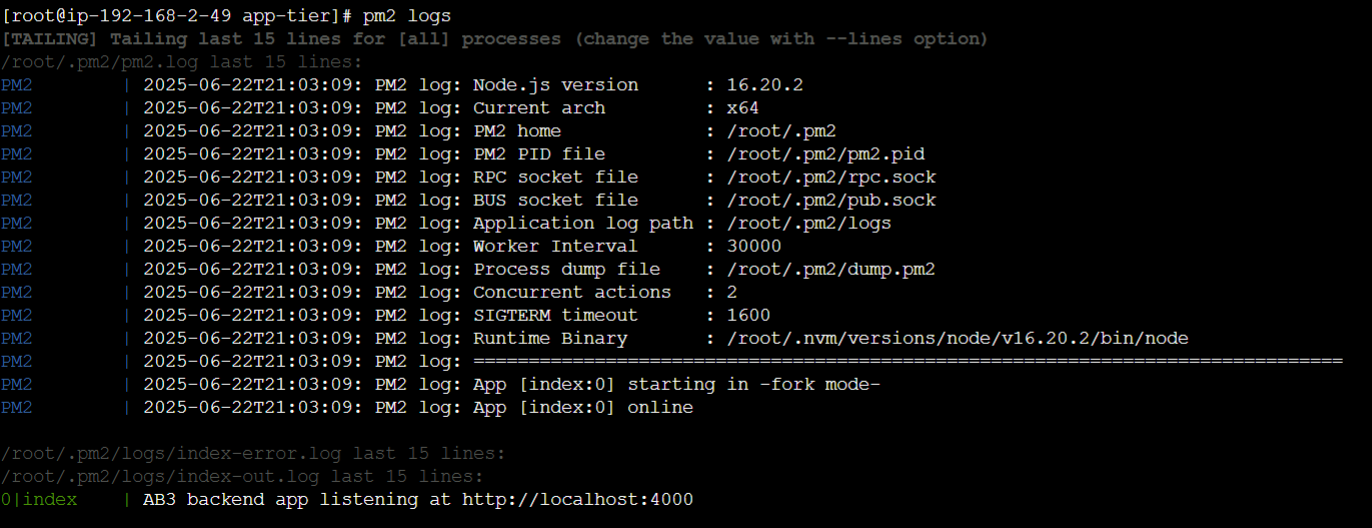
cd app-tier  
npm install  
pm2 start index.js  
pm2 save

****

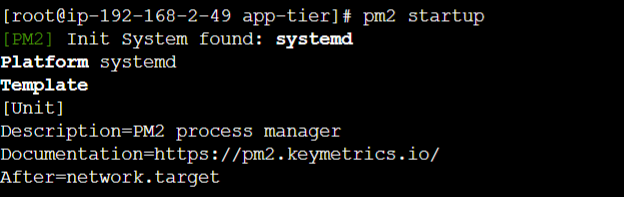
****

To verify: pm2 status

To check the logs: pm2 logs

****

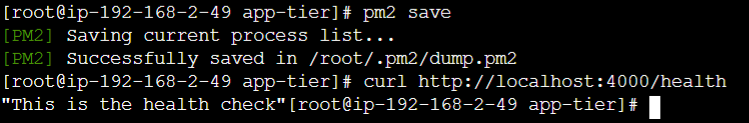
To start the pm2: pm2 startup

****

To save the pm2: pm2 save

Verify that the application is running by executing

curl <http://localhost:4000/health>

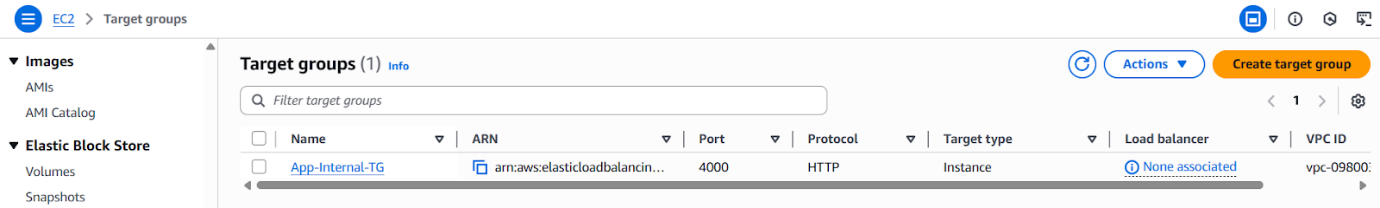
****

### 

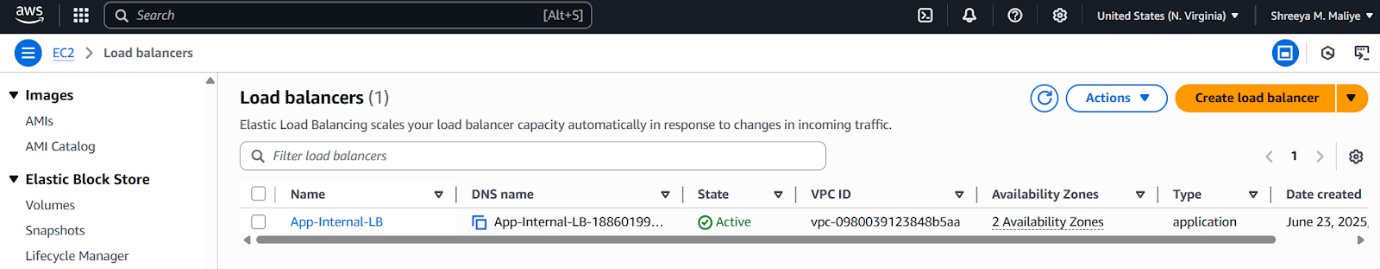
### Step 7: Create Internal Load Balancer

Create one internal load balancer, before creating the load balancer we have to create target Group and attach the target group to load balancer

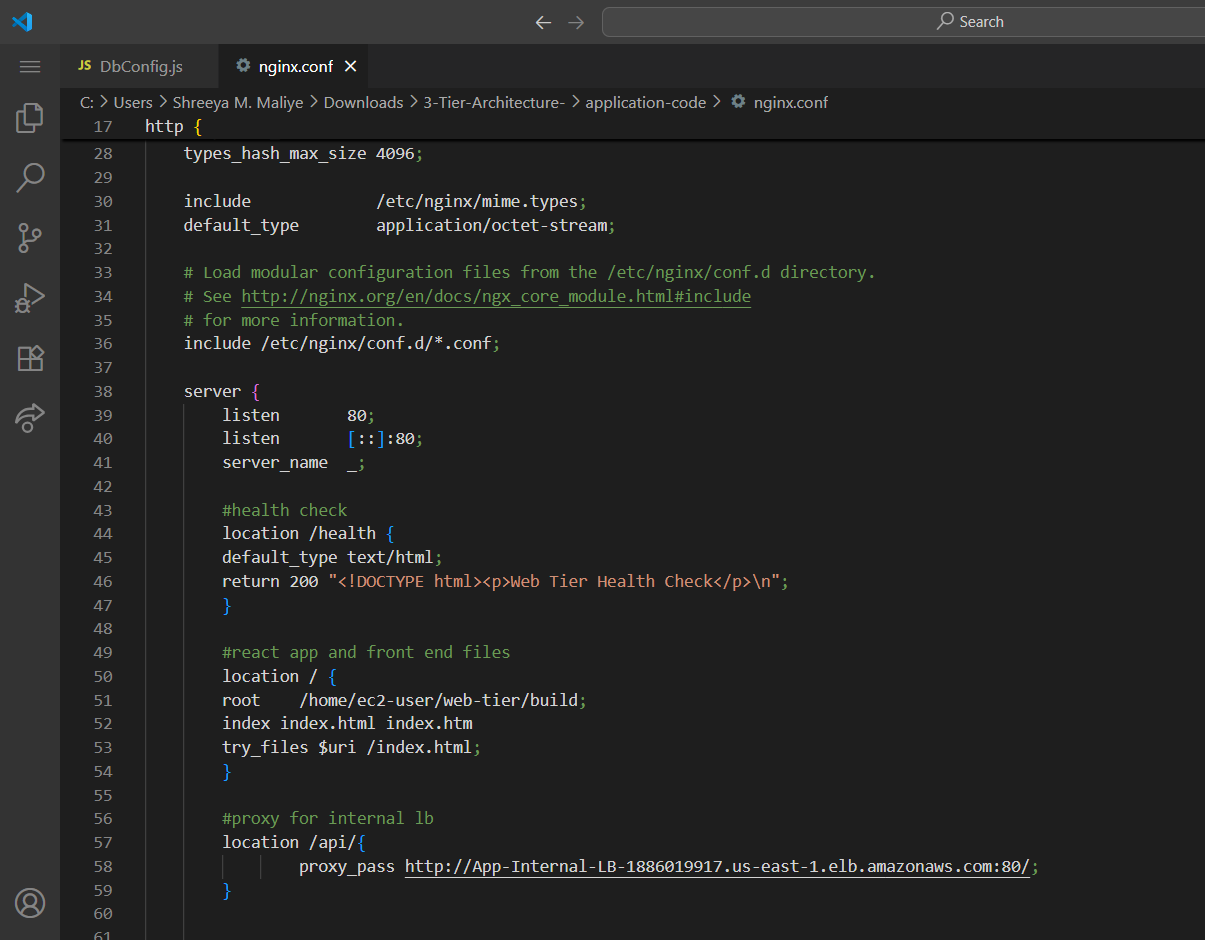
* Create Target Group on port 4000

****

* Attach App Tier EC2 instance
* Create Internal Load Balancer and attach Target Group

****

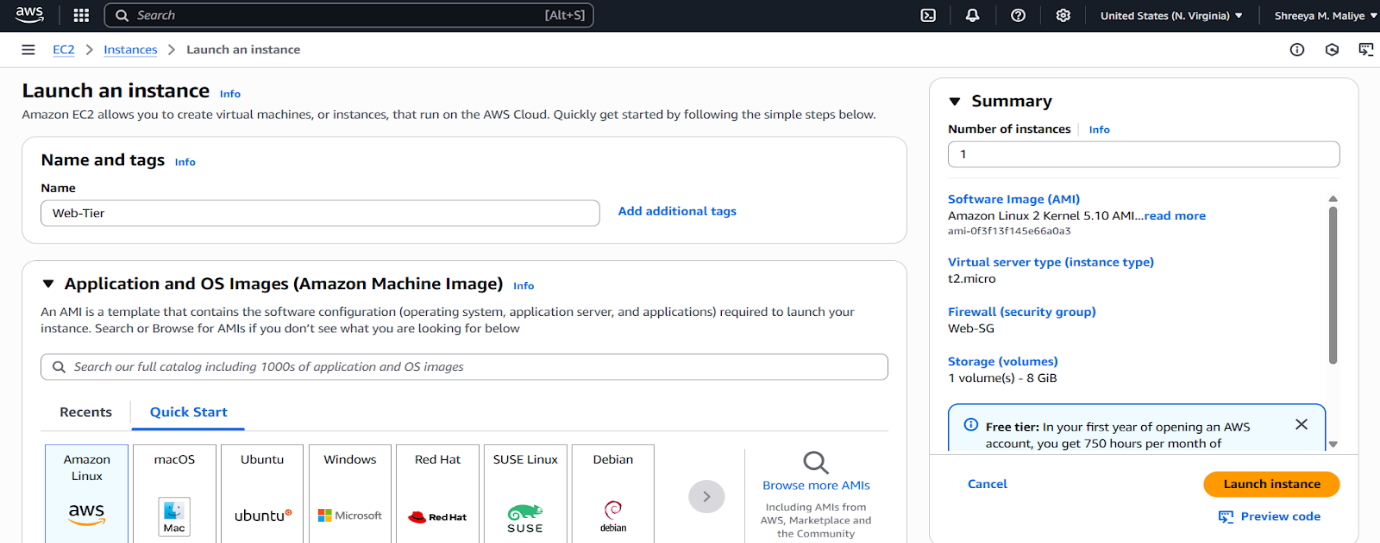
Copy the DNS name of the internal Load balancer and add it into the code of nginx.conf and upload the updated file into the S3 bucket

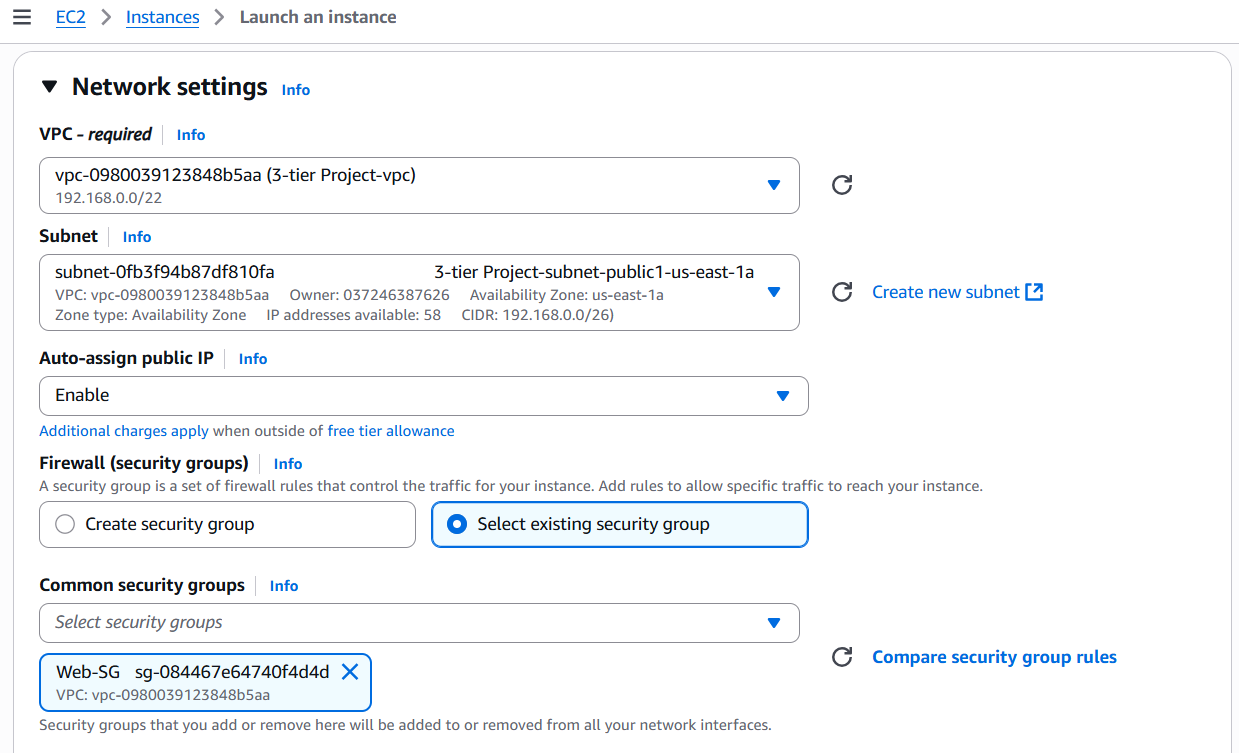
****

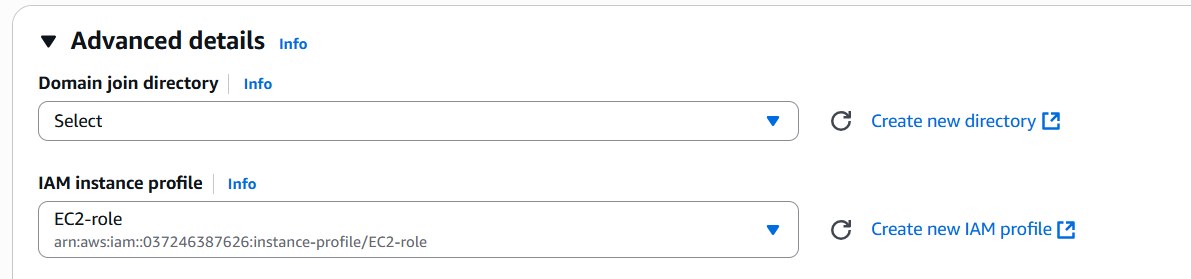
Successfully created the App tier resources !!!

### Step 8: Web Tier EC2 Setup

* Launch EC2 in public subnet

****

****

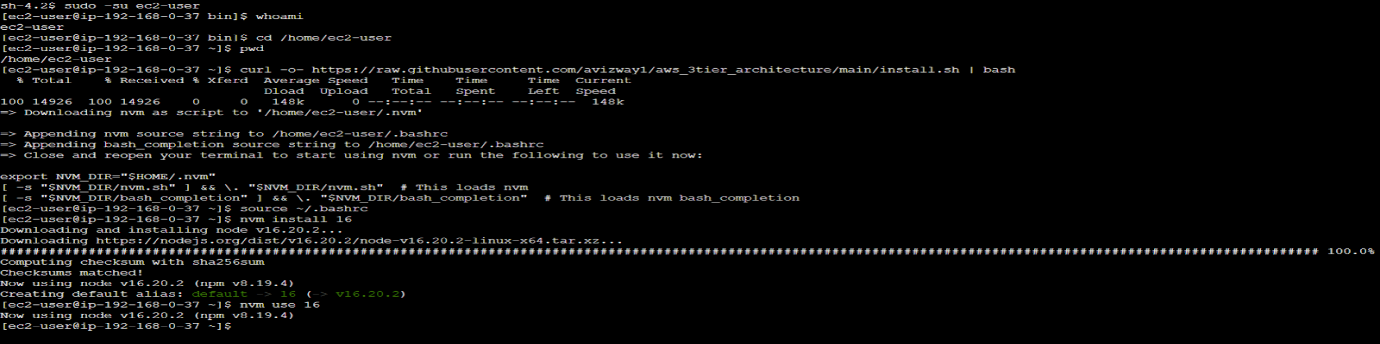
****

* Connect via EC2 Connect or Session Manager

sudo -su ec2-user (To work as an ec2-user)

cd /home/ec2-user

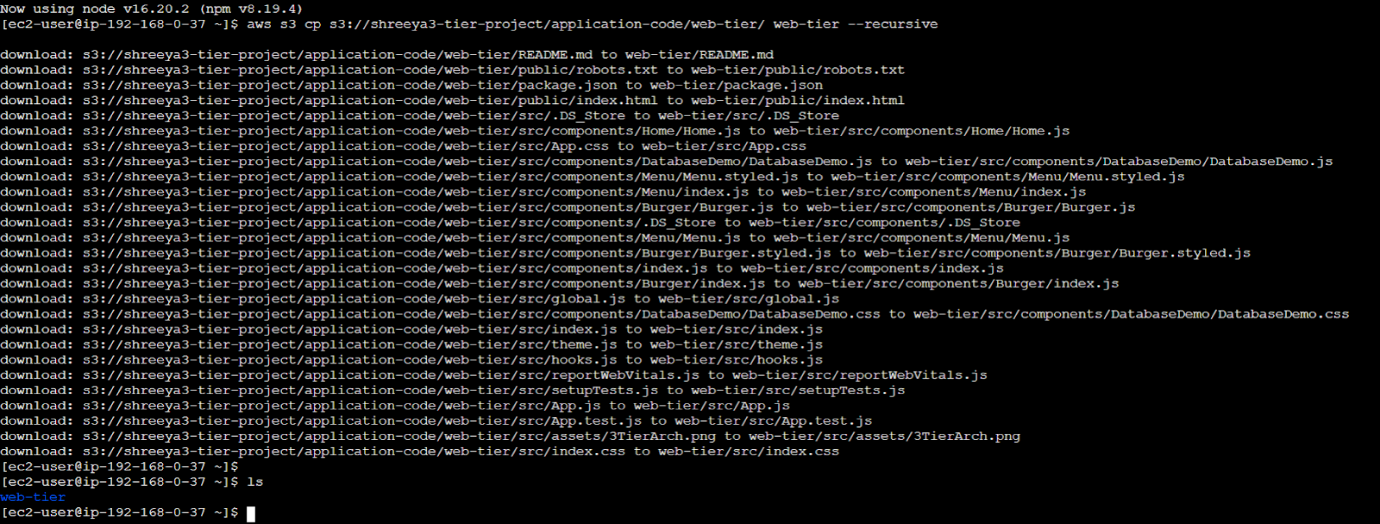
* Install Node.js & nginx

****

To install nginx server: sudo amazon-linux-extras install nginx1 -y

* Pull frontend code:

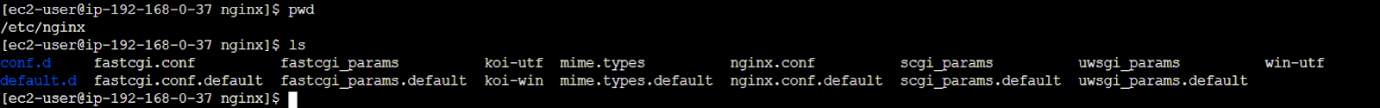
aws s3 cp s3://<bucket-name>/application-code/web-tier/ web-tier --recursive  
cd web-tier  
npm install  
npm run build

****

* Configure nginx:

cd /etc/nginx

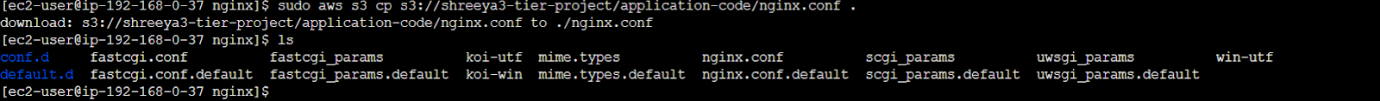
ls

****

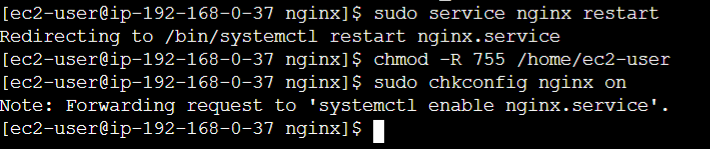
**Remove the nginx file:** sudo rm nginx.conf

**Upload the nginx file from the S3 bucket**

sudo aws s3 cp s3://<bucket-name>/application-code/nginx.conf .

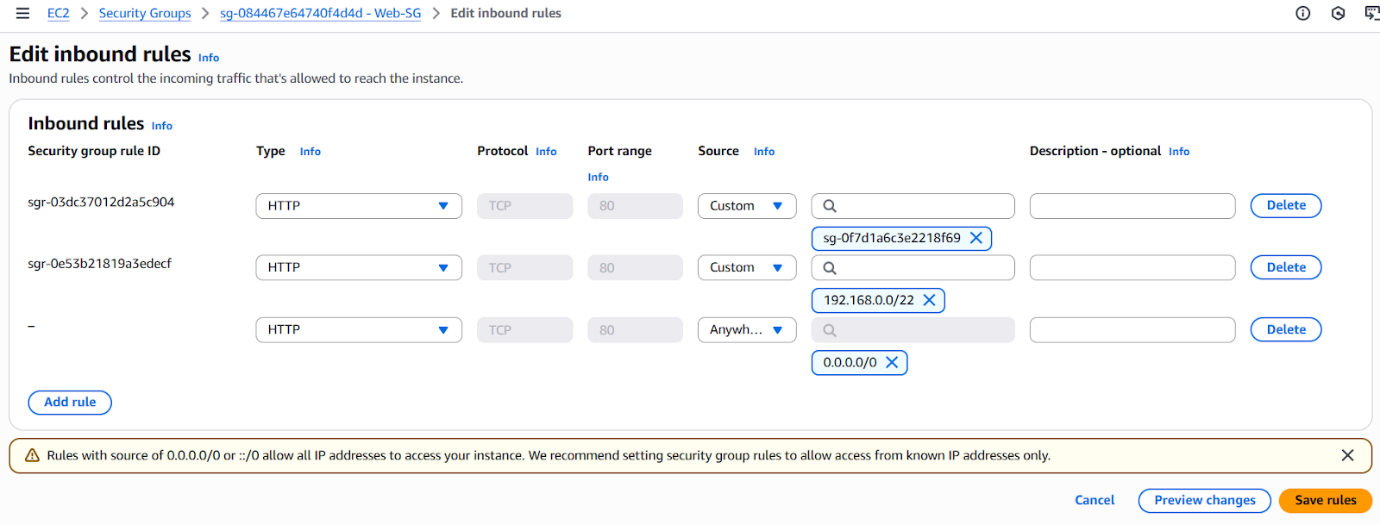
****

Restart the nginx server, give the permission to it a and turn on the nginx server  
sudo service nginx restart  
chmod -R 755 /home/ec2-user  
sudo chkconfig nginx on



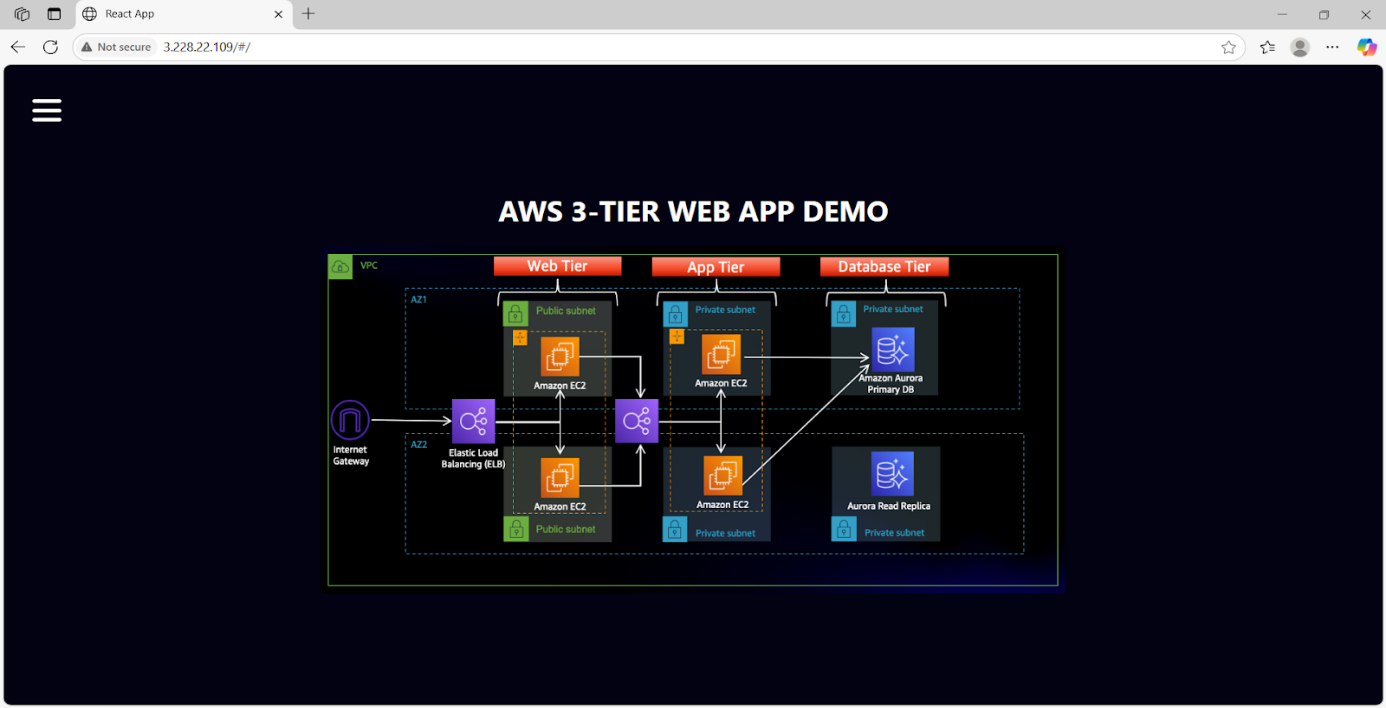
To check the output of the App, we can check using the Web-Tier-Instance public IP.

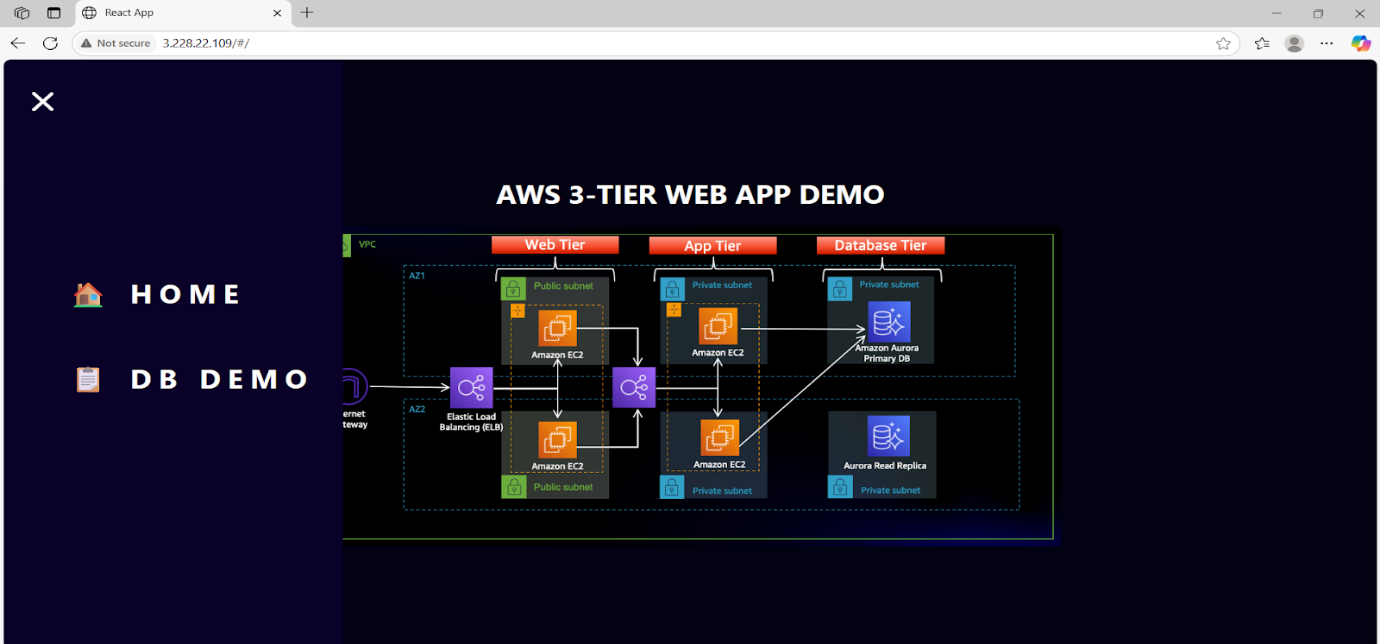
Edit the inbound rule of the web-tier instance

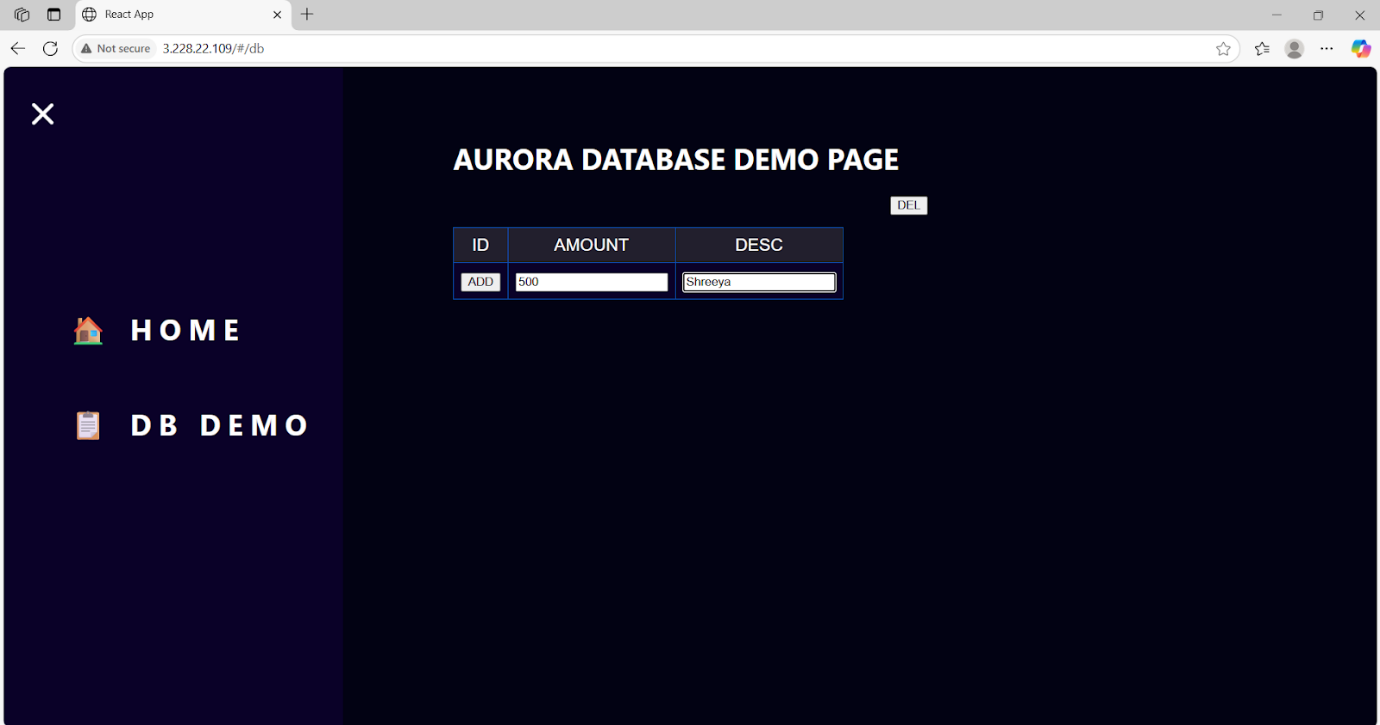
****

### Step 9: Final Testing

* Access frontend using Web Tier’s public IP
* Try inserting data into the app

****

****

****

## 5. Security & Best Practices

* RDS in private subnet
* IAM role instead of access keys
* Security groups scoped to specific ports and IPs
* No hardcoded credentials in code

## 6. Challenges Faced

* Internal communication between tiers
* RDS subnet misconfiguration initially
* PM2 logs not retained until pm2 save was used

## 7. Conclusion

This project showcases the deployment of a modular, secure, and scalable 3-tier architecture on AWS using real-world services. Each component is isolated and independently deployable, ensuring maintainability and security.