**Deploying a Multi-Tier Application on AWS with EC2 and RDS**

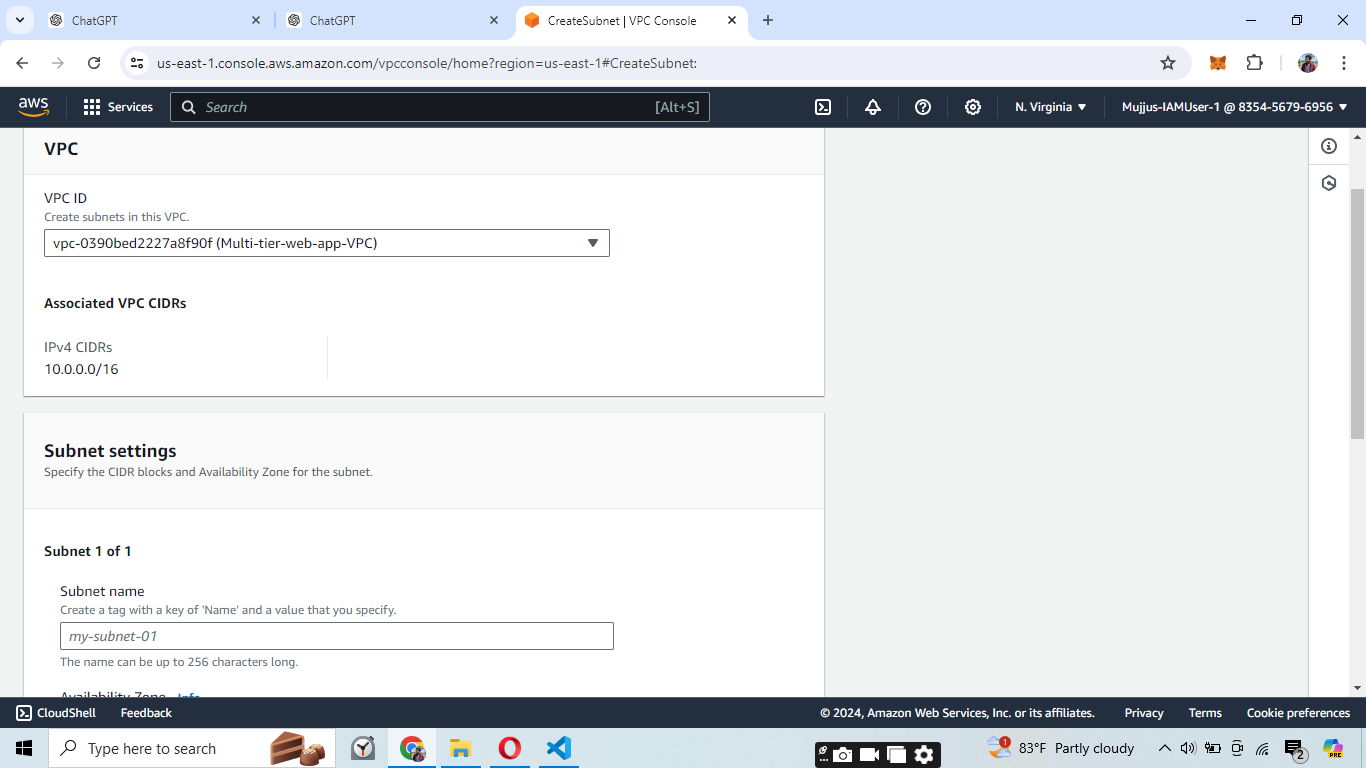
By **Mohammad Mujahid**

In this tutorial, we will walk through the process of deploying a multi-tier application on AWS using EC2 instances for the frontend and backend. This comprehensive guide covers everything from initial setup to security configurations.

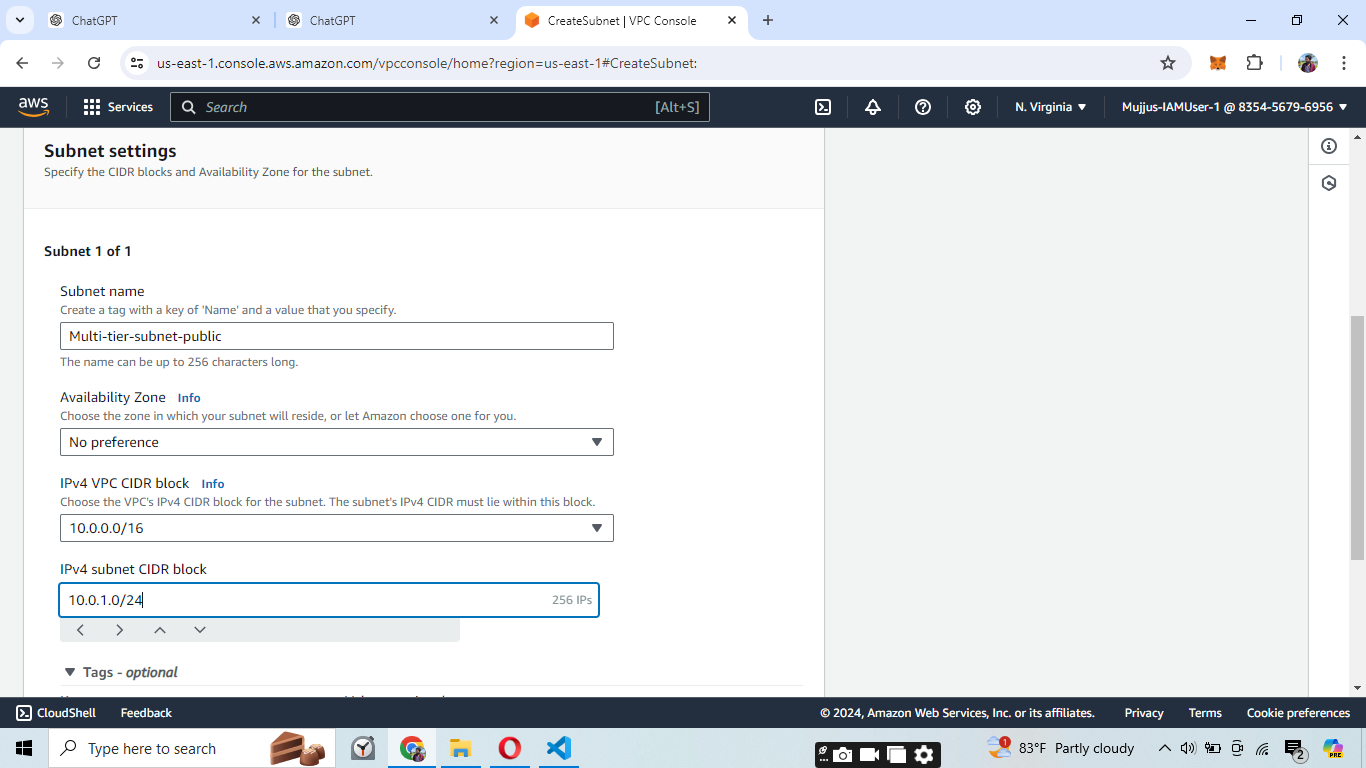
1, Our Project starts by creating a VPC(virtual-private-network) which is our own private network in cloud.



Enter the name and IPv4 CIDR which assigns IP addresses.

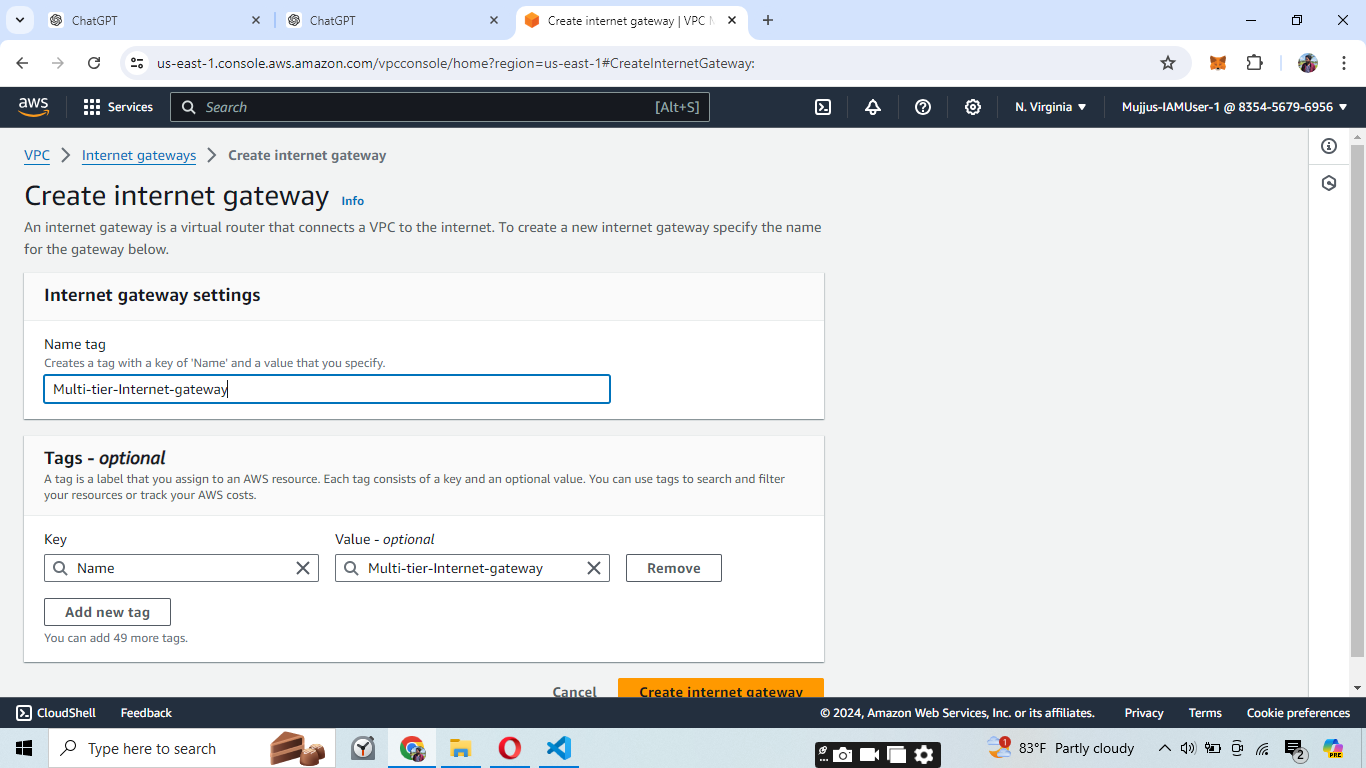


Create a subnet for our VPC which divides the portion of the IP addresses.



And include subnet CIDR block.

**Step 3: Create an Internet Gateway**

****

An Internet Gateway is needed to allow internet access to resources in our public subnet.

In the VPC dashboard, click on "Internet Gateways" and then "Create Internet Gateway."

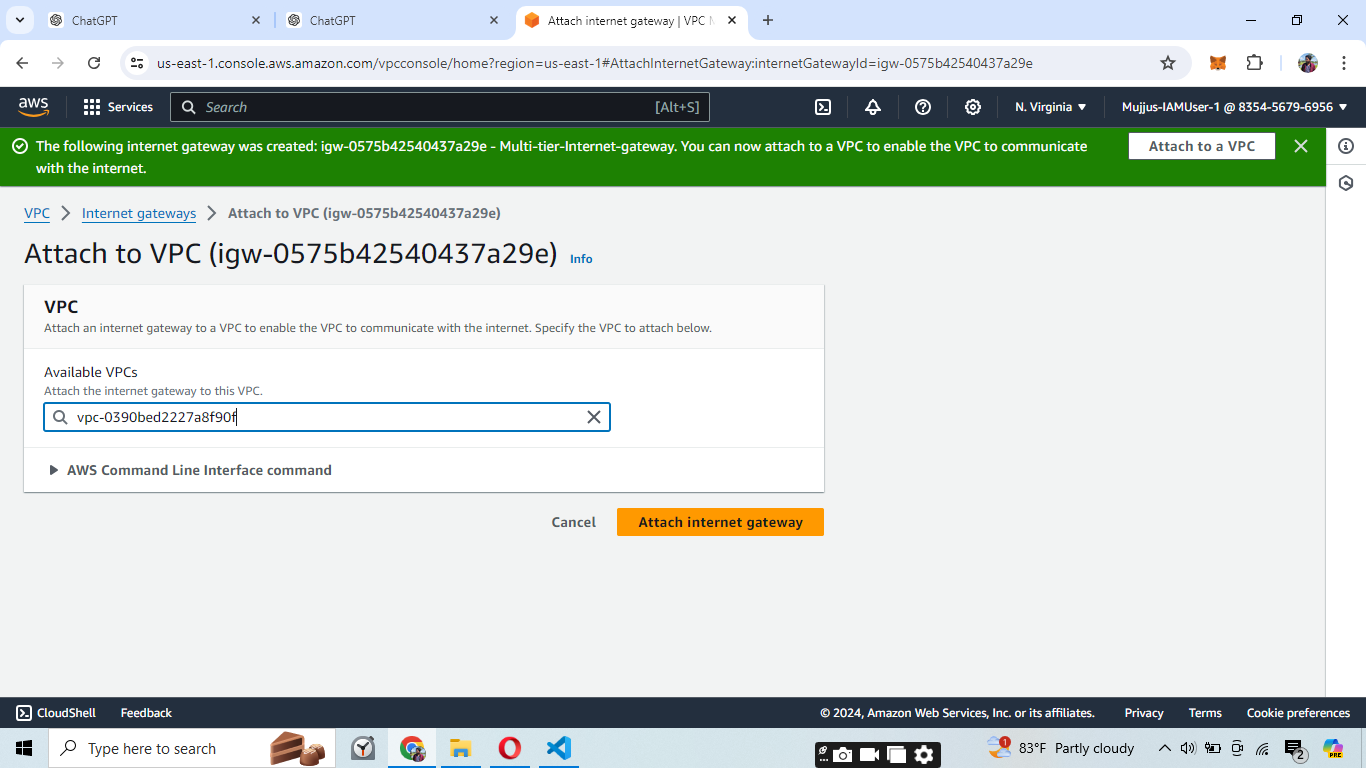
Name your internet gateway (e.g., "Multi-tier-Internet-gateway").

Click "Create Internet Gateway."

Once created, attach the internet gateway to your VPC.

### Step 4: Configure Route Tables

Now we need to configure route tables to manage traffic within our VPC.

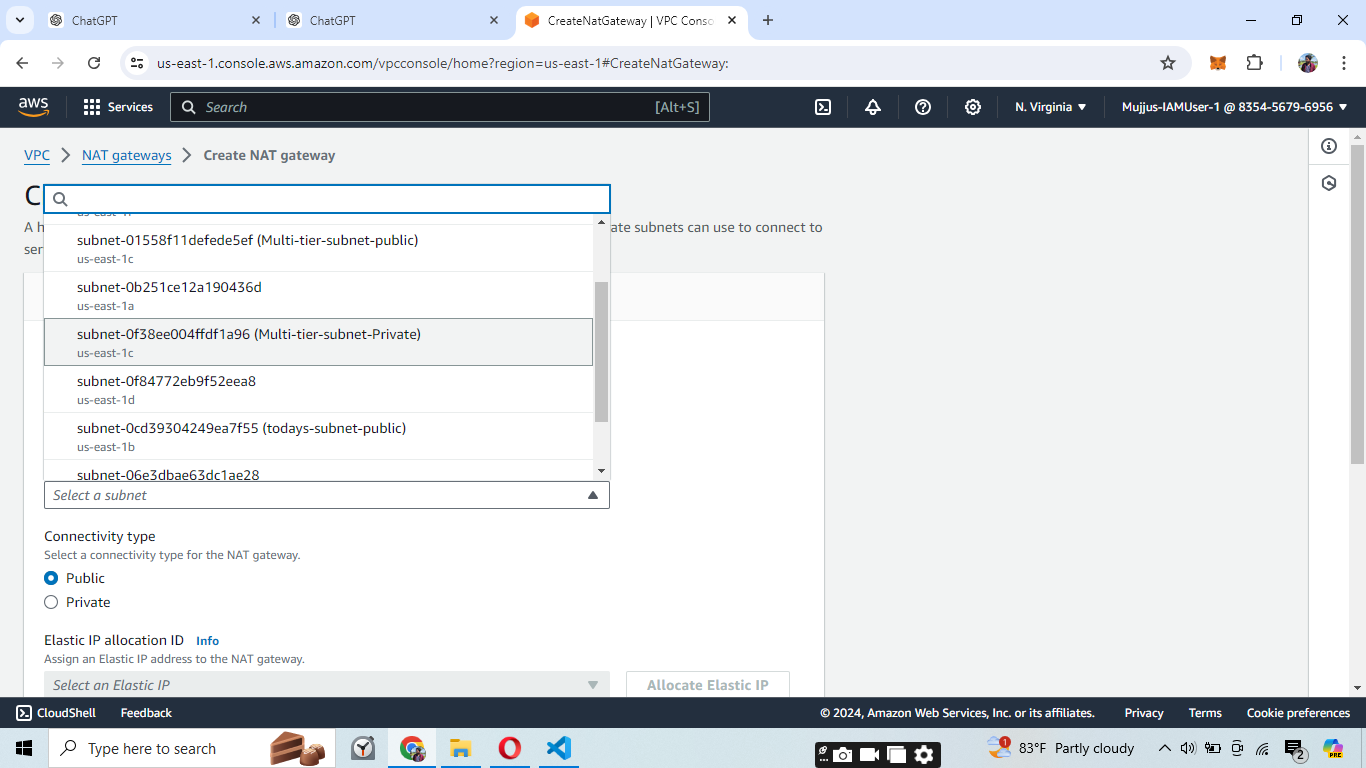


 In the VPC dashboard, click on "Route Tables" and then "Create Route Table."

 Name your route table (e.g., "Multi-tier-route-public-subnet").

 Select your VPC.

 Click "Create Route Table."

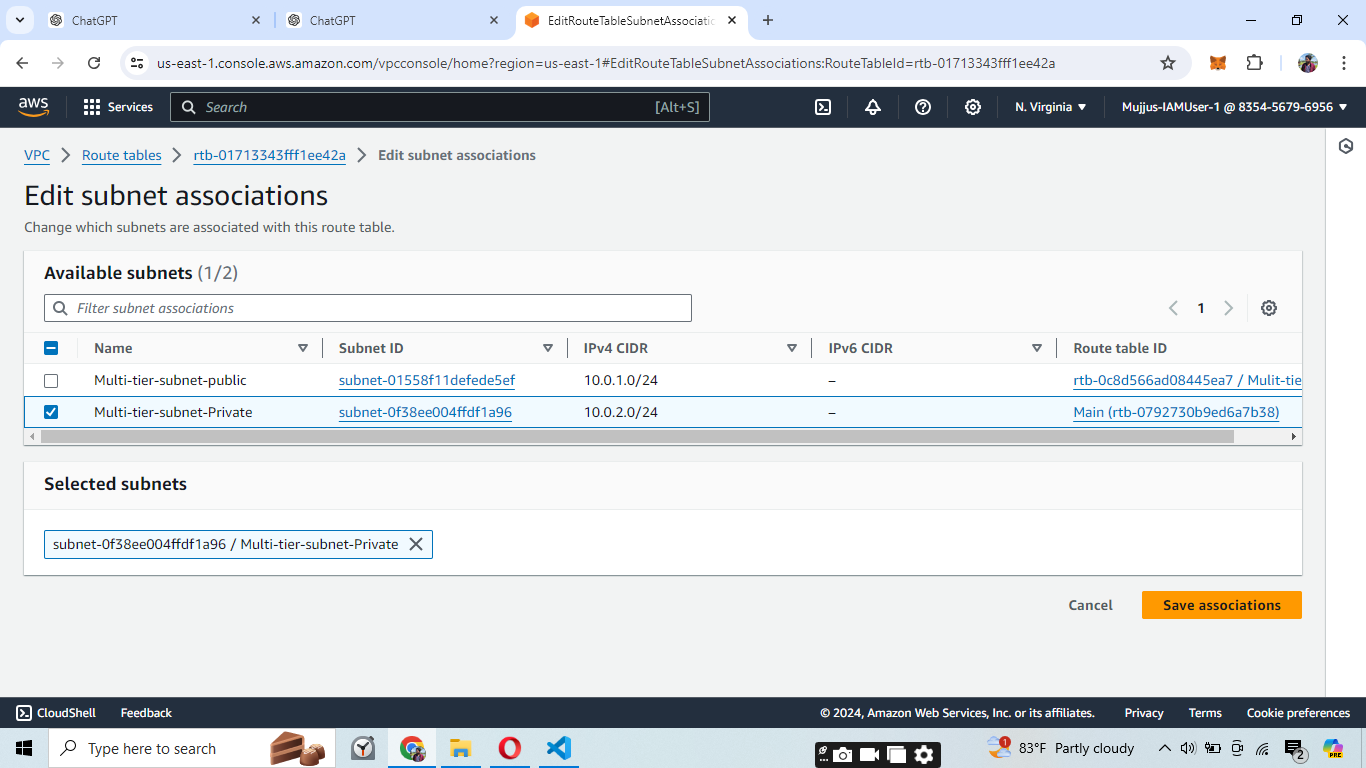


 Under the route table, go to "Subnet associations."

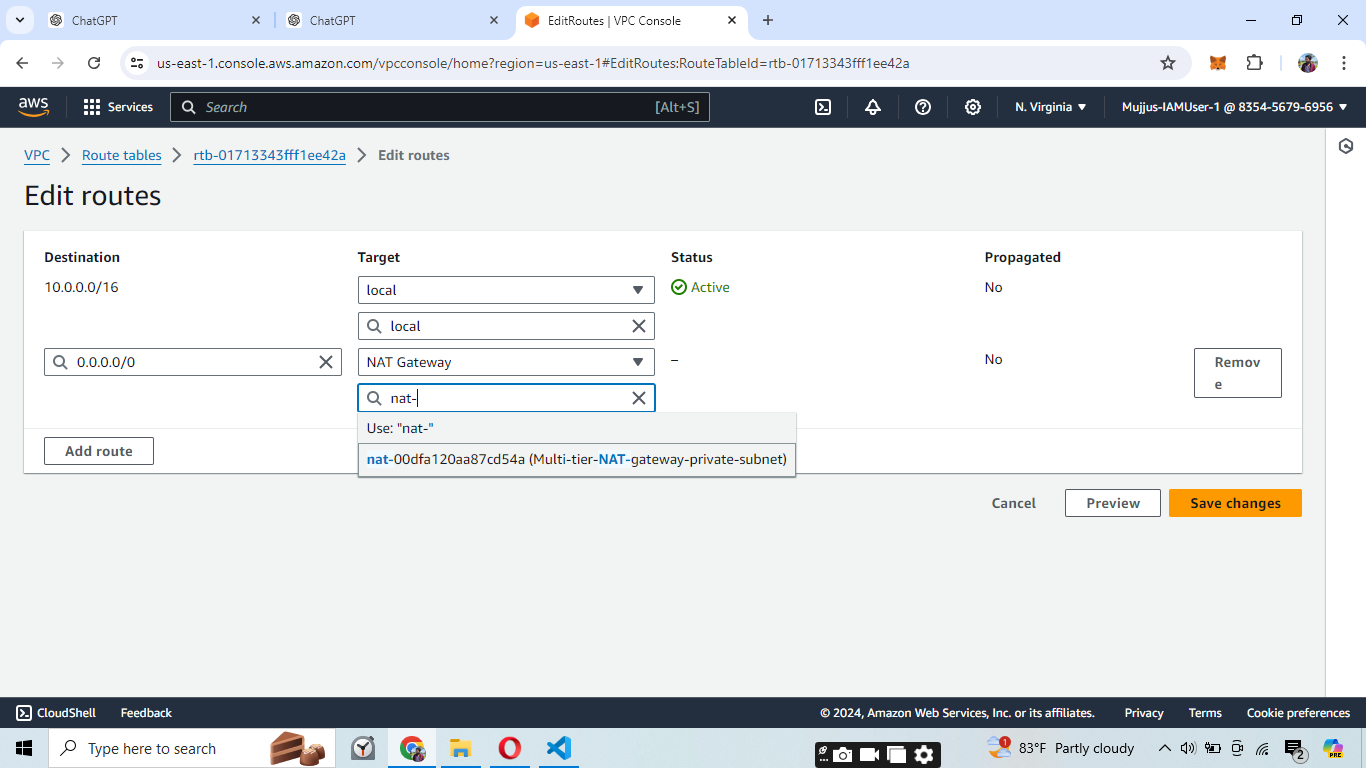
 Click "Edit subnet associations."

 Select the public subnet and click "Save."

### Step 6: Associate Route Tables with Subnets

We need to associate our route tables with the respective subnets to ensure proper routing of traffic.

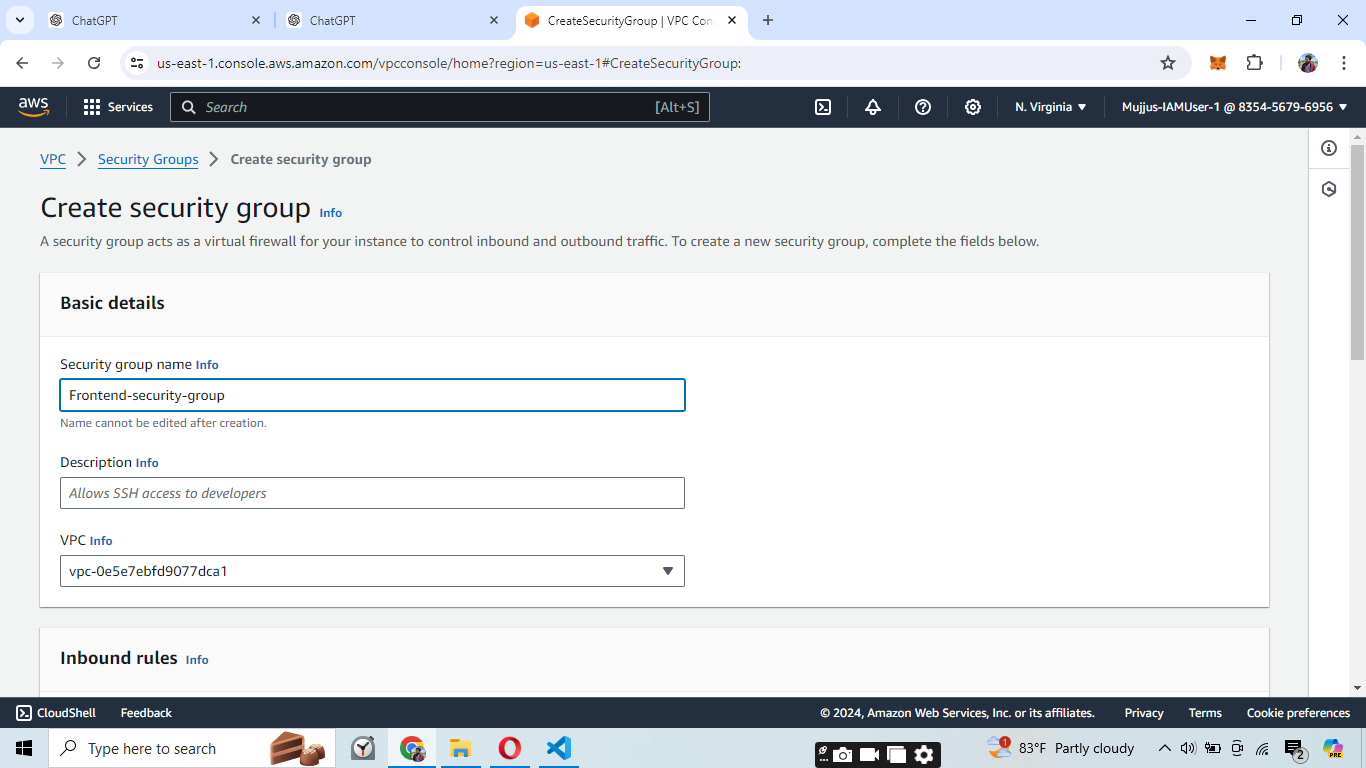
1. In the VPC dashboard, click on "Route Tables" and select the route table for the private subnet.
2. Under "Subnet associations," click "Edit subnet associations."
3. Select the private subnet (e.g., "Multi-tier-subnet-Private") and click "Save."



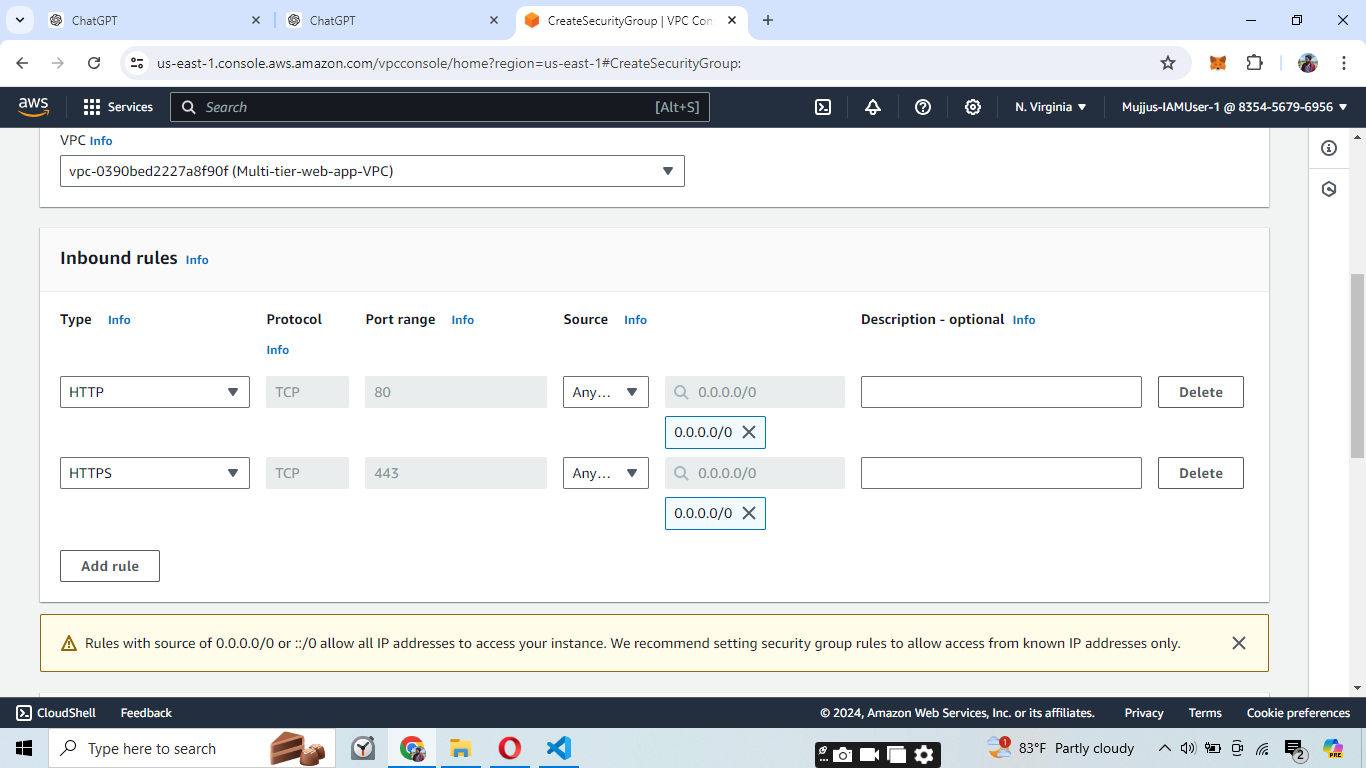
1. With the route table for the private subnet selected, go to "Routes" and click "Edit routes."
2. Add a route with destination 0.0.0.0/0 and target as the NAT Gateway created earlier.
3. Click "Save changes."

**Step 7: Create Security Groups**

Security groups act as virtual firewalls for your instances to control inbound and outbound traffic.



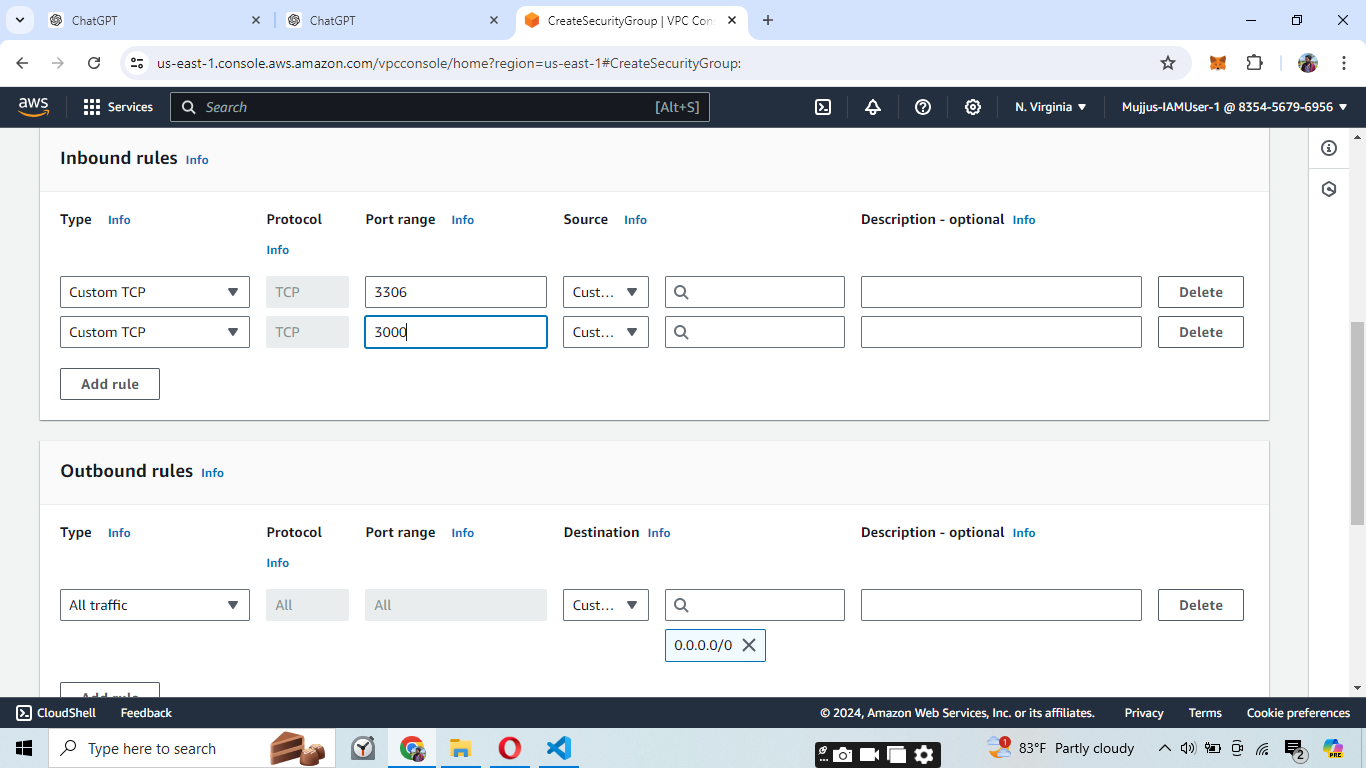
1. In the VPC dashboard, click on "Security Groups" and then "Create security group."
2. Name your security group (e.g., "Frontend-security-group").
3. Provide a description (e.g., "Allows SSH access to developers").
4. Select your VPC.
5. Click "Create security group."



1. Select the newly created security group.
2. Under "Inbound rules," click "Edit inbound rules."
3. Add rules for HTTP (port 80), HTTPS (port 443), and custom TCP ports as needed (e.g., port 3000 for your application).
4. Click "Save rules."

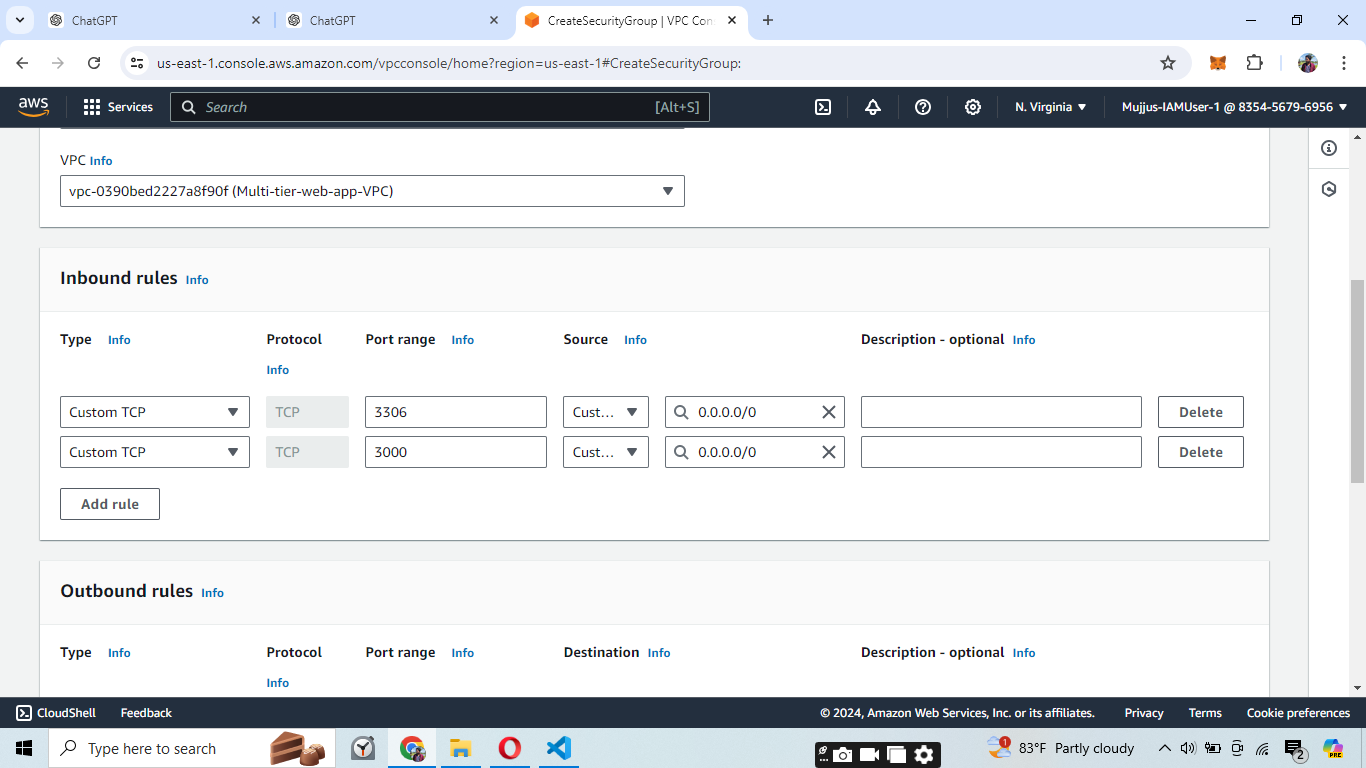
**Step 8: Initialize Backend and Frontend Projects**

Next, we set up the backend and frontend environments using Visual Studio Code.

**Image 15: Initializing Backend Project**

1. Open Visual Studio Code and navigate to your project directory.
2. Initialize the backend project by running npm init -y in the terminal.

**Image 16: Installing Backend Dependencies**

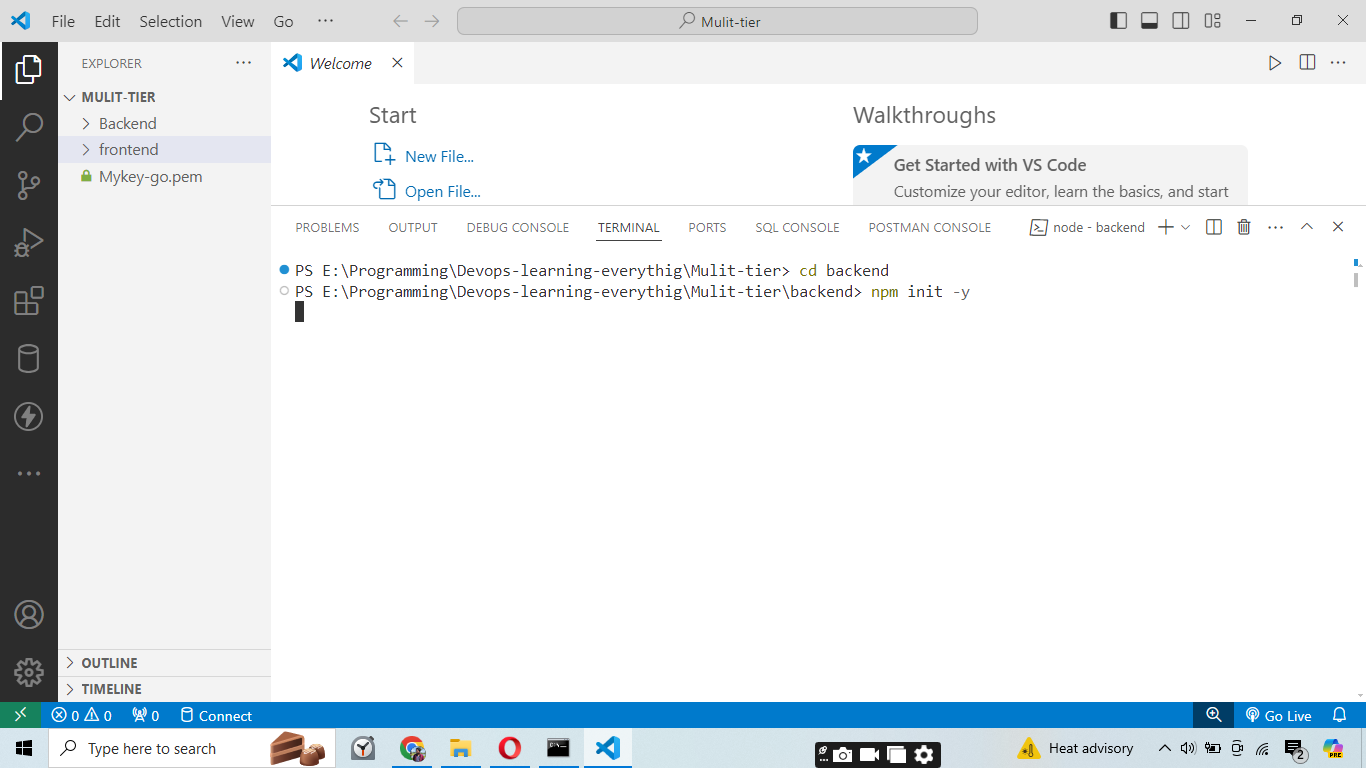
****

1. Install necessary dependencies by running npm install express mysql dotenv.

**Step 9: Launch EC2 Instances**

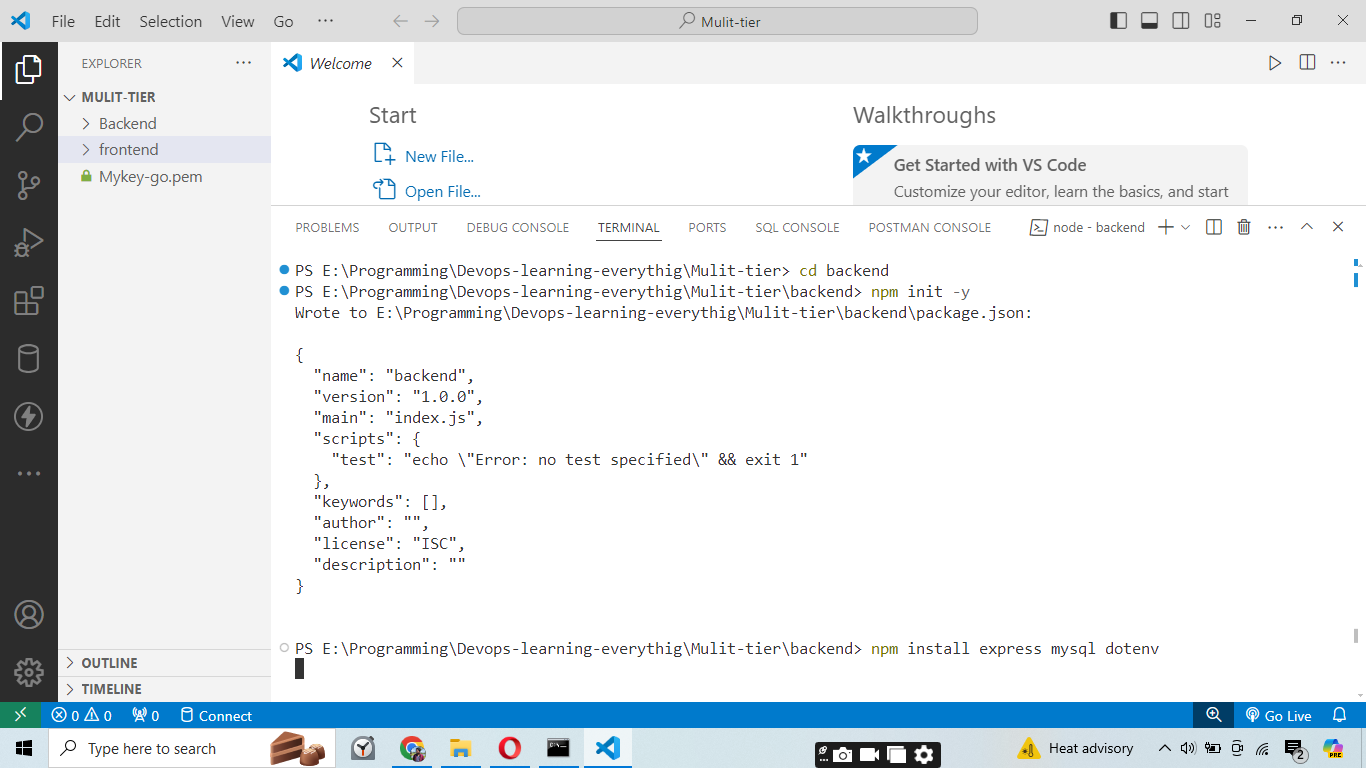
Finally, we launch EC2 instances to host our backend and frontend services.

**Image 17: Launching EC2 Instance**

****

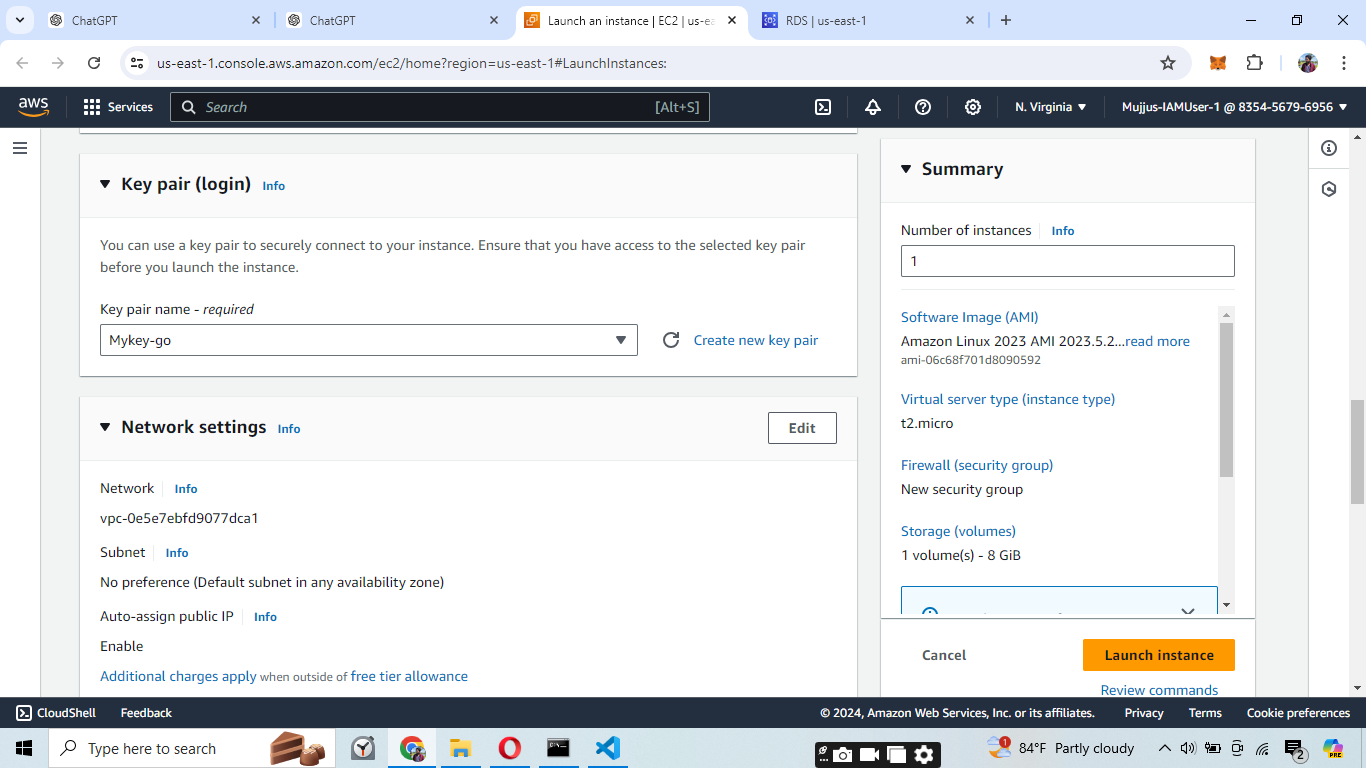
1. In the EC2 dashboard, click on "Launch Instance."
2. Choose an Amazon Machine Image (AMI) (e.g., Amazon Linux 2).
3. Select an instance type (e.g., t2.micro).
4. Click "Next: Configure Instance Details."

**Image 18: Configuring EC2 Instance Details**

****

1. Select your VPC and subnet (e.g., "Multi-tier-subnet-Private").
2. Enable auto-assign public IP.
3. Click "Next: Add Storage."

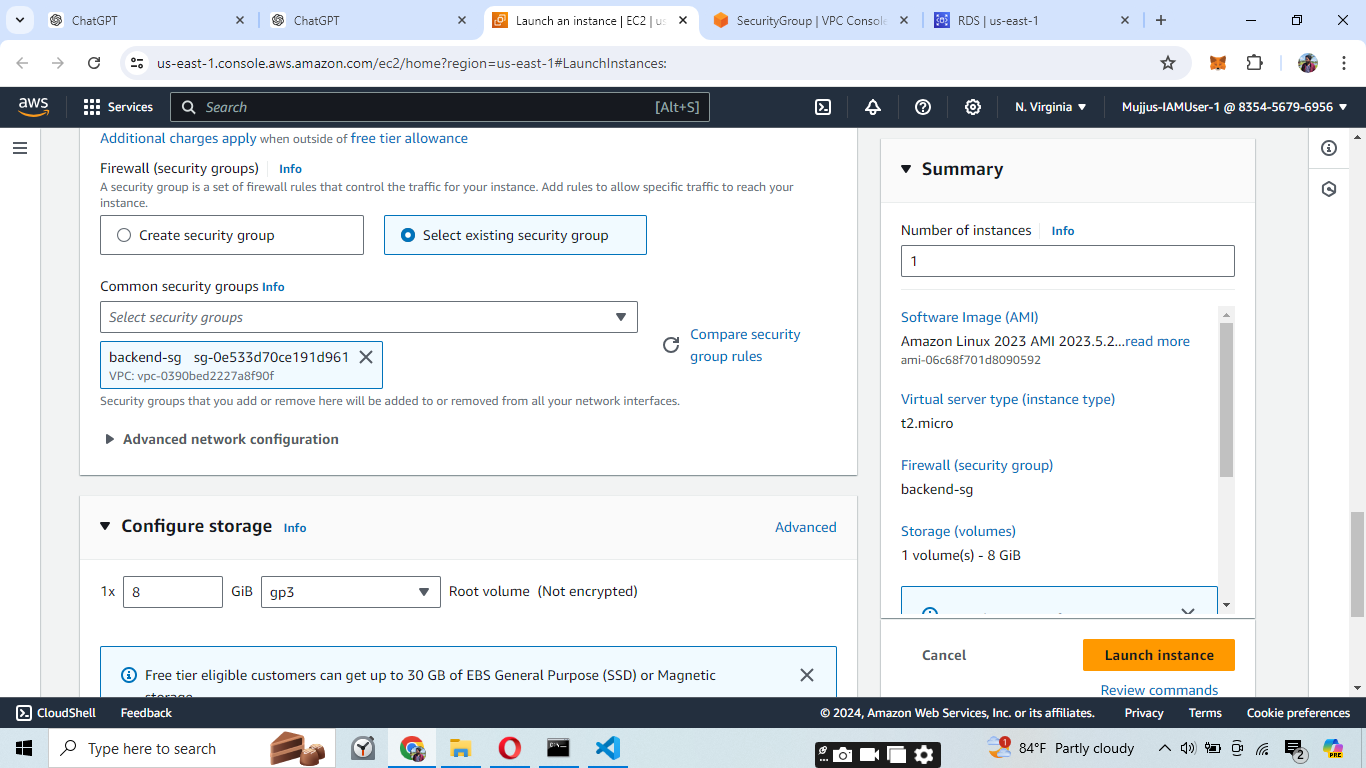
**Image 19: Configuring Security Group for EC2 Instance**

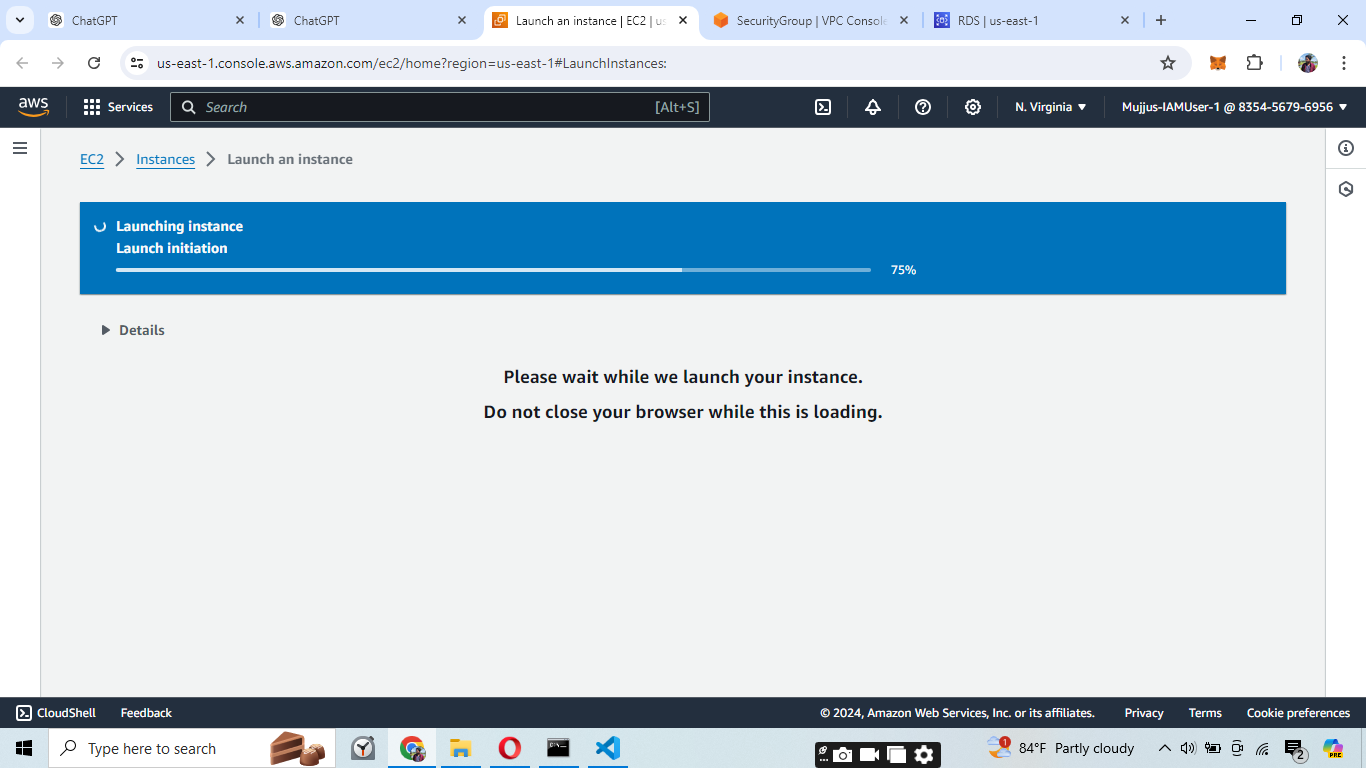
****

1. Choose an existing security group (e.g., "Frontend-security-group") or create a new one with appropriate rules.
2. Click "Review and Launch."

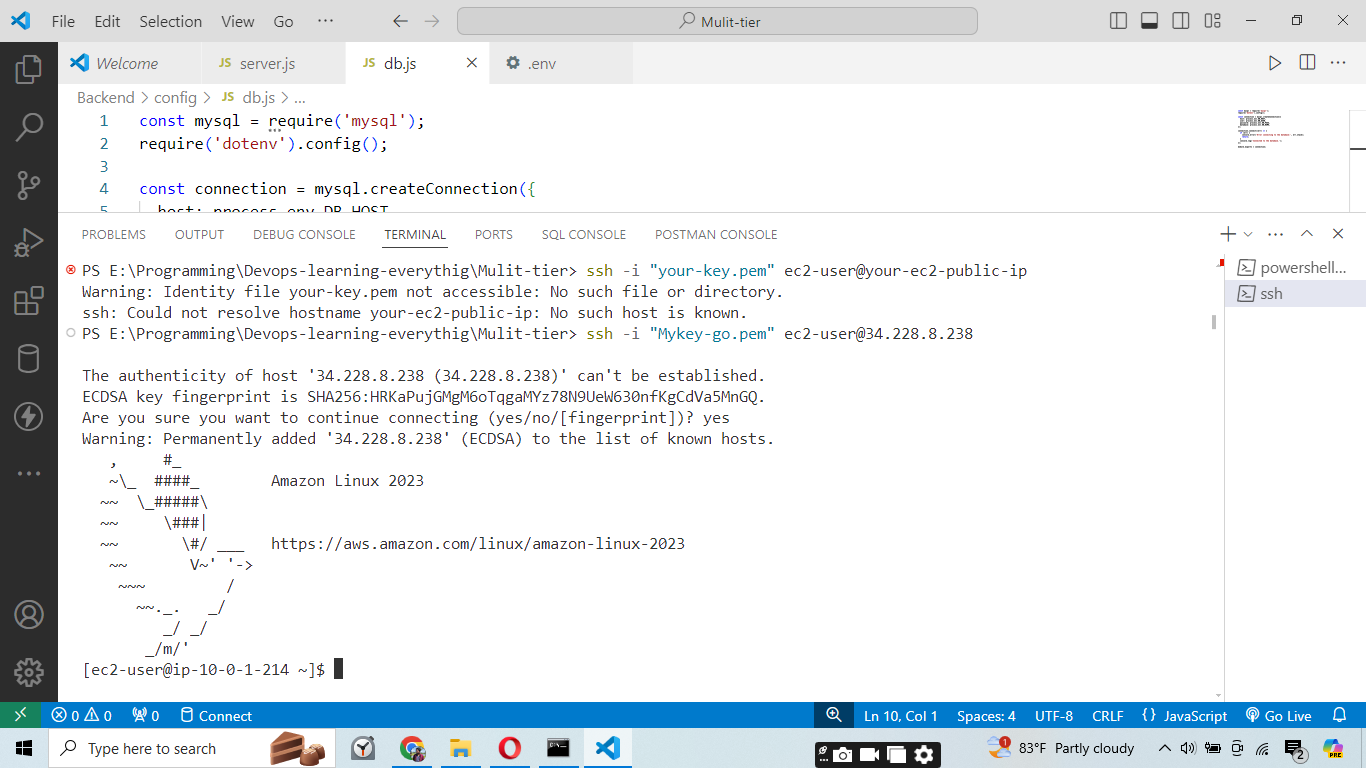
**Image 20: Launching the Instance**

****

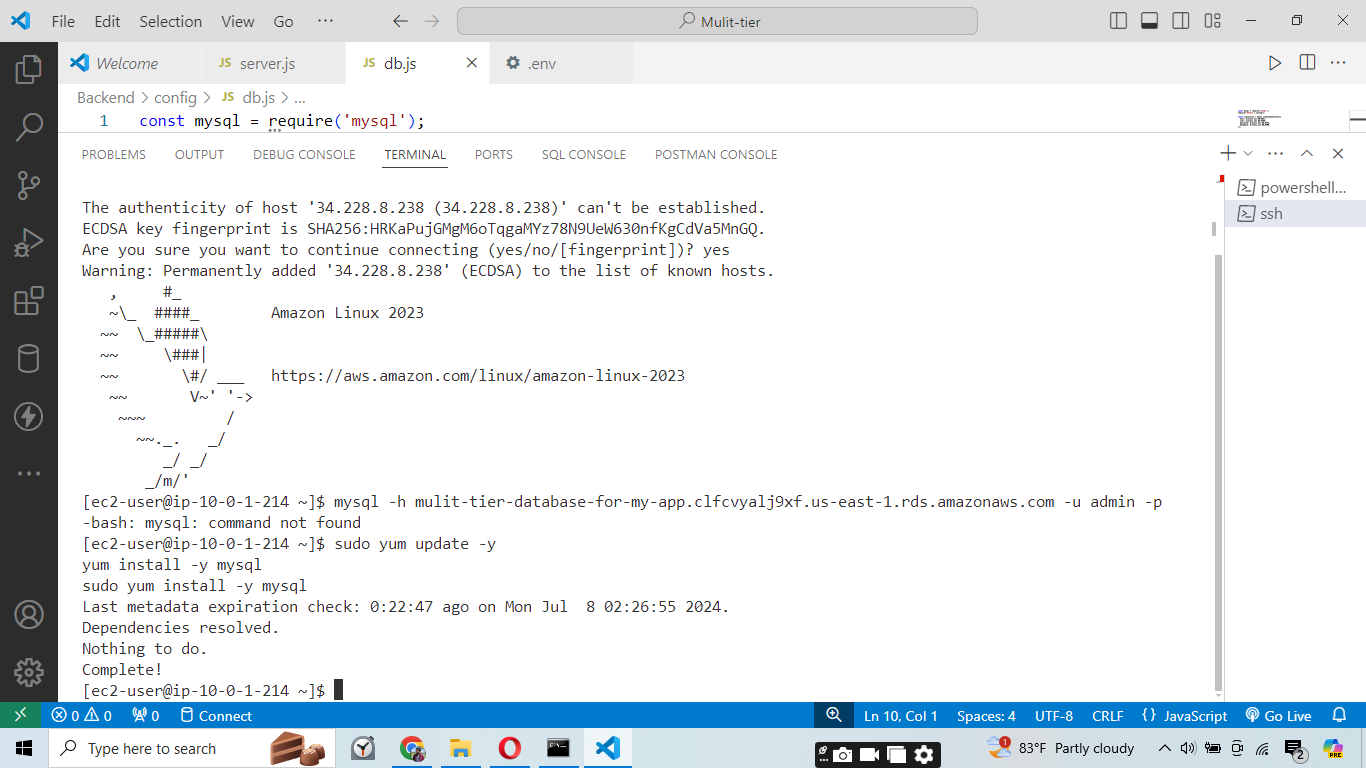




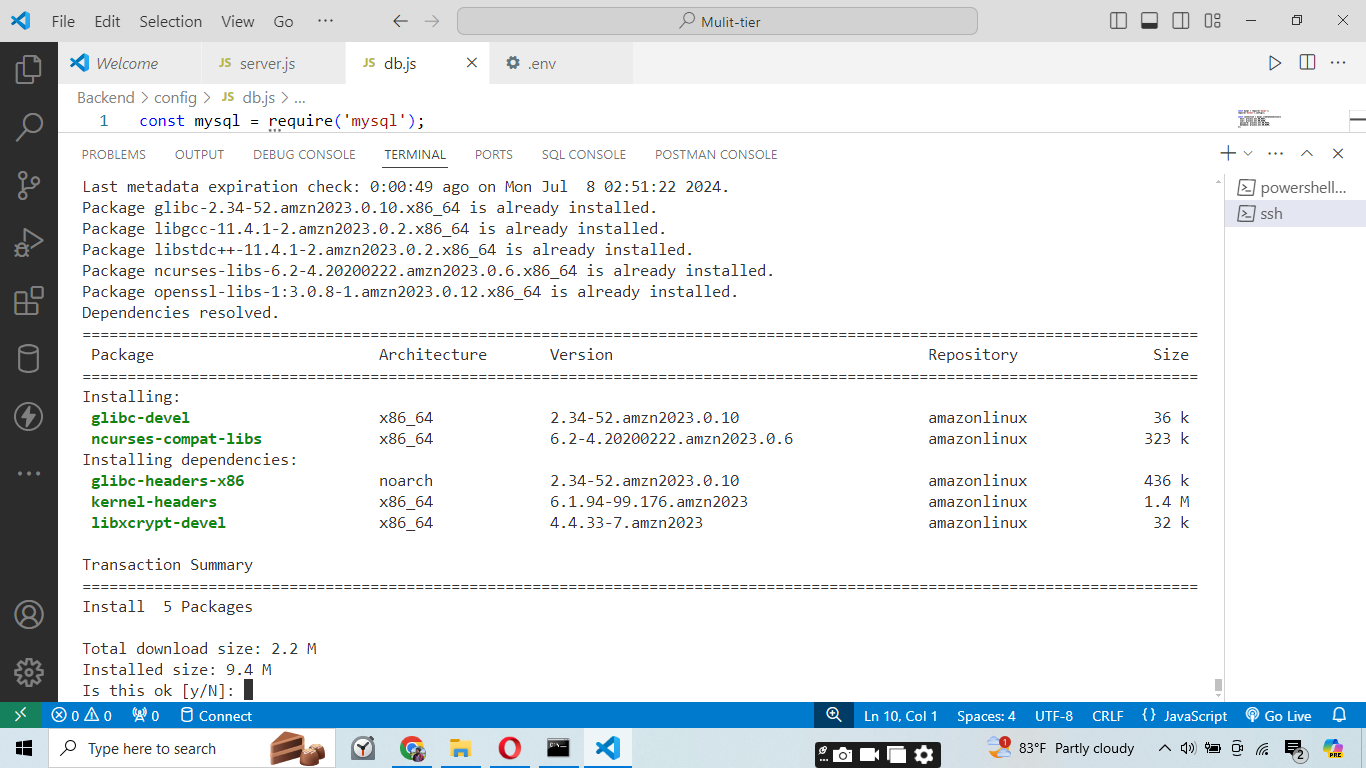
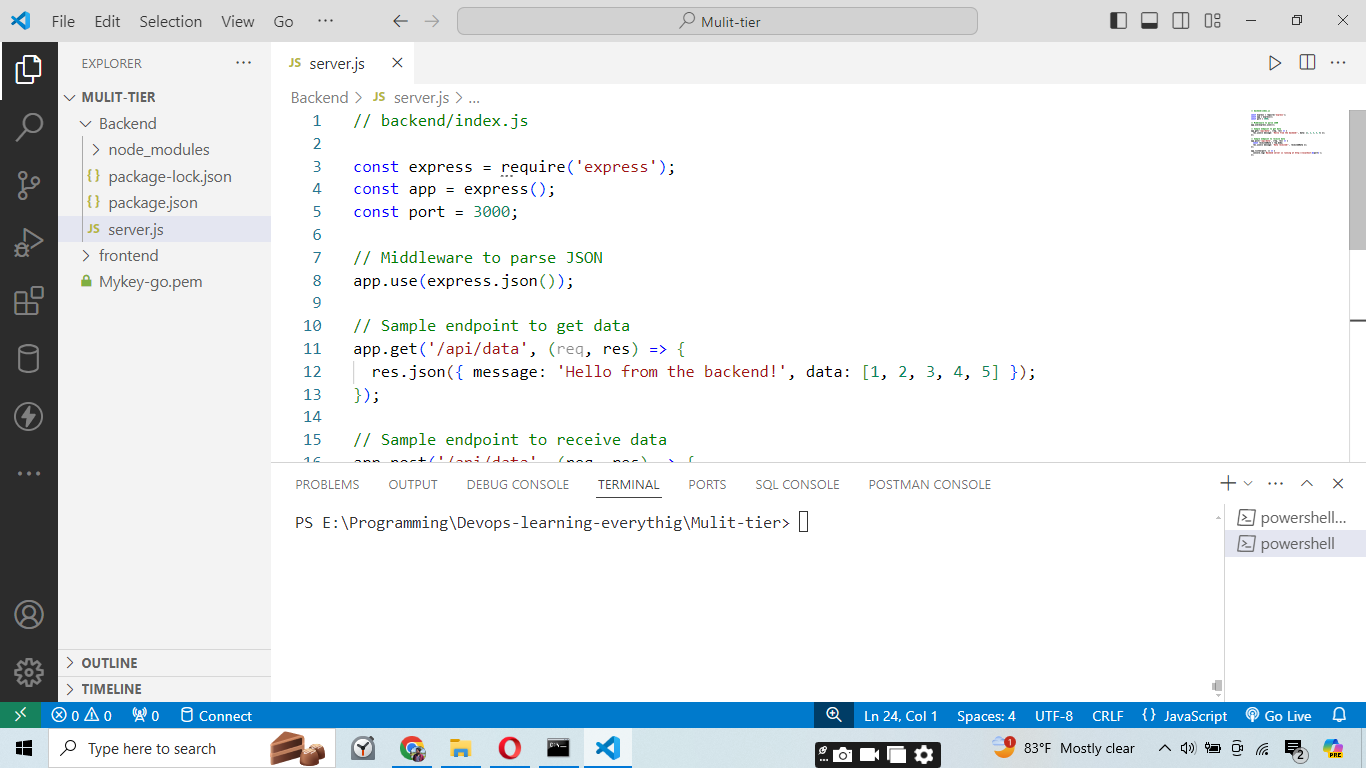
1. the instance details and click "Launch."
2. Select an existing key pair or create a new one.
3. Click "Launch Instances."



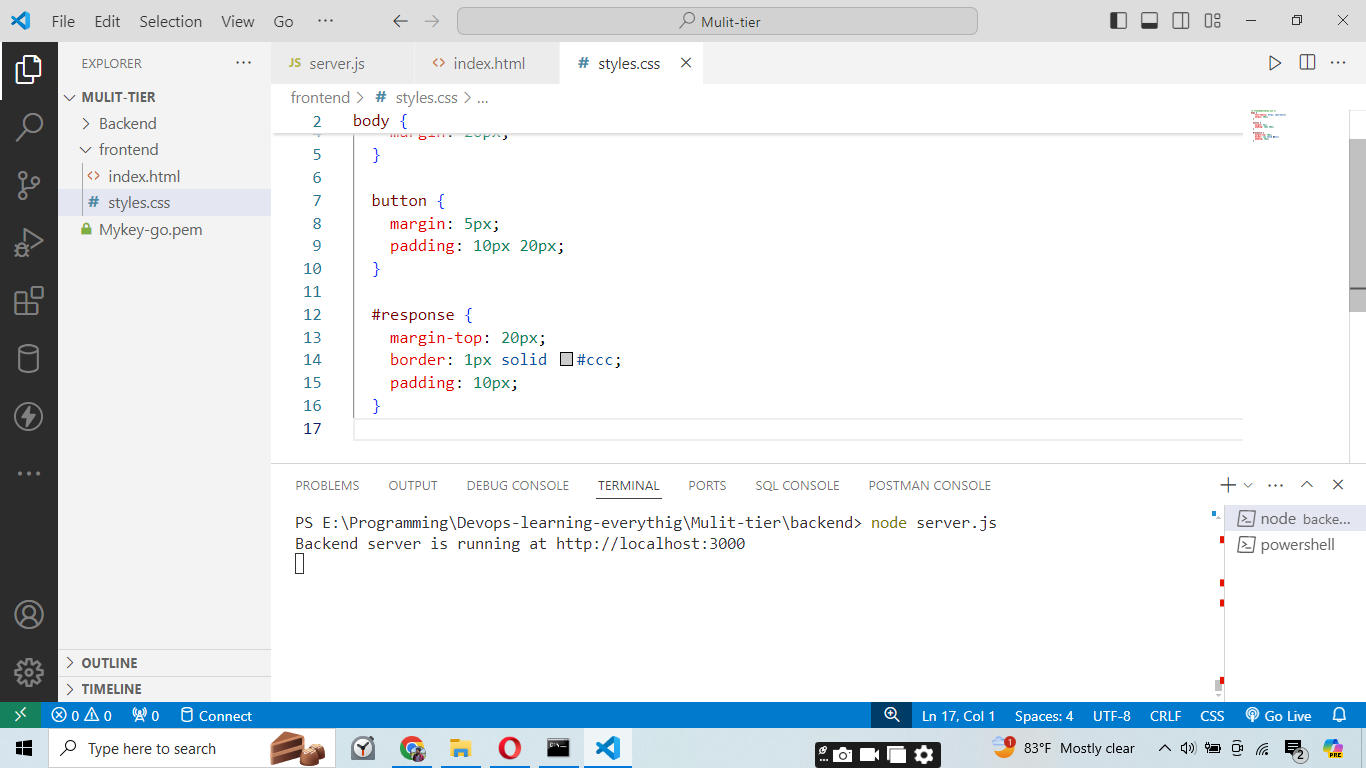
Connecting to the EC2 instance using the pem key generated.



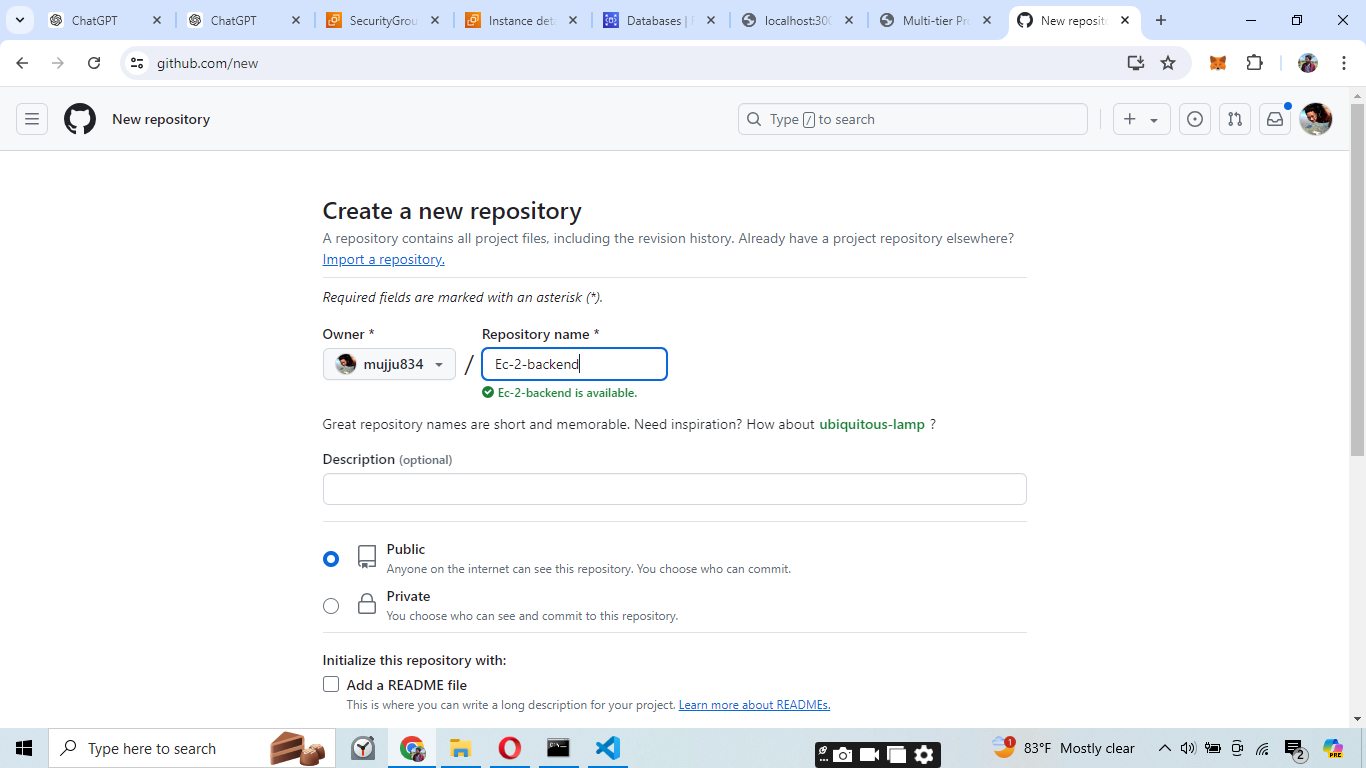
Updating the dependencies.



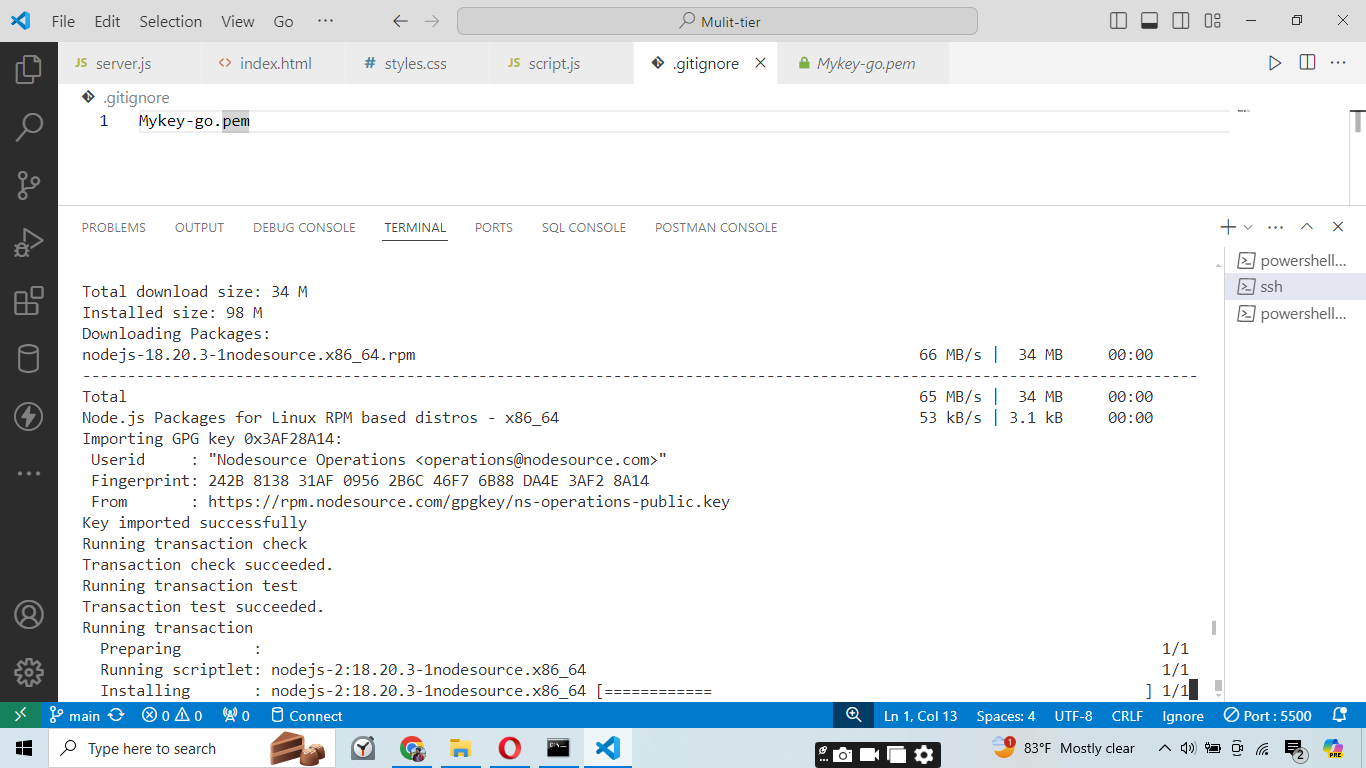
Fixing the server.js file. Which has the backend code.



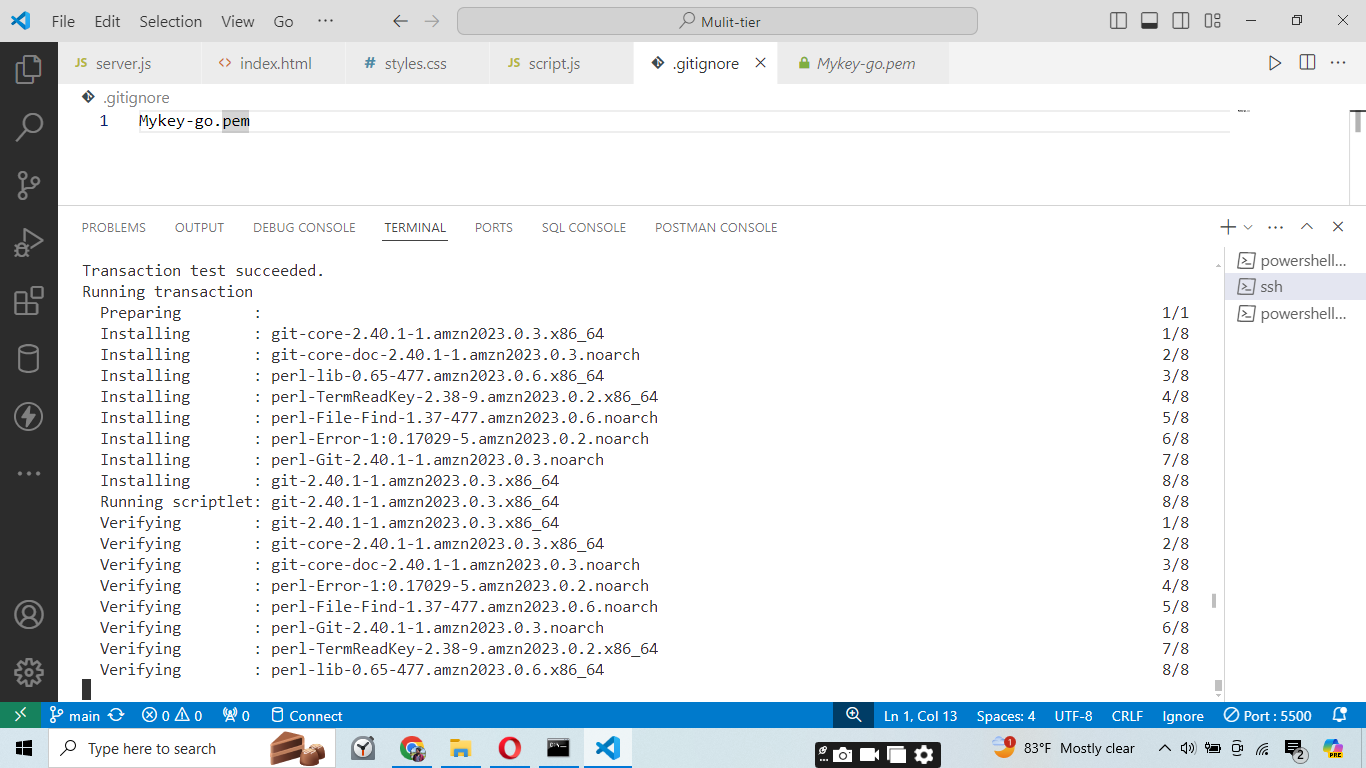
Running the backend code locally.



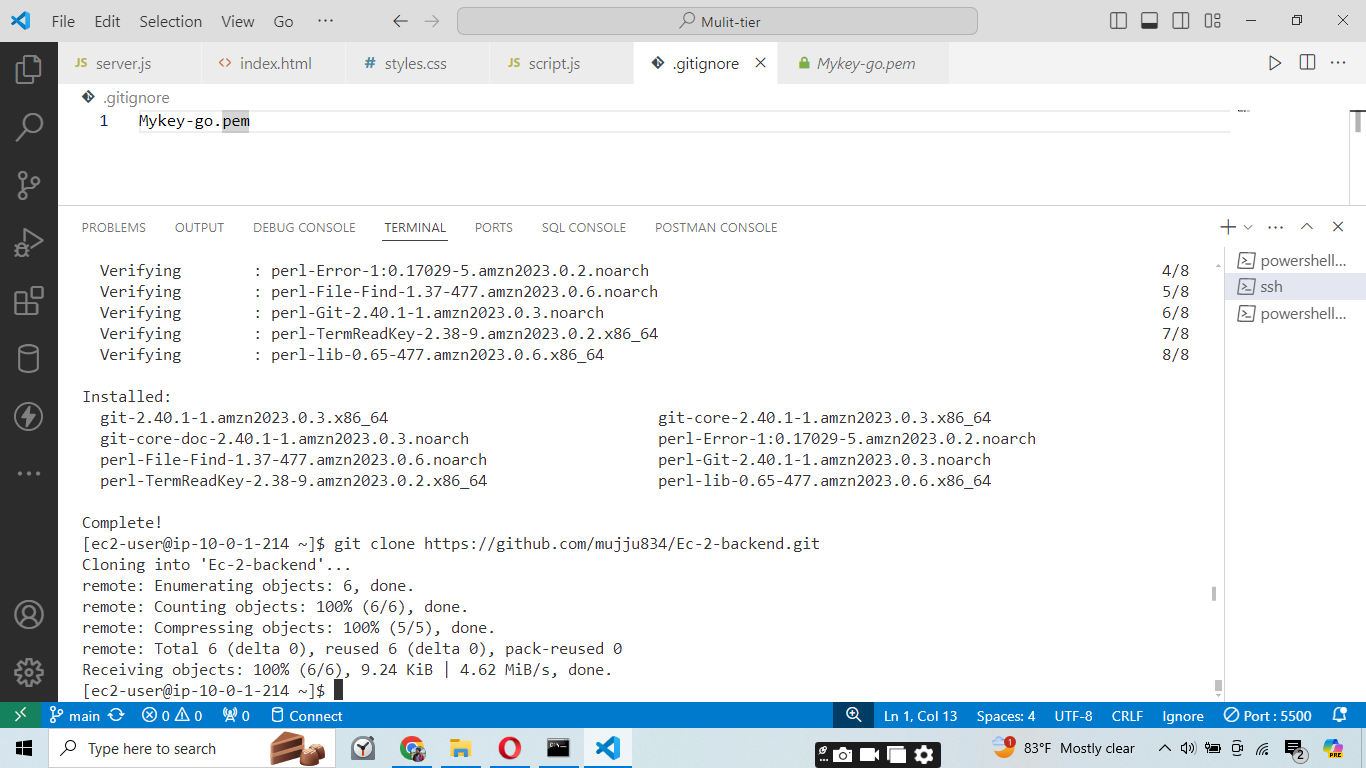
Creating a github repository to push our backend code.



Installing the nodejs on our EC-2 instance.

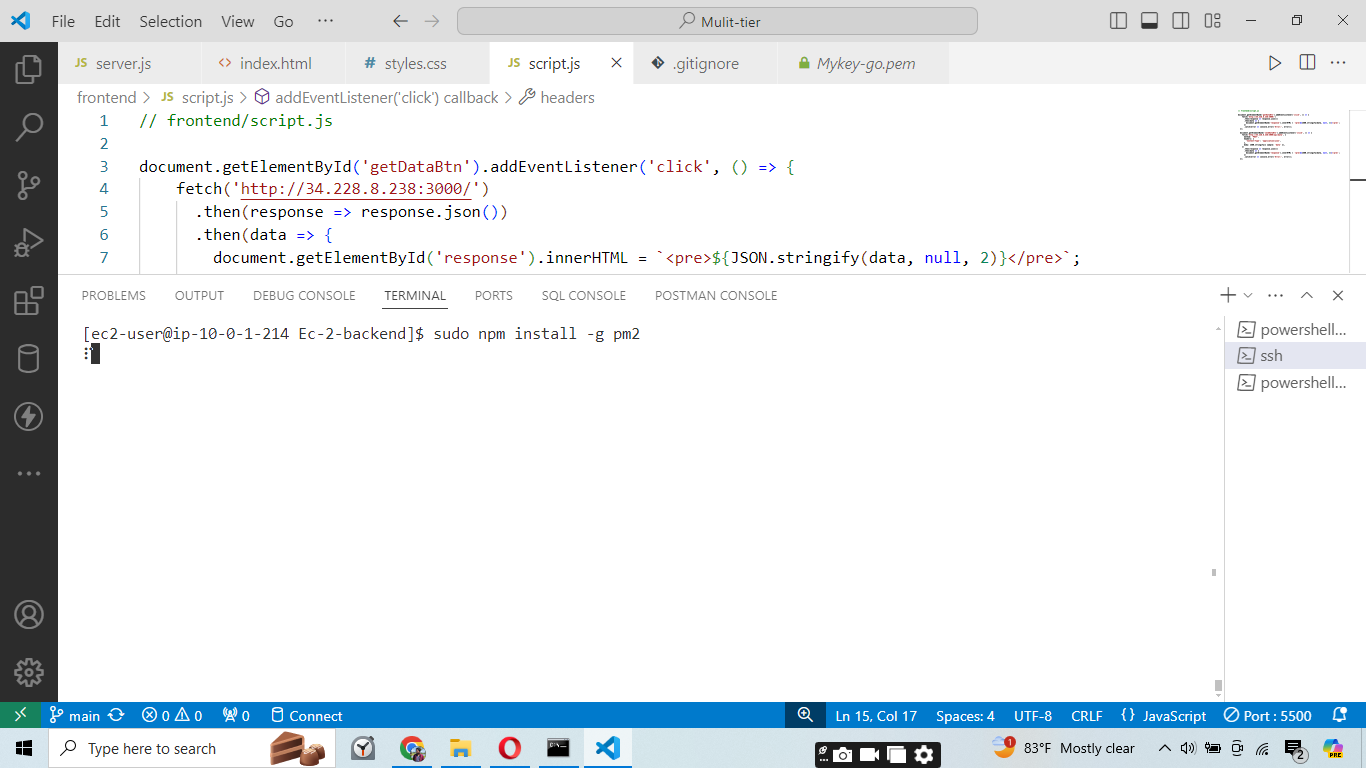


Installing github on our instance.

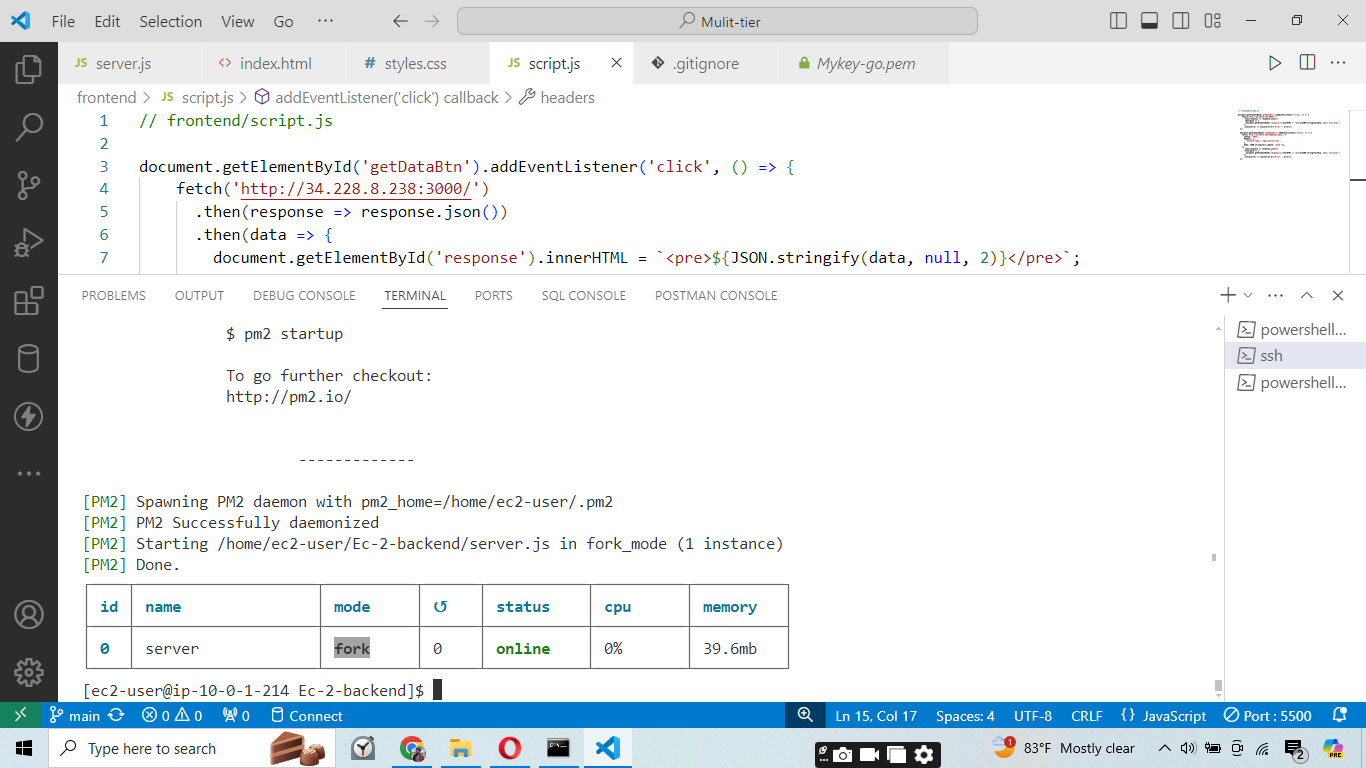


Cloning the github repositoy in our EC2 instance.

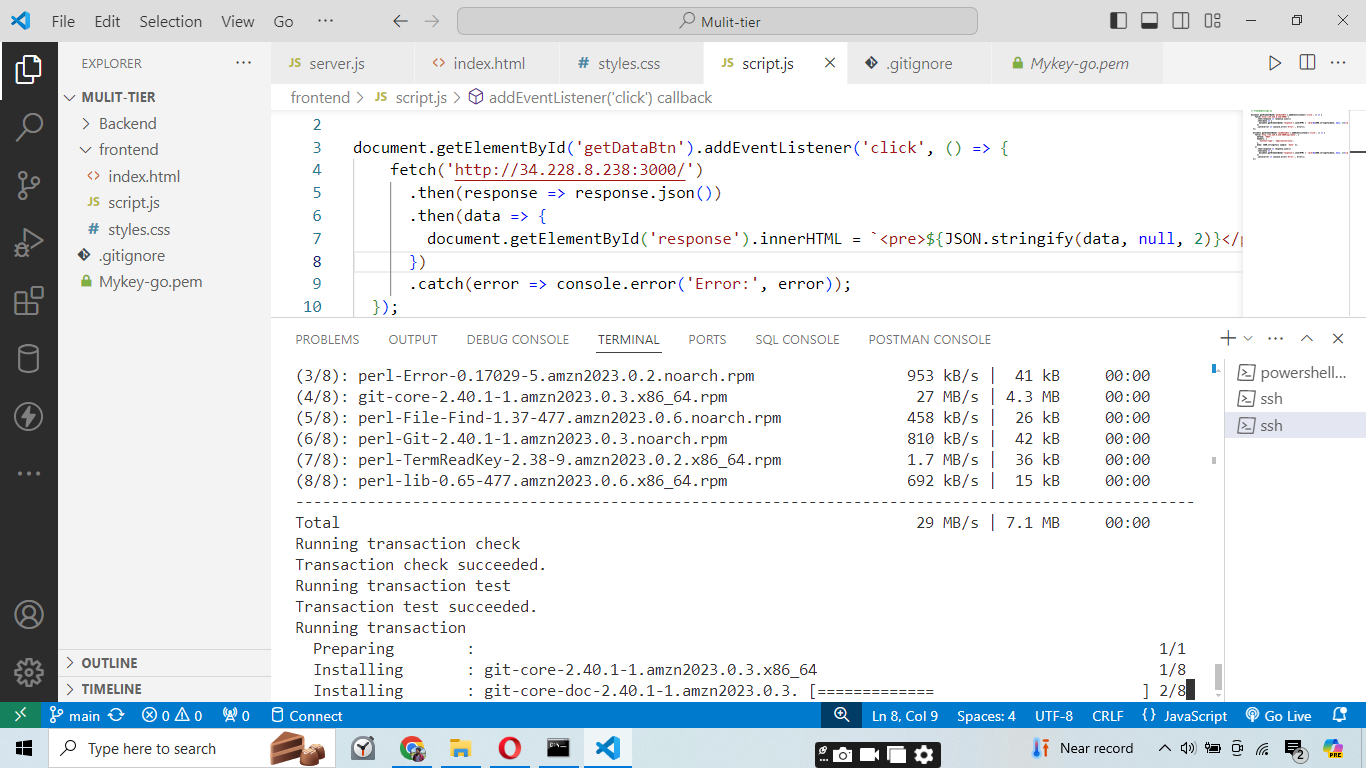




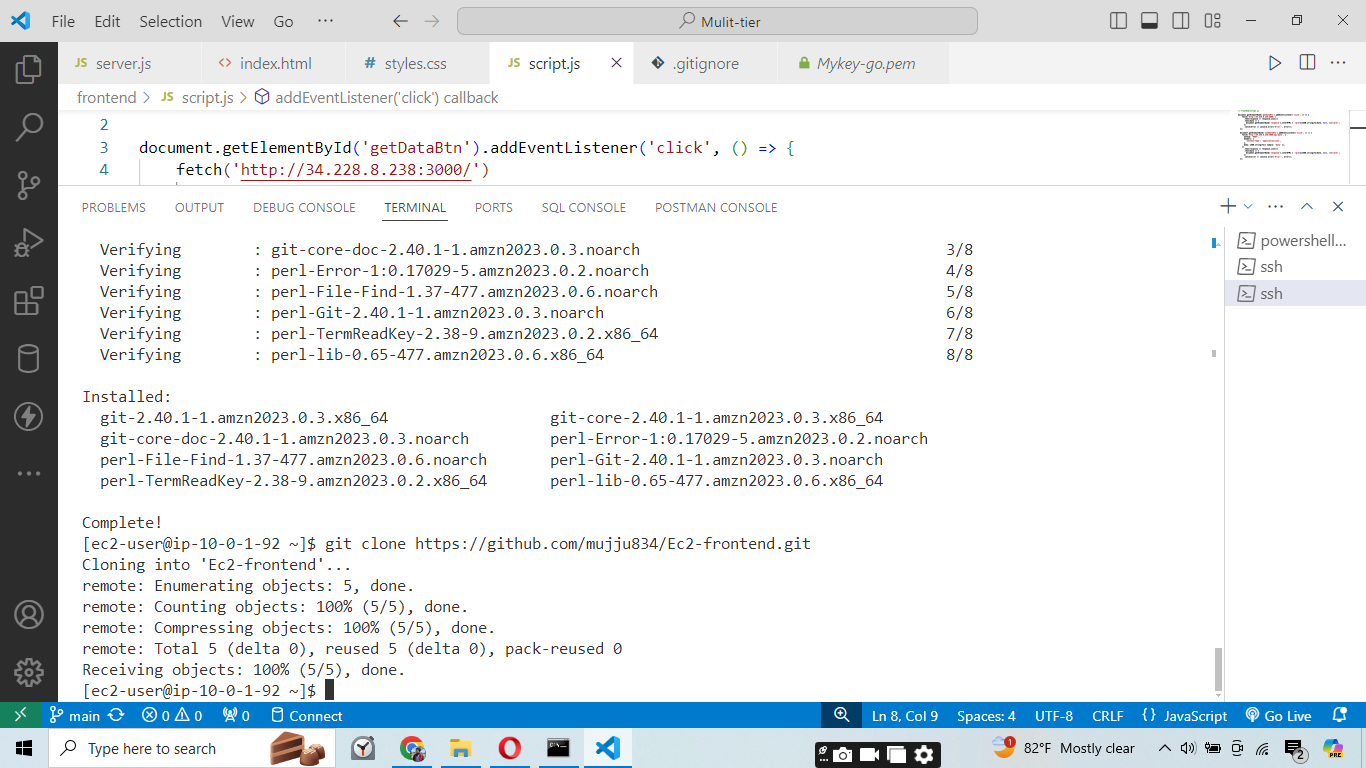
Installing pm2 to deploy the backend service globally.



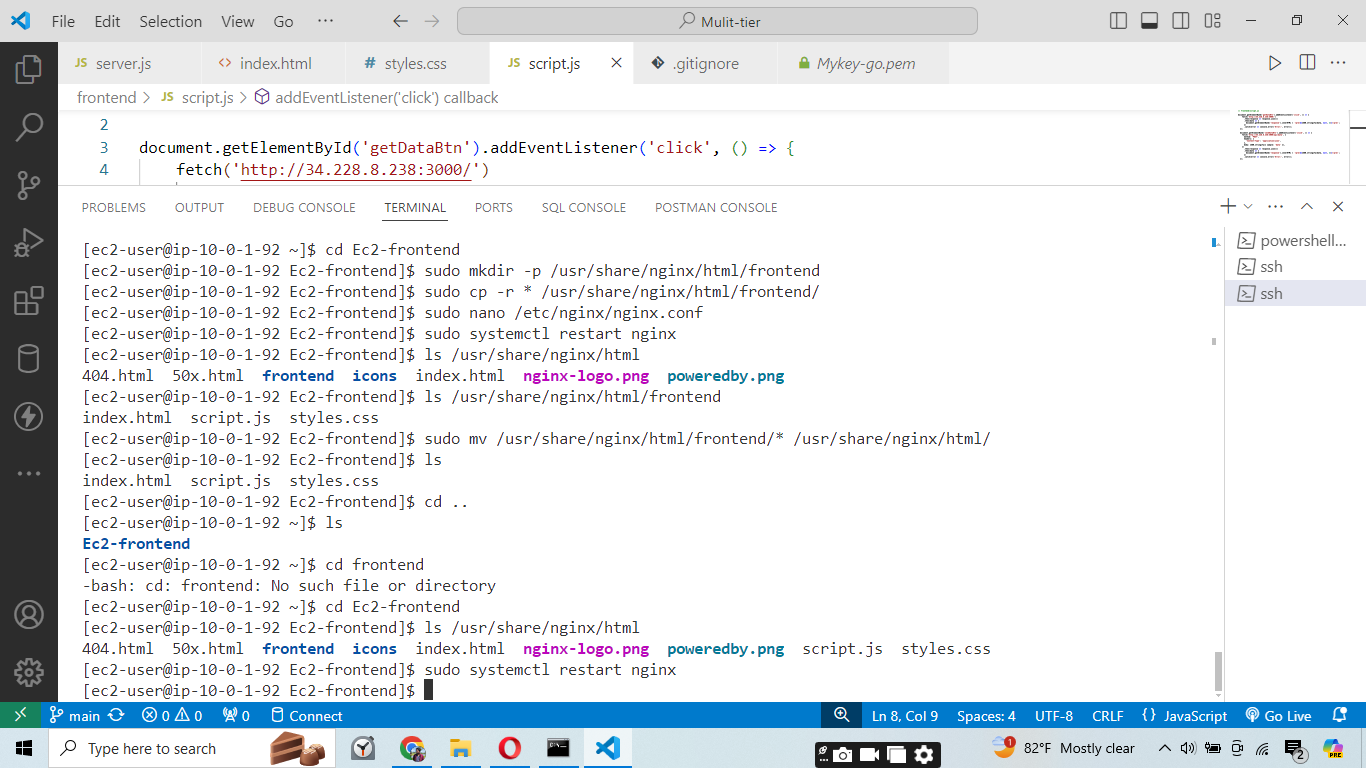
Deployed our backend code globally.



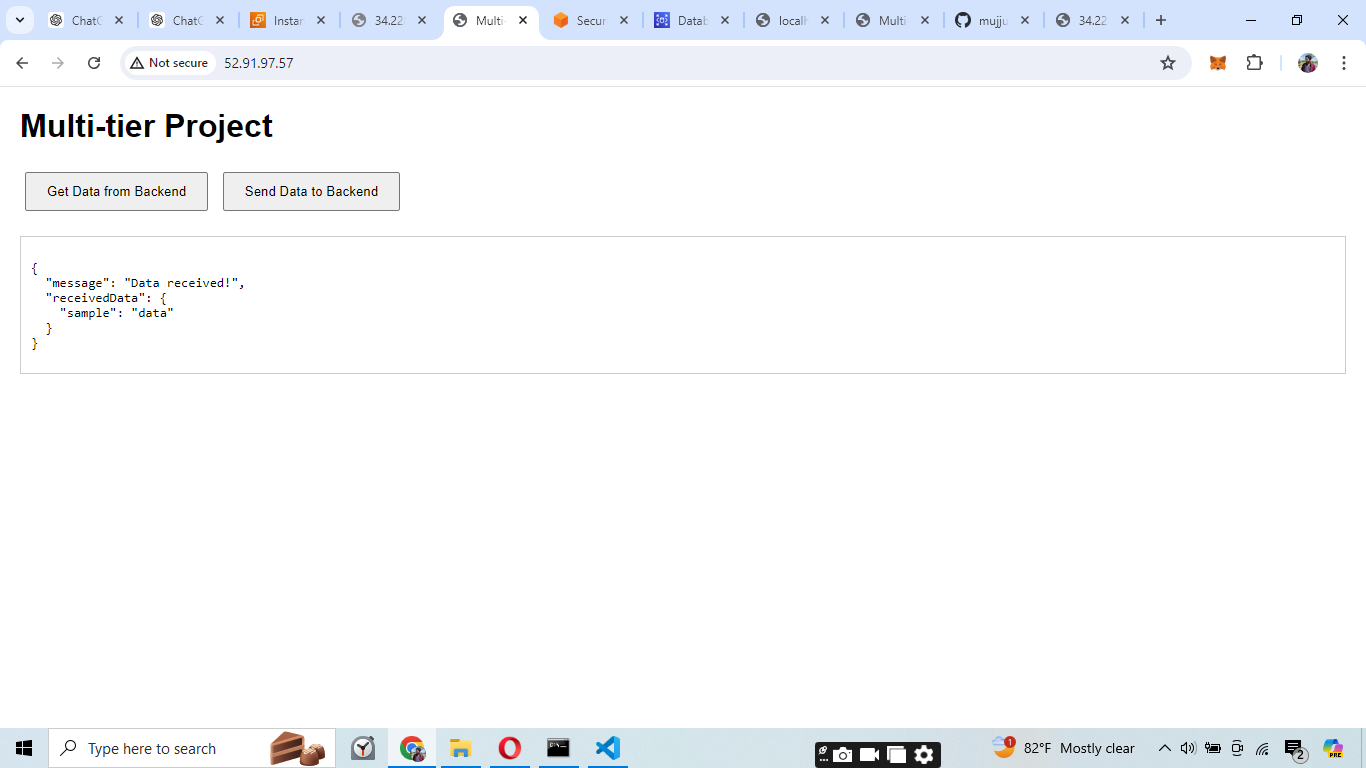
Installing the git and on frontend EC2 instance.



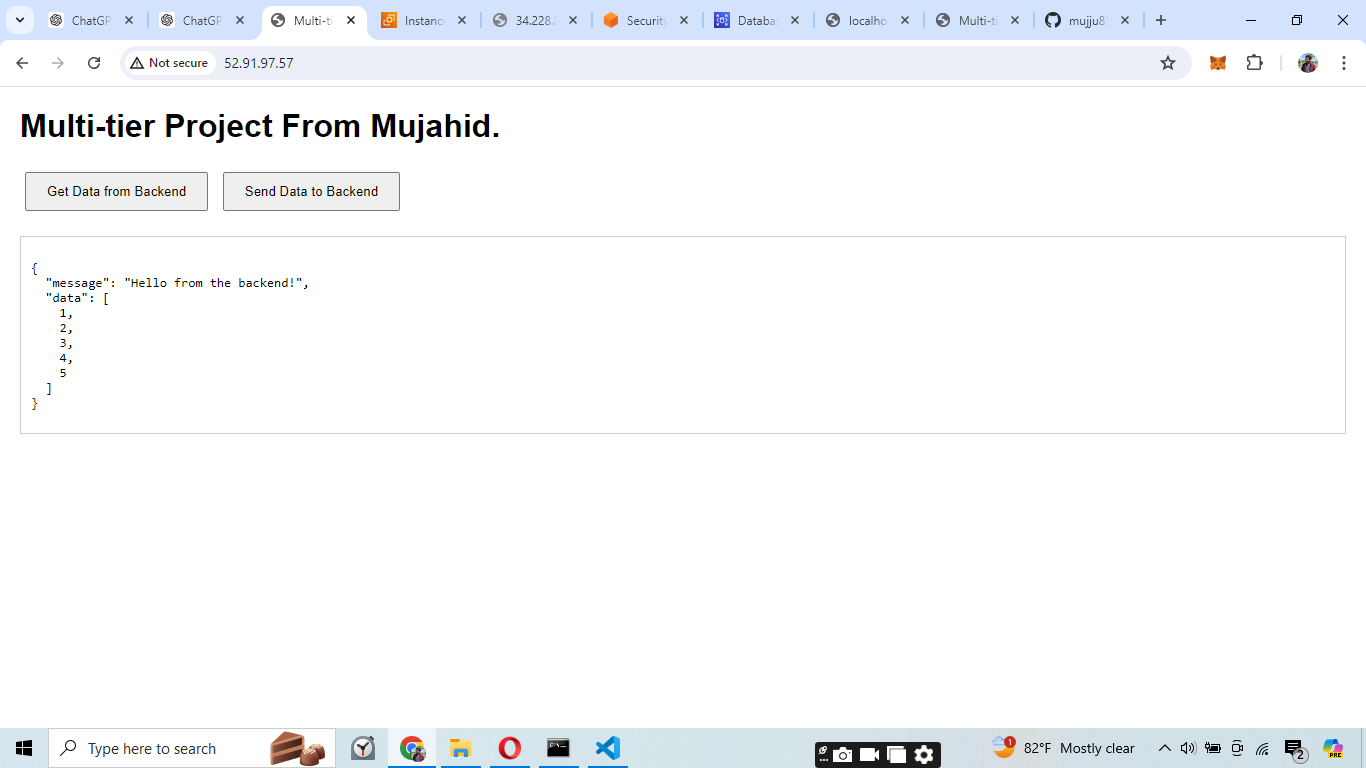
Clone the frotend code to our ec2 instance.



Serve our frontend code using the nginx server on the fronted EC2 instance.



Deployed the frontend and getting response from the backend successfully.



**Problems Faced**

1. **Subnet Association Issues**:
   * Ensuring correct subnets are associated with the correct route tables for proper network traffic management.
2. **Route Table Configuration**:
   * Properly configuring routes to use the NAT Gateway for instances in the private subnet to access the internet.
3. **Security Group Misconfigurations**:
   * Setting up security groups with the correct inbound and outbound rules to allow necessary traffic while maintaining security.
4. **EC2 Instance Setup**:
   * Correctly launching an EC2 instance with the appropriate AMI, instance type, key pair, and security group.
5. **SSH Access Problems**:
   * Establishing SSH access using the key pair and troubleshooting issues related to identity file accessibility.
6. **Backend Server Configuration**:
   * Setting up and running a Node.js backend server with Express, ensuring proper middleware and endpoint configurations.

**Learnings**

1. **Network Configuration**:
   * Gained experience in associating subnets with route tables and understanding how different components of a VPC interact.
2. **Routing and NAT Gateway**:
   * Learned how to configure route tables and NAT Gateways to allow instances in a private subnet to access the internet.
3. **Security Best Practices**:
   * Understanding the importance of security group rules and how to configure them to balance accessibility and security.
4. **Instance Management**:
   * Experience in launching EC2 instances, selecting appropriate AMIs, and using key pairs for secure access.
5. **Troubleshooting SSH**:
   * Developed skills in troubleshooting SSH access issues and ensuring the correct setup of identity files.
6. **Dependency Management**:
   * Learned how to resolve dependency issues when installing software on EC2 instances and the importance of understanding package dependencies.
7. **Backend Development**:
   * Gained experience in setting up a Node.js backend server, using Express, and creating API endpoints.
8. **Database Connectivity**:
   * Understood how to configure and test database connections from a backend server to an RDS instance.

**Solutions**

1. **Subnet Association**:
   * Ensure that the correct subnets are associated with their respective route tables during the initial VPC setup.
2. **Route Table Configuration**:
   * Double-check route table configurations and ensure NAT Gateway is correctly set up for the private subnet.
3. **Security Group Setup**:
   * Carefully configure security group rules and regularly review them to maintain security while allowing necessary traffic.
4. **EC2 Instance Setup**:
   * Follow a checklist when launching instances to ensure all configurations, such as AMI, instance type, key pair, and security group, are correctly set.
5. **SSH Access**:
   * Verify the correct path and permissions for the key pair file and ensure the correct username and public IP are used.
6. **Backend Server Setup**:
   * Follow best practices in setting up middleware and endpoint configurations in the backend server.
7. **Database Connection**:
   * Test database connections thoroughly and handle any connectivity issues by checking security group rules, RDS configurations, and network settings.

**Conclusion**

The project provided a comprehensive learning experience in setting up a multi-tier architecture on AWS. Key challenges included network configuration, instance management, security setup, and dependency management. By systematically addressing these issues, we developed a robust understanding of AWS services and best practices in deploying scalable and secure applications. The knowledge gained will be invaluable in future projects, ensuring efficient and effective infrastructure setups.