Three-Tier Architecture Deployment on AWS

Index

Part 0: Setup

Objectives:

1. Download Code from GitHub Repository
2. IAM EC2 Instance Role Creation
3. S3 Bucket Creation

Part 1: Networking and Security

Objectives:

1. Create VPC
2. Configure Subnets
3. Setup Route Tables
4. Attach Internet Gateway
5. Setup NAT Gateway
6. Configure Security Groups

Part 2: Database Deployment

Objectives:

1. Deploy Database Layer
2. Configure Subnet Groups
3. Implement Multi-AZ Database Configuration

Part 3: App Tier Instance Deployment

Objectives:

1. Create App Tier Instance
2. Configure Software Stack for Node.js Application
3. Setup Database Schema
4. Test Database Connectivity

Part 4: Internal Load Balancing and Auto Scaling

Objectives:

1. Create AMI of App Tier Instance
2. Create a Launch Template
3. Configure Auto Scaling
4. Deploy Internal Load Balancer

Part 5: Web Tier Instance

DeploymentObjectives:

1. Update NGINX Configuration Files
2. Create a Web Tier Instance
3. Configure Software Stack for NGINX and React.js Website

Part 6: External Load Balancer and

Auto ScalingObjectives:

1. Create AMI of Web Tier Instance
2. Create a Launch Template
3. Configure Auto Scaling
4. Deploy External Load Balancer

Introduction

This documentation outlines the steps to deploy a three-tier architecture on Amazon Web Services (AWS). The architecture consists of separate layers for presentation, application, and data, ensuring scalability, flexibility, and security.

Technologies, Services, and Tools Used:

- GitHub
- AWS Cloud (Amazon Web Services)
- EC2 (Elastic Compute Cloud)
- Security Groups
- AWS RDS (Relational Database Service)
- Route Table
- Internet Gateway
- NAT Gateway
- Nginx
- Elastic Load Balancer
- Auto Scaling Group

Part 0: Setup

Step-by-Step Guide:

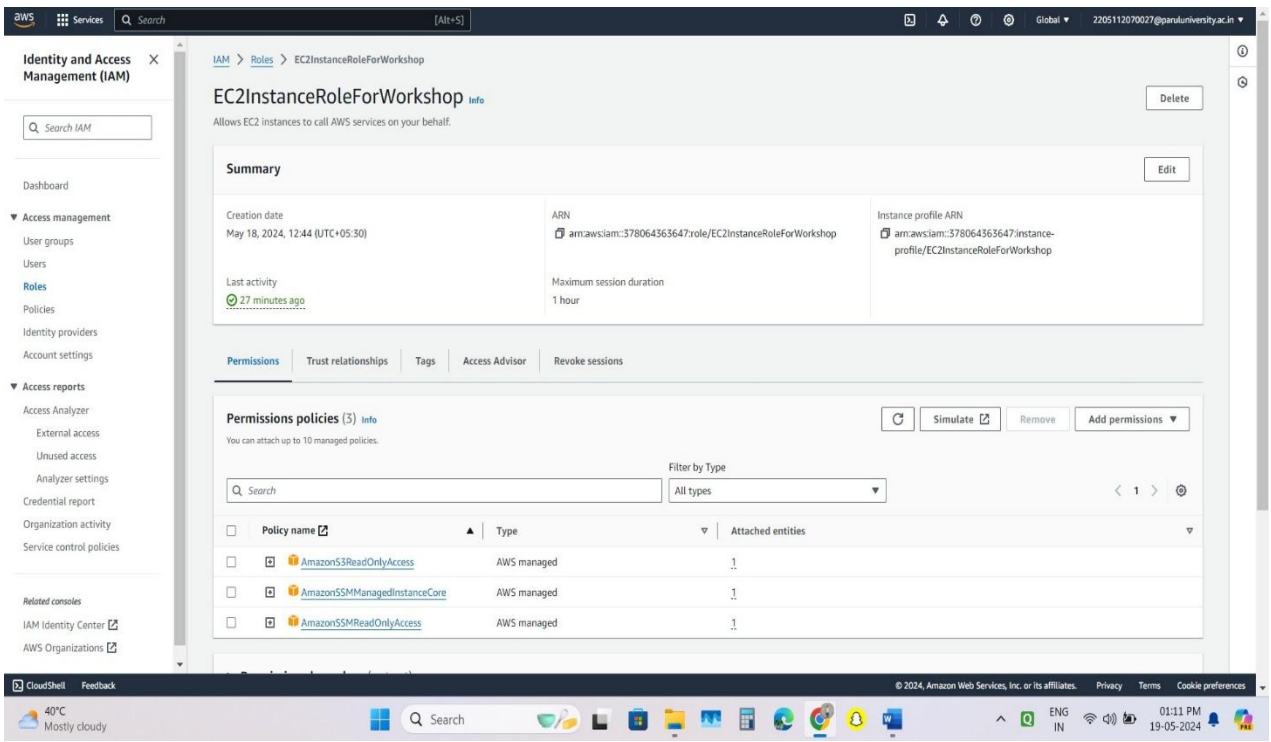
1. Download Code from GitHub Repository:

- Clone or download the project code from the GitHub repository provided.

```
git clone https://github.com/aws-samples/aws-three-tier-web-architecture-workshop.git
```

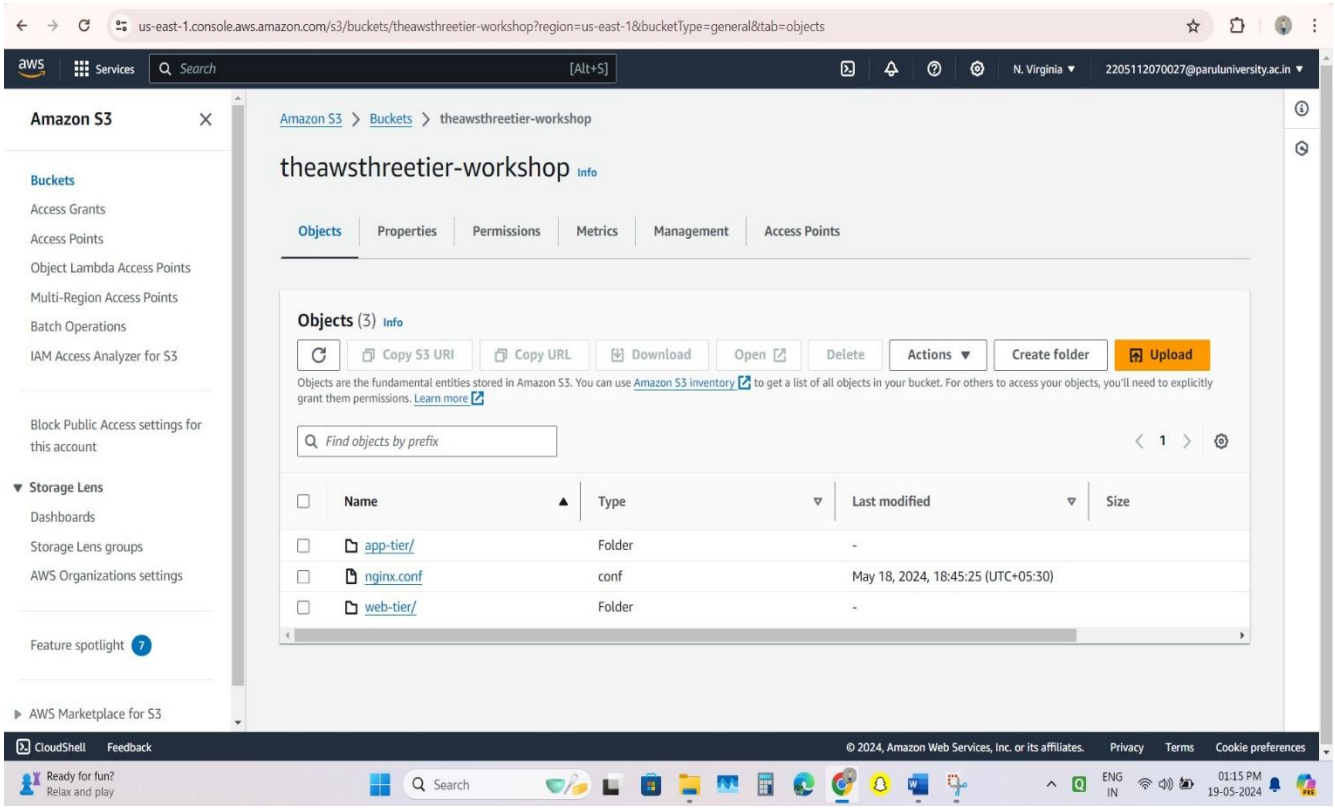
2. IAM EC2 Instance Role Creation:

- Create an IAM role for EC2 instances with necessary permissions foraccessing AWS services.
- Attach 2 policy –
- **AmazonSSMManagedInstanceCore**
- **AmazonS3ReadOnlyAccess**



3. S3 Bucket Creation:

- Create an S3 bucket and accept the rest of the default settings to create the bucket. It is used for storing assets or other resources required for the project.

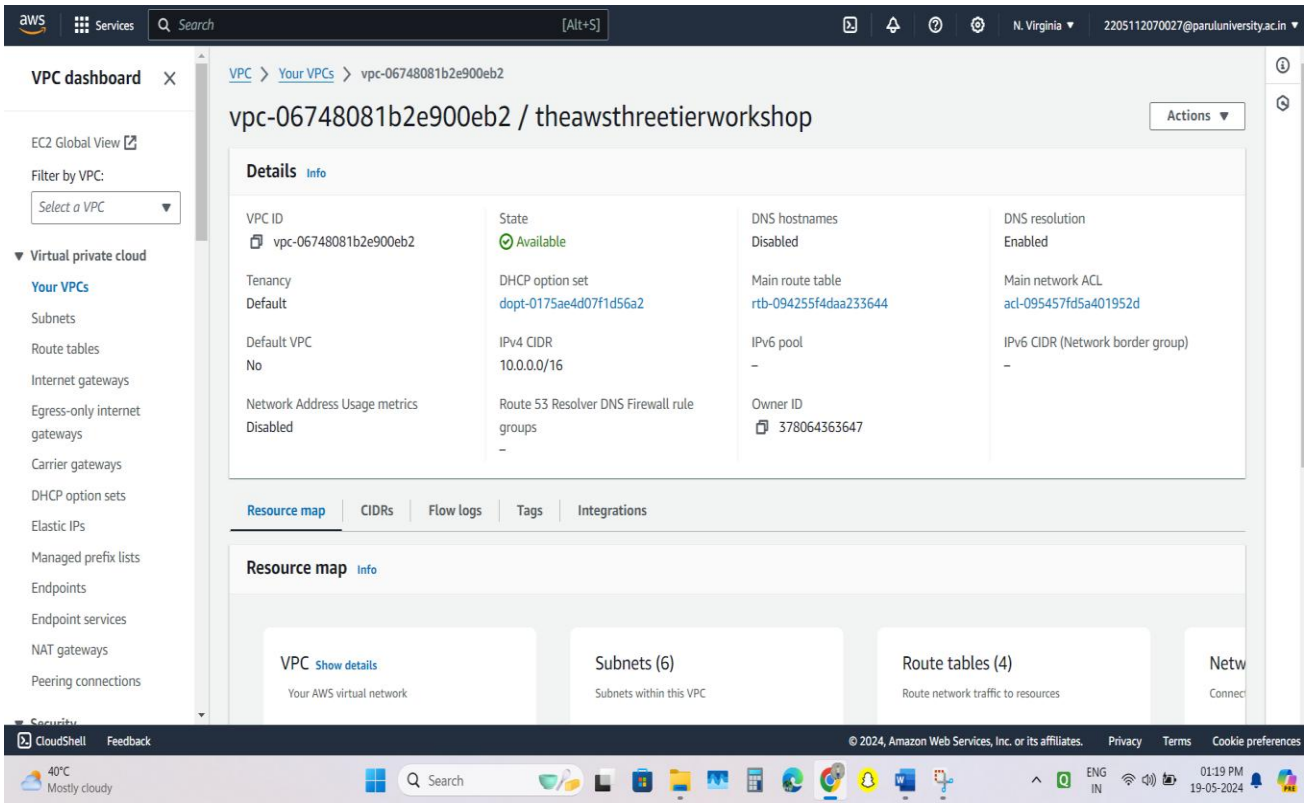


Part 1: Networking and Security

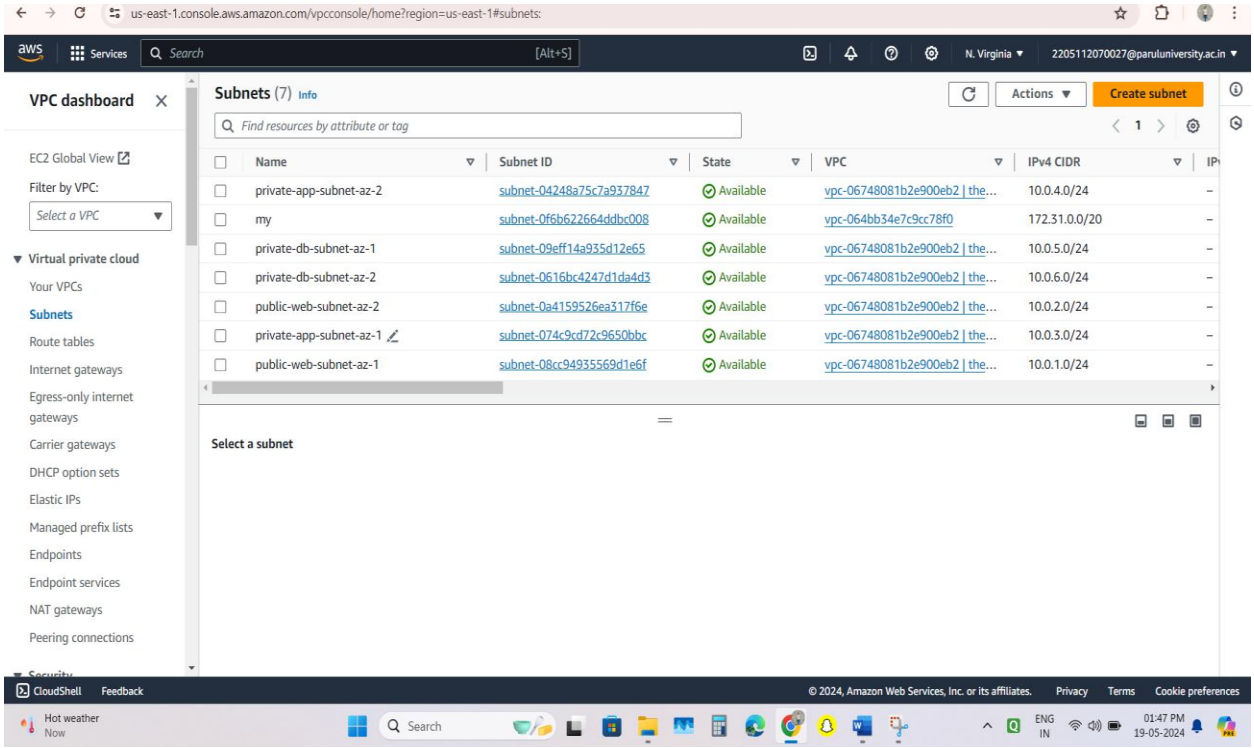
Step-by-Step Guide:

1. Create VPC:

- Set up a Virtual Private Cloud (3twebbapp-demo) to isolate the networkenvironment.
- Define CIDR block (10.0.0.0/16)

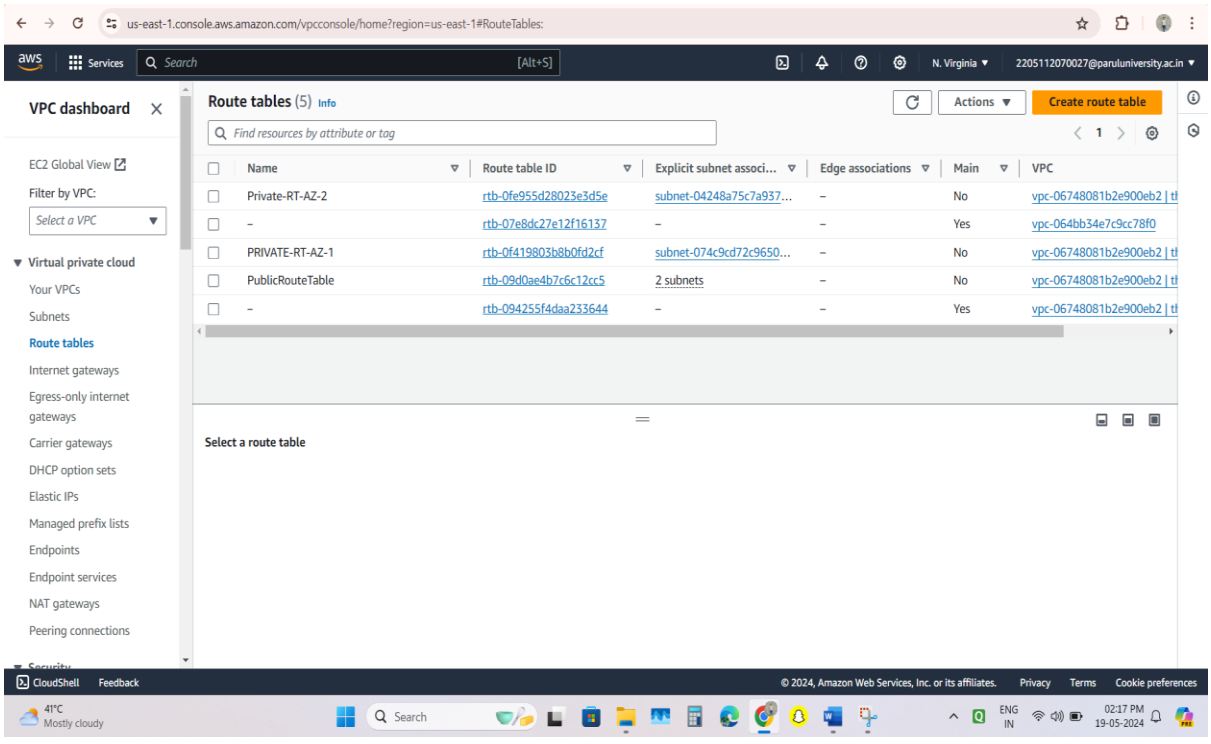


2. **Configure Subnets :-**
3. Created 2 public, 4 private subnets and 2 Availability Zone within theVPC including CIDR Block.
4. Private DB means (For Database)



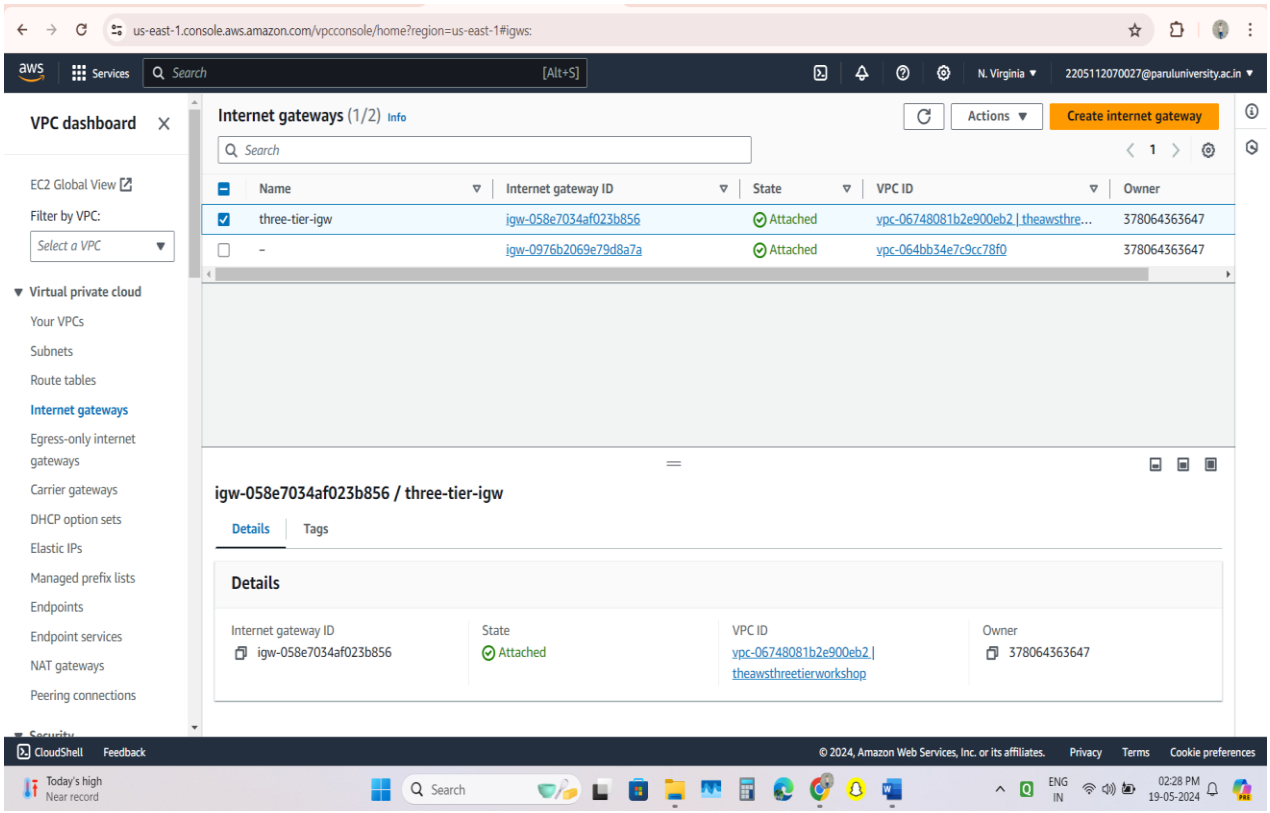
3 Setup Route Tables:

- Configured 2 route tables (Private-Route-Table & Public-Route-Table)to control the routing of network traffic.



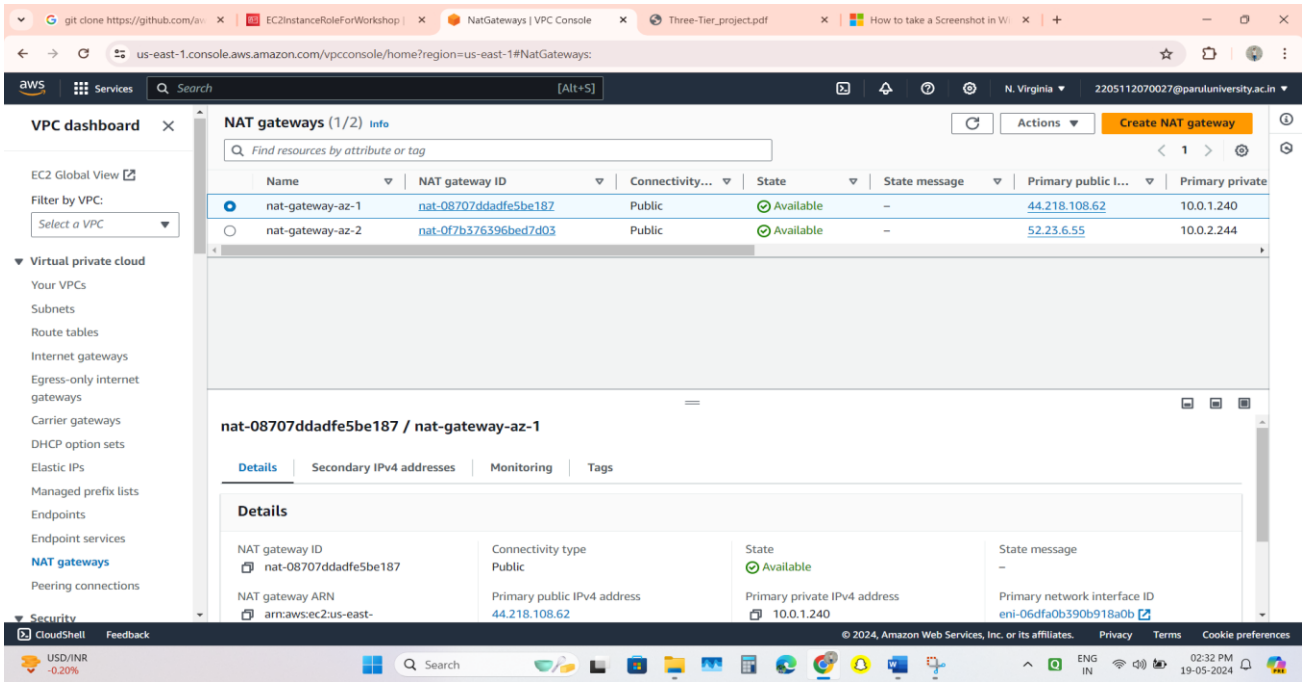
4. Create and Attach Internet Gateway:

Create and Attach an Internet Gateway to the VPC for public internetaccess.



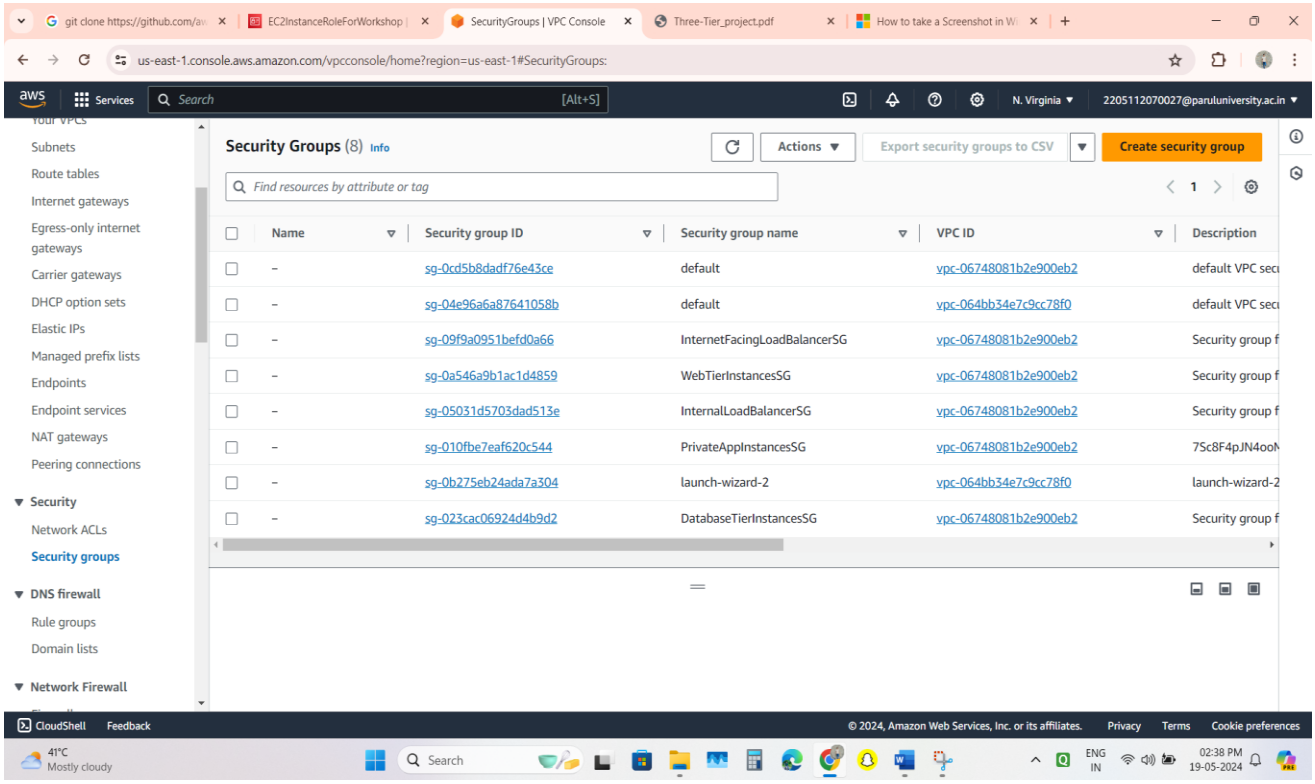
5 Setup NAT Gateway:

- Deploy a NAT Gateway for private subnet instances to access theinternet.
- While creating we will going to choose **Public Subnet AZ-1** and for Second NAT Gateway **Public Subnet AZ-2** in which to create the NATGateway.
- In Both NAT Gateways we will assign an Elastic IP.



6 Configure Security Groups:

- Defined 5 security groups to control inbound traffic to EC2 instances,RDS, and other resources.

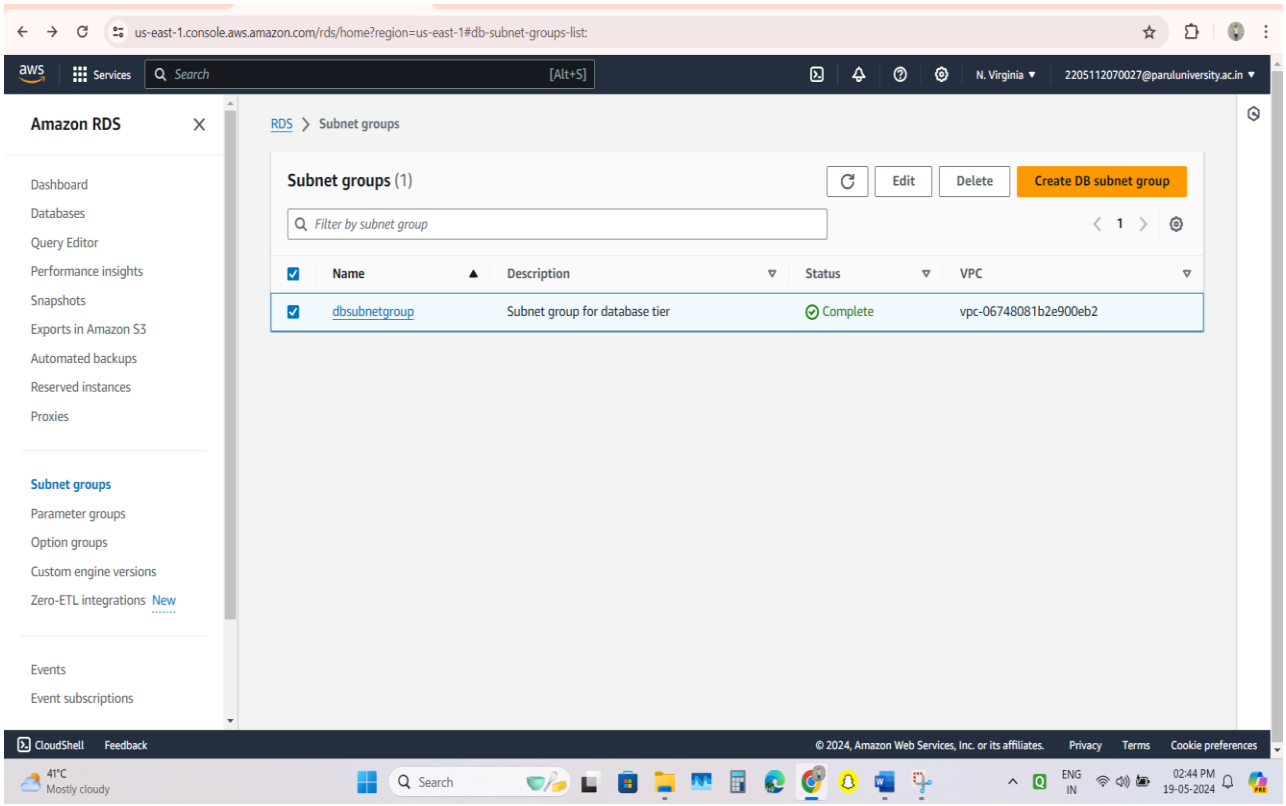


Part 2: Database Deployment

Step-by-Step Guide:

1. Creating Subnet Group:

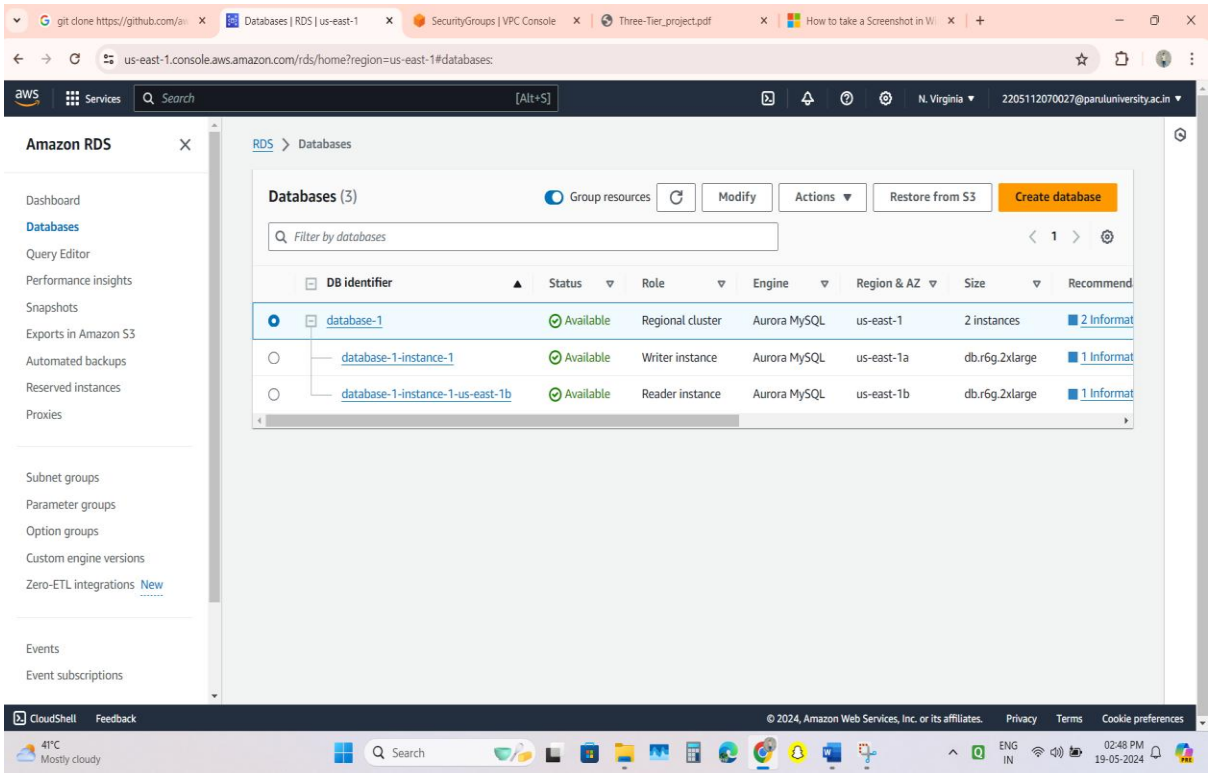
- After Going into the RDS service Click Subnet Group to create one andadd **2 Private-DB-Subnet (AZ-1 & AZ-2)** For The Database Tier for Added in Subnet Groups
- Make Sure to select Both Availability Zones (**1a & 1b**) in the Creation ofthe Subnet Group to ensure High Availability.



Deploying Database

1. Navigate to Databases on the left-hand side of the RDS dashboard andclick Create Database.
2. We will now go through several configuration steps. Start with a **Standard created** for this **MySQL-Compatible Amazon Aurora** database. Leave therest of the defaults in the **Engine options** as default.
3. Under the Templates section, select the Dev/Test since this is not being usedfor production now.
4. Under Settings Database cluster identifier, keep the default name database 1. We will keep the username as ‘admin, set a password to welcome-db.’,
5. Under the Cluster storage configuration section, we will keep Aurora Standard and keep the default option under Instance Configuration Next, under Availability and Durability, we will keep the option to create an Aurora Replica or reader node in a different availability zone as recommended along with our VPC under Connectivity.
6. We will also choose the subnet group we created earlier and select no forpublic access.

Now, let us set the security group we created for the database layer. We will not make any changes under password authentication because password authentication is always on. Click Create to create the database

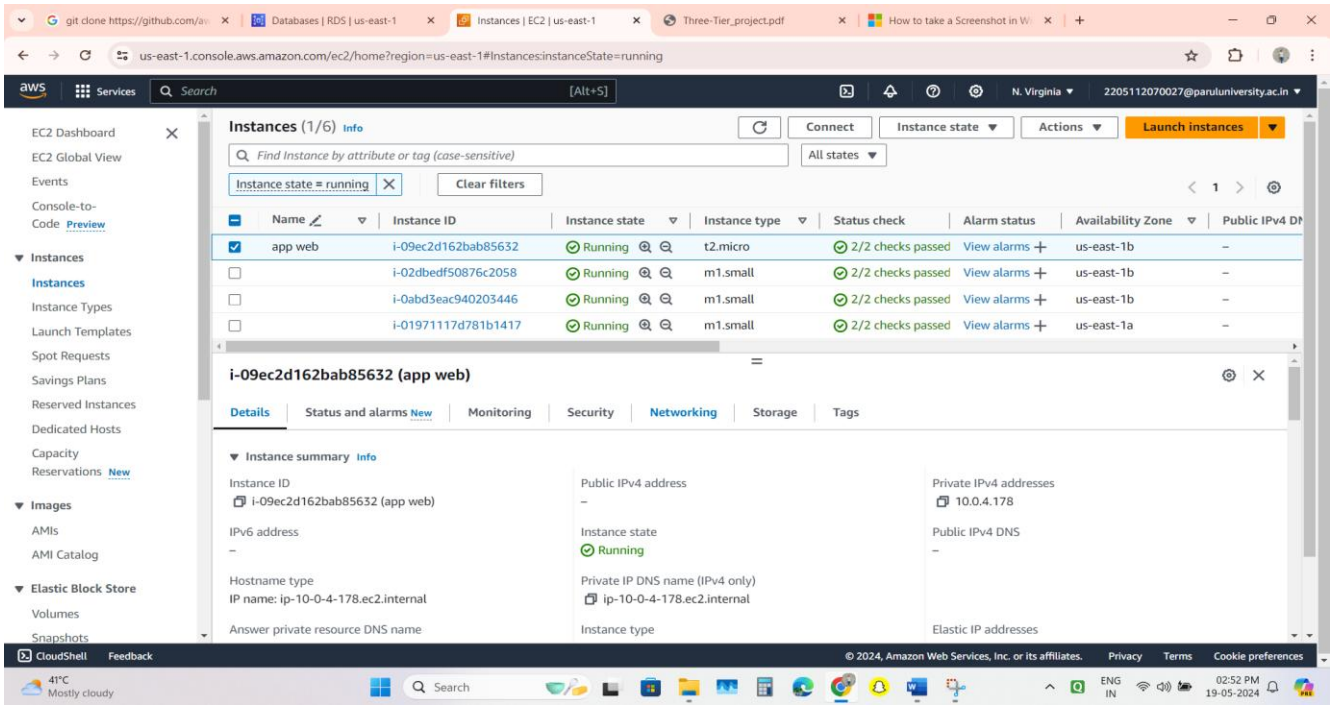


Part 3: App Tier Instance Deployment

Step-by-Step Guide:

1. Create App Tier Instance:

- Launch an EC2 instance for hosting the application layer.
- Select **Amazon Linux 2 AMI** and select free tier eligible **T.2 micro** asinstance type.
- Next, we will proceed without a key pair for architecture.
- In Subnet select **Private-Subnet-Az1** and under SG select **Private-instance-SG**
- In an additional setting attach the role which we have created at thestart.
- At last Launch the Instance.


git clone https://github.com/aw
Databases | RDS | us-east-1
Instances | EC2 | us-east-1
Three-Tier_project.pdf
How to take a Screenshot in W
us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#Instances:instanceState=running
aws
Services
Search
[Alt+S]
N. Virginia
2205112070027@paruluniversity.ac.in
EC2 Dashboard
EC2 Global View
Events
Console-to-Code Preview
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Capacity Reservations New
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Instances (1/6) Info
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
Instance state = running
Clear filters
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DN
app web
i-09ec2d162bab85632
Running
t2.micro
2/2 checks passed
View alarms
us-east-1b
i-02dbedf50876c2058
Running
m1.small
2/2 checks passed
View alarms
us-east-1b
i-0abd3eac940203446
Running
m1.small
2/2 checks passed
View alarms
us-east-1b
i-01971117d781b1417
Running
m1.small
2/2 checks passed
View alarms
us-east-1a
i-09ec2d162bab85632 (app web)
Details
Status and alarms New
Monitoring
Security
Networking
Storage
Tags
Instance summary Info
Instance ID
i-09ec2d162bab85632 (app web)
Public IPv4 address
–
Private IPv4 addresses
10.0.4.178
IPv6 address
–
Instance state
Running
Public IPv4 DNS
–
Hostname type
IP name: ip-10-0-4-178.ec2.internal
Private IP DNS name (IPv4 only)
ip-10-0-4-178.ec2.internal
Answer private resource DNS name
Instance type
Elastic IP addresses
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
41°C
Mostly cloudy
Search
ENG
IN
02:52 PM
19-05-2024

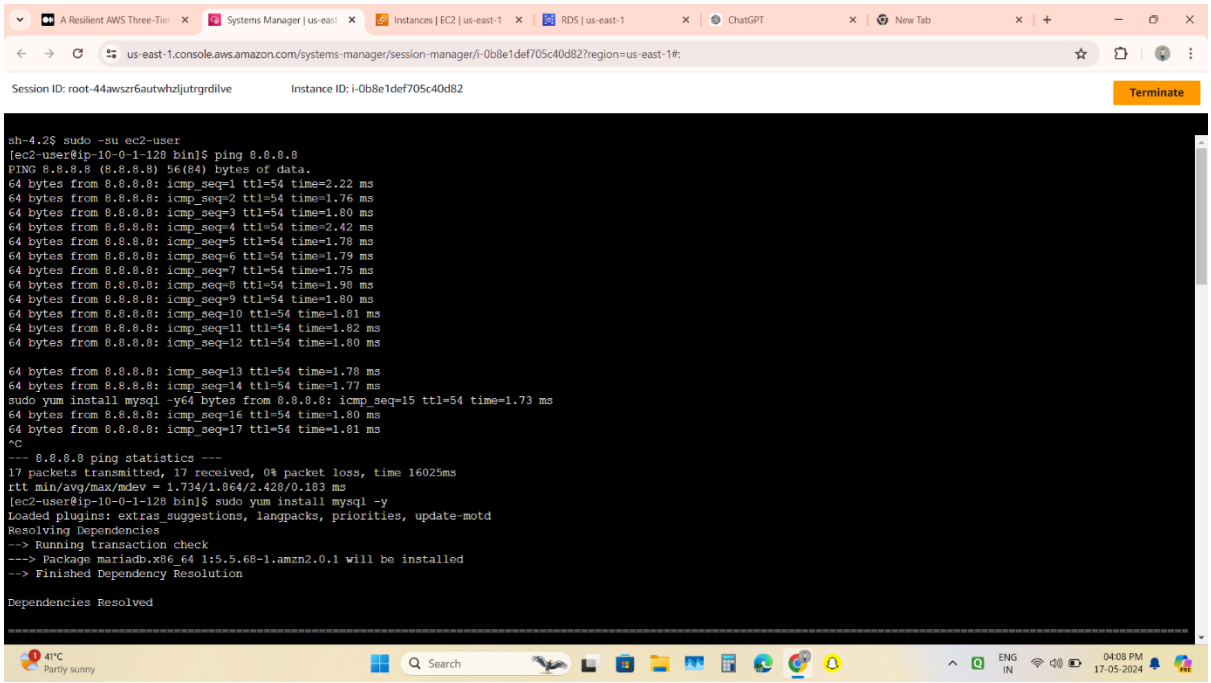
2 Configure Software Stack for Node.js Application:

- First Connect the instance through Session Manager.
- When you first connect to your instance, you will be logged in as ssm-user which is the default user. Switch to ec2-user

  `Sudo -su ec2-user`

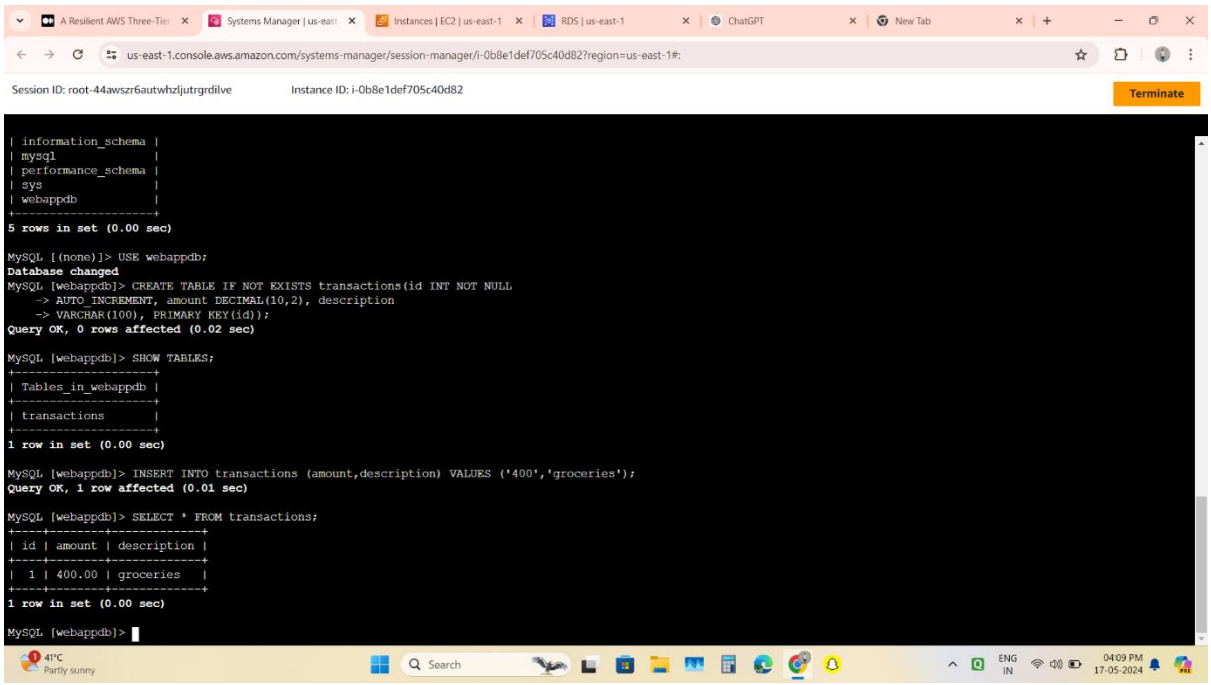
- Now check that you are connected to the Internet By

  `Ping 8.8.8.8`



3 Setup Database Schema:

- Configure the database schema and tables required by the application.
- Run All these Commands for configuring the Database.



Configure App Instance

By Running these commands in Session Manager

Step	Command/Instruction
1. Upload app-tier folder to S3 bucket	`aws s3 cp app-tier/ s3://BUCKET_NAME/app-tier/ --recursive`
2. Install NVM	`curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/insta`
	`source ~/.bashrc`
3. Install Node.js and use version 16	`nvm install 16`
	`nvm use 16`
4. Install PM2	`npm install -g pm2`
5. Download code from S3 bucket to instance	`aws s3 cp s3://BUCKET_NAME/app-tier/ app-tier --recursive`
6. Navigate to app directory	`cd ~/app-tier`
7. Install dependencies	`npm install`
8. Start the app with PM2	`pm2 start index.js`
9. Check if the app is running	`pm2 list`
10. View logs if necessary	`pm2 logs`
11. Configure PM2 startup	`pm2 startup`
	Copy and paste the command provided by the output in your terminal,

Test App Tier

Now let's run a couple tests to see if our app is configured correctly and can retrievedata from the database.

To check out our health check endpoint, copy this command into your SSM terminal.This is our simple health check endpoint that tells us if the app is simply running. **sudo curl** **http://localhost:4000/health**

The command responded with the following message: "This is the health check"which means our health check is running correctly as indicated below.

Next, let's test our database connection. We can do that by hitting the followingendpoint locally:

curl **http://localhost:4000/transaction**

Received the following response after executing the command:

The two above responses indicate that our networking, security, database, and appconfigurations are correct. Our app layer is fully configured and ready to go.

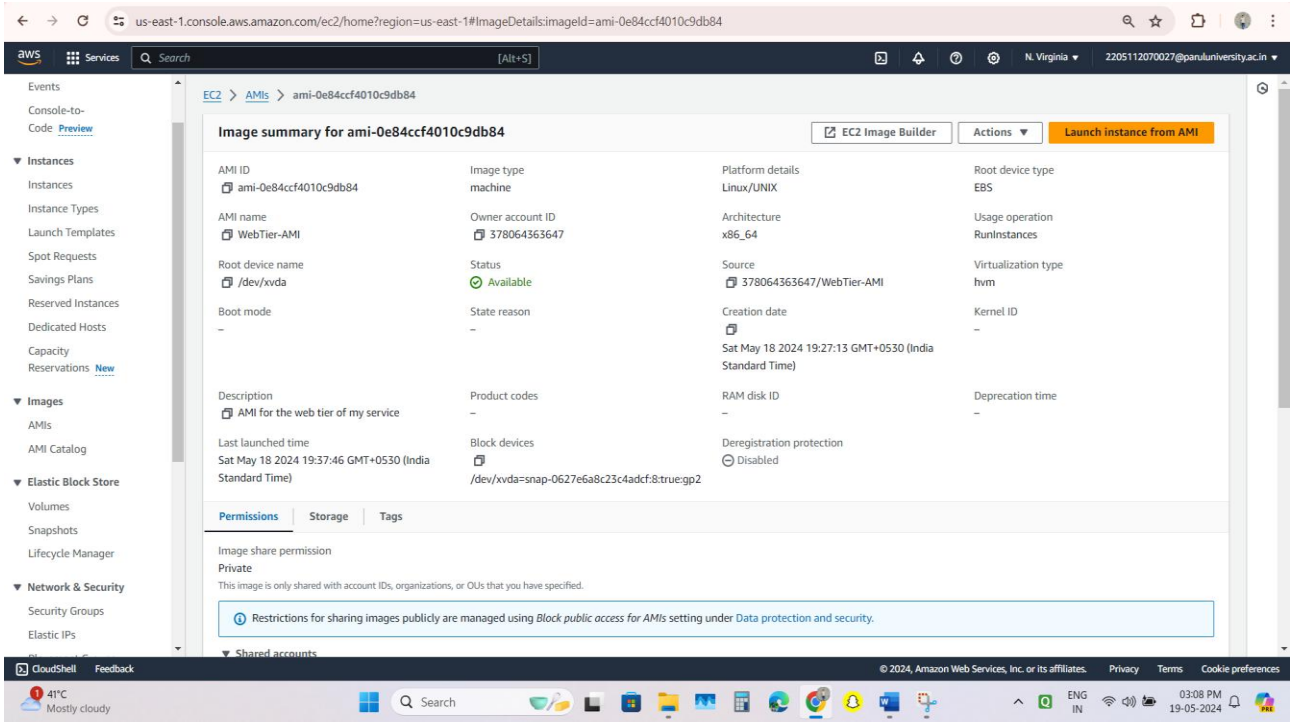
Part 5: Internal Load Balancing and Auto Scaling

In this section, we will create an Amazon Machine Image (AMI) of the app tier instance we just created,and use that to set up autoscaling with a load balancer in order to make this tier highly available.

App Tier AMI

Let's navigate to **Instances** on the left-hand side of the EC2 dashboard. Select the app tier instance wecreated and under **Actions** select **Image and Templates**. Click **Create Image**.

Let's give the image a name and description and then click **Create image**. This will take a fewminutes, but if you want to monitor the status of image creation you can see it by clicking **AMIs** under **Images** on the left-hand navigation panel of the EC2 dashboard.



Target Group

While the AMI is being created, let's go ahead and create our target group to use with the load balancer.On the EC2 dashboard, navigate to **Target Groups** under **Load Balancing** on the left-hand side.
Click on **Create Target Group**.

The purpose of forming this target group is to use our load balancer so it may balance traffic across ourprivate app tier instances. Let's select Instances as the target type and give it a name.

Let's set the protocol to **HTTP** and the port to 4000. Remember that this is the port our Node.js app isrunning on. Select the VPC we've been using thus far, and then change the health check path to
be **/health** to indicate the health check endpoint of our app and click **Next**.
We'll **NOT** register any targets for now, so let's just skip that step, click **Next** and create the targetgroup.

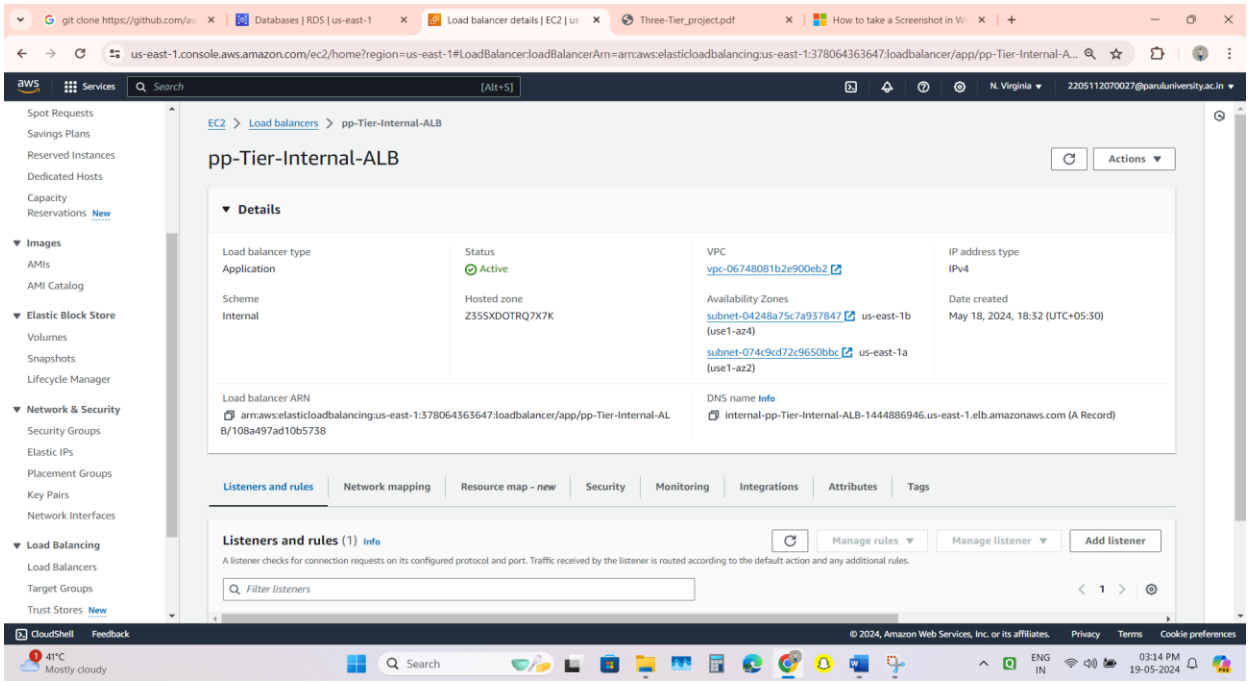
Internal Load Balancer

We're going to create an internal load balancer for our three-tier. On the left-hand side of the EC2dashboard select **Load Balancers** under **Load Balancing** and click **Create Load Balancer**.

We'll be using an **Application Load Balancer** for our **HTTP** traffic, give it a name (App-Tier-Internal-lg), select **Internal** under Scheme, and click the create button for that option.

Let's select the correct network configuration for our VPC and private subnets.

Next, select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our **target group** that we just created. Let'sselect it from the dropdown list, and create the load balancer.



Internal Load Balancer is created with two availability zones.

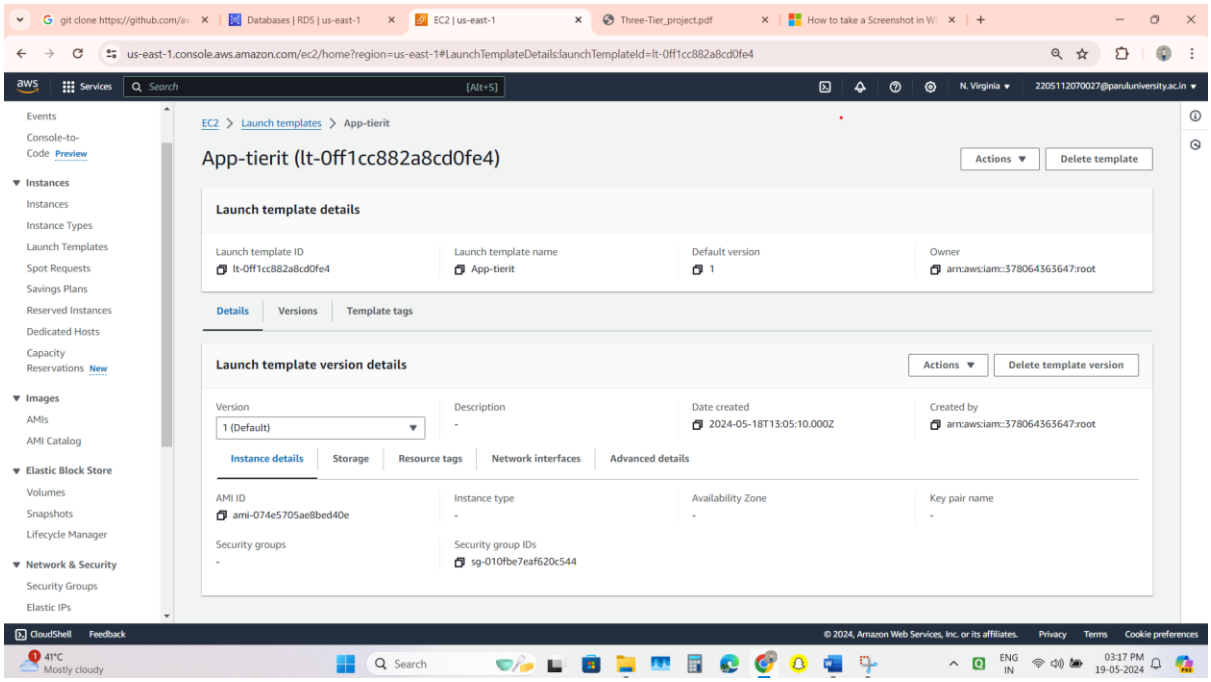
Launch Template

Now, we need to create a Launch template with the AMI we created earlier before we configure Auto Scaling. On the left side of the EC2 dashboard, let's navigate to **LaunchTemplate** under **Instances** and click **Create Launch Template**.

Name the Launch Template, and then under **Application and OS Images** select **MY AMI** and include the app tier AMI we previously created.

Under **Instance Type** select t2.micro. For **Key pair** and **Network Settings** don't include it in the template. We don't need a key pair to access our instances and we'll be setting the network informationin the autoscaling group.

Set the correct security group for our app tier, and then under **Advanced details** use the same IAMinstance profile we have been using for our EC2 instances.



Auto Scaling

We will now create the Auto Scaling Group for our app instances. On the left-hand side of the EC2 dashboard, navigate to **Auto Scaling Groups** under **Auto Scaling** and click **Create Auto ScalingGroup**.

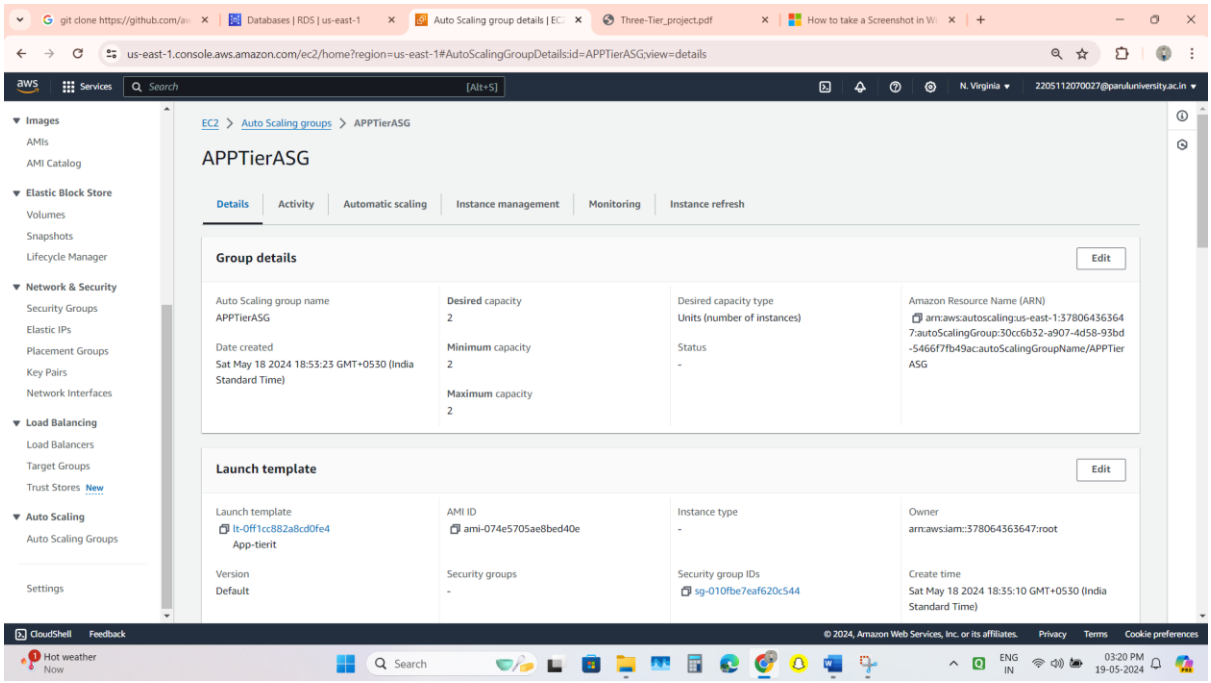
Let's give our Auto Scaling group a name, and then select the Launch Template we just created andclick **Next**.

Next, we're going to select on the **Choose instance launch options** page, set our VPC, and the private instance subnets for the app tier, and continue.

For this next step, we'll attach this Auto Scaling Group to the Load Balancer we just created by selectingthe existing load balancer's target group from the dropdown. Then, click next.

We're going to set the desired 2, minimum 2, and maximum 2 capacity of our Auto Scaling G**roup sizeand Scaling policies.** Review and then Create Auto Scaling Group.

Let's find out if our internal load balancer and autoscaling group are configured correctly. The autoscaling group will spin up 2 new app tier instances. We can test if this is working correctly by deleting one of our new instances manually and waiting to see if a new instance is booted up to replaceit.



Part 6: Web Tier Instance Deployment

In this section, we will deploy an EC2 instance for the web tier and make all necessary softwareconfigurations for the NGINX web server and React.js website.

- Update Config File

Before we create and configure the web instances (web tier), let's modify the **application- code/nginx.conf** file from the repo we previously downloaded. First, navigate to your internalload balancer's details page and copy the DNS entry into a notepad and save.

Next, open the folder where the repo was downloaded on your command prompt terminal toupdate the **application-code/nginx.conf** file. Edit the file using this command:

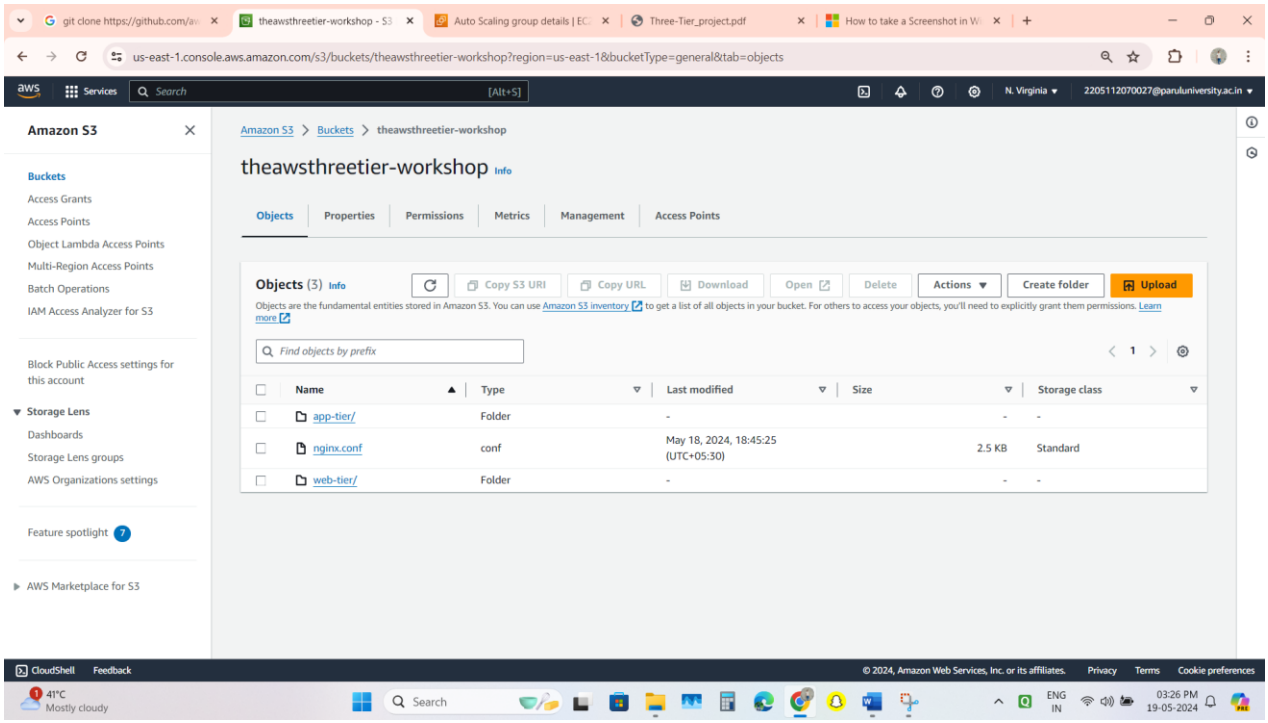
vim nginx.conf

```
:set nu
```

Scroll down to **line 58** and replace [INTERNAL-LOADBALANCER-DNS] with your internal load balancer's DNS entry. Save the file by pressing the '**esc keypad, shift : + wq**' without the single quotes.

```
35      # for more information.
36      include /etc/nginx/conf.d/*.conf;
37
38      server {
39          listen       80;
40          listen       [::]:80;
41          server_name  _;
42
43          #health check
44          location /health {
45          default_type text/html;
46          return 200 "<!DOCTYPE html><p>Web Tier Health Check</p>\n";
47          }
48
49          #react app and front end files
50          location / {
51          root   /home/ec2-user/web-tier/build;
52          index index.html index.htm
53          try_files $uri /index.html;
54          }
55
56          #proxy for internal lb
57          location /api/{
58                  proxy_pass http://internal-App-Tier-Internal-LB-1960878858.us-east-1.elb.amazonaws.com:80/;
59          }
```

Now, let's upload the '**nginx.config'** file **and** the **application-code/web-tier** folder to the s3 bucket we created for this lab. Navigate back to the Amazon S3 dashboard, click '**Buckets'** on theleft hand, and select our bucket | **Upload.**



Web Instance Deployment

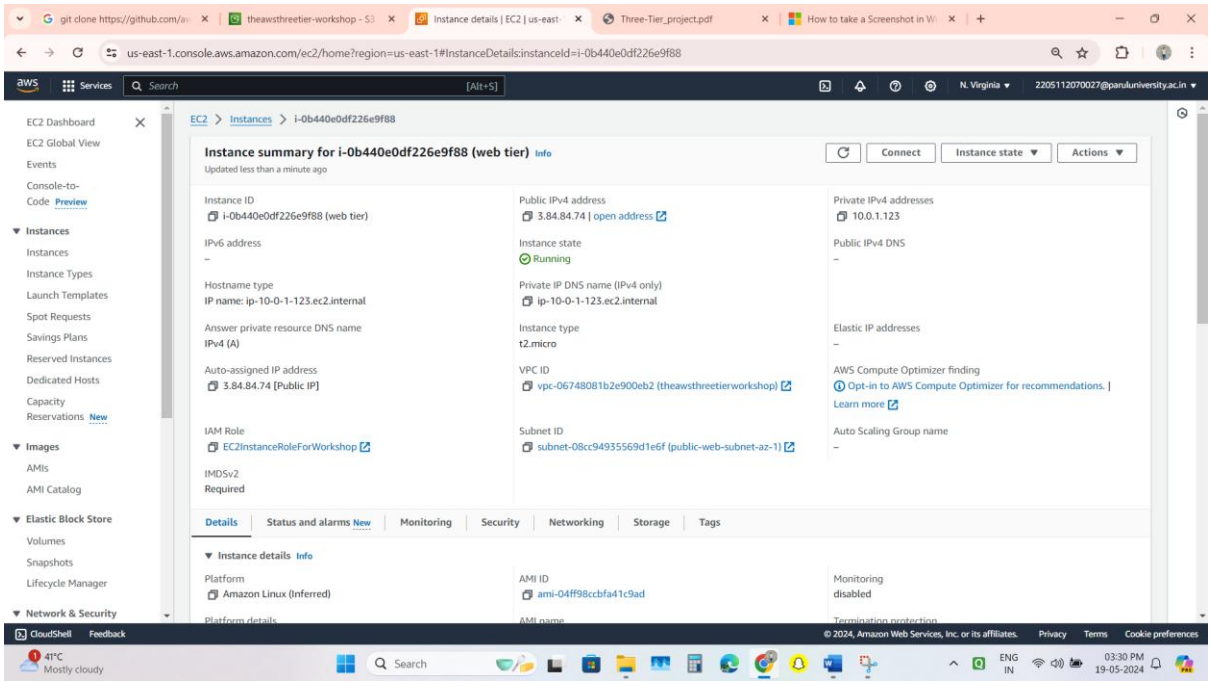
Using the EC2 service dashboard, click on **Instances** on the left-hand side and then **Launch Instances** to start the process.

Let's name our app instance **'Web-Tier'** for the three-tier architecture and select the **Amazon Linux2 AMI** for our application and operating system image.

We'll be using the free tier eligible **T.2 micro** instance type so let's select that to proceed. Even though 'Proceed without a key is (Not recommended)', we'll proceed without a key pair for architecture.

Earlier we created a security group for our public web layer instances, so go ahead and select that alongwith our VPC and the '**public-web-subnet-az-1'** under **Network settings**.

We'll keep the default configuration settings for **Configure storage**, and in advanced details add **IAM role ec2-three-tier-access-role** that we have created at starting, but review the **Summary**,and click **Launch** instance to create the '**appLayer**' instance.



Connect to Instance

Let's follow the same steps we used to connect to the first app instance and change the user to **ec2-user**. Test connectivity here via ping as well since this instance should have internet connectivity:

Sudo -su ec2-user
Ping 8.8.8.8

Note: If you don't see a transfer of packets then you'll need to verify your route tables attached to the

subnet that your instance is deployed in.

I was able to ping the Google server IP Address 8.8.8.8 from the App Tier instance command promptterminal successfully.

Configure Web Instance

We now need to install all of the necessary components needed to run our front-end application. Let's start by installing NVM and node on the instance

curl -o- https://raw.githubusercontent.com/nvm- sh/nvm/v0.38.0/install.sh | bash

source ~/.bashrc nvm install 16 nvm use 16

Now we need to download our web tier code from our s3 bucket:

cd ~/

aws s3 cp s3://BUCKET_NAME/web-tier/ web-tier --recursive

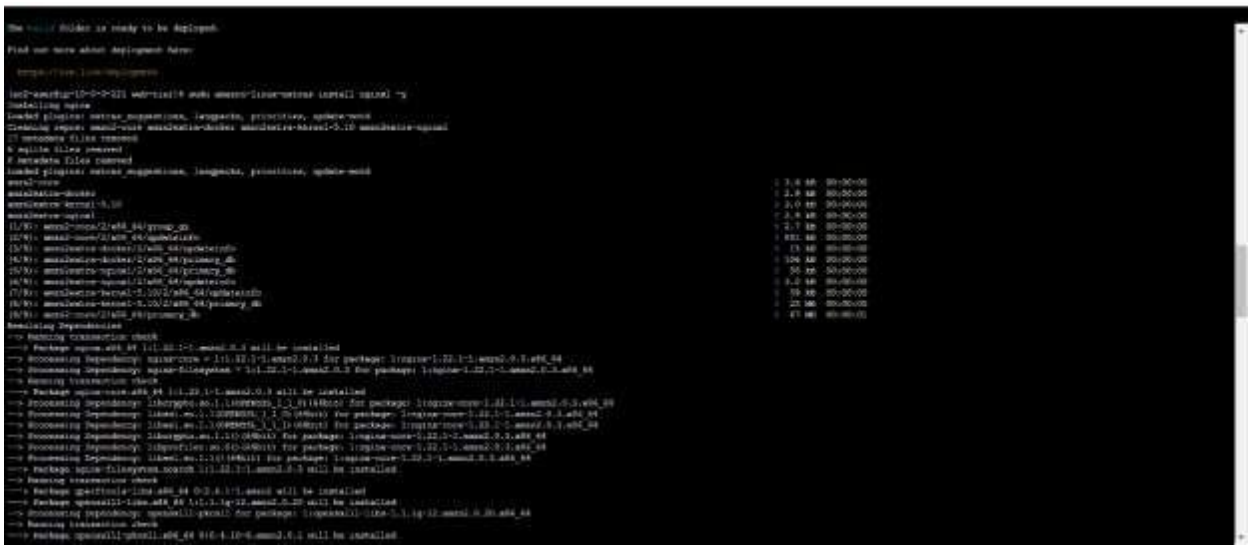
Replace [BUCKET_NAME] with your bucket name.

Navigate to the web-layer folder and create the build folder for the react app so we can serve our code using the below commands:

cd ~/web-tier npm install npm run build

NGINX can be used for different use cases like load balancing, content caching etc, but we will be using it as a web server that we will configure to serve our application on port 80, as well as help direct our API calls to the internal load balancer. Let's run the below command to proceed

sudo amazon-linux-extras install nginx1 –y



We will now have to configure NGINX. Navigate to the Nginx configuration file with the followingcommands and list the files in the directory:

cd /etc/nginx

ls

Let’s update the **‘nginx.conf’** file with the one we uploaded to S3 bucket. We’ll remove the file and replace the bucket name below:

sudo rm nginx.conf

sudo aws s3 cp s3://BUCKET_NAME/nginx.conf .

The bucket name should be replaced.

Let’s restart Nginx with the following command:

sudo service nginx restart

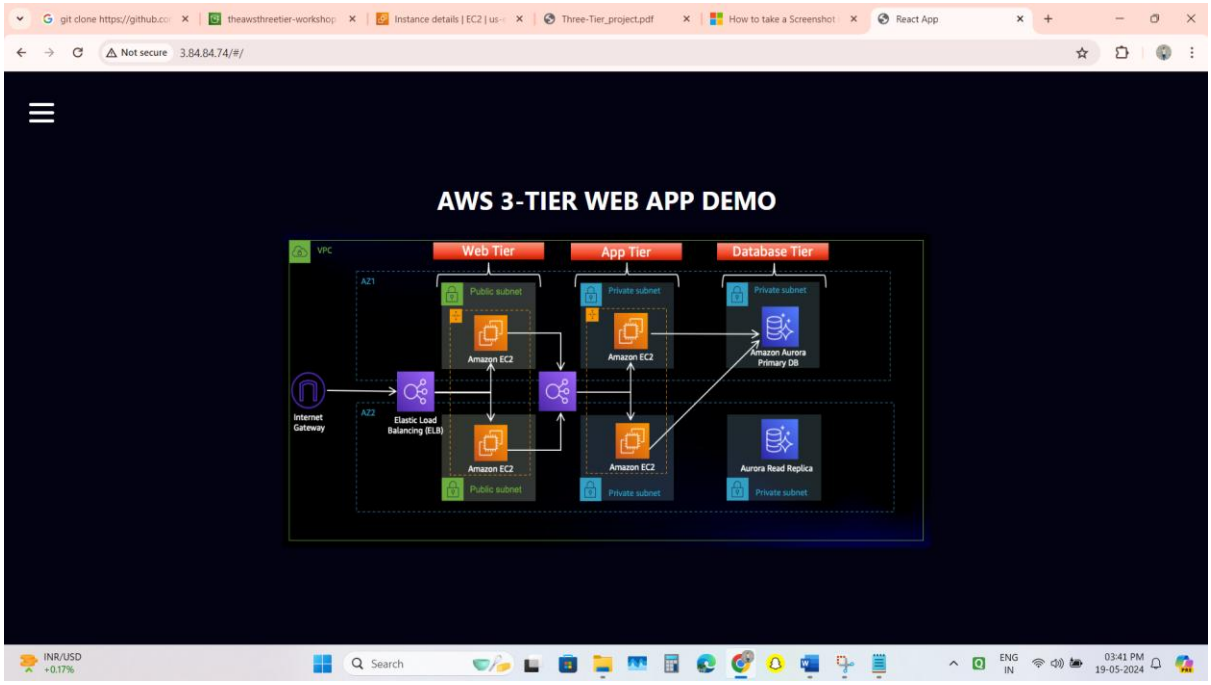
Let’s make sure Nginx has permission to access our files by executing the command:

chmod -R 755 /home/ec2-user

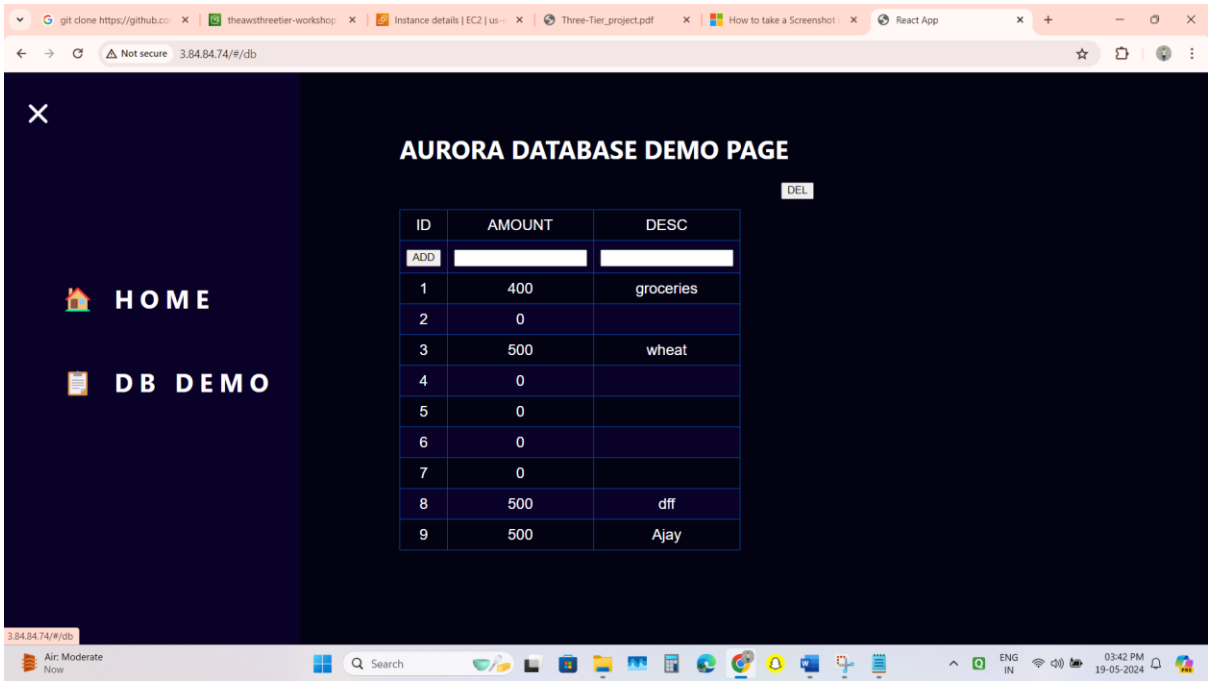
And to make ensure the service starts on boot, run this command:

sudo chkconfig nginx on

Now let’s copy and plug in the public IP of our web tier instance to see our website. The public IP can be found on the App Tier Instance details page on the EC2 dashboard. Voila, the website is working correctly.



Let’s do the same for our database tier and if it’s connected and working correctly.



Part 7: External Load Balancer and Auto Scaling

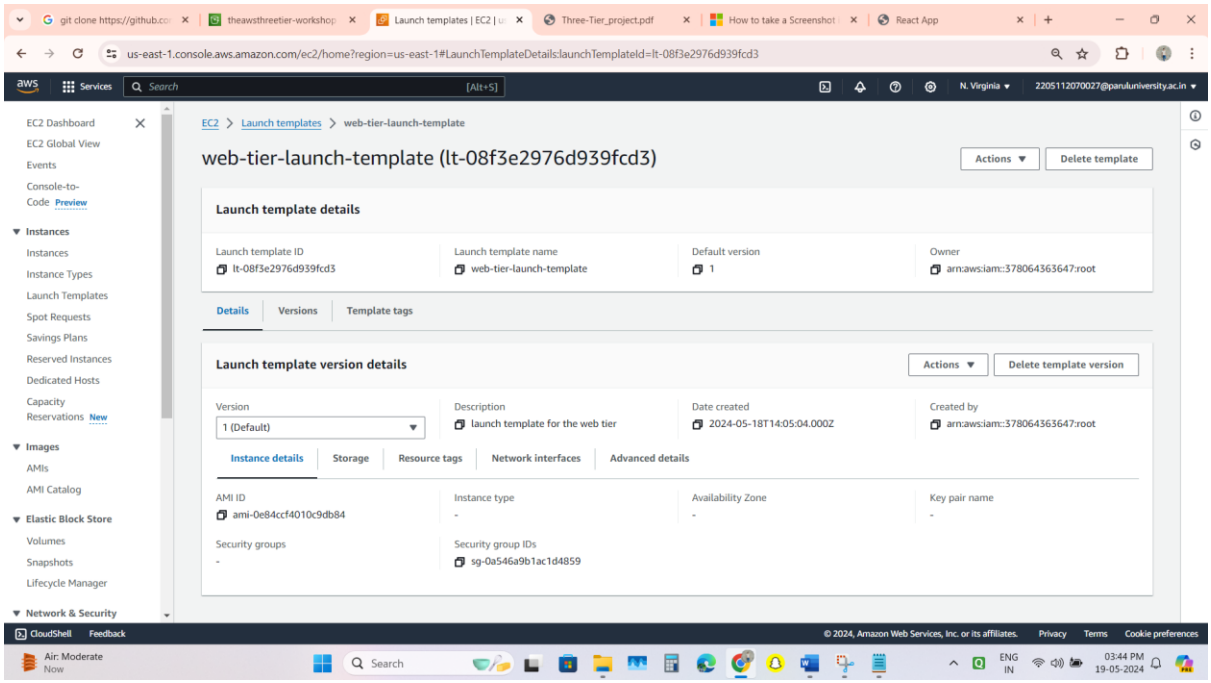
In this section of the workshop, we will create an Amazon Machine Image (AMI) of the web tier instance we just created, and use that to set up autoscaling with an external facing load balancer in order to make this tier highly available.

Web Tier AMI

Let’s go to Instances on the left-hand side of the EC2 dashboard. Select the web tier instance we created and under Actions select Image and Templates. Click Create Image.

We’ll give the image a name and description and then click Create image. This will take a few minutes, but if you want to monitor the status of image creation you can see it by

clicking AMIs under Images on the left-hand navigation panel of the EC2 dashboard.



Target Group

On the EC2 dashboard let's navigate to Target Groups under Load Balancing on the left-hand side. Click on Create Target Group.

The purpose of forming this target group is to use our load balancer so it may balance traffic across our public web tier instances. Select Instances as the target type and give it a name.

Let's set the protocol to HTTP and the port to 80. Remember this is the port NGINX is listening on.

Select the VPC we've been using thus far, and then change the health check path to be /health. Click Next.

Go ahead and create the target group. We are NOT going to register any targets for now.

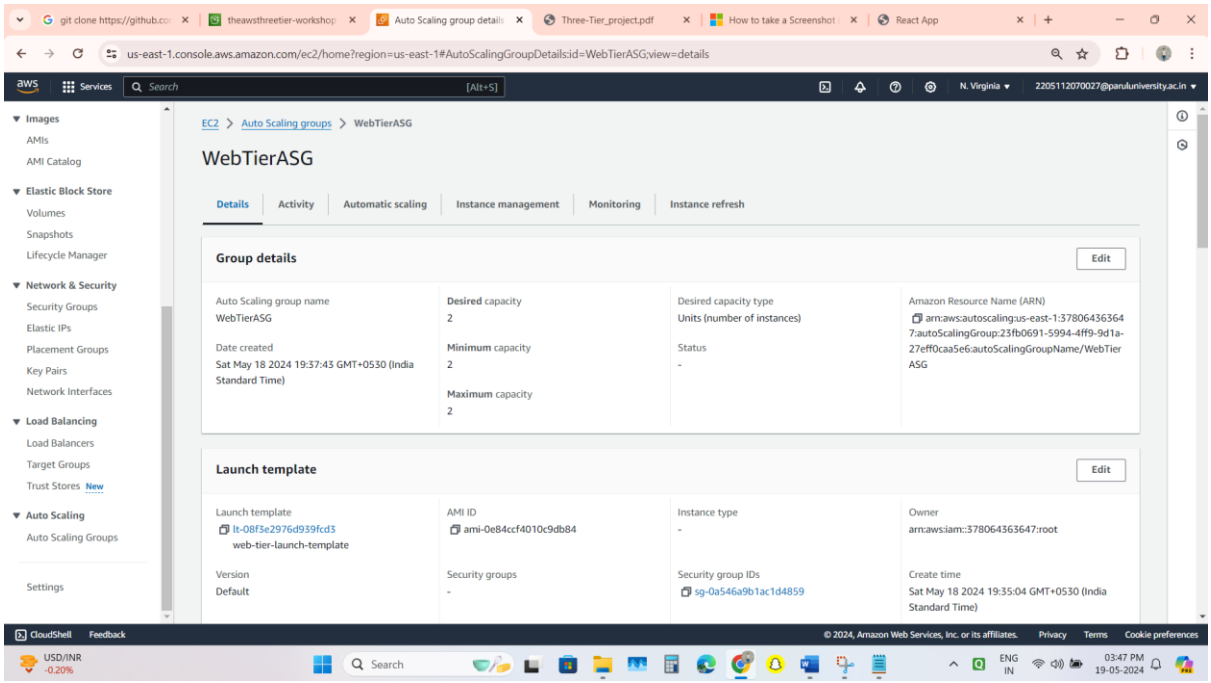
Internet Facing Load Balancer

On the left-hand side of the EC2 dashboard select Load Balancers under Load Balancing and click Create Load Balancer.

Select Application Load Balancer for our HTTP traffic. Click the Create button for that option. After giving the load balancer a name, be sure to select internet facing since this one will not be public facing, but rather it will route traffic from our web tier to the app tier.

Select the two public subnets to proceed.

Select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our target group that we just created, so select it from the dropdown, and create the load balancer.



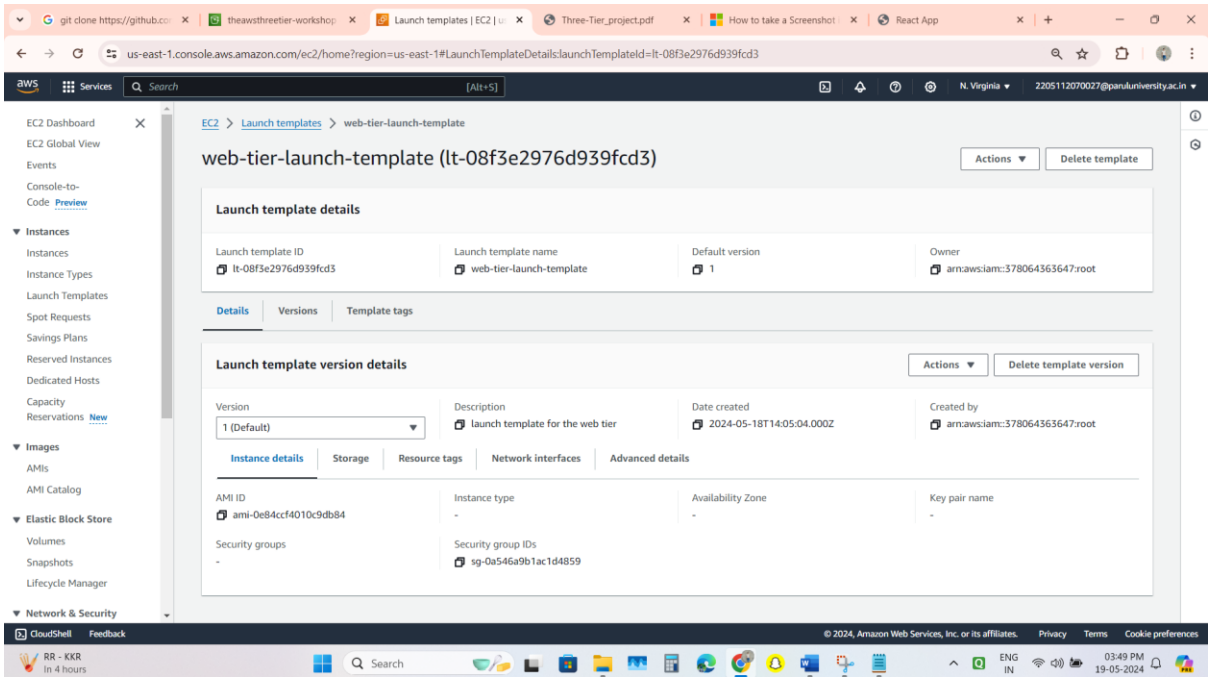
Launch Template

Before we configure Auto Scaling, we need to create a Launch template with the AMI we created earlier.On the left-hand side of the EC2 dashboard navigate to **Launch Template** under **Instances** and click **Create Launch Template**.

Let's name our Launch Template, and then under **Application and OS Images** include the app tierMY AMI you created.

Let's set the correct security group for our web tier, and then under **Advanced details** use the sameIAM instance profile we have been using for our EC2 instances.

Accept the rest of the default configuration settings and click **Create launch template.**



Auto Scaling

Once we have got the Launch Template created, let's go ahead to create the Auto Scaling Group for ourweb instances. On the left side of the EC2 dashboard navigate to **Auto Scaling Groups** under **AutoScaling** and click **Create Auto Scaling group**.

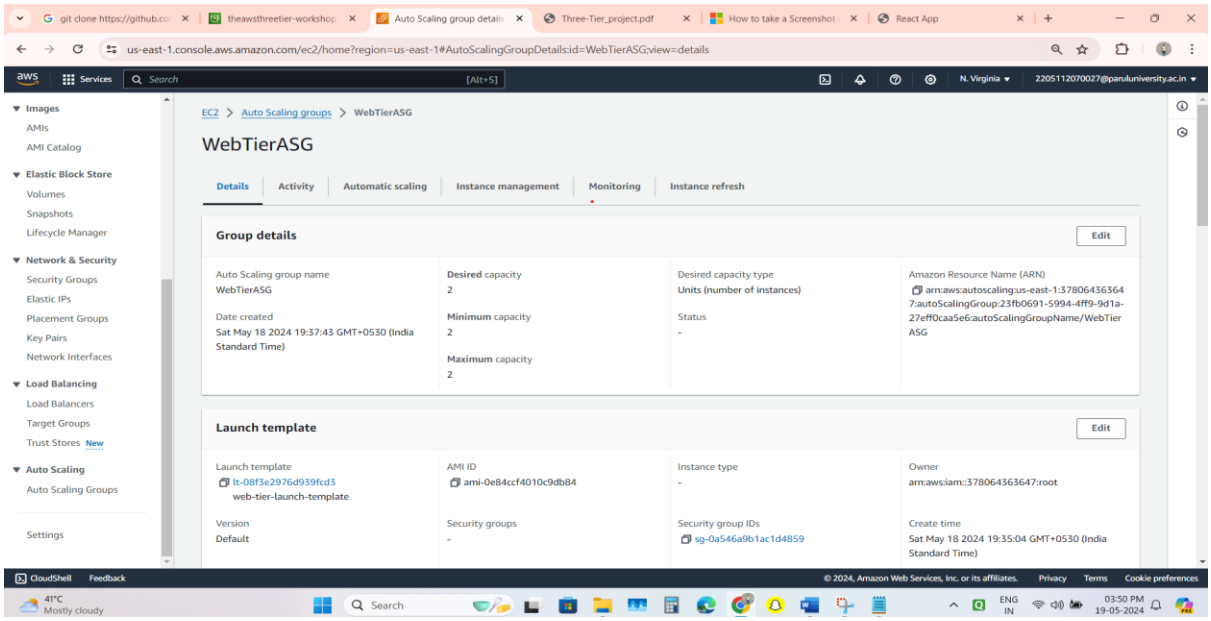
Let's give our Auto Scaling group a name, and then select the Launch Template we just created and click next.

On the **Choose instance launch options** page, let's set our VPC, and the public subnets for the web

tier and proceed to the next step.

For this next step, we're going to attach the Auto Scaling Group to the Load Balancer we just created by selecting the existing web-tier load balancer's target group from the dropdown. Then, click next.

For **Configure group size and scaling policies**, set desired, minimum and maximum capacityto **2**. Click Next (3x) to review and then Create Auto Scaling Group.



Let's test both our external load balancer and autoscaling group to see if they are configured correctly. The autoscaling group is spinning up 2 new web tier instances. Let's head over to the EC2 instances dashboard and delete one manually and wait to see if a new instance is booted up to replace it. Voila, itworked as intended!

Let's test if our entire architecture is working by plugging in our external facing load balancer, DNS name into your browser.