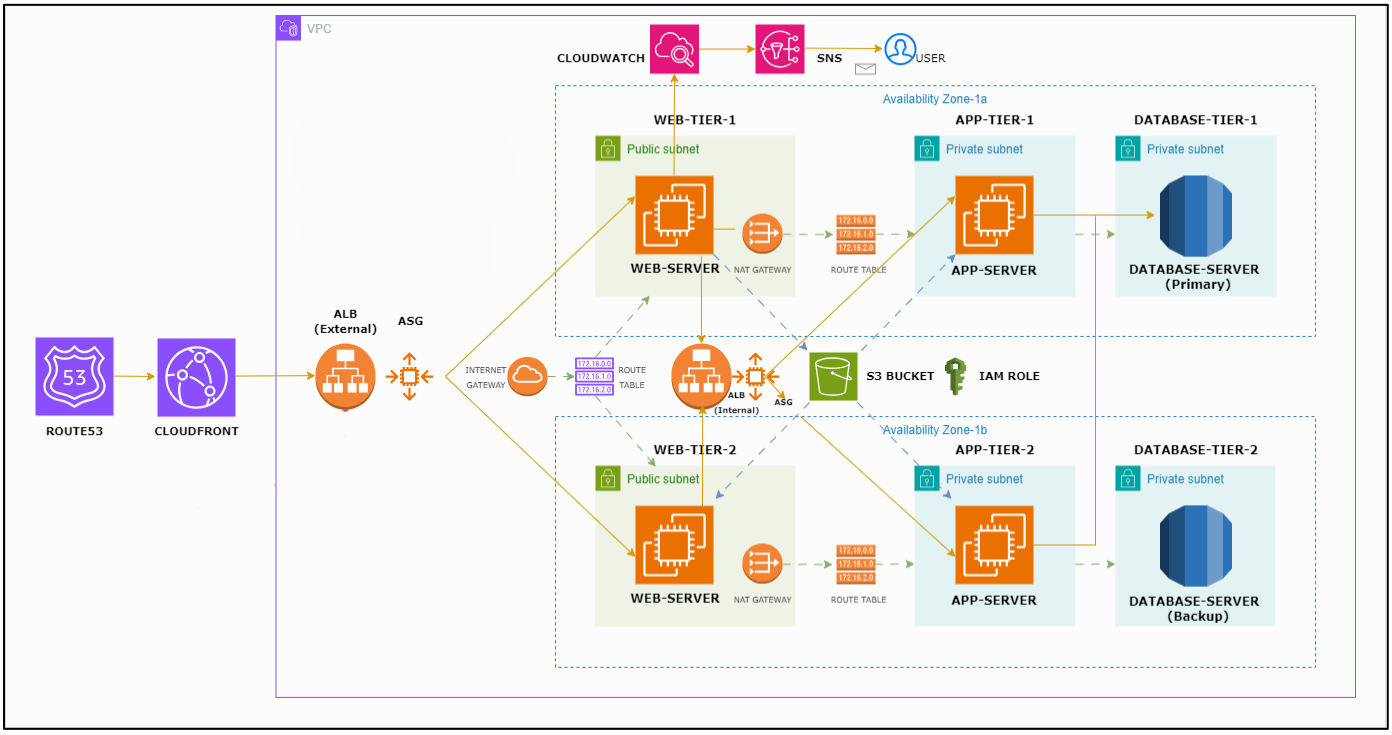
UDDHAV BAPU GUND
AWS Project
AWS Three Tier Web Architecture

◉ INTRODUCTION:

Architecture Overview:



‣ Designed and deployed a **scalable, fault-tolerant web application** using AWS **3-Tier Architecture** across **multiple availability zones** to ensure high availability.
‣ Developed a **multi-layer cloud infrastructure**, separating **presentation, application, and database layers** for improved security, performance, and scalability.
‣ Configured **VPC, subnets, security groups, IAM (attached role to EC2 for application code access), and WAF** to enforce secure access and protection. Deployed **EC2 instances with Auto Scaling Groups (ASG)** and distributed traffic using **Application Load Balancer (ALB)** for seamless scalability.
‣ Used **S3** to store source code and **RDS Aurora** for a highly available and scalable database solution. Set up **CloudWatch** for monitoring and integrated **SNS alerts** for proactive issue resolution.
‣ Optimized content delivery using **Route 53** for domain management and **CloudFront** for caching and acceleration, improving response times and user experience.
‣ Enhanced **availability, security, automation, and cost-efficiency**, ensuring seamless application performance across multiple AWS availability zones.

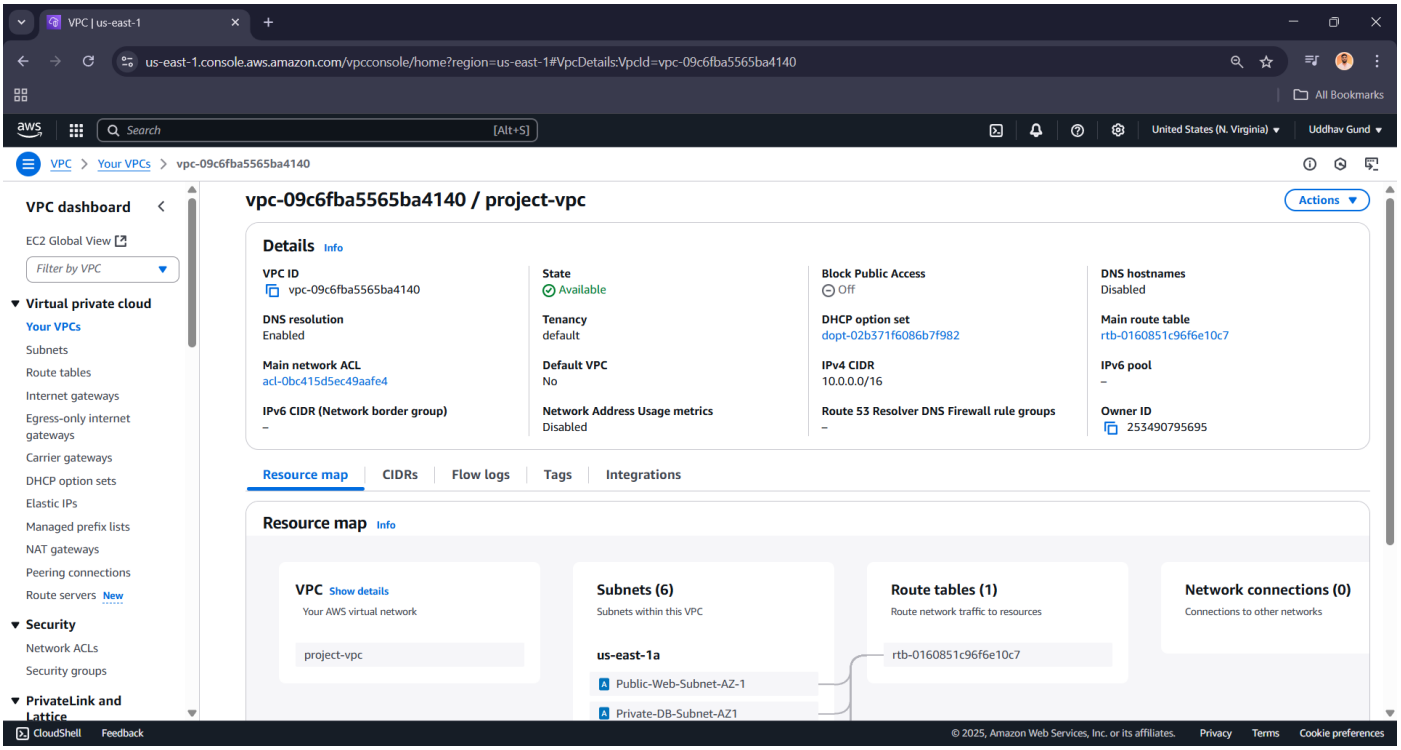
Tools & Technologies :

- VPC
- IAM
- EC2 (Installed Nginx, Node.js, and React),
- ASG,ALB,
- S3,
- Aurora DB
- CloudWatch
- SNS
- Route 53, CloudFront
- WAF

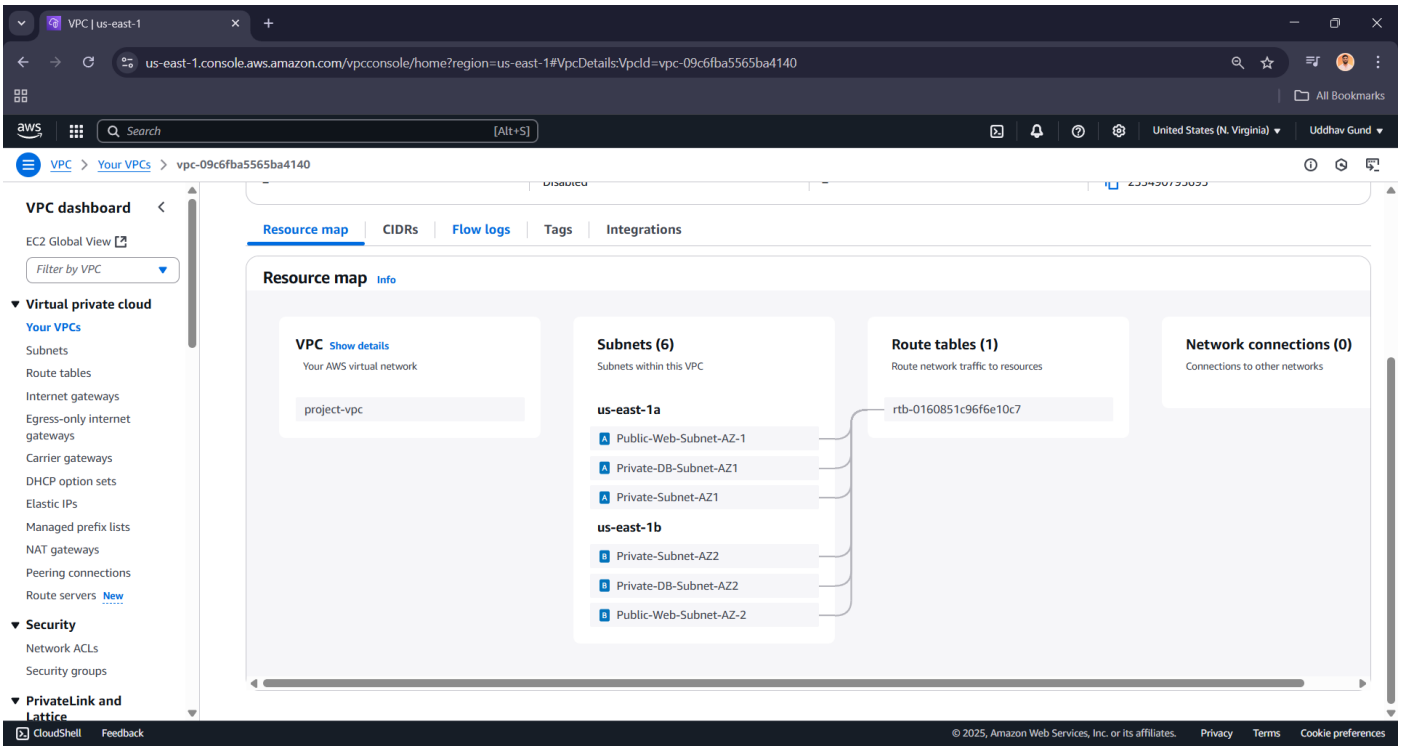
Part 1: Networking and Security

- Created an isolated network with the following components:
  - VPC
  - Subnets
  - Route Tables
  - Internet Gateway
  - NAT gateway
  - Security Groups

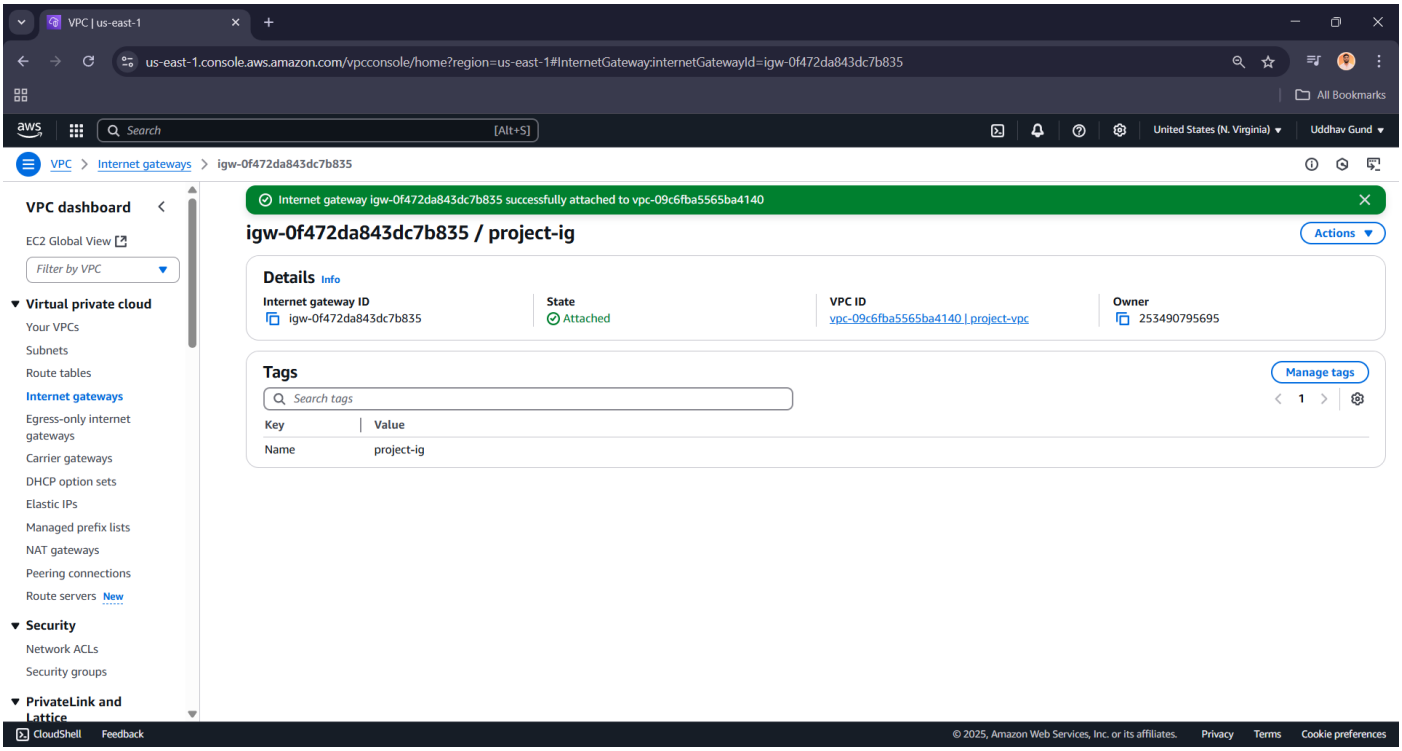
VPC



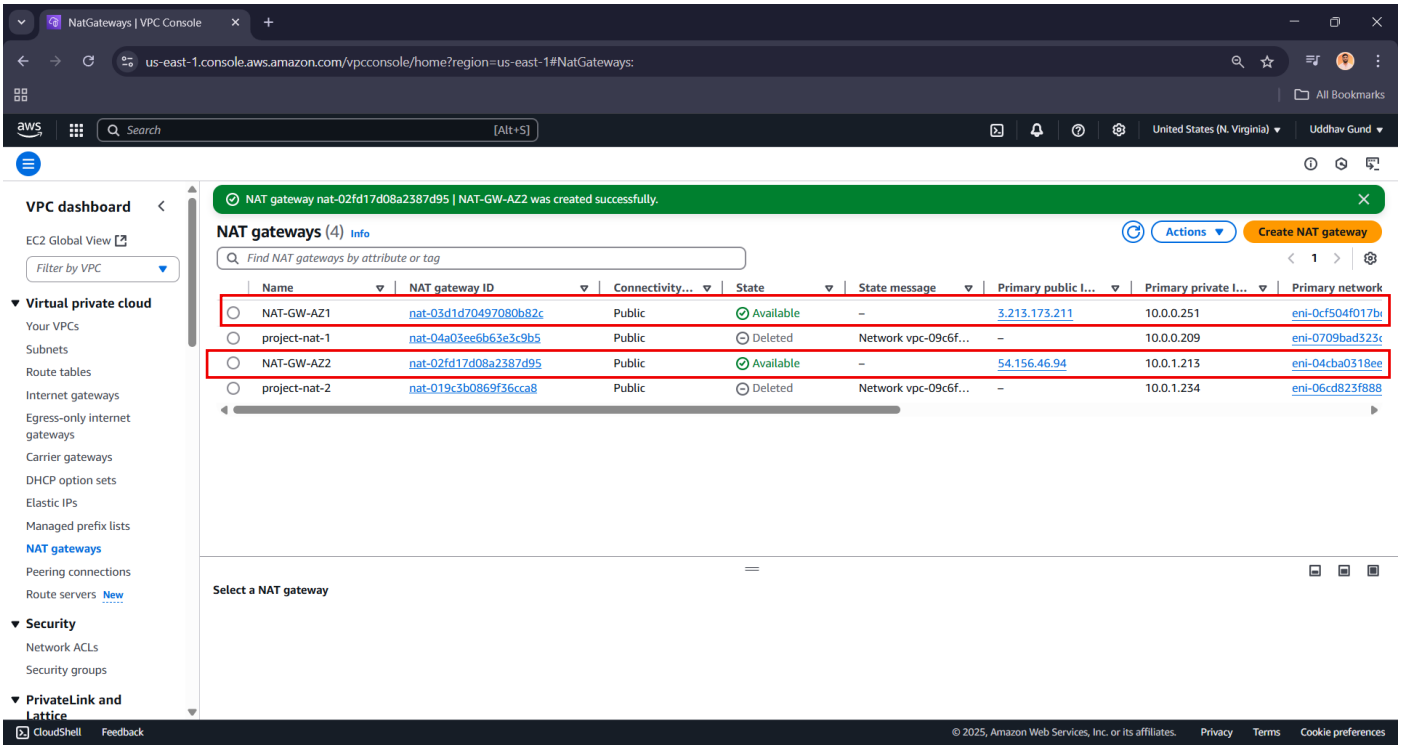
Subnet



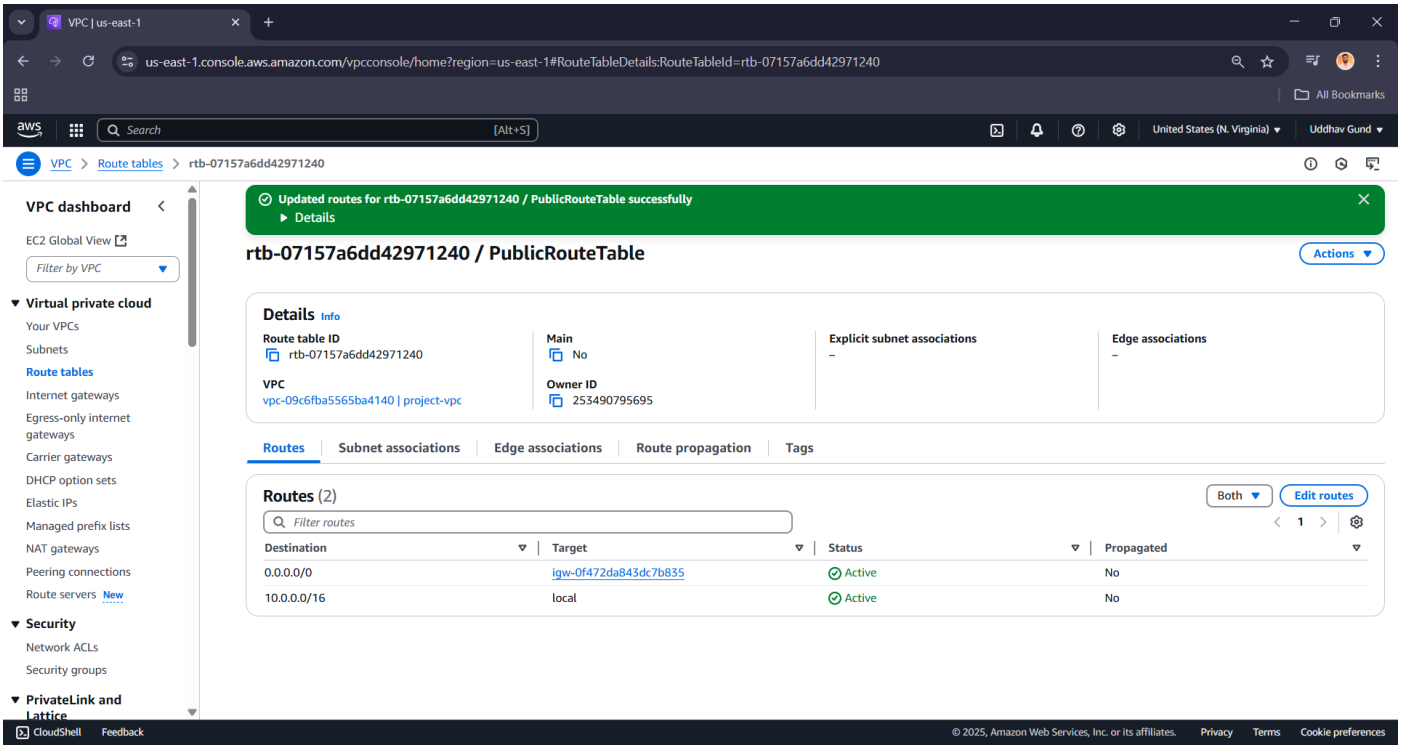
Internet Gateway



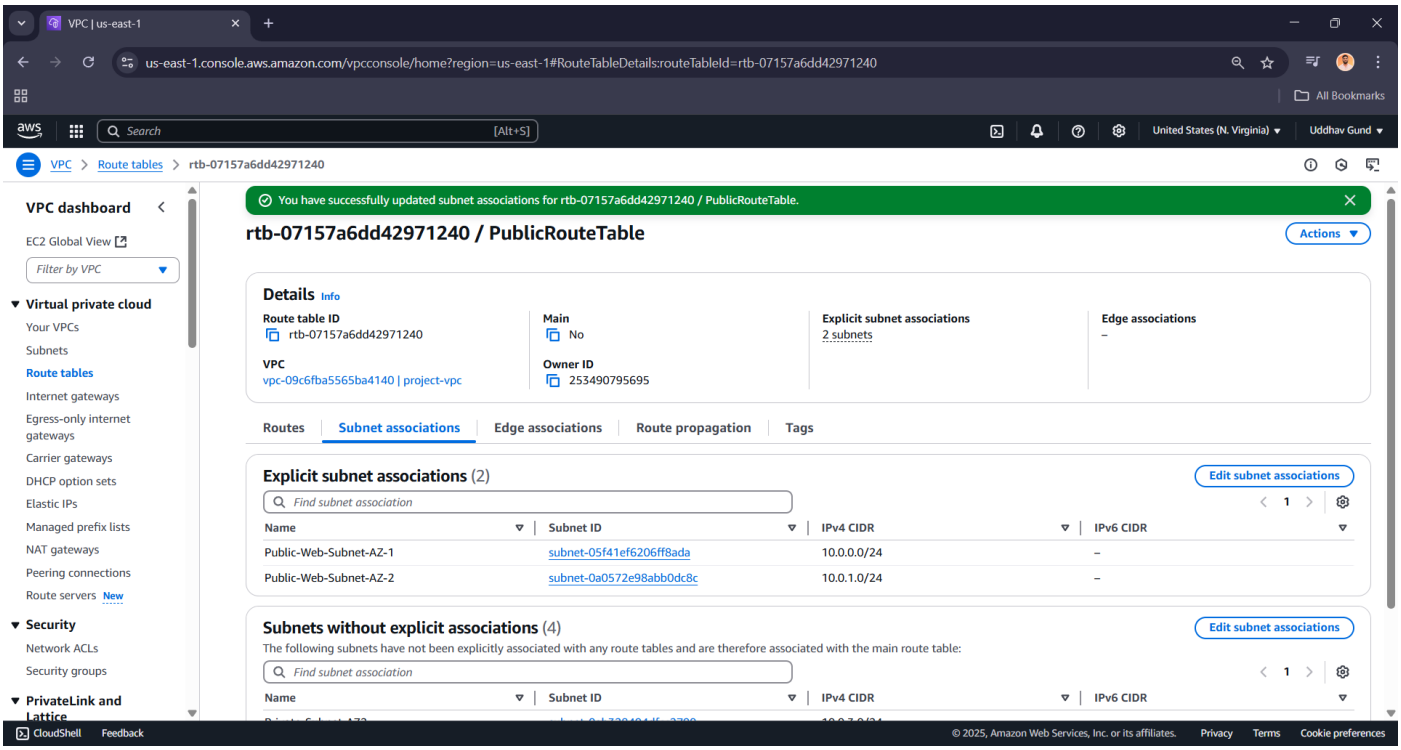
NAT Gateways



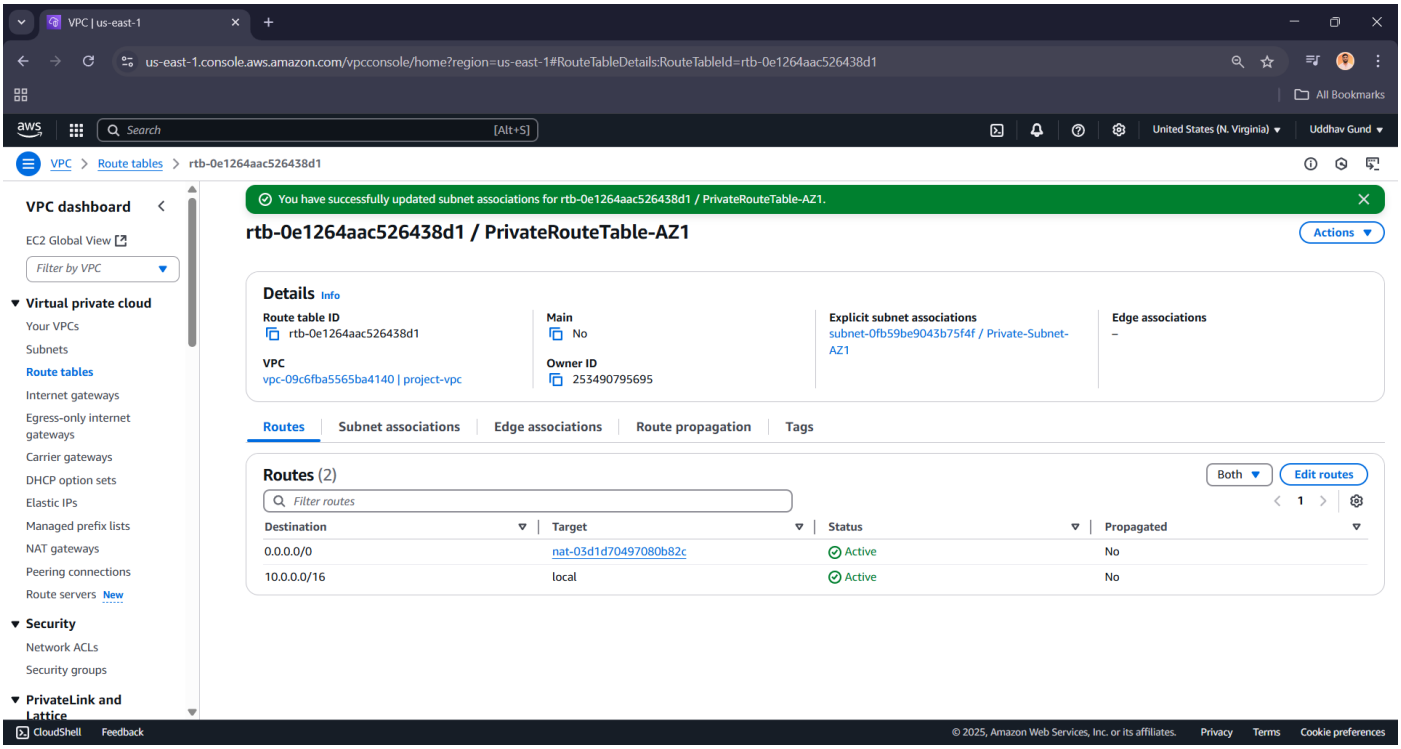
Route Tables :

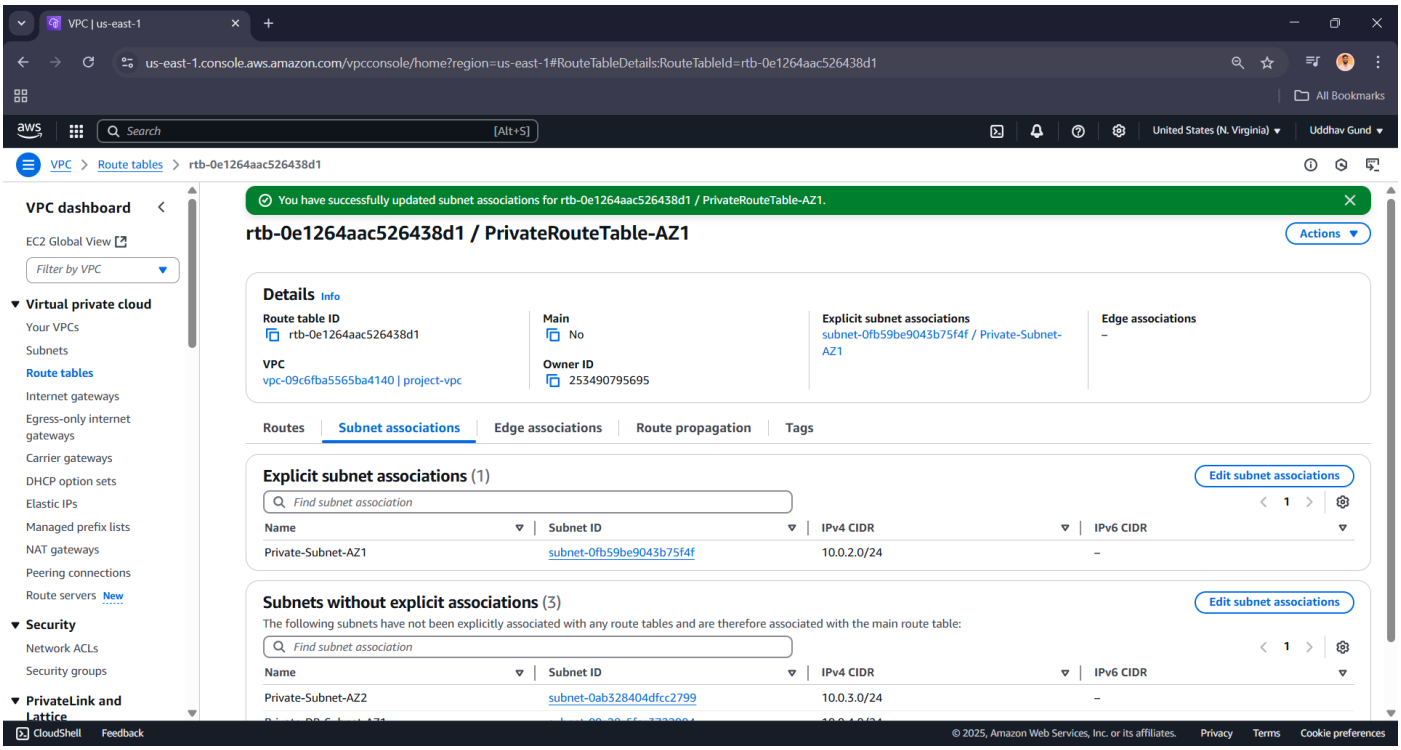


Edited **Subnet Association** and attached Route Table to **both Public Subnet**.

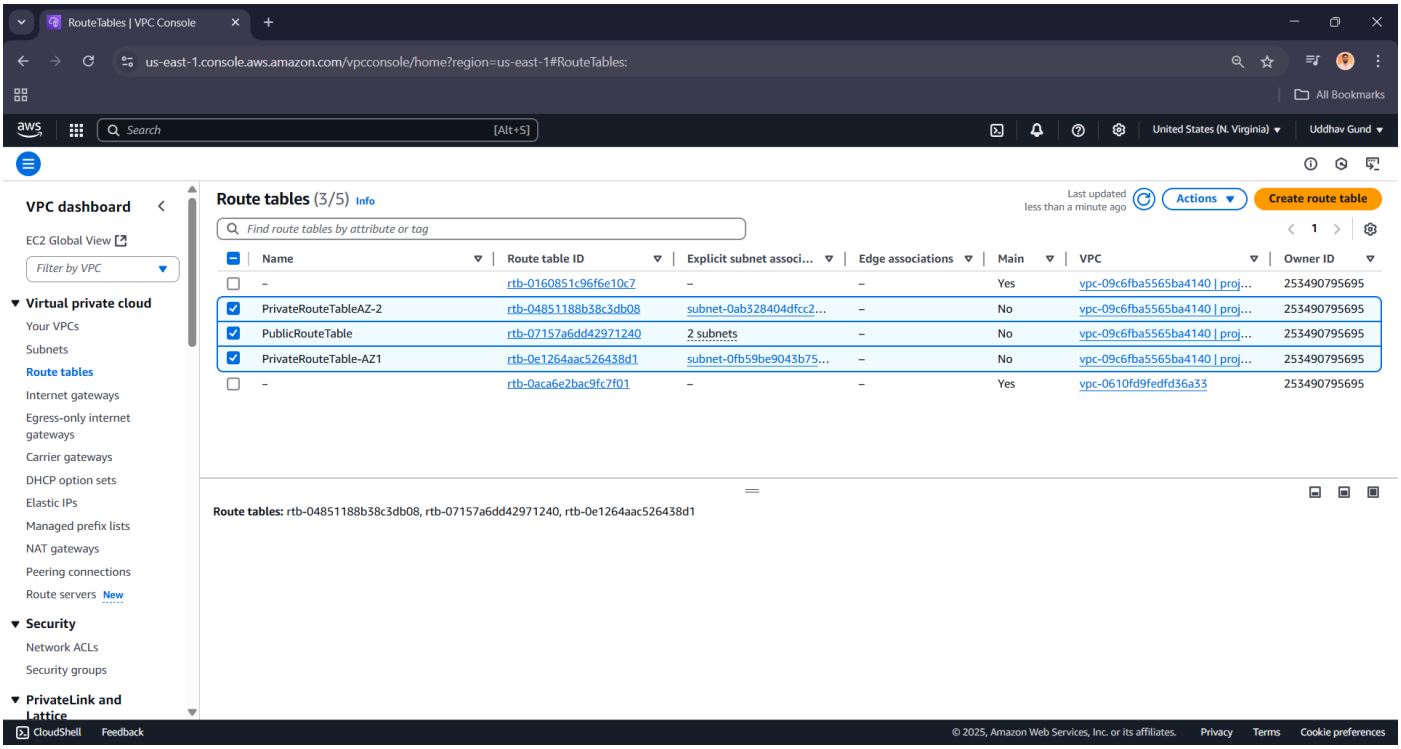


Created and Attached **Route Tables to Both Private Subnets**

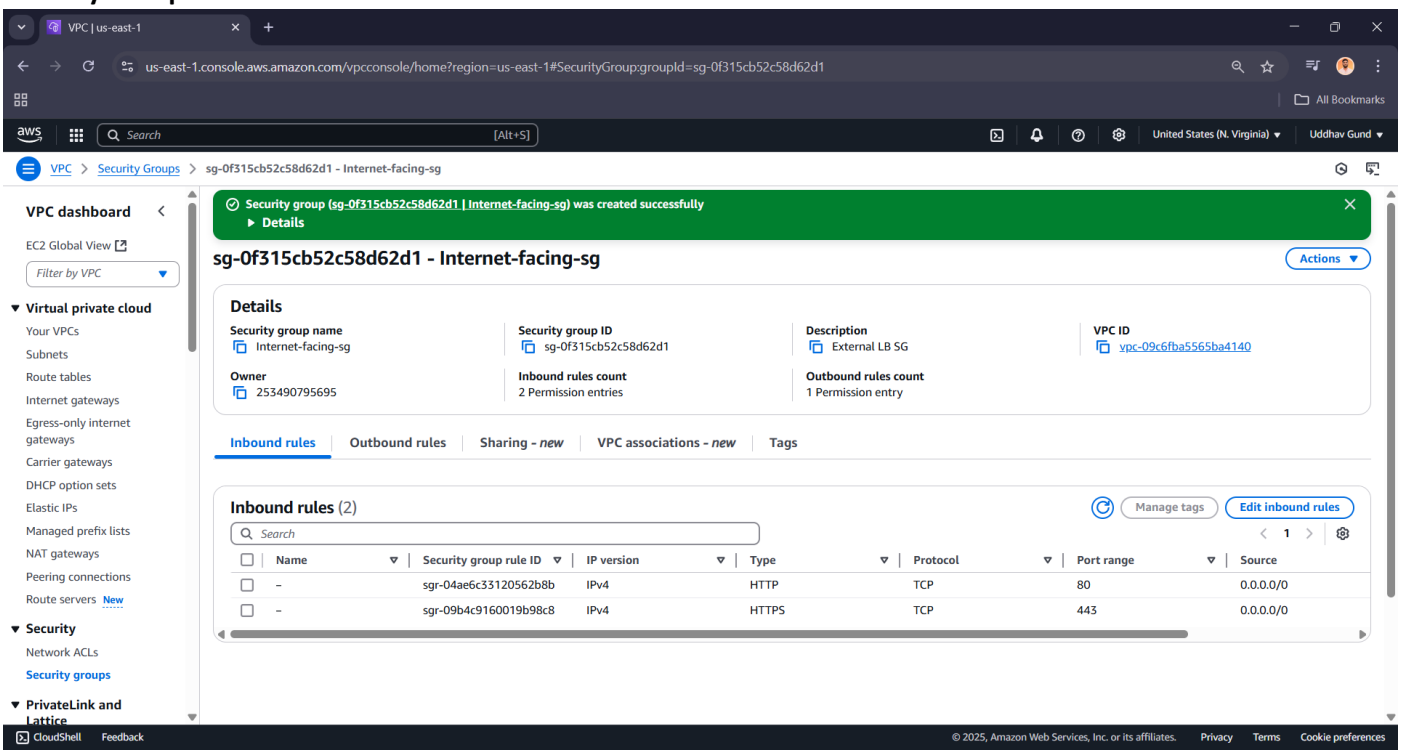




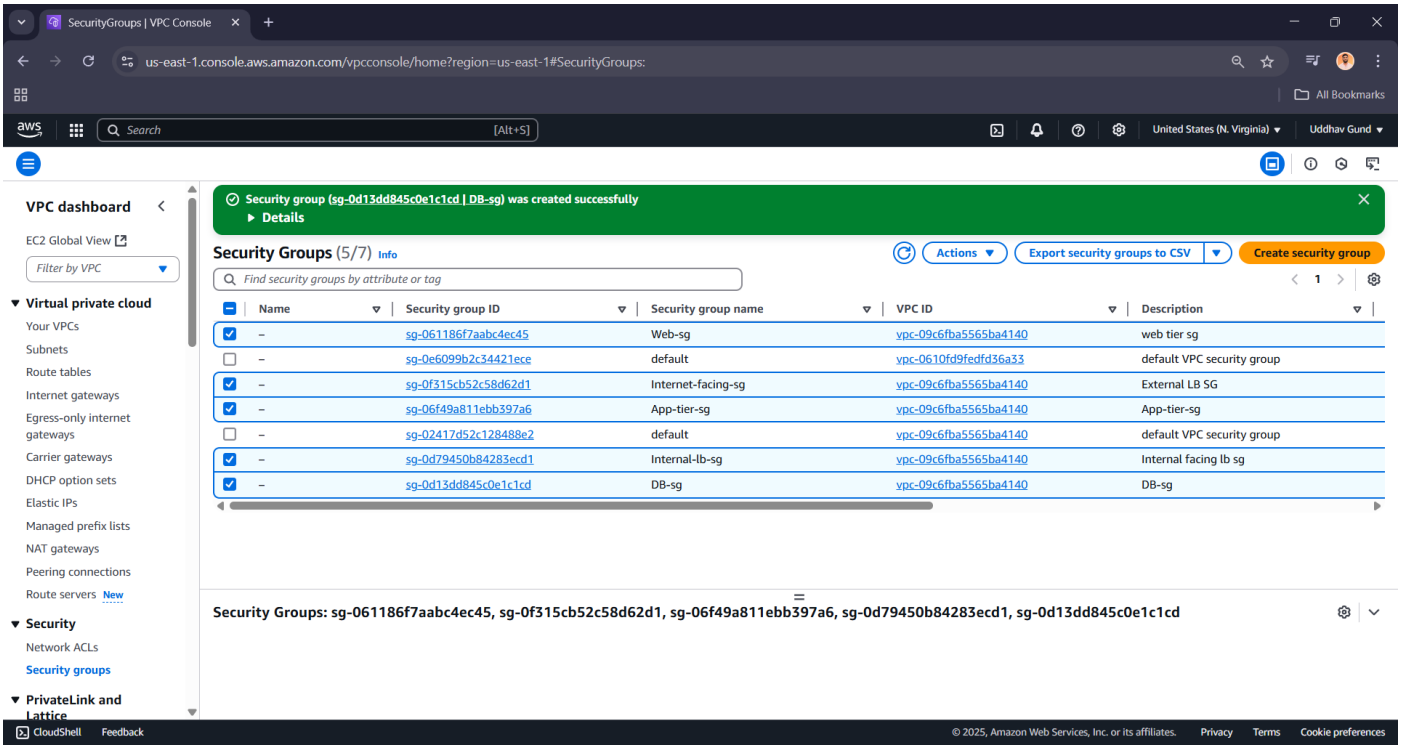
All Route Tables



Security Groups



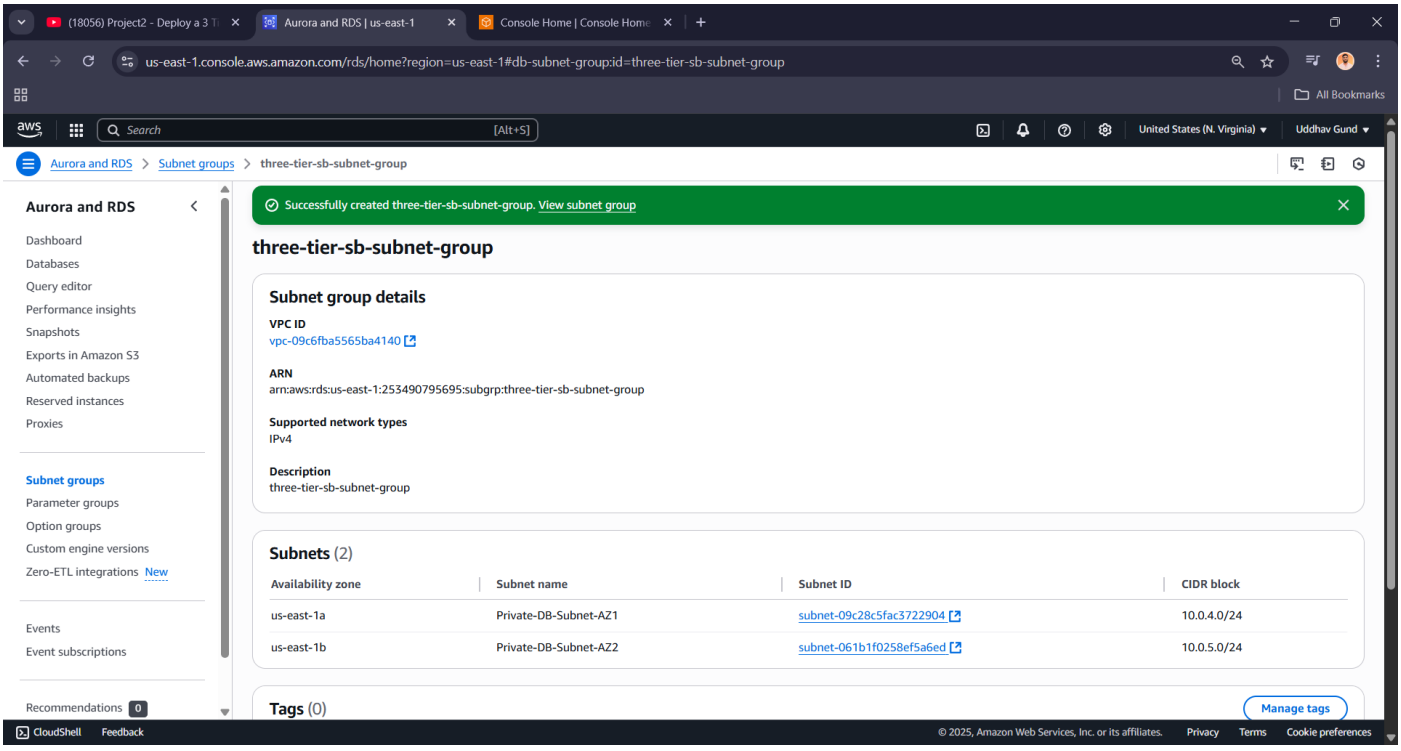
All Security Groups Created And Added Inbound Rules According to need



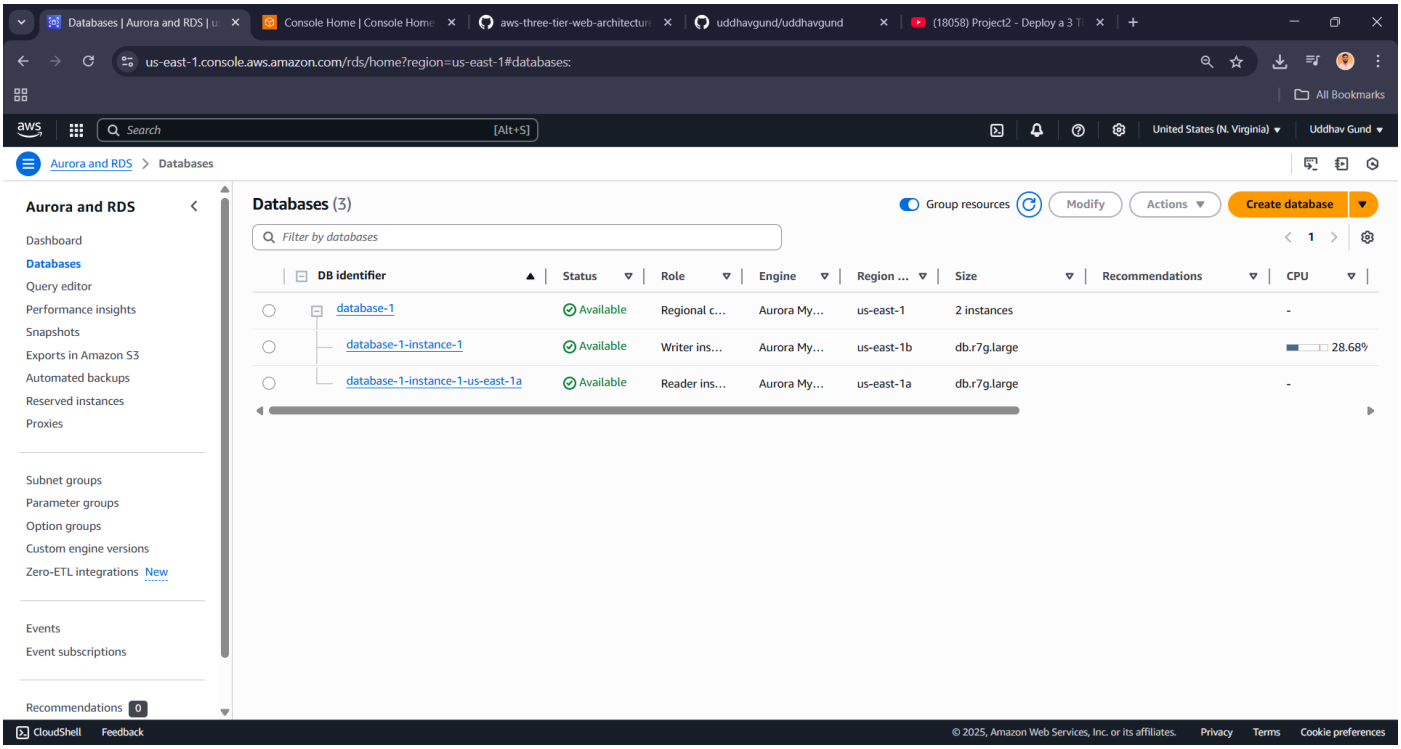
Part 2: Database Deployment

- Deploy Database Layer
  - Subnet Groups
  - Multi-AZ Database

Subnet Groups Created



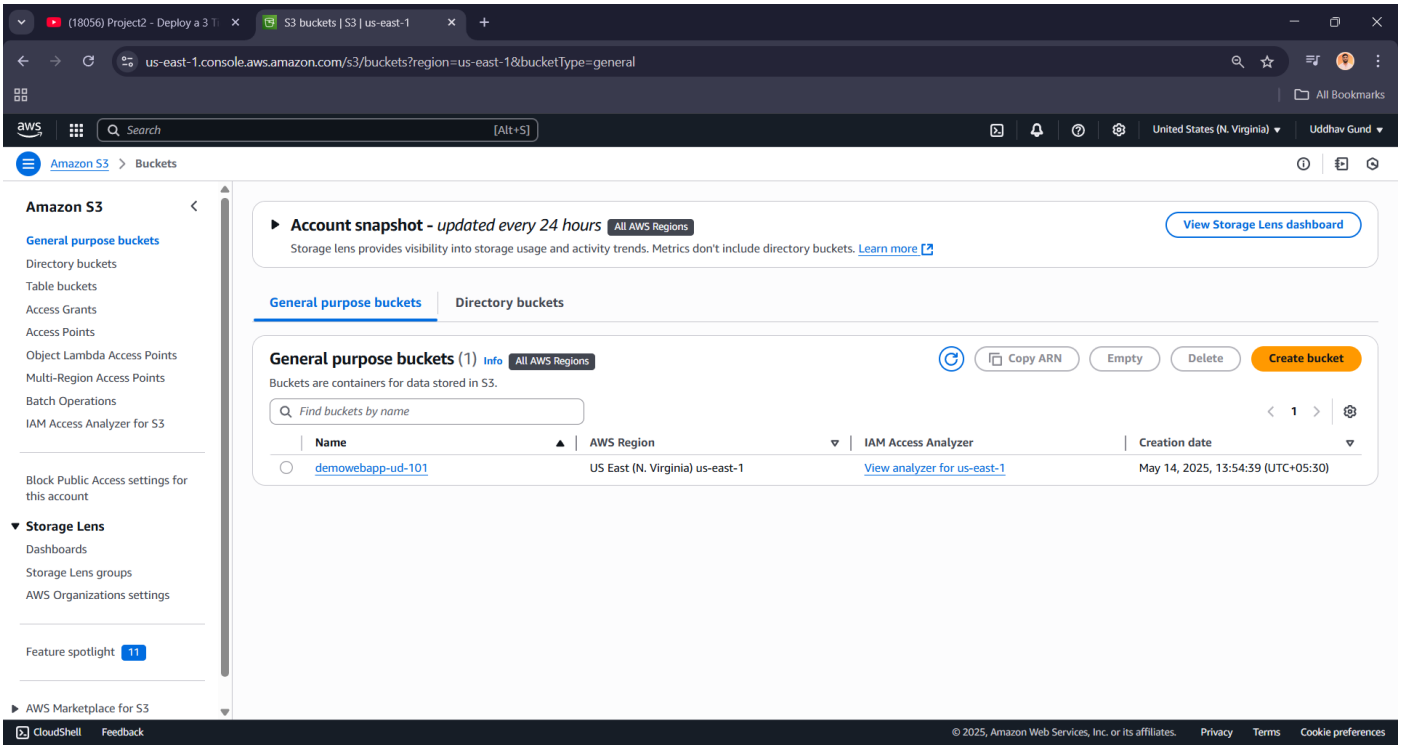
MySQL-Compatible Amazon Aurora database Created with Reader Node in Different AZ



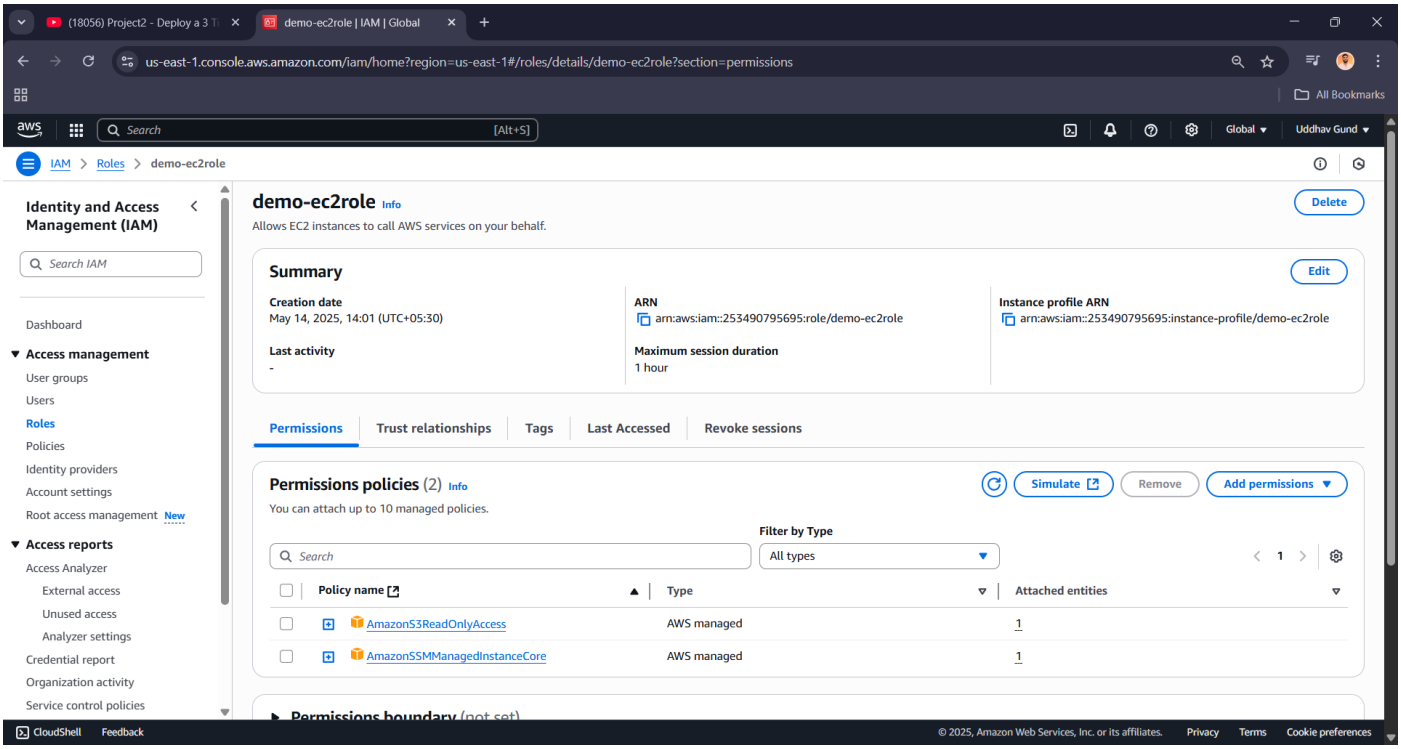
Part 3: App Tier Instance Deployment

Uploaded Source code of web Application to S3 so, our instances can access it and also created an AWS IAM EC2 role with AWS S3 **AmazonSSMManagedInstanceCore**,**AmazonS3ReadOnlyAccess** permission to connect to our instances securely and without needing to created SSH key pairs and access source code from bucket.

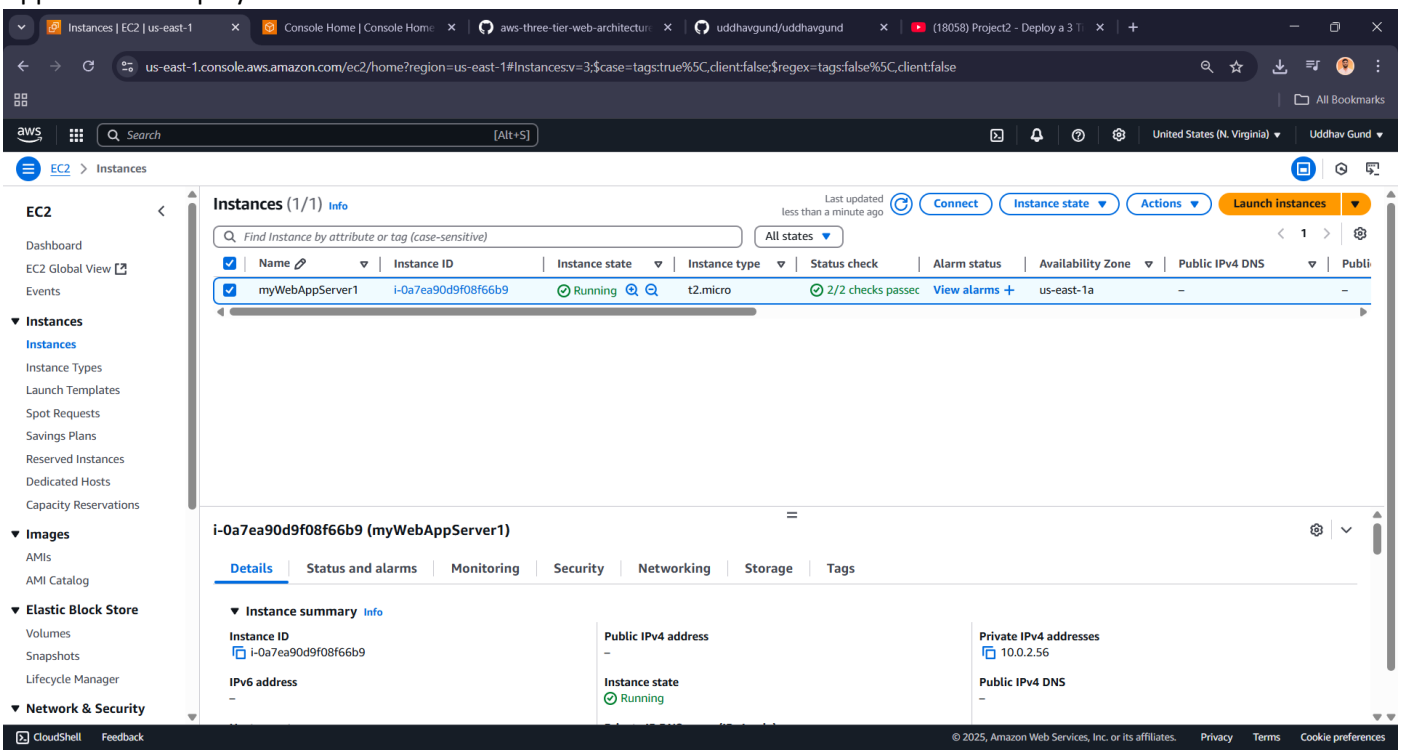
S3 Bucket



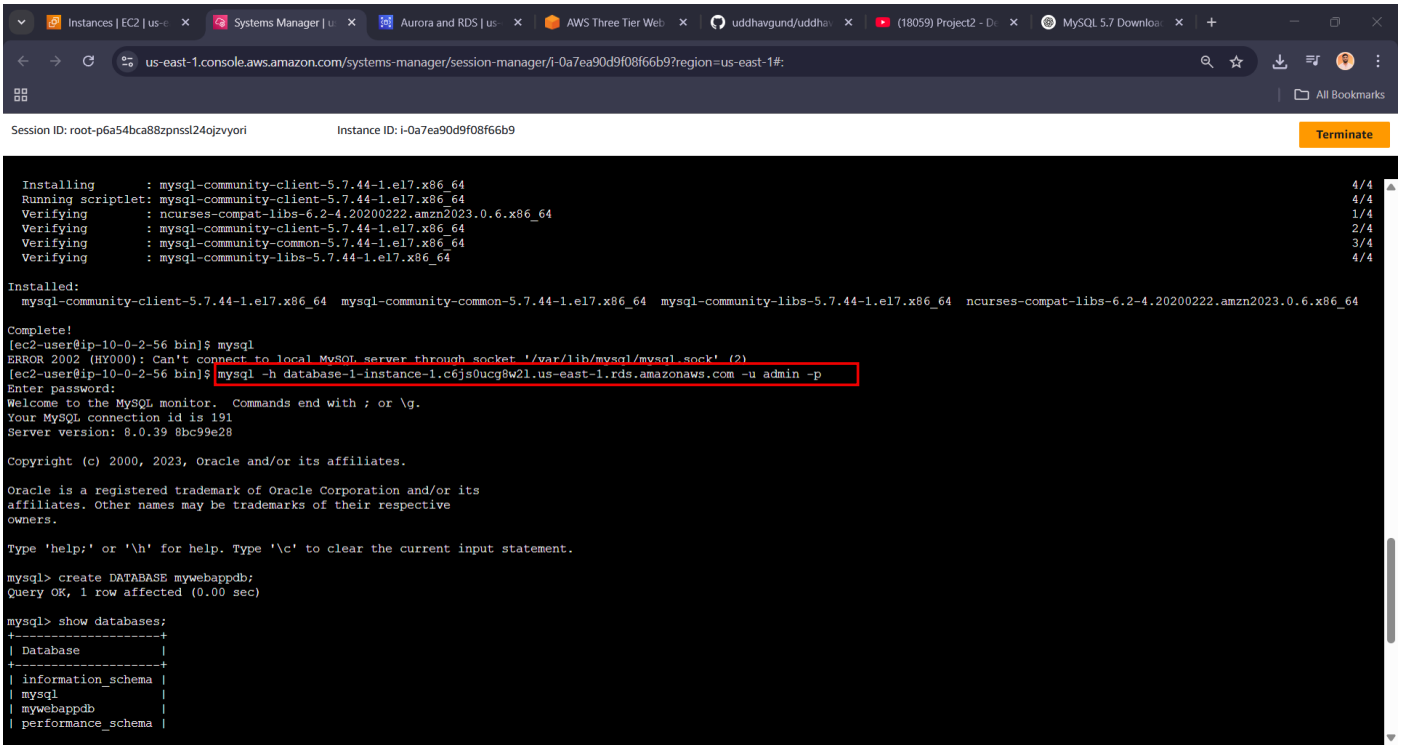
IAM Role



App Instance Deployment

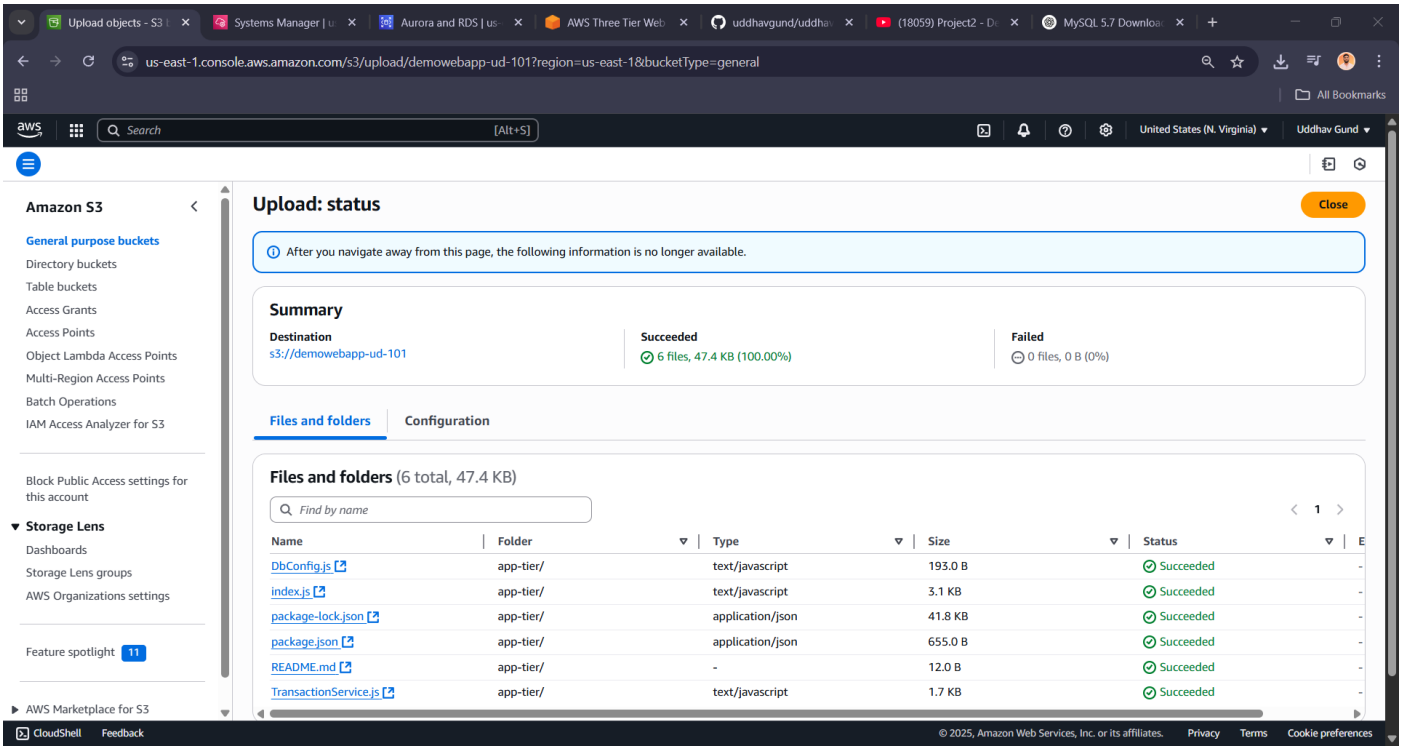


Configure Database



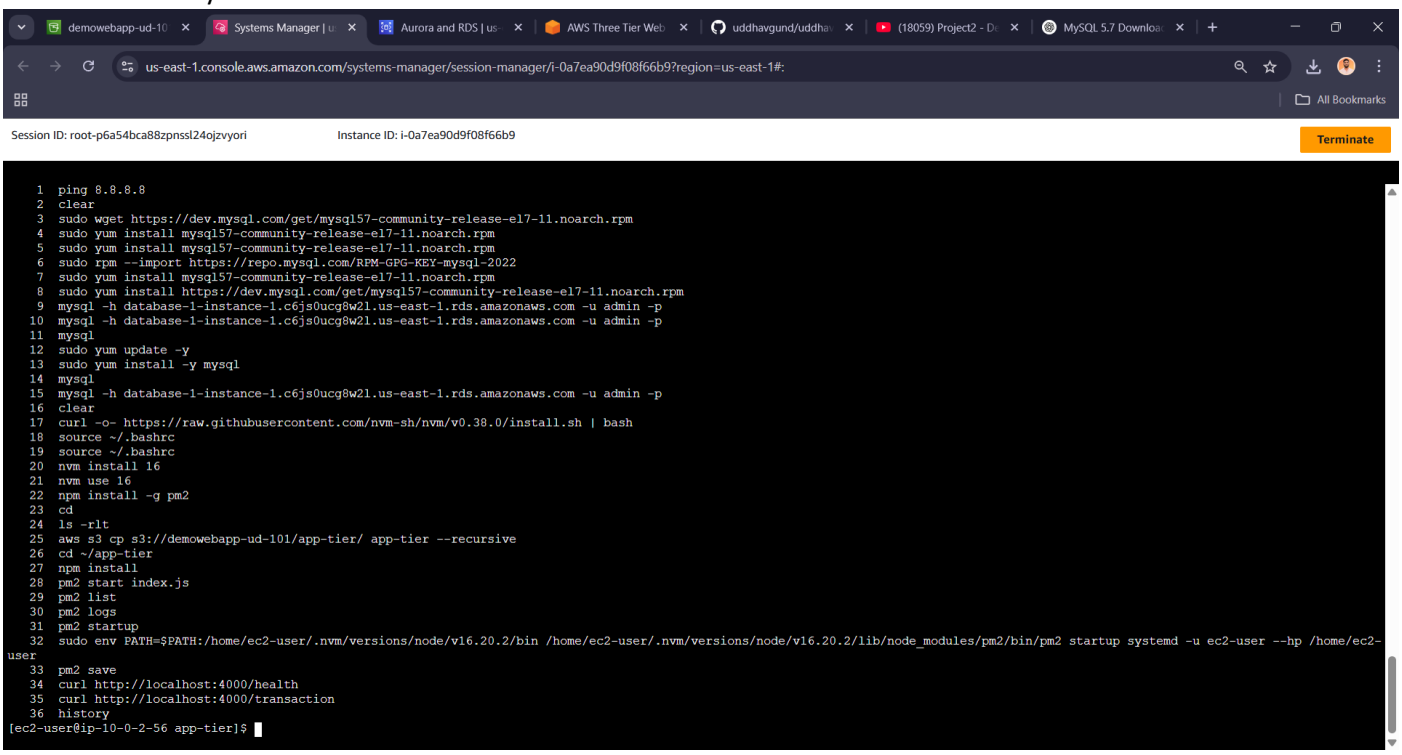
Configure AppInstance

Uploded AppTier Back-End Source code to S3 Bucket



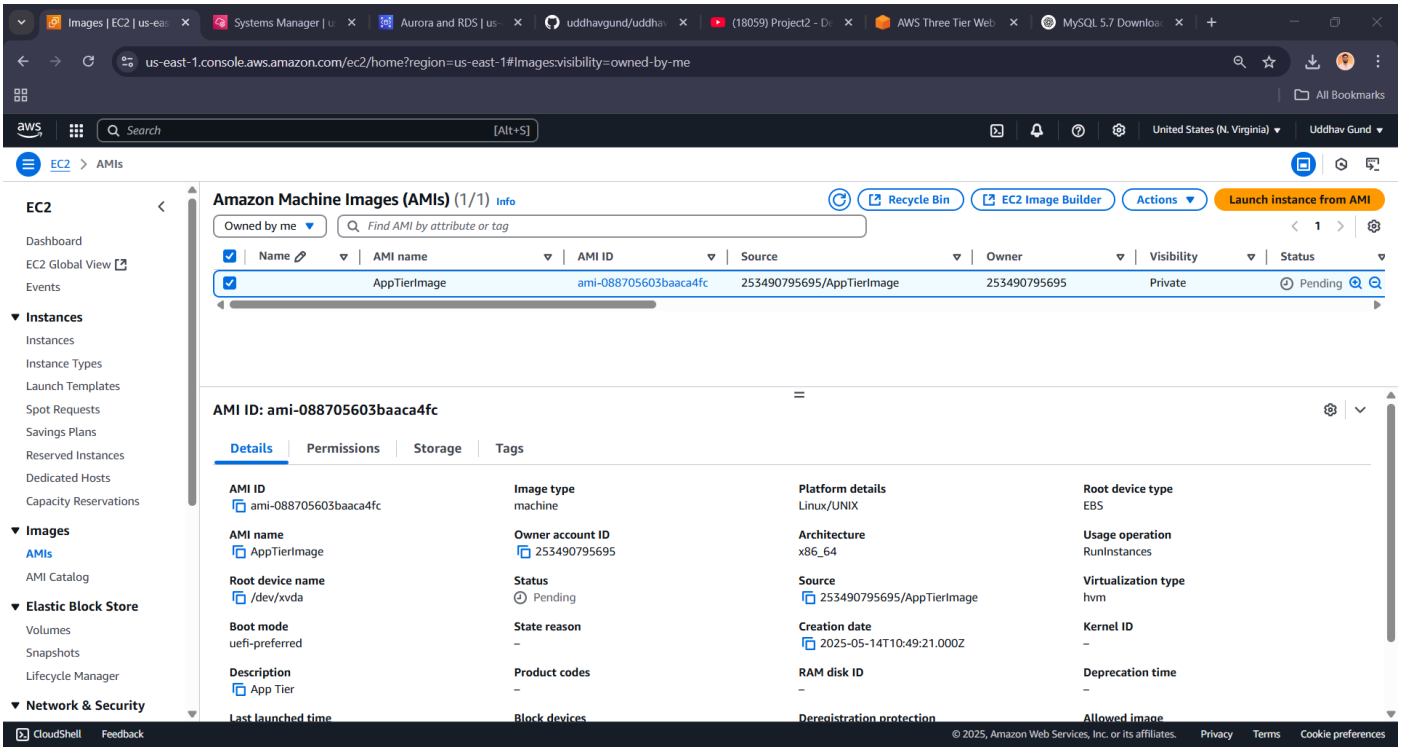
Connected to EC2 Instance to SSM and configured AppInstance and tested DB connectivity

Command History :

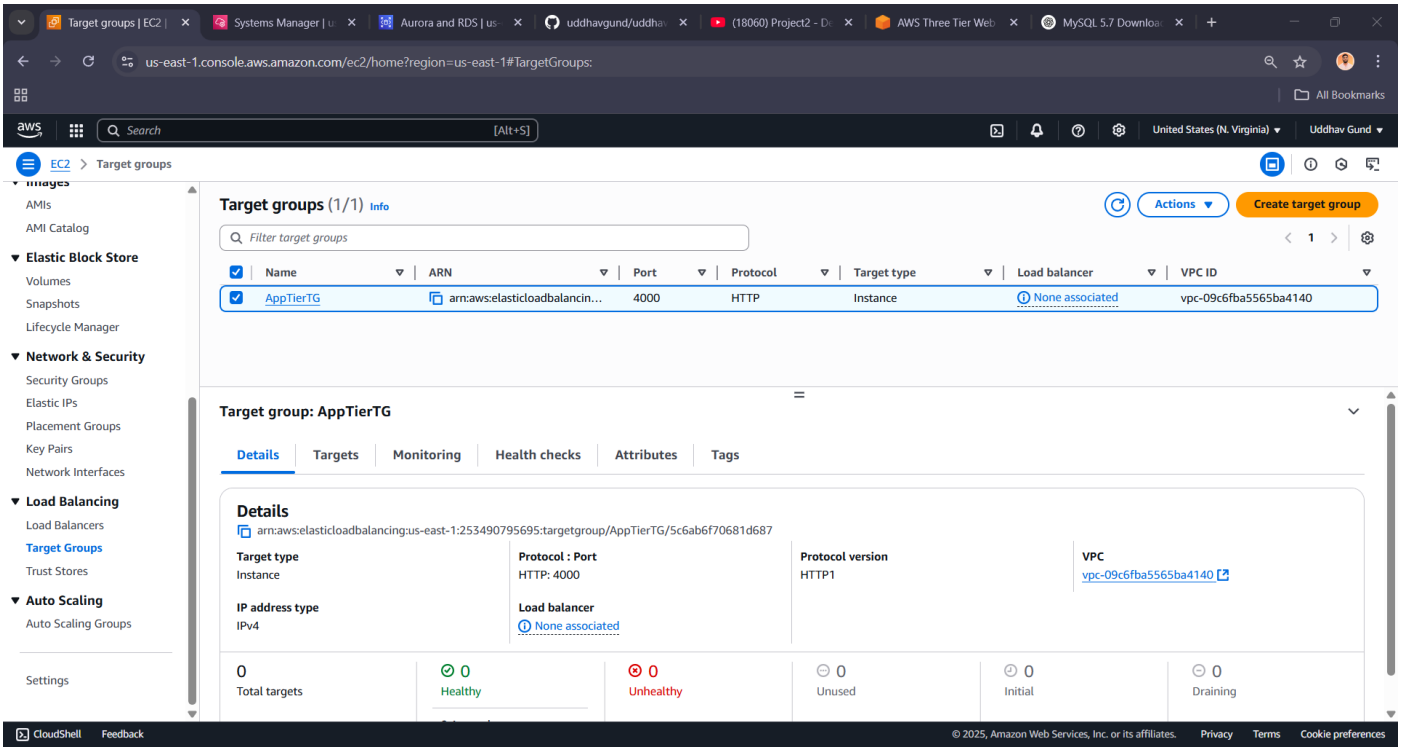


Part 4: Internal Load Balancing and Auto Scaling

Created an AMI of App Tier using previously created AppTier EC2 Instance Image

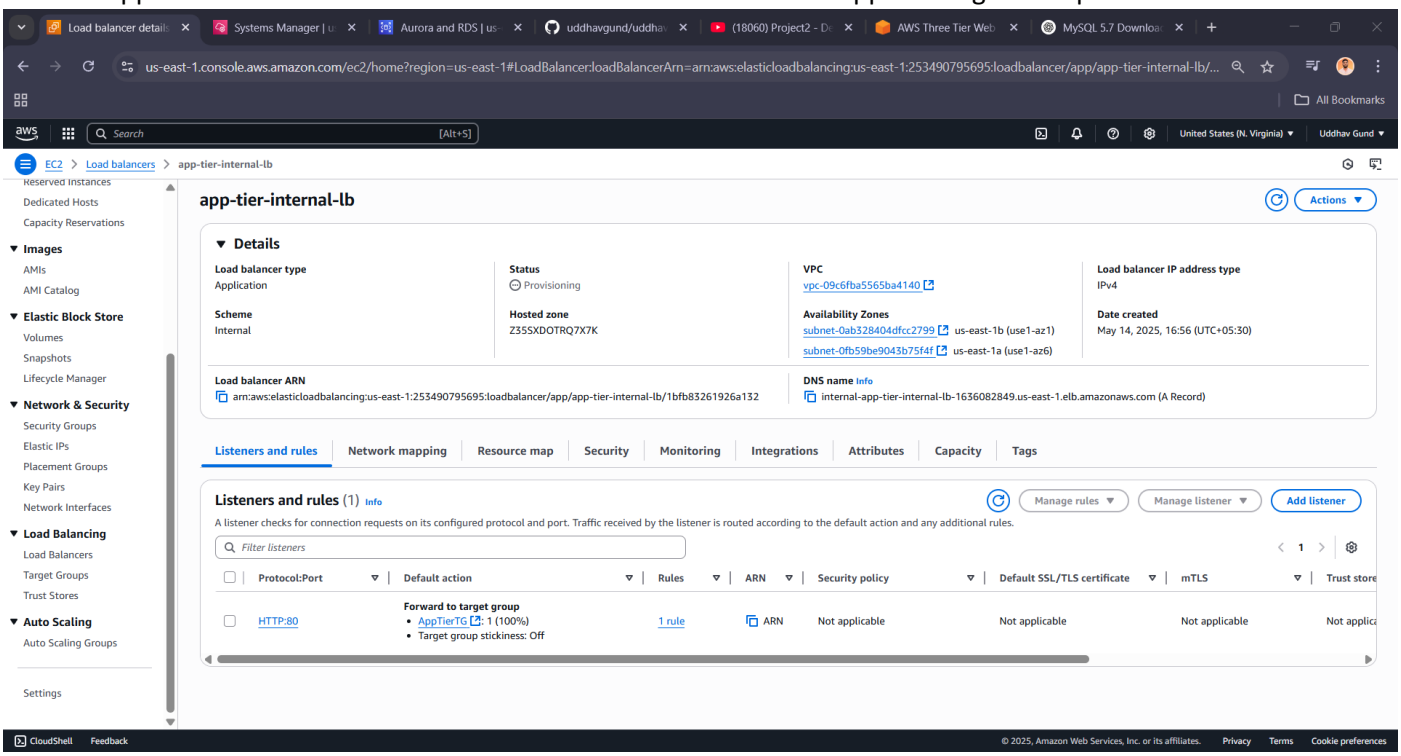


AppTier Target Group Created

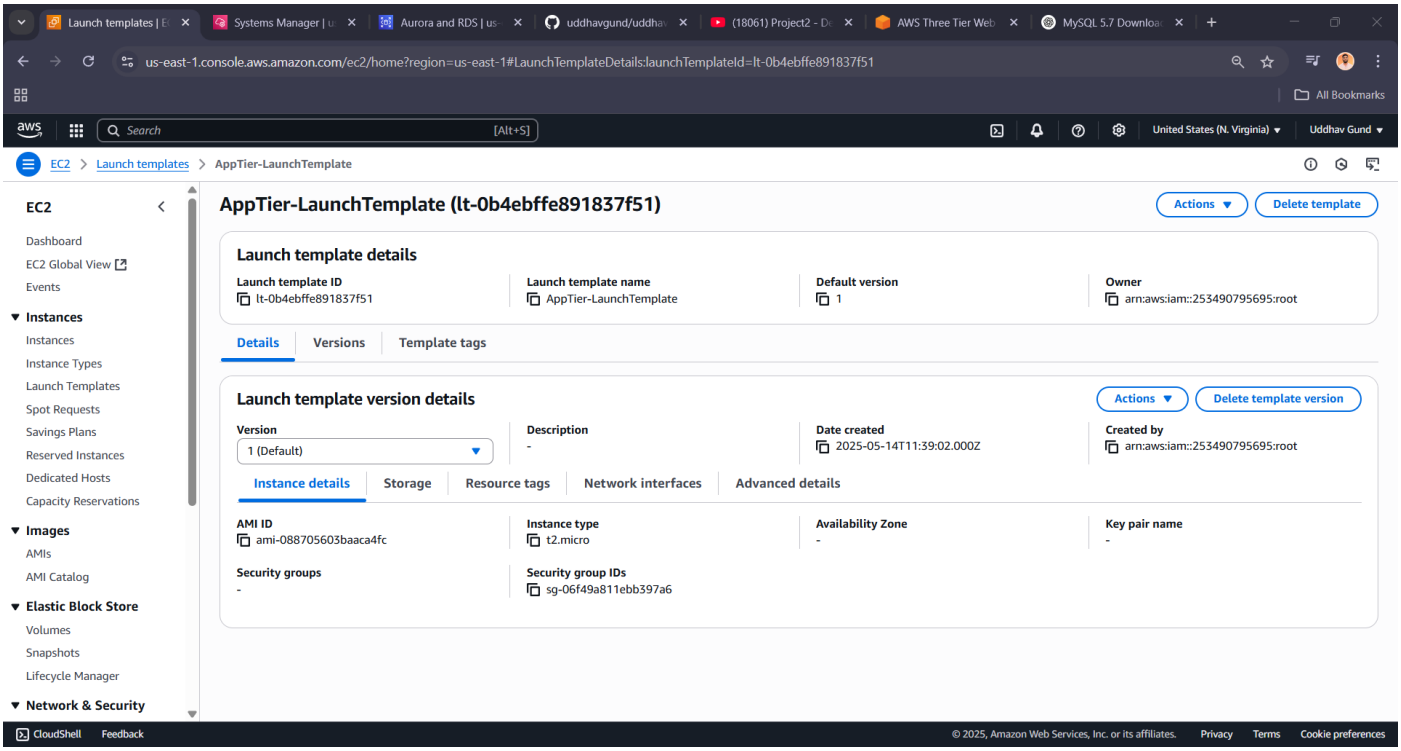


AppTier Internal Load Balancer Created

Created Application Load Balancer and added HTTP Listener to it with AppTier Target Group

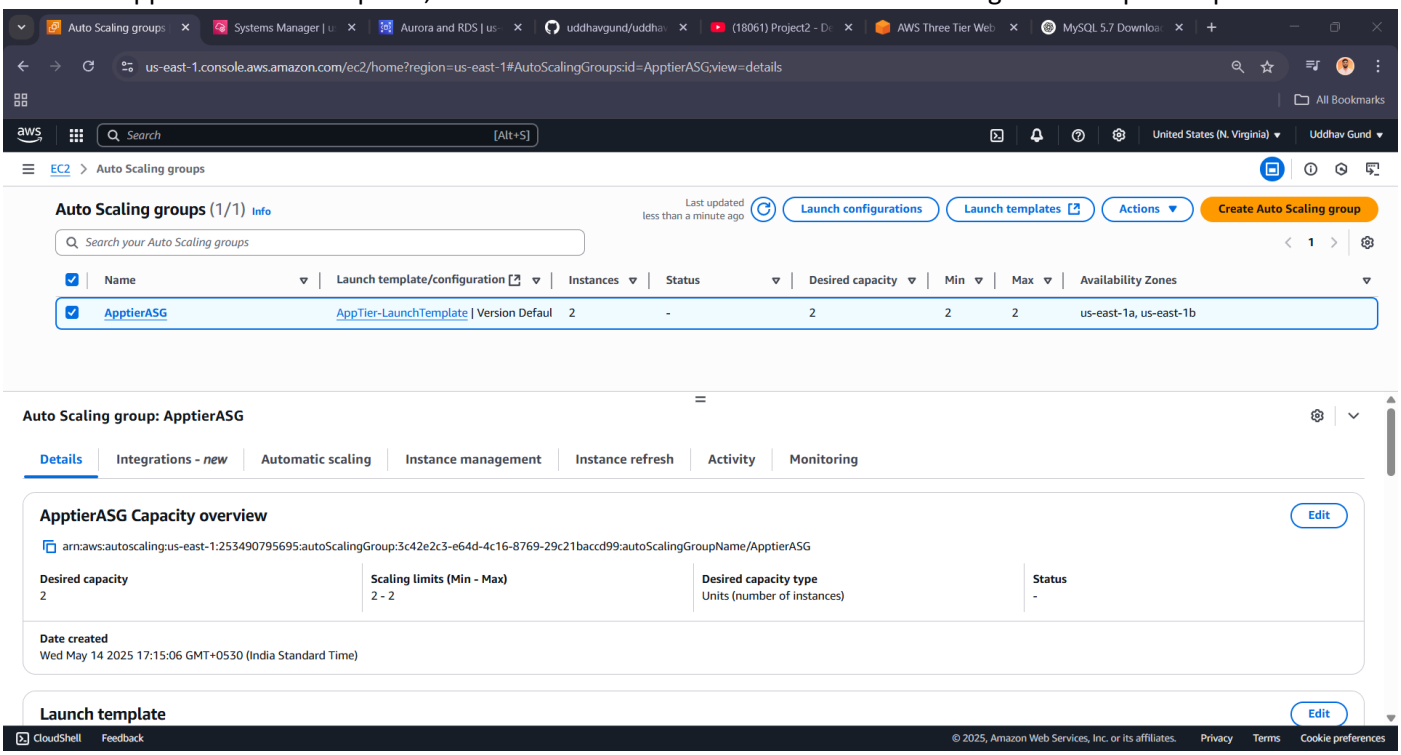


Launch Template created using AppTier AMI which we created previously stage



Auto Scaling Group Created.

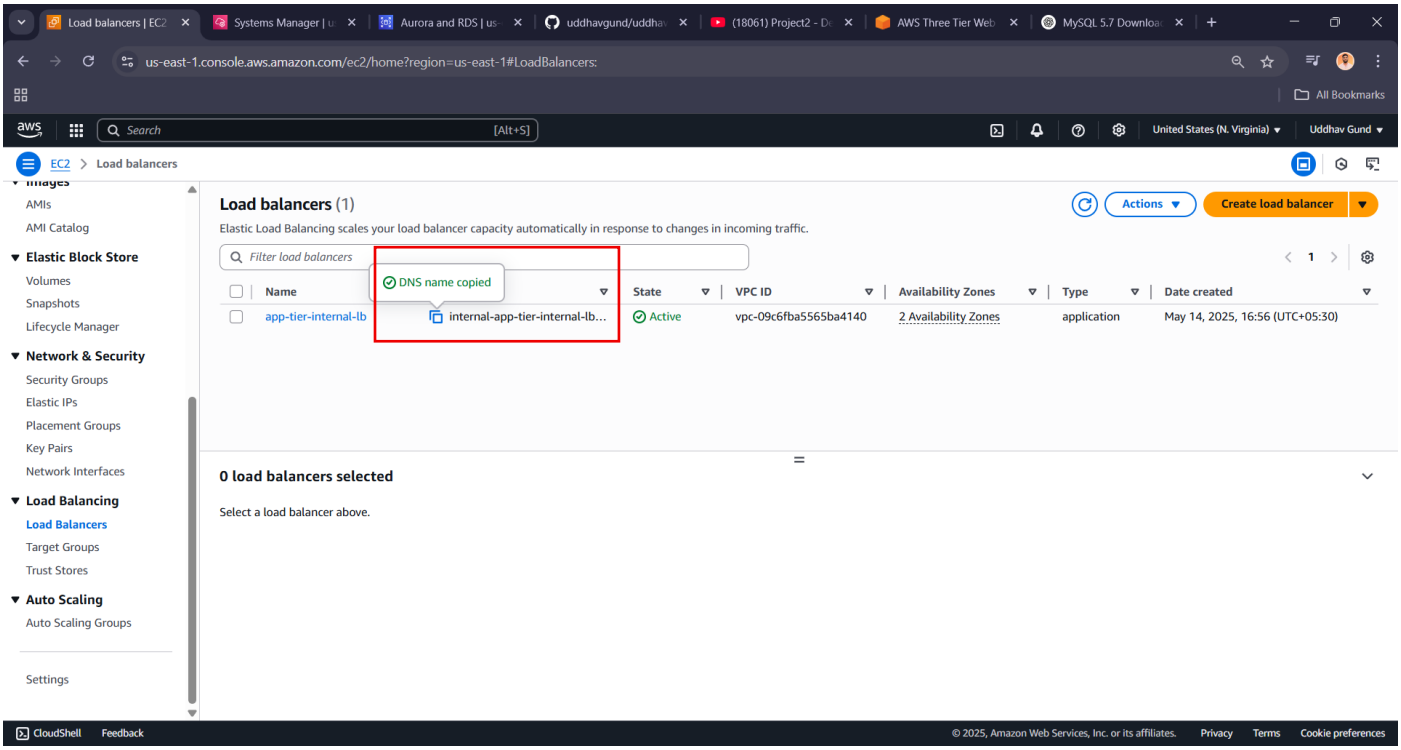
Attached AppTier Launch Template ,also attached internal load balancer to it. Configured Group Size upto 2.



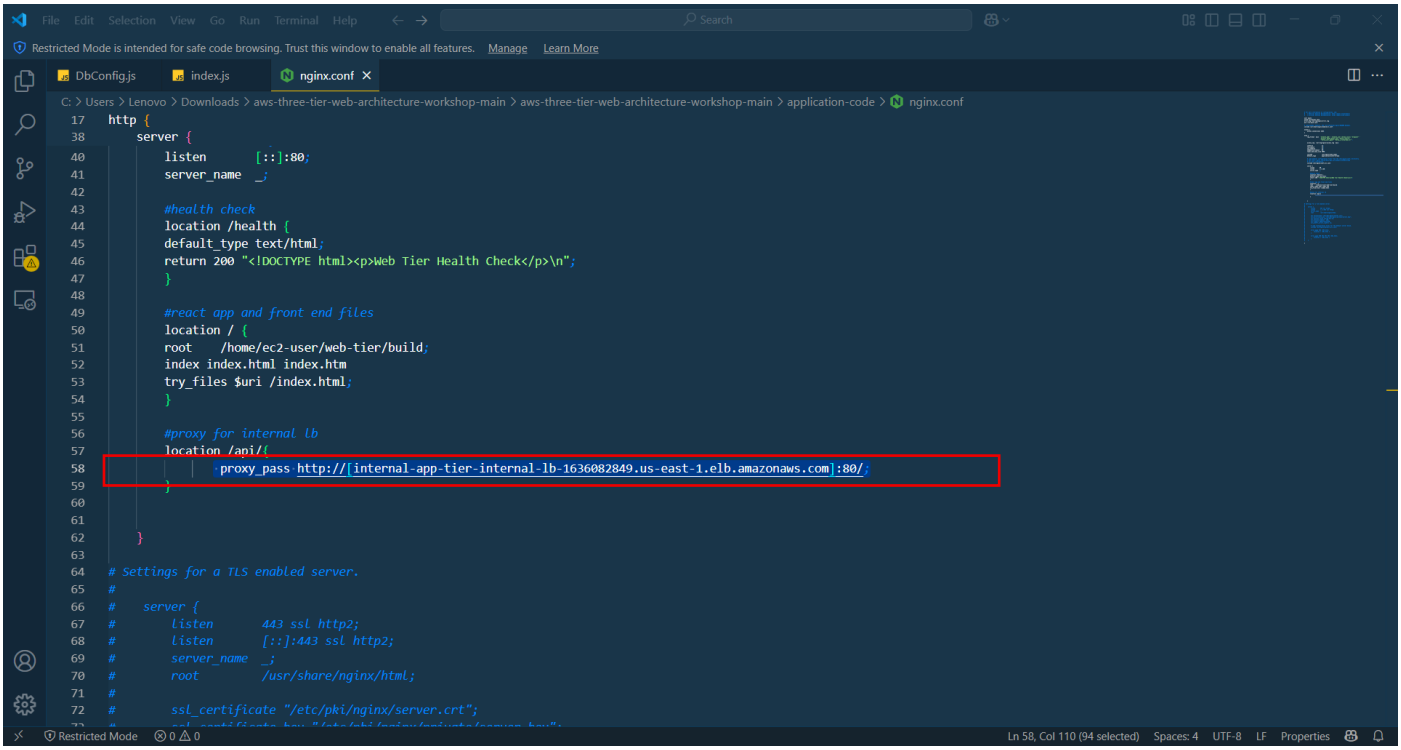
Part 5: Web Tier Instance Deployment

Updated NGINX Configuration File. open up the **application-code/nginx.conf** file from the source code. Scroll down to **line 58** and replaced [INTERNAL-LOADBALANCER-DNS] with our internal load balancer's DNS entry.

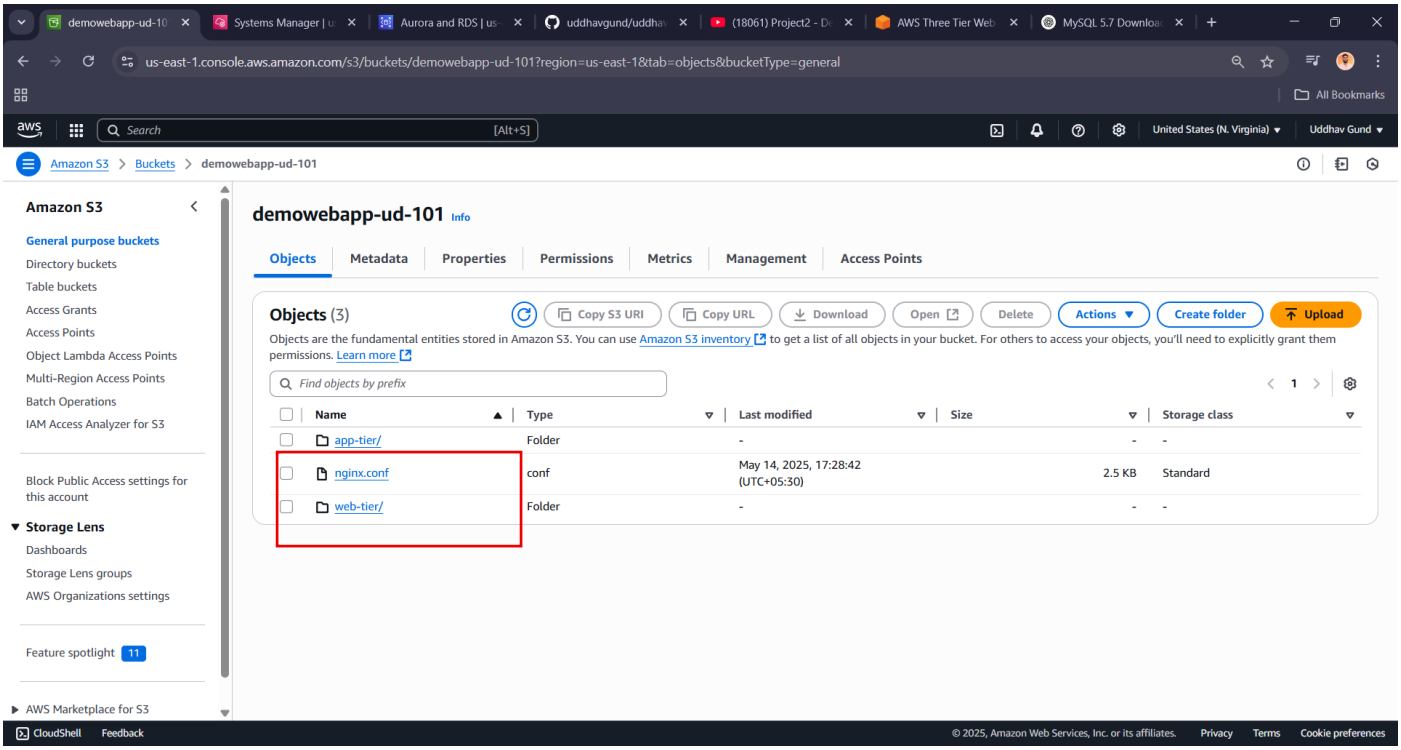
DNS name Copied!



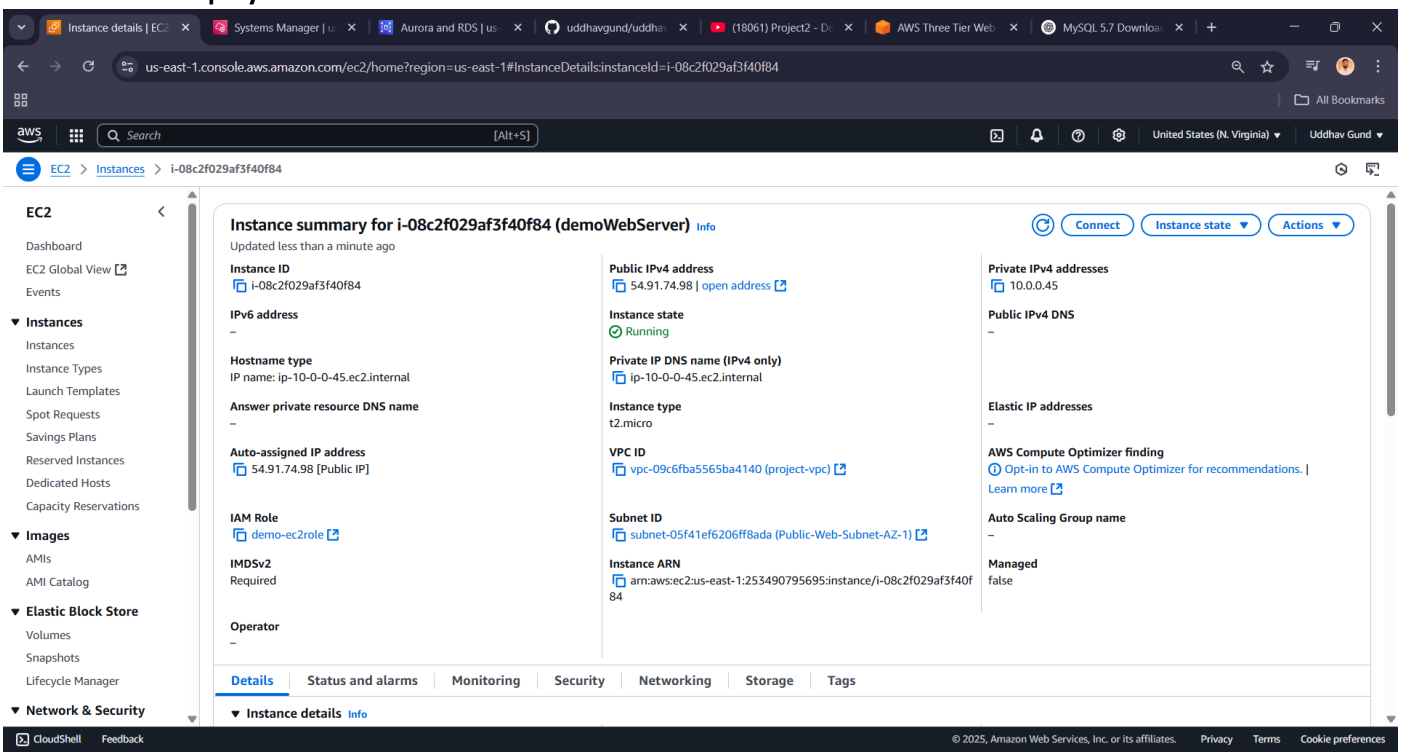
nginx.conf file



Uploaded WebTier code in S3 bucket also nginx.conf



Web Instance Deployment



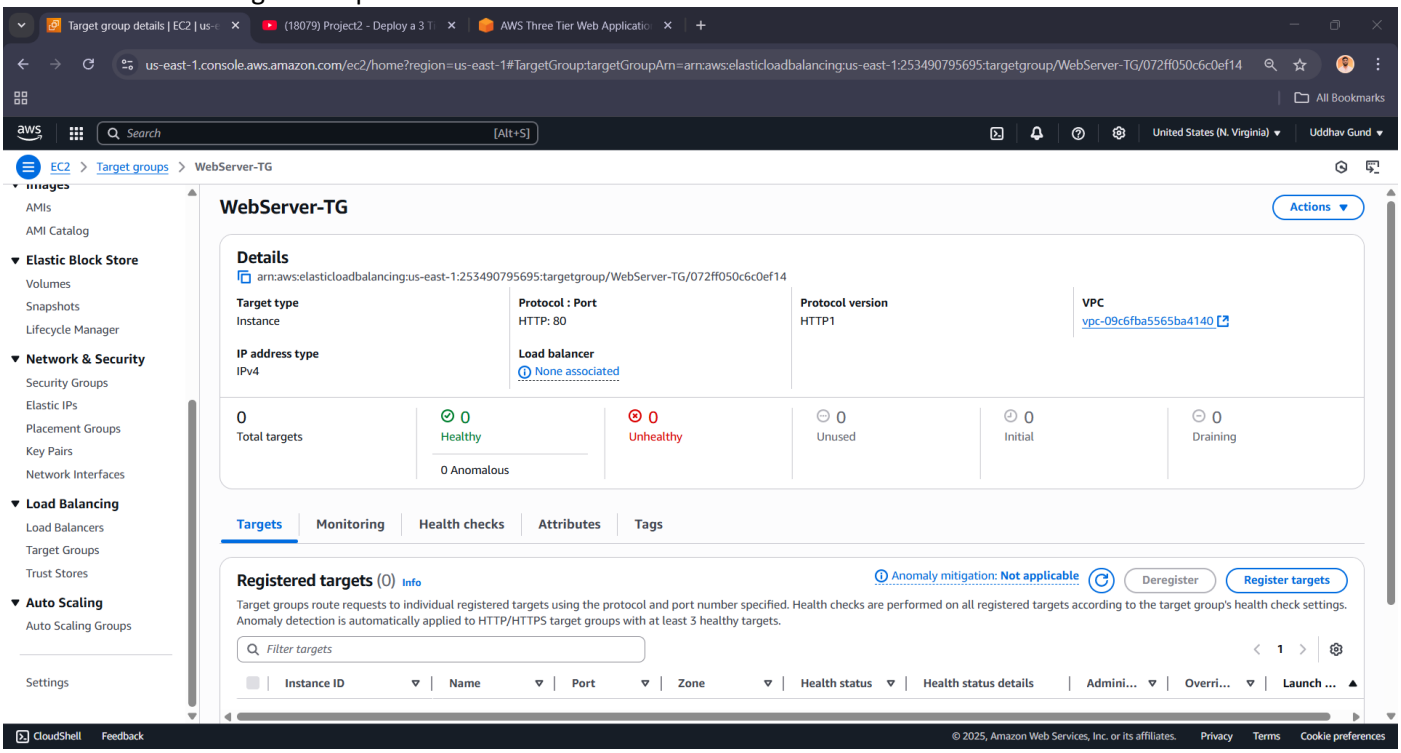
Connected to WebTier EC2 Instance and configure it using below commands:

```
# Switch to ec2-user with root privileges
sudo -su ec2-user
ping 8.8.8.8

# Install NVM (Node Version Manager)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
source ~/.bashrc
nvm install 16
nvm use 16
```

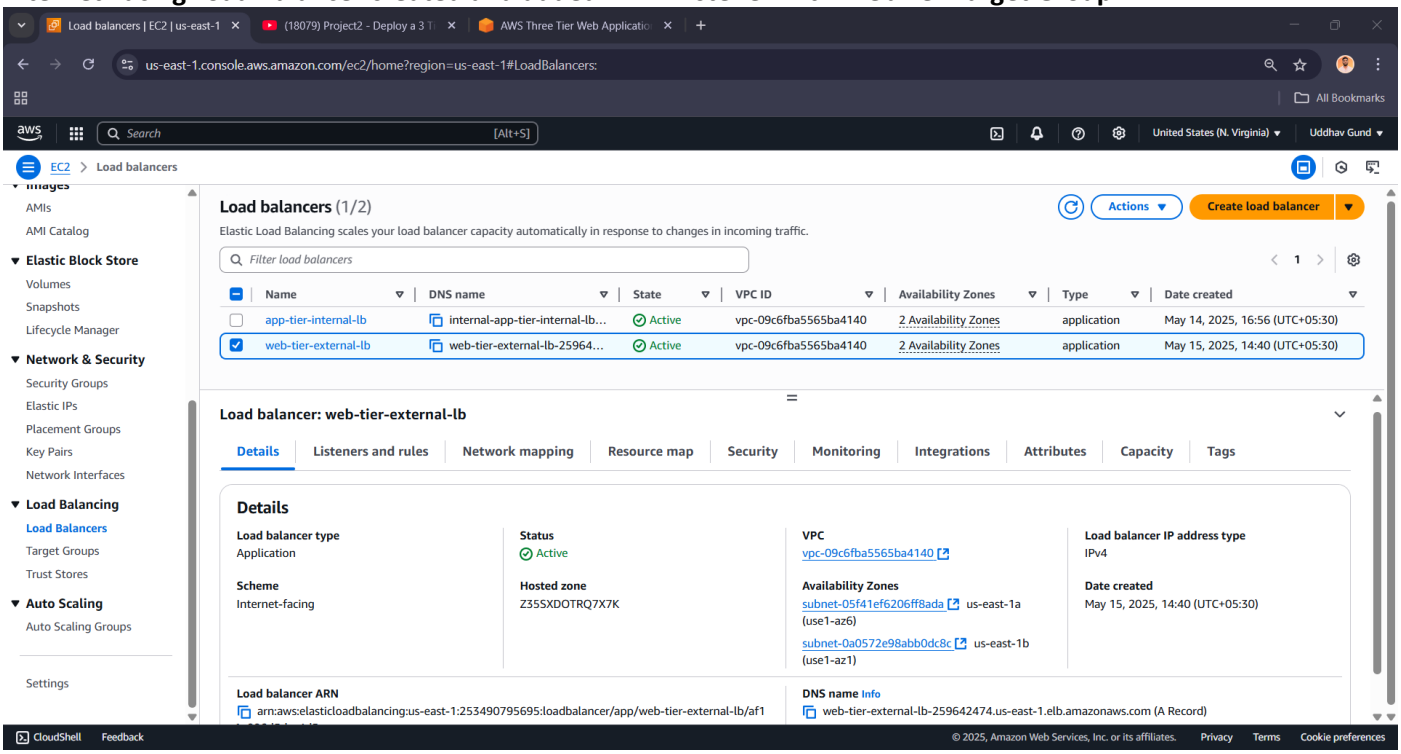
```
# Change to home directory
cd ~/
# Download web-tier files from S3 bucket
aws s3 cp s3://BUCKET_NAME/web-tier/ web-tier --recursive

#install and build Node.js Application
cd ~/web-tier
npm install
npm run build

# Install Nginx from Amazon Linux Extras
sudo amazon-linux-extras install nginx1 -y

cd /etc/nginx
ls

# Remove default Nginx config file and download custom Nginx config file
sudo rm nginx.conf
sudo aws s3 cp s3://BUCKET_NAME/nginx.conf .

# Restart Nginx to apply new configuration
sudo service nginx restart

# Set proper permissions for ec2-user home directory
chmod -R 755 /home/ec2-user

# Enable Nginx to start on boot
sudo chkconfig nginx on
```

Part 6: External Load Balancer and Auto Scaling

Created WebTier Target Group

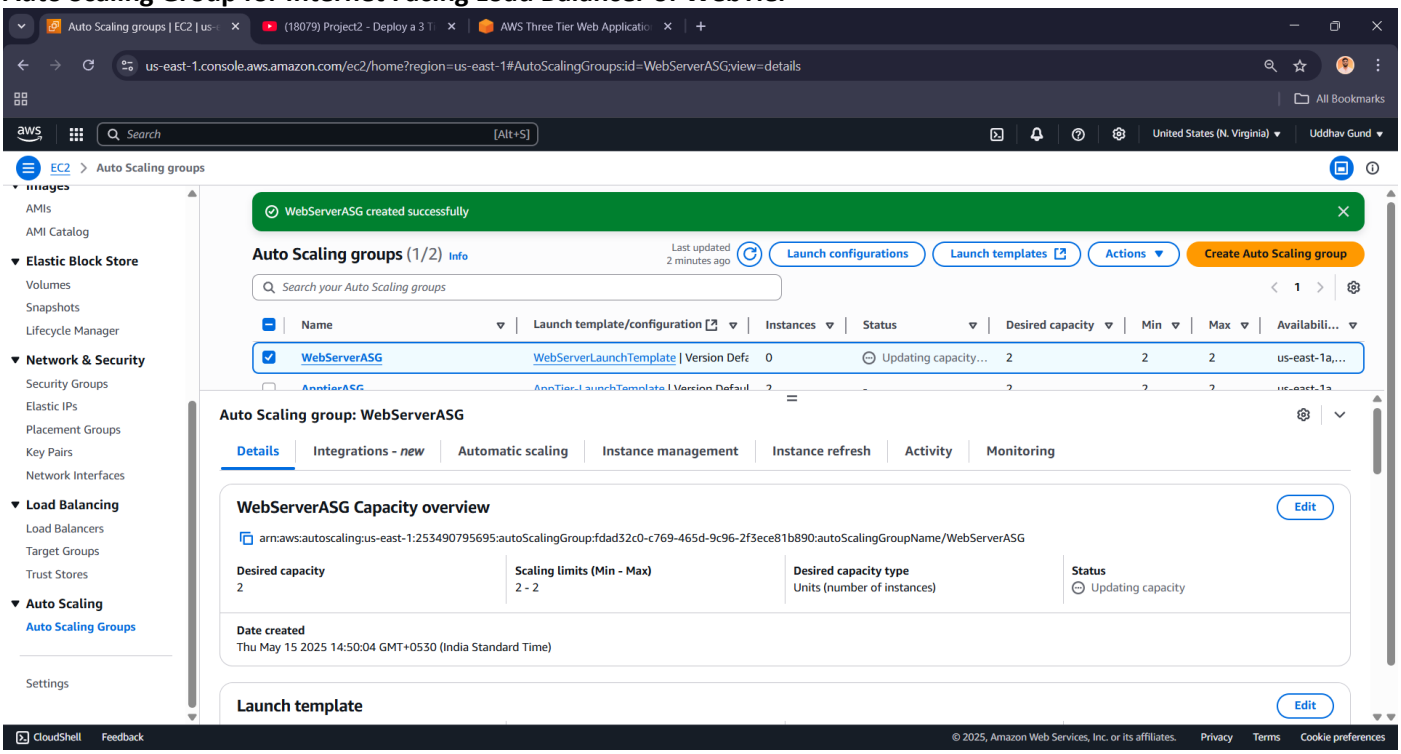


Internet Facing Load Balancer created and added HTTP Listener with WebTier Target Group

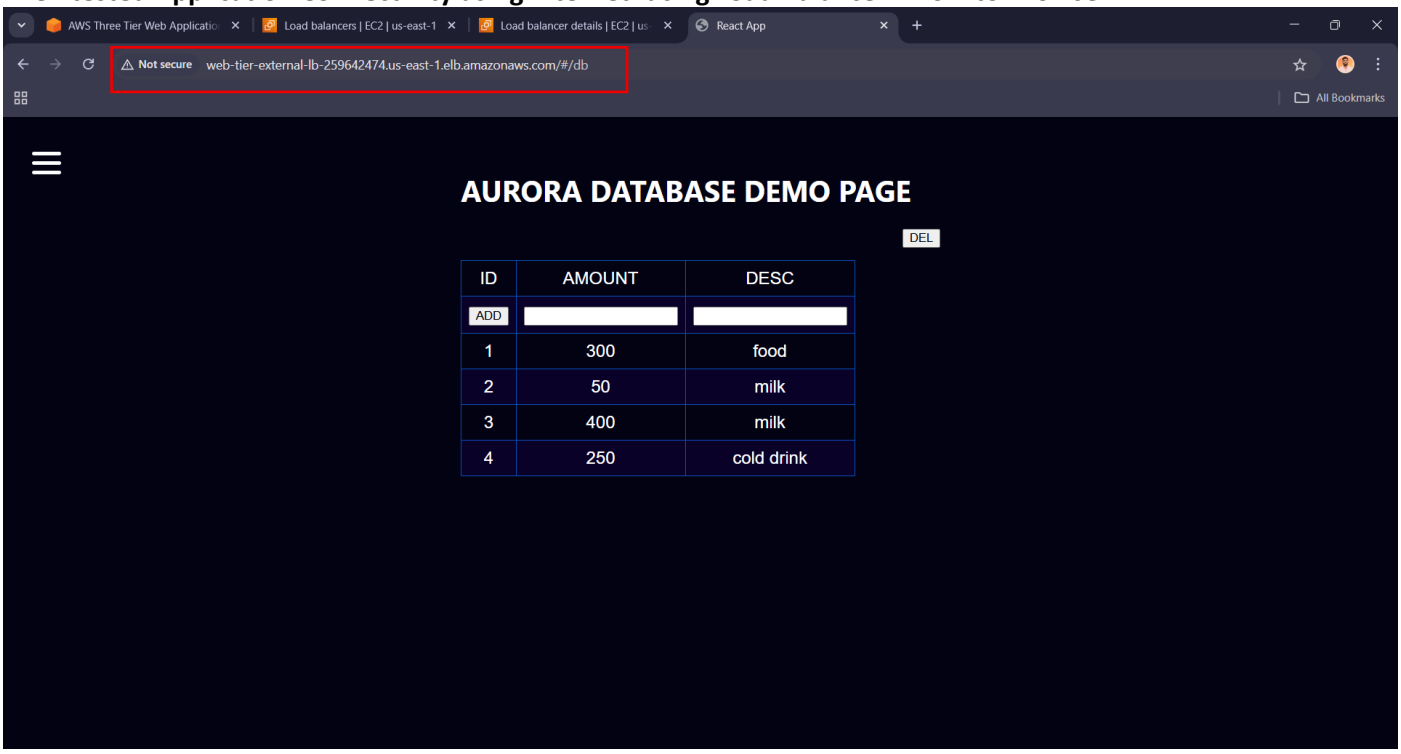


Afterthat Launch Template created for ASG in WebTier using AMI of WebTier Instance.

Auto Scaling Group for Internet Facing Load Balancer of WebTier

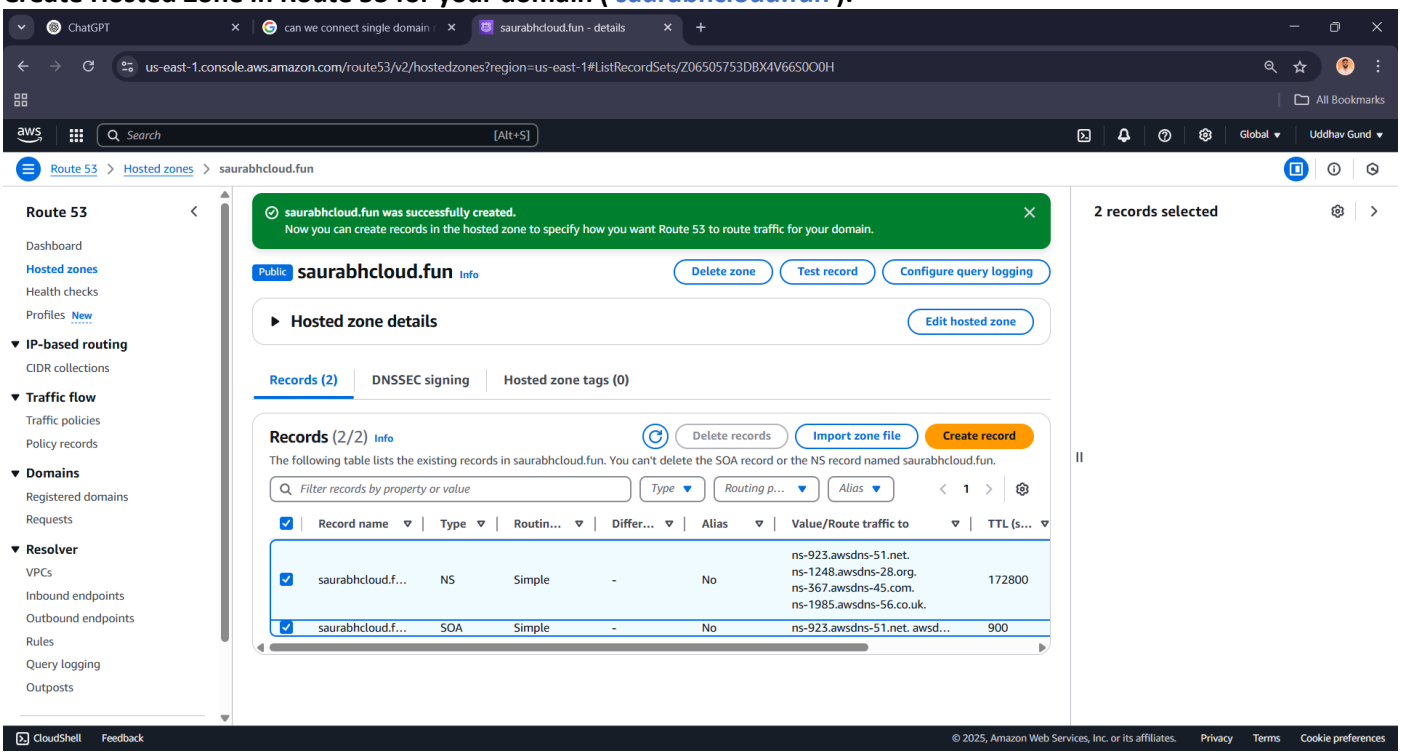


Then tested Application Connectivity using Internet facing Load Balancer DNS into Browser.

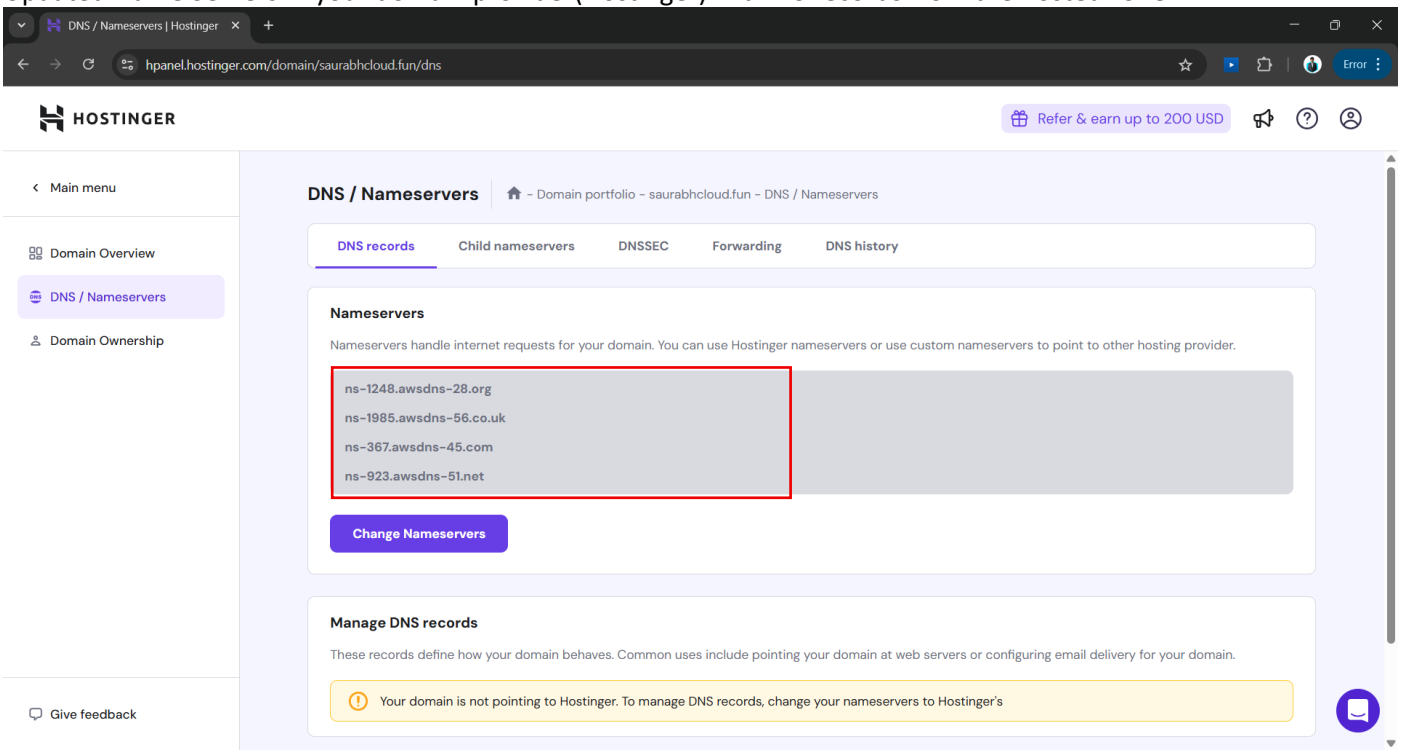


Part 7: Website Hosting using Route53 and Cloudfront

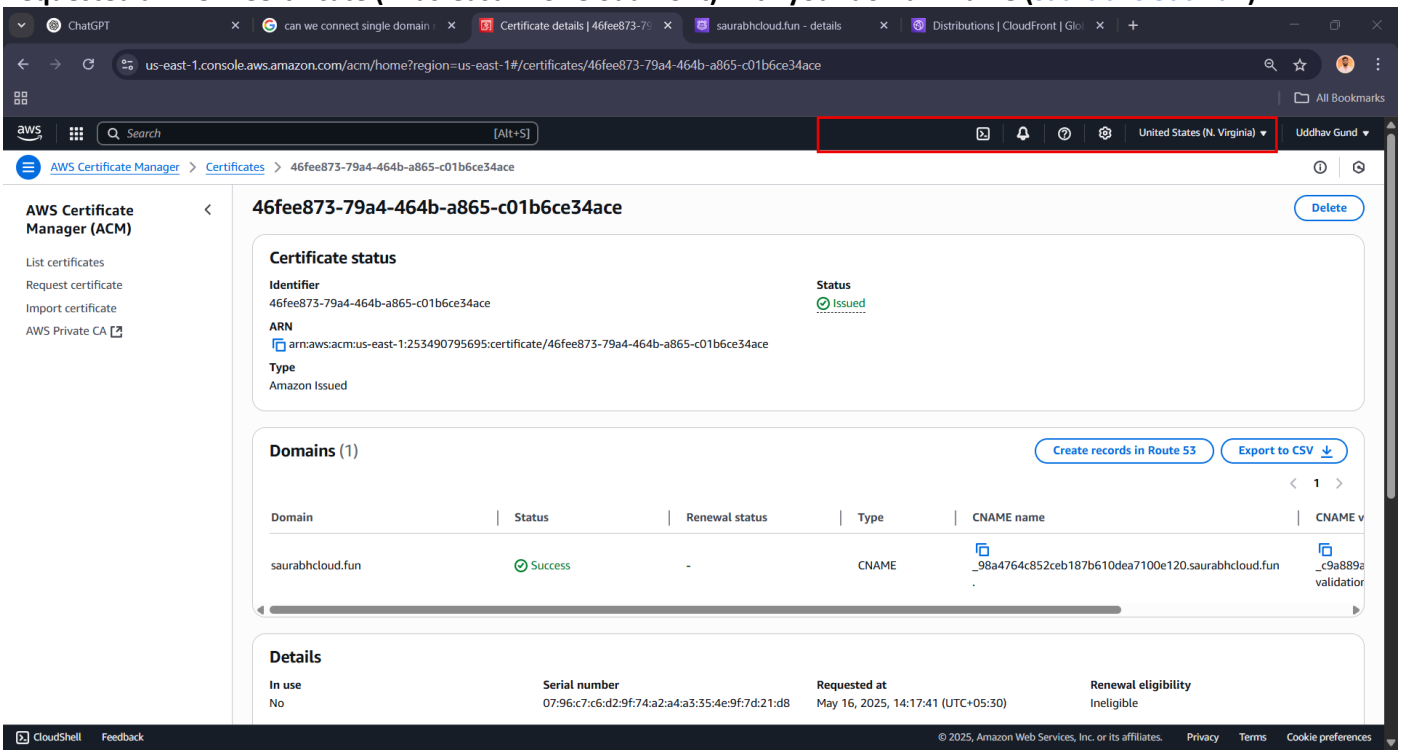
Create Hosted Zone in Route 53 for your domain (saurabhcloud.fun).



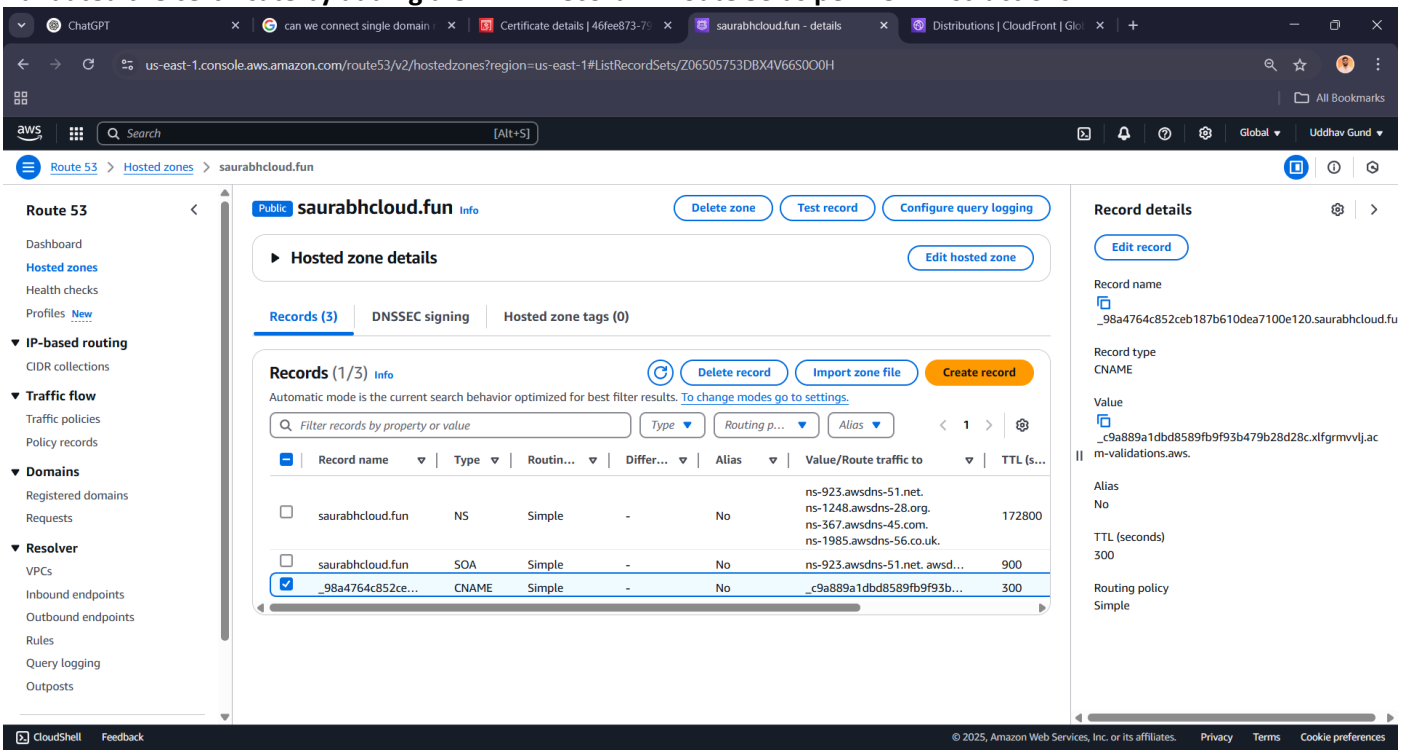
Updated Name Servers in your domain provider(Hostinger) with NS records from the hosted zone.



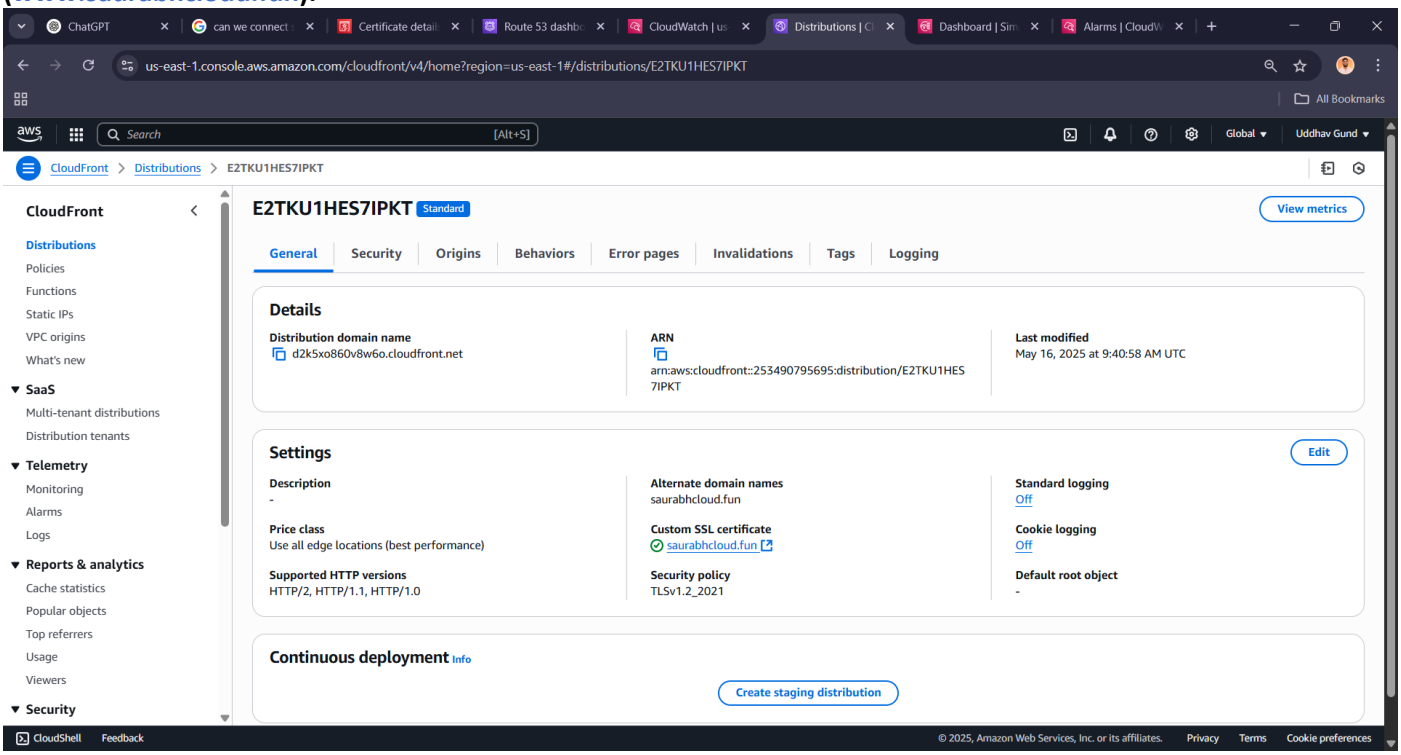
Requested an ACM Certificate (in us-east-1 for CloudFront) with your domain name (saurabhcloud.fun).



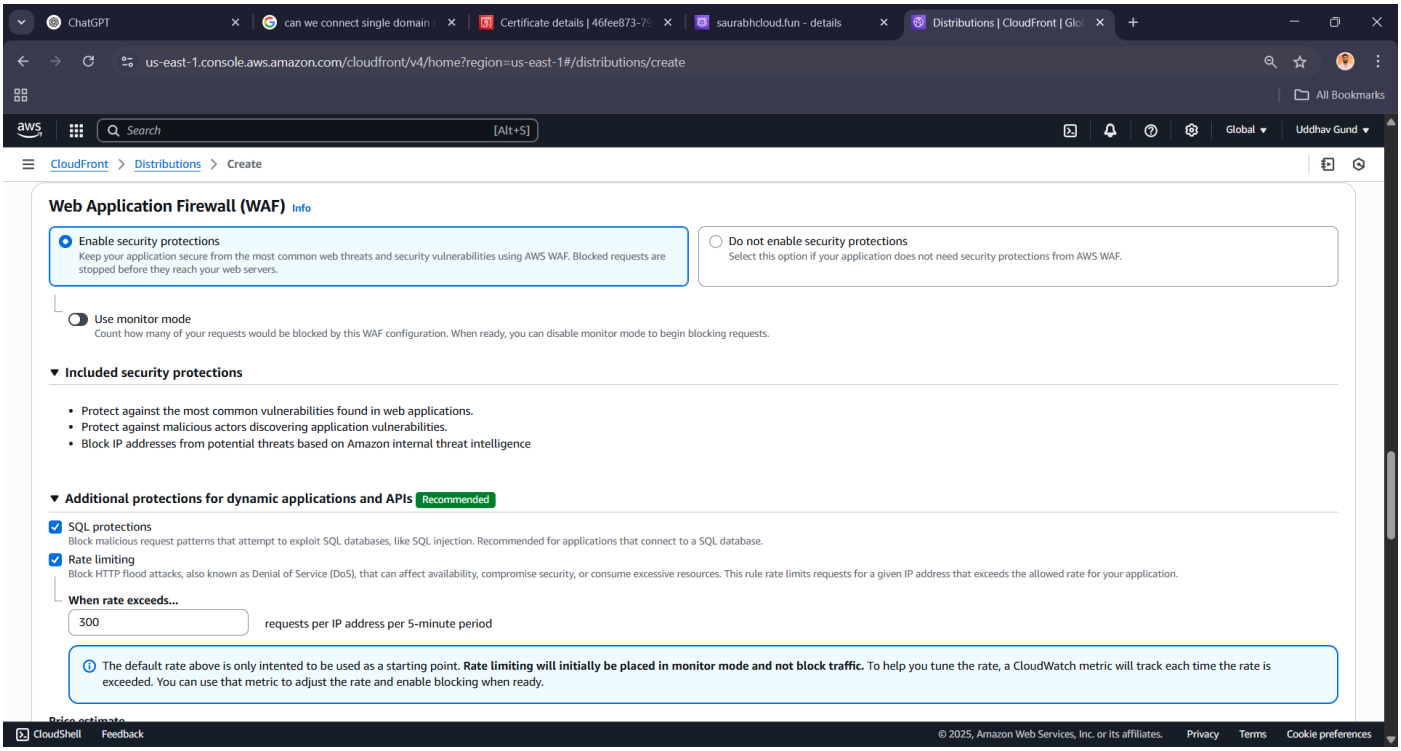
Validated the certificate by adding a CNAME record in Route 53 as per ACM instructions.



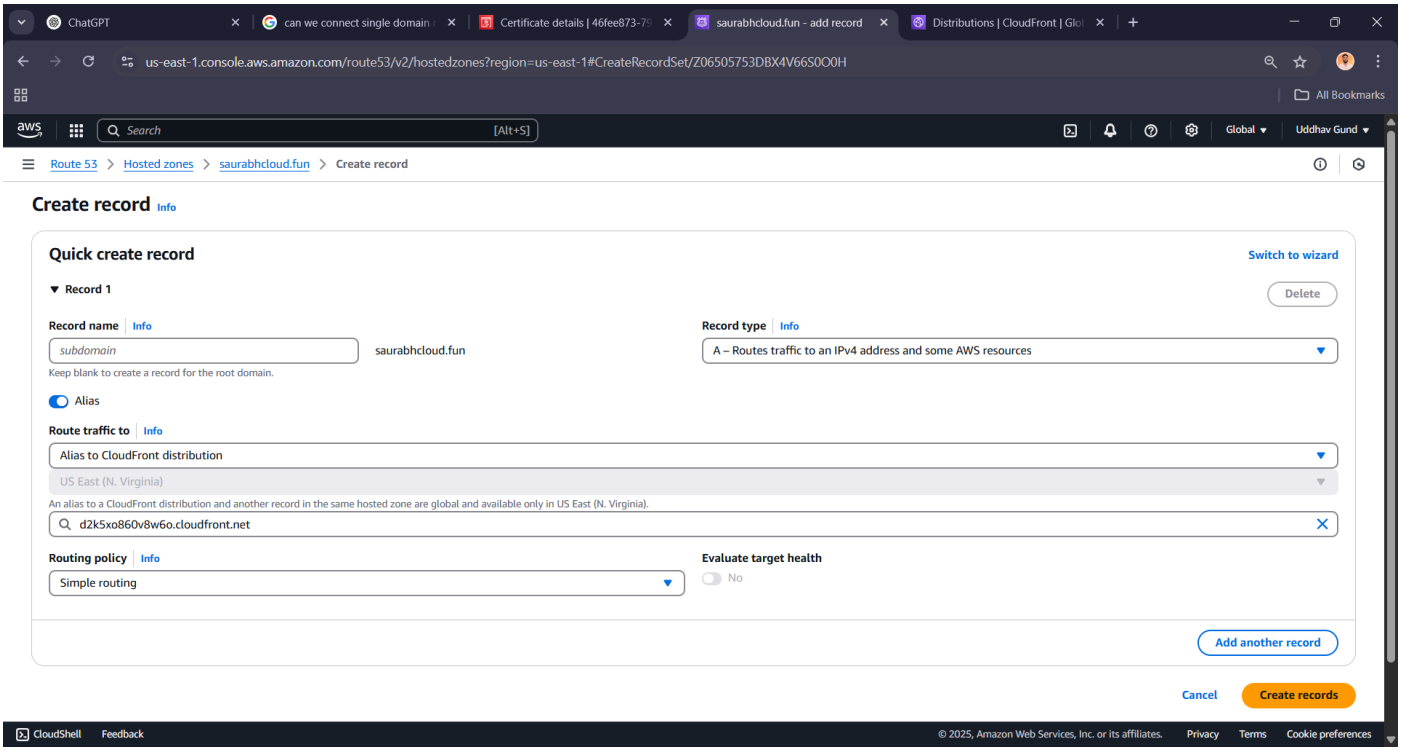
Created a CloudFront distribution with the ALB DNS as the origin and custom domain name set (www.saurabhcloud.fun).



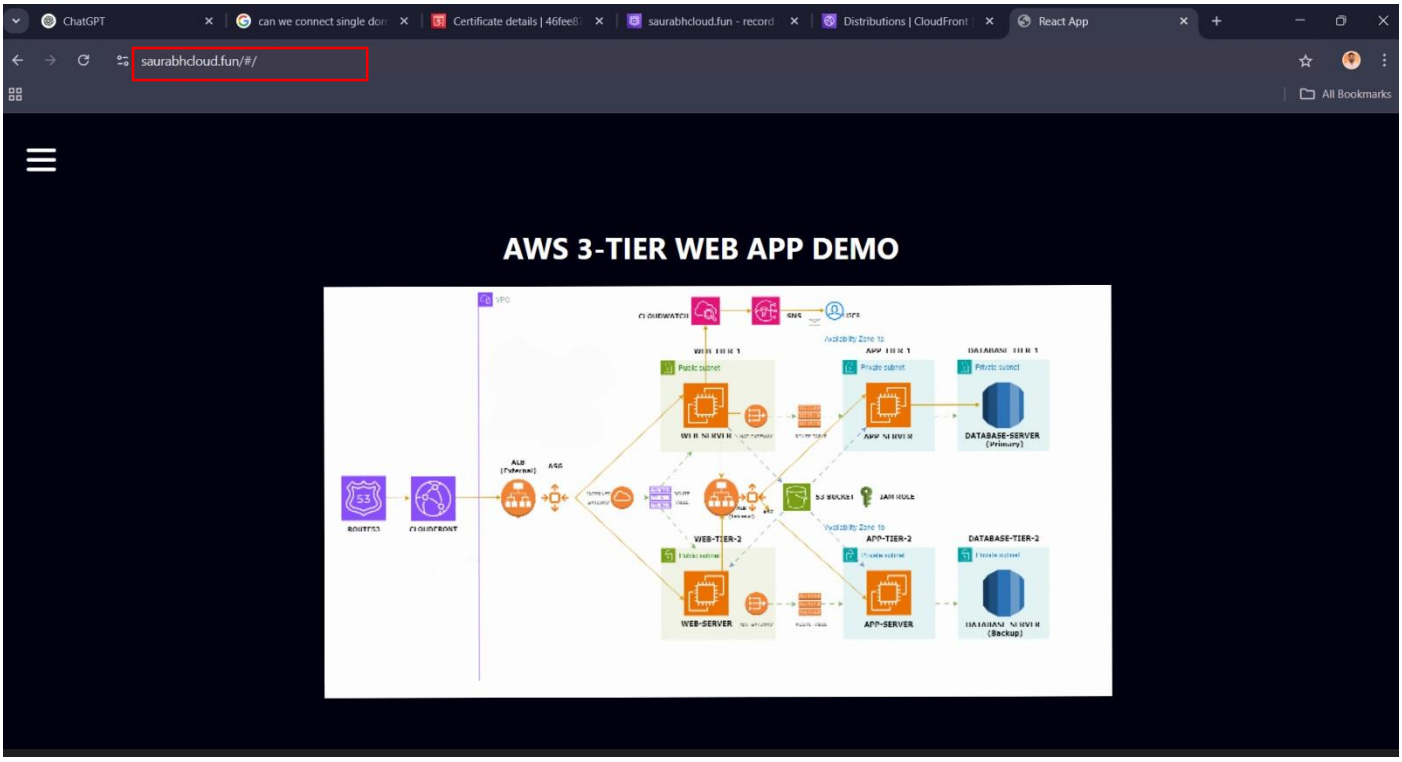
Also Attached the WAF Web ACL to your CloudFront distribution via its settings.

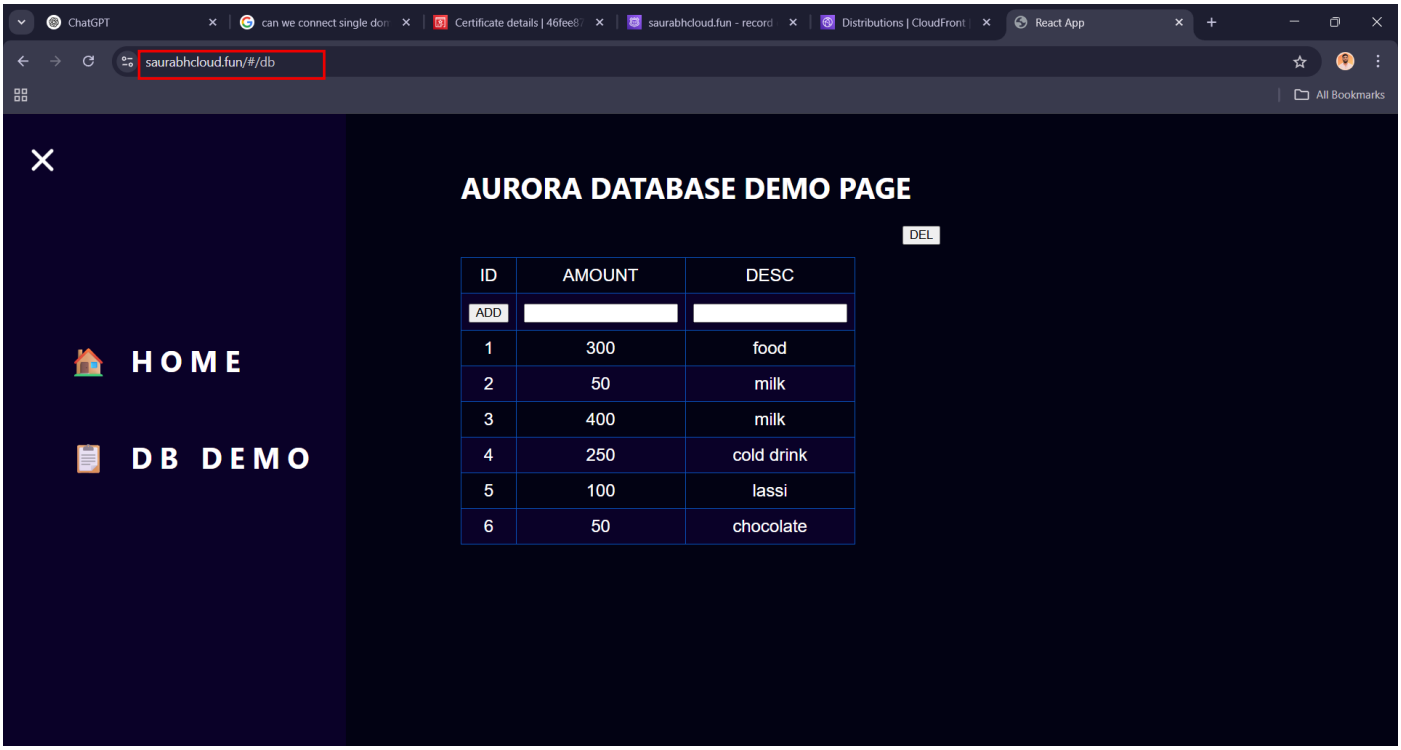


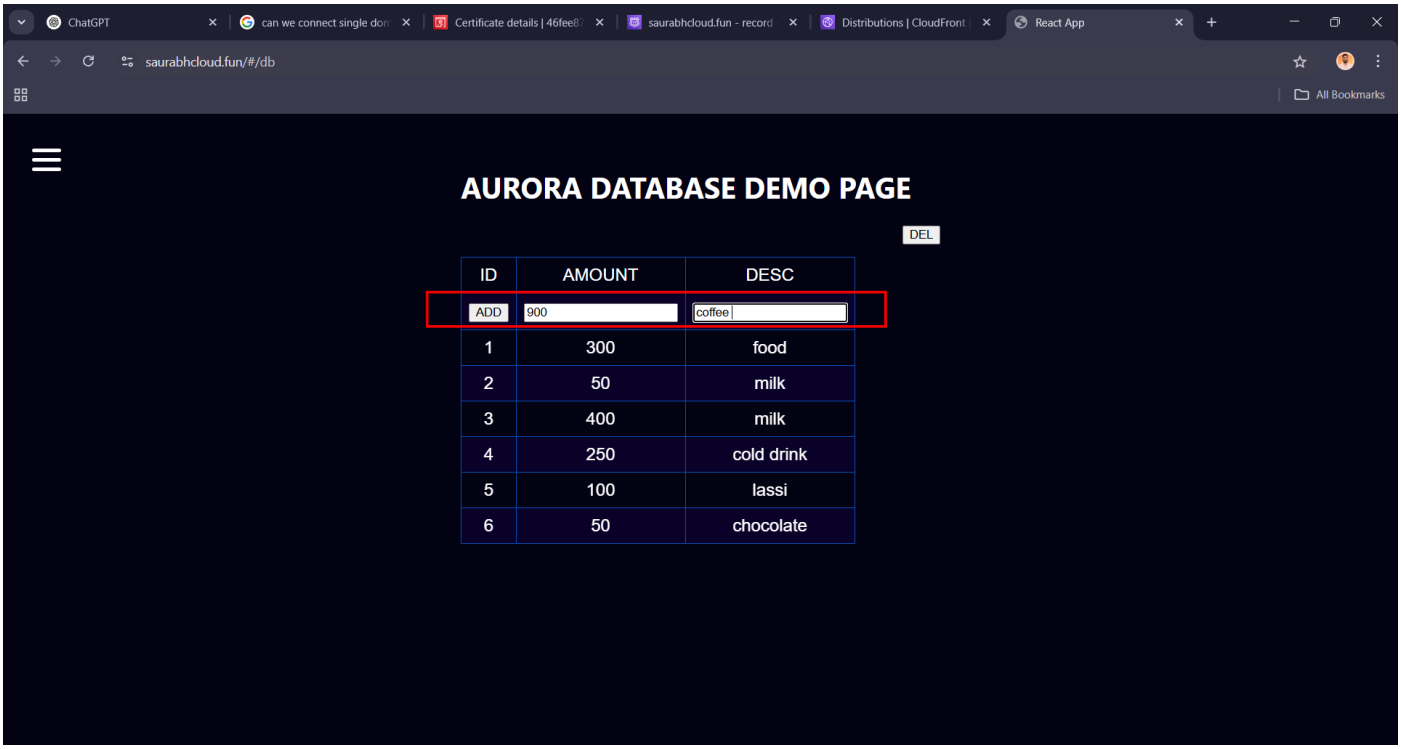
Created an A (Alias) Record in Route 53 pointing www.saurabhcloud.fun to the CloudFront distribution.

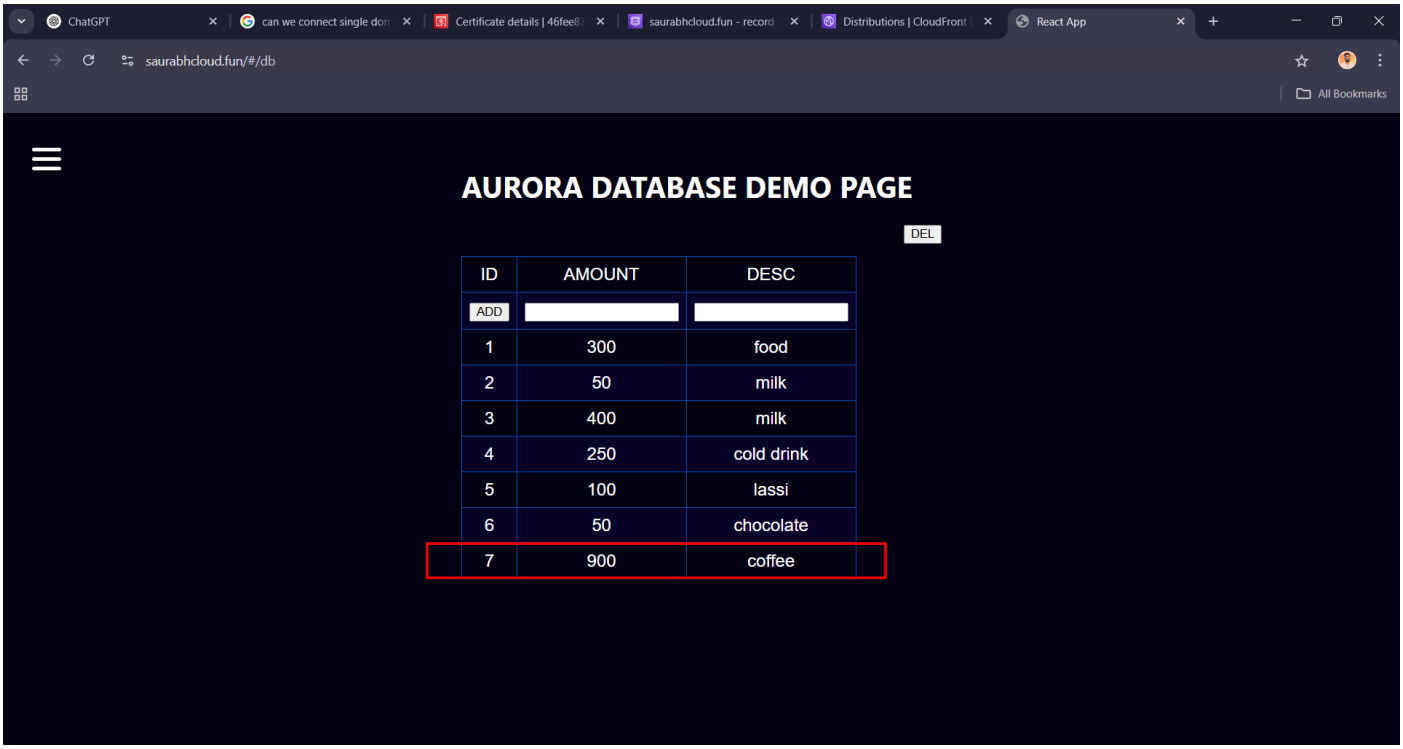


Afterthat Accessed My web application with Domain Name saurabhcloud.fun

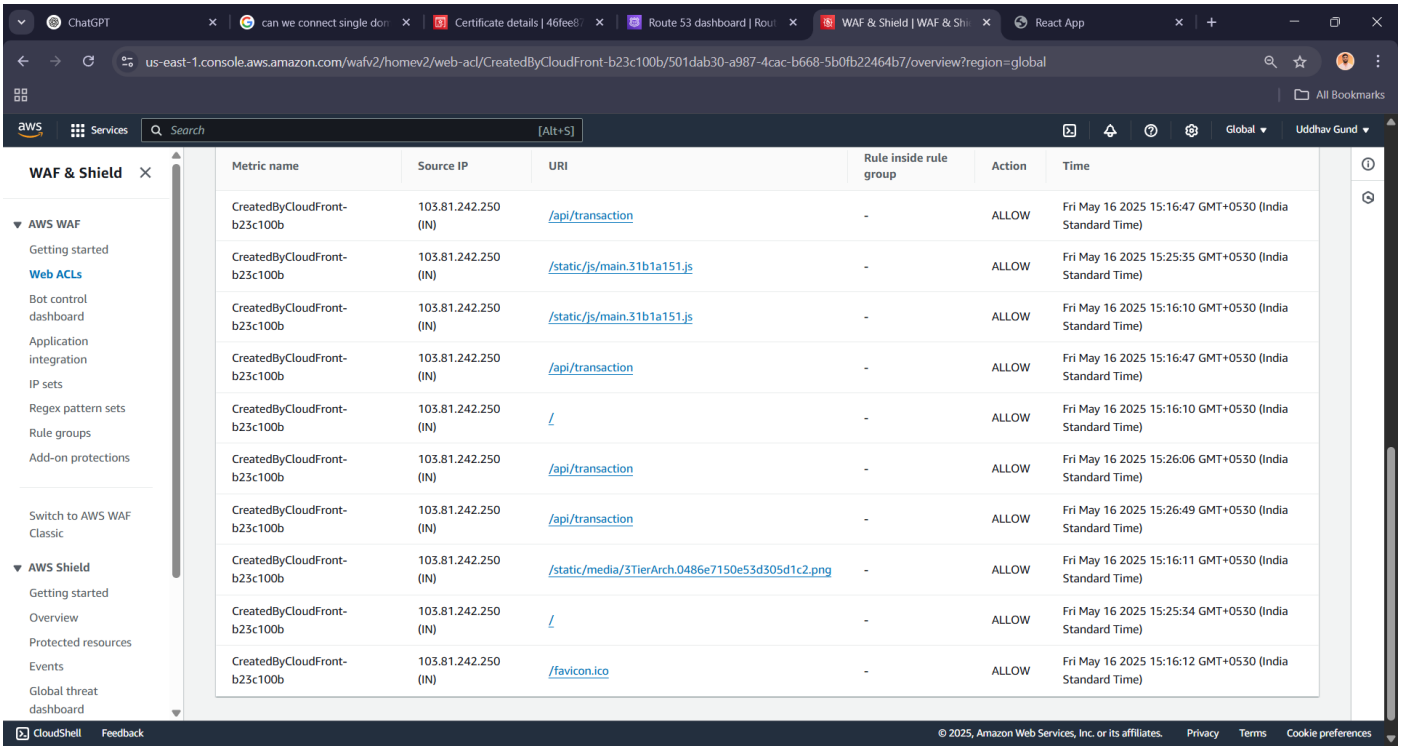


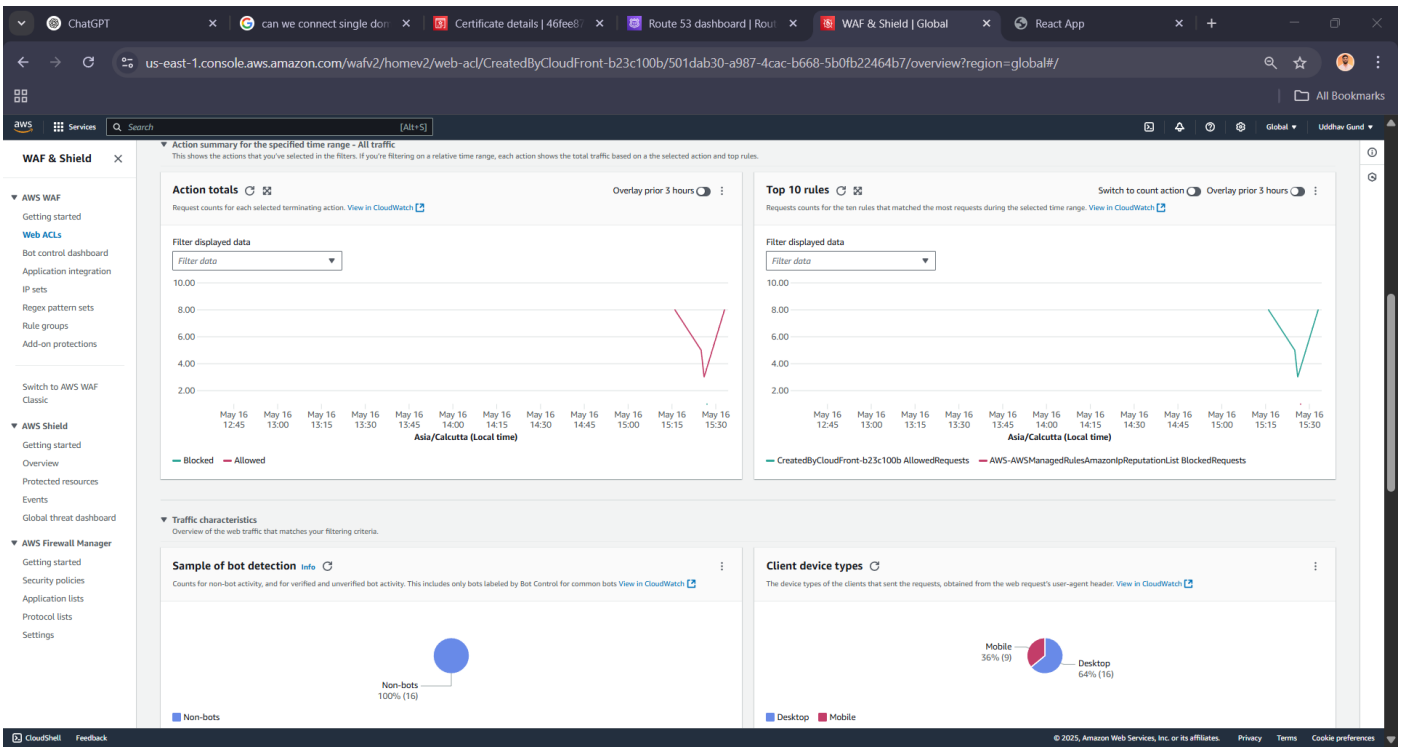


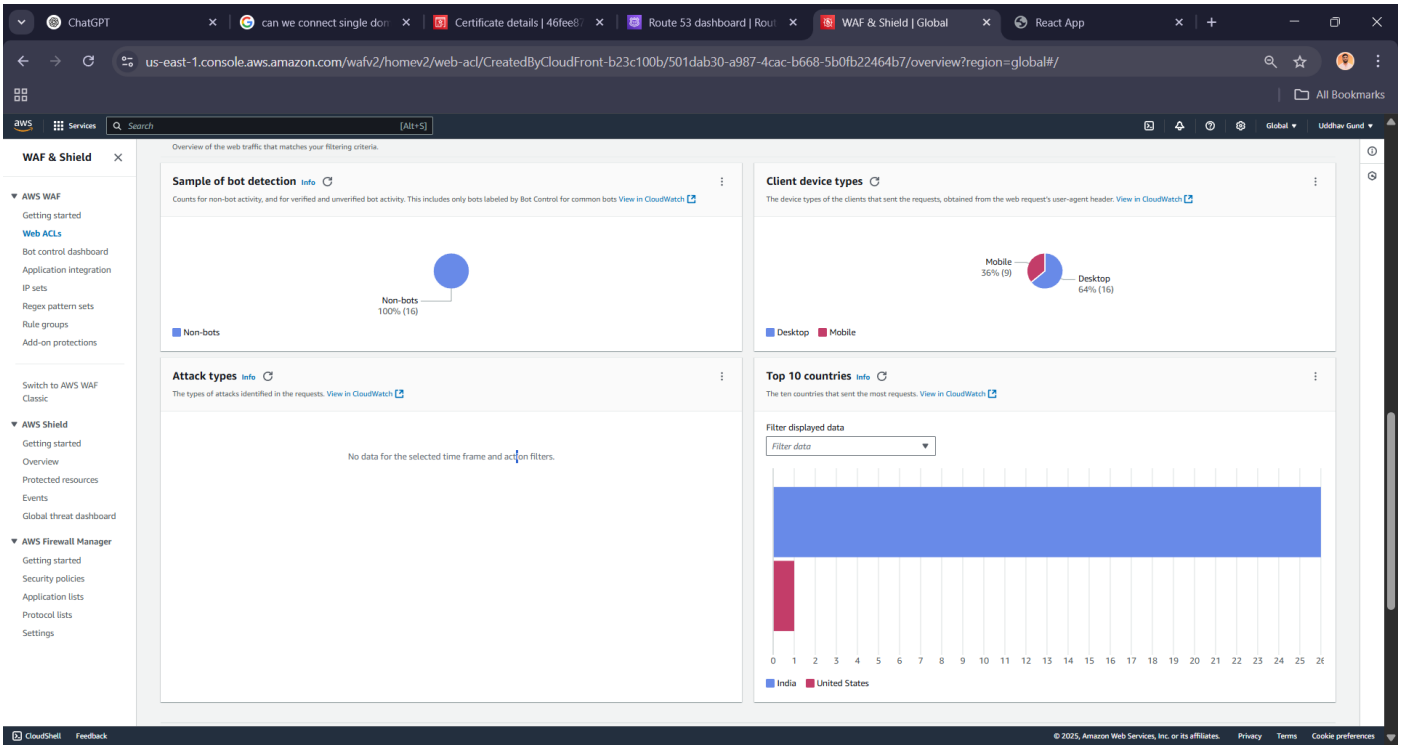

ChatGPT
can we connect single dom
Certificate details | 46fee87
saurabhcloud.fun - record
Distributions | CloudFront
React App
saurabhcloud.fun/#/db
All Bookmarks
AURORA DATABASE DEMO PAGE
DEL
ID
AMOUNT
DESC
ADD
900
coffee
1 300 food
2 50 milk
3 400 milk
4 250 cold drink
5 100 lassi
6 50 chocolate


ChatGPT
can we connect single dom
Certificate details | 46fee87
saurabhcloud.fun - record
Distributions | CloudFront
React App
saurabhcloud.fun/#/db
All Bookmarks
AURORA DATABASE DEMO PAGE
DEL
ID
AMOUNT
DESC
ADD
1 300 food
2 50 milk
3 400 milk
4 250 cold drink
5 100 lassi
6 50 chocolate
7 900 coffee

All WAF ACL's Report:



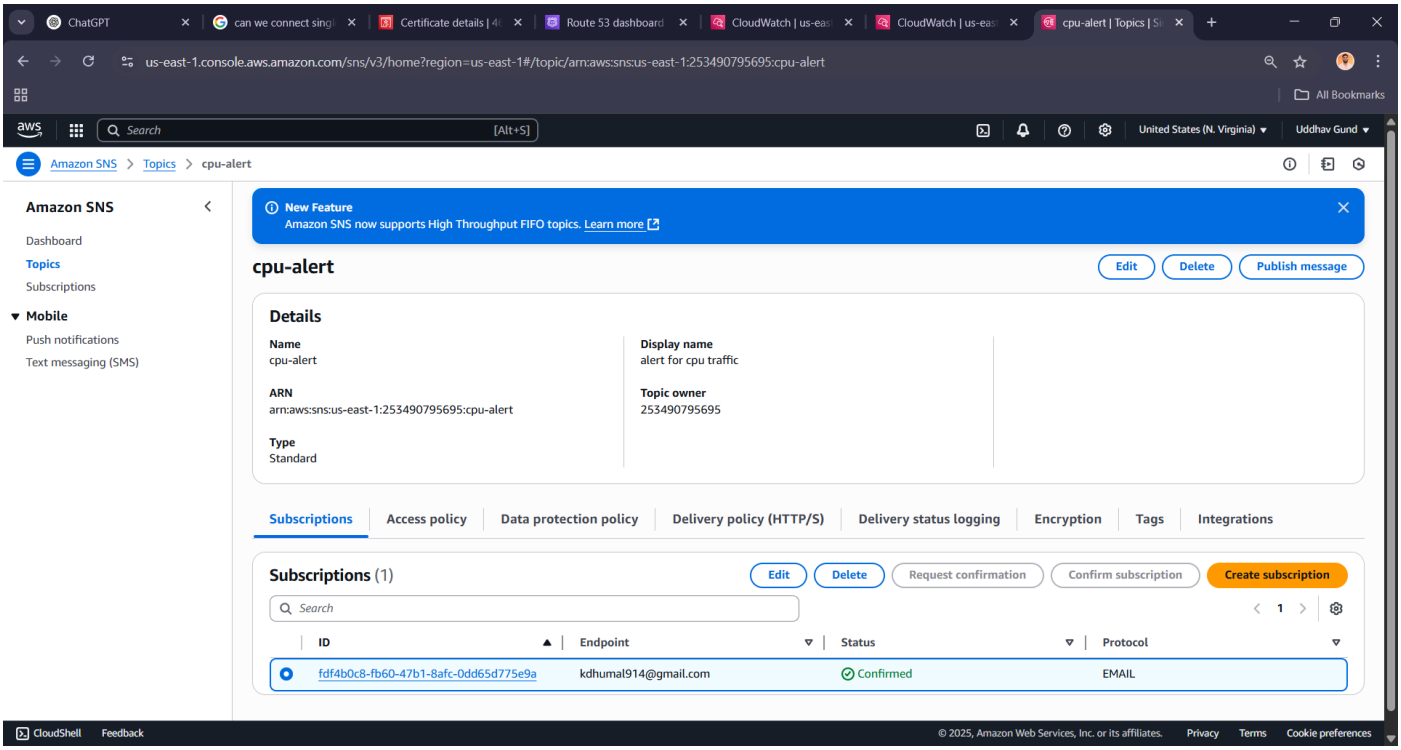




Part 8: CloudWatch and SNS integration

Created an **SNS topic (cpu-alert)** in Amazon SNS.

Subscribed my **email address to the SNS topic** and confirmed the subscription via email.

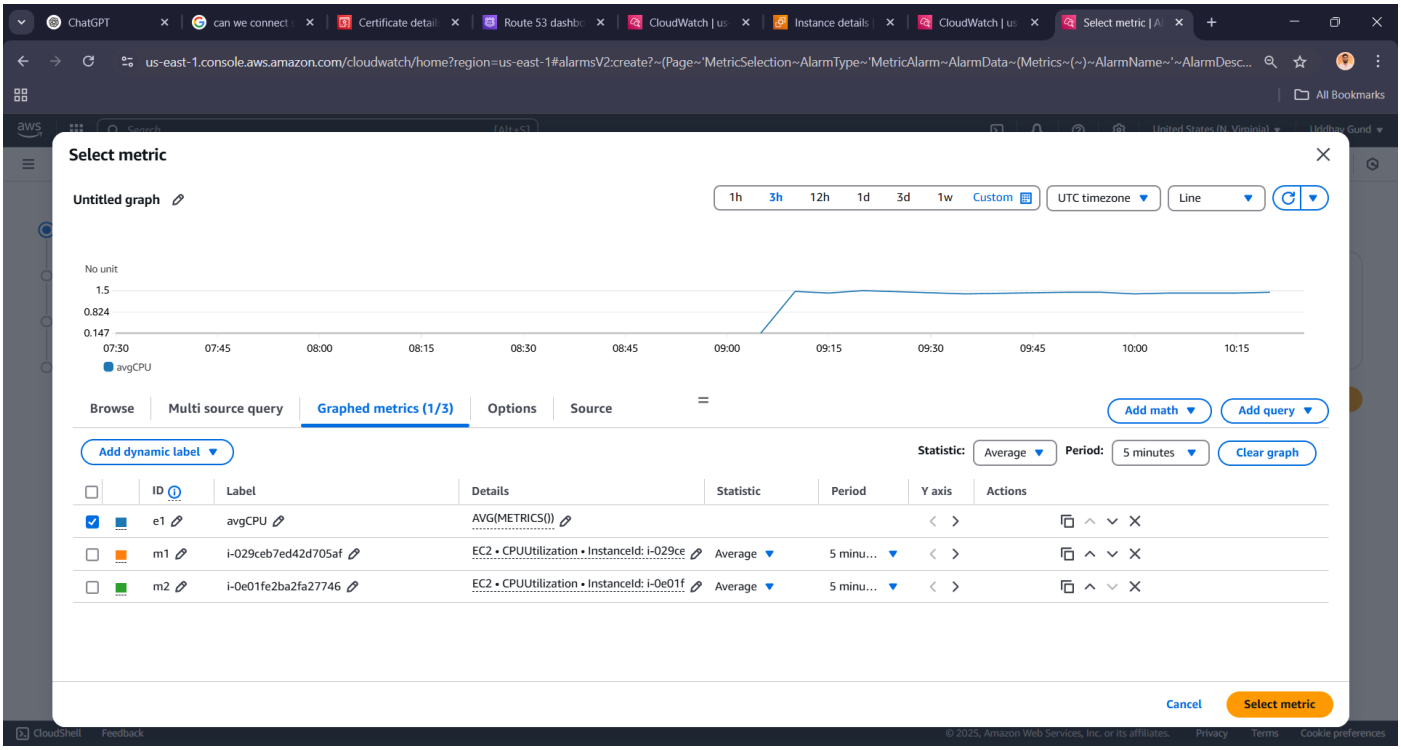


Navigated to CloudWatch → Alarms and clicked "Create Alarm".

Selected EC2 → Per-Instance Metrics → CPUUtilization for the desired instance(s).

Added a math expression (if needed) like AVG([m1, m2]) to calculate average CPU.

Chose the math expression or single instance metric as the alarm target.

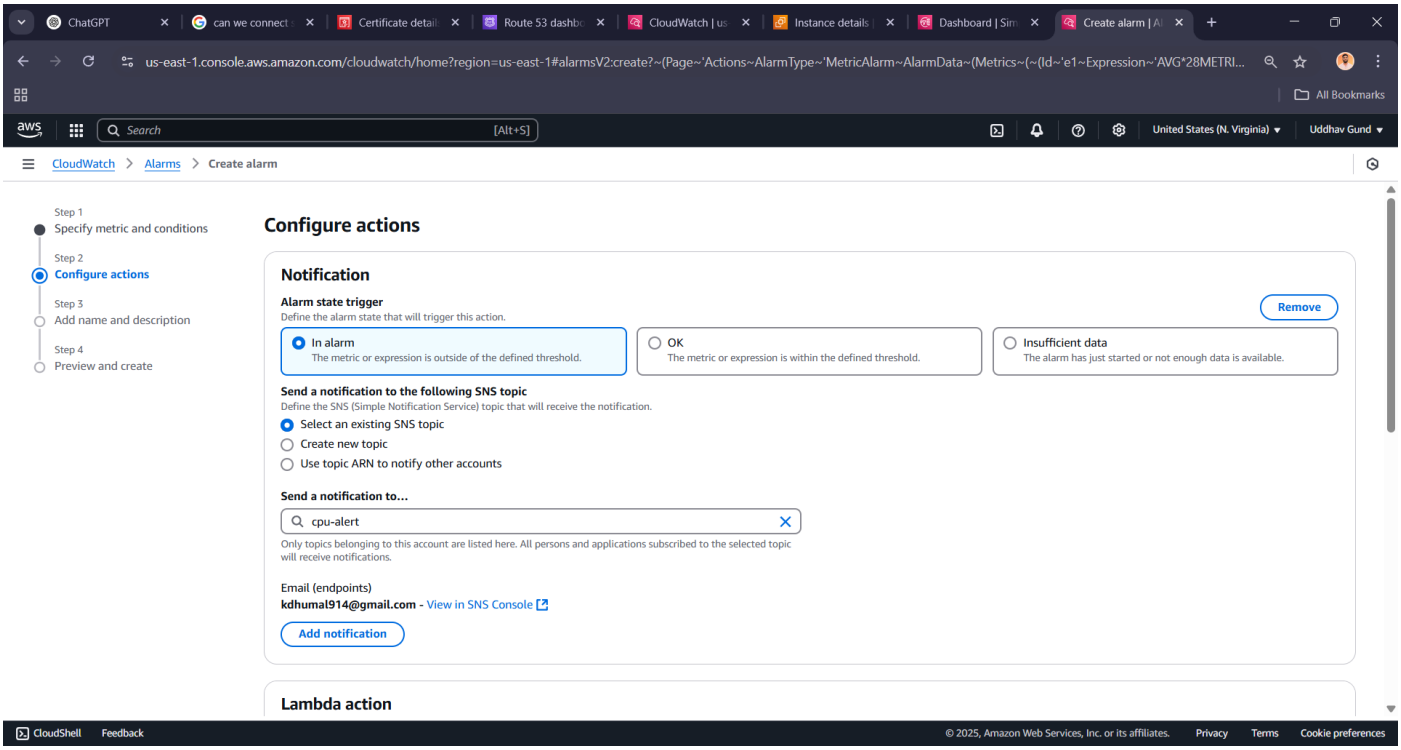


Chose the math expression or single instance metric as the alarm target.

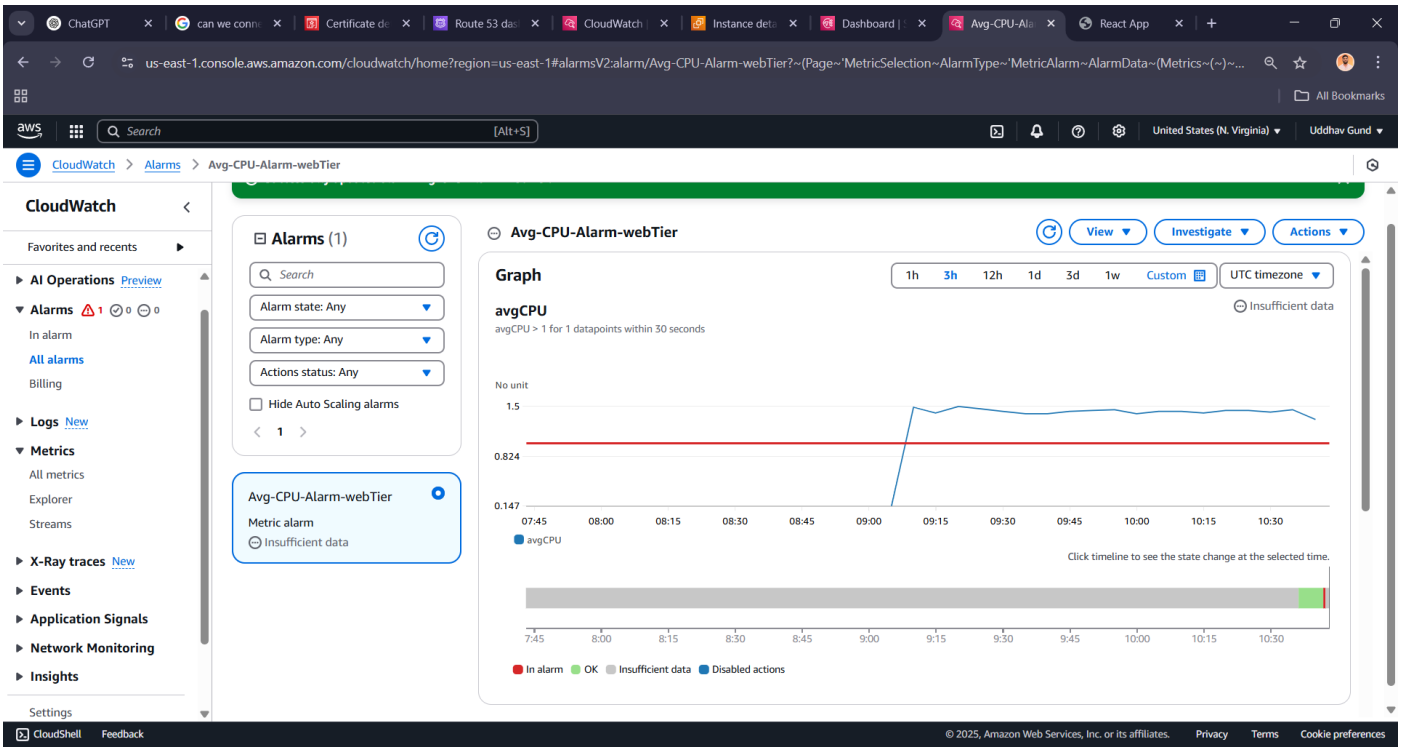
Configured the alarm condition (e.g., CPUUtilization > 1% for 30 seconds).

Attached the previously created SNS topic as the notification action.

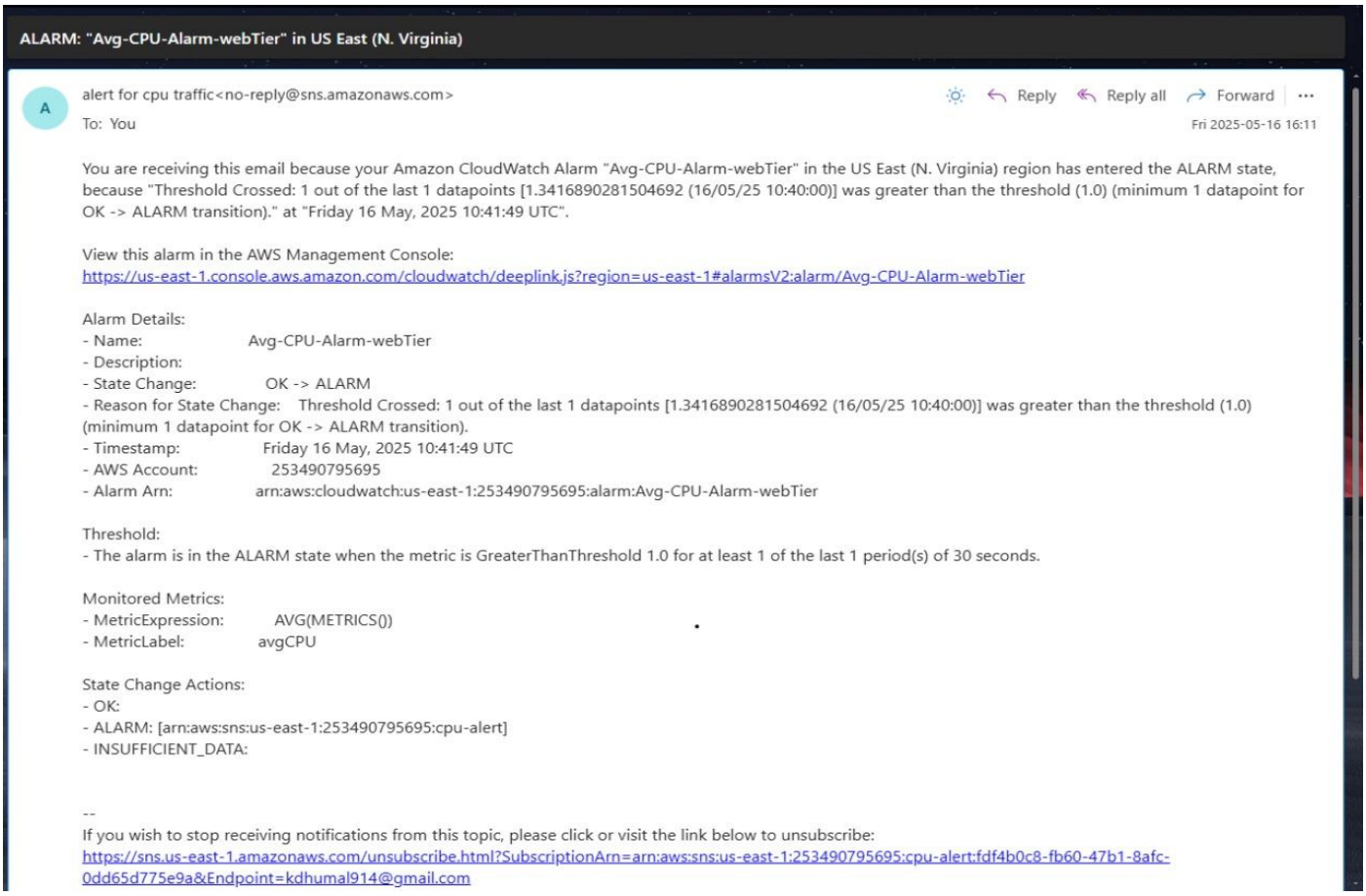
Named the alarm and created it.



Triggered high CPU usage to test the alarm.



Received an **email notification from SNS** when the alarm state changed to ALARM.



Successfully! Implemented AWS 3-Tier Web Architecture!