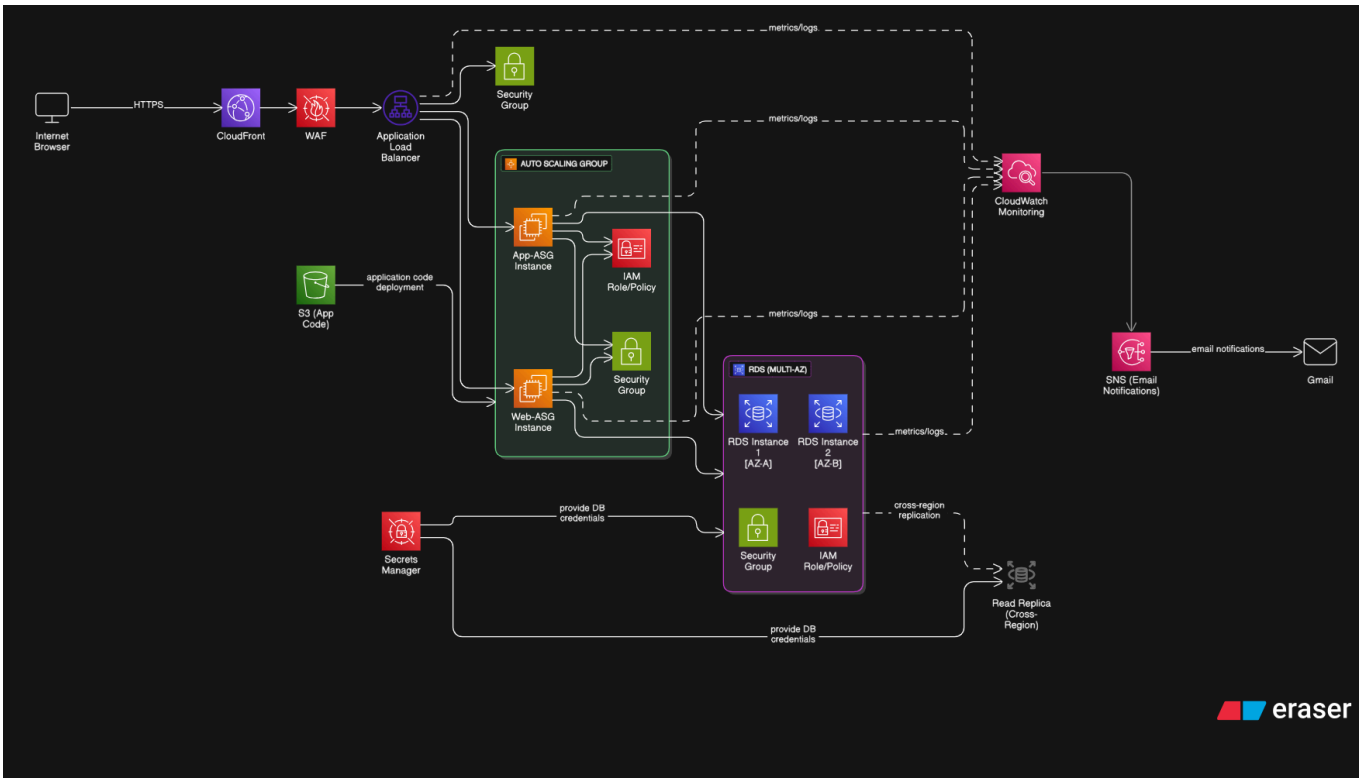
Three Tier 15 Microservices Application

In this project I will be showing you step-by-step how I built and deployed a three tier application which had 15 microservices on AWS EC2 with Multi-AZ RDS instances.



This project is by **Harish Shetty** and it has a lot of AWS services used as you can see in the project diagram. There are EC2 instances in an Auto-Scaling Group which has the **Application Code** and the **RDS Database** also has two instances for **High-Availability** with its credentials stored in a **Secrets Manager**.

Step-by-Step Project Guide:

- Deploying the infrastructure
- Configuring the Jump Servers
- Createing Jump Servers AMIs
- Creating a CloudTrail
- Cloning the Application Repository
- Creating the RDS
- Creating Launch Templates
- Creating Target Groups
- Creating Load Balancers
- Updating Application Files

- Creating Auto Scaling Groups
- Accessing the Application
- Creating a CloudFront Distribution
- Customizing WAF and CloudFront
- Testing Auto Scaling

1. Deploying the infrastructure

First of all, its infrastructure is mainly deployed and provisioned with **Terraform** by deploying its main services like **VPC**, **Security Groups**, **EC2 Instances**, **S3 Buckets**, **RDS Secrets**, **IAM Roles**, and **SNS Notifications**.

- Download these terraform files from here.
- Run `ssh-keygen` and name it `3-tier-app` to make an **EC2 Key-Pair**.
- In this first step, don't include the `secrets.tf` file as we don't have the **RDS Endpoint** yet.
- Update the `variables.tf` file according to you.
- Execute these commands to deploy the infrastructure:

```
terraform init
terraform validate
terraform plan
terraform apply -auto-approve
```

This will take some time to deploy the infrastructure.

2. Configuring the Jump Servers

Now you have to configure the **Jump Servers** which will then be used for deploying your application automatically in an **Auto-Scaling Group**.

- SSH into `jump-server-web`.
- Run these commands to install **nginx** and **git**:

```
sudo yum install nginx -y
sudo systemctl start nginx
sudo systemctl enable nginx
sudo service nginx restart
sudo chkconfig nginx on
sudo yum install git -y
```

- Run these commands to install **AWS CLI**:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

- Now run `aws configure` to configure your **Access Keys** and **Secret Access Keys**.
- Run these commands to install **Node JS**:

```
# Download and install nvm:
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash

# in lieu of restarting the shell
\. "$HOME/.nvm/nvm.sh"

# Download and install Node.js:
nvm install 22

# Verify the Node.js version:
node -v # Should print "v22.19.0".
nvm current # Should print "v22.19.0".

# Verify npm version:
npm -v # Should print "10.9.3".
```

- Now SSH into `jump-server-app` .
- Run these commands to install **MySQL Client**:

```
sudo wget https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm
sudo dnf install mysql80-community-release-el9-1.noarch.rpm -y
sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2023
sudo dnf install mysql-community-client -y
mysql --version
```

- Run these commands to install **AWS CLI**:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

- Now run `aws configure` to configure your **Access Keys** and **Secret Access Keys**.
- Run these commands to install **Node JS**:

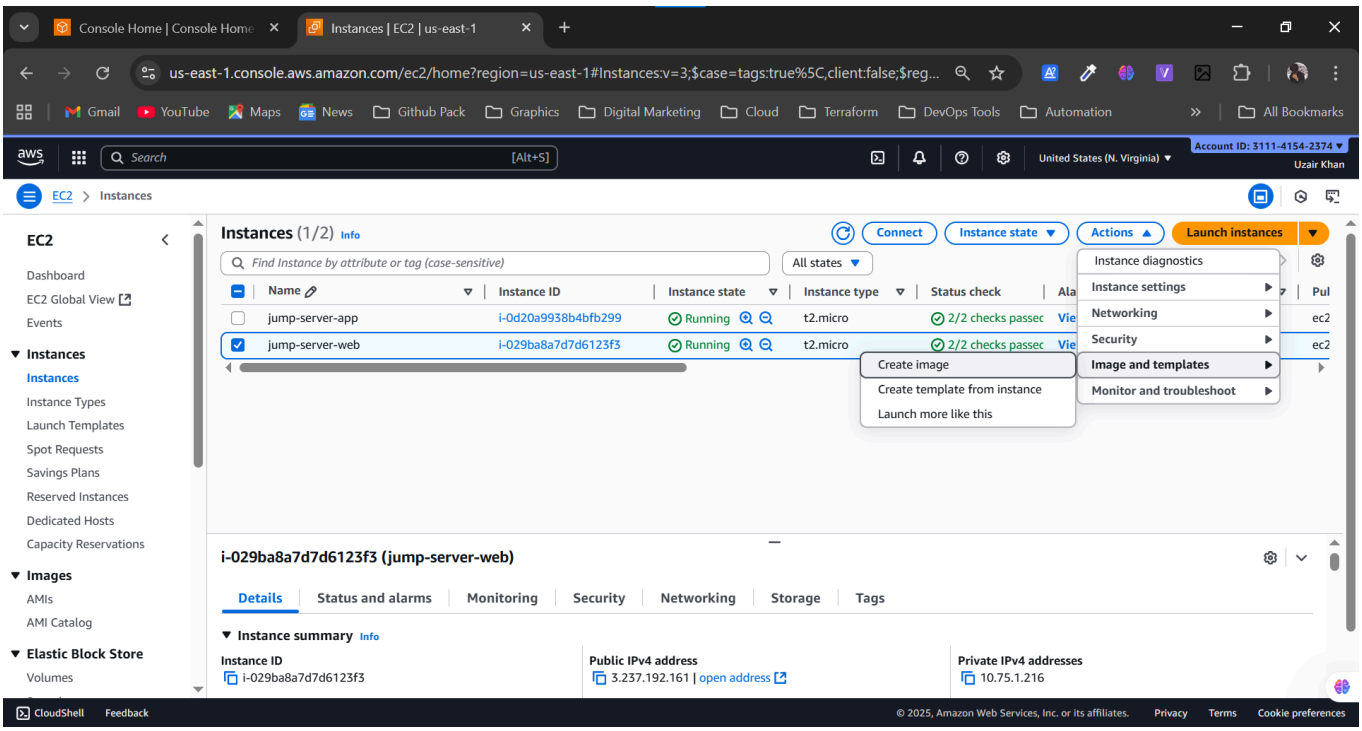
```
# Download and install nvm:
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
```

```
3
4   # in lieu of restarting the shell
5   \. "$HOME/.nvm/nvm.sh"
6
7   # Download and install Node.js:
8   nvm install 22
9   npm install -g pm2
10
11  # Verify the Node.js version:
12  node -v # Should print "v22.19.0".
13  nvm current # Should print "v22.19.0".
14
15  # Verify npm version:
16  npm -v # Should print "10.9.3".
```

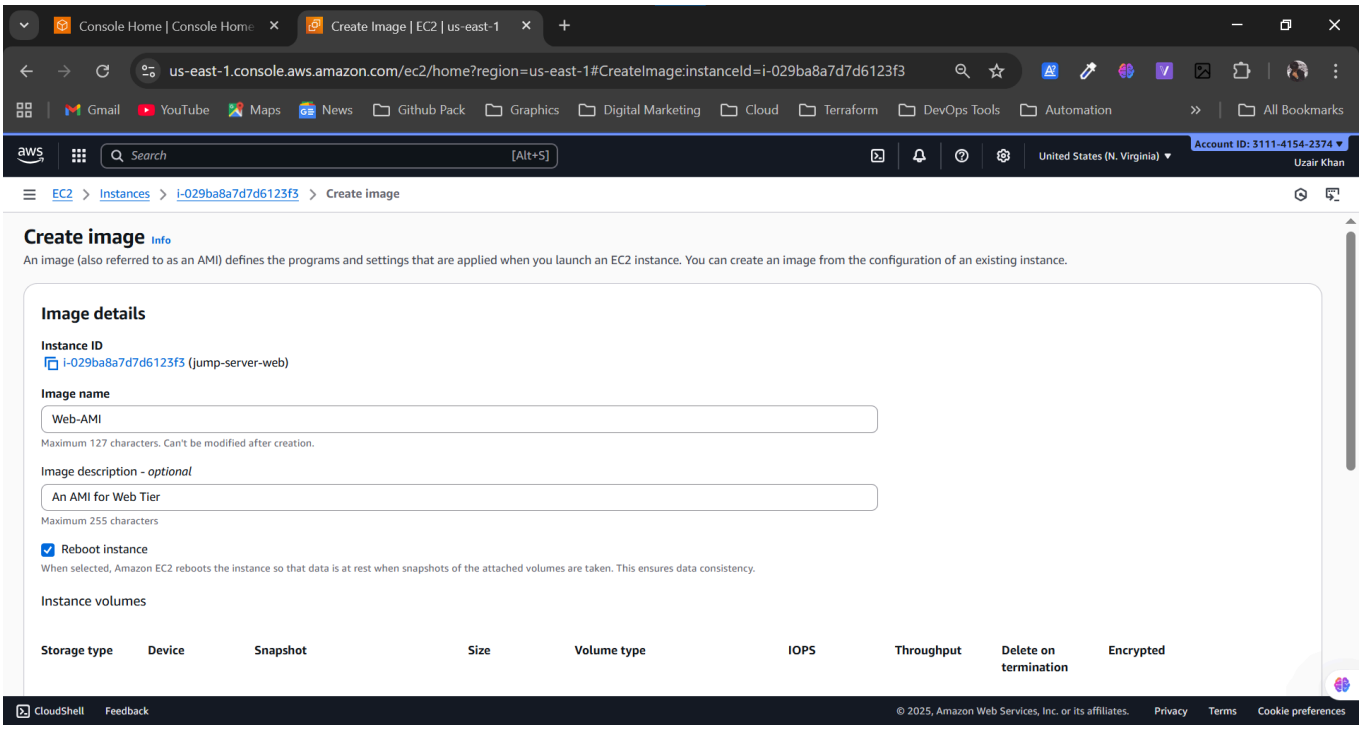
3. Creating Jump Servers AMIs

Now you'll have to create **Amazon Machine Images (AMIs)** of the configured **Jump Servers** so that it'll be later used to create **Launch Templates**.

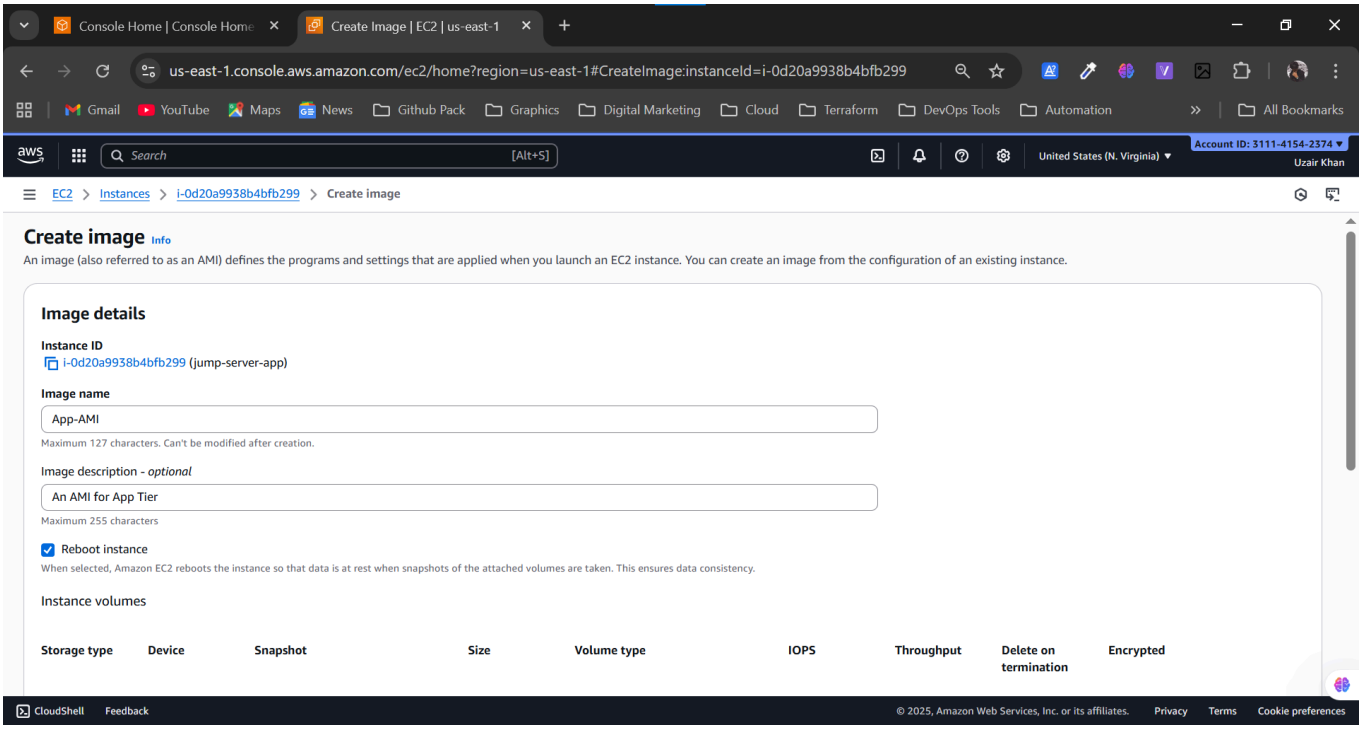
- Go to **EC2 --> Instances**.
- Select `jump-server-web`.
- Go to **Actions --> Image and templates --> Create image**.



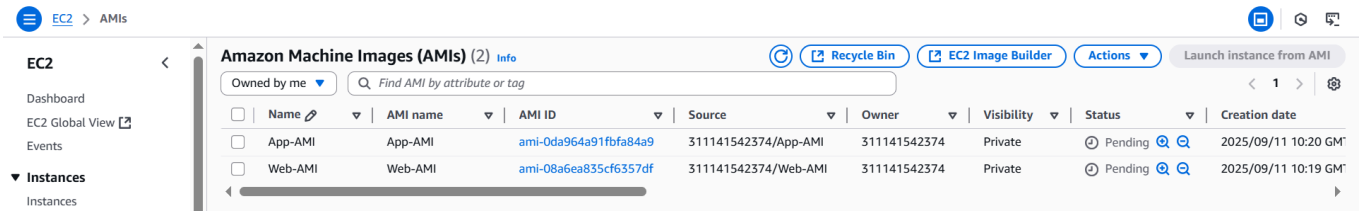
- Enter **Image name** = `Web-AMI`.
- Enter **Image description** = `An AMI for Web Tier`.



- Click **Create image**.
- Now select `jump-server-app`.
- Go to **Actions --> Image and templates --> Create image**.
- Enter **Image name** = `App-AMI`.
- Enter **Image description** = `An AMI for App Tier`.



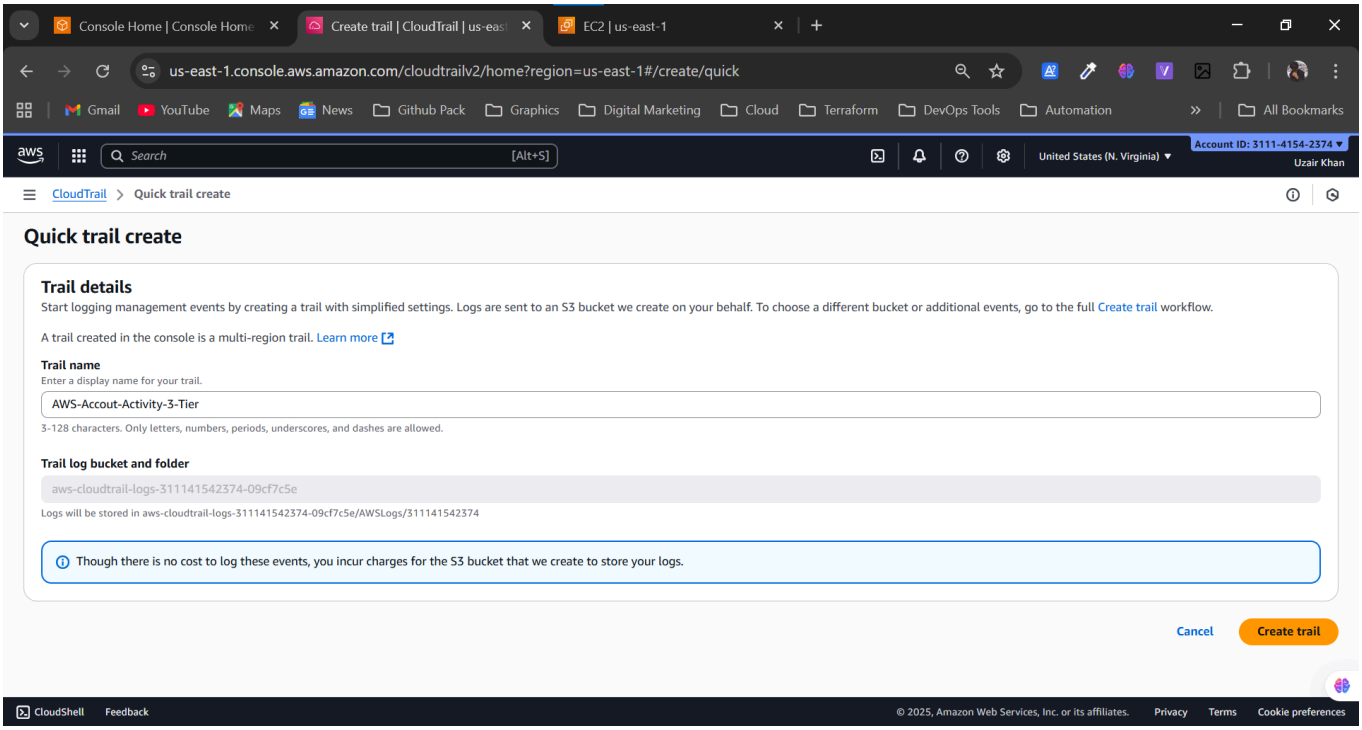
Now check both of the AMIs.



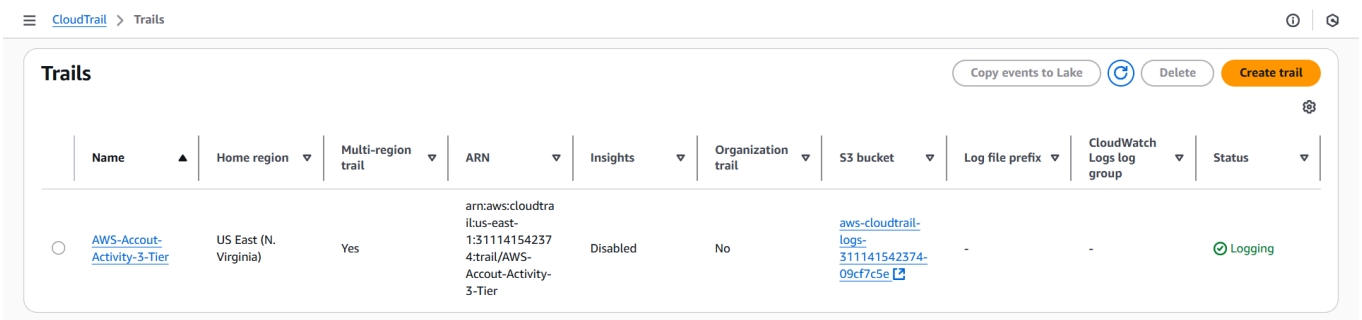
4. Creating a CloudTrail

Now you'll have to create a **CloudTrail** to track your **AWS Account Activity** and dumping it to an **S3 Bucket**.

- Go to **CloudTrail**.
- Click on **Create a trail**.
- Enter **Trail name** = `AWS-Account-Activity-3-Tier` .
- Click **Create trail**.



*Now your **CloudTrail** and an **S3 Bucket** for it will be created.*



5. Cloning the Application Repository

Now you'll have to clone the **Aplication Repository** on your **local computer**.

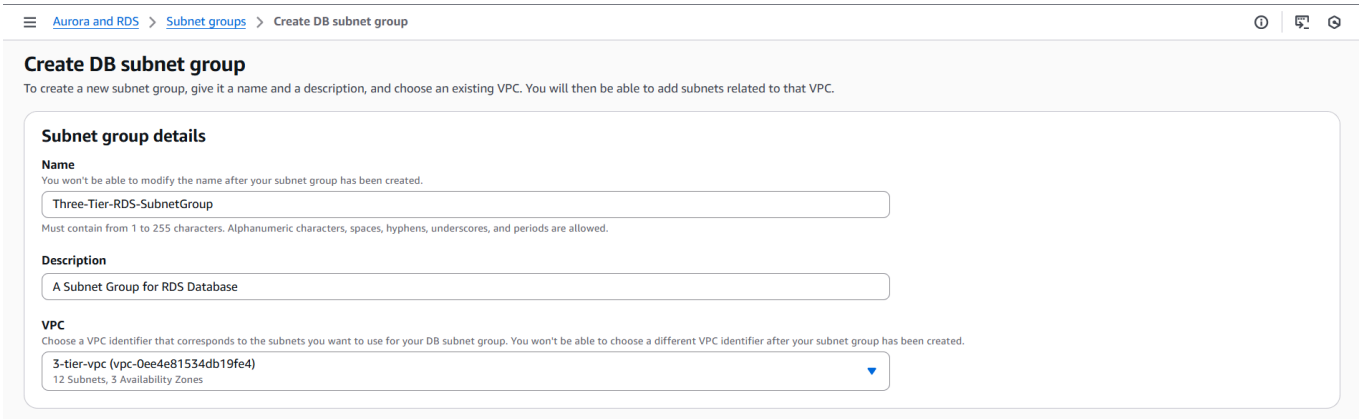
- Run this command to clone it:

```
git clone https://github.com/harishnshetty/3-tier-aws-15-services.git
```

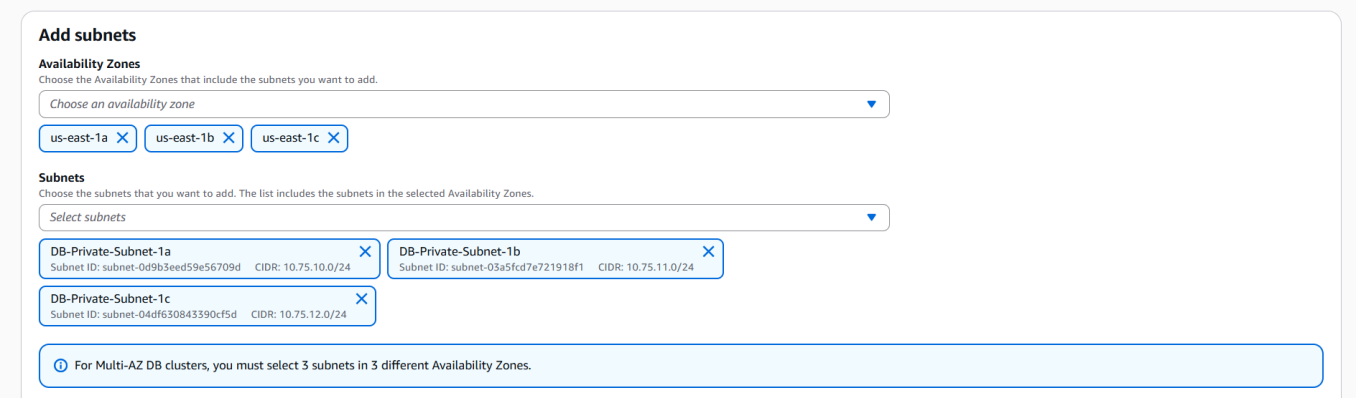
6. Creating the RDS Database

In this step, you will create an **RDS Database** with **Multi-AZ** deployment for **High-Availability** and **Fault Tolerance**.

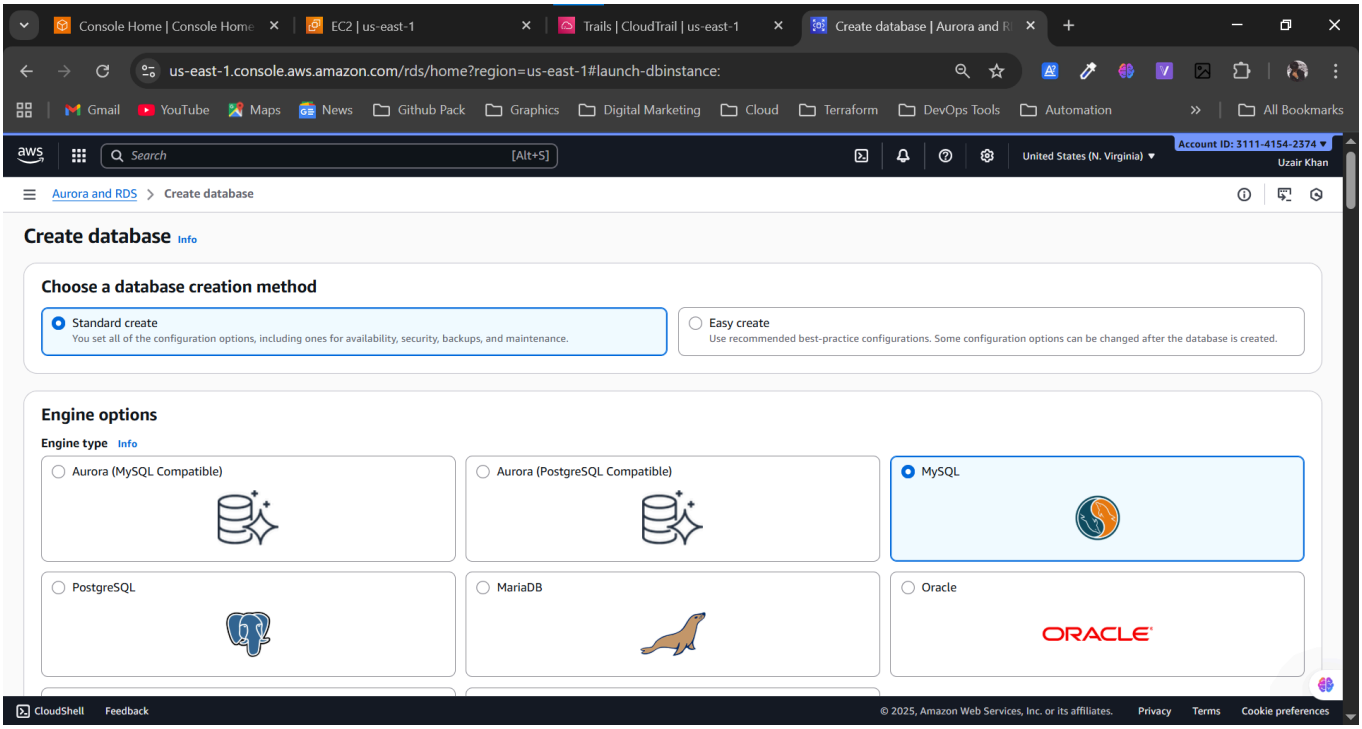
- Go to **Aurora and RDS**.
- Go to **Subnet groups**.
- Click on **Create DB subnet group**.
- Enter the following:
  - **Name** = `three-tier-rds-subnetgroup`
  - **Description** = `A Subnet Group for RDS Database`
  - **VPC** = `3-tier-vpc`



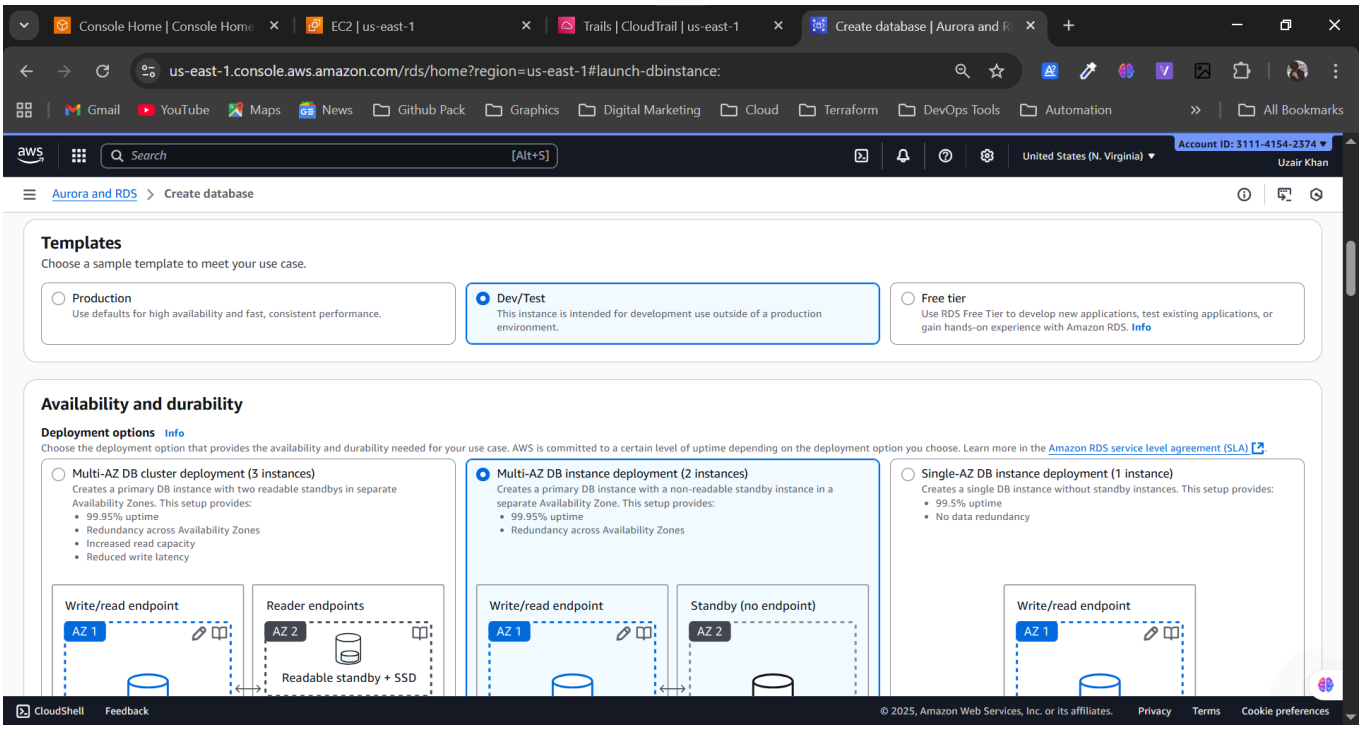
- Now select:
  - **Availability Zones** = `us-east-1a` , `us-east-1b` , `us-east-1c`
  - **Subnets** = `DB-Private-Subnet-1a` , `DB-Private-Subnet-1b` , `DB-Private-Subnet-1c`



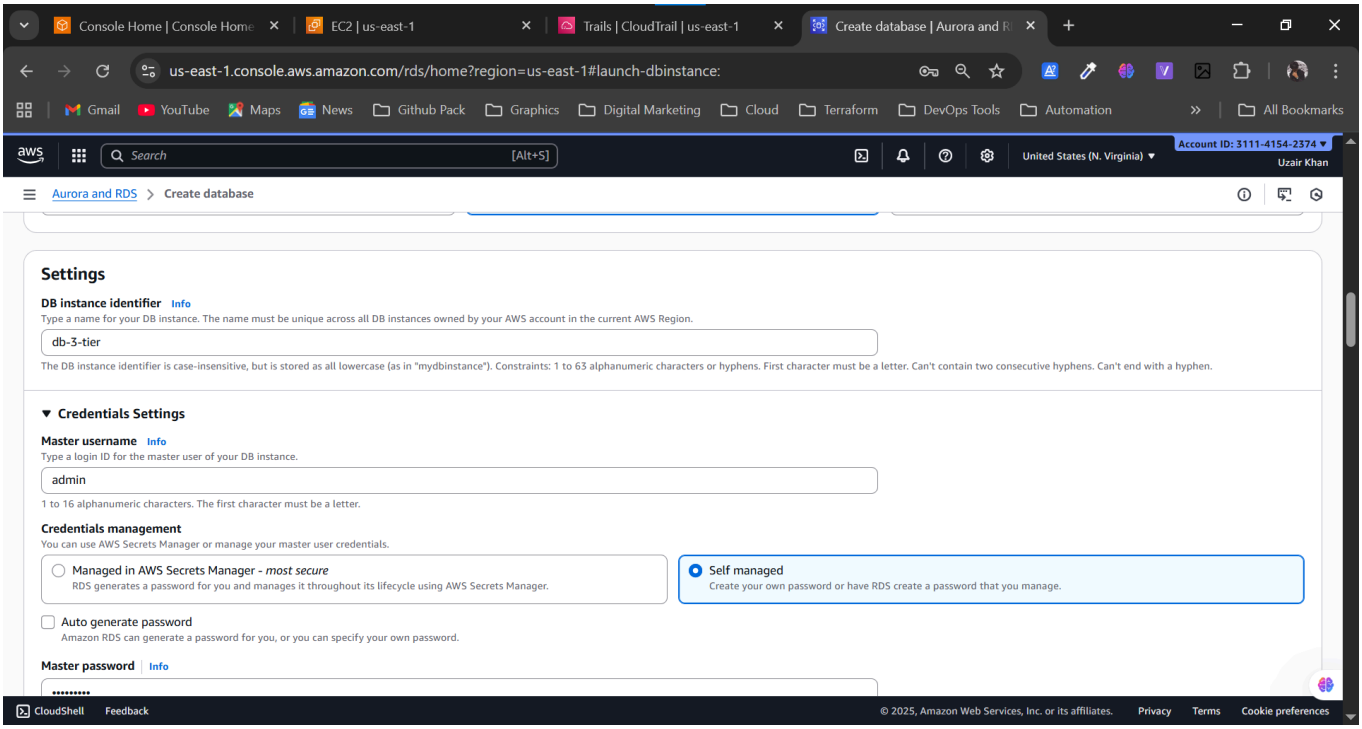
- Click **Create**.
- Now go to **Databases**.
- Click on **Create database**.
- Choose the following:
  - **Choose a database creation method** = `Standard create`
  - **Engine type** = `MySQL`



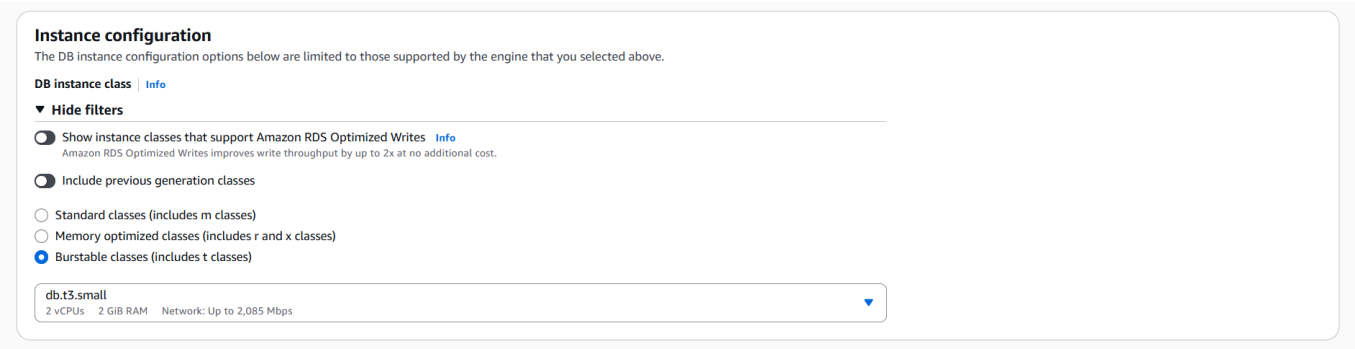
- Now choose:
  - **Templates** = `Dev/Test`
  - **Deployment options** = `Multi-AZ DB instance deployment (2 instances)`



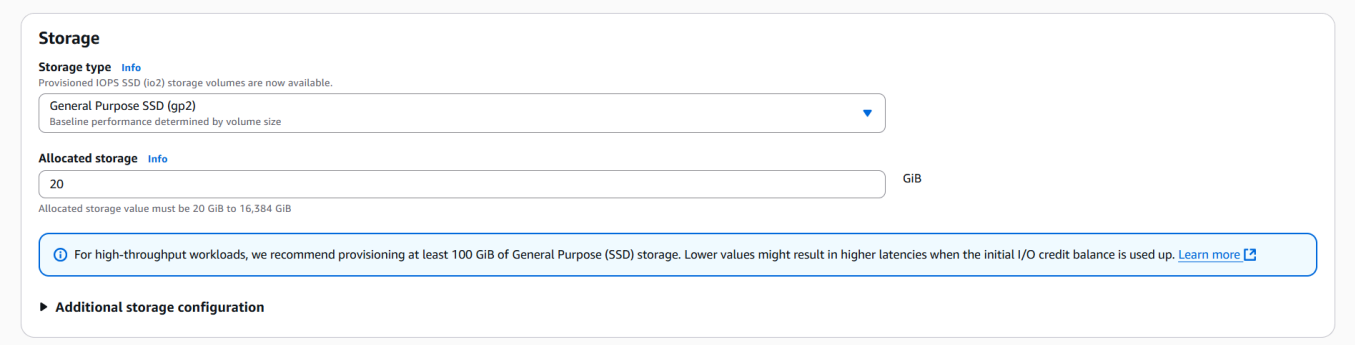
- Now Enter:
  - **DB instance identifier** = `db-3-tier`
  - **Master username** = `admin`
  - **Credentails Management** = `Self managed`
  - **Master password** = `[Your RDS DB Password]`



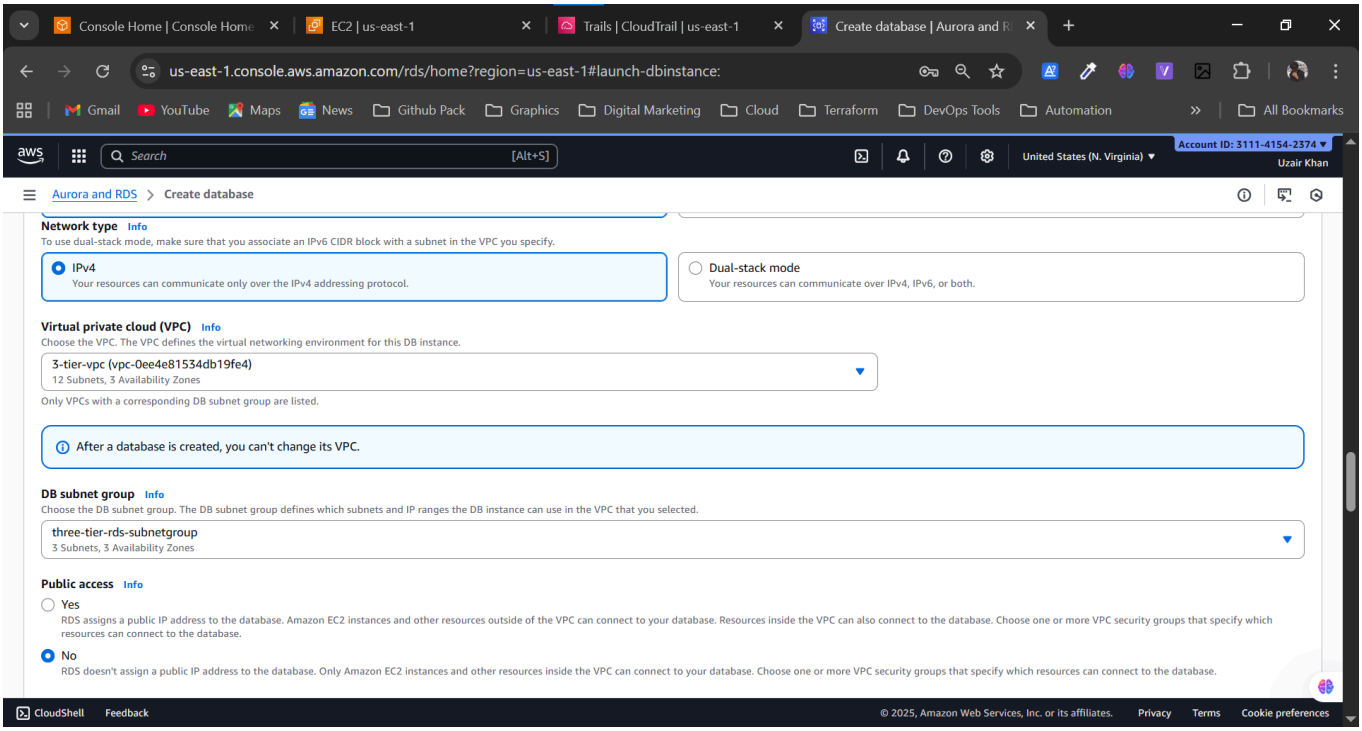
- In **Instance configuration** choose:
  - `Burstable classes (includes t classes)`
  - `db.t3.small`



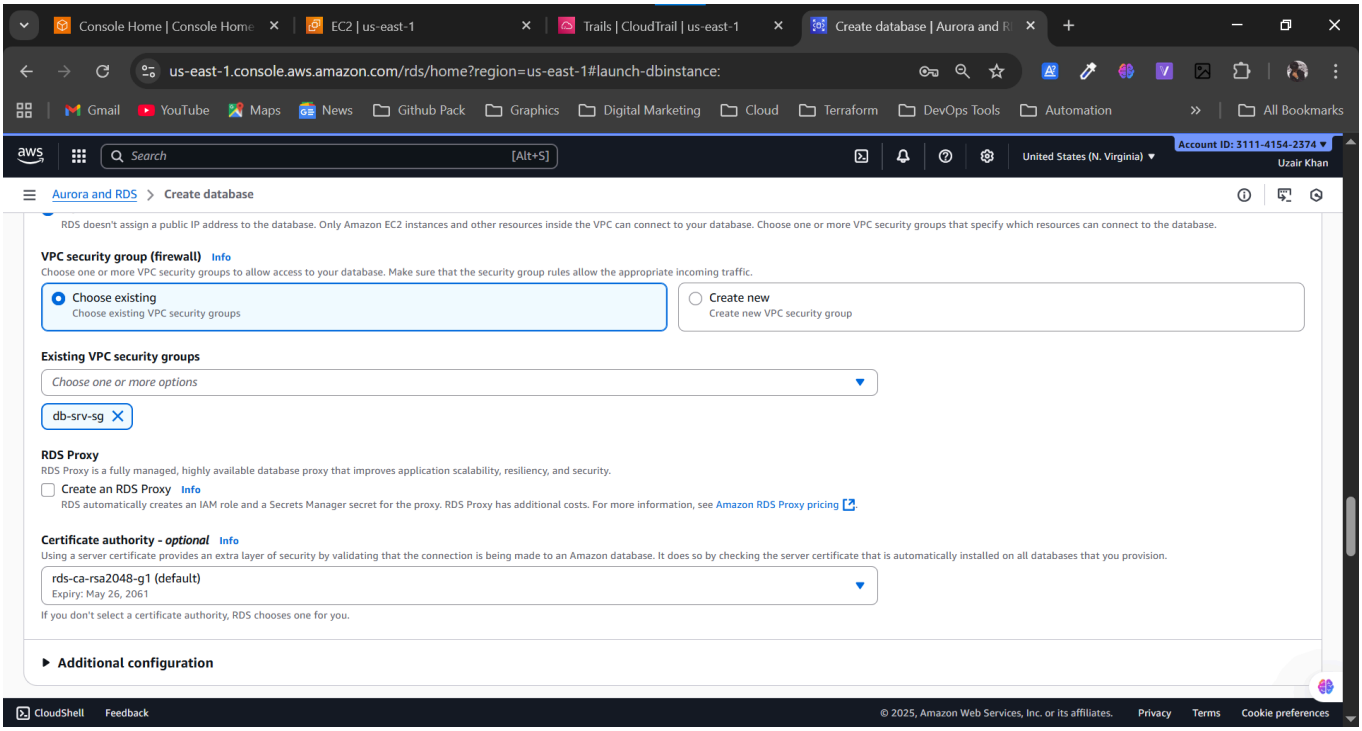
- In **Storage** choose:
  - **Storage type** = `General Purpose SSD (gp2)`
  - **Allocated storage** = `20`



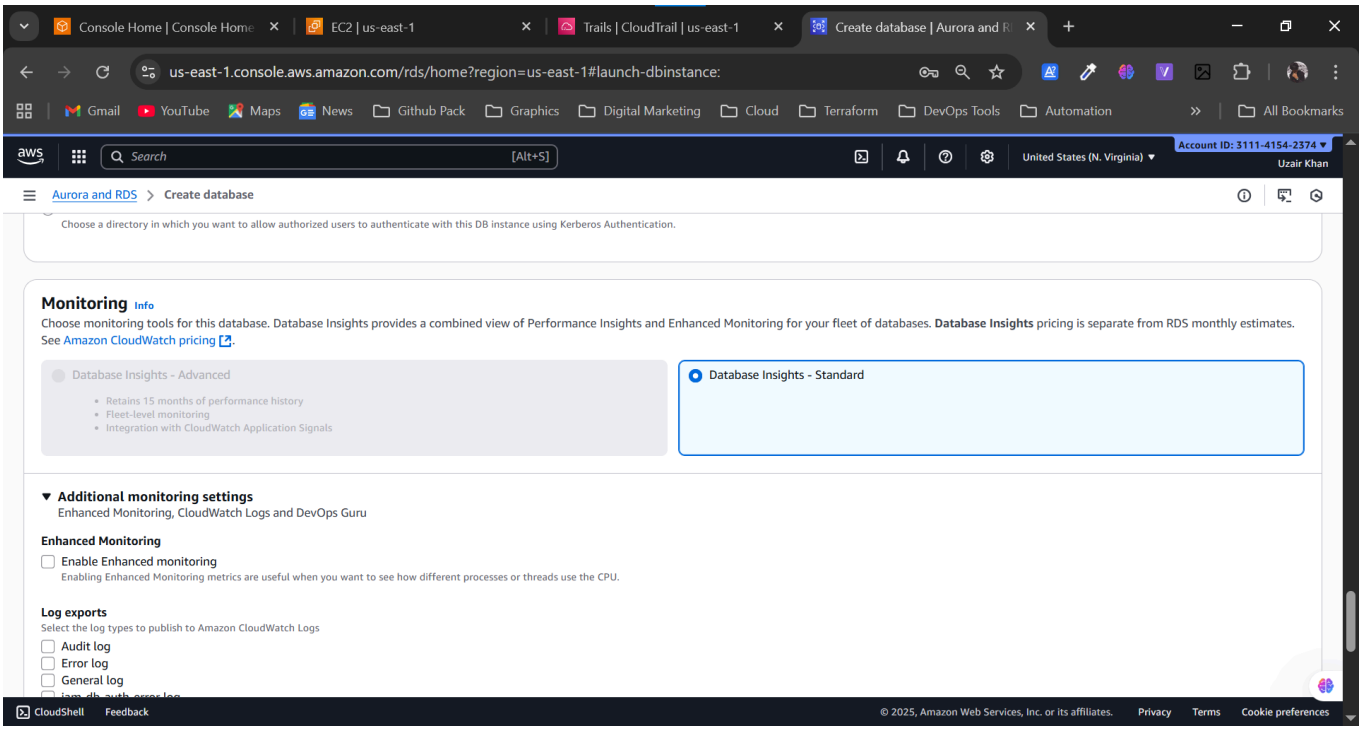
- In **Connectivity** choose:
  - **Virtual Private Cloud (VPC)** = `3-tier-vpc`
  - **DB subnet group** = `three-tier-rds-subnetgroup`
  - **Public access** = `No`



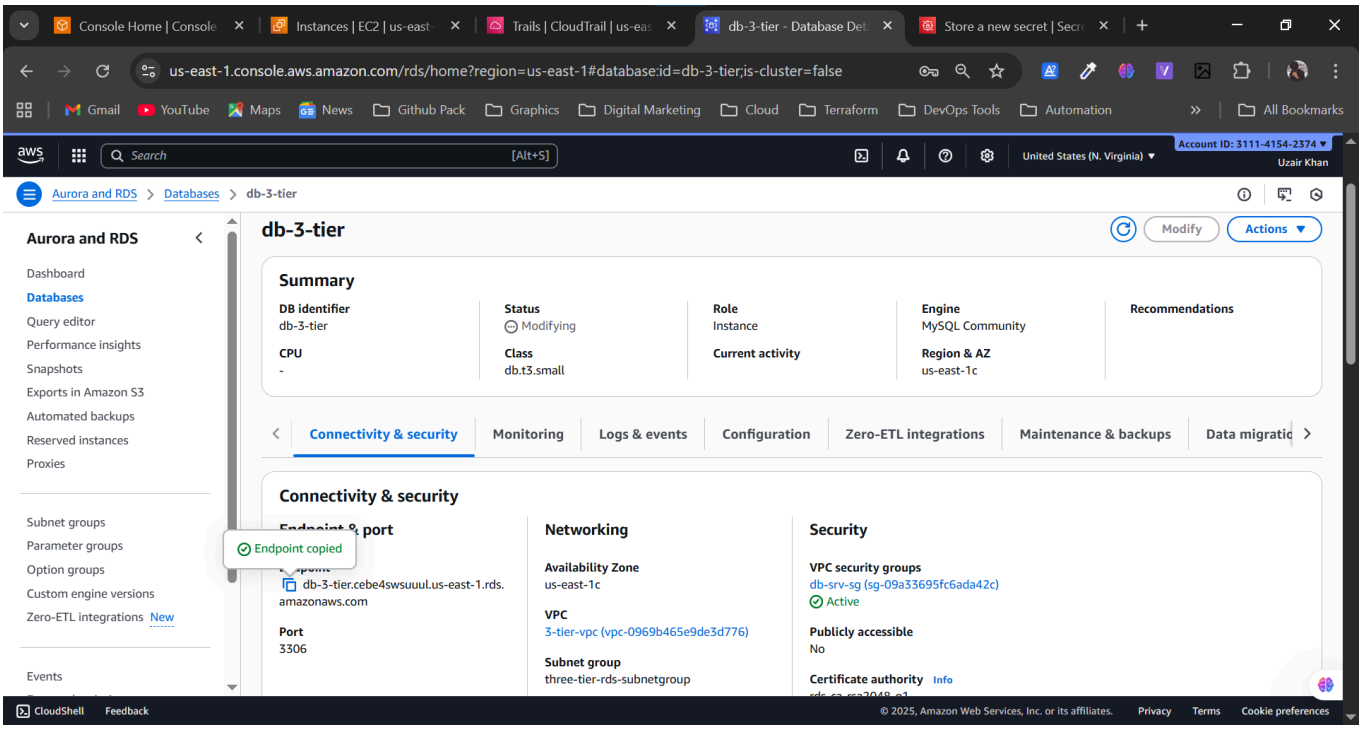
- Now choose **VPC security group (firewall) =** `db-srv-sg`



- In **Monitoring** uncheck the **Enable Enhanced monitoring**.



- Click **Create database**.
- Now update the `variables.tf` file with your RDS Endpoint which you can get from here:



- Now **Download/Copy** the `secrets.tf` file from here.
- Apply it by executing:

```
terraform validate
terraform plan
terraform apply -auto-approve
```

- Now SSH to `jump-server-app` .
- Connect to your database by executing:

```
mysql -h CHANGE-TO-YOUR-RDS-ENDPOINT -u admin -p
```

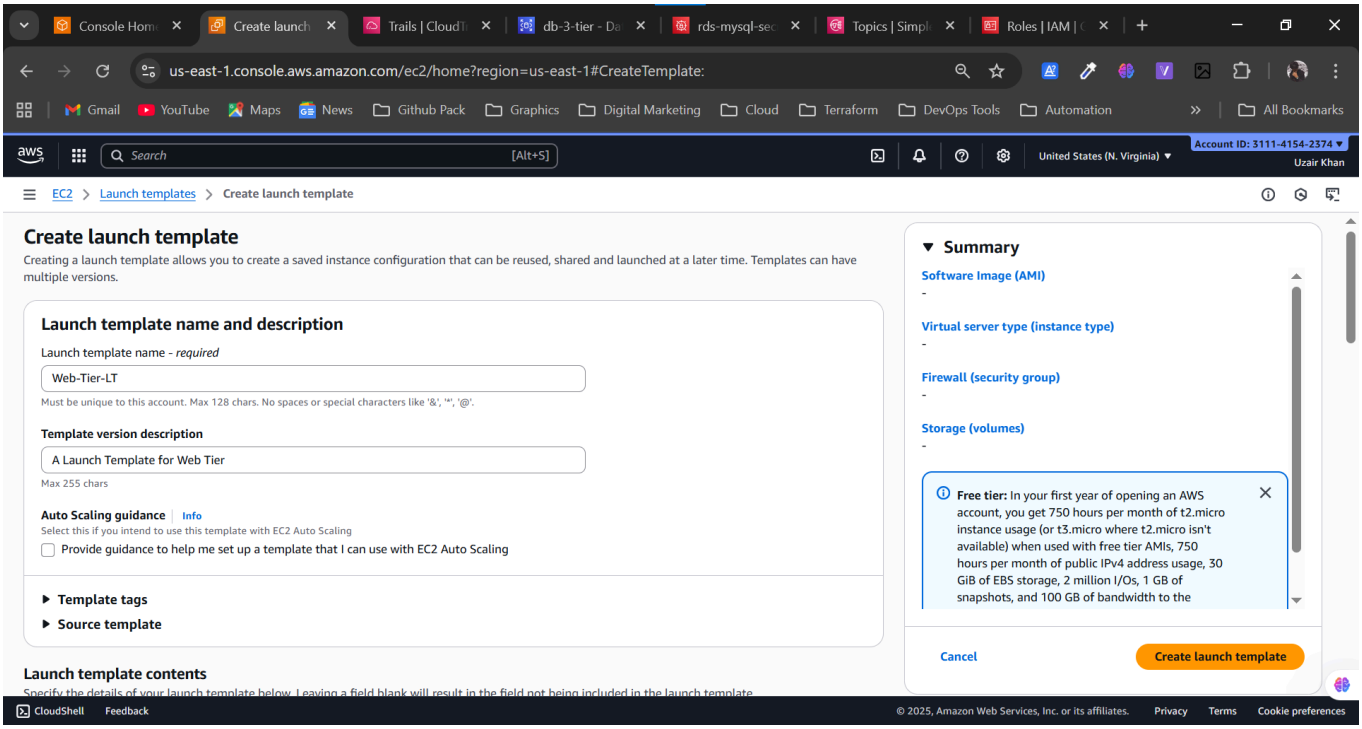
- Paste this SQL query to create a **Transactions** table in **webappdb** database:

```
CREATE DATABASE webappdb;
SHOW DATABASES;
USE webappdb;
CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL AUTO_INCREMENT, amount
DECIMAL(10,2), description VARCHAR(100), PRIMARY KEY(id));
SHOW TABLES;
INSERT INTO transactions (amount,description) VALUES ('400','groceries');
SELECT * FROM transactions;
```

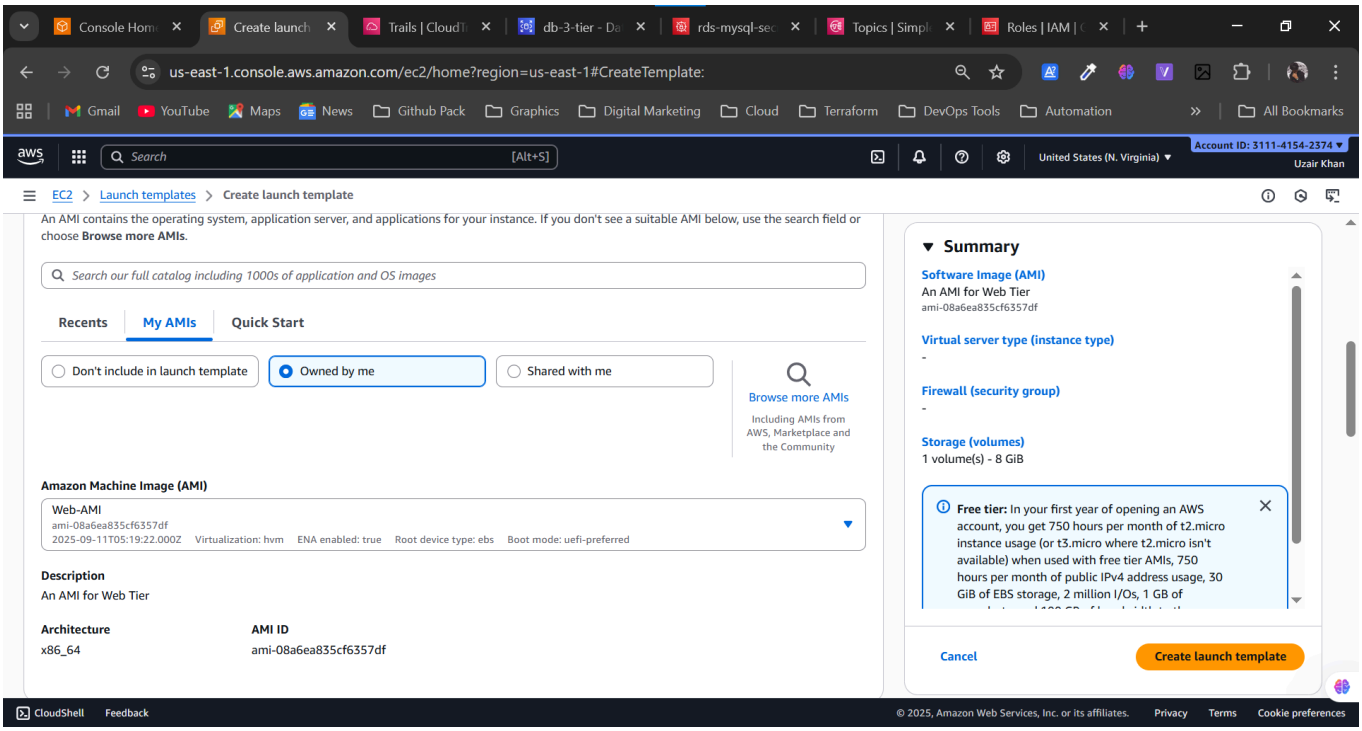
7. Creating Launch Templates

Now you'll have to create the **Launch Templates** from the **AMIs** you have created before.

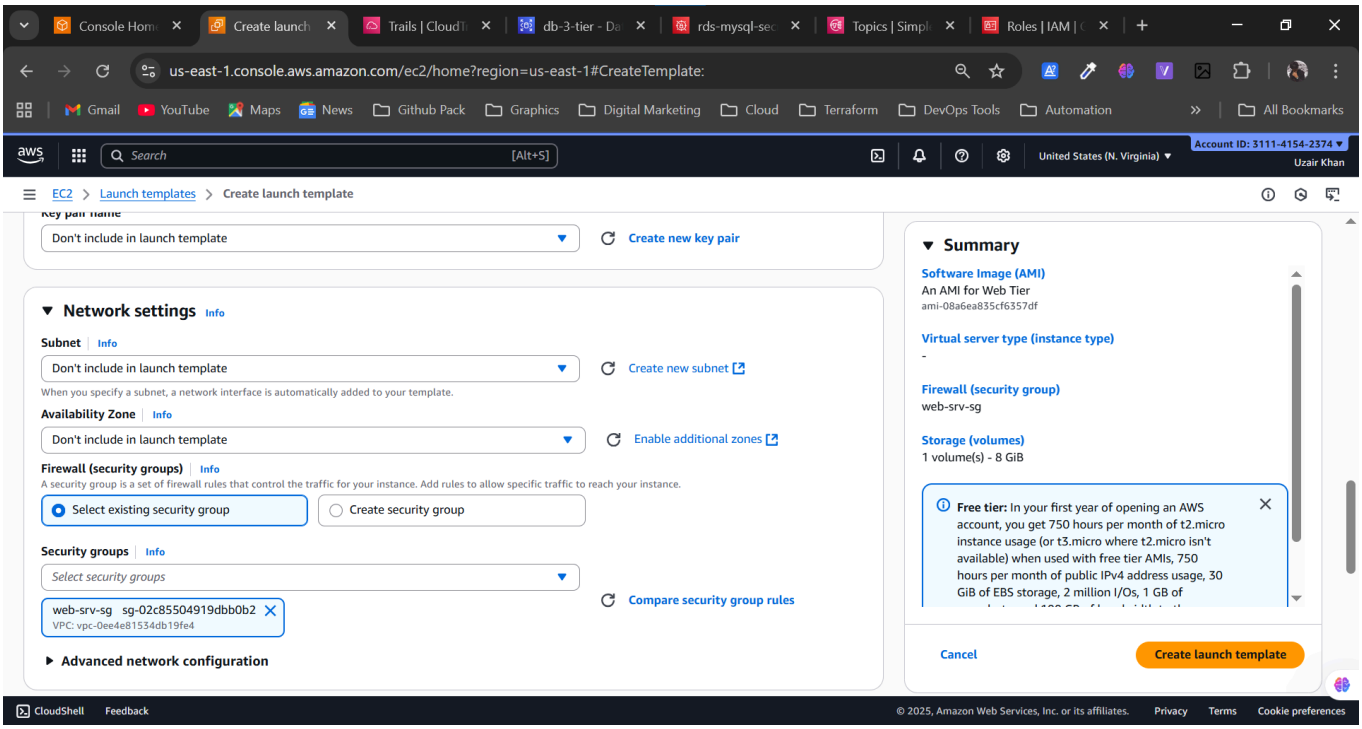
- Go to **EC2 --> Launch Templates**.
- Click on **Create launch template**.
- Enter the following:
  - **Launch template name =** `Web-Tier-LT`
  - **Template version description =** `A Launch Template for Web Tier`



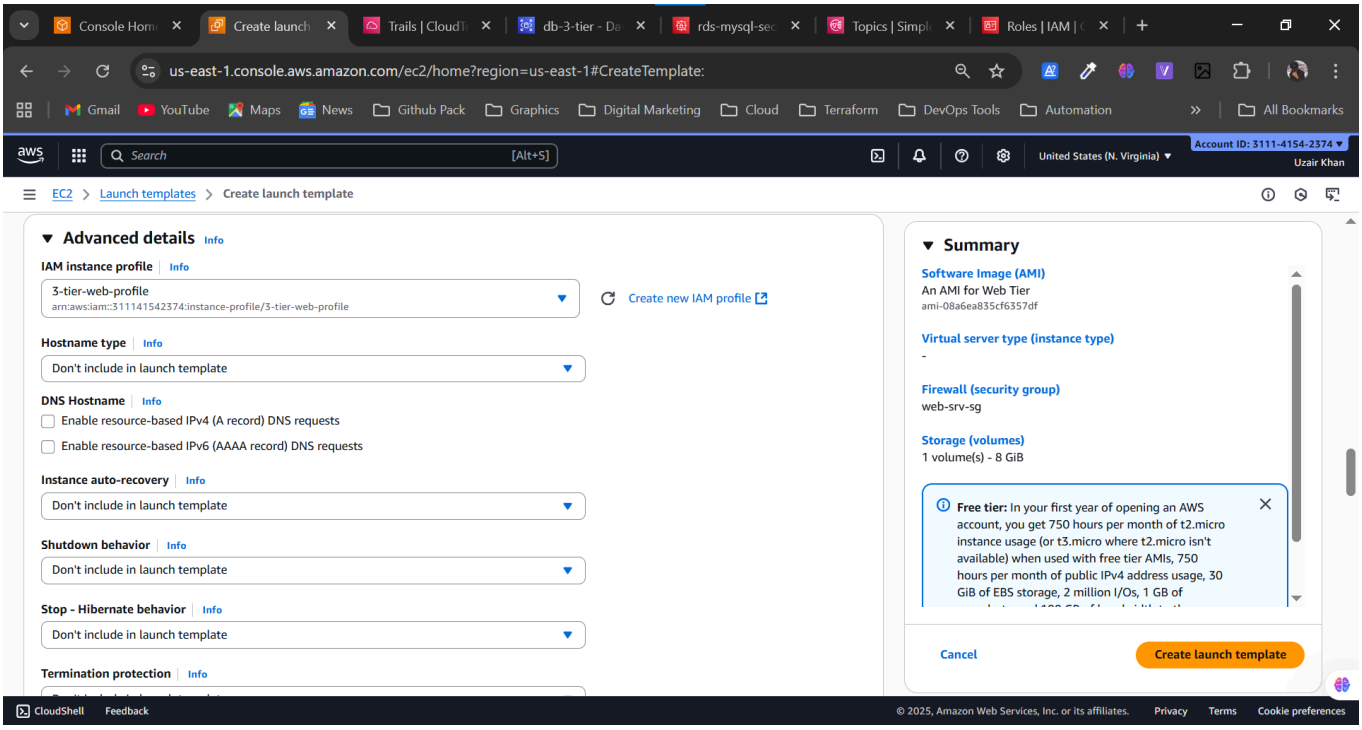
- In the **AMI** section, choose `Web-AMI`.



- In **Security groups** choose `web-srv-sg`.



- In the **IAM instance profile** choose `3-tier-web-profile`.



- In the **User data** write:

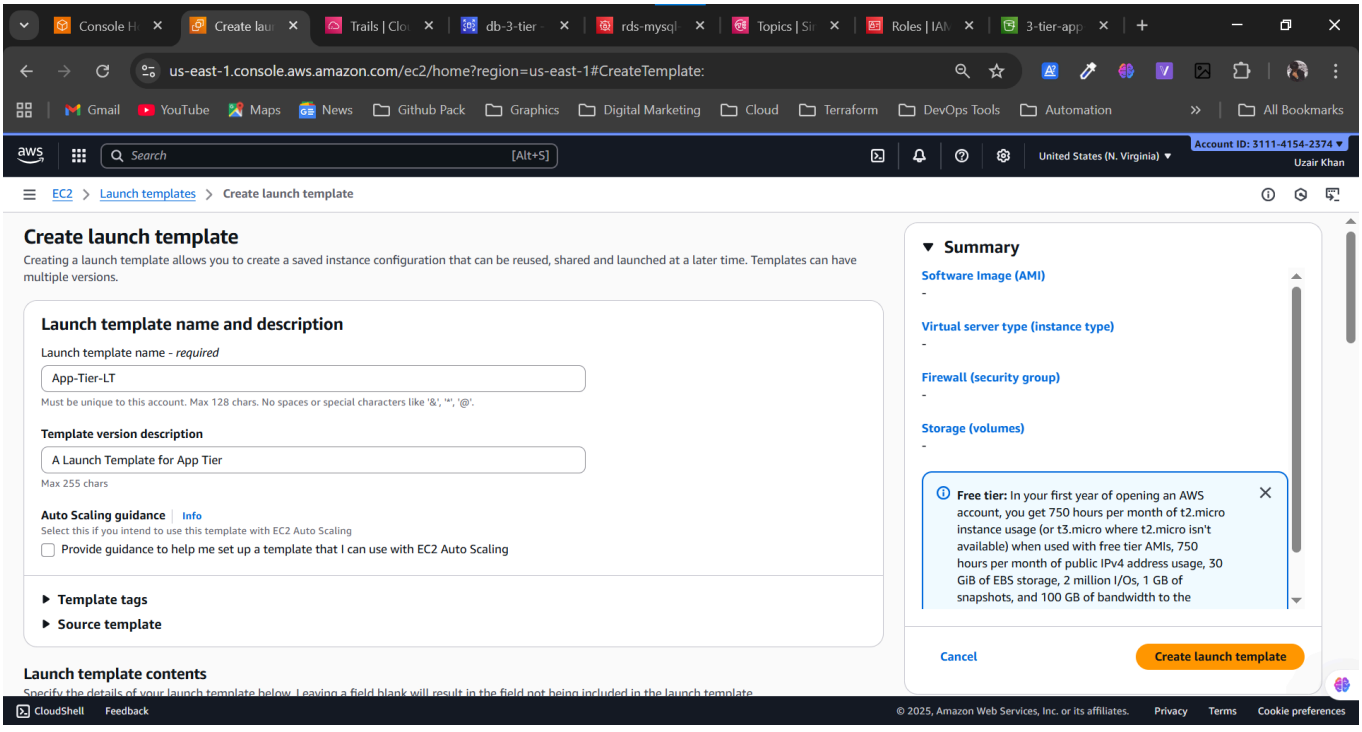
```
#!/bin/bash
# Log everything to /var/log/user-data.log
exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console) 2>&1

# Install AWS CLI v2 (if not already)
yum install -y awscli
```

```
7
8   # Download application code from S3
9   aws s3 cp s3://<YOUR-S3-BUCKET-NAME>/application-code /home/ec2-user/application-
    code --recursive
10
11  # Go to app directory
12  cd /home/ec2-user/application-code
13
14  # Make script executable and run it
15  chmod +x web.sh
16  sudo ./web.sh
```

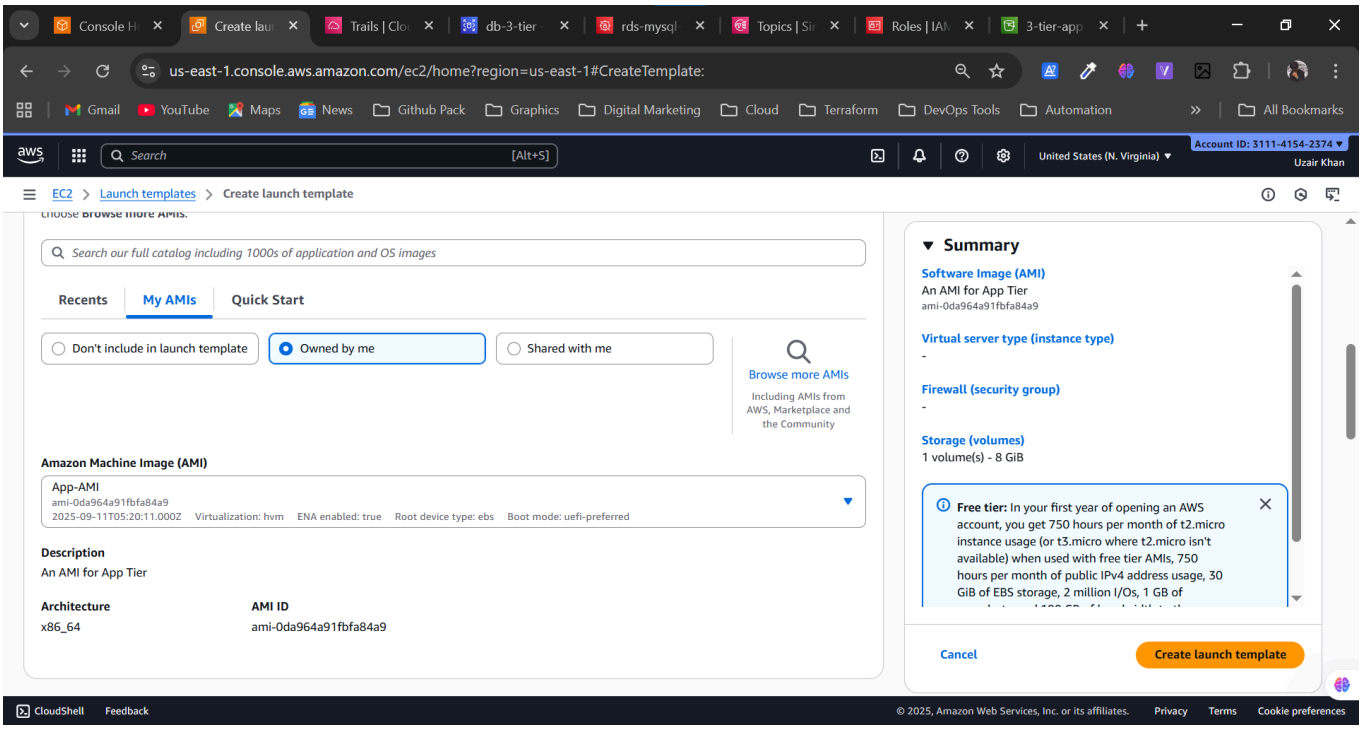
Important

Update the [YOUR-S3-BUCKET-NAME] with your actual S3 Bucket name for the Application Code.

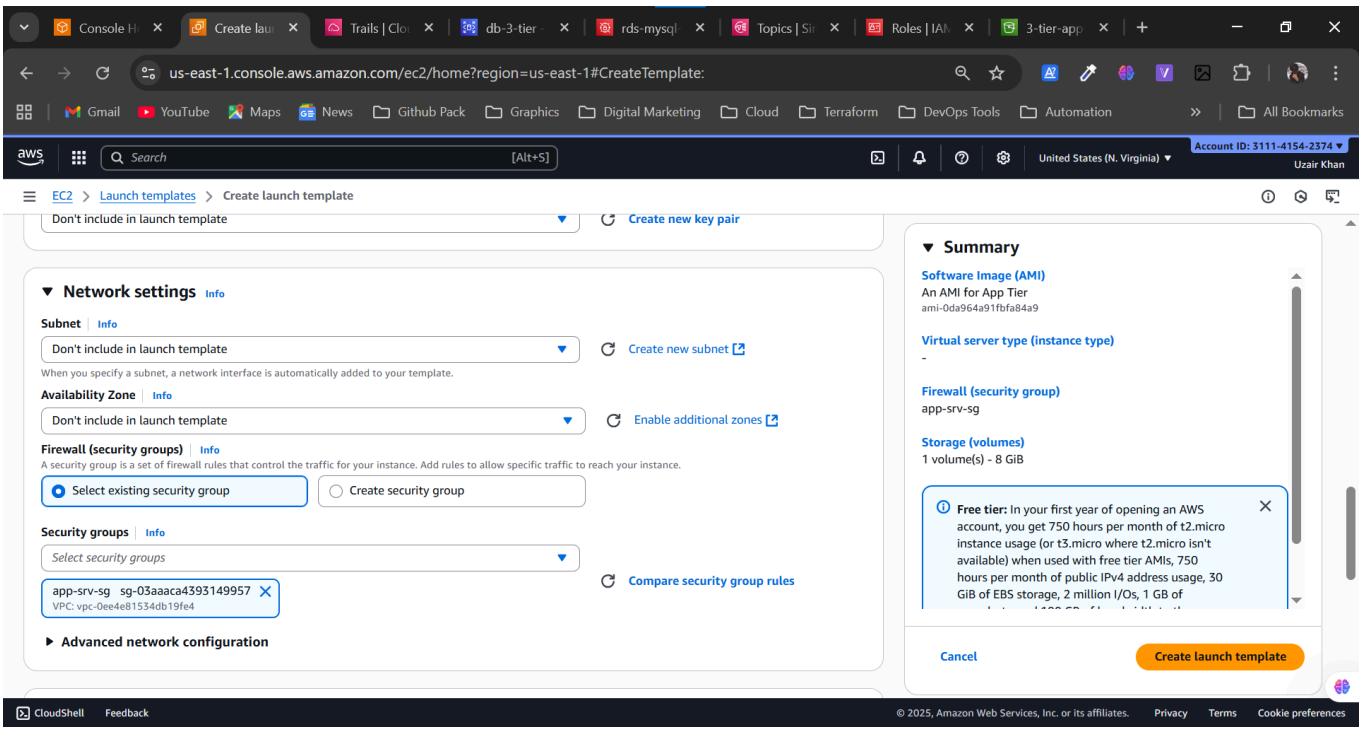
- Click **Create launch Template**.
- Now create another **Launch Template** by clicking again on **Create launch template**.
- Enter the following:
  - **Launch template name** = `App-Tier-LT`
  - **Template version description** = `A Launch Template for App Tier`



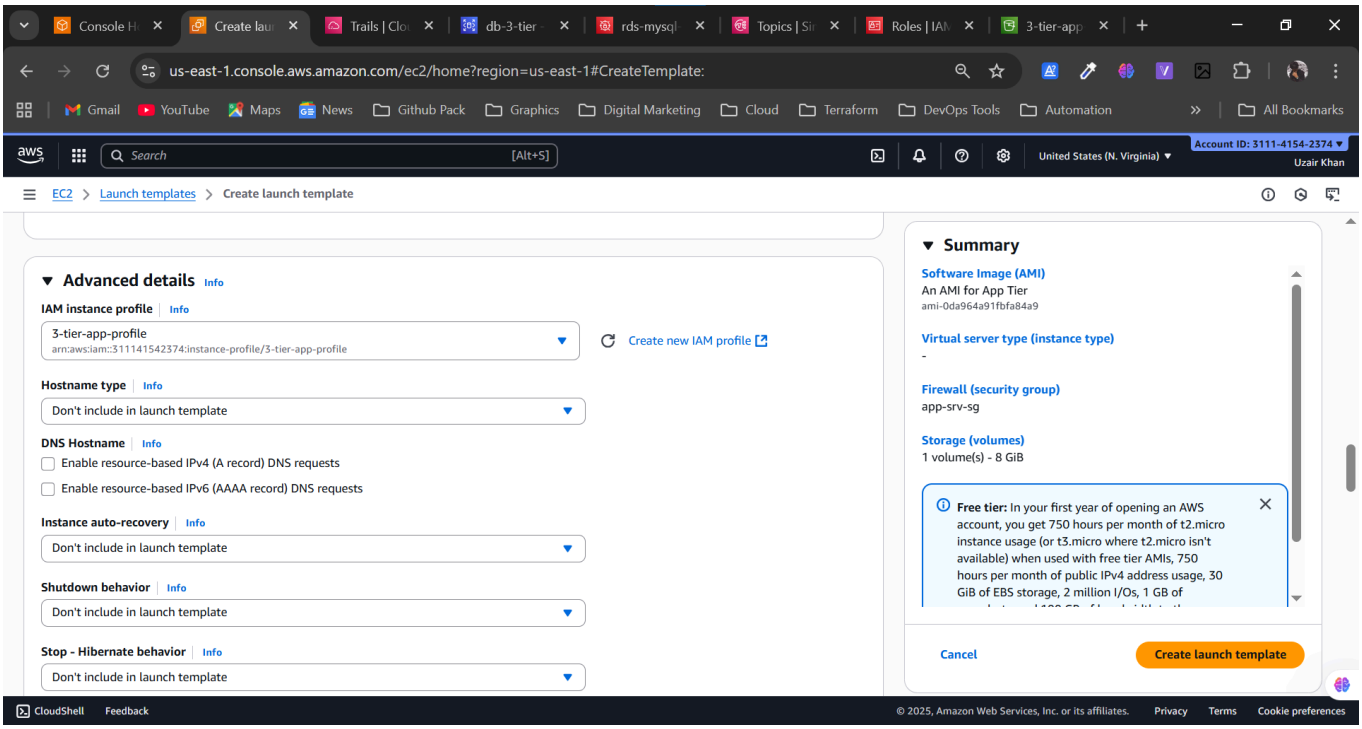
- In the **AMI** section, choose `App-AMI` .



- In **Security groups** choose `app-srv-sg` .



- In the **IAM instance profile** choose `3-tier-app-profile` .



- In the **User data** write:

```
#!/bin/bash
# Log everything to /var/log/user-data.log
exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console) 2>&1

# Install AWS CLI v2 (if not already)
yum install -y awscli

# Download application code from S3
aws s3 cp s3://<YOUR-S3-BUCKET-NAME>/application-code /home/ec2-user/application-code --recursive

# Go to app directory
cd /home/ec2-user/application-code

# Make script executable and run it
chmod +x app.sh
sudo ./app.sh
```

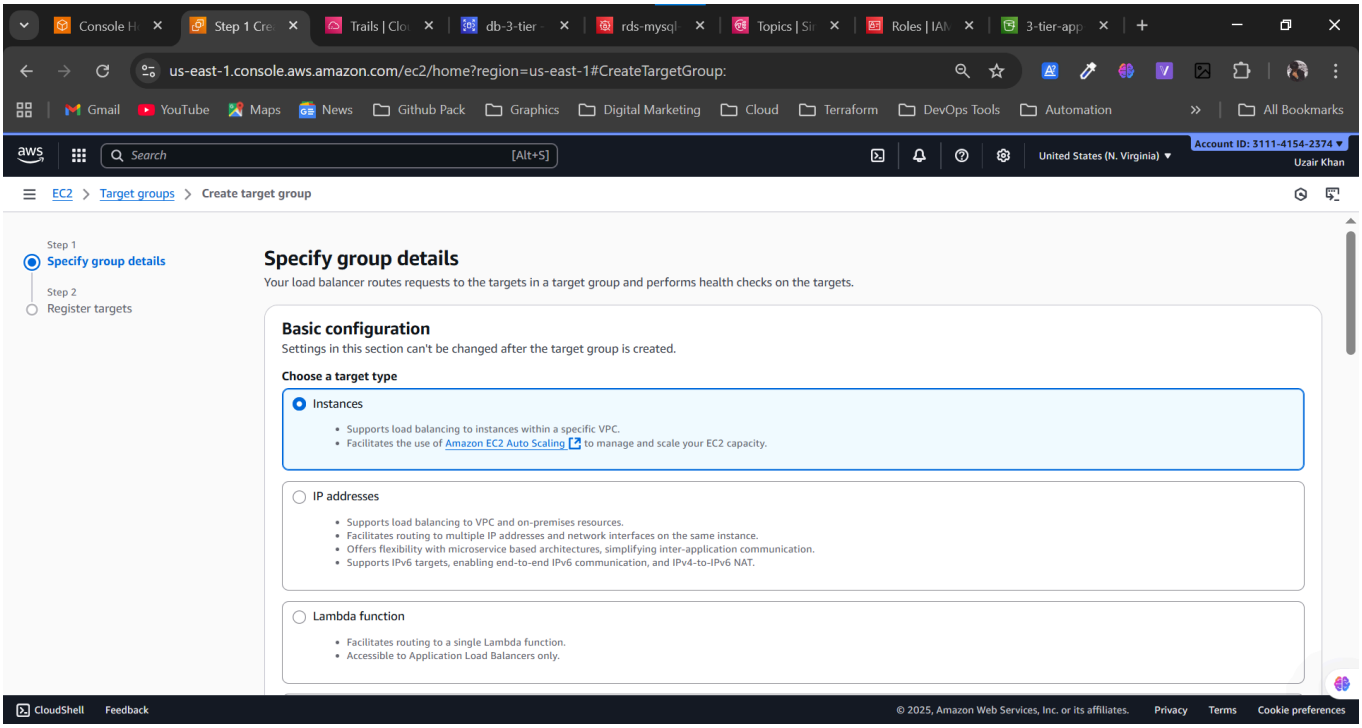
> **Important**
>
> Update the [YOUR-S3-BUCKET-NAME] with your actual S3 Bucket name for the Application Code.

- Click **Create launch template**.

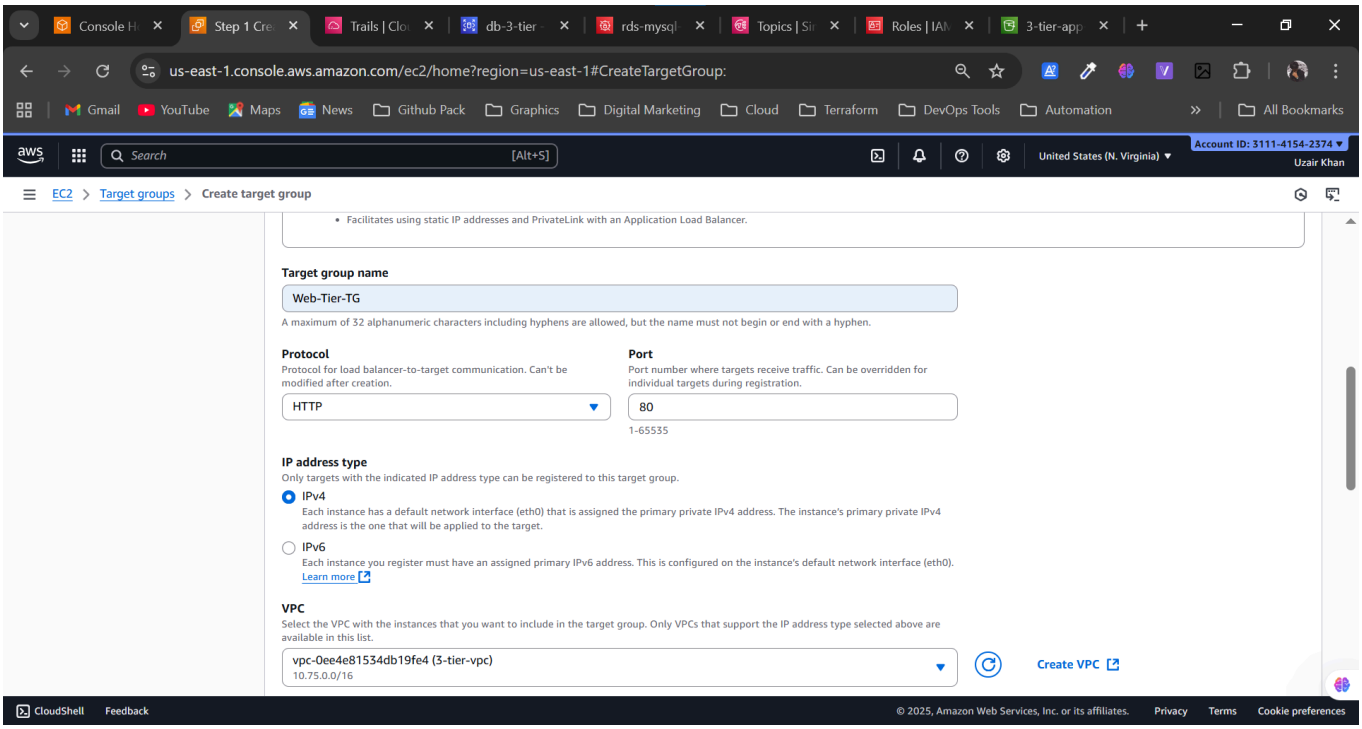
8. Creating Target Groups

In this step, you'll have to create **Target Groups** for both **Web Tier** and **App Tier** so that it can be later used by the **Load Balancer** as a listener.

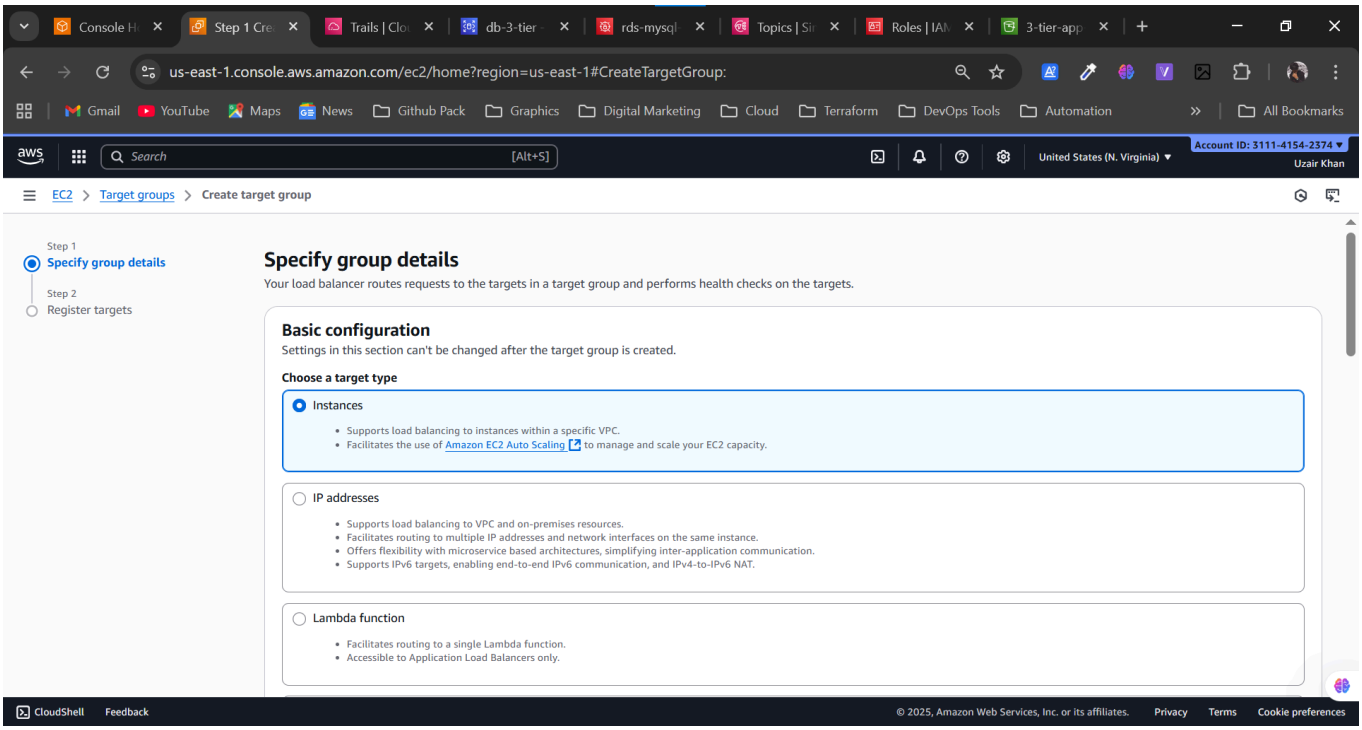
- Go to **EC2 --> Target groups**.
- Click on **Create target group**.
- Choose a **Target type** as `Instances`.



- Enter the following:
  - **Target group name** = `Web-Tier-TG`
  - **Protocol** = `HTTP`
  - **Port** = `80`
  - **VPC** = `3-tier-vpc`

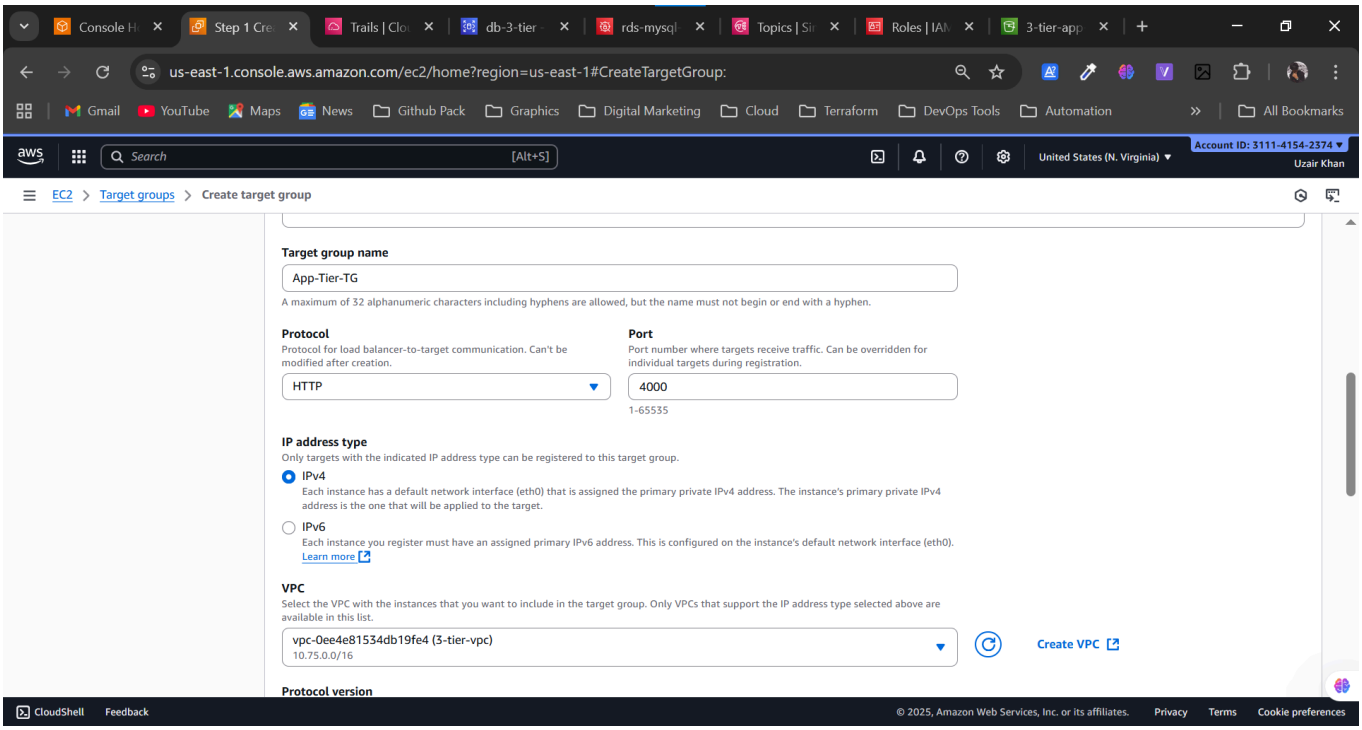


- Click **Next**.
- Click **Create target group**.
- Now create the **App Tier Target Group**.
- Choose a **Target type** as `Instances`.

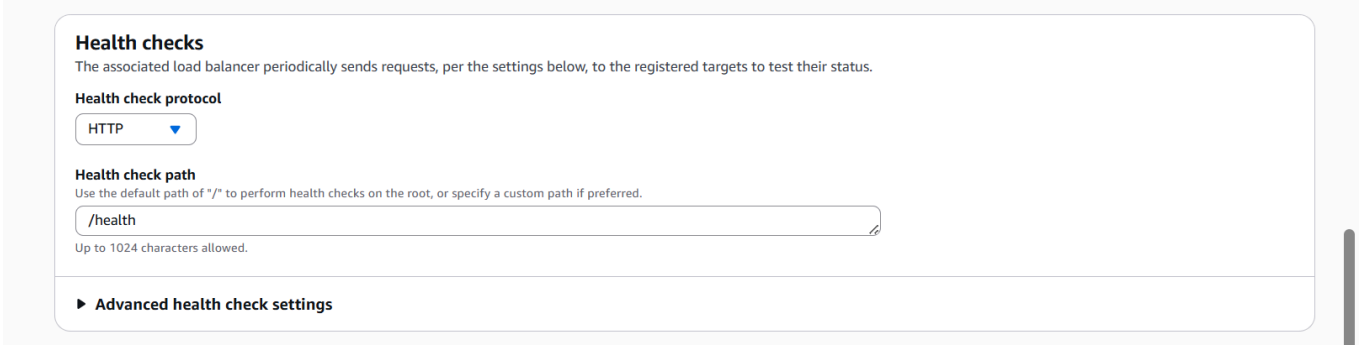


- Enter the following:
  - **Target group name** = `App-Tier-TG`
  - **Protocol** = `HTTP`
  - **Port** = `4000`

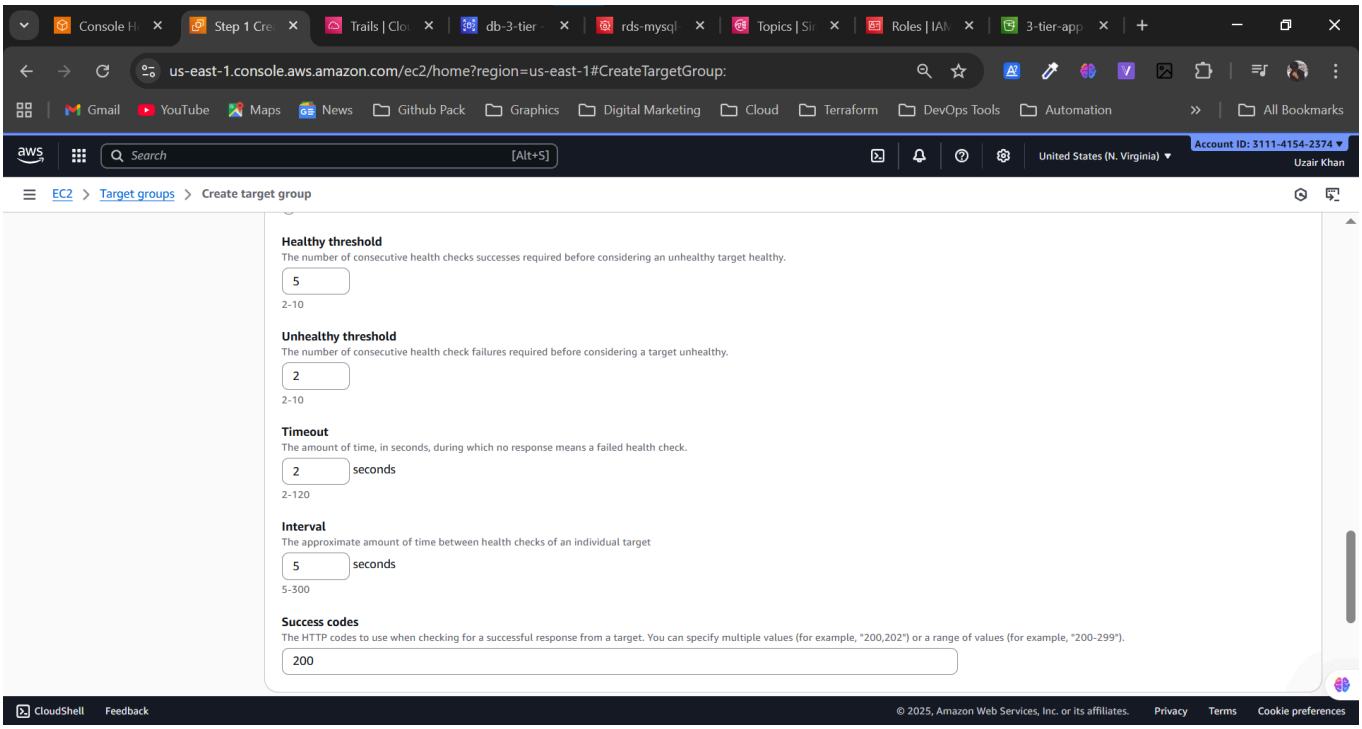
- **VPC =** `3-tier-vpc`



- In **Health checks** choose:
  - **Health check path =** `/health`



- In **Advanced health check settings** choose:
  - **Timeout =** `2`
  - **Interval =** `5`

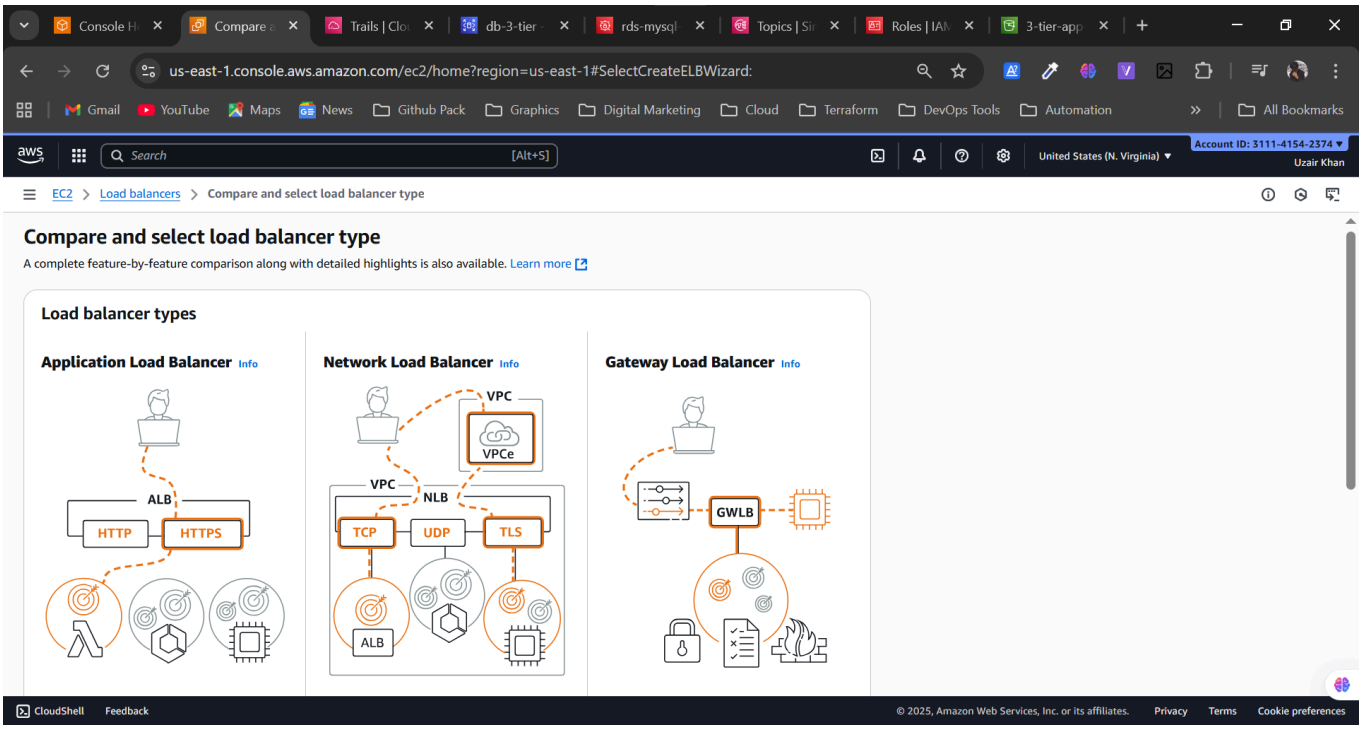


- Click **Next**.
- Click **Create target group**.

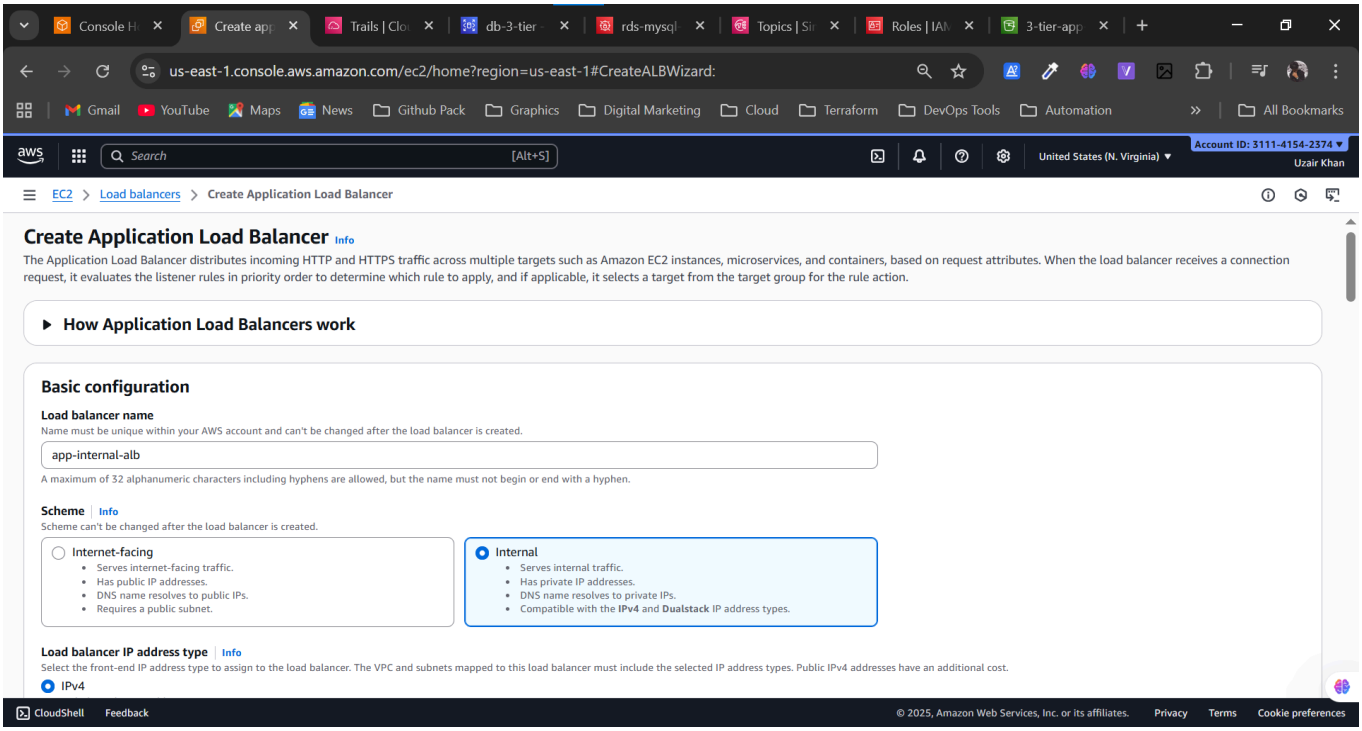
9. Creating Load Balancers

In this step, You'll be creating **Load Balancers** which will be load-balancing the traffic between multiple EC2 instances in your **Auto-Scaling Group**.

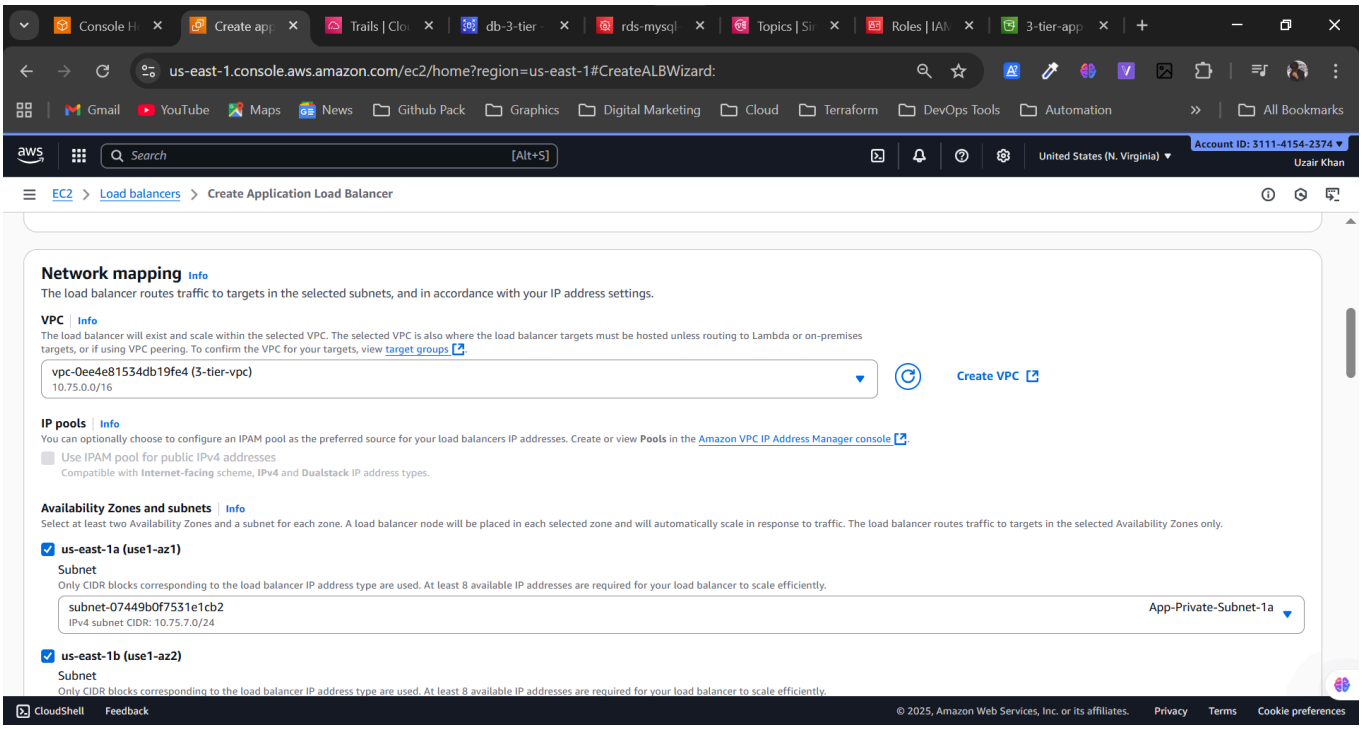
- Go to **EC2** --> **Load Balancers**.
- Click on **Create load balancer**.
- In **Load balancer types** select `Application Load Balancer` .



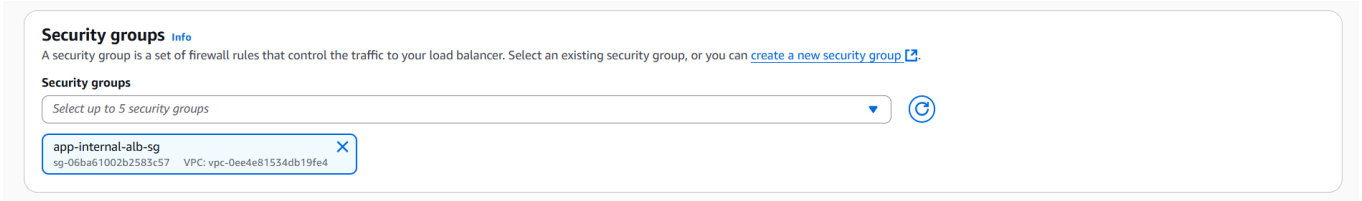
- Now choose:
  - **Load balancer name** = `app-internal-alb`
  - **Scheme** = `Internal`



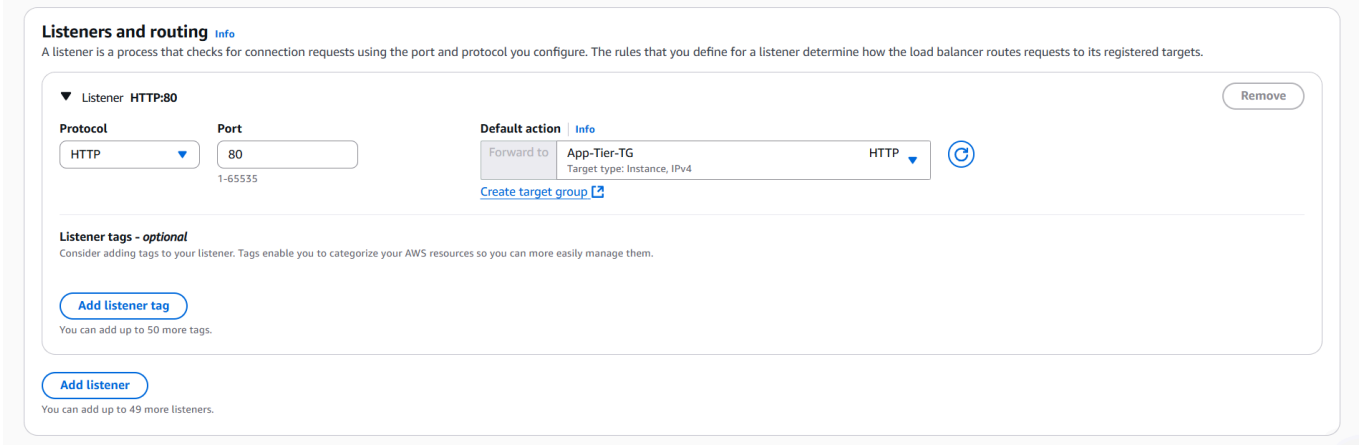
- In **Network mapping** choose:
  - **VPC** = `3-tier-vpc`
  - **Availability Zones and subnets** = `App-Private-Subnet-1a`, `App-Private-Subnet-1b`, `App-Private-Subnet-1c`



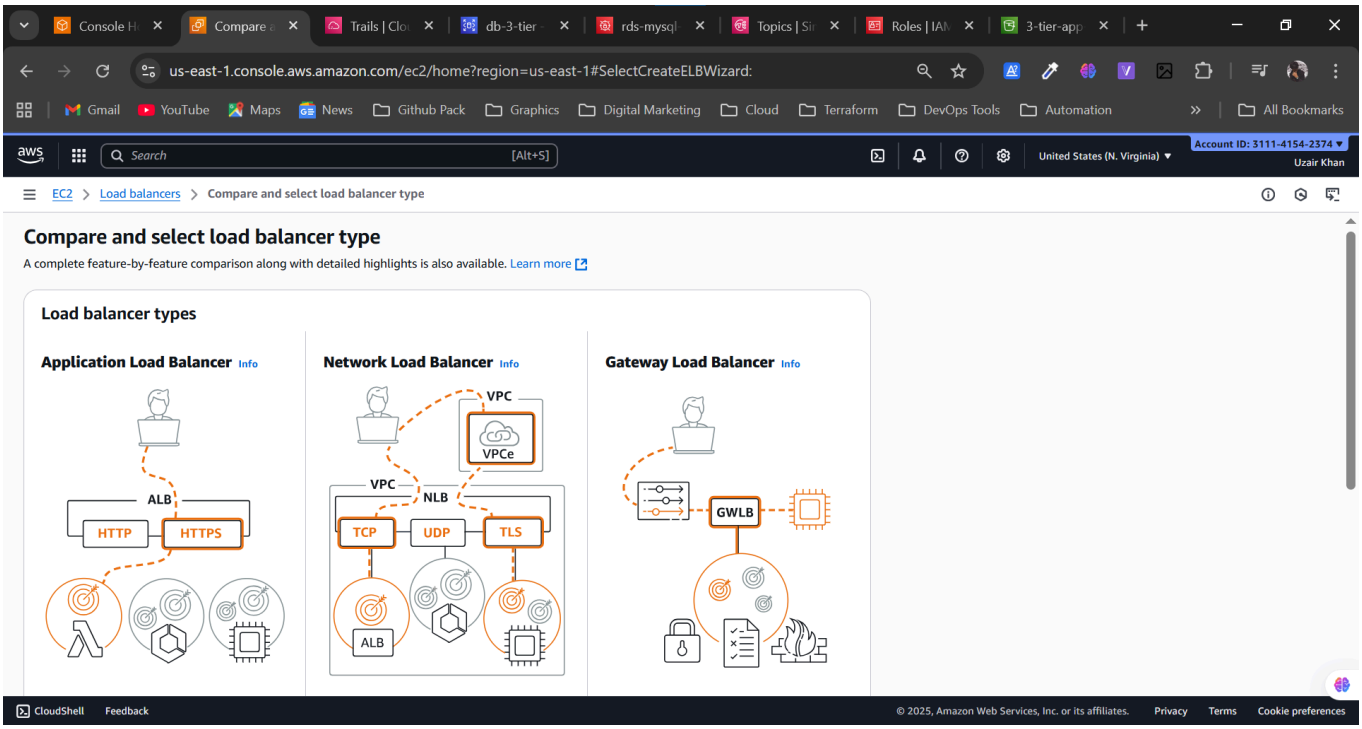
- In **Security groups** choose `app-internal-alb-sg` .



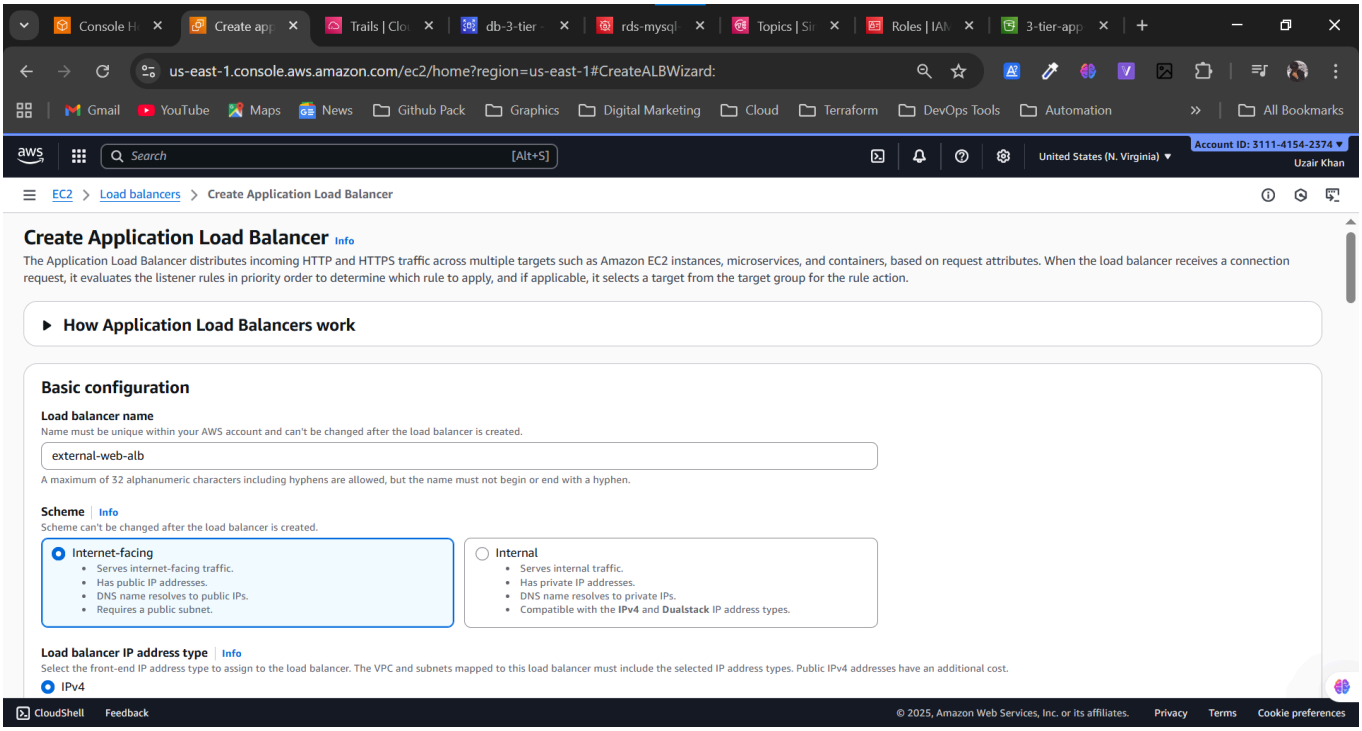
- In **Listeners and routing** choose:
  - **Protocol** = `HTTP`
  - **Port** = `80`
  - **Default action** = `App-Tier-TG`



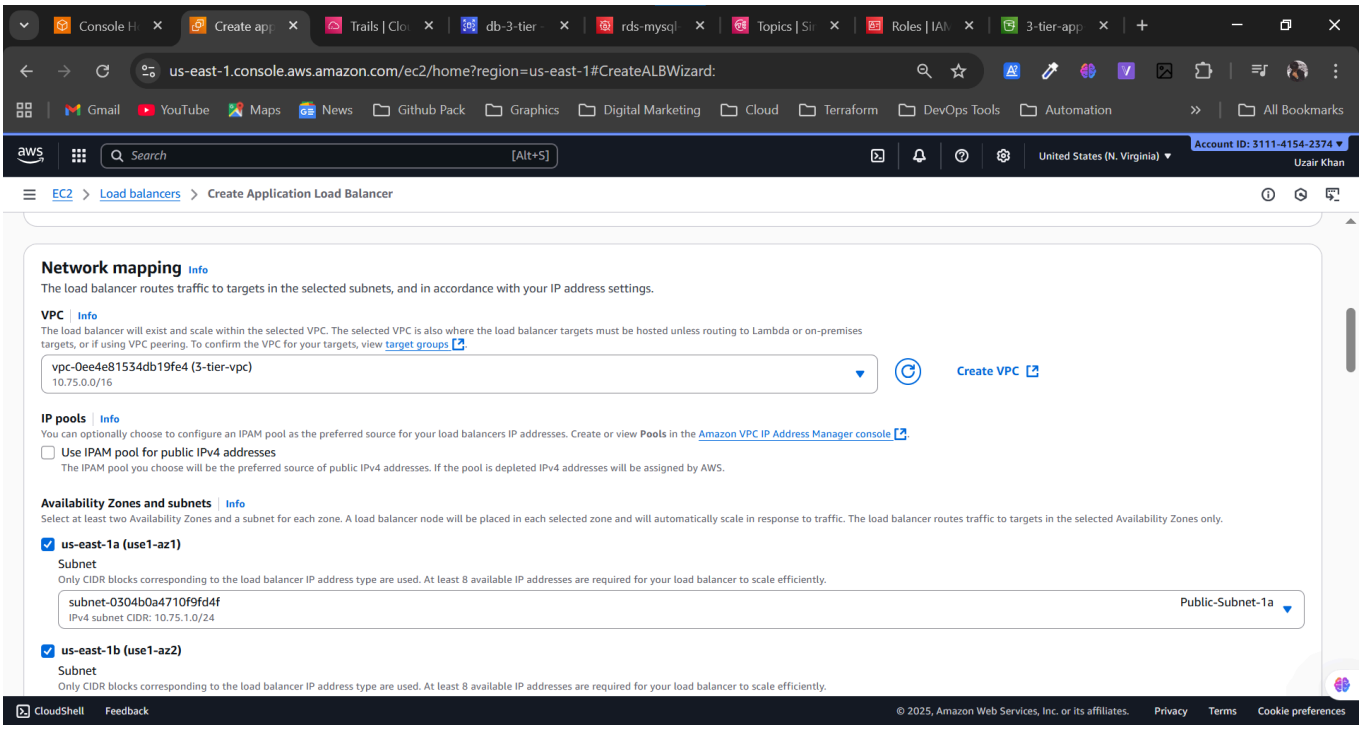
- Click on **Create load balancer**.
- Now create **Web External LB**.
- In **Load balancer types** select `Application Load Balancer` .



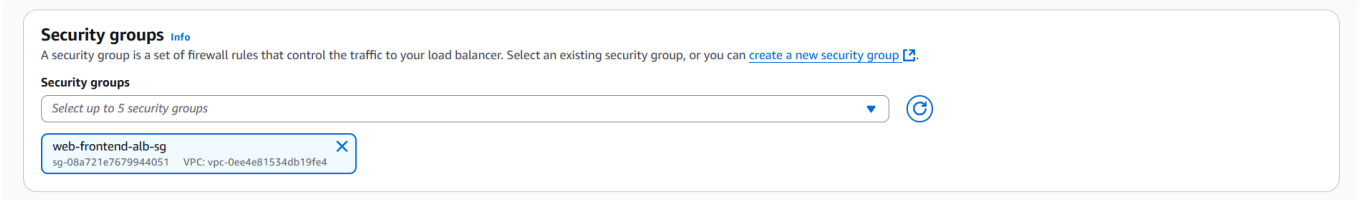
- Now choose:
  - **Load balancer name** = `external-web-alb`
  - **Scheme** = `Internet-facing`



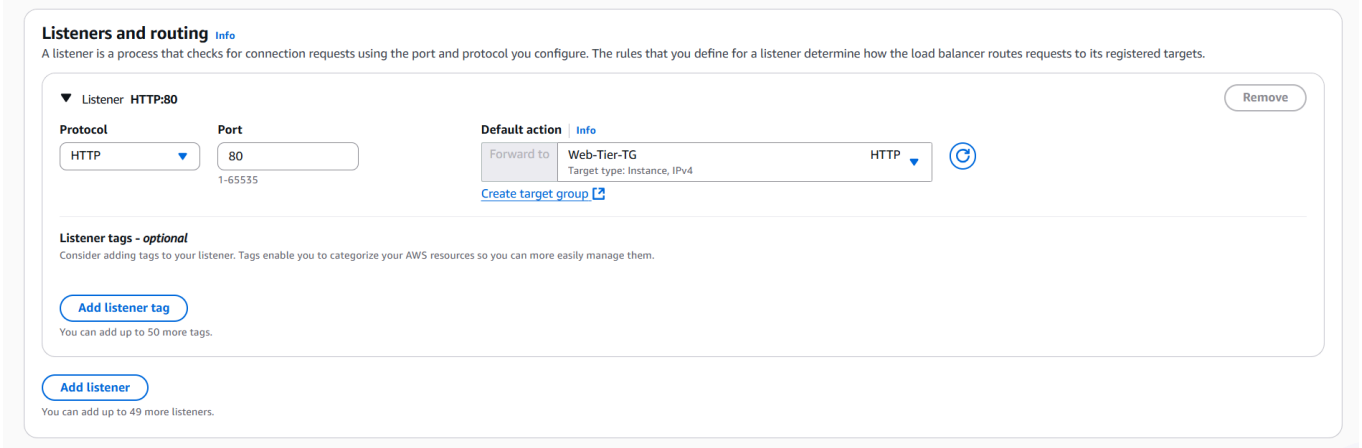
- In **Network mapping** choose:
  - **VPC** = `3-tier-vpc`
  - **Availability Zones and subnets** = `Public-Subnet-1a` , `Public-Subnet-1b` , `Public-Subnet-1c`



- In **Security groups** choose `web-frontend-alb-sg` .



- In **Listeners and routing** choose:
  - **Protocol** = `HTTP`
  - **Port** = `80`
  - **Default action** = `Web-Tier-TG`

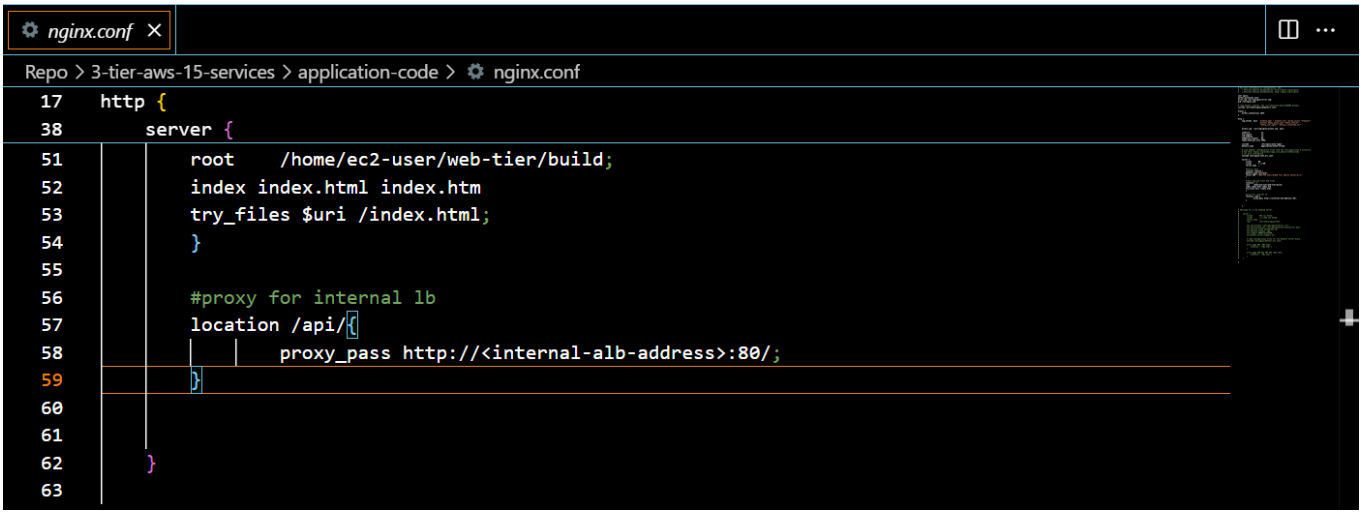


- Click on **Create load balancer**.

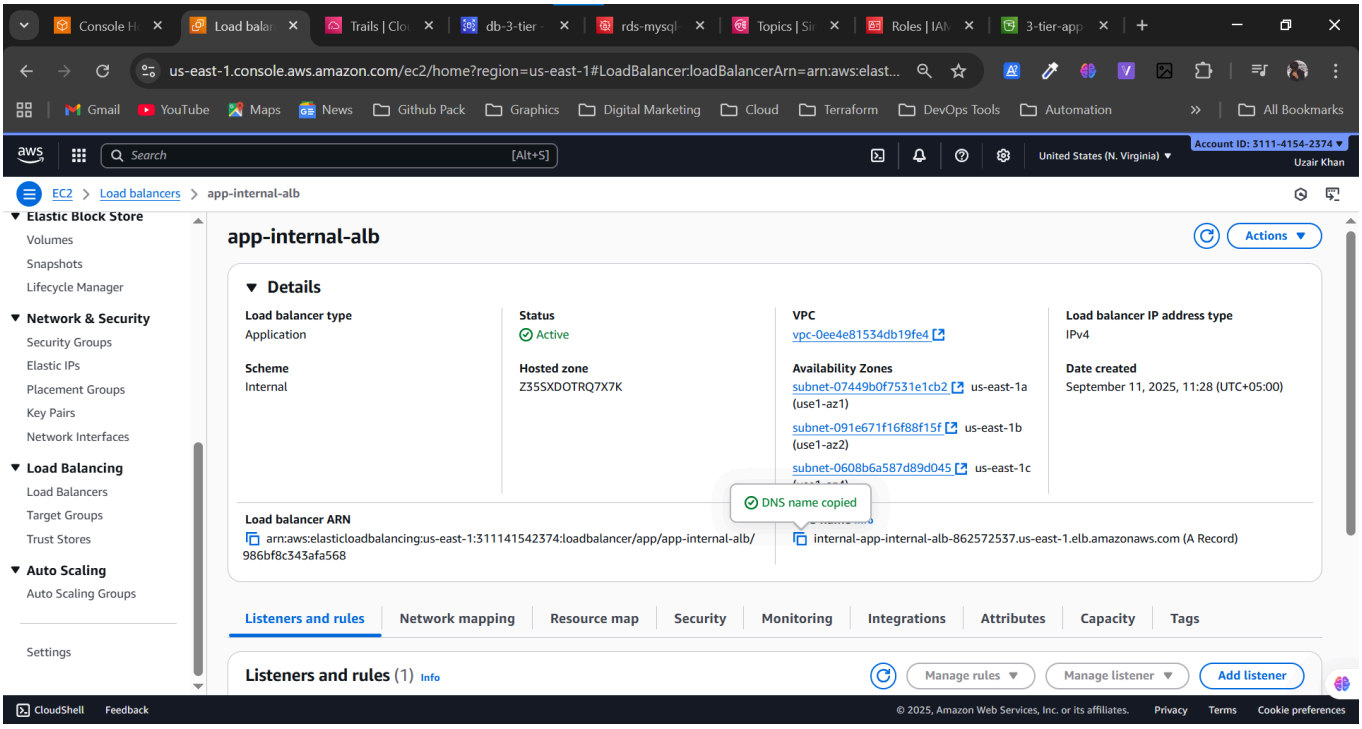
10. Updaing Application Files

Now you'll have to update the **Application Files** in your cloned repo to get it to work correctly acording to you. Then you'll upload it to your **Application Code** S3 Bucket.

- Open **application-code --> nginx.conf** in VS Code.



- In **application-code/nginx.conf** update the `Internal-ALB-Address` to the one you just created.



- Open **application-code --> app.sh** in VS Code.

nginx.conf M | $ app.sh

Repo > 3-tier-aws-15-services > application-code > $ app.sh

```
#!/bin/bash
set -e   # exit on error

# Download app-tier code
cd /home/ec2-user
aws s3 cp s3://<your-bucket-name>/application-code/app-tier app-tier --recursive

# Ensure correct ownership/permissions
sudo chown -R ec2-user:ec2-user /home/ec2-user
sudo chmod -R 755 /home/ec2-user/app-tier

# Run as ec2-user so nvm/npm/pm2 are available
su - ec2-user <<'EOF'
# Load nvm environment
export NVM_DIR="$HOME/.nvm"
```

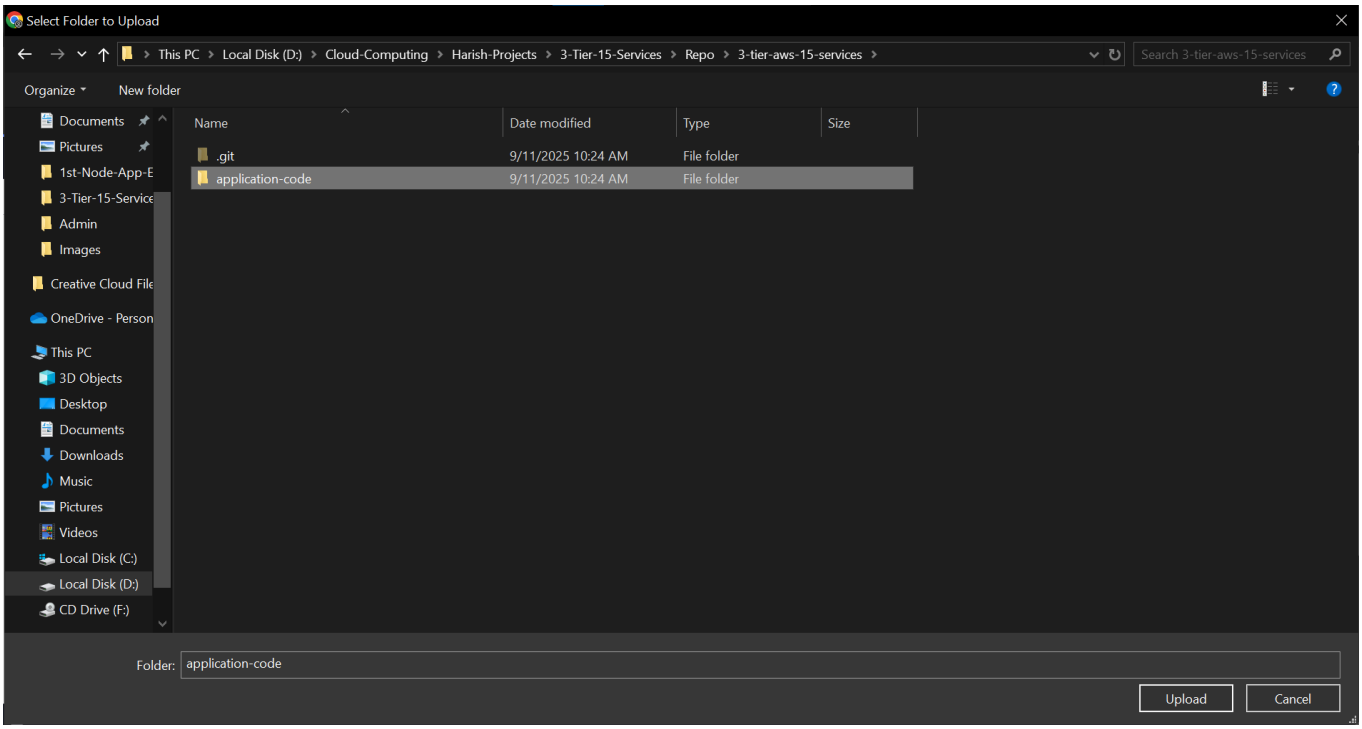
- Update this **Bucket Name** with your `Application Code Bucket` name.
- Now open **application-code** --> **app-tier** --> **DbConfig.js** in VS Code.

nginx.conf M | $ app.sh M | JS DbConfig.js

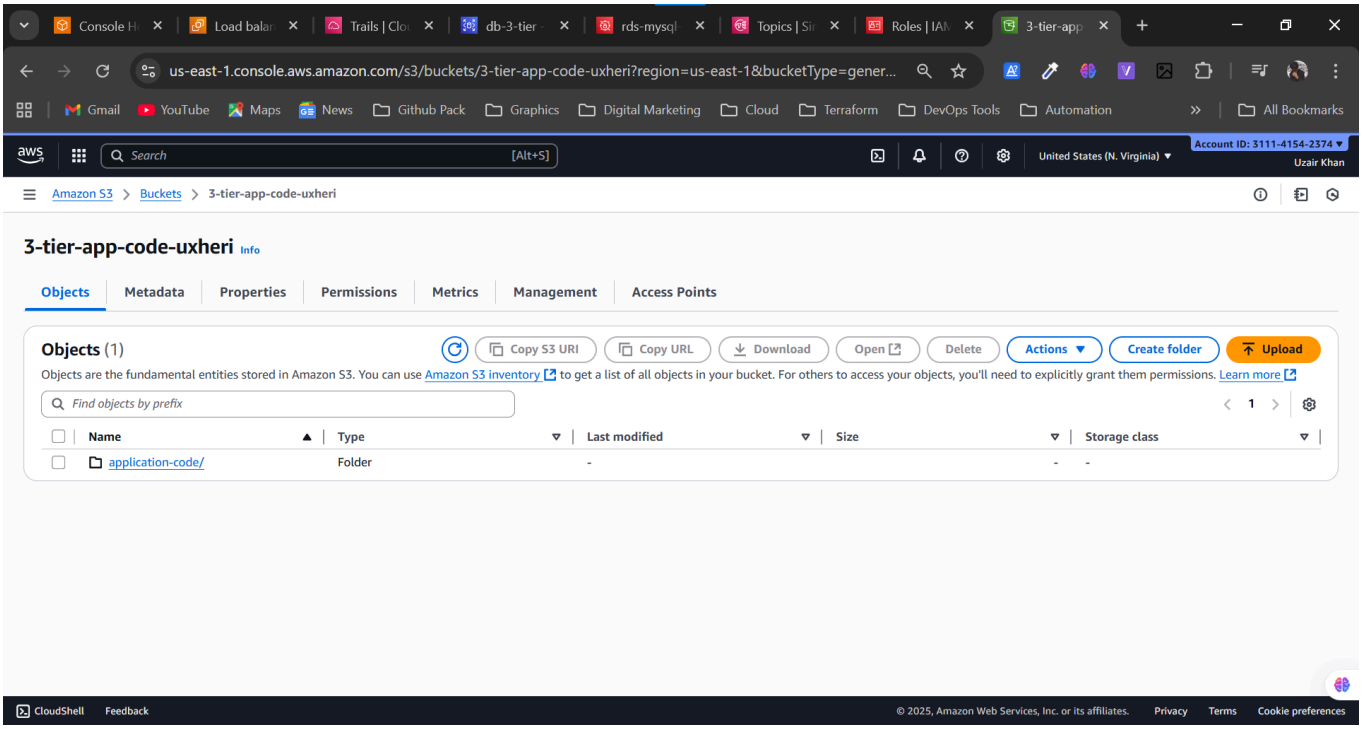
Repo > 3-tier-aws-15-services > application-code > app-tier > JS DbConfig.js > getDatabaseSecrets > secretName

```
const AWS = require('aws-sdk');
const secretsManager = new AWS.SecretsManager({
  region: process.env.AWS_REGION || 'us-east-1'
});

async function getDatabaseSecrets() {
  try {
    const secretName = process.env.DB_SECRET_NAME || "rds-mysql-secret-3";
    const data = await secretsManager.getSecretValue({ SecretId: secretName }).promise();

    if ('SecretString' in data) {
      return JSON.parse(data.SecretString);
    } else {
      throw new Error('Secret binary not supported');
    }
```

- Update the following:
  - **AWS Region** = `Your AWS Region`
  - **Secret Name** = `Your RDS Secrets Name`
- Now go to your **Application Code** S3 Bucket.
- Click on **Upload folder**.
- Select the **application-code** folder and upload it.



Wait for the upload to get complete.

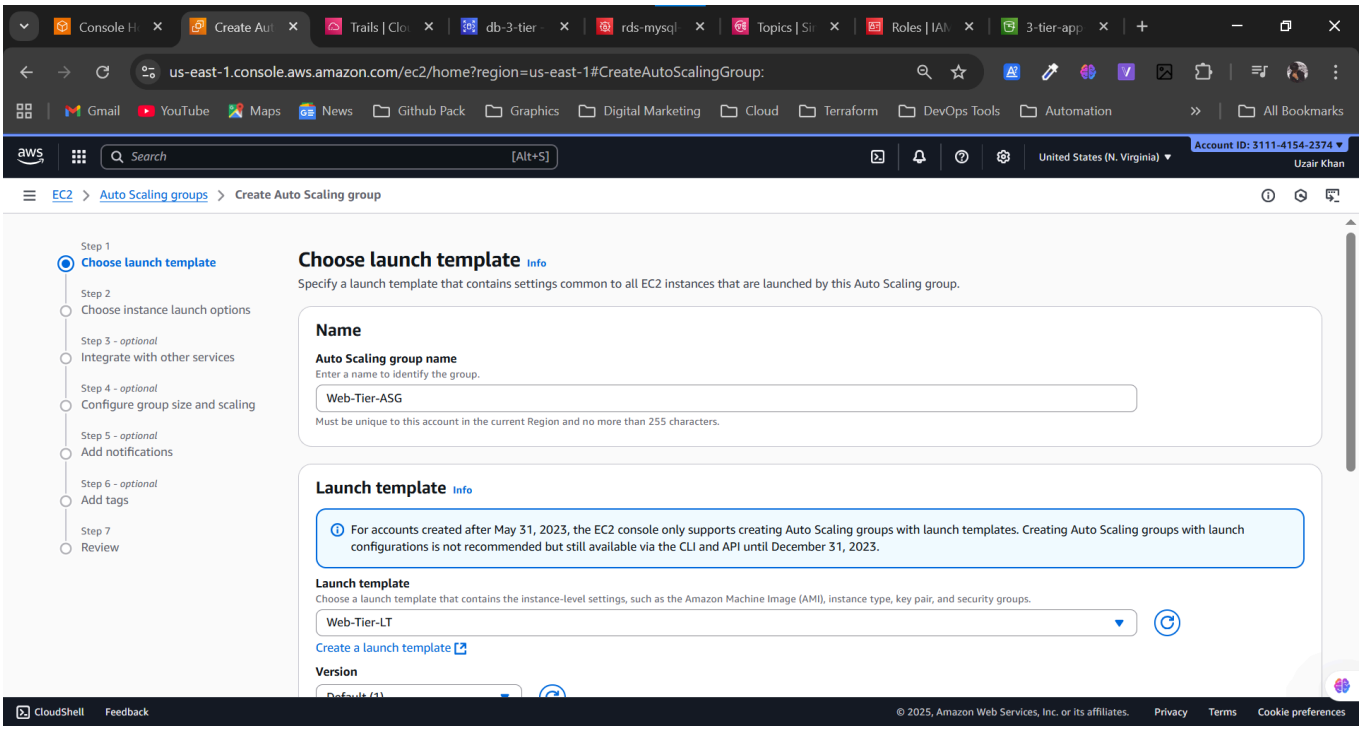


11. Creating Auto-Scaling Groups

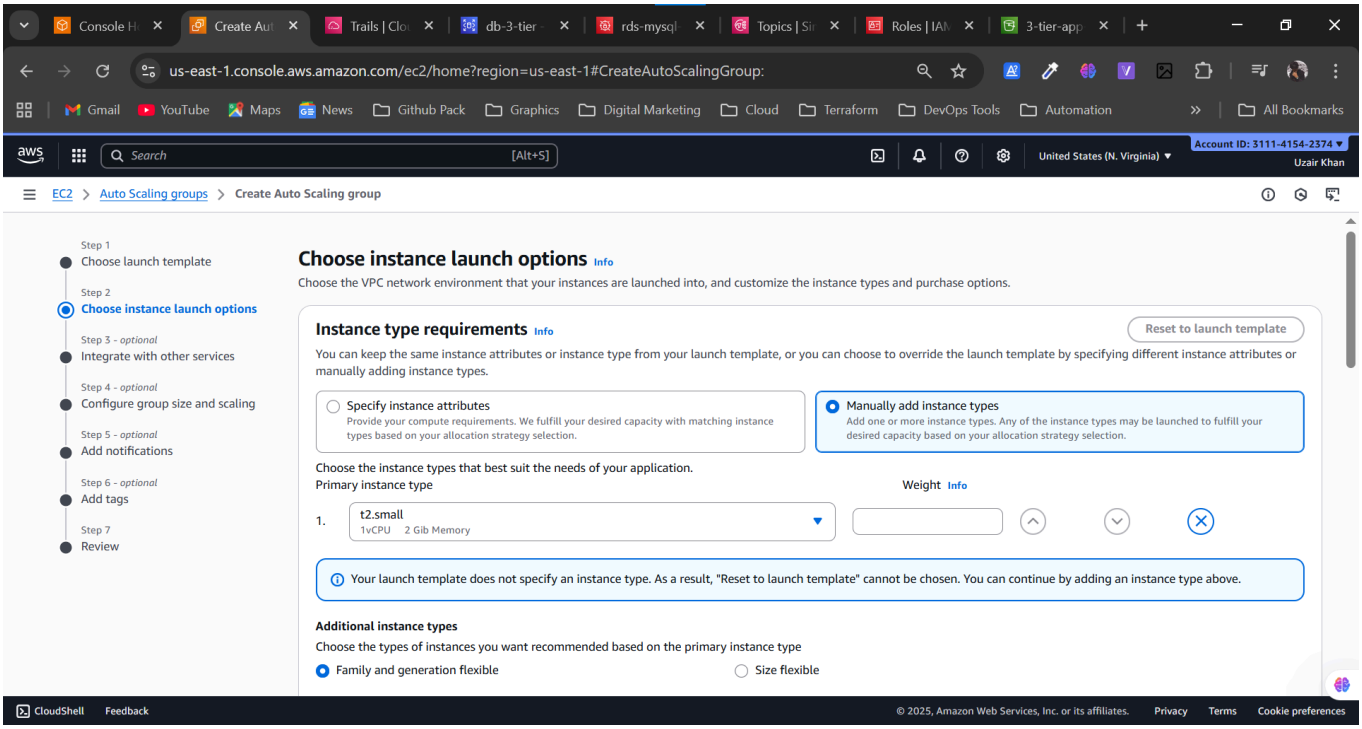
Now you'll create the main component which is **Auto-Scaling Group** for scaling-up and scaling-down your EC2 Instances according to the traffic.

- Go to **EC2** --> **Auto scaling groups**.
- Click on **Create Auto Scaling group**.
- Now select:

- **Name =** `Web-Tier-ASG`
- **Launch template =** `Web-Tier-LT`

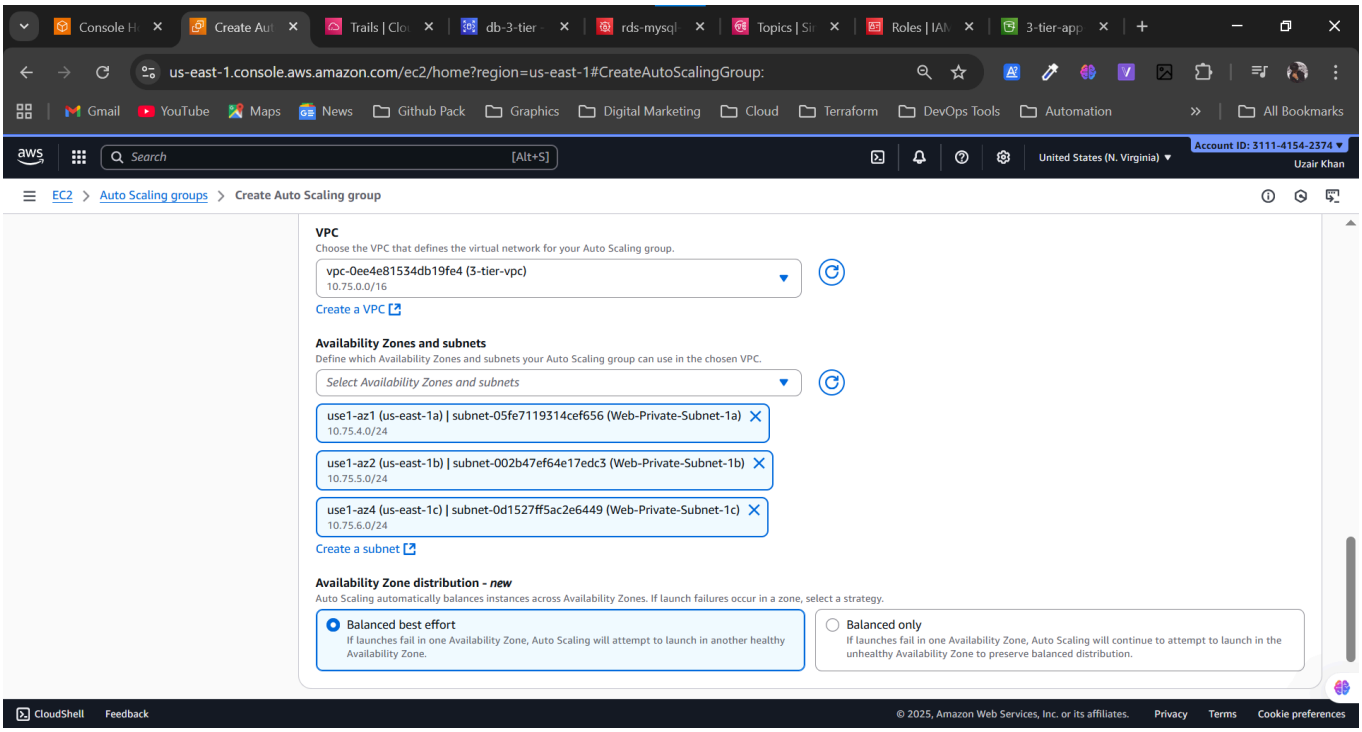


- Now in **Instance type requirements** select:
  - `Manually add instance types`
  - `t2.small`
  - `Family and generation flexible`

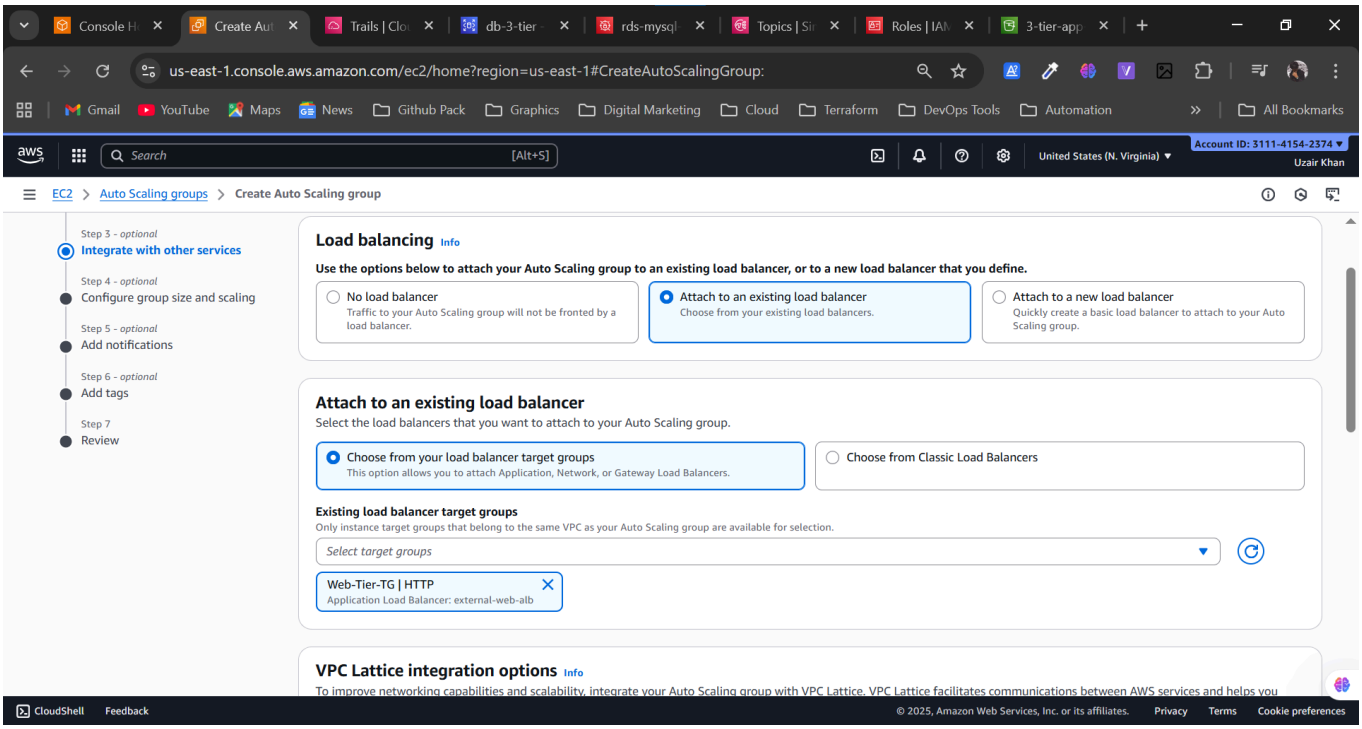


- Now select:

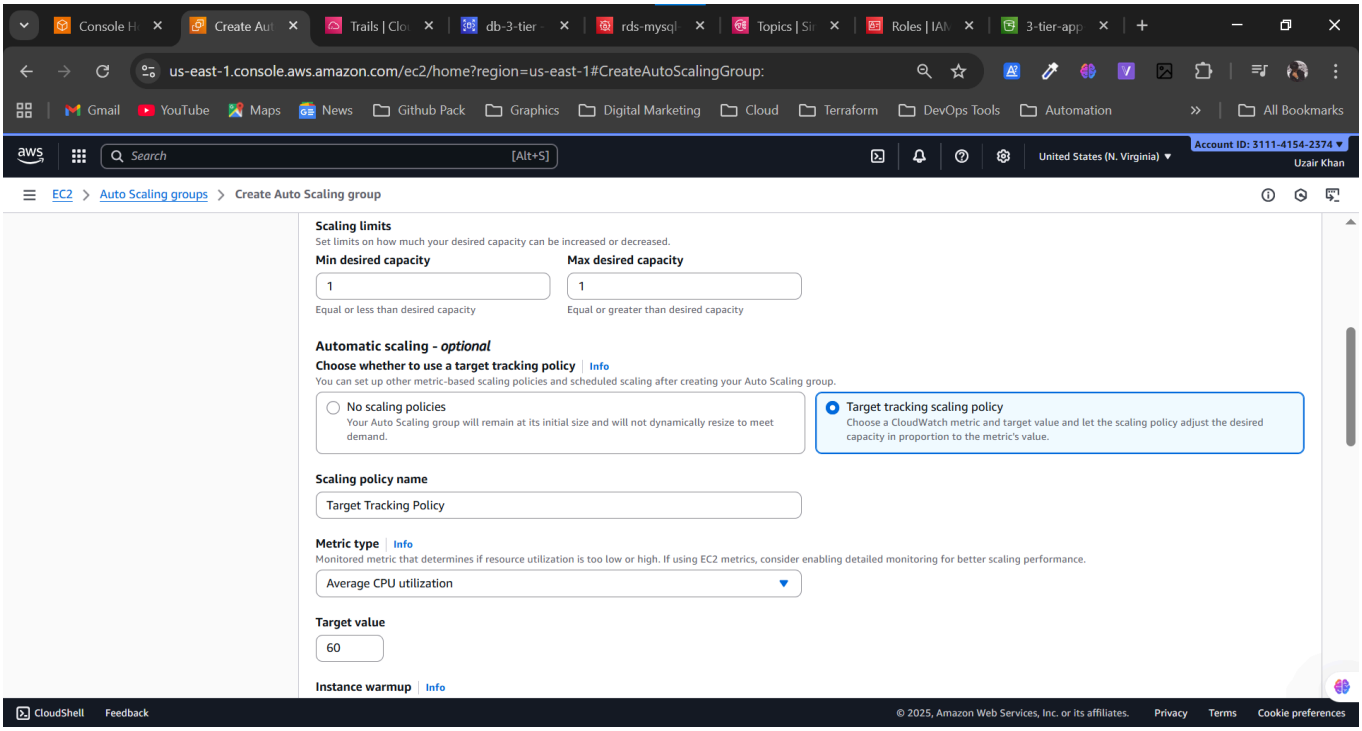
- **VPC** = `3-tier-vpc`
- **Availability Zones and subnets** = `Web-Private-Subnet-1a`, `Web-Private-Subnet-1b`, `Web-Private-Subnet-1c`
- **Availability Zone distribution** = `Balanced best effort`



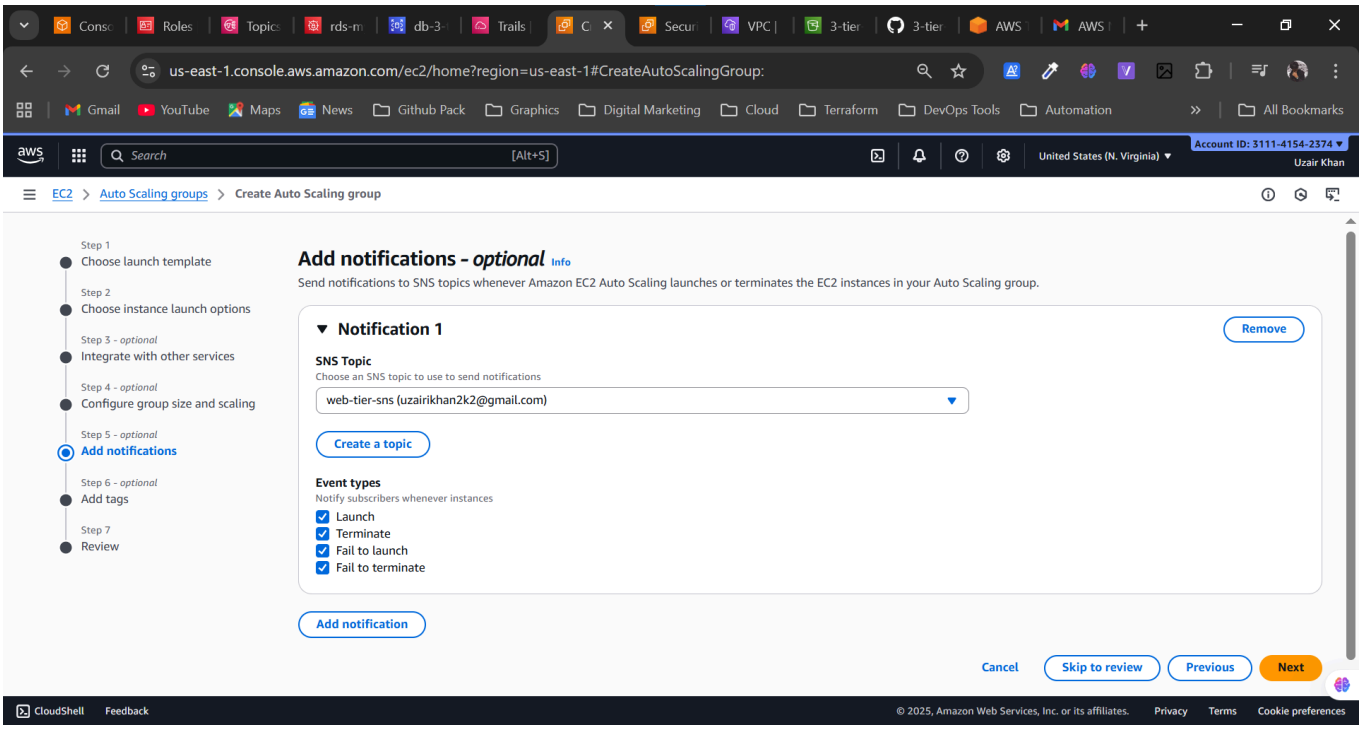
- Now choose:
  - **Load balancing** = `Attach to an existing load balancer`
  - **Attach to an existing load balancer** = `Choose from your load balancer target groups`
  - **Existing load balancer target groups** = `Web-Tier-TG | HTTP`



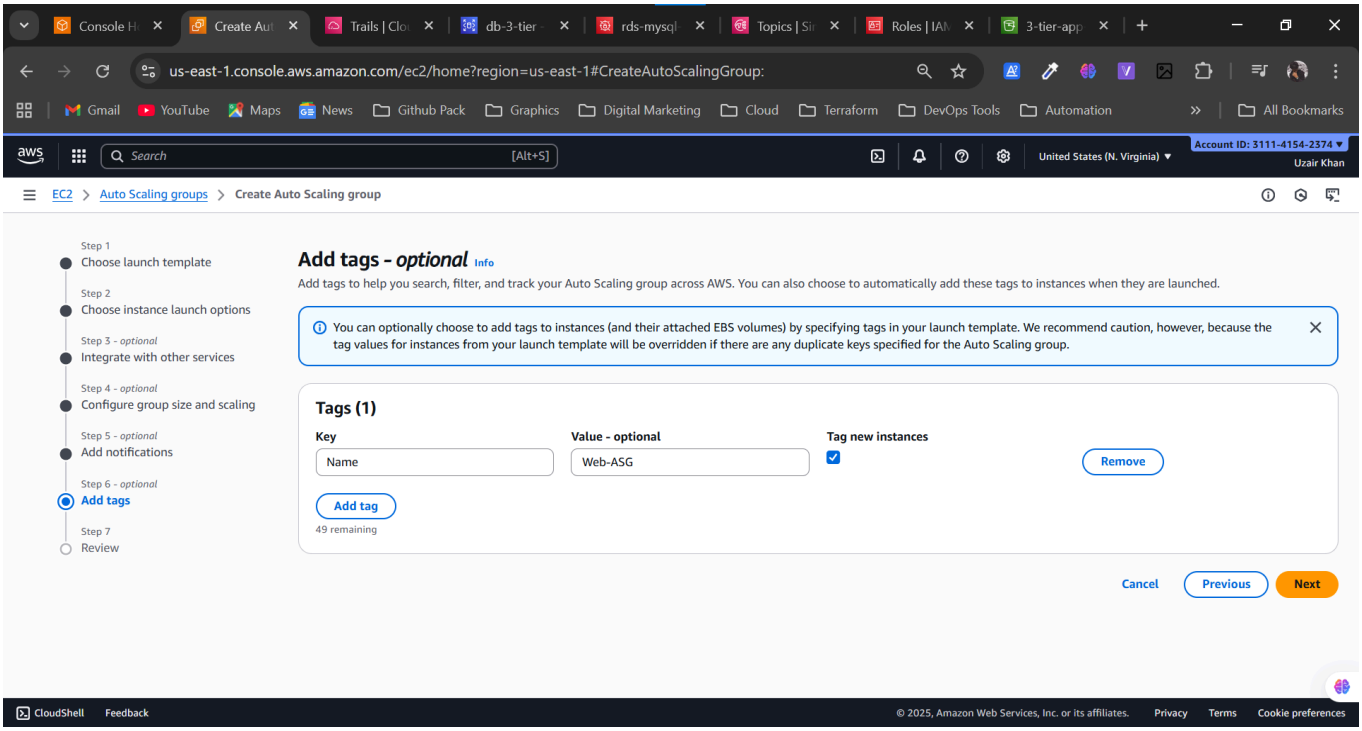
- In the **Scaling** section, choose:
  - **Automatic Scaling** = `Target tracking scaling policy`
  - **Target value** = `60`



- In **SNS Topic** choose `web-tier-sns`

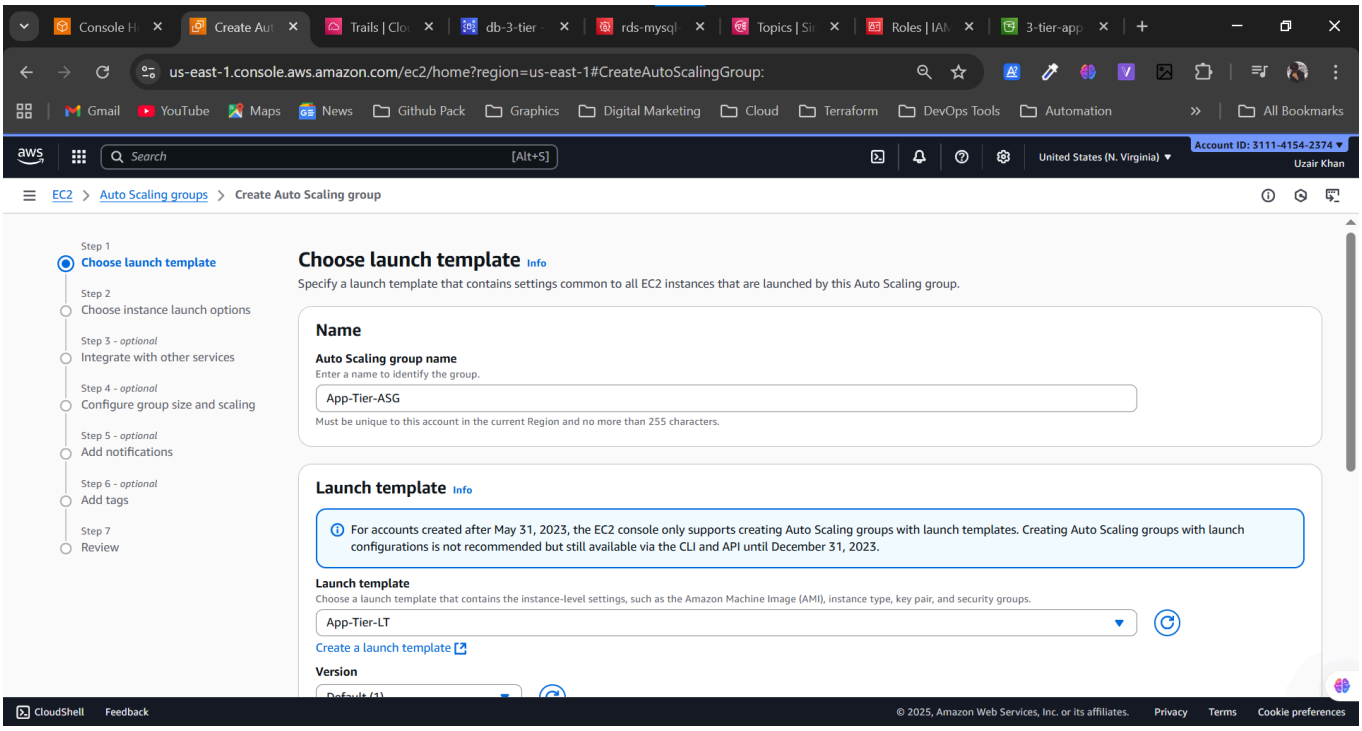


- Add a **tag**:
  - **Key** = `Name`
  - **Value** = `Web-ASG`

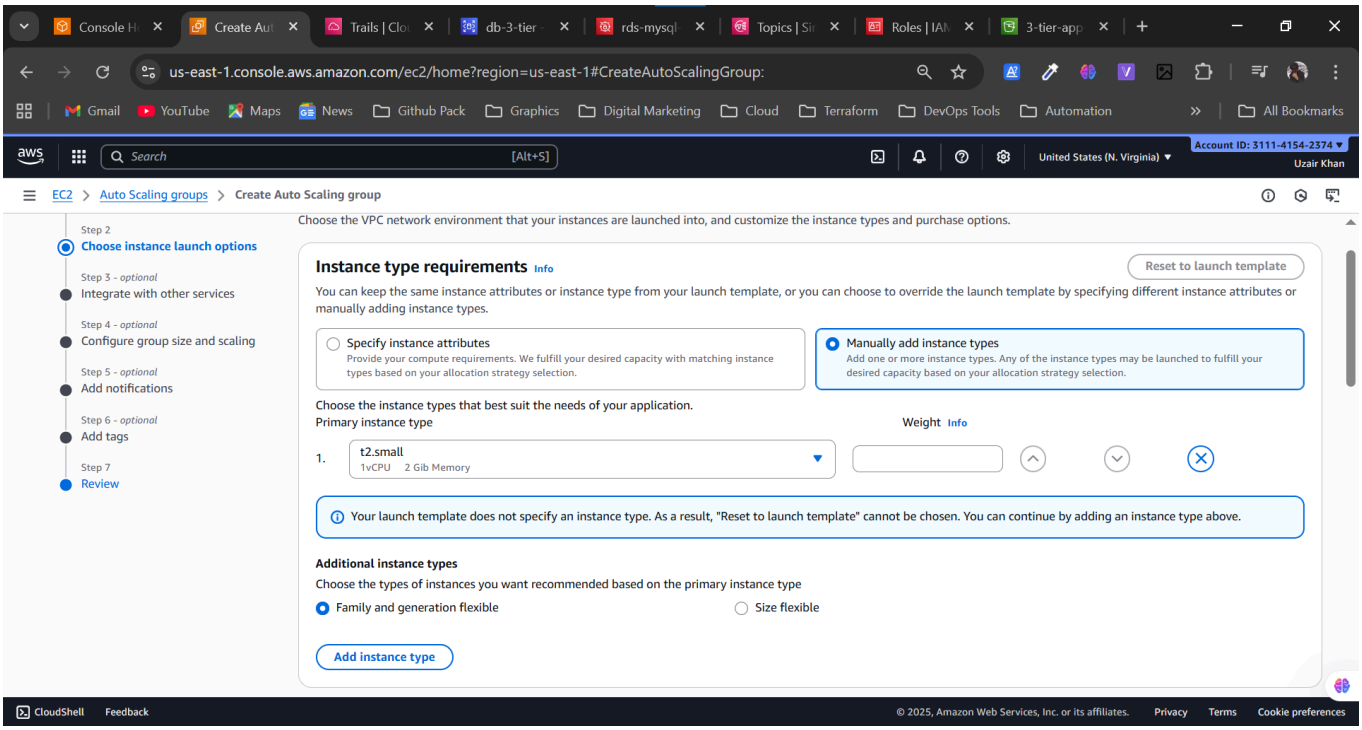


- Click **Next**
- Click **Create Auto Scaling group***.
- Now create the **App Tier** Auto-Scaling Group.
- Now select:
  - **Name** = `App-Tier-ASG`

- **Launch template =** `App-Tier-LT`

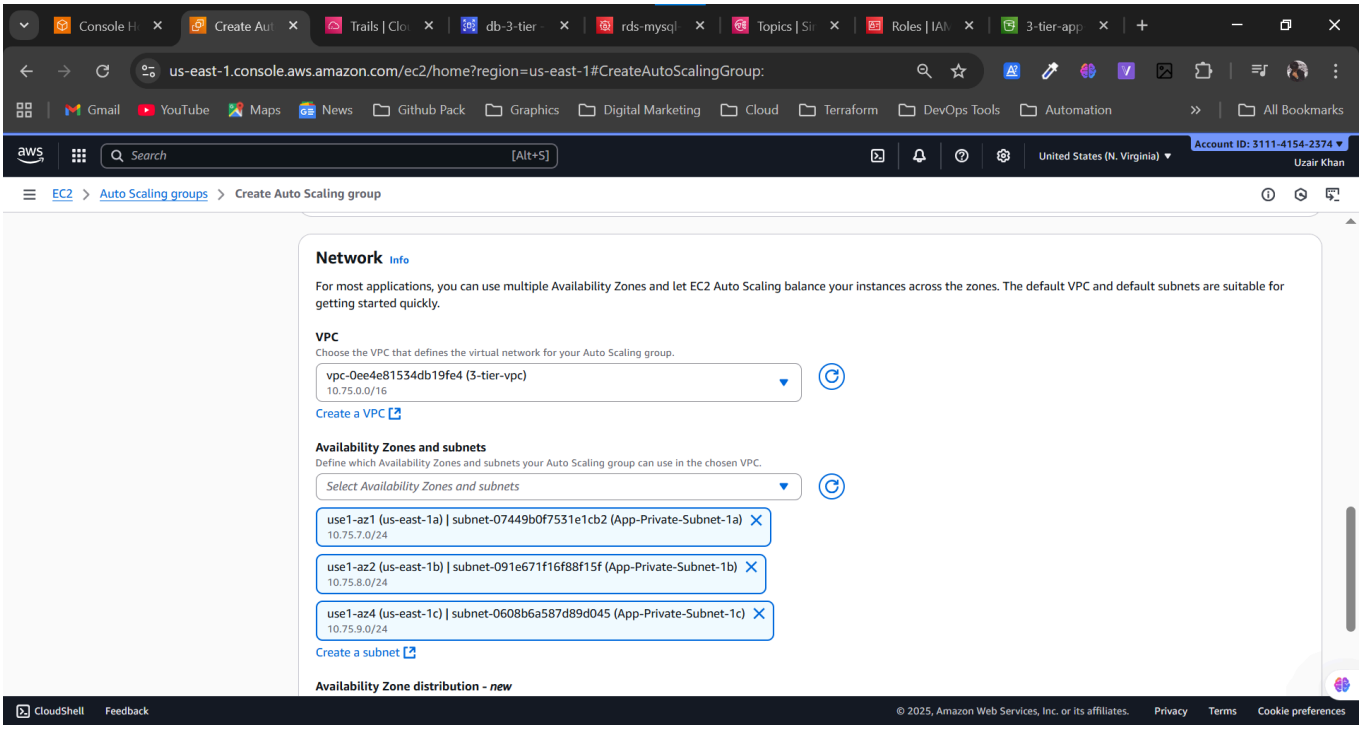


- Now in **Instance type requirements** select:
  - `Manually add instance types`
  - `t2.small`
  - `Family and generation flexible`

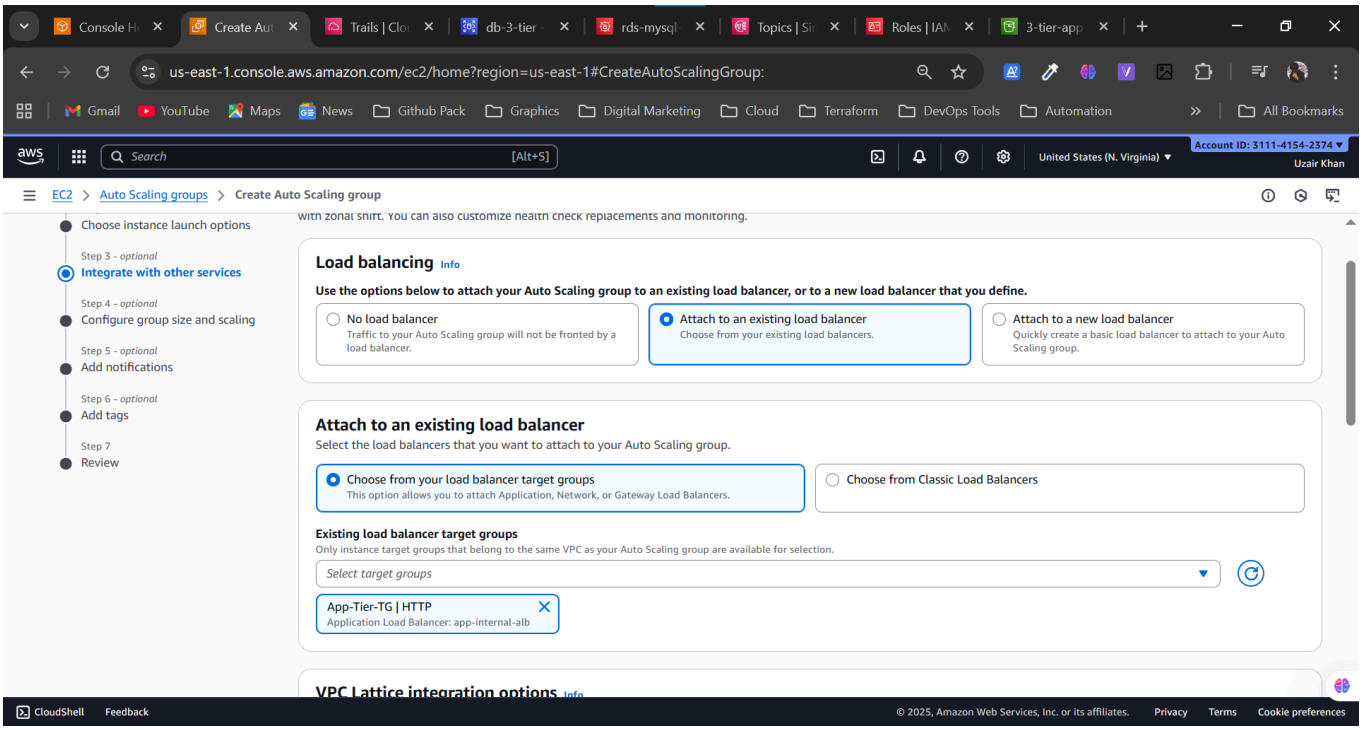


- Now select:
  - **VPC =** `3-tier-vpc`

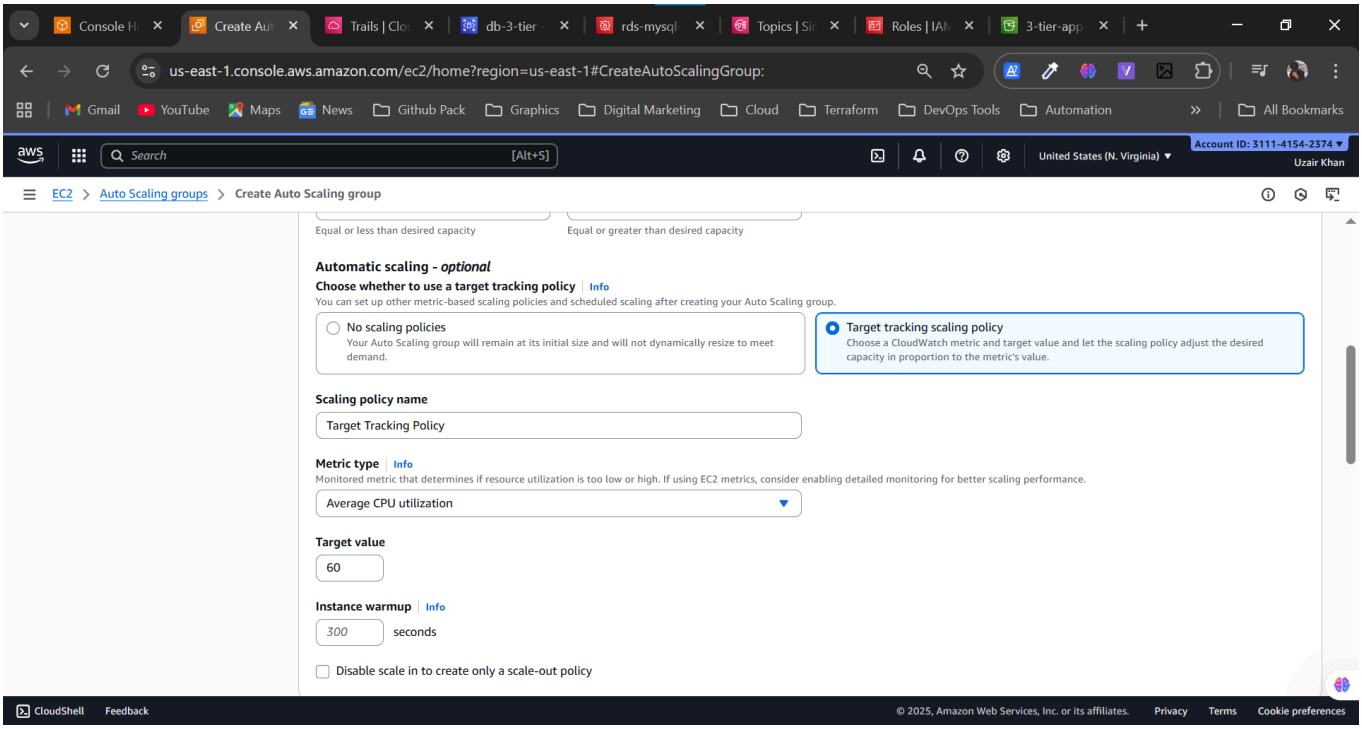
- **Availability Zones and subnets** = `App-Private-Subnet-1a`, `App-Private-Subnet-1b`, `App-Private-Subnet-1c`
- **Availability Zone distribution** = `Balanced best effort`



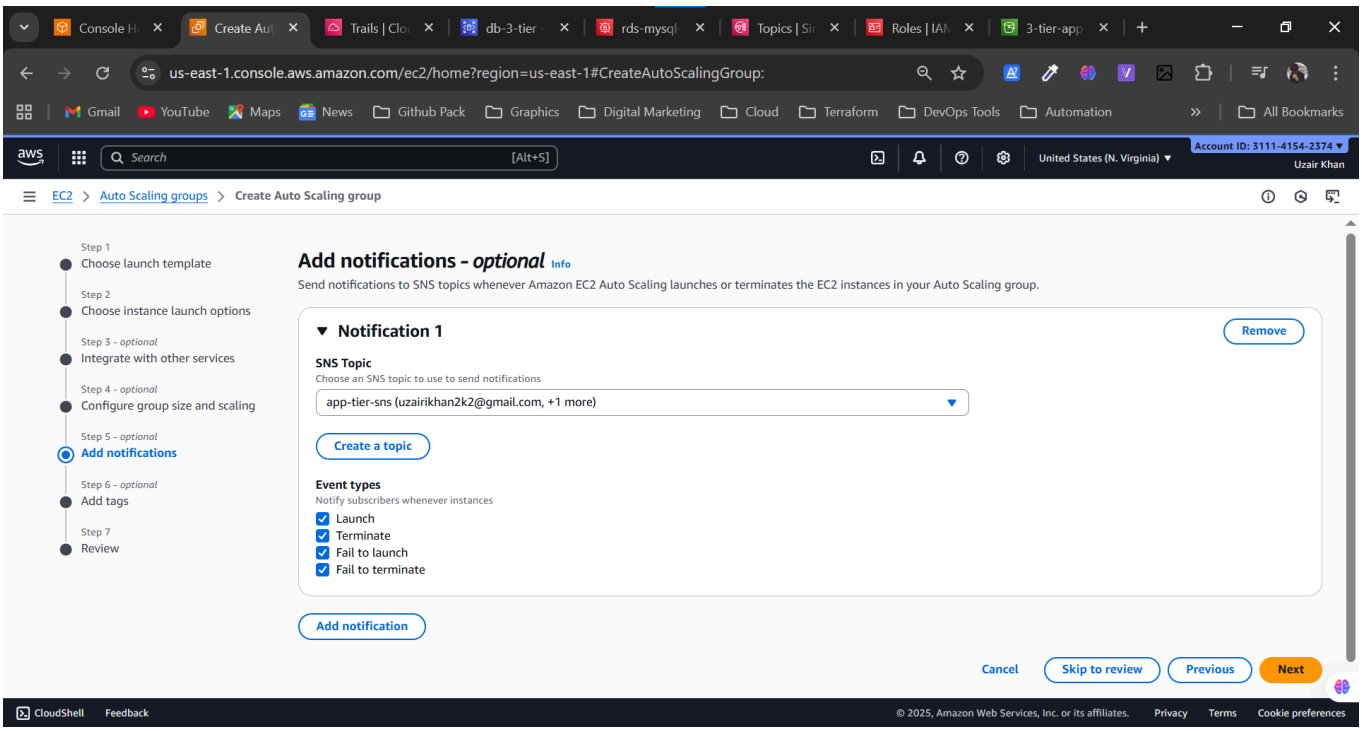
- Now choose:
  - **Load balancing** = `Attach to an existing load balancer`
  - **Attach to an existing load balancer** = `Choose from your load balancer target groups`
  - **Existing load balancer target groups** = `App-Tier-TG | HTTP`



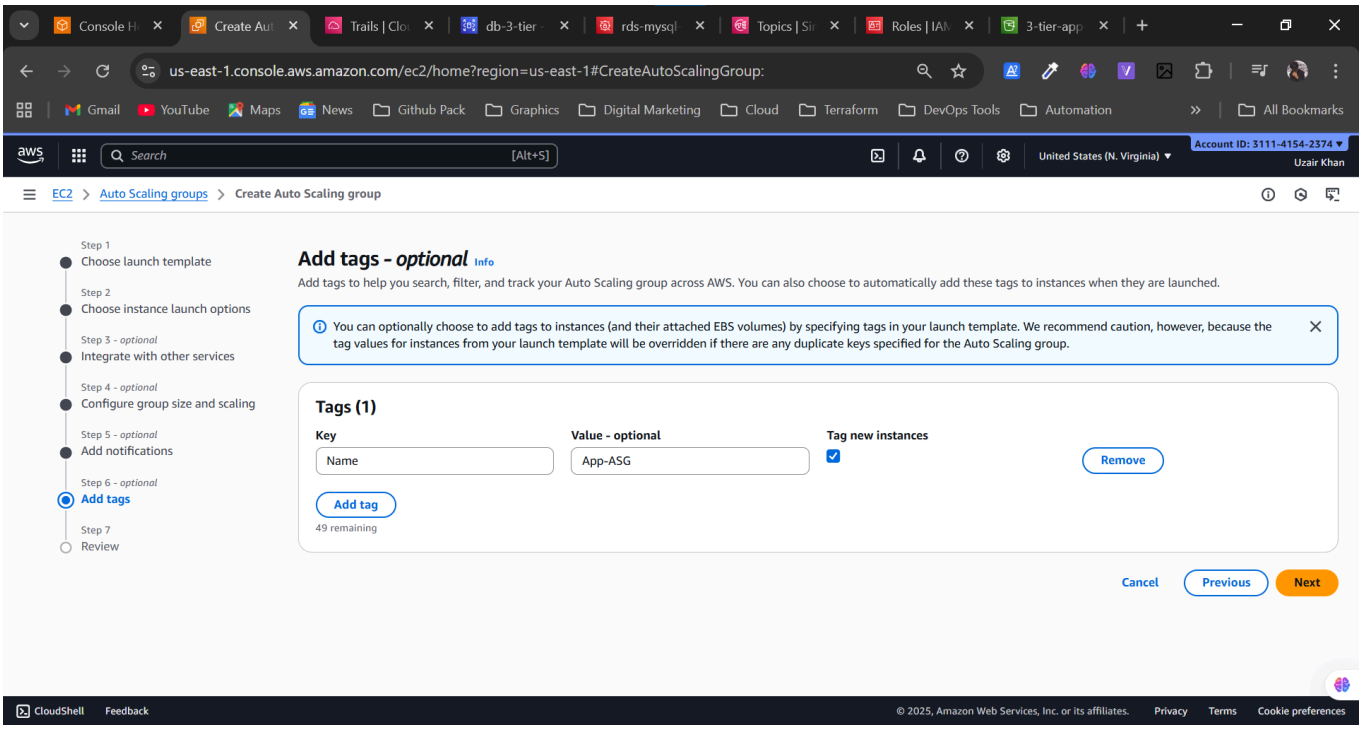
- In the **Scaling** section, choose:
  - **Automatic Scaling** = `Target tracking scaling policy`
  - **Target value** = `60`



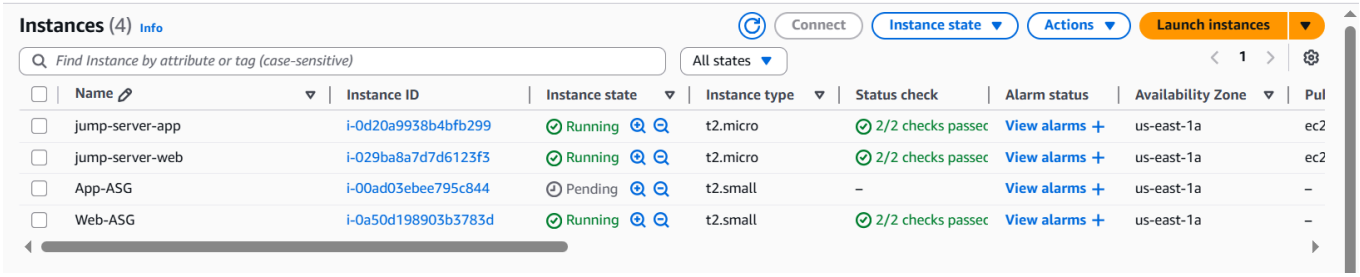
- In **SNS Topic** choose `app-tier-sns`



- Add a **tag**:
  - **Key** = `Name`
  - **Value** = `App-ASG`



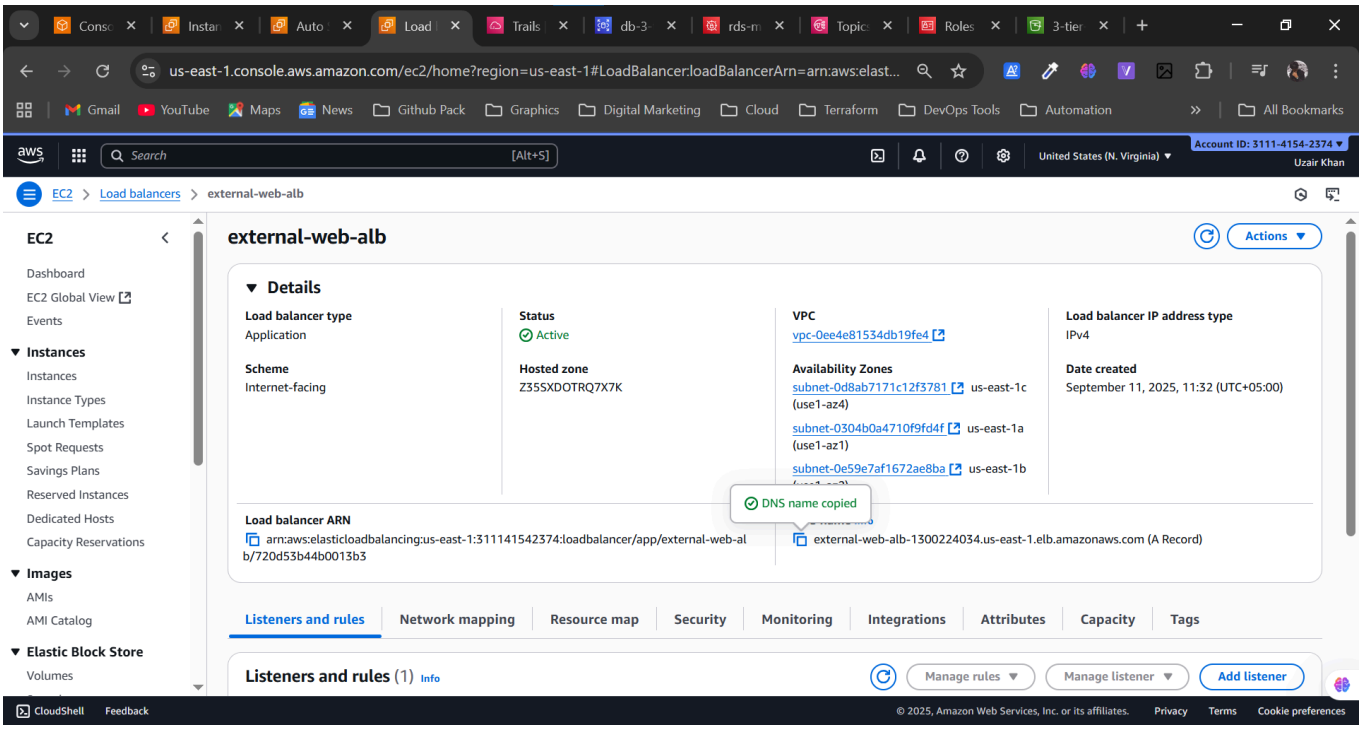
- Click **Next**
- Click **Create Auto Scaling group***.
  Now it will create two new EC2 instances.



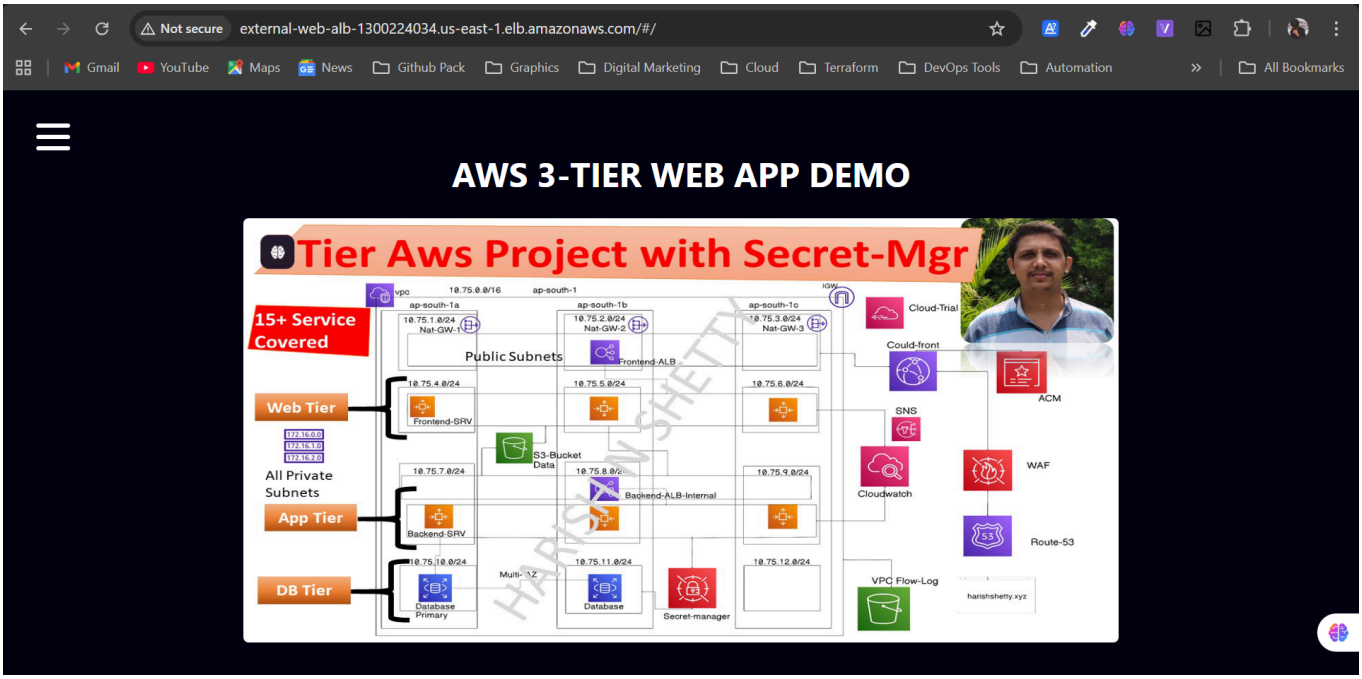
12. Accessing the Application

Now you can access your **Application** with your **External Load Balancer DNS**.

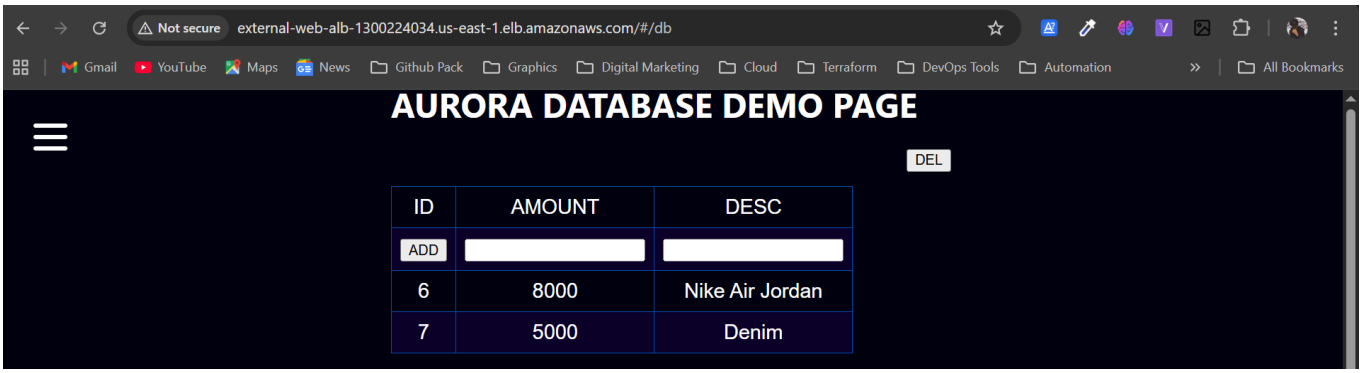
- Go to **external-web-alb**.
- Copy this **DNS**.



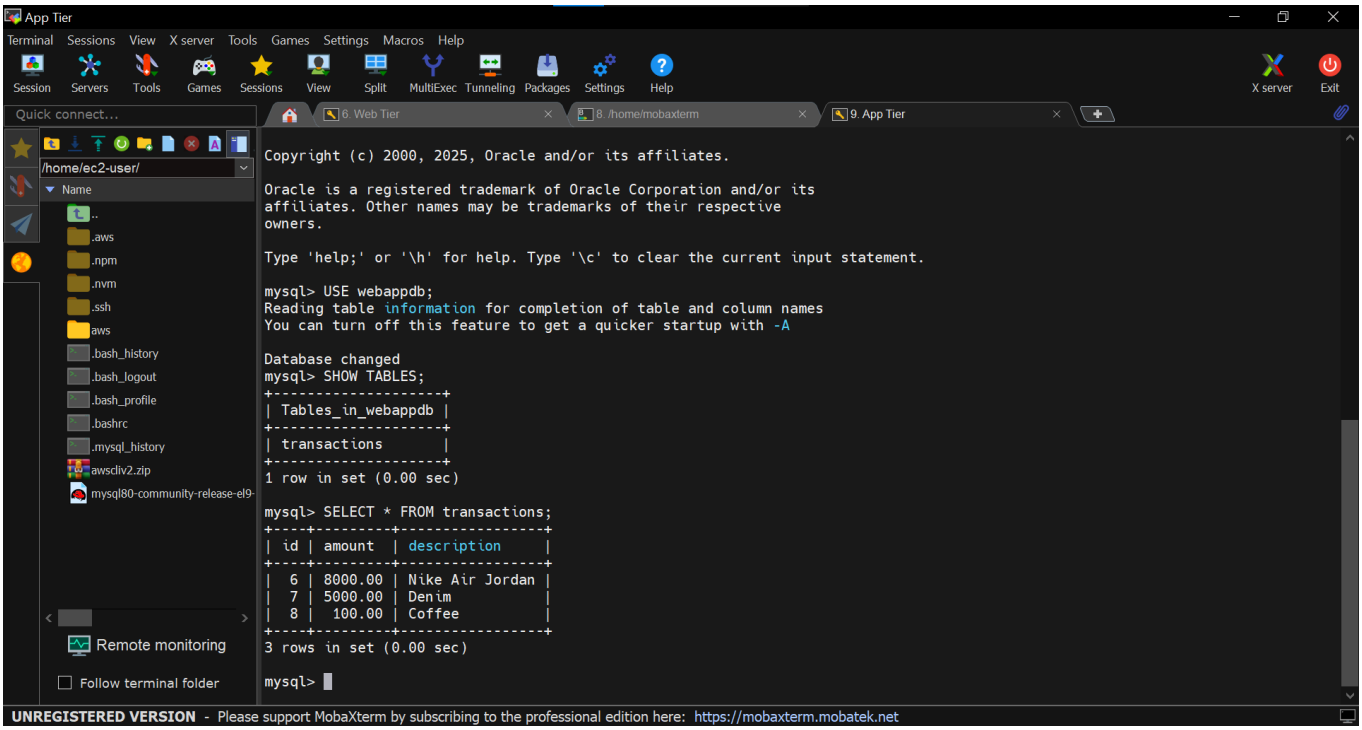
- Paste it in your **web browser**.



- Now add some data in the database.



- You'll see it also in your Database internally.

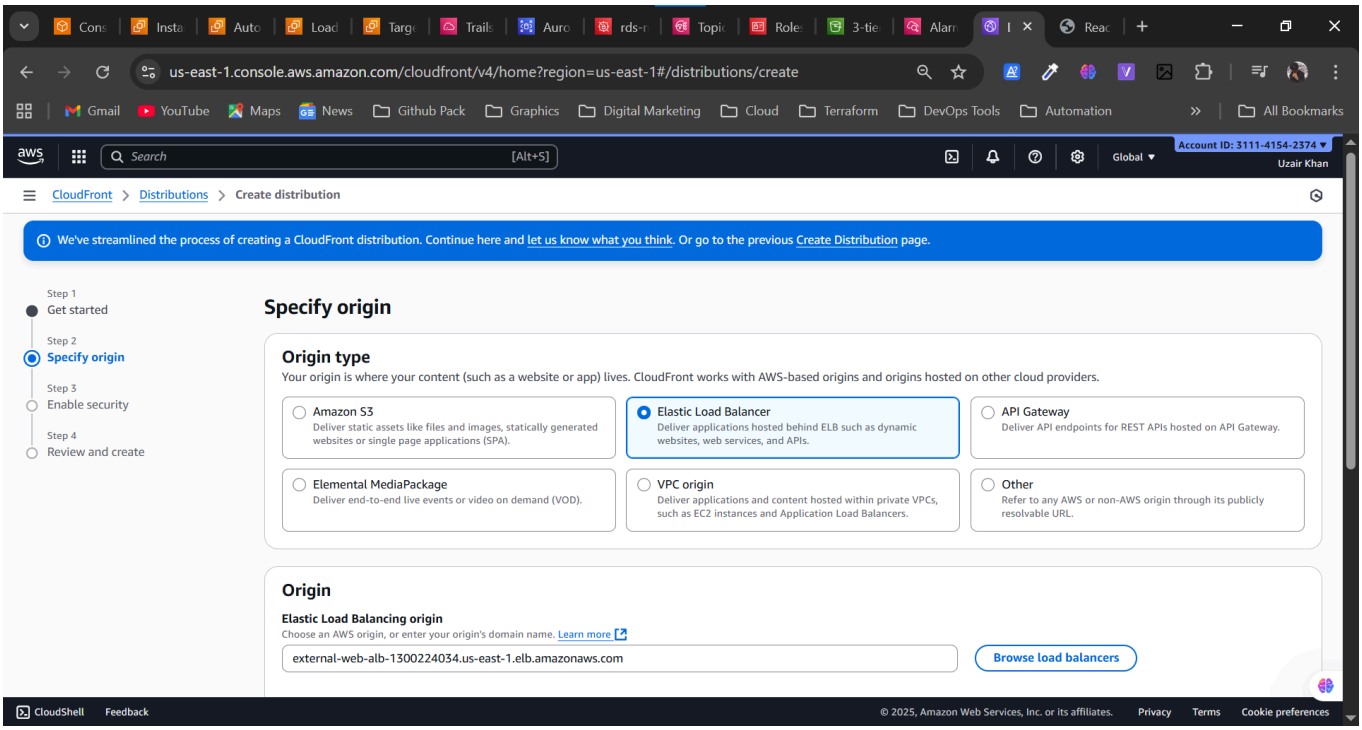


13. Creating a CloudFront Distribution

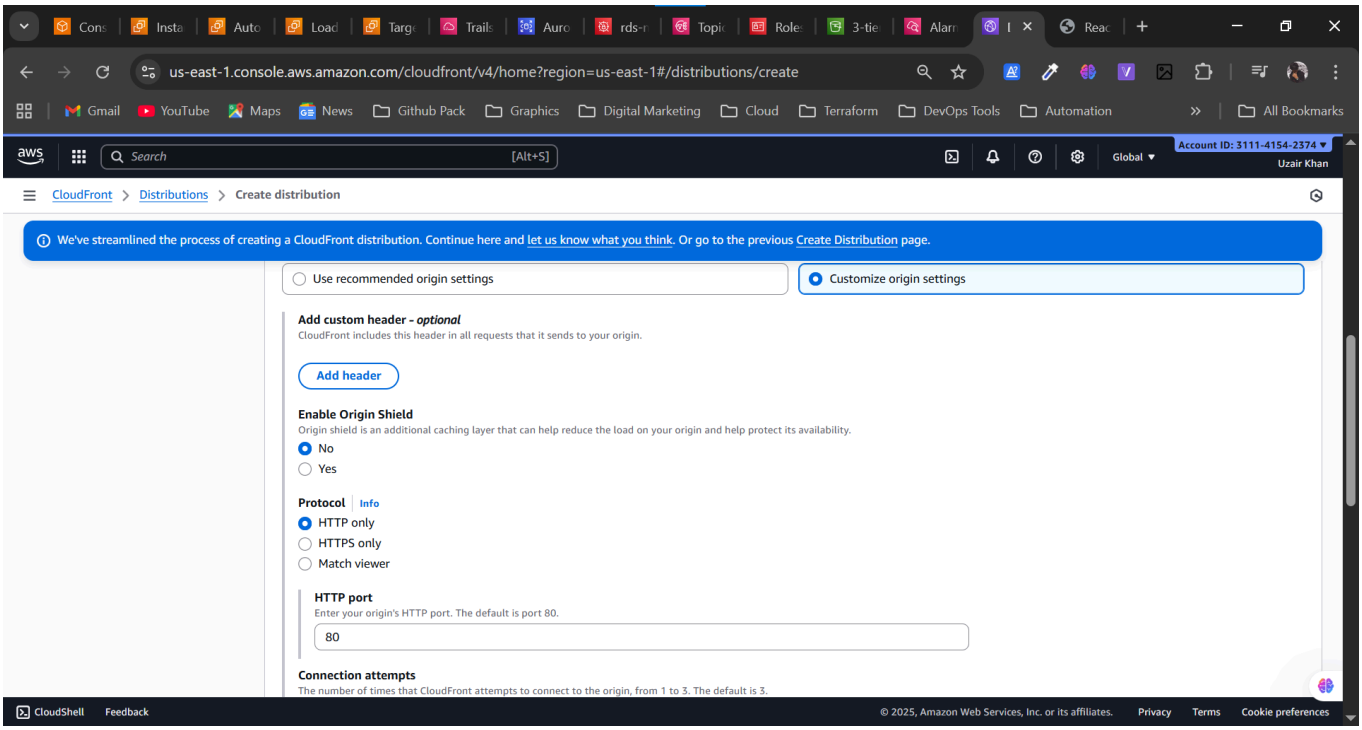
Now you'll have to create a **CloudFront Distribution** for a better latency and access to your application.

- Go to **CloudFront**.
- Click on **Create distribution**.
- Enter the following:
  - **Distribution name** = `three-tier-aws`
  - **Description** = `A CloudFront Distribution for Three Tier App`.
  - **Distribution type** = `Single website or app`
  - **Custom domain** = `Add if you have one otherwise leave it blank`
  - **Tag**
    - `Name` = `three-tier-aws`

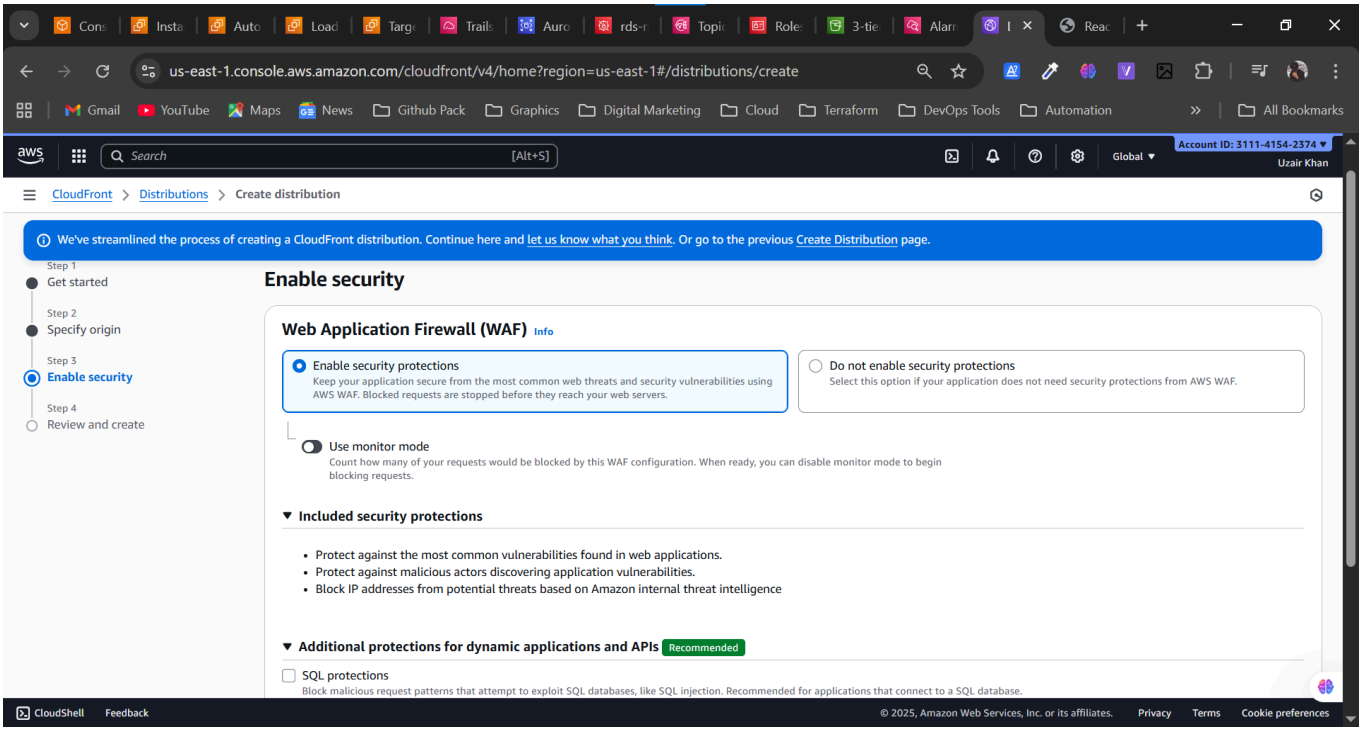
- Click **Next**.
- In **Specify origin** select:
  - **Origin type** = `Elastic Load Balancer`
  - **Elastic Load Balancing origin** = `external-web-alb`



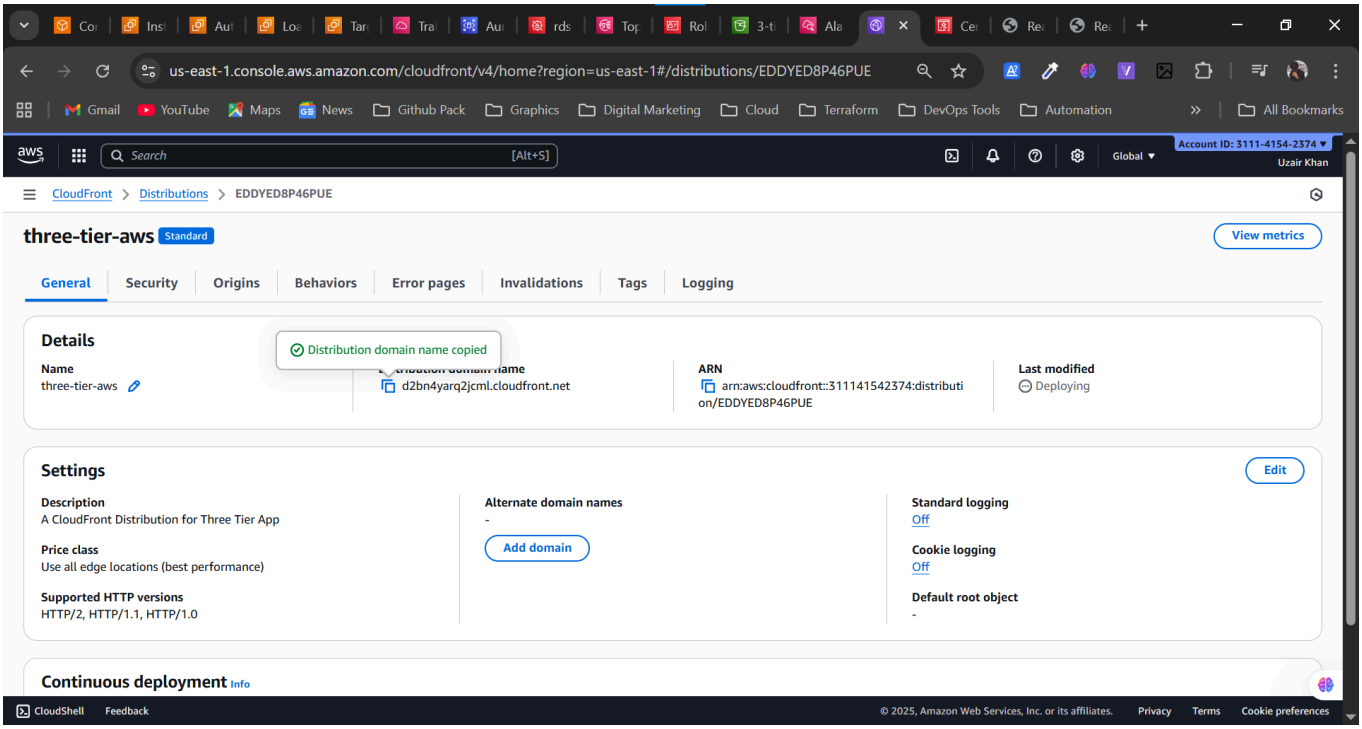
- In **Customize origin settings** choose:
  - **Enable Origin Shield** = `No`
  - **Protocol** = `HTTP only`
  - **HTTP port** = `80`



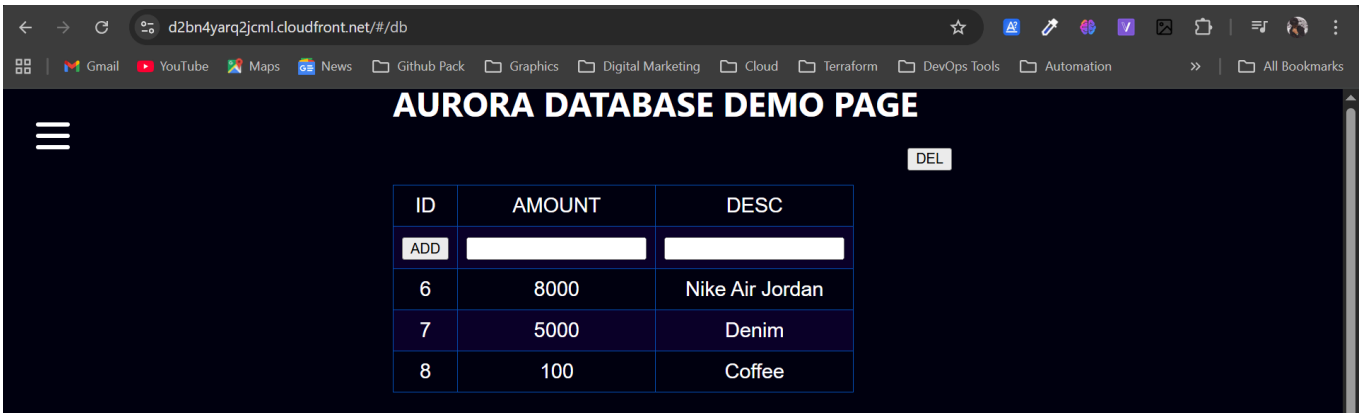
- In **Web Application Firewall (WAF)** select `Enable security protections` .



- Click **Next**.
- Click **Create distribution**.
- Now copy this **DNS**.



- Paste it into your web browser.

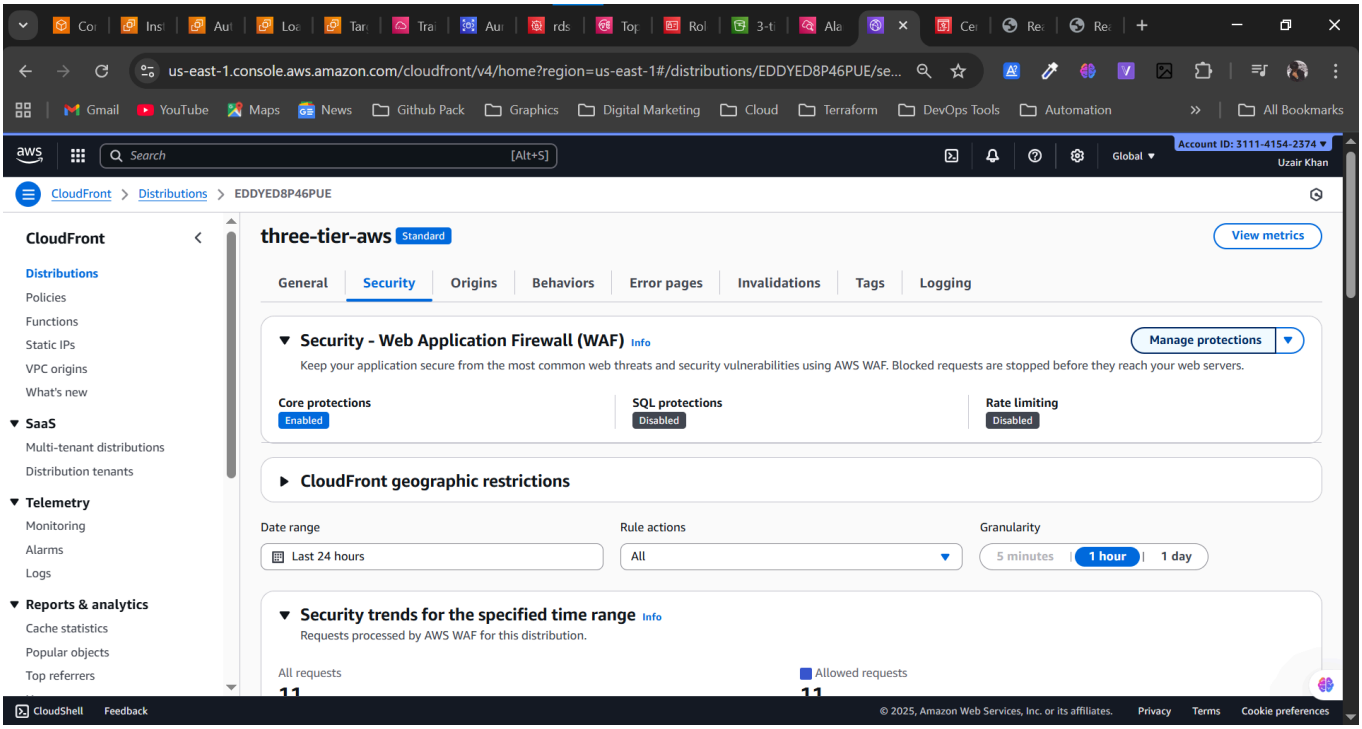


Welldone! You are now accessing your application with CloudFront + AWS WAF for Website Security

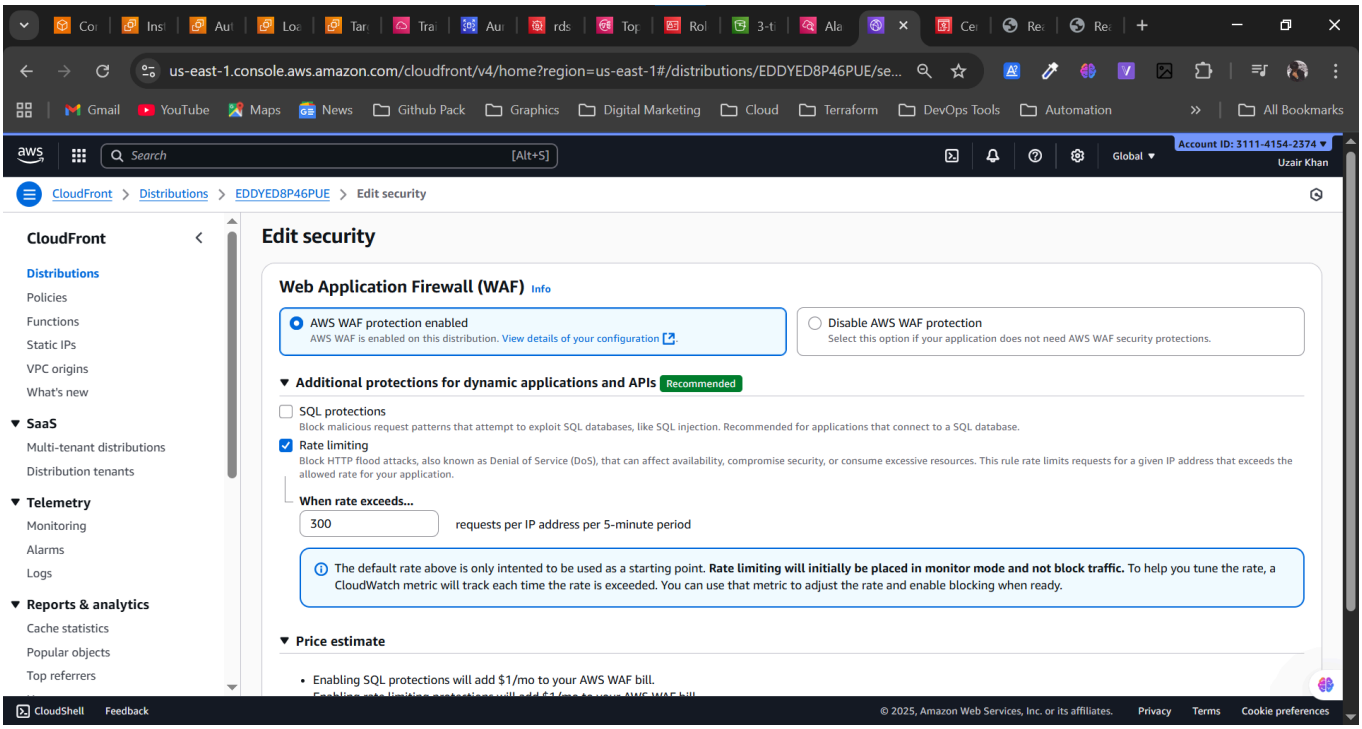
14. Customizing WAF and CloudFront

Now this is the epic and fun part where you'll be customizing **Web ACL rules** for any incoming traffic to your application.

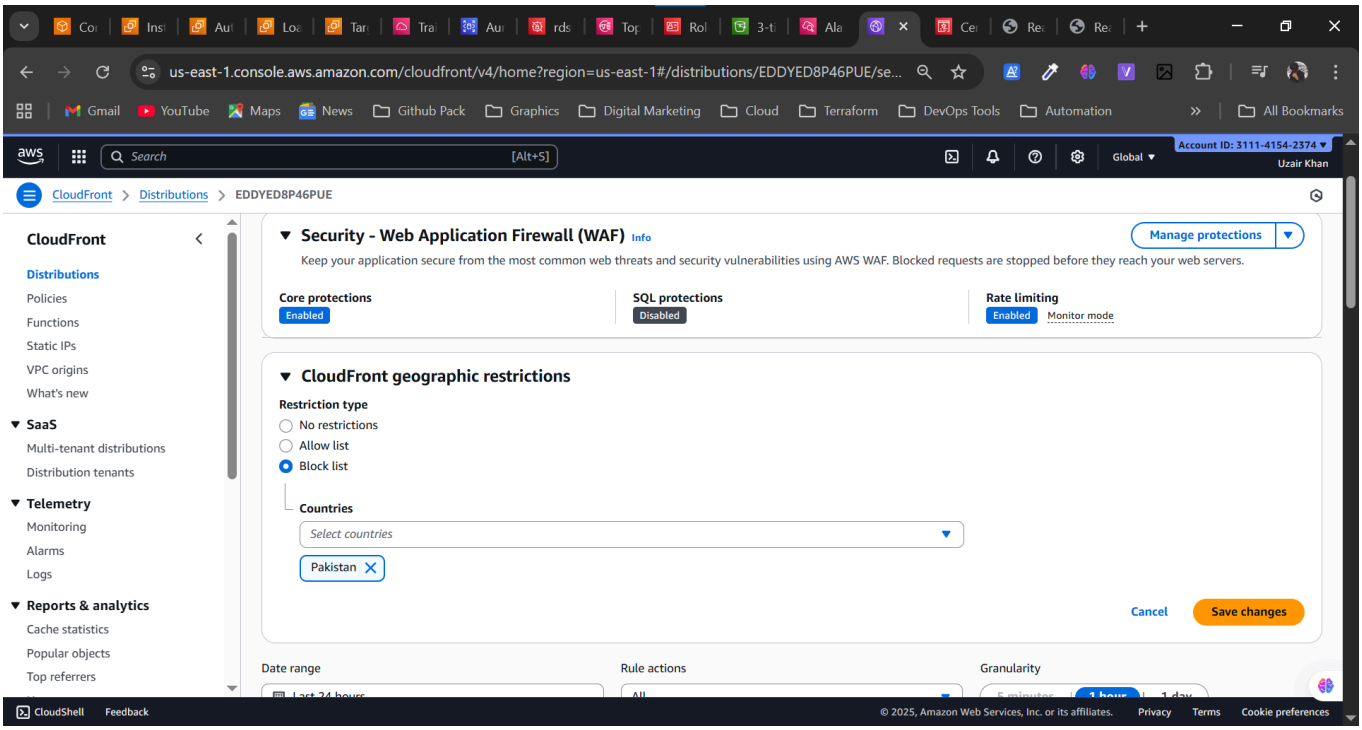
- Go to your **three-tier-aws** distribuiton.
- Go to **Security** tab.
- Click **Manage protections**.



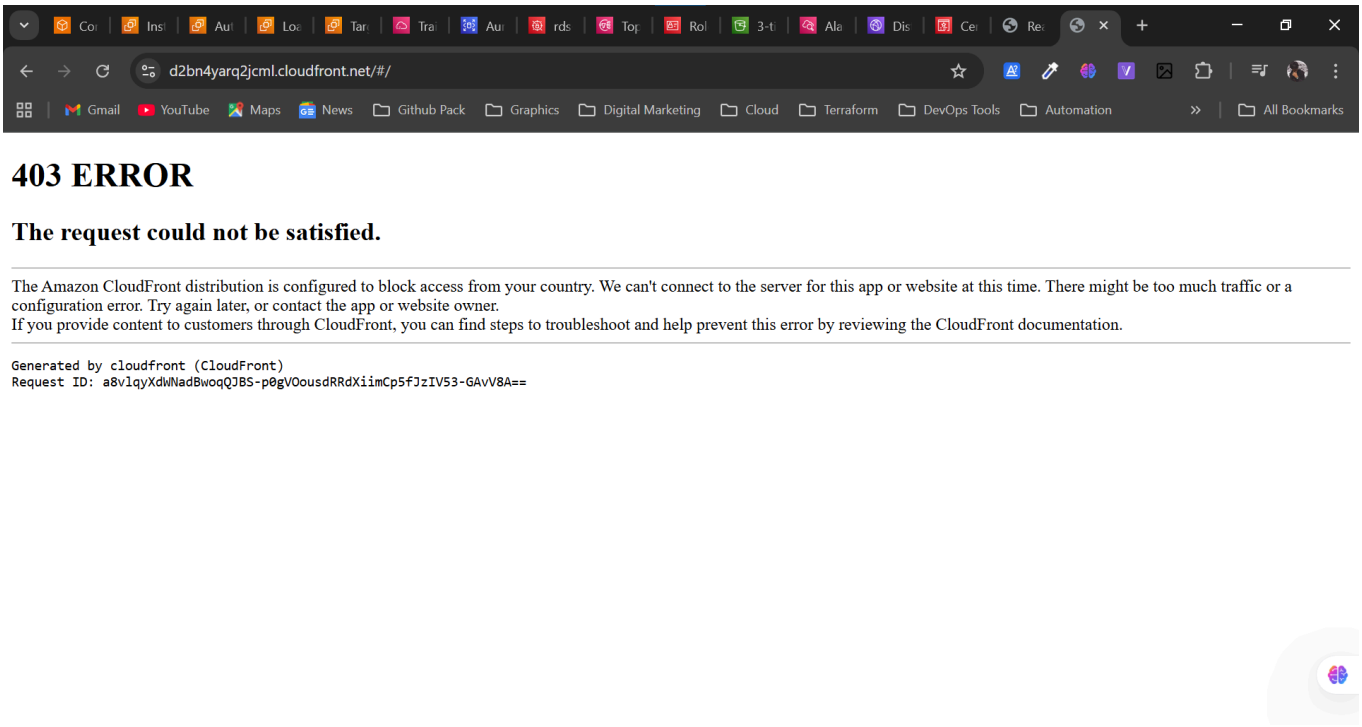
- Check-Mark **Rate limiting**.



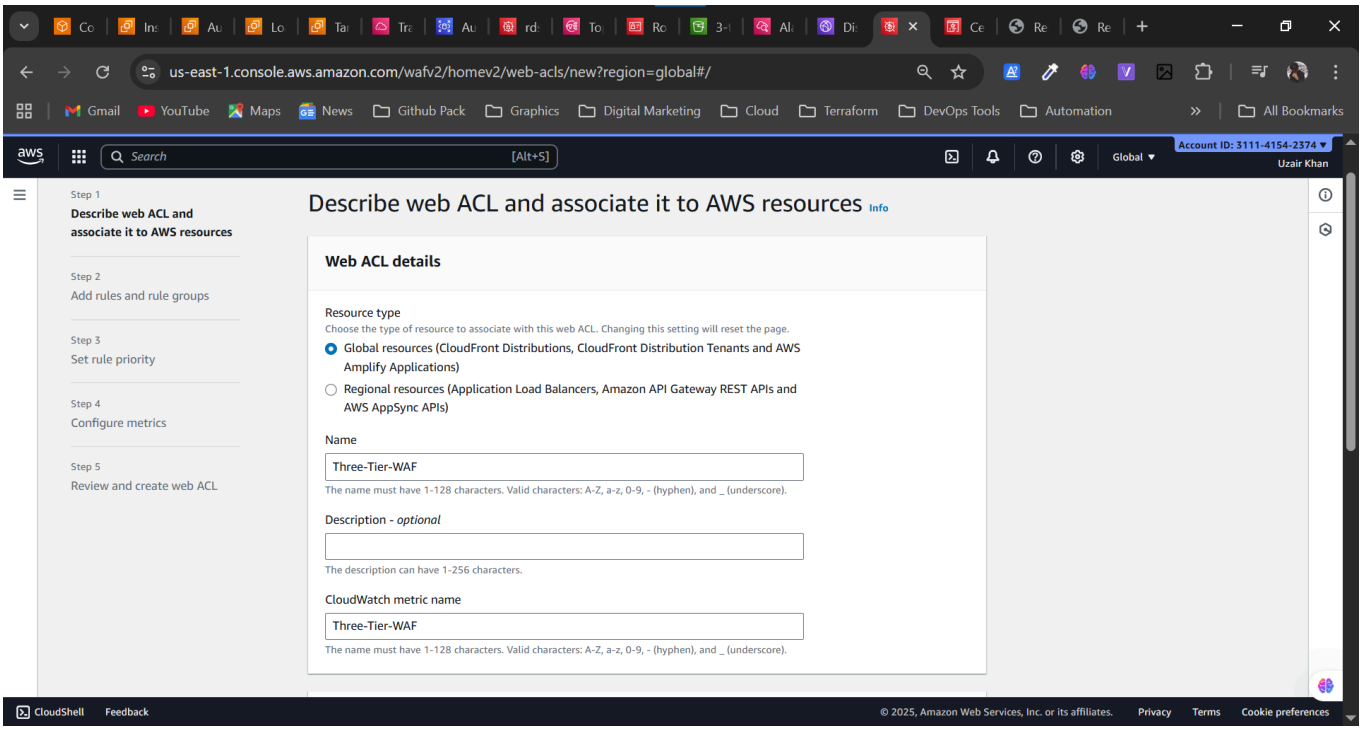
- Click **Save changes**.
- In **CloudFront geographic restrictions** add `[Your Country]` to **Block list**.



- Click **Save changes**.
- Now you'll get a **403 error** from **CloudFront**.

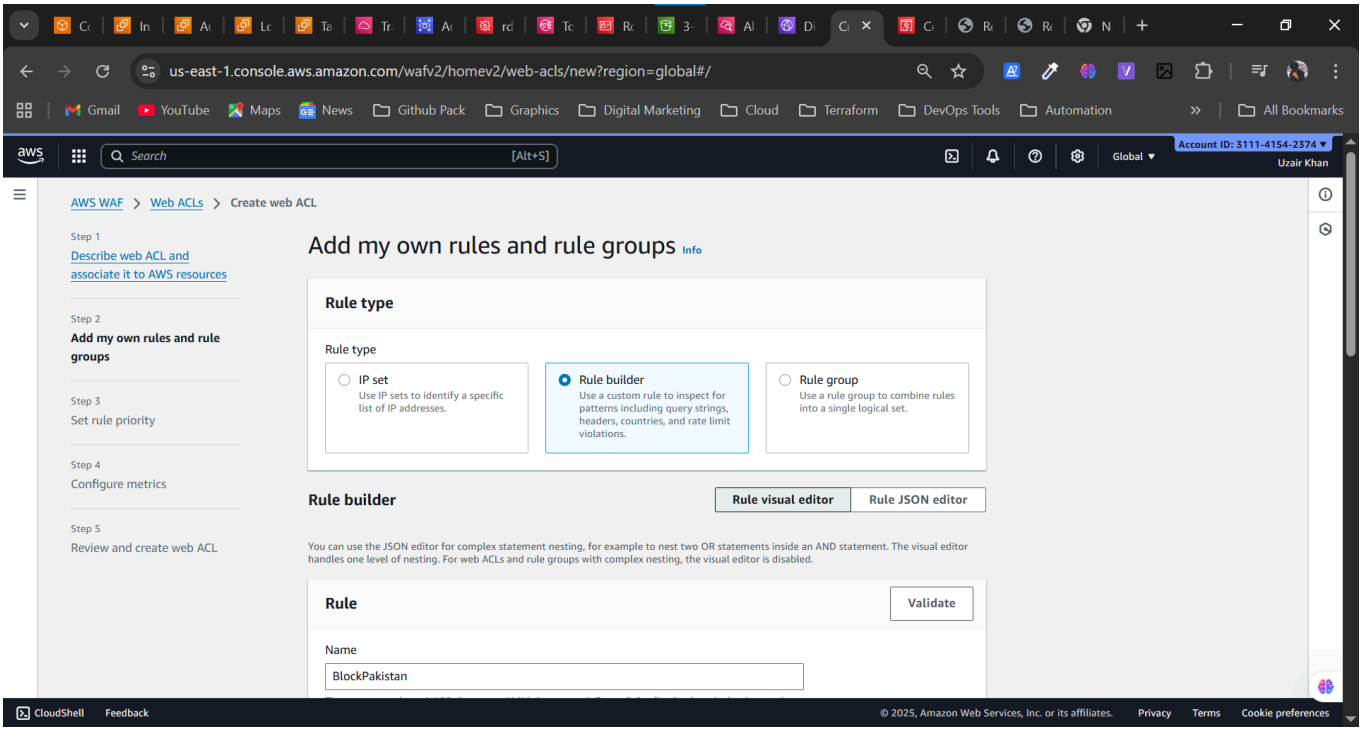


- Now go to **WAF & Shield**.
- Click **Create web ACL**.
- Select **Global resources**.
- Enter **Name** = `Three-Tier-WAF`

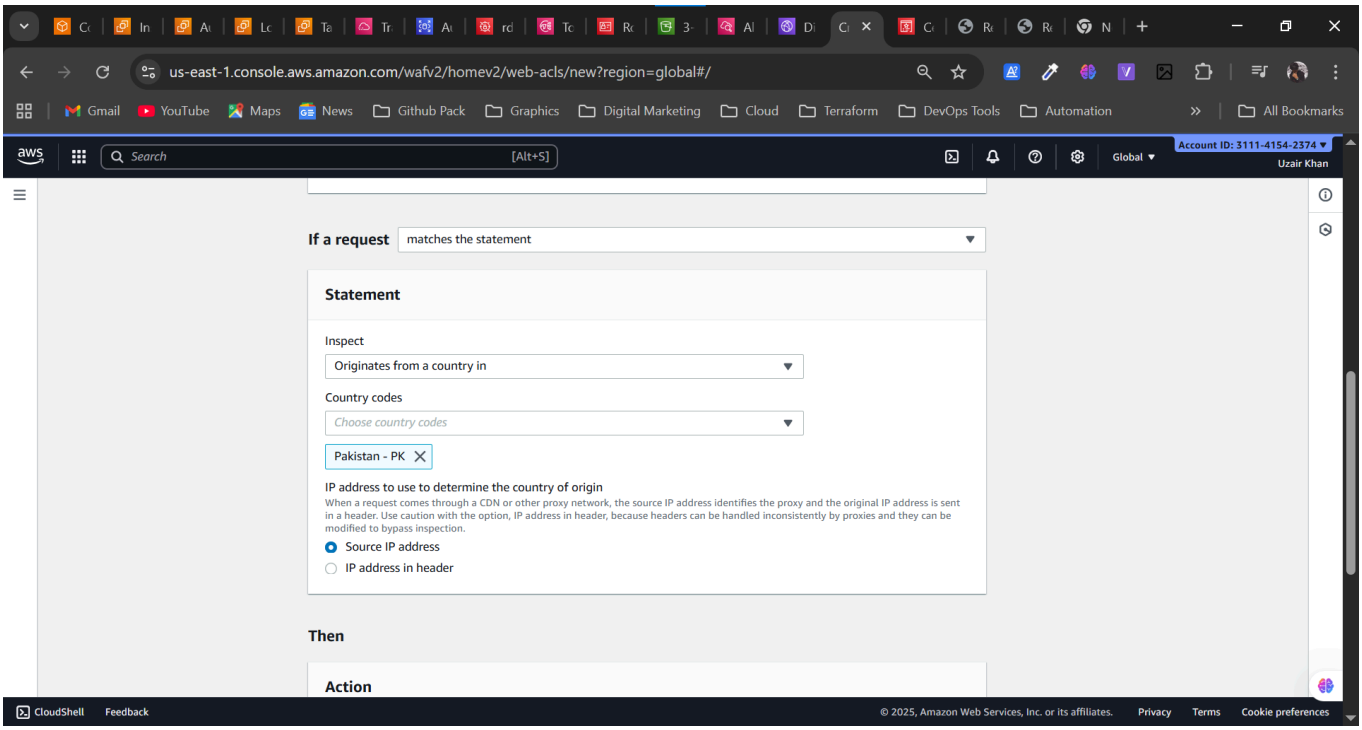


- Click on **Add AWS resources** and add your **CloudFront distribution**.
- Click **Next**.
- Select **Add my own rules and rule groups**.
- In **Rule type** select `Rule builder`.

- Enter **Name** = `Block[MyCountry]WithCustomResponse`.



- Now select:
  - **If a request** = `matches the statement`
  - **Inspect** = `Originates from a country in`
  - **Country codes** = `[Your Country Code]`



- In **Action** choose `Block`.
- In **Custom response** select:

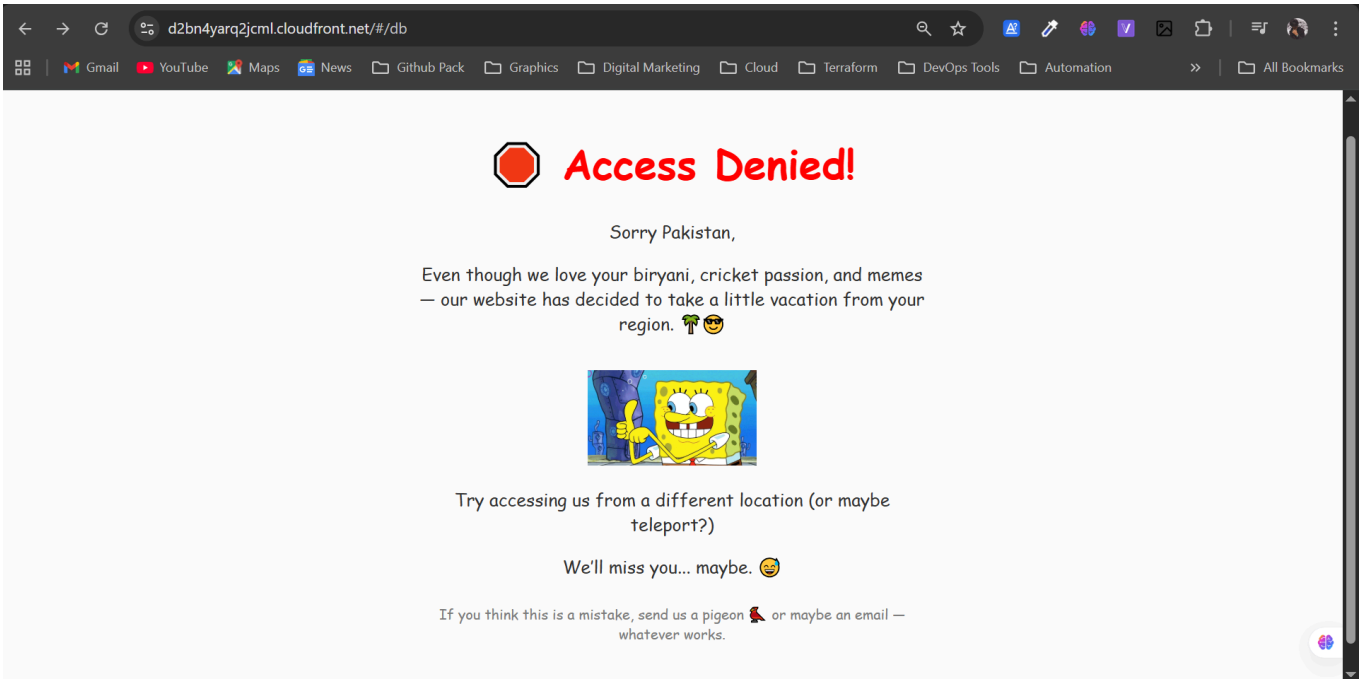
- **Response code** = `404`
- **Choose how you would like to specify the response body** = `Create a custom reponse body`
- **Response body object name** = `Block[MyCountry]WithCustomResponse`
- **Content type** = `HTML`

- Paste this in **Response body**:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Oops! PK Not Today, Pakistan</title>
  <style>
    body {
      font-family: 'Comic Sans MS', cursive, sans-serif;
      background-color: #fefefe;
      color: #333;
      text-align: center;
      padding: 50px;
    }
    img {
      max-width: 200px;
      margin-top: 20px;
    }
    .container {
      max-width: 600px;
      margin: auto;
    }
    h1 {
      font-size: 3em;
      color: #ff0000;
    }
    p {
      font-size: 1.3em;
      margin-top: 20px;
    }
    .tiny {
      font-size: 0.8em;
      color: gray;
      margin-top: 30px;
    }
  </style>
</head>
<body>
```

```
38    <div class="container">
39      <h1>🛑 Access Denied!</h1>
40      <p>Sorry Pakistan,</p>
41      <p>Even though we love your biryani, cricket passion, and memes — our website
    has decided to take a little vacation from your region. 🌴😎</p>
42      <img src="https://media.giphy.com/media/3o7abKhOpu0NwenH3O/giphy.gif" alt="No
    Access">
43      <p>Try accessing us from a different location (or maybe teleport?)</p>
44      <p>We'll miss you... maybe. 😅</p>
45      <div class="tiny">
46        <p>If you think this is a mistake, send us a pigeon 🐦 or maybe an email —
    whatever works.</p>
47      </div>
48    </div>
49  </body>
50  </html>
```

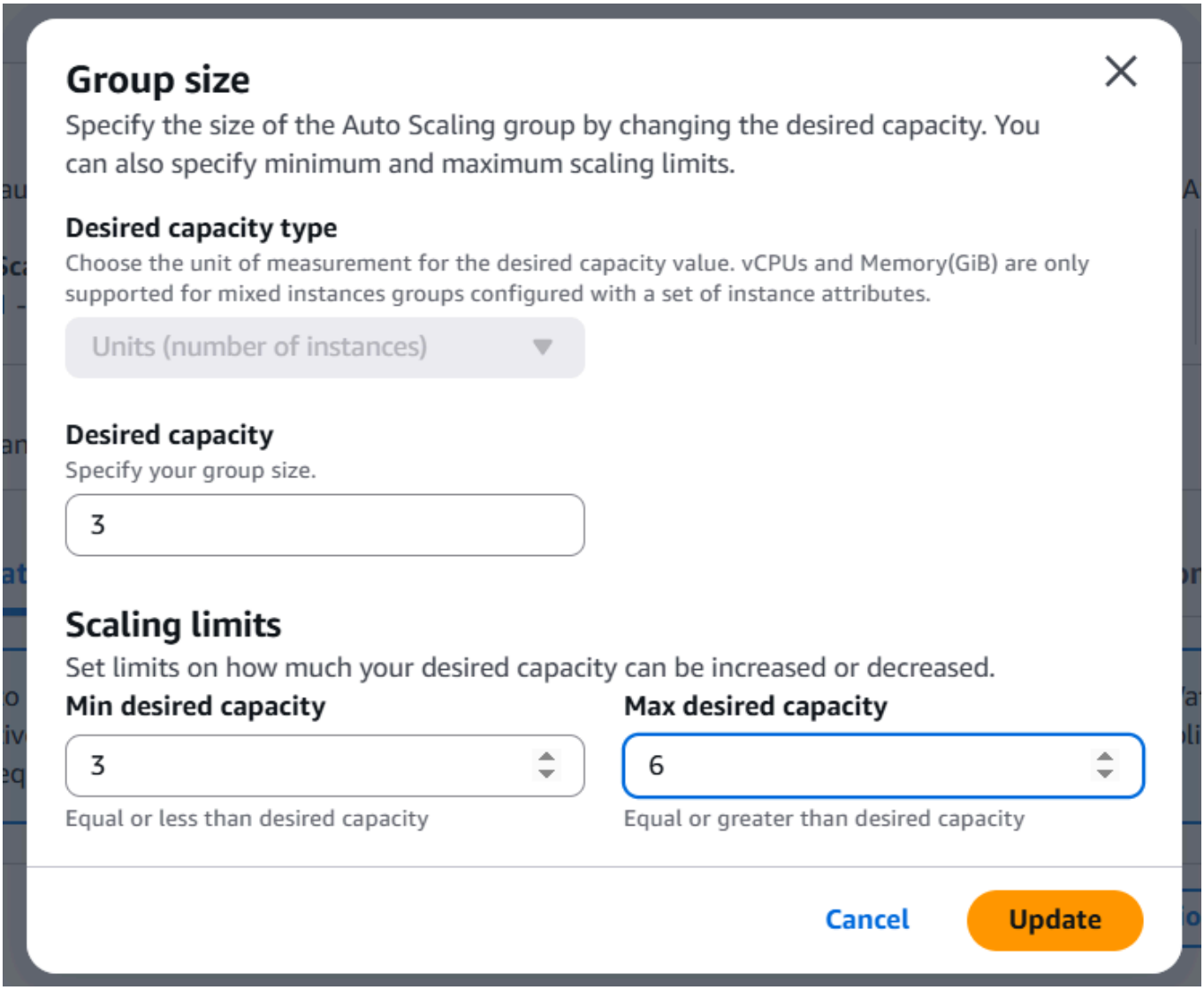
- Click **Add rule**.
- Select `Block[MyCountry]WithCustomResponse`.
- Click **Create web ACL**.
- Now access it again with your **CloudFront DNS**.



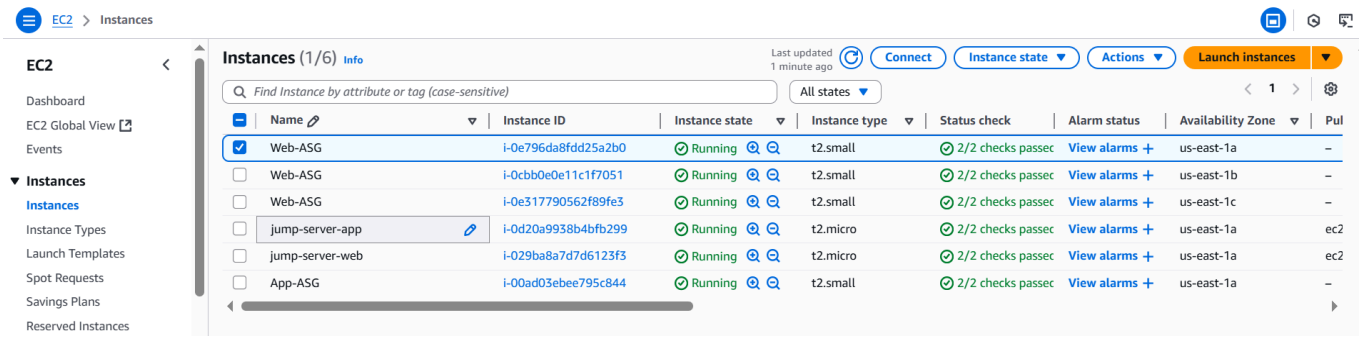
15. Testing Auto Scaling

Now finally, you have to test the **Auto-Scaling** of your application to handle the traffic load.

- Go to both of your **ASGs**.
- Update the **capacity** as:



Now your ASG will create 2 new EC2 instances



Congratulations to you on successfully building this entire project from end-to-end and achieving scalability, fault tolerance, high-availability and security.