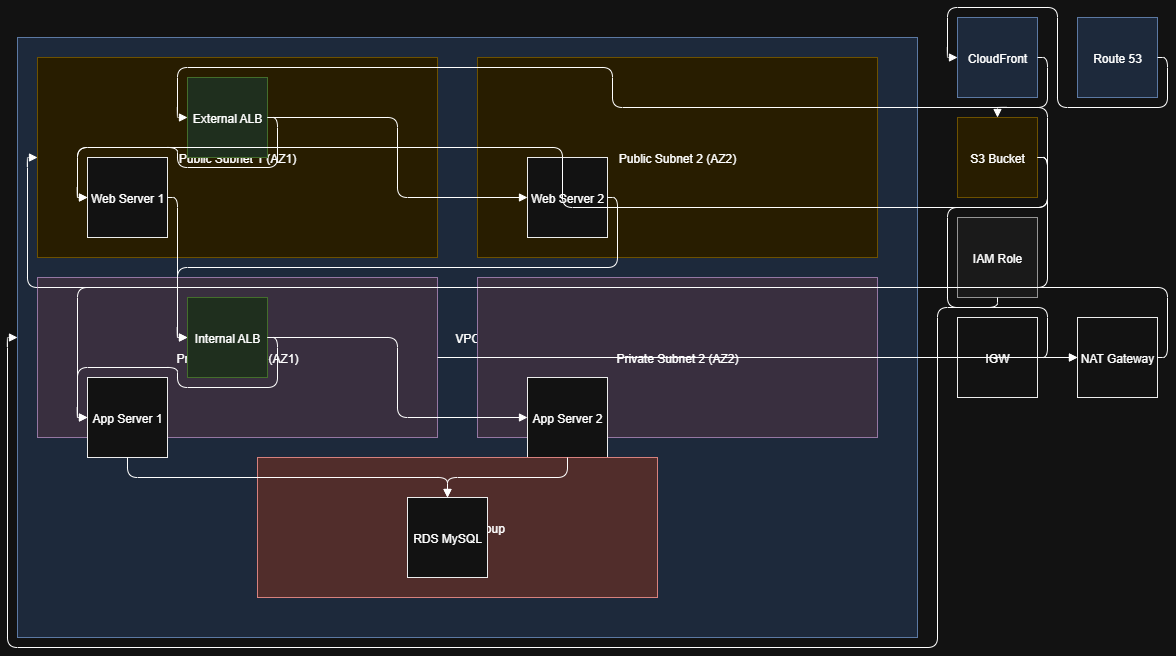
**AWS 3-Tier Project (Webtier,Apptier,Dbtier)**

**Overview**

I successfully designed and deployed a **highly available and secure 3-tier web application architecture** on AWS. This setup ensures scalability, reliability, and security by leveraging multiple AWS services and best practices for infrastructure design.

**Architecture Diagrams**



| **Service / Tool** | **Purpose** |
| --- | --- |
| **VPC (Virtual Private Cloud)** | To create a dedicated, isolated network for the 3-tier architecture |
| **Subnets** | To logically separate **public** and **private** resources (Web / App / DB tiers) |
| **Internet Gateway** | Allows instances in public subnets to communicate with the internet |
| **NAT Gateway** | Allows **private** subnet instances to access the internet **securely** for updates |
| **Route Tables** | Controls routing of traffic between subnets and to/from the internet/NAT |
| **Security Groups** | Acts as a virtual firewall to control inbound/outbound traffic per tier |
| **EC2 Instances** | Hosts the **Web Server** and **Application Server** workloads |
| **Application Load Balancer (External)** | Distributes internet traffic to **Web Tier** (public-facing) |
| **Application Load Balancer (Internal)** | Distributes traffic from Web Tier to **App Tier** (private) |
| **Auto Scaling Group** | Automatically scales EC2 instances based on demand (both Web & App tiers) |
| **RDS (MySQL)** | Managed relational database in the **Database Tier** |
| **IAM Role + S3 ReadOnly Policy** | Allows EC2 instances to securely pull code from S3 buckets |
| **Elastic IP** | Provides a fixed public IP for NAT Gateway (used in Private subnets) |
| **CloudFront (CDN)** | Caches and delivers web content with low latency and **SSL termination** |
| **ACM (AWS Certificate Manager)** | Provides the **SSL certificate** used by CloudFront for HTTPS access |
| **Route 53 (DNS Service)** | Manages domain names and routes user traffic to **CloudFront / Load Balancer** with DNS mapping |

🡪Clone the code from Repo

Download the code from [this repository](https://github.com/aws-samples/aws-three-tier-web-architecture-workshop) into your local environment by running the command below. If you don't have git installed, you can just download the zip. Save it somewhere you can easily access.

From <<https://catalog.us-east-1.prod.workshops.aws/workshops/85cd2bb2-7f79-4e96-bdee-8078e469752a/en-US/part0/code>>

Create Folder Named : AWS-Project

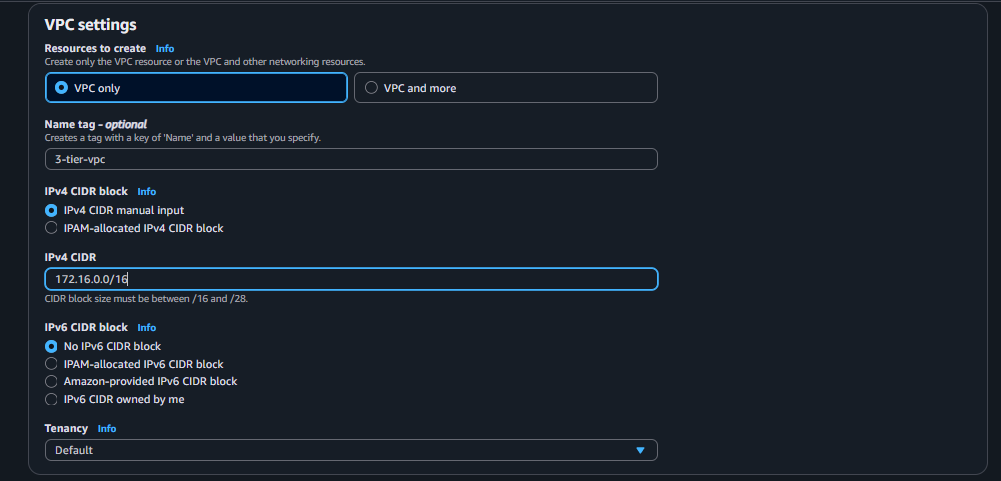
Open the folder and Open gitbash or command prompt for this location (if you are not install gitbash install : [[https://git-scm.com/downloads](https://git-scm.com/downloads)](https://git-scm.com/downloads%5d(https:/git-scm.com/downloads)))

And clone the code from remote by ref below url :

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

**Step 1 – Create a VPC (Virtual Private Cloud)**

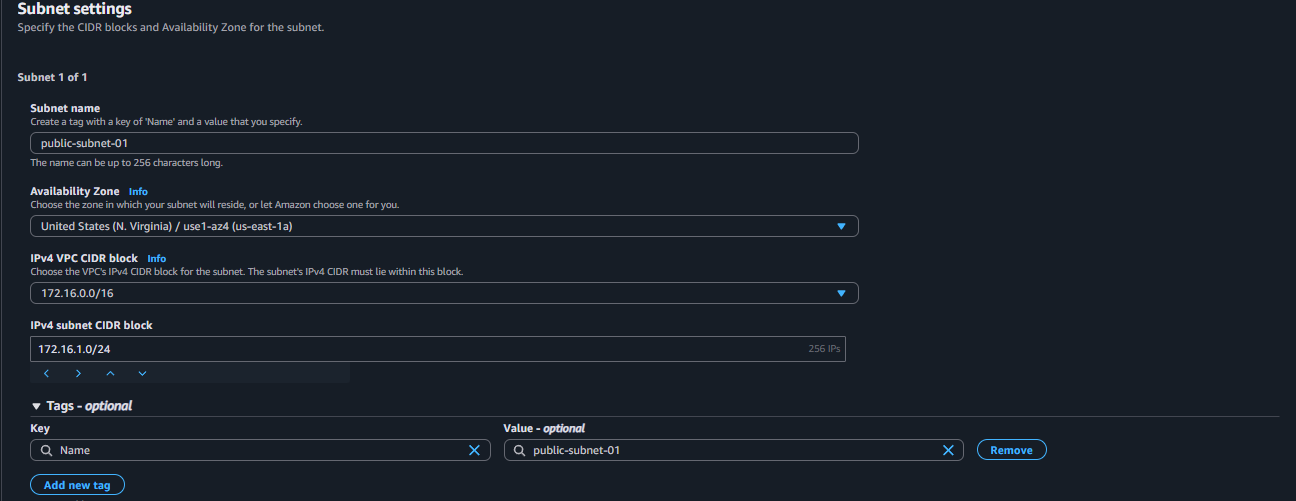
1. In the AWS Console, search for **VPC** and open the VPC Dashboard.
2. Select **Create VPC** and choose *VPC only* option.
3. Provide a name (e.g., “3‑tier‑vpc”) and CIDR block (e.g., 172.16.0.0/16), then click **Create VPC**.



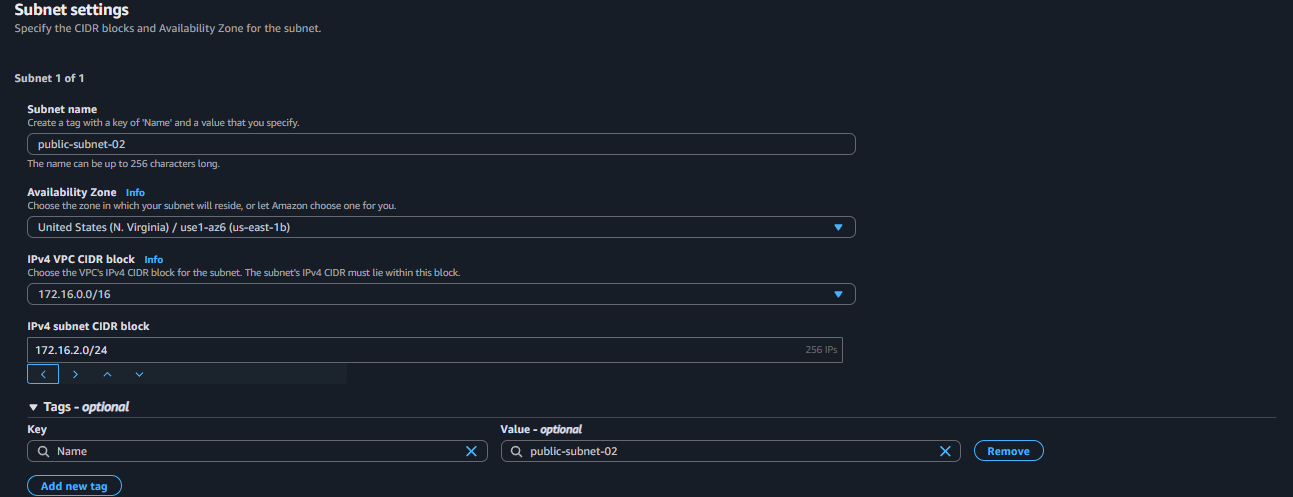
**Step 2 – Create Subnets**

1. In the VPC Dashboard, select **Subnets** and choose **Create subnet**.
2. Add 2 public subnets (e.g., 172.16.1.0/24 and 172.16.2.0/24) and 2 private subnets (e.g., 172.16.3.0/24 and 172.16.4.0/24).
3. Ensure the subnets are spread across at least 2 Availability Zones.
   * Select your custom VPPC

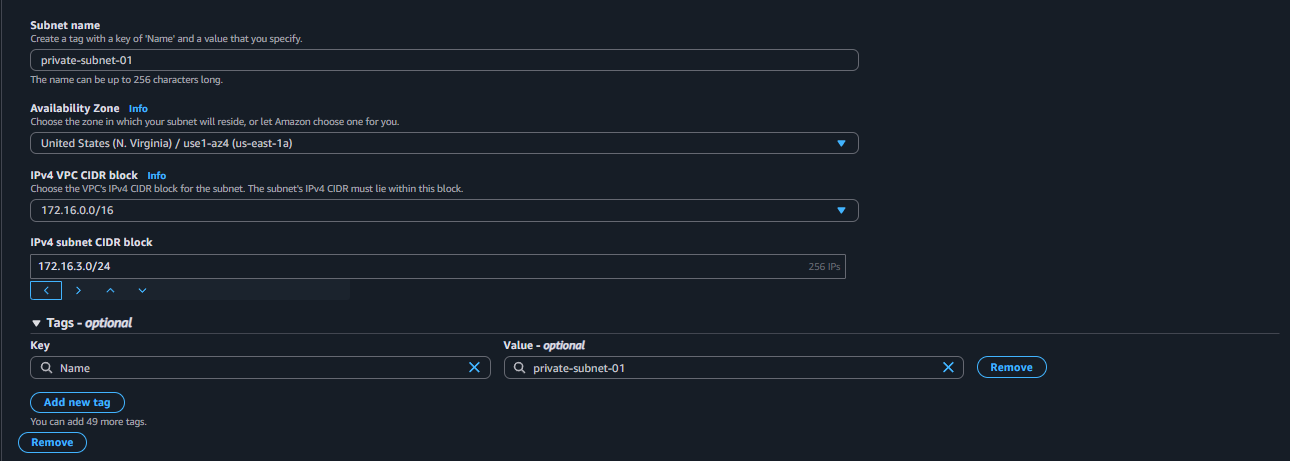


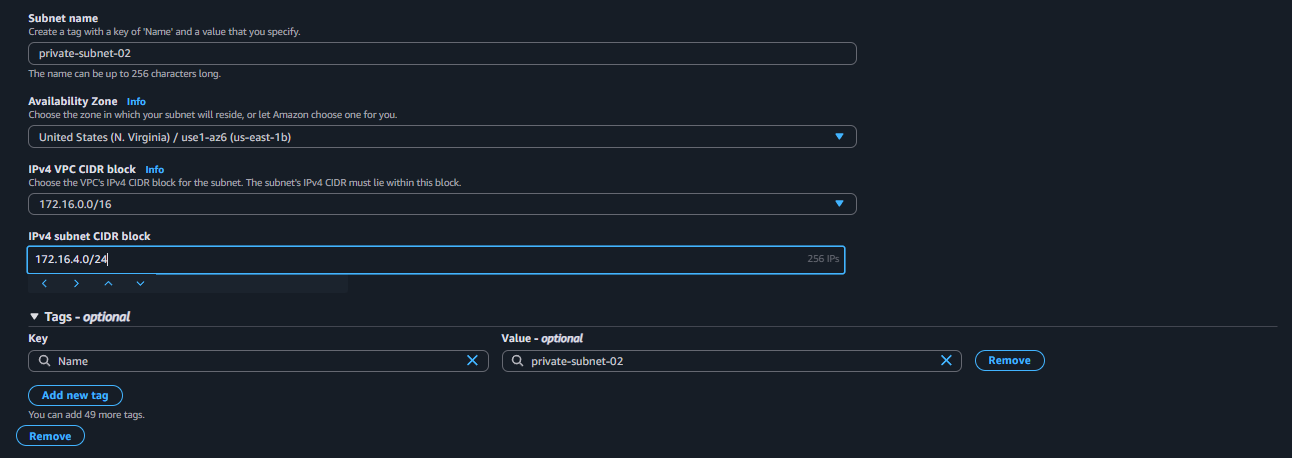
**Subnet-1🡪Public-subnet**

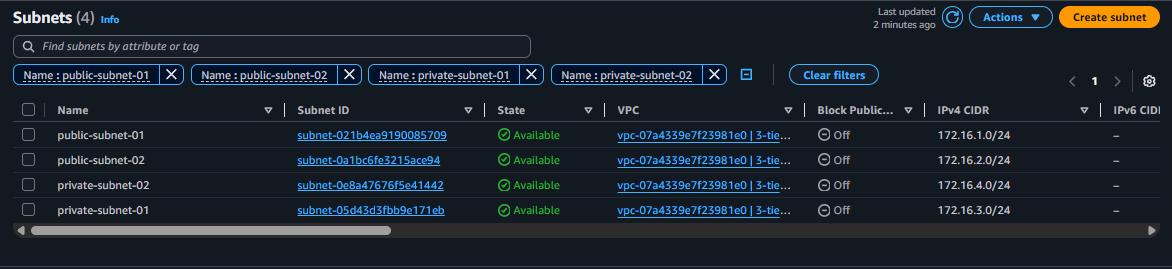
**Subnet-2🡪Public-subnet**



**Subnet-3🡪Private-subnet**

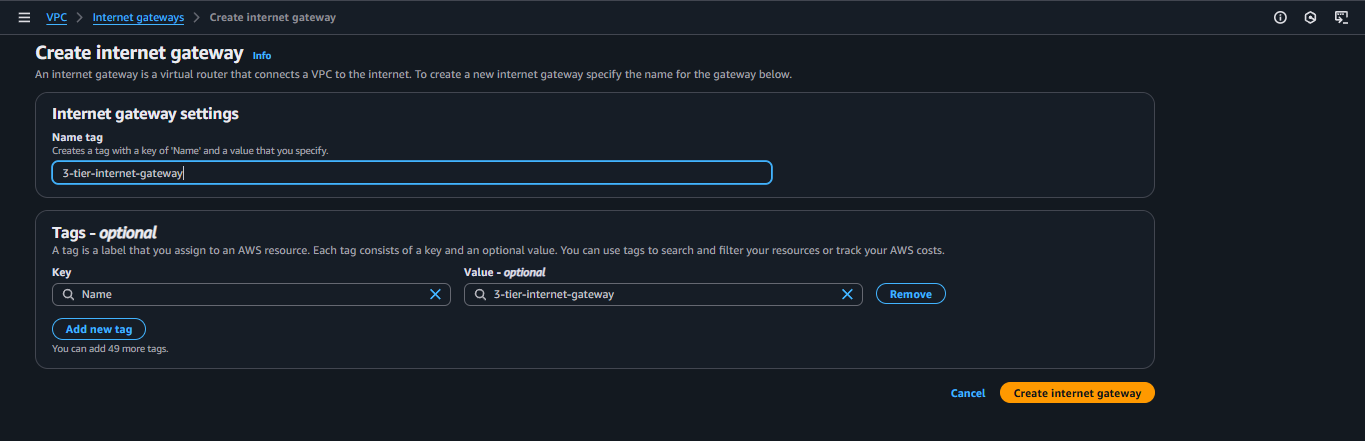


**Subnet-4🡪Private-subnet -** 

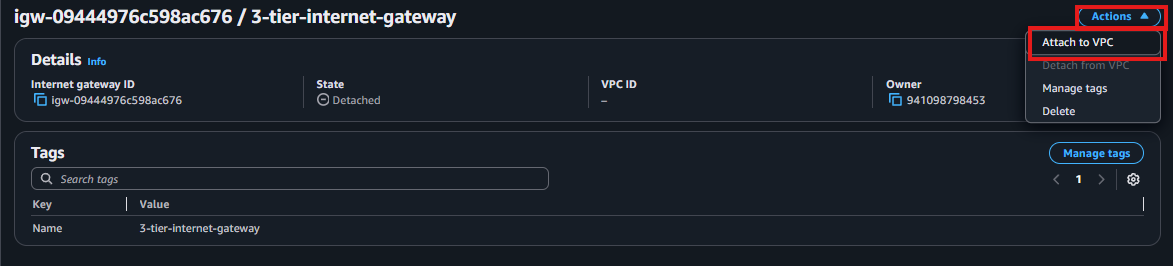


**Step 3 – Configure Internet Gateway**

1. In the VPC Dashboard, select **Internet Gateways** and create a new Internet Gateway.



1. Attach the Internet Gateway to your VPC.
   * In the action button select attach to VPC



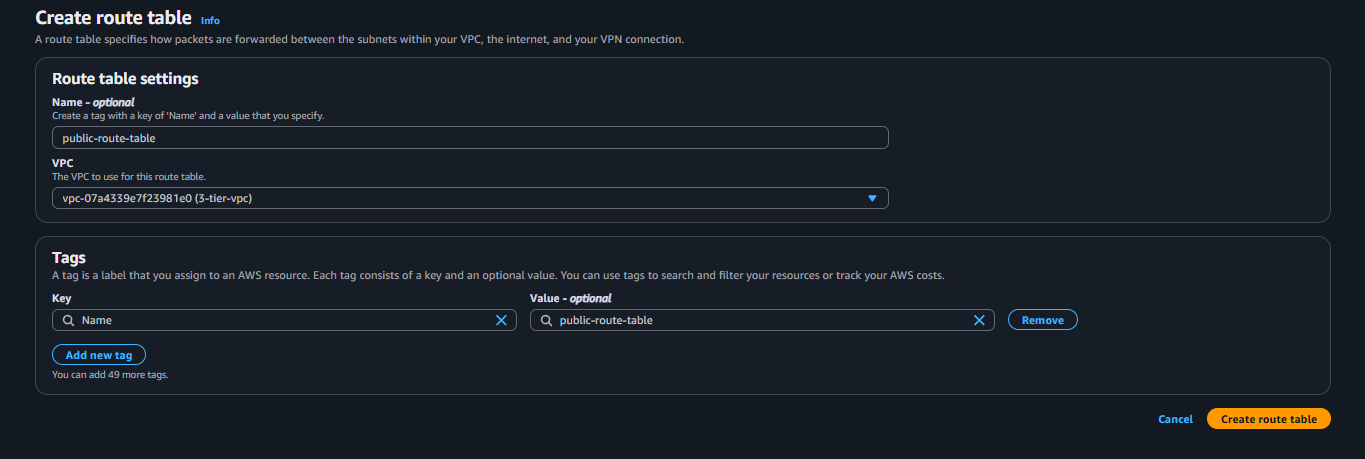
1. Select your custom VPC then click on “Attach internet gateway”

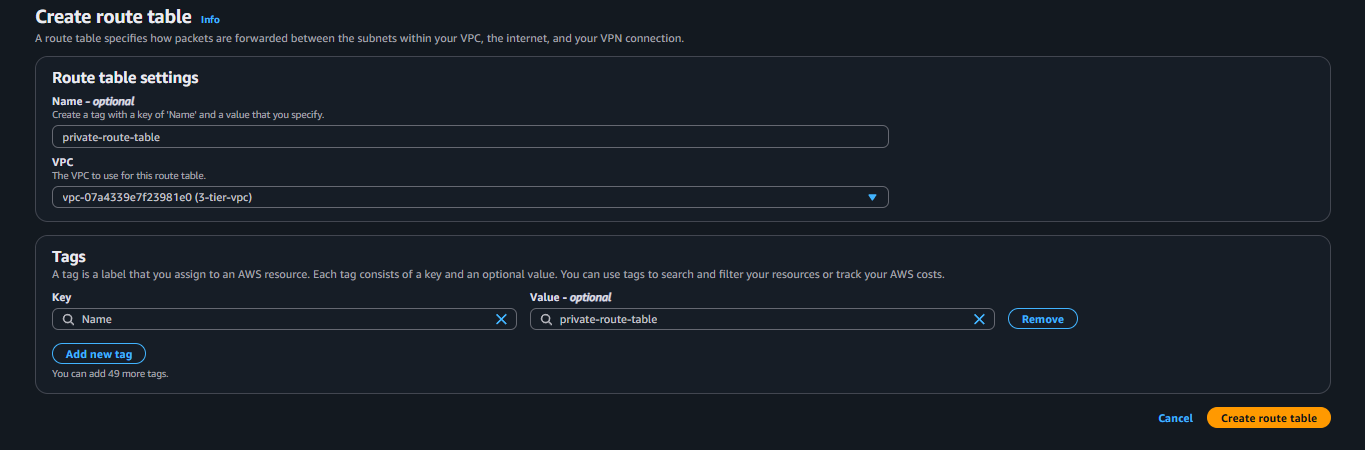


**Step 4 – Configure Route Tables**

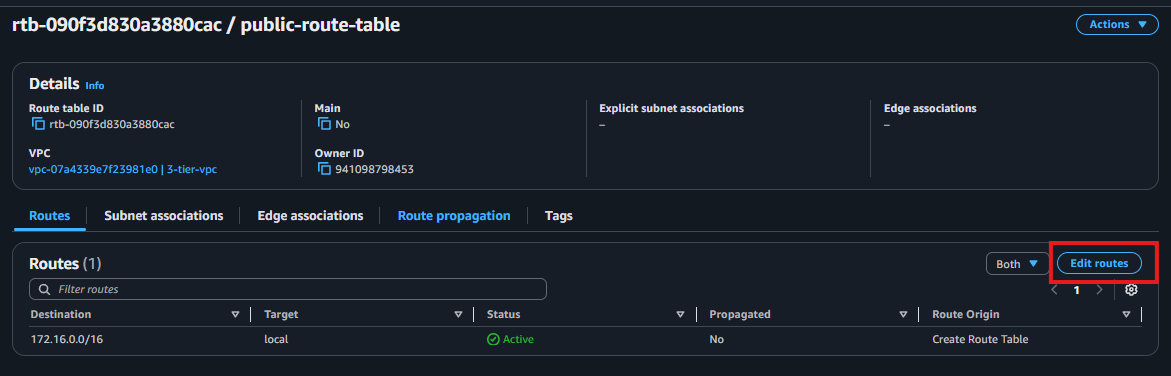
1. In **Route Tables**, create one route table for public subnets and another for private subnets.

**Public-Route-Table:**

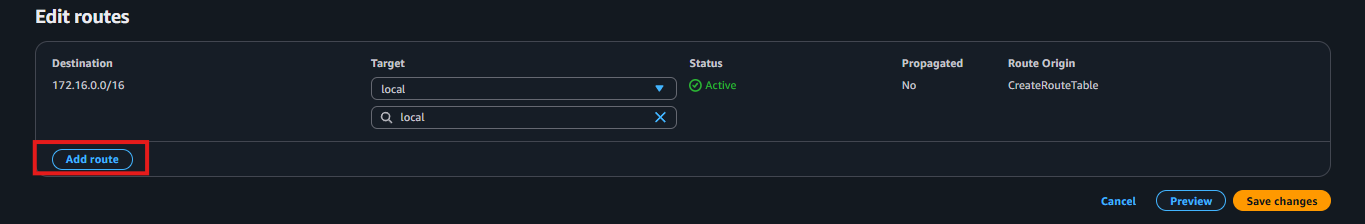


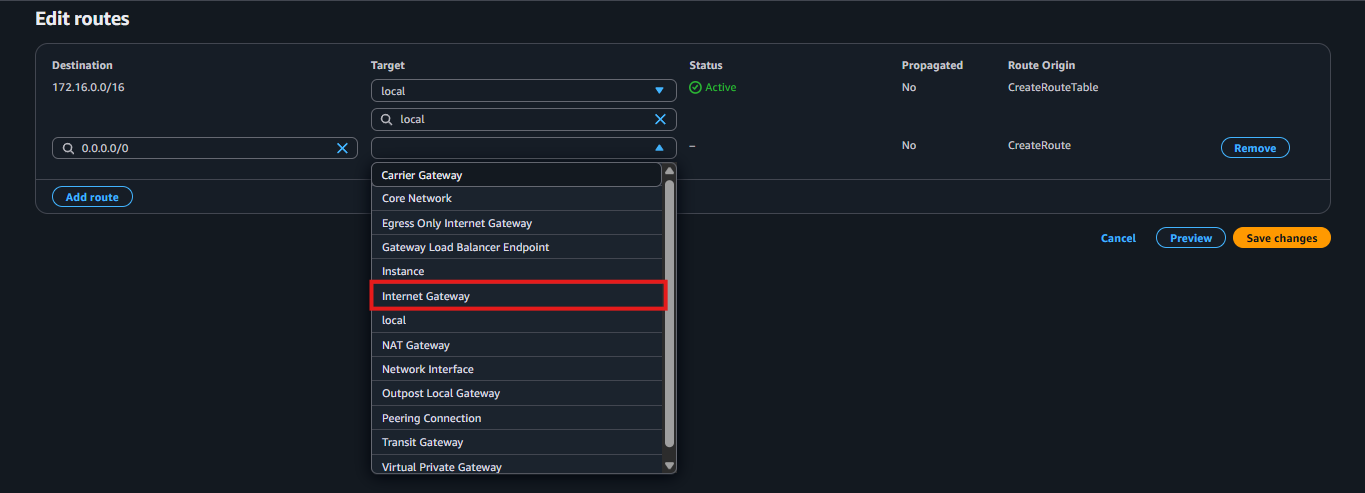
**Private-Route-Table:** 

1. Add a route (0.0.0.0/0) to the public route table and target the Internet Gateway. Click on “Edit routes”

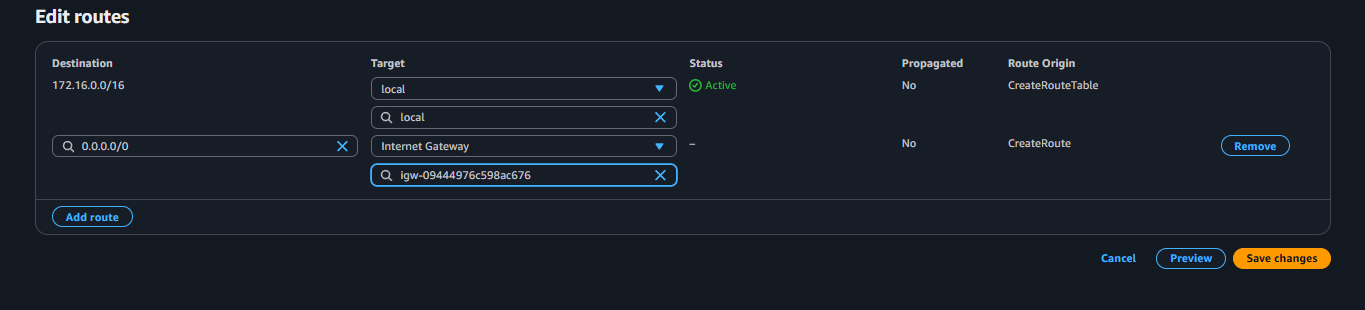


* + Click on “Add route” button

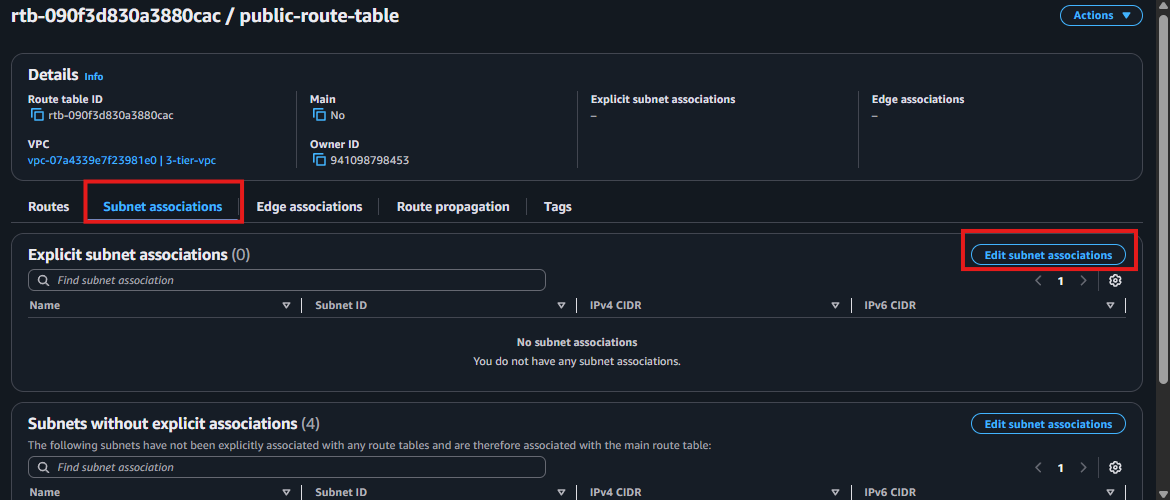




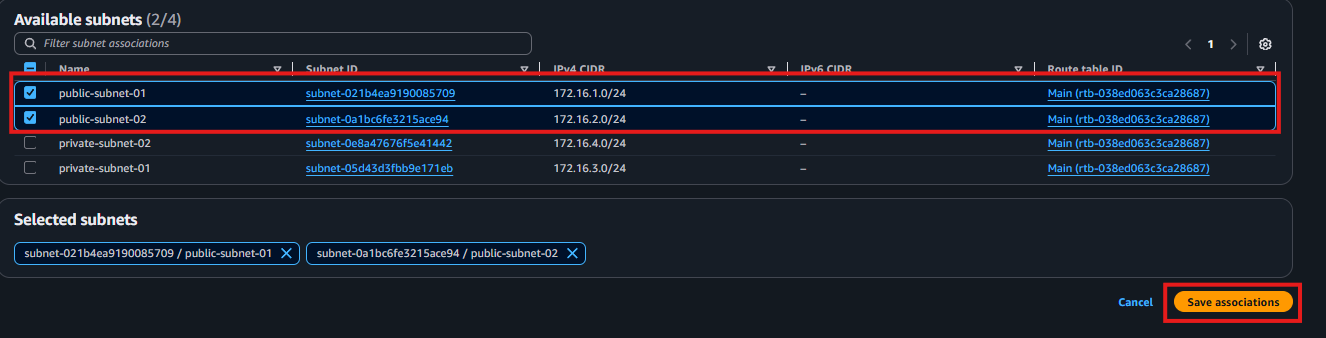
* + Select your internet gateway and “Save changes”



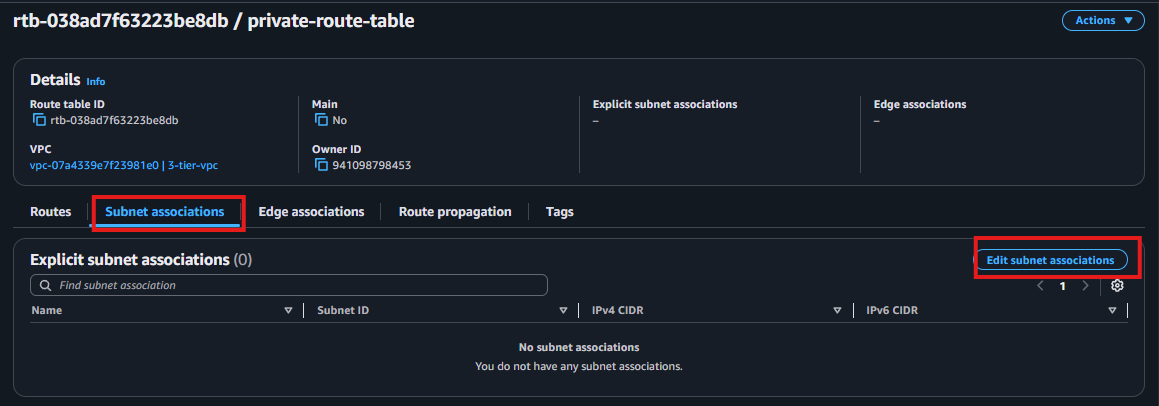
1. Associate public subnets with the public route table.
   * Click “Subnet associations” then click “Edit subnet associations”



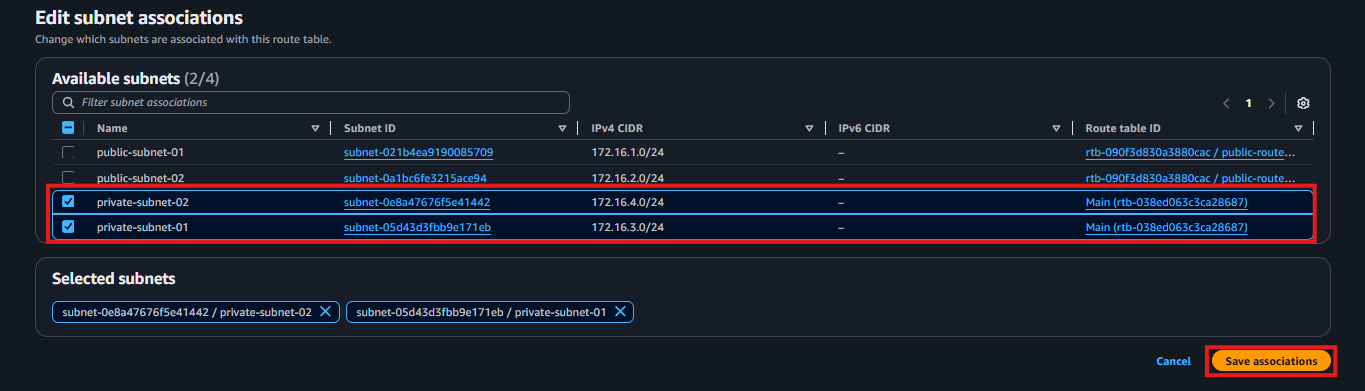
* + Select the public subnets and click on “Save associations”



1. Associate private subnets with the private route table.
   * Click “Subnet associations” then click “Edit subnet associations”



* + Select the private subnets and click on “Save associations”

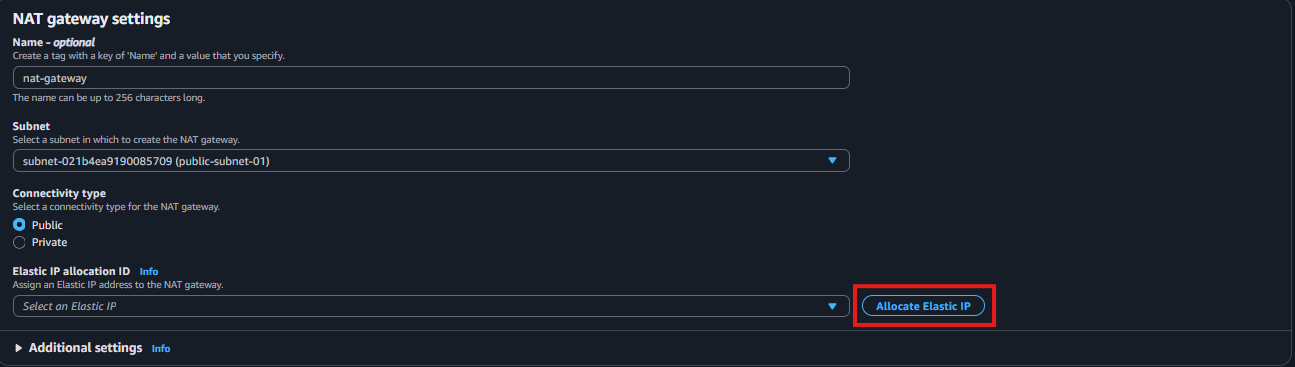


**Step 4a – Create NAT Gateway**

1. In the VPC Dashboard, select **NAT Gateways** and click **Create NAT Gateway**.
2. Select one of the public subnets and associate an Elastic IP address.

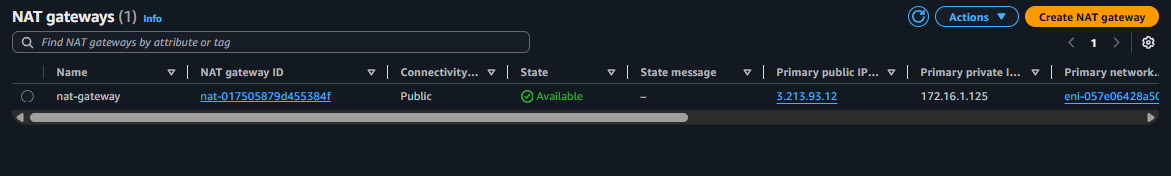


* + Then click on “Allocate Elastic IP”

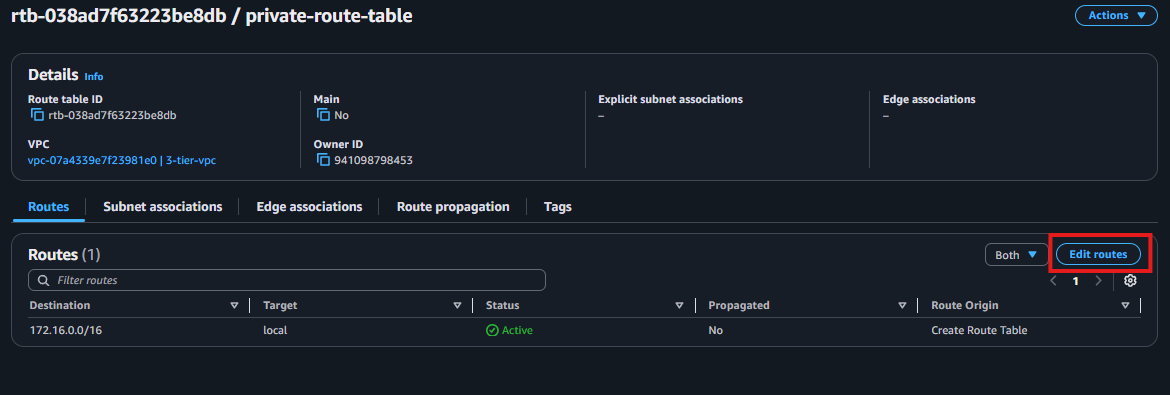


* + And click the “Save changes”

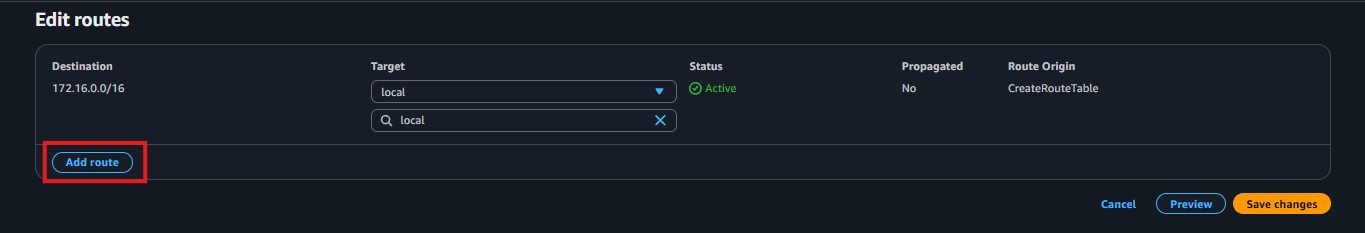
1. Create the NAT Gateway and wait until it becomes available.

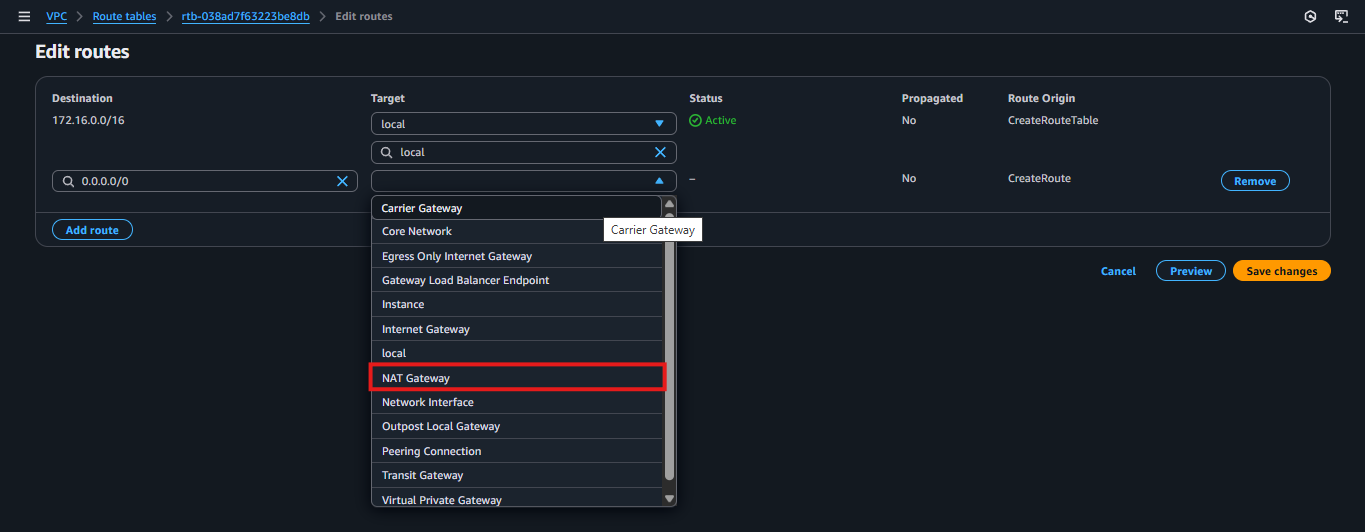


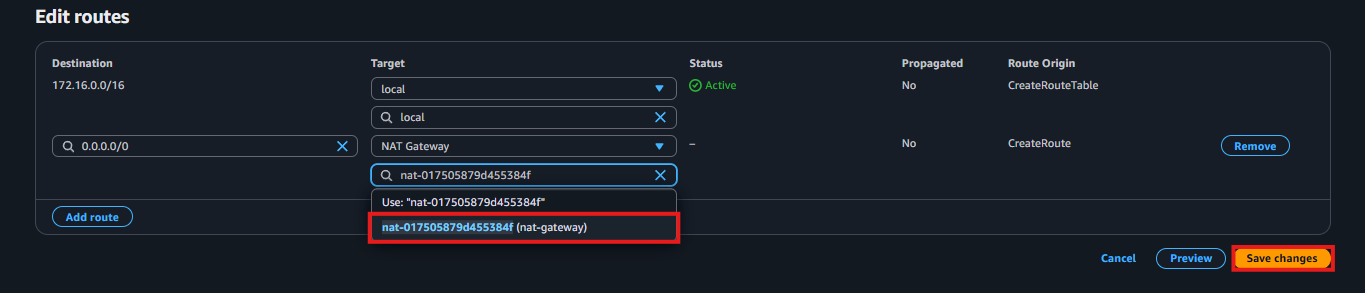
1. In the private route table, add a route (0.0.0.0/0) and target the new NAT Gateway.



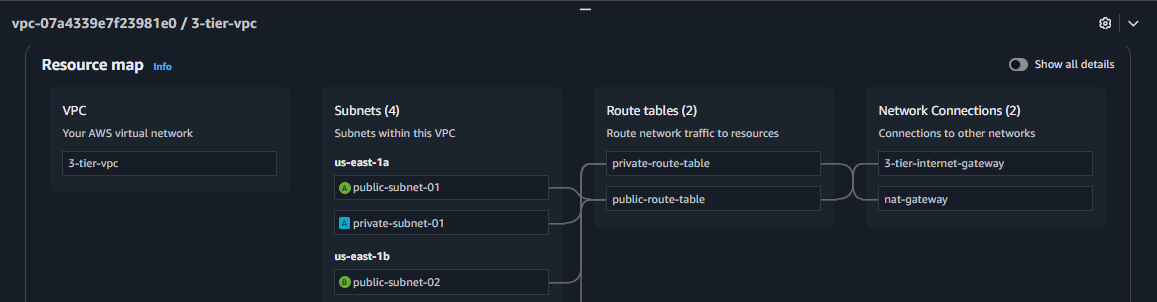
* + Click on “Add route” and select “NAT Gateway” then in the drop down select the NAT we have just created. And enter the “Save changes”







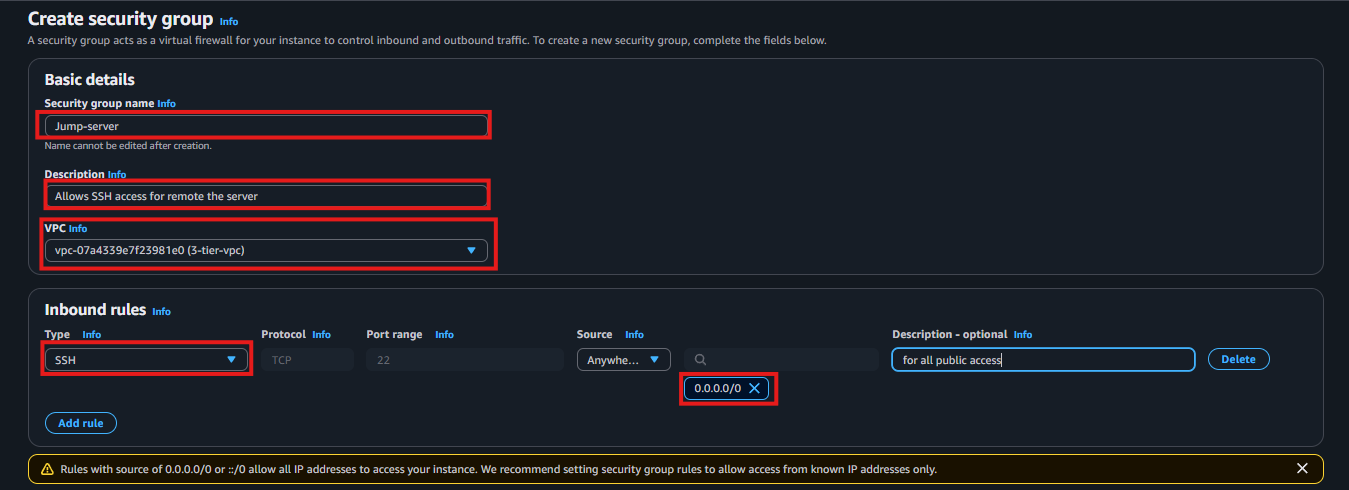
* + Finally your VPC 🡪 **Resource map** look like this we are done with “Network part”.



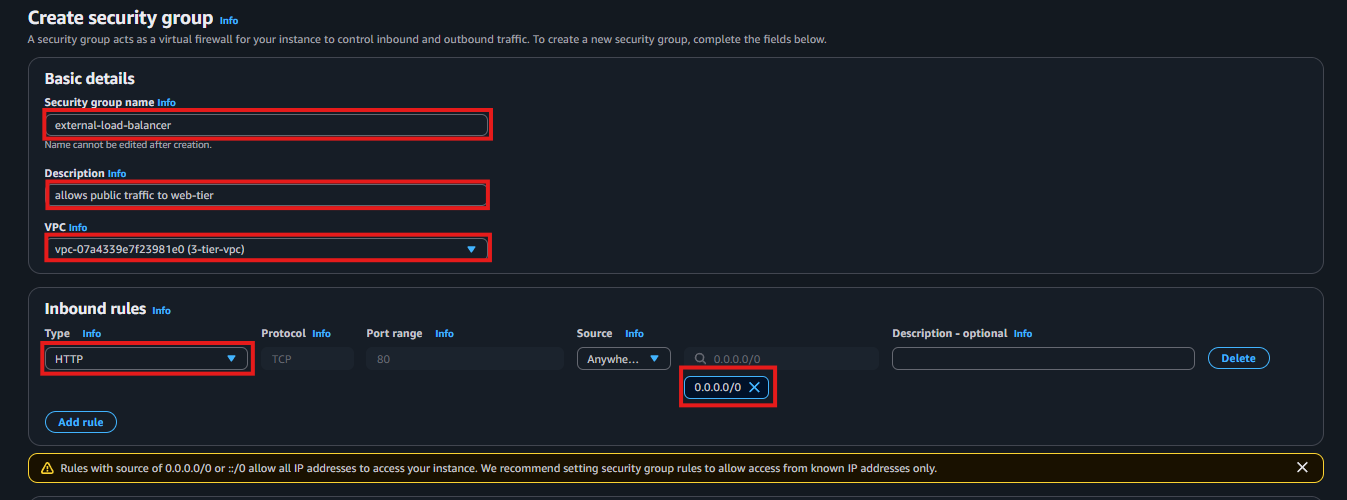
**Step 5 – Create Security Groups**

Create the following six custom security groups:

1. Jump Server SG – allows SSH (port 22) from all public IP addresses.



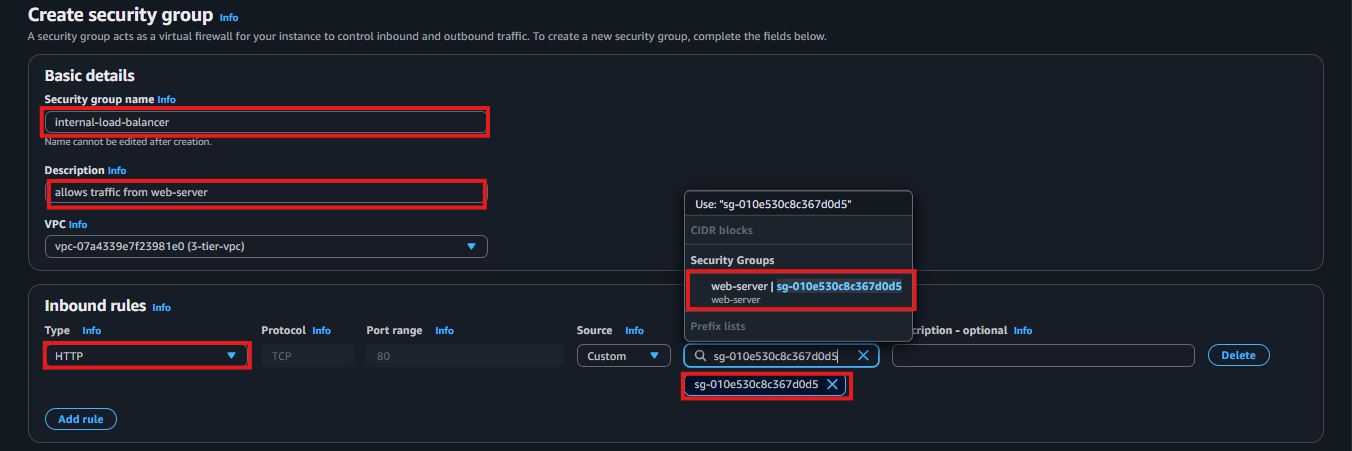
1. External Load Balancer SG – allows HTTP (port 80) from all public IP addresses.



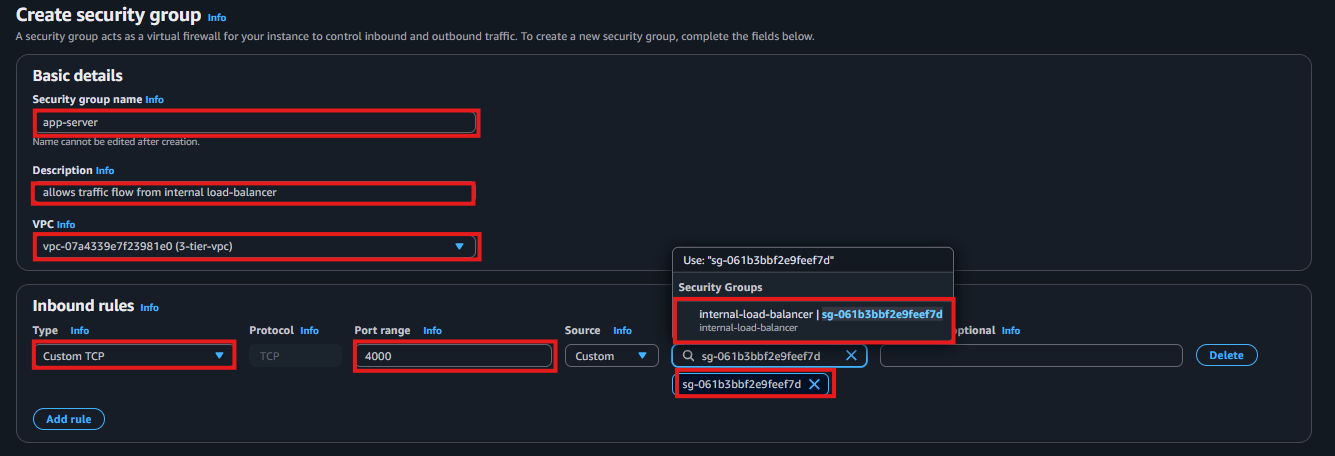
1. Web Server SG – allows HTTP (port 80) from the External Load Balancer SG and SSH (port 22) from the Jump Server SG.

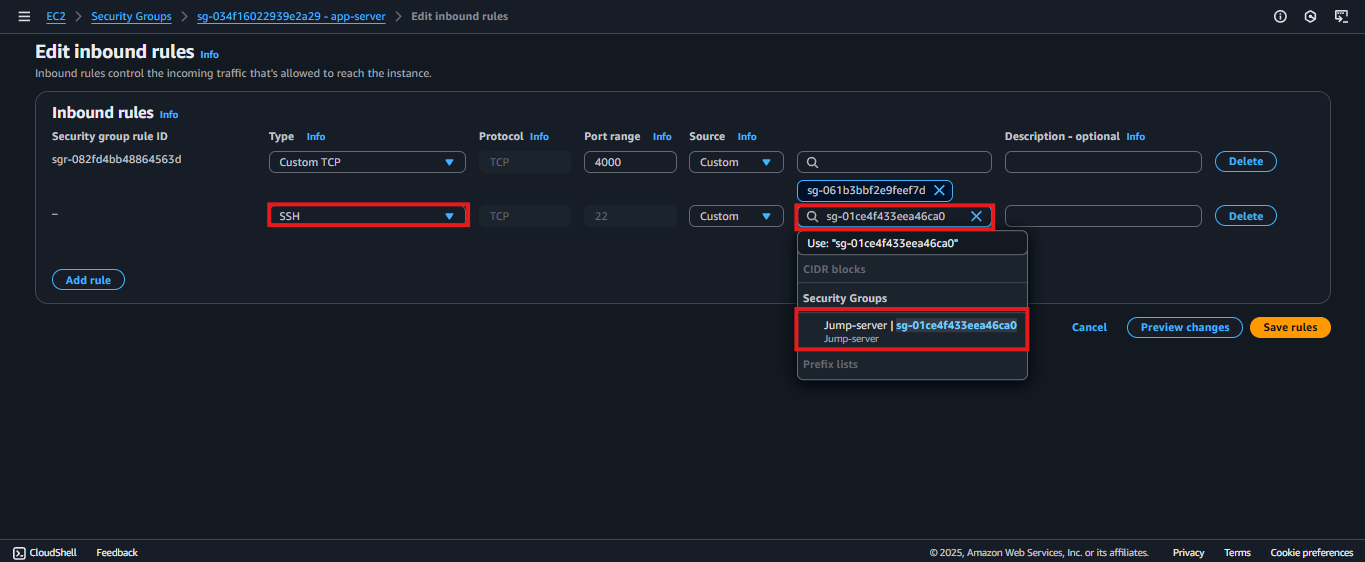


1. Internal Load Balancer SG – allows HTTP (port 80) from the Web Server SG.

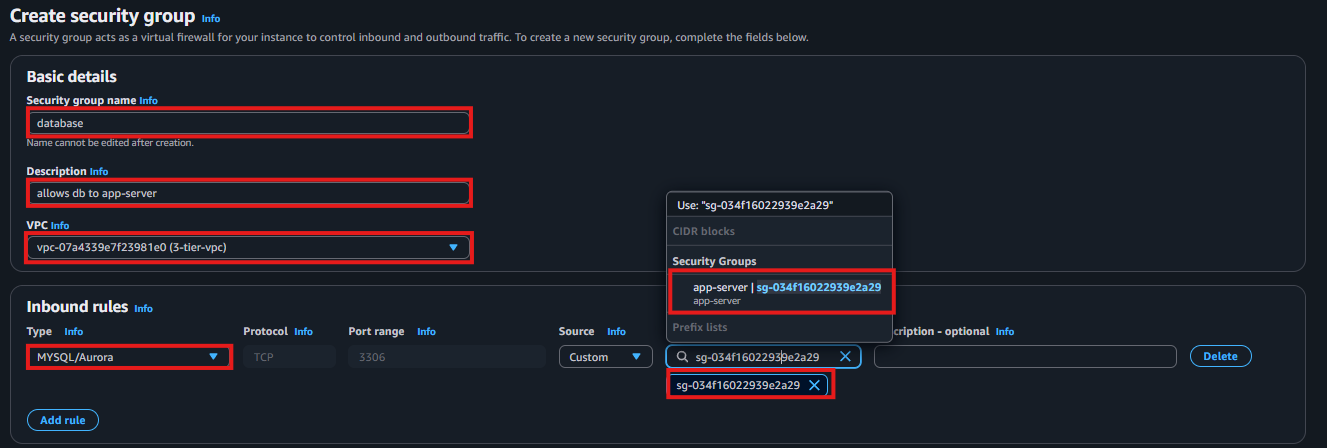


1. App Server SG – allows port 4000 from the Internal Load Balancer SG. And Allow port 22 from the Jump-server SG for remote access from jump-server to app-server



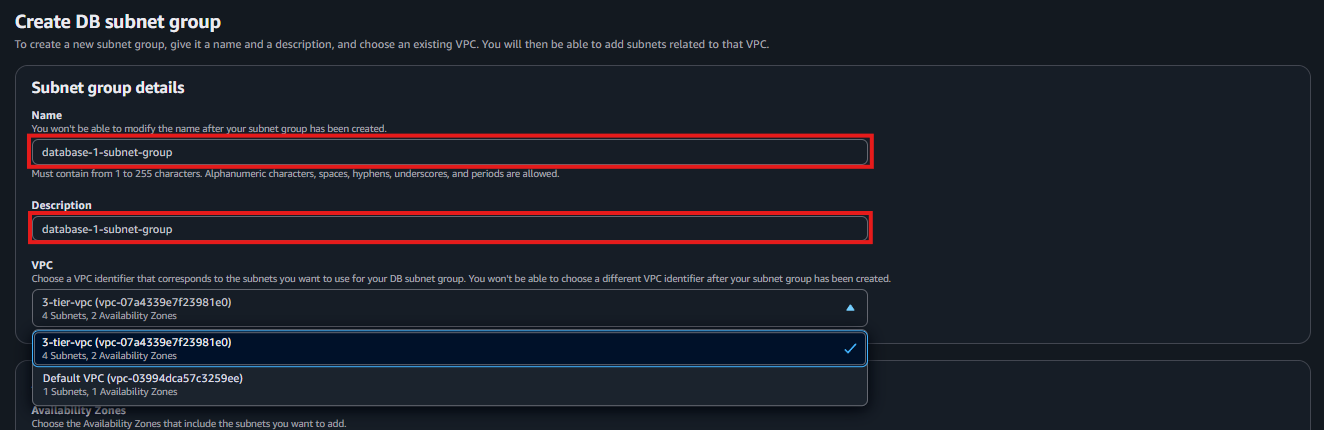


1. Database SG – allows MySQL (port 3306) from the App Server SG

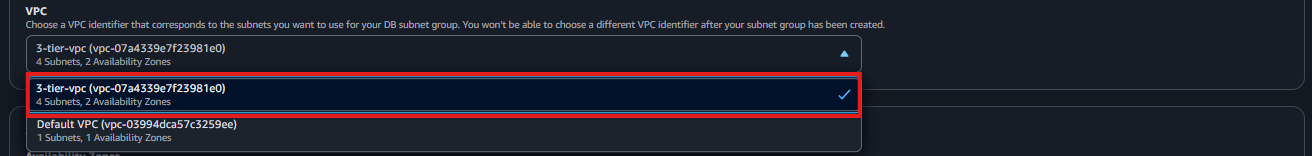


**Step 6 – Create RDS Subnet Group**

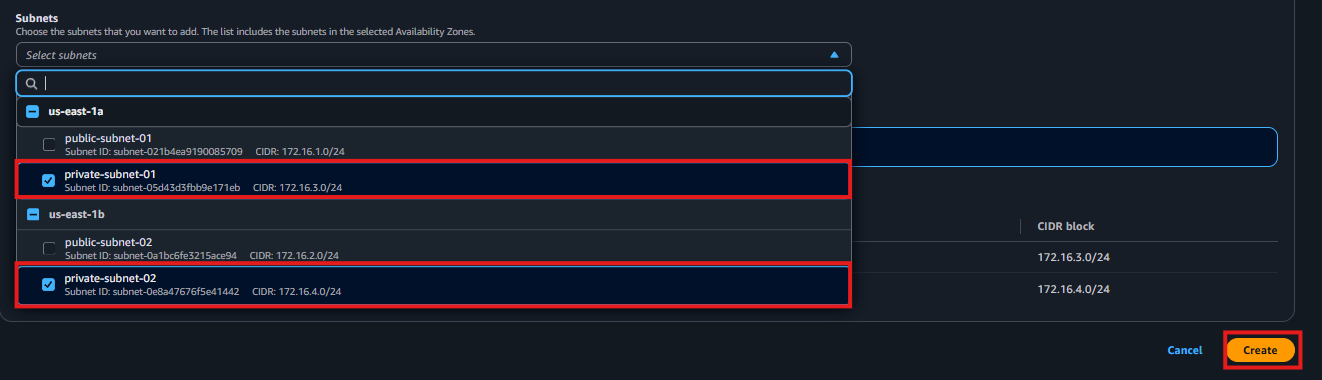
1. In the RDS dashboard, select Subnet groups and click Create DB subnet group.
2. Provide a name and description.

****

1. Select your VPC and choose the AZ (Availability Zone) where the private subnets are created. Then add the private subnets that you created earlier. Finally click on “Create”.

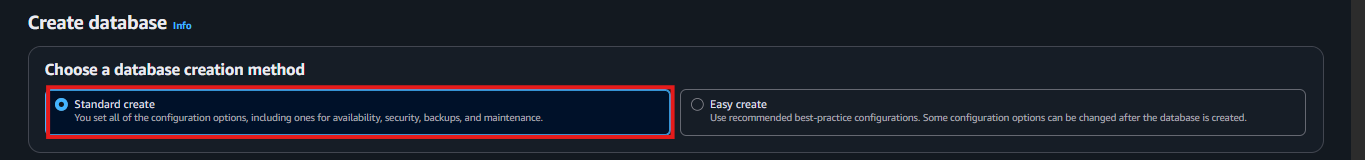
****

****

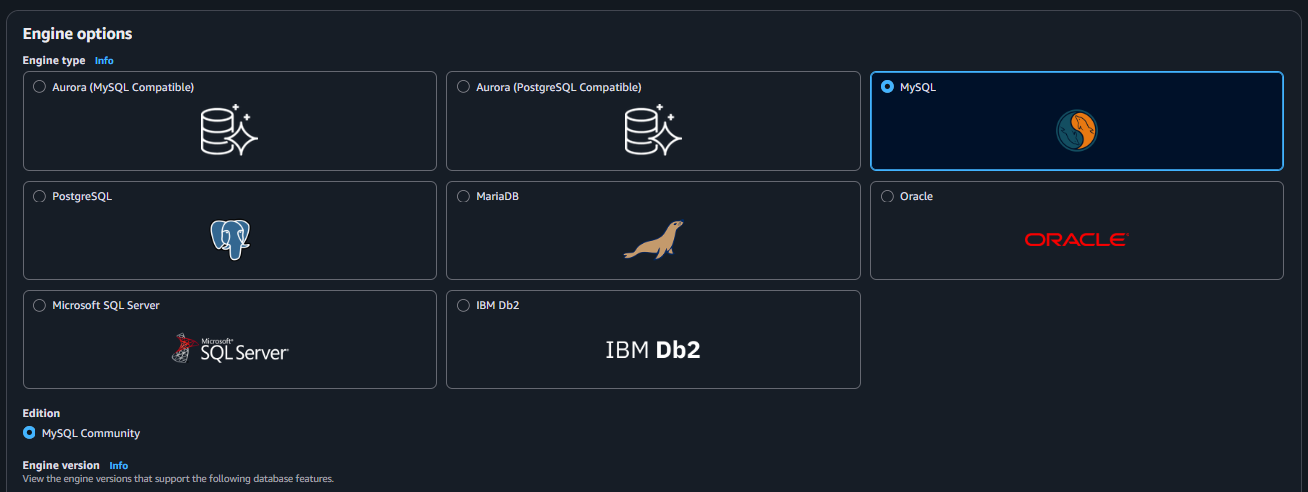
****

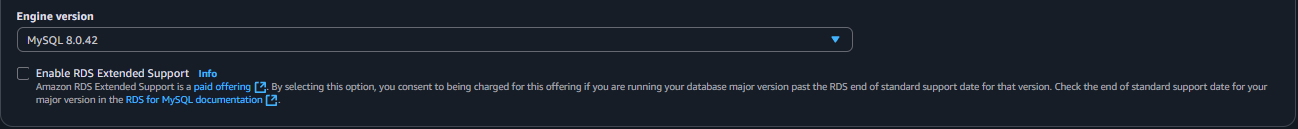
**Step 6a – Configure RDS Database**

1. In the AWS Console, search for **RDS** and select **Create Database**.
2. Choose Standard Create and select the **Free Tier** options.

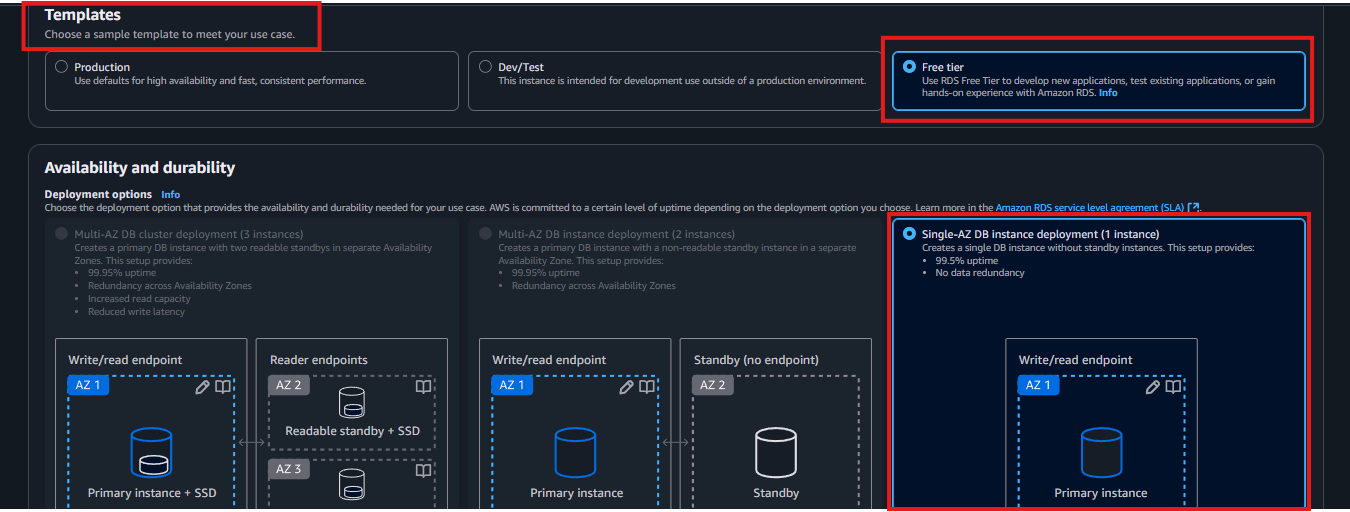


1. Choose **MySQL** as the engine type

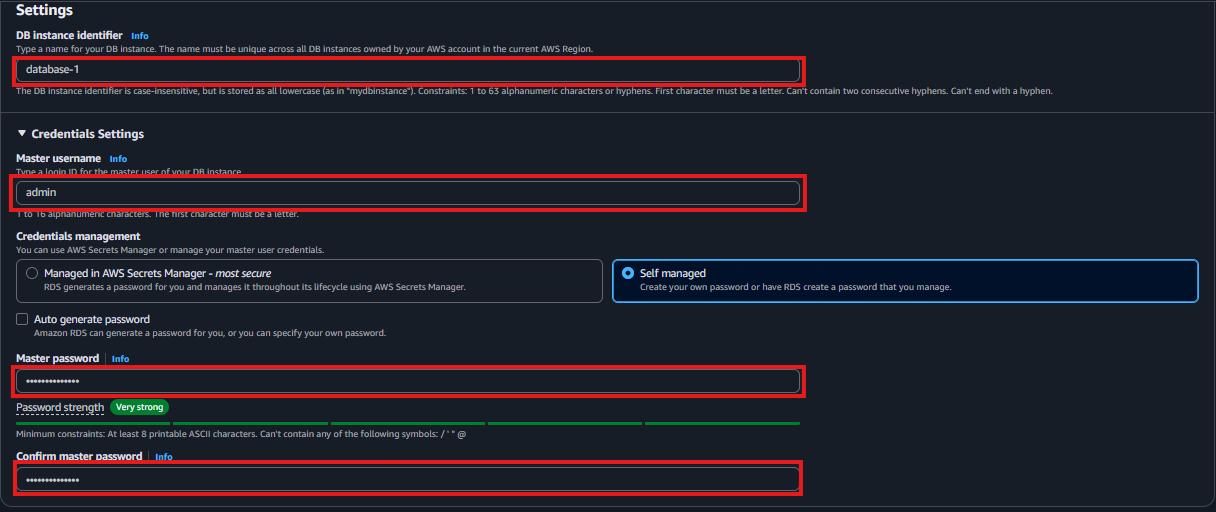




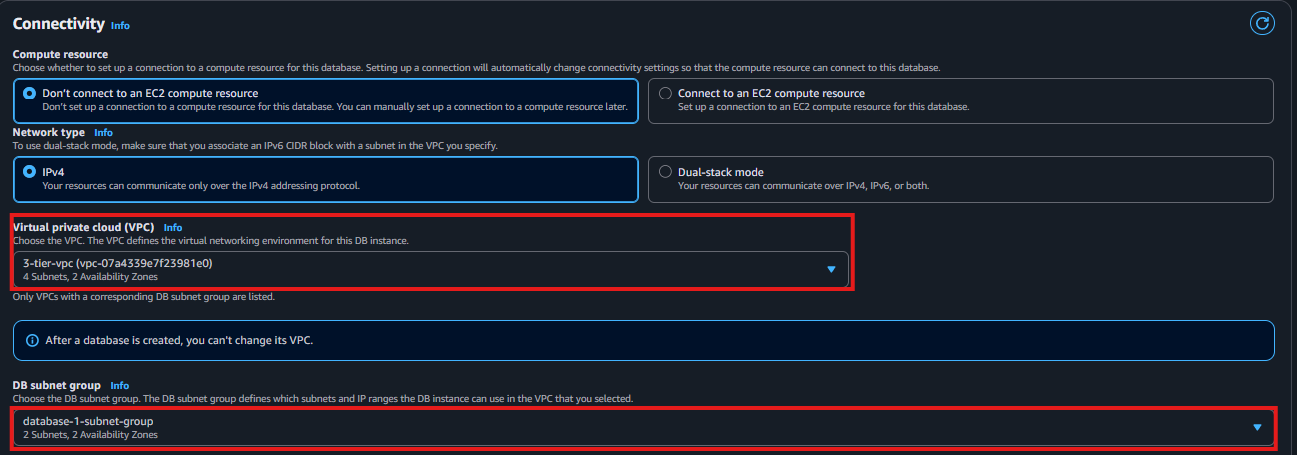
* + In the “Templates” chose free tier.



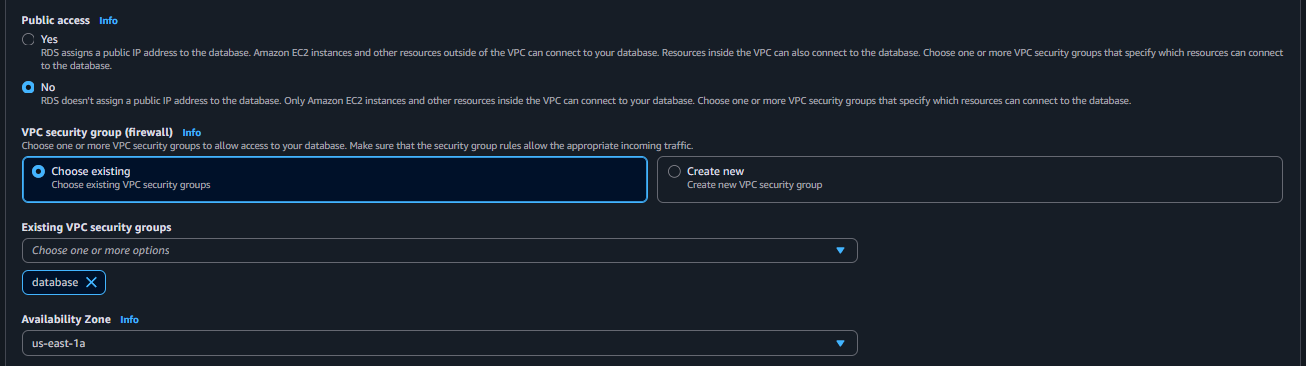
* + configure DB name, username, and password.



* + Choose custom VPC and select the DB subnet group that we created early.



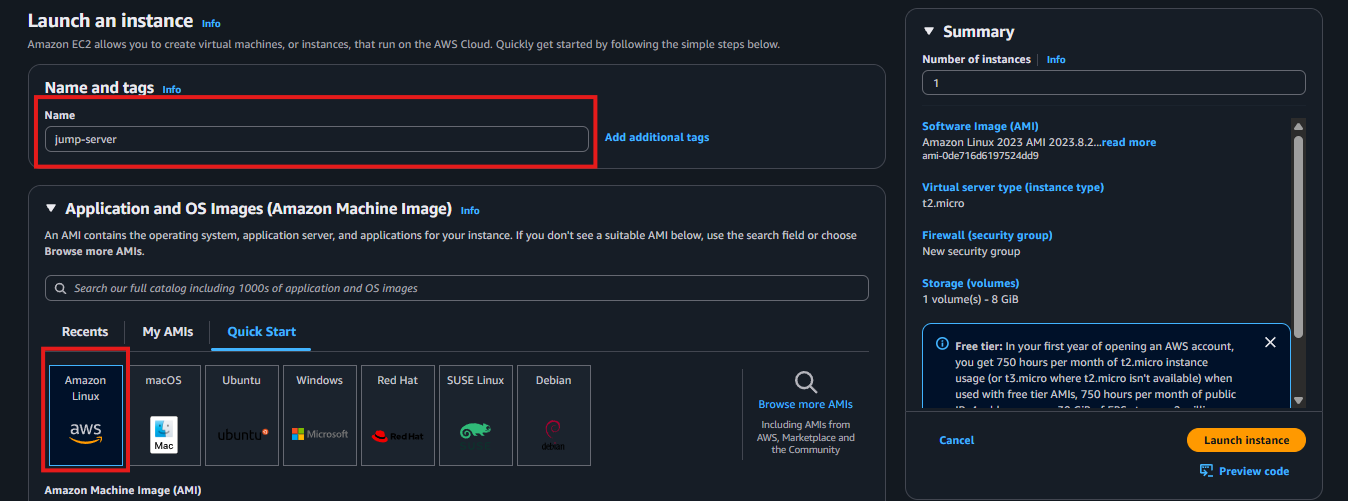
1. Choose the security group for the database allowing access only from the application tier security group. and disable public access, choose the AZ (availability zone).

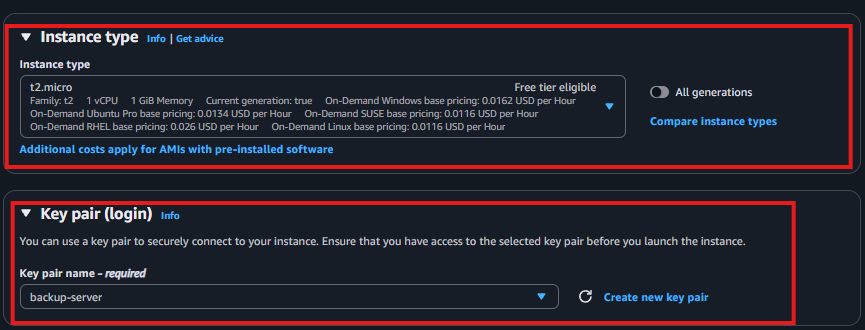


* + Then Create database.

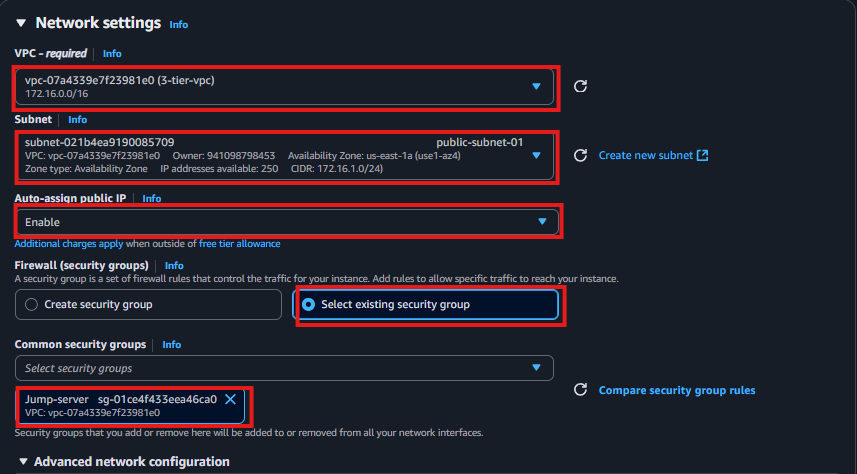
**Step 7 – Launch Jump Server (Bastion Host)**

1. Navigate to EC2 and click Launch Instance.
2. Choose an Amazon Linux 2 AMI and t2.micro instance type.

****

****

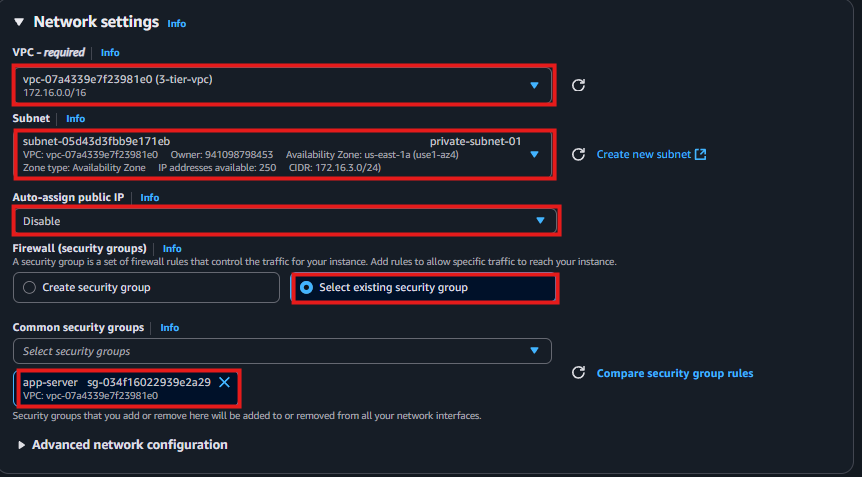
1. Select a public subnet and assign the Jump Server SG security groups

****

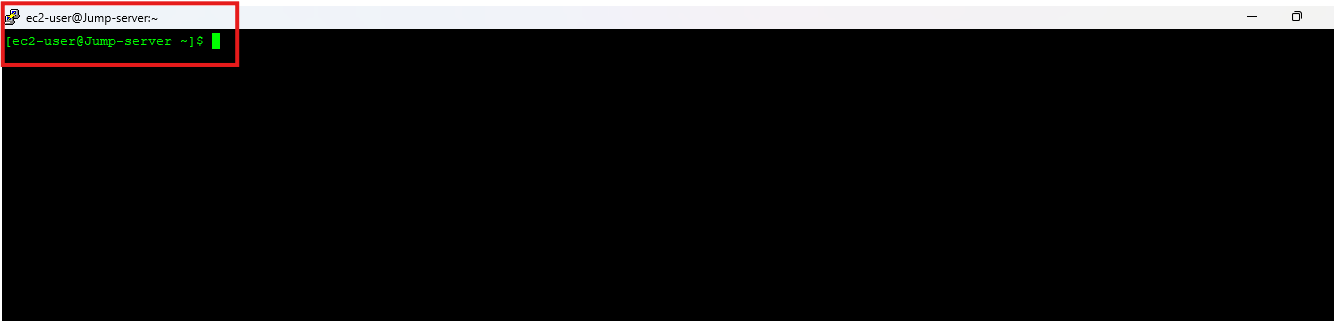
1. Launch the instance and note the public IP address (used for SSH to other private servers).

**Step 8 – Launch EC2 Instances (Application Tier)**

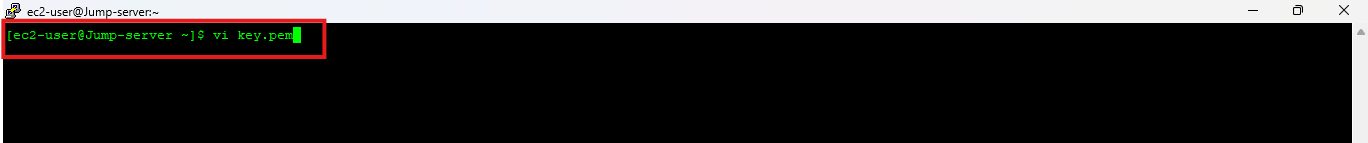
1. Repeat the steps for launching new EC2 instances, but select the private subnets. And choose the security group that we have created early.

****

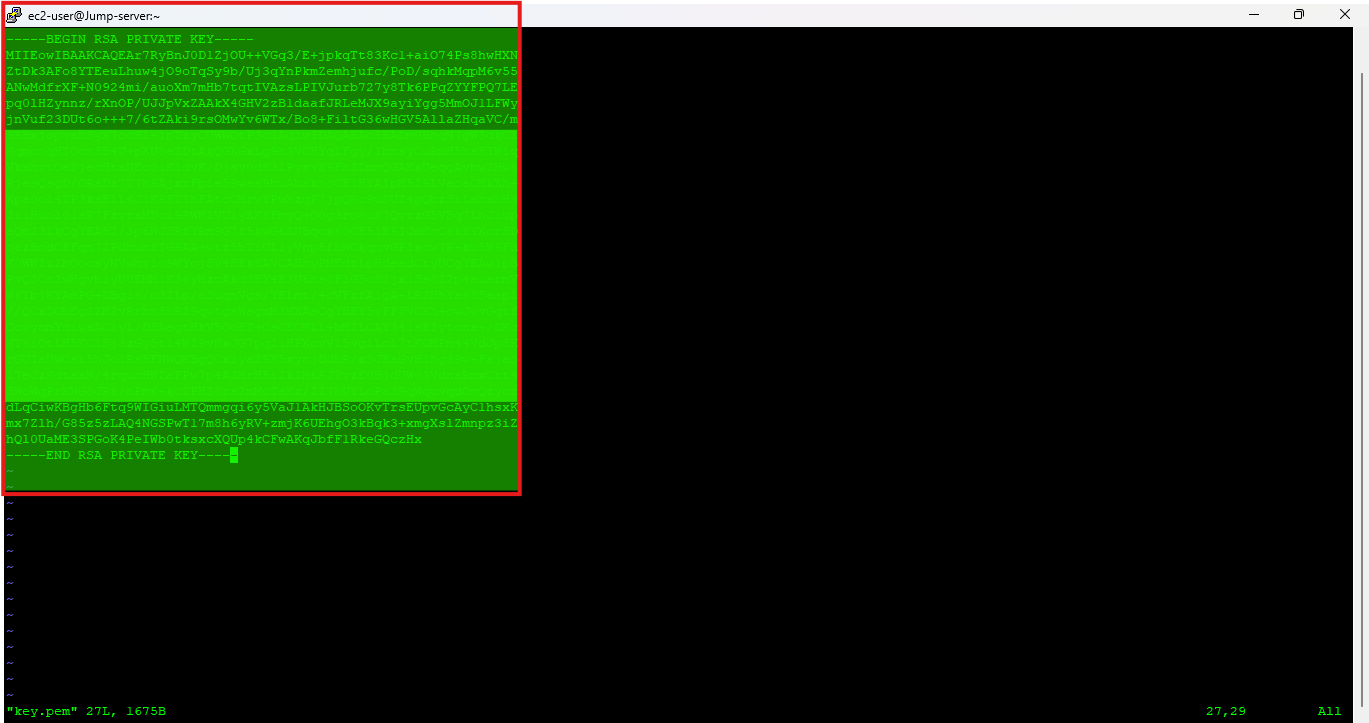
**Step 9 - Open the Jump server and take ssh to configure App server and database.**

****

* + Create a file using your custom key name (like key.pem)



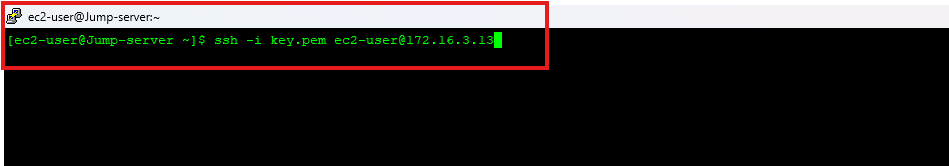
* + Paste your .pem key Into the file just created.



* + Then change permission for the key file in the Jump-server
  + sudo chmod 400 key.pem



* + Then take remote of your app-server using ssh
  + Bash: **ssh -i key.pem username@private IP**





At App-Server: Install MySql Agent for running RdS-DB Engine

Commands:

Install MySQL Client :

yum list mariadb\*

sudo yum install mariadb105-server.x86\_64

sudo systemctl enable mariadb

sudo systemctl start mariadb

sudo systemctl status mariadb.service

sudo mysql –version

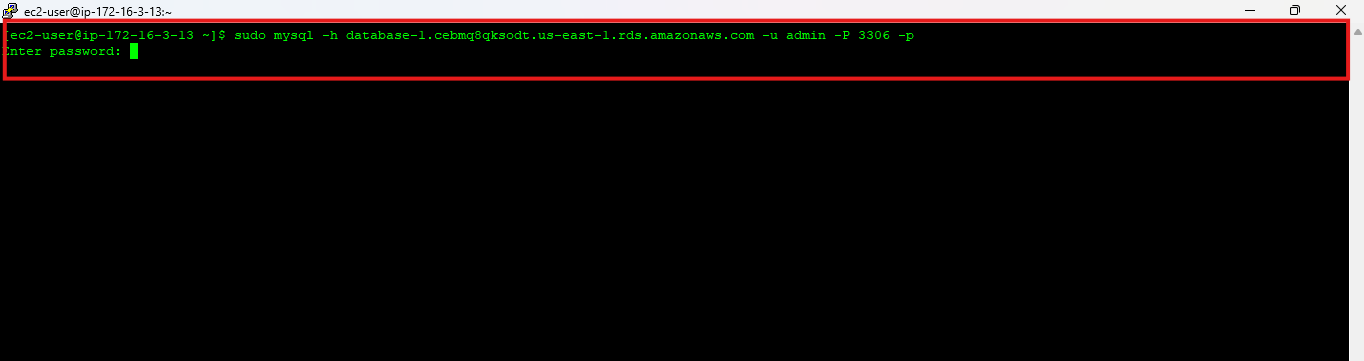
* + After executed all the command the you will see the version of your mysql client



Logint to your MySql\_DB Engine

Command :

mysql -h <Host\_Name> -P 3306 -u <User\_Name> -p then enter your password



For Ex

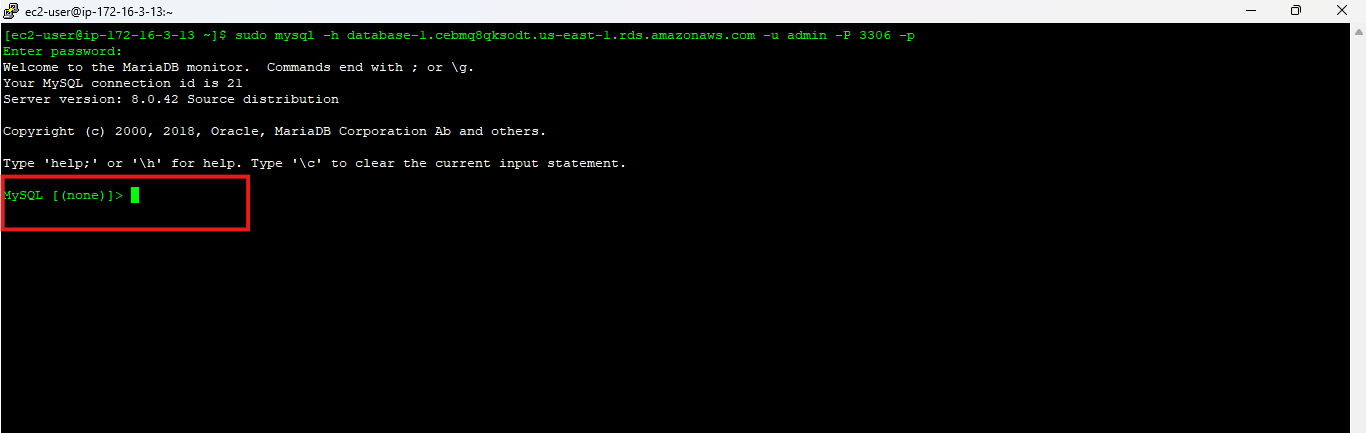
Hostname : database-1.cxz1od1wh1ta.us-east-1.rds.amazonaws.com

Username:admin

Passwrd: Admin123

The command is : mysql -h database-1.cxz1od1wh1ta.us-east-1.rds.amazonaws.com -P 3306 -u admin -p

You will then be prompted to type in your password. Once you input the password and hit enter, you should now be connected to your database.



*NOTE: If you cannot reach your database, check your credentials and security groups.*

A. Create a database called webappdb with the following command using the MySQL CLI:

|  |  |  |
| --- | --- | --- |
|  |  | A. Create a database called webappdb with the following command using the MySQL CLI:    CREATE DATABASE webappdb;  You can verify that it was created correctly with the following command:    SHOW DATABASES;   * + 1. Create a data table by first navigating to the database we just created:     USE webappdb;  Then, create the following transactions table by executing this create table command:    CREATE TABLE IF NOT EXISTS transactions (  id INT NOT NULL AUTO\_INCREMENT,  amount DECIMAL(10,2),  description VARCHAR(100),  PRIMARY KEY (id) );  Verify the table was created:    SHOW TABLES;   * + 1. Insert data into the table for use/testing later:     INSERT INTO transactions (amount, description) VALUES ('400', 'groceries');  Verify that your data was added by executing the following command:    SELECT \* FROM transactions;   * + 1. When finished, just type exit and hit enter to exit the MySQL client. |

Create S3 Bucket & Create IAM Role

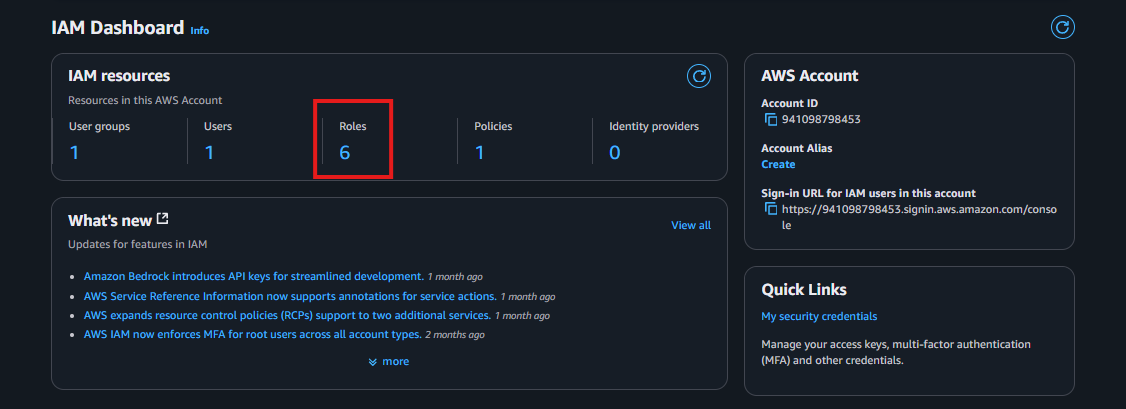
A . Create S3 bucket with default setting

* 1. Create IAM Role for EC2 ----> the policies needs to be attached
  + **AmazonSSMManagedInstanceCore,**
  + **AmazonS3ReadOnlyAccess**

🔐 Step-by-Step: Create IAM Role for EC2 (S3 Read-Only Access)

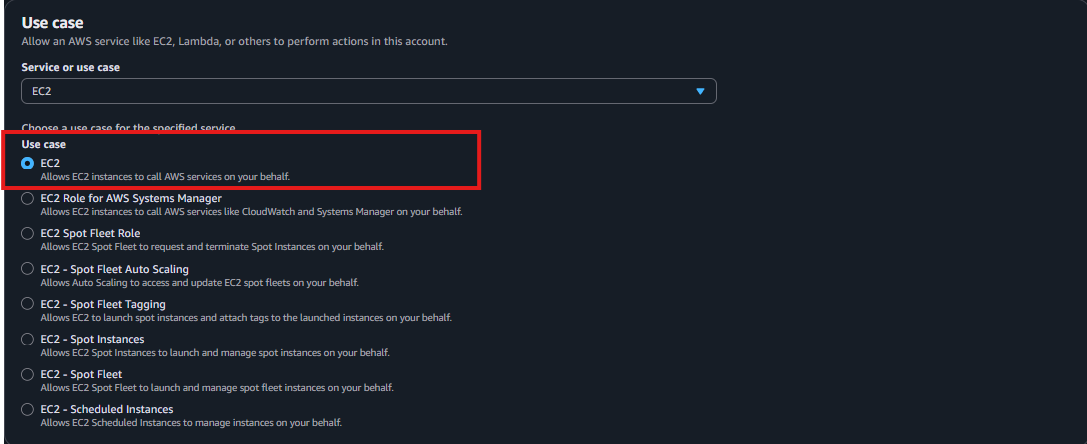
Step 1 – Navigate to IAM

1. Log in to the AWS Console and search for IAM.
2. In the left-hand navigation pane, click Roles.



Step 2 – Create New Role

1. Click Create role.
2. Under Trusted entity type, select AWS service.
3. Choose EC2 as the use case (this allows the role to be assumed by EC2 instances).
4. Click Next.

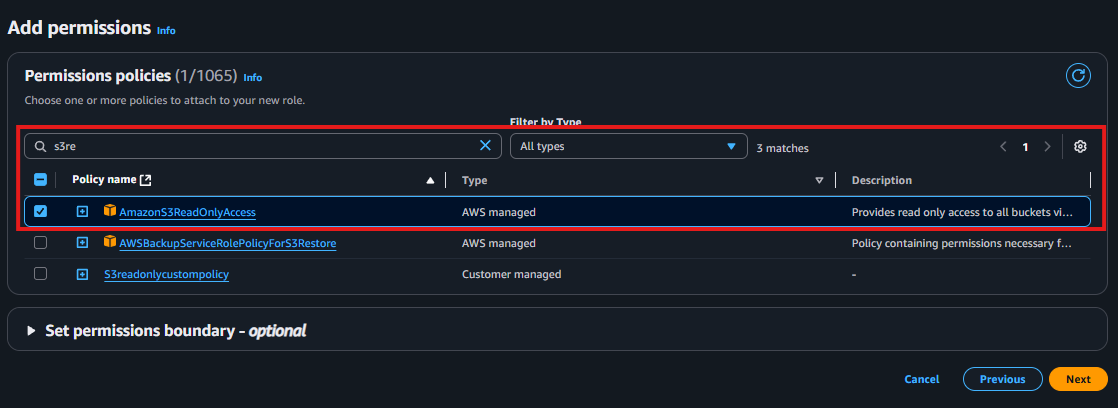


Step 3 – Attach Permission Policy

1. In the policy list, search for AmazonS3ReadOnlyAccess.
2. Select the checkbox for AmazonS3ReadOnlyAccess.

✅ This policy allows the EC2 instance to list and retrieve objects from any S3 bucket.

1. Click Next.



Step 4 – Name and Tag the Role

1. Provide a name, e.g. EC2-S3ReadOnlyRole.
2. (Optional) Add tags such as:
   * Key = Project, Value = 3TierArchitecture
3. Click Create role.

🟢 The role is now created.

Step 5 – Attach the Role to EC2 Instances

1. Navigate to EC2 > Instances.
2. Select the Web or App instance that requires S3 access.
3. Click Actions → Security → Modify IAM role.
4. In the dropdown, select EC2-S3ReadOnlyRole.
5. Click Update IAM role.

* Upload your aws-three-tier-web-architecture-workshop 🡪 application-code then upload the app-tier, web-tier, nginx.conf.

**Configure App-Server**

* + The first thing we will do is update our database credentials for the app tier. To do this, open the **application-code/app-tier/DbConfig.js** file from the github repo in your favorite text editor on your computer. You’ll see empty strings for the hostname, user, password and database. Fill this in with the credentials you configured for your database, the **writer** endpoint of your database as the hostname, and **webappdb** for the database. Save the file.
    1. Open the foler you download the code from Git Clone
       1. Open-->aws-three-tier-web-architecture-workshop---> application-code--->app-tier--->**DbConfig.js**

**And add thr host name username & Password---> Then Save it**

module.exports = Object.freeze({

DB\_HOST : 'database-1.cxz1od1wh1ta.us-east-1.rds.amazonaws.com',

DB\_USER : 'admin',

DB\_PWD : 'Admin123',

DB\_DATABASE : 'webappdb'

});

aws s3 cp s3://mynewbsocjod/app-tier/ . --recursive s3://mynewbsocjod/web-tier/

* + After Updating the credentials Updating the credentials to your Dbconfig.json upload the App\_tier foler to your S3 Bucket
    1. Open S3 buckeg what you created in step 1
    2. Choose upload
    3. Select the App-tier folder only
    4. Then Upload the folder
  + Go Back to your App-Tier Putty Session

a. Start by installing NVM (node version manager).

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

source ~/.bashrc

b. Next, install a compatible version of Node.js and make sure it's being used

nvm install 16

nvm use 16

* + PM2 is a daemon process manager that will keep our node.js app running when we exit the instance or if it is rebooted. Install that as well.

npm install -g pm2

* + Now we need to download our code from our s3 buckets onto our instance. In the command below, replace BUCKET\_NAME with the name of the bucket you uploaded the **app-tier** folder to:

cd ~/

aws s3 cp s3://BUCKET\_NAME/app-tier/ app-tier --recursive

* 1. Navigate to the app directory, install dependencies, and start the app with pm2.

cd ~/app-tier

npm install

pm2 start index.js

|  |  |
| --- | --- |
| F: To make sure the app is running correctly run the following: |  |

pm2 list

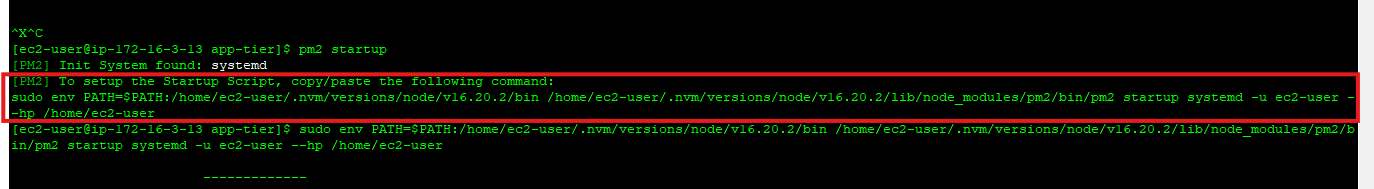
G: you see a status of online, the app is running. If you see errored, then you need to do some troubleshooting. To look at the latest errors, use this command:

pm2 logs

H: Right now, pm2 is just making sure our app stays running when we leave the SSM session. However, if the server is interrupted for some reason, we still want the app to start and keep running. This is also important for the AMI we will create:

pm2 startup

Ater that run the comment from the terminal



After you run it, save the current list of node processes with the following command:

pm2 save

Test App Tier

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

To hit out 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.

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

The response should looks like the following:

"This is the health check"

Next, test your database connection. You can do that by hitting the following endpoint locally:

1  
curl <http://localhost:4000/transaction>

You should see a response containing the test data we added earlier:

1  
{"result":[{"id":1,"amount":400,"description":"groceries"},{"id":2,"amount":100,"description":"class"},{"id":3,"amount":200,"description":"other groceries"},{"id":4,"amount":10,"description":"brownies"}]}

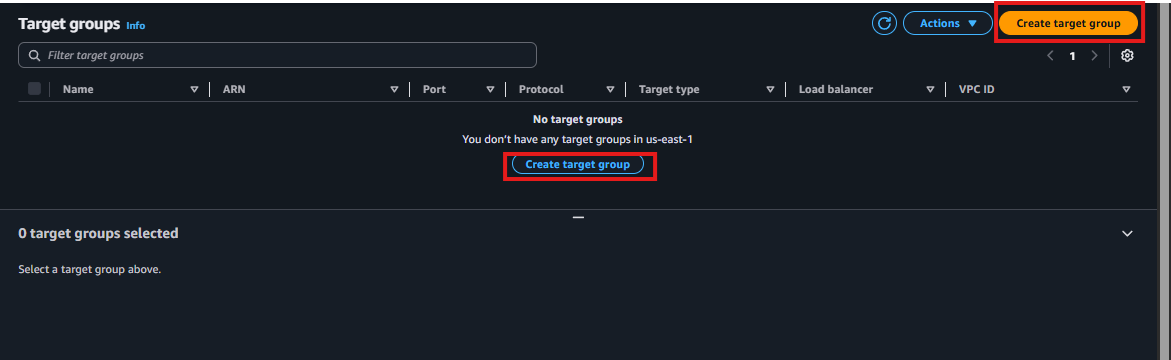
If you see both of these responses, then your networking, security, database and app configurations are correct.

* 1. Congrats! Your app layer is fully configured and ready to go.

**Step 10 – Internal load balancer and Auto scaling the app-server**

**Step 1 – Create Target Group (App Tier)**

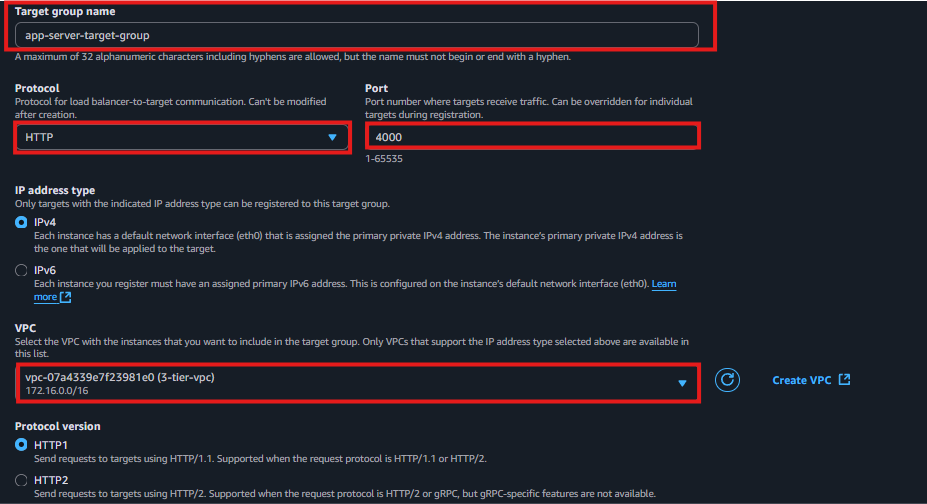
1. Go to EC2 → Target Groups in the AWS Console.
2. Click Create target group.

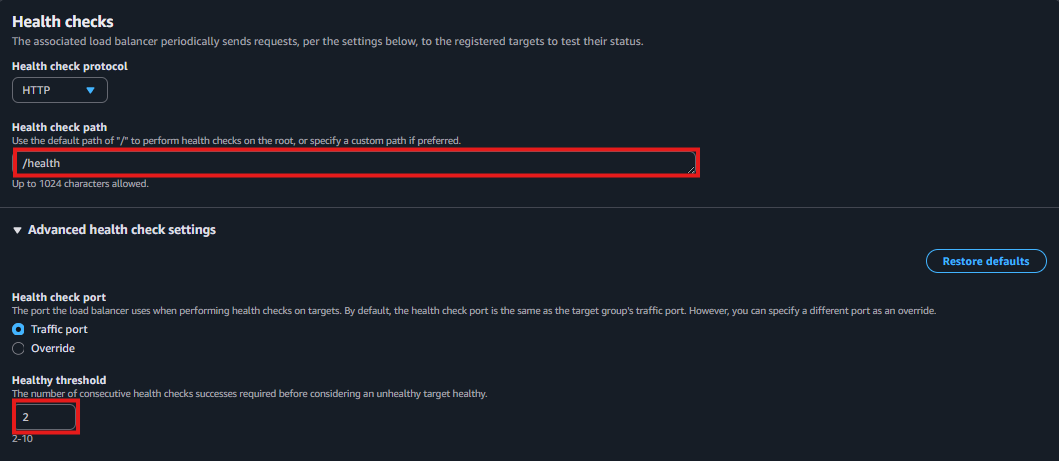
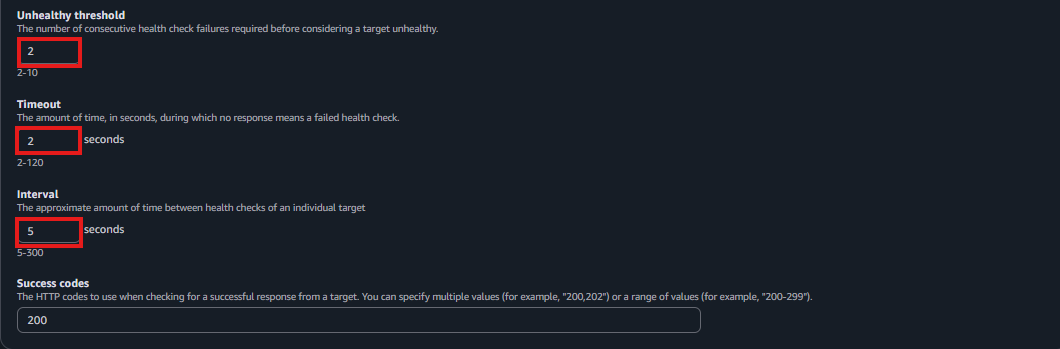
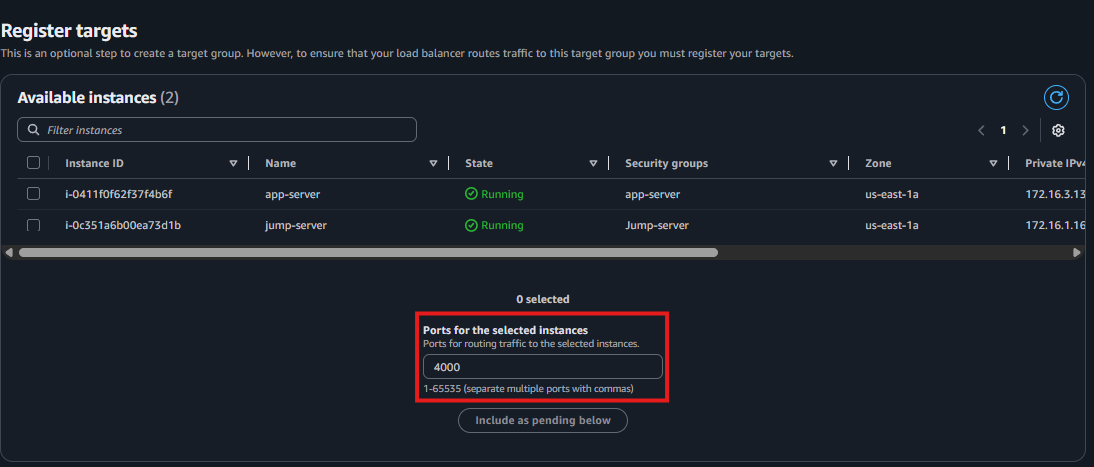
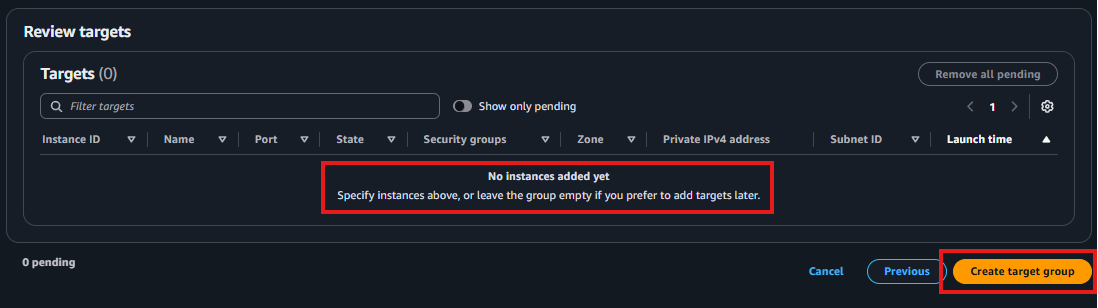


1. Configure the following options:

| Field | Value |
| --- | --- |
| Target type | Instance |
| Name | app-tier-target-group |
| Protocol | HTTP |
| Port | 4000 |
| VPC | Select your custom VPC |
| Health Check Protocol | HTTP |
| Health Check Path | /health |

1. Click Next.
2. Do not register any targets yet (leave it empty).

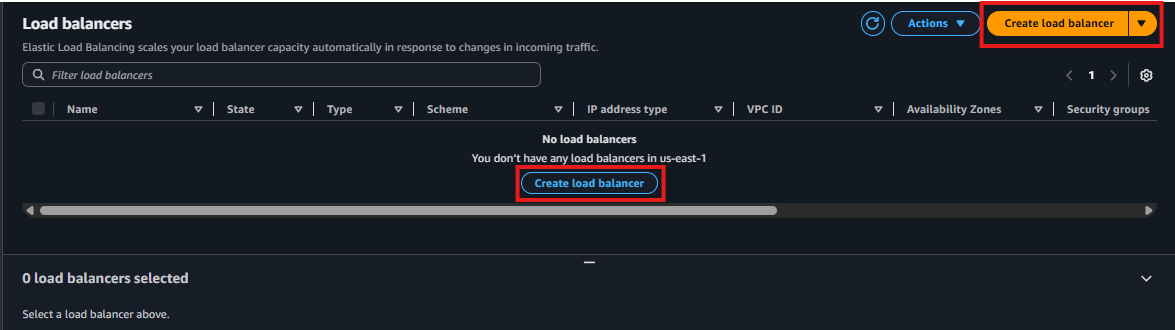


1. Click Create target group.

Step 2 – Create Internal Application Load Balancer

1. Go to EC2 → Load Balancers.



1. Click Create Load Balancer → Application Load Balancer.

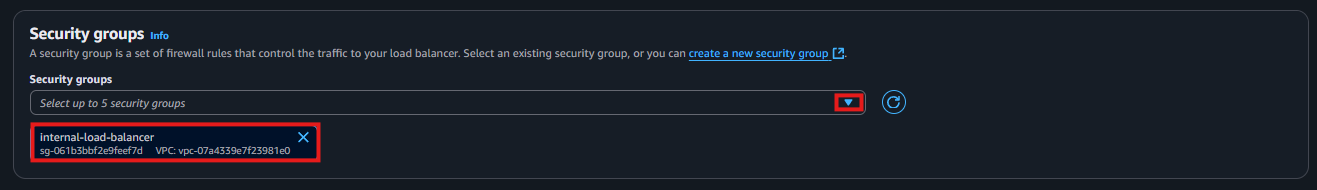
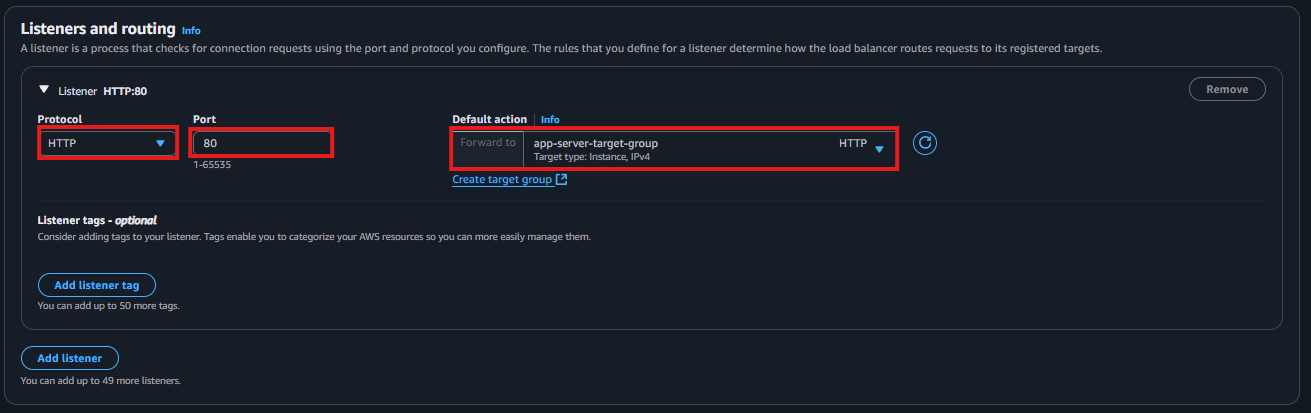


1. Configure the following:

| Field | Value |
| --- | --- |
| Name | internal-app-alb |
| Scheme | Internal |
| IP address type | IPv4 |
| VPC | Select your custom VPC |
| Availability Zones | Select Private subnets (App subnets) |

Click Next: Configure Security Settings (leave default).

Click **Next** and **Create Load Balancer.**

**🔧 Step 1 – Create an AMI from the App Server**

1. Navigate to EC2 > Instances.
2. Select your existing App EC2 instance (the one you’ve already configured and tested).
3. Click Actions → Image → Create Image.
4. Provide a name (e.g. *app-server-ami*) and description.
5. Leave other values as default and click Create Image.
6. Wait a few minutes until the AMI status becomes Available (check under AMI section in the EC2 console).

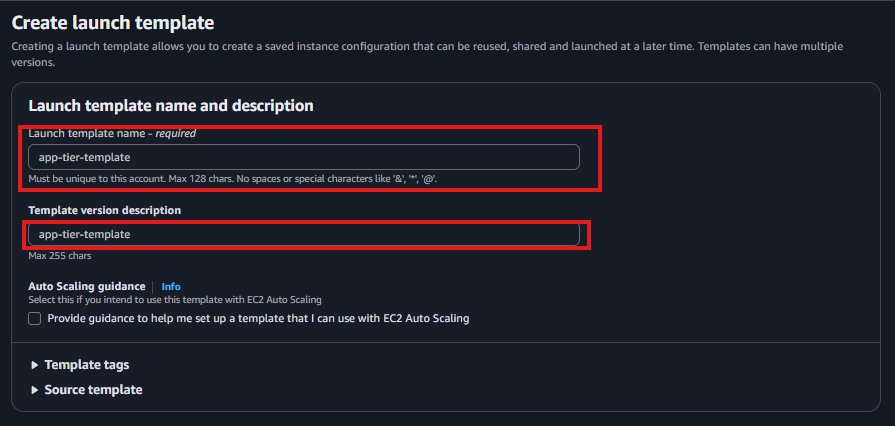
**🔧 Step 2 – Create a Launch Template Using the AMI**

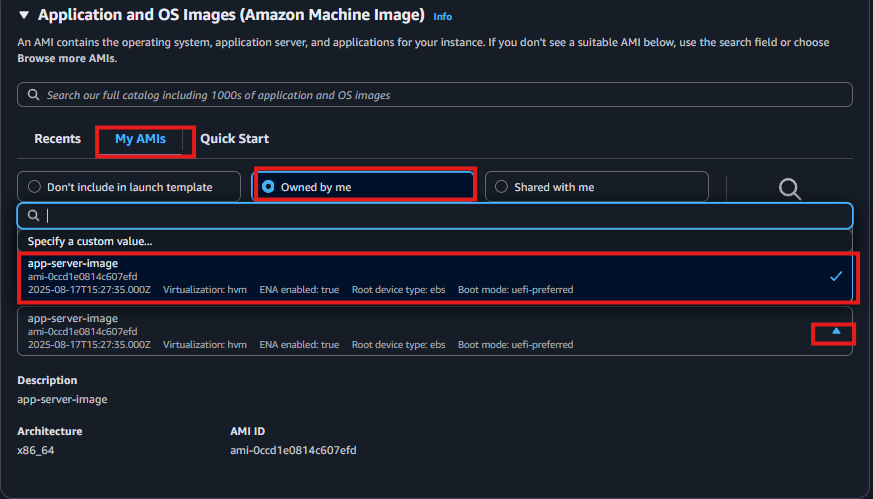
1. Go to EC2 → Launch Templates → Create launch template

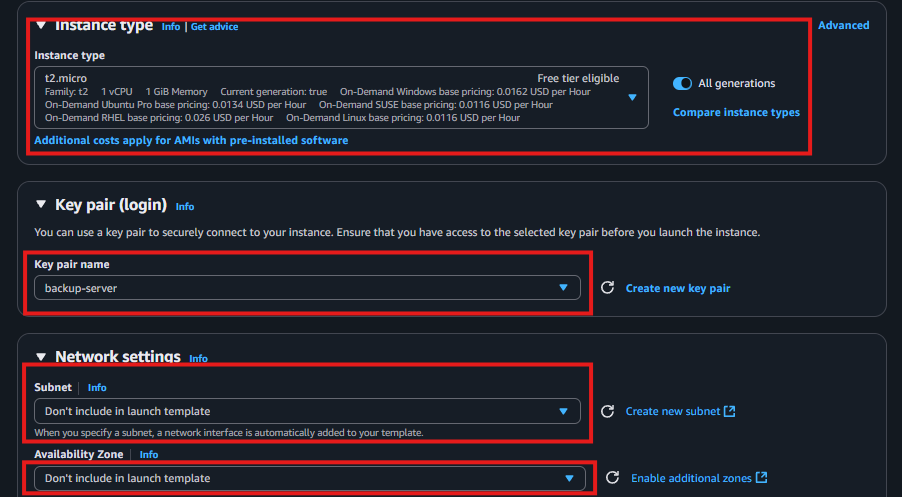


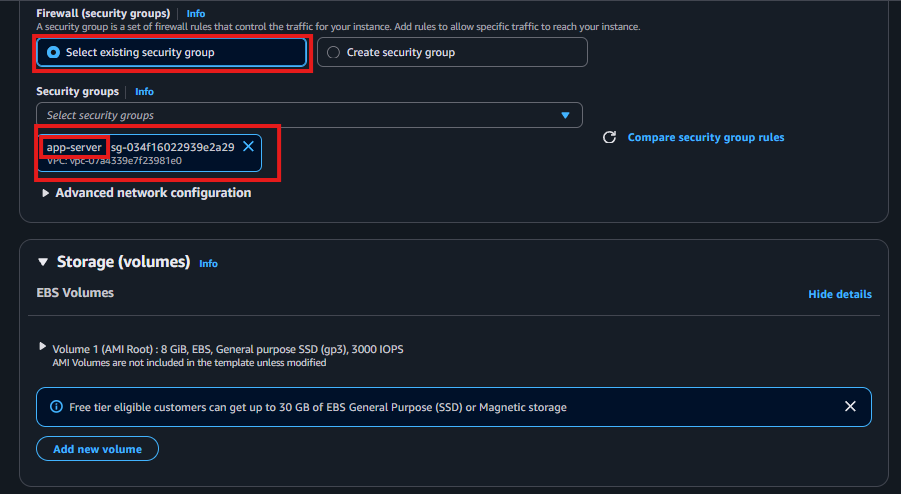
1. Set the following:

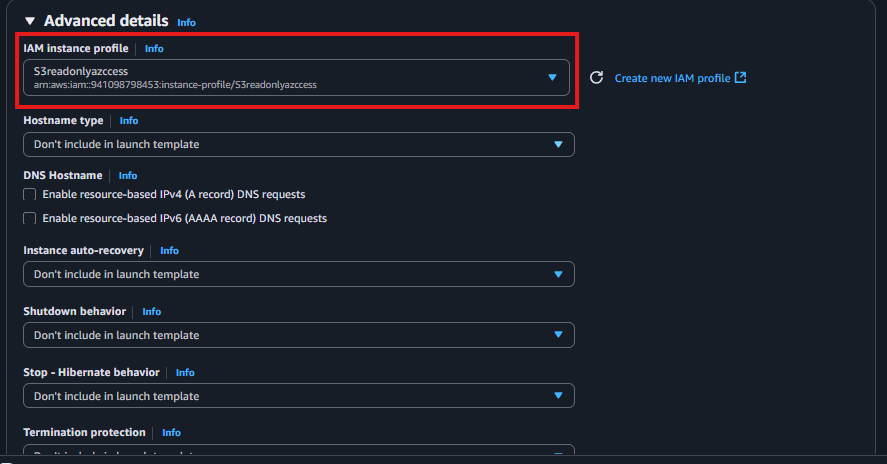
| Field | Value |
| --- | --- |
| Launch template name | app-tier-template |
| AMI ID | Select the AMI created in Step 1 |
| Instance type | (same type used for the App Server, e.g. t2.micro) |
| Key pair | Select the required key pair |
| Security group | App Server SG (allows port 4000 from the internal ALB) |
|  |  |

****

****

****

****

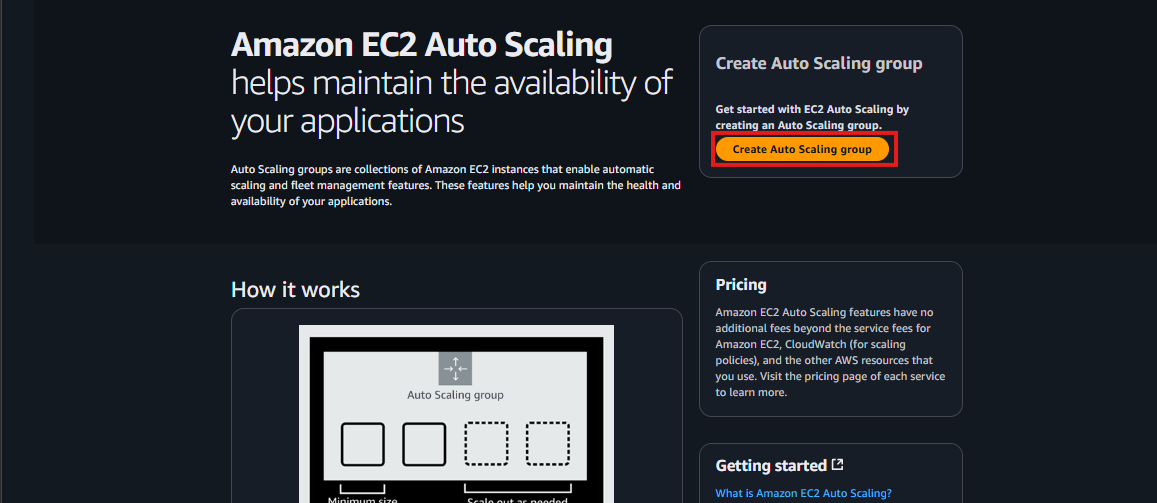
****

1. Click Create launch template

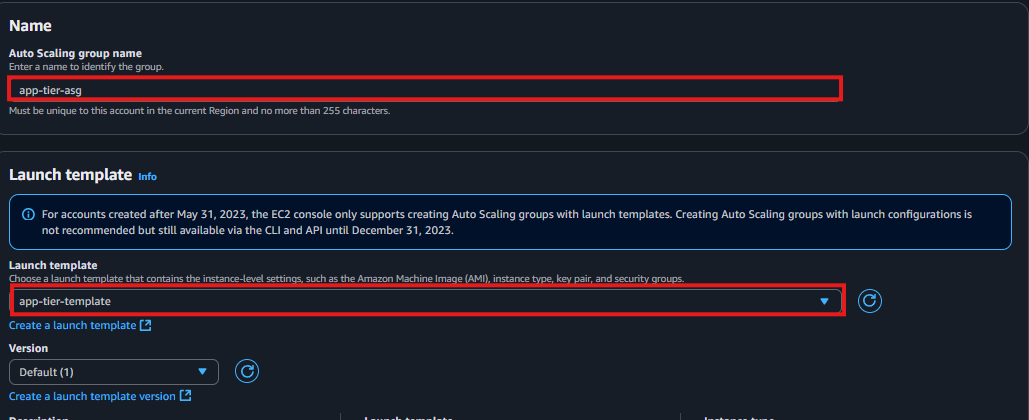
🔧 Step-by-Step: App Target Group + Internal Load Balancer (Port 4000)

**🔄 Step 3 – Create Auto Scaling Group Based on the Launch Template**

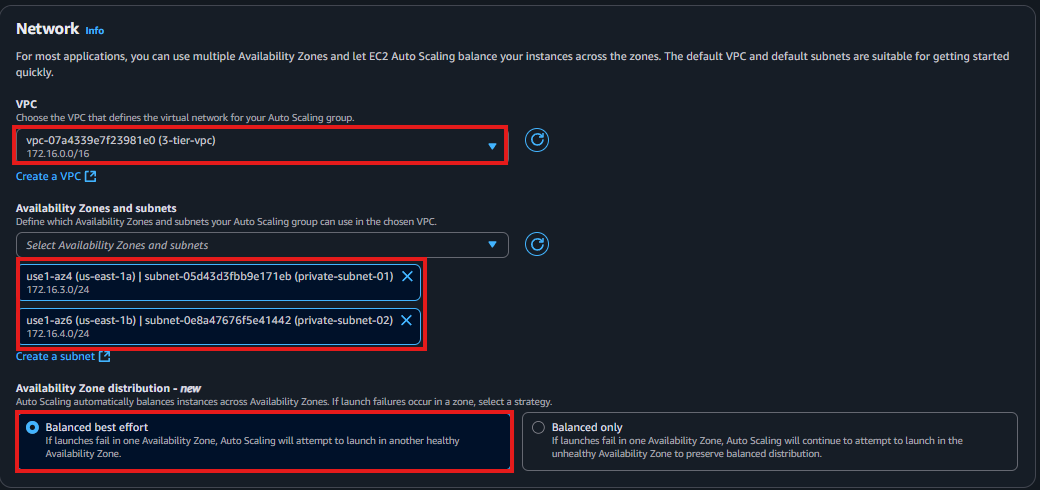
1. Navigate to **Auto Scaling Groups → Create Auto Scaling group**



1. **Auto Scaling group name**: app-tier-asg
2. **Launch template**: select app-tier-template



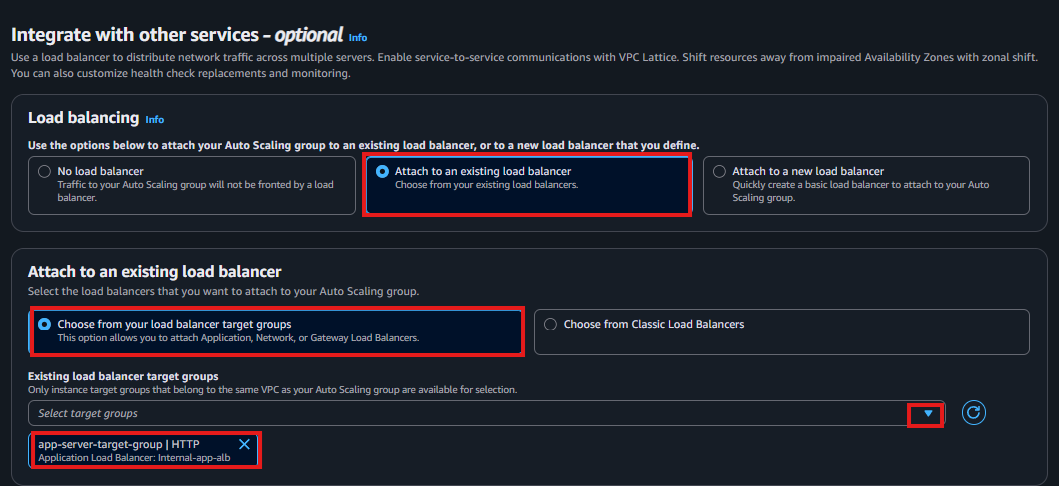
1. **VPC**: Select the custom VPC
2. **Subnets**: Select **private subnets**

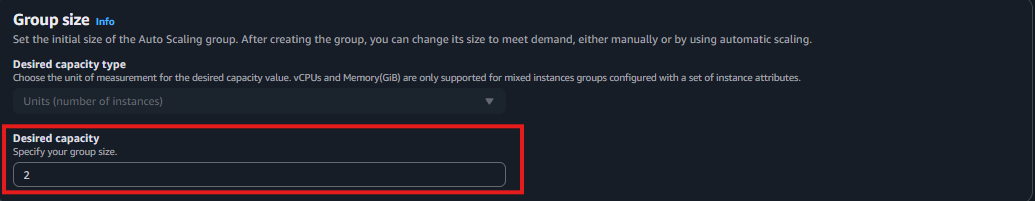


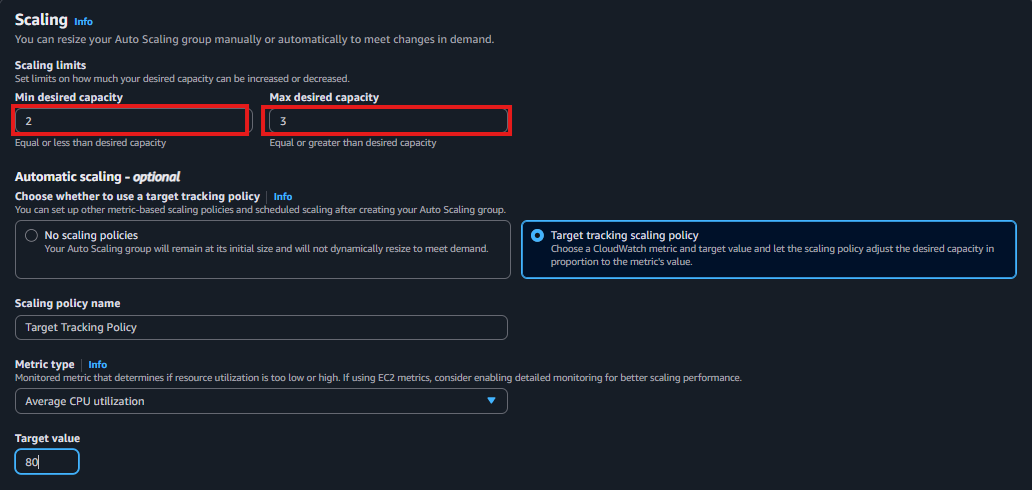
Click **Next**

**🌐 Step 4 – Attach to the Internal Load Balancer**

1. Select **Attach to an existing load balancer**
2. Choose **Application Load Balancer**
3. Select **internal-app-alb**
4. Select the **target group** you created for the App Tier (e.g. app-target-group)



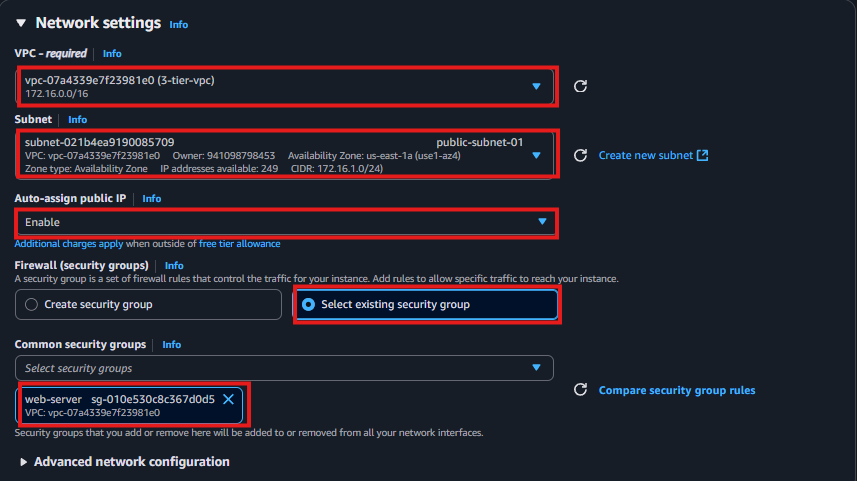


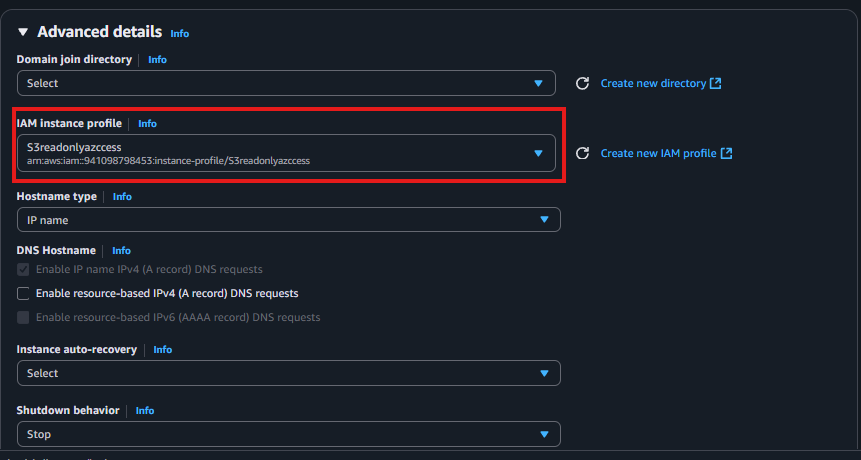


Click **Next** and **finish** the creation of auto-scaling group

**Step 11 – Launch EC2 Instances (Web-Tier)**

1. Repeat the steps for launching new EC2 instances, but select the public subnets. And choose the web-tier security group that we have created early. And enable public IP assigning

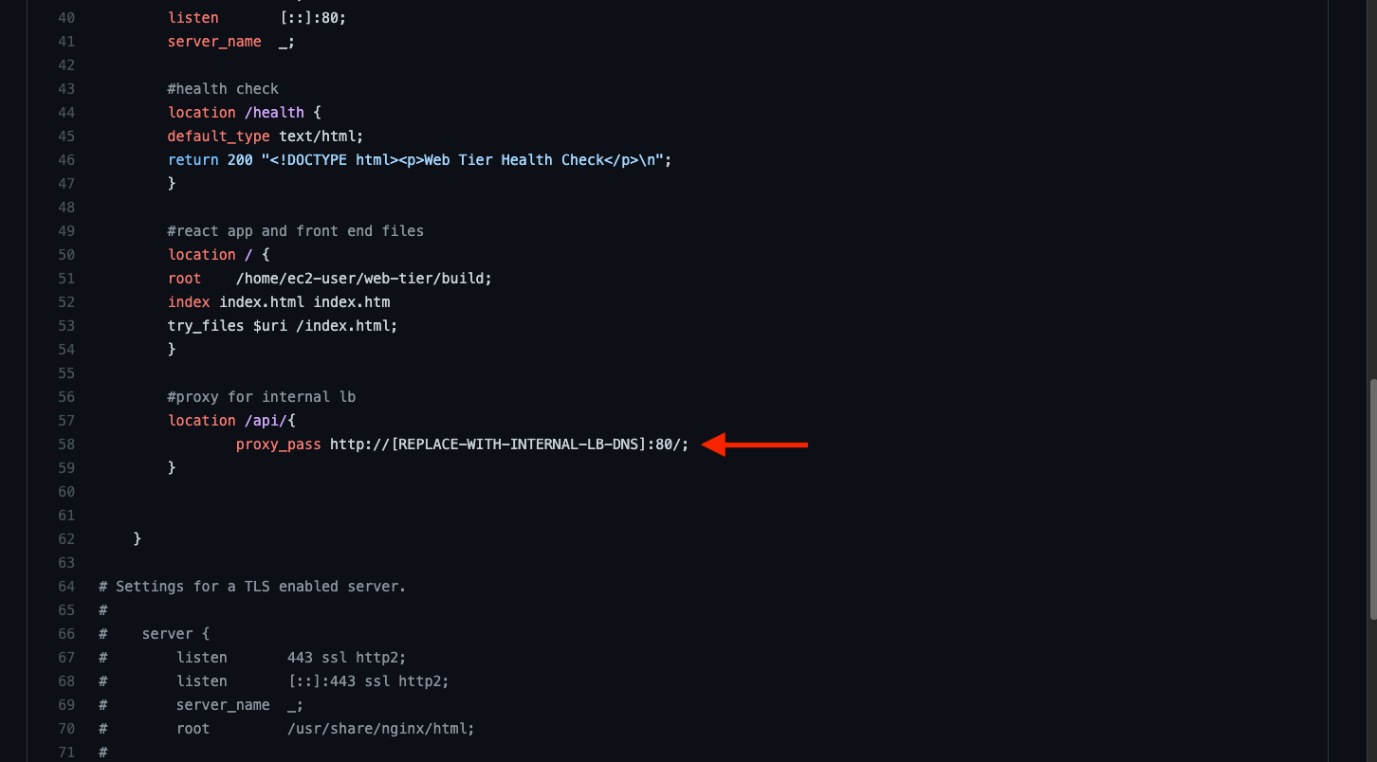
****

****

* Then click on **“Launch Instance”**

Update Config File

* Before we create and configure the web instances, open up the **application-code/nginx.conf** file from the repo we downloaded. Scroll down to **line 58** and replace [INTERNAL-LOADBALANCER-DNS] with your internal load balancer’s DNS entry. You can find this by navigating to your internal load balancer's details page.



<http://internal-pvt-lb-1627883891.ap-south-1.elb.amazonaws.com/>:

From <<http://internal-pvt-lb-1627883891.ap-south-1.elb.amazonaws.com/>>

**Configure Web Instance**

* 1. We now need to install all of the necessary components needed to run our front-end application. Again, start by installing NVM and node :

1 curl -o- <https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh> | bash  
2 source ~/.bashrc  
3 nvm install 16  
4 nvm use 16

* 1. 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

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

1 cd ~/web-tier  
2 npm install  
3 npm run build

* 1. 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.

1 sudo yum install nginx1 -y

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

1 cd /etc/nginx  
2 ls

You should see an nginx.conf file. We’re going to delete this file and use the one we uploaded to s3. Replace the bucket name in the command below with the one you created for this workshop:

1 sudo rm nginx.conf  
2 sudo aws s3 cp s3://BUCKET\_NAME/nginx.conf .  
  
Then, restart Nginx with the following command:

1 sudo service nginx restart  
To make sure Nginx has permission to access our files execute this command:

1 chmod -R 755 /home/ec2-user

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

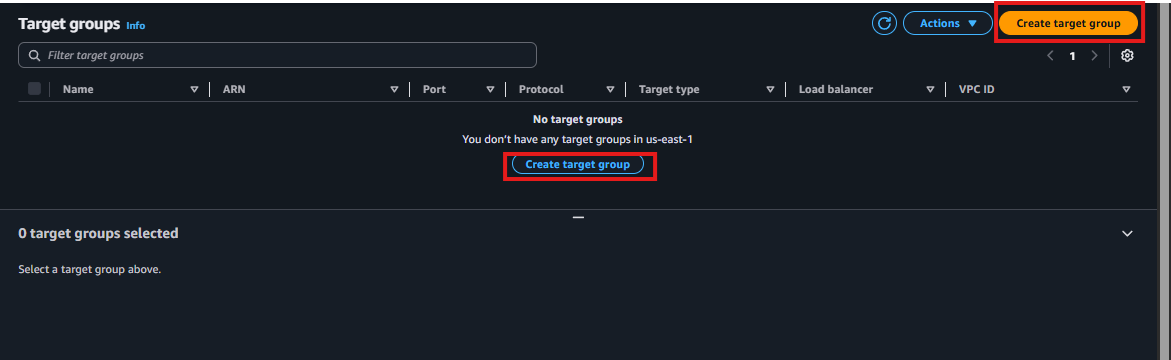
1 sudo chkconfig nginx on

Now when you plug in the public IP of your web tier instance, you should see your website, which you can find on the Instance details page on the EC2 dashboard. If you have the database connected and working correctly, then you will also see the database working. You’ll be able to add data. Careful with the delete button, that will clear all the entries in your database.

**Step 12 – External load balancer and Auto scaling the web-server**

**Step 1 – Create Target Group (Web-Tier)**

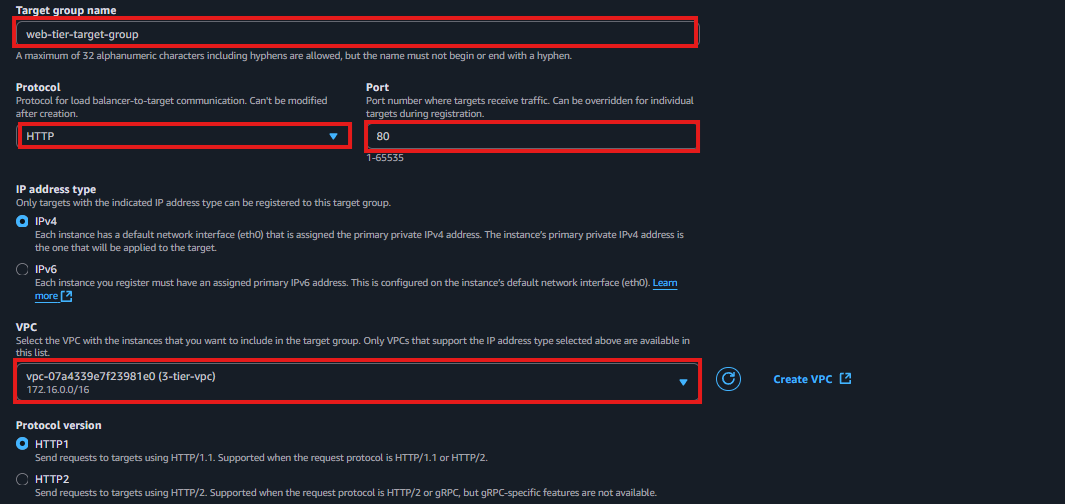
1. Go to EC2 → Target Groups in the AWS Console.
2. Click Create target group.

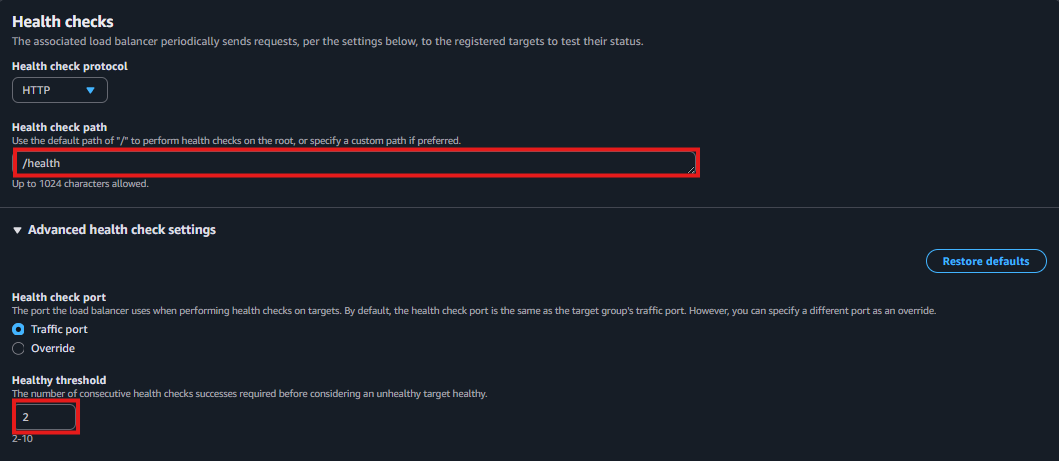
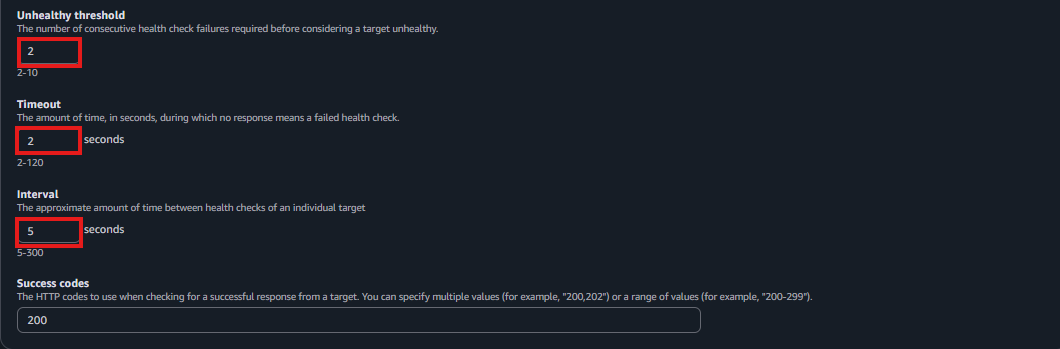
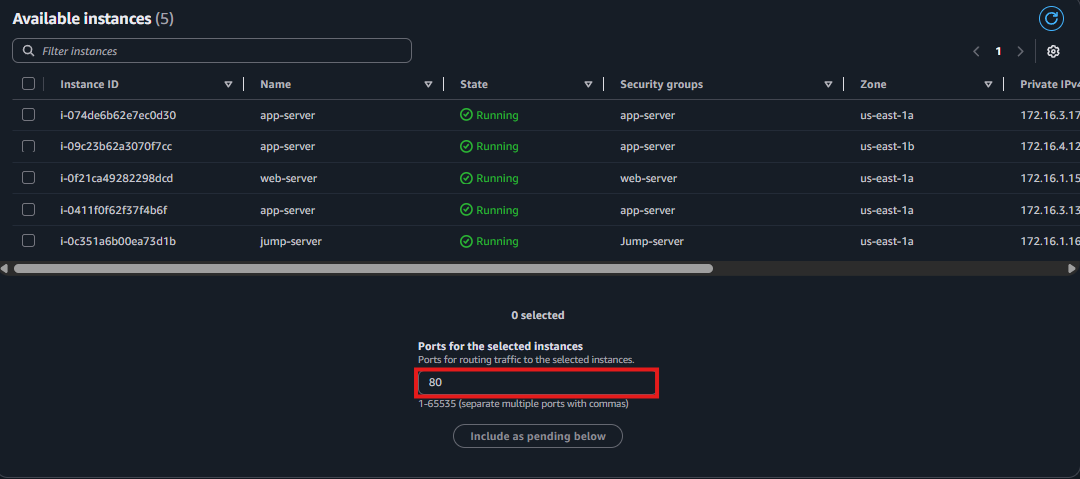
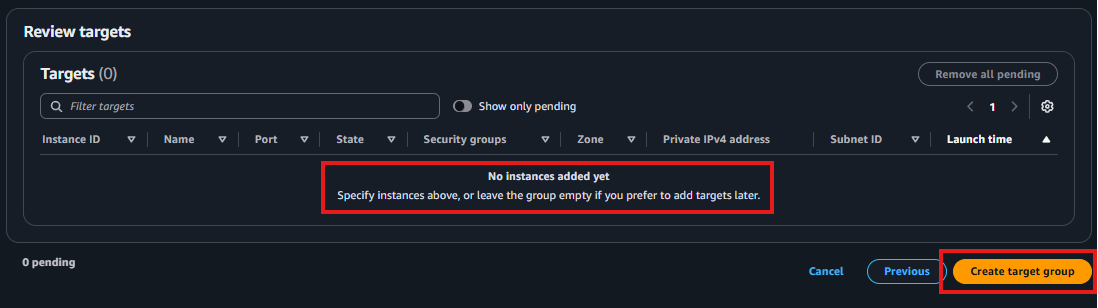


1. Configure the following options:

| Field | Value |
| --- | --- |
| Target type | Instance |
| Name | web-tier-target-group |
| Protocol | HTTP |
| Port | 80 |
| VPC | Select your custom VPC |
| Health Check Protocol | HTTP |
| Health Check Path | /health |

1. Click Next.
2. Do not register any targets yet (leave it empty).

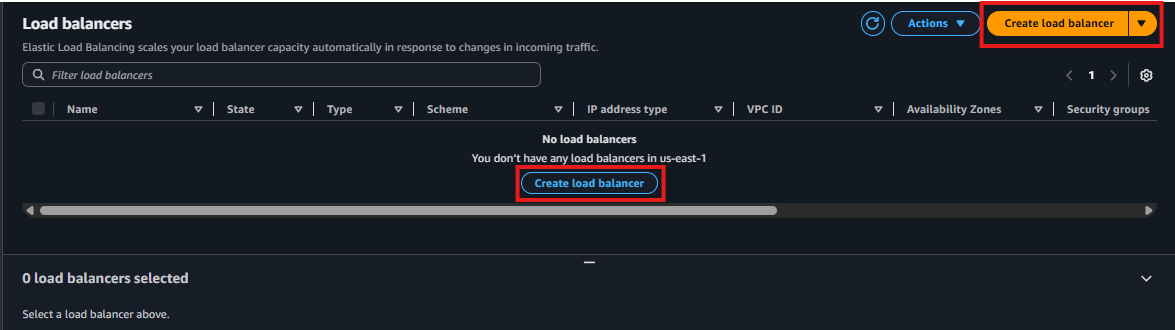


1. Click Create target group.

Step 2 – Create Internal Application Load Balancer

1. Go to EC2 → Load Balancers.



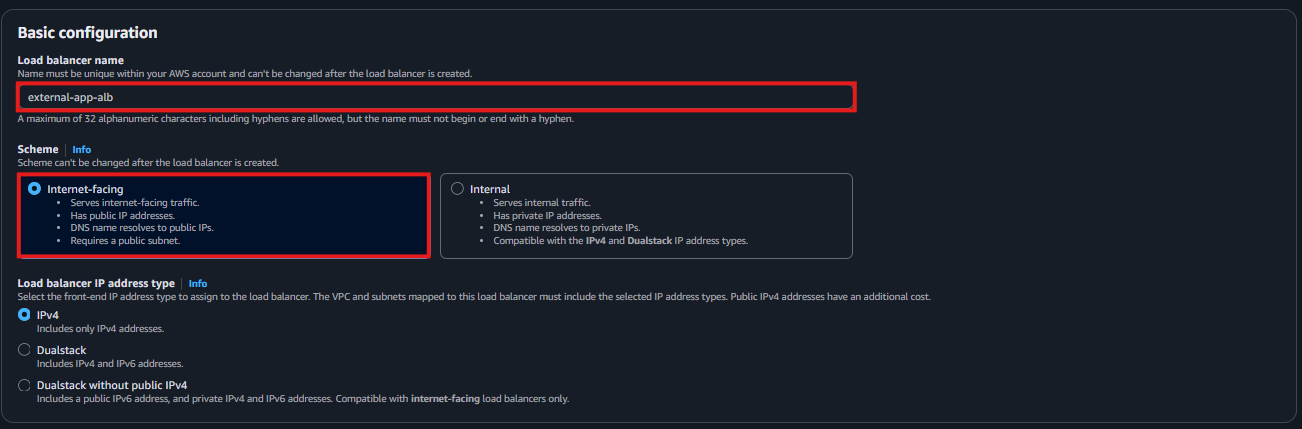
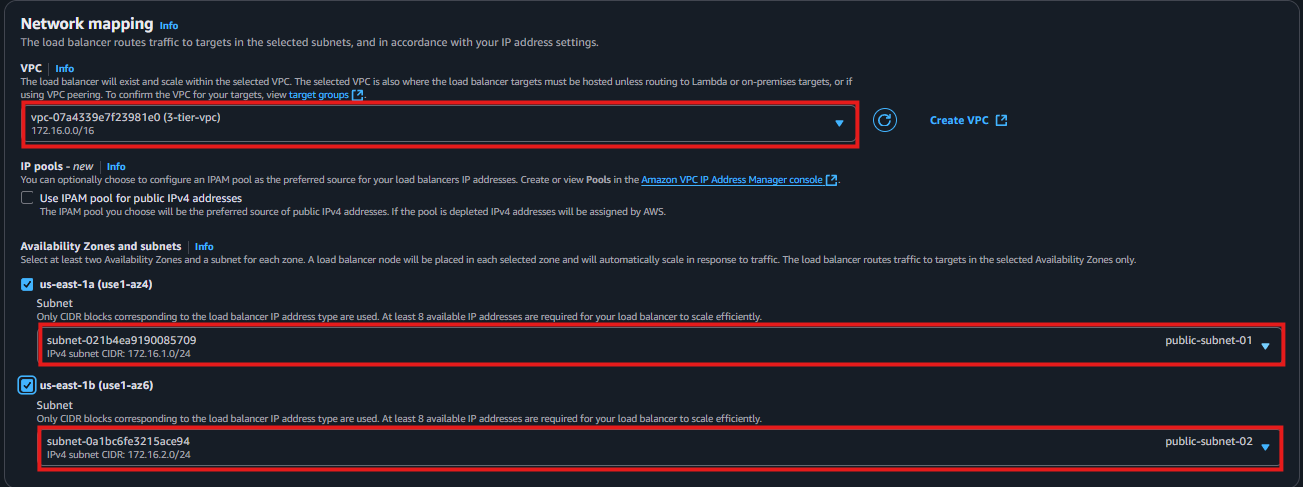
1. Click Create Load Balancer → Application Load Balancer.

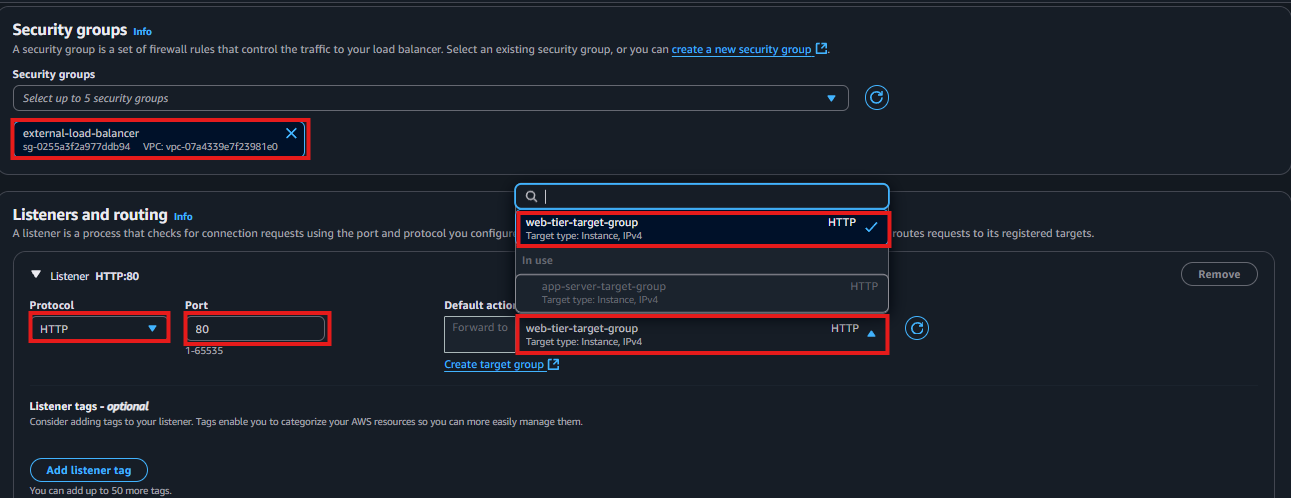


1. Configure the following:

| Field | Value |
| --- | --- |
| Name | external-app-alb |
| Scheme | Internet-facing |
| IP address type | IPv4 |
| VPC | Select your custom VPC |
| Availability Zones | Select Public subnets (Web subnets) |

Click Next: Configure Security Settings (leave default).



Click **Next** and **Create Load Balancer.**

**🔧 Step 1 – Create an AMI from the App Server**

1. Navigate to EC2 > Instances.
2. Select your existing Web EC2 instance (the one you’ve already configured and tested).
3. Click Actions → Image → Create Image.
4. Provide a name (e.g. *web-tier-ami*) and description.
5. Leave other values as default and click Create Image.
6. Wait a few minutes until the AMI status becomes Available (check under AMI section in the EC2 console).

**🔧 Step 2 – Create a Launch Template Using the AMI**

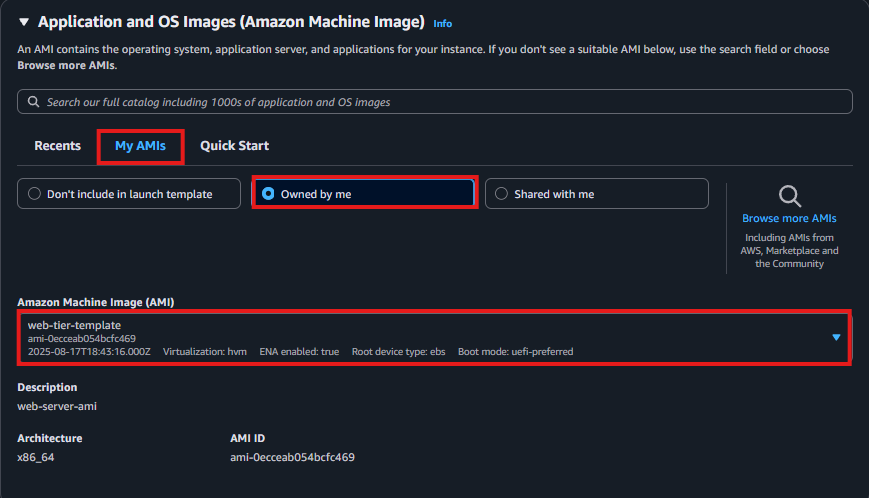
1. Go to EC2 → Launch Templates → Create launch template

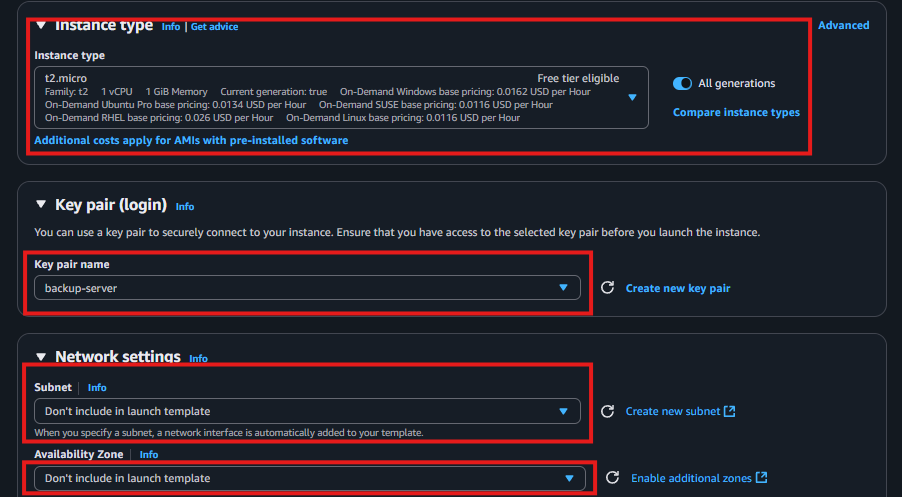


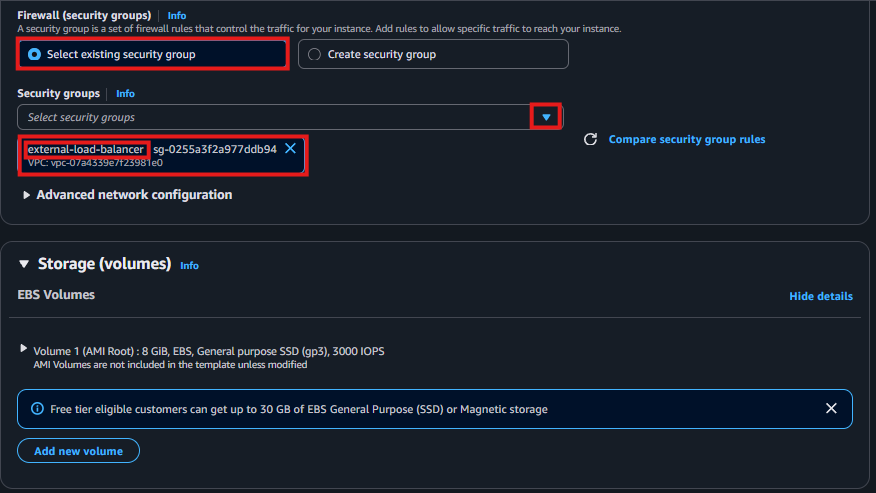
1. Set the following:

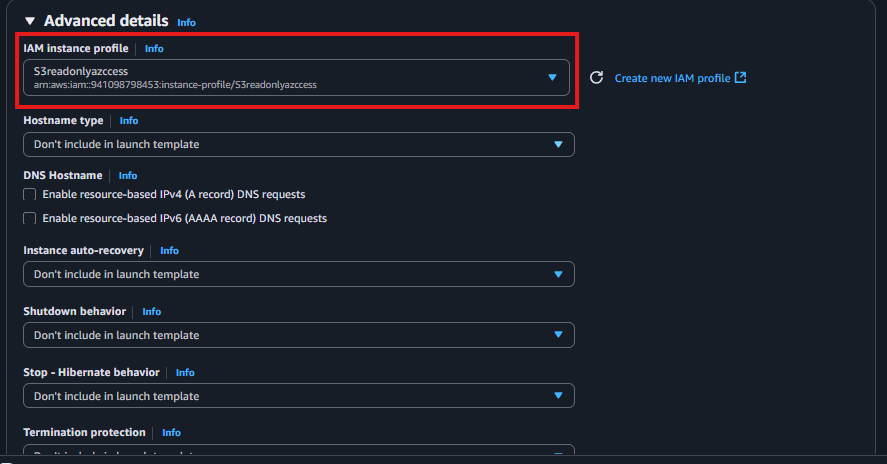
| Field | Value |
| --- | --- |
| Launch template name | web-tier-template |
| AMI ID | Select the AMI created in Step 1 |
| Instance type | (same type used for the web Server, e.g. t2.micro) |
| Key pair | Select the required key pair |
| Security group | web Server SG (allows port 80 from the External ALB) |
|  |  |

****

****

****

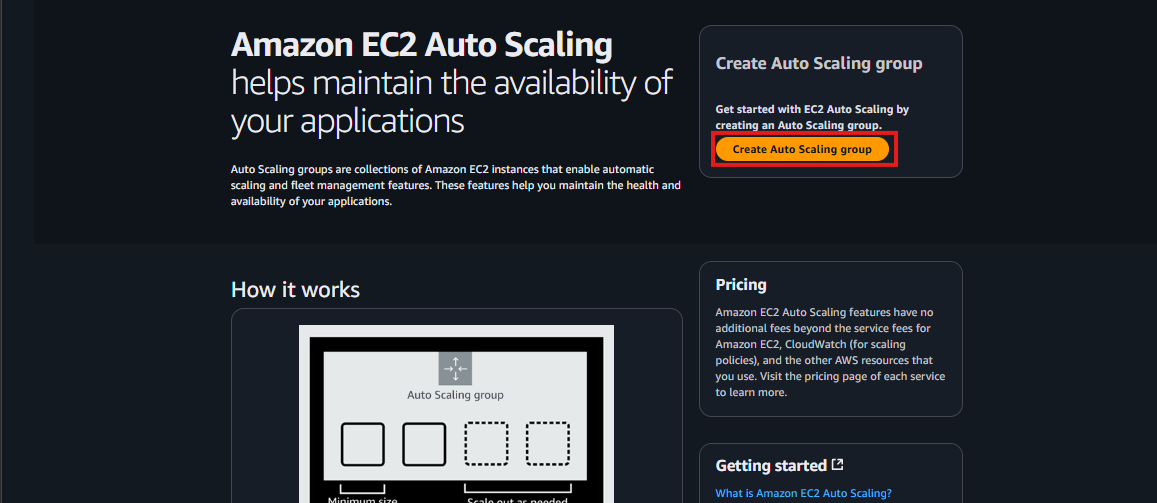
****

****

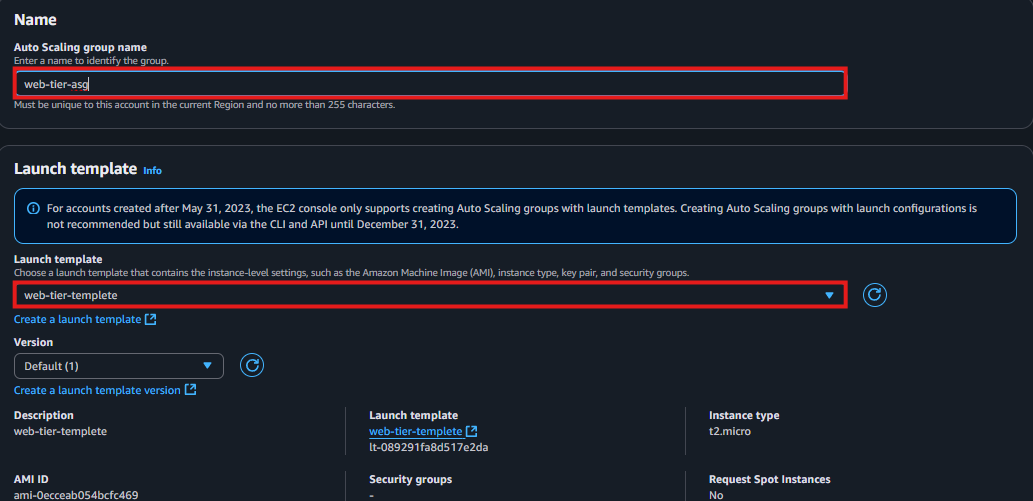
1. Click Create launch template

**🔄 Step 3 – Create Auto Scaling Group Based on the Launch Template**

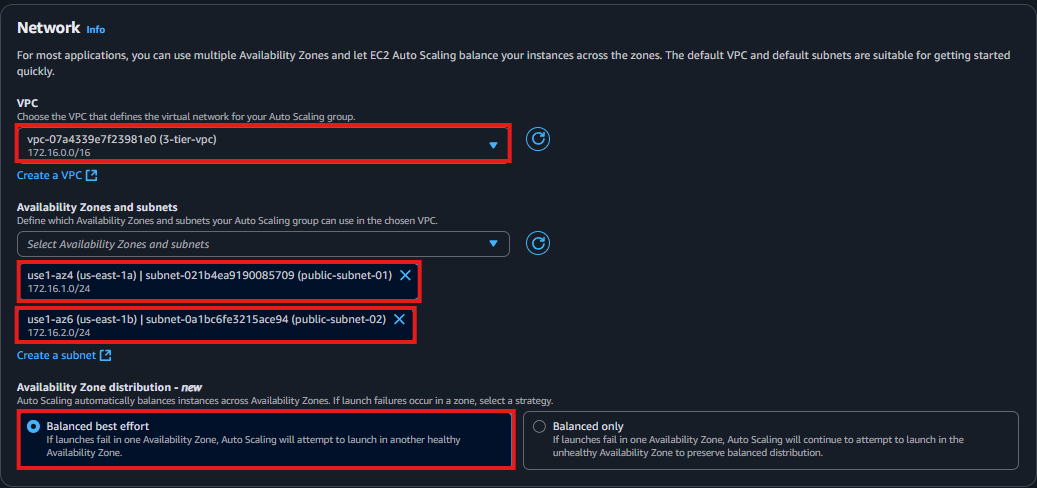
1. Navigate to **Auto Scaling Groups → Create Auto Scaling group**



1. **Auto Scaling group name**: web-tier-asg
2. **Launch template**: select web-tier-template



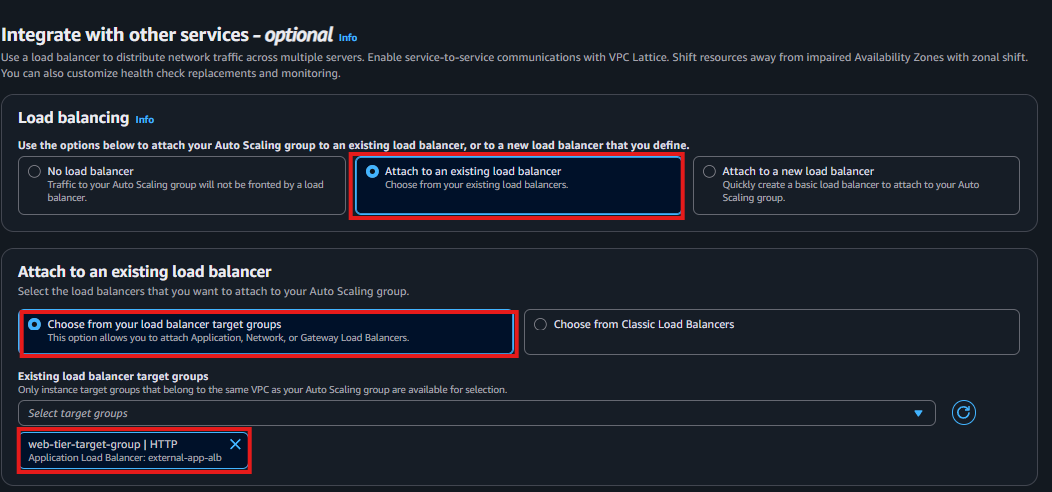
1. **VPC**: Select the custom VPC
2. **Subnets**: Select **Public subnets**

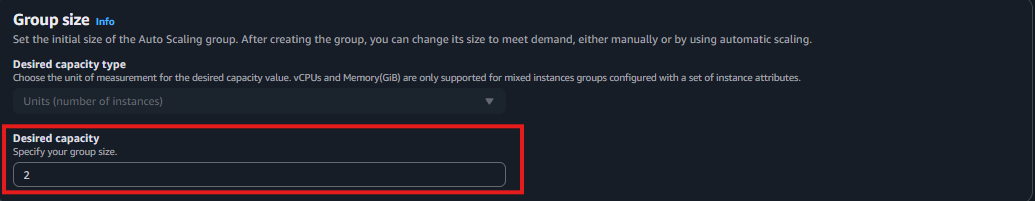


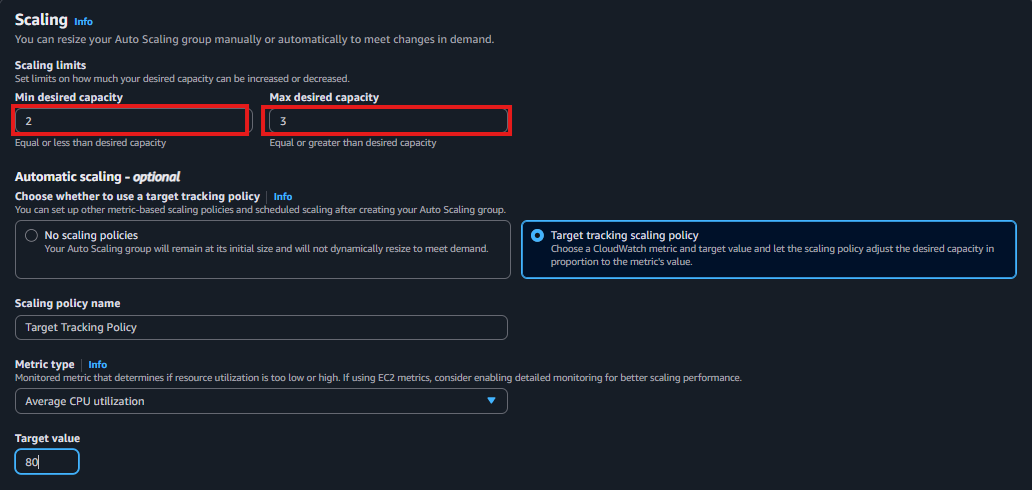
Click **Next**

**🌐 Step 4 – Attach to the External Load Balancer**

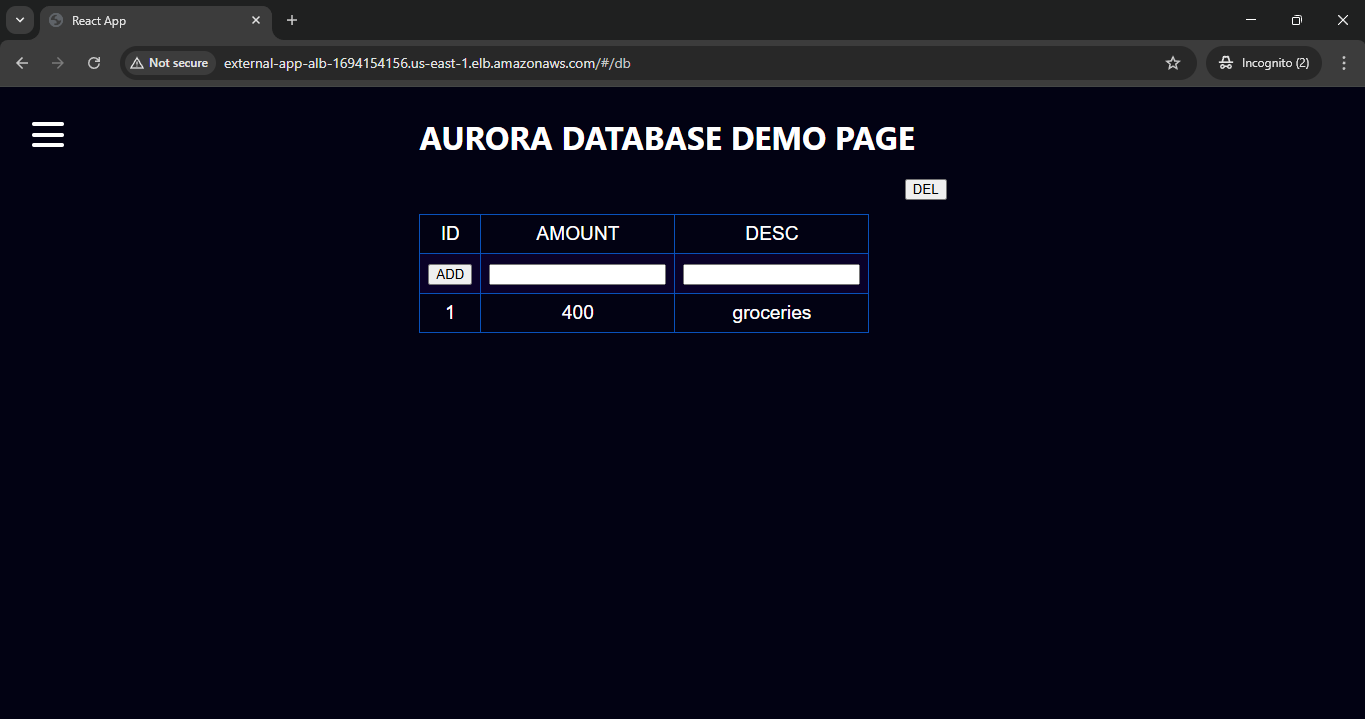
1. Select **Attach to an existing load balancer**
2. Choose **Application Load Balancer**
3. Select **external-app-alb**
4. Select the **target group** you created for the Web Tier (e.g. Web-target-group)







Click **Next,** then **finish** the creation. Once the auto-scaling group created copy the external load balancer Domain name and paste it into browser and press enter. Output must be the web page that is just configured. The output look like this



The request may still in HTTP. But application working fine.

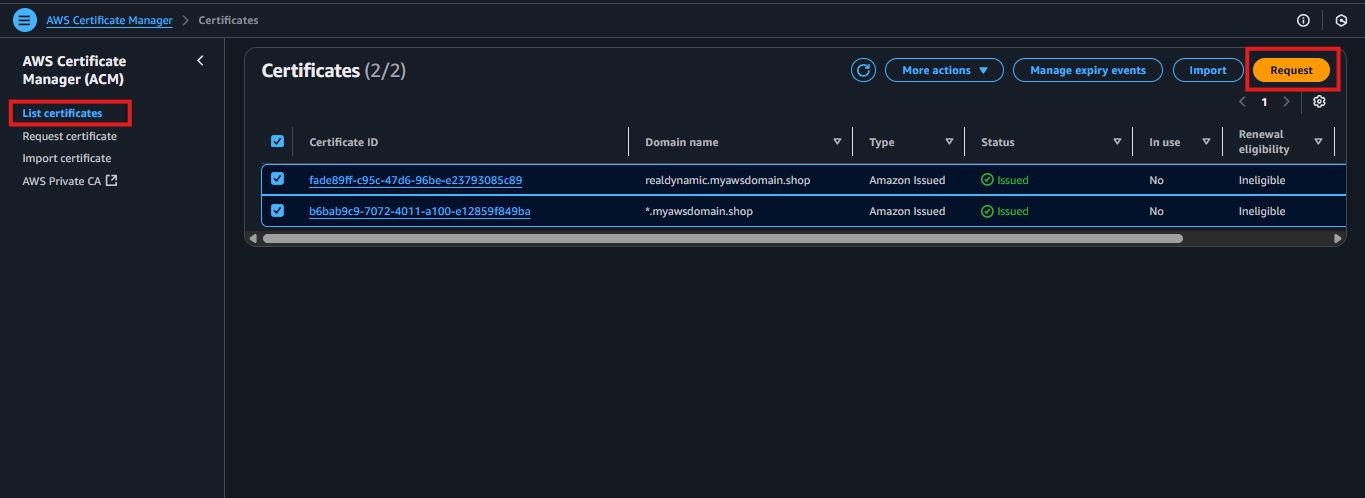
**🌐 Step - 13 Configure CloudFront → External Load Balancer (with SSL)**

**Step 1 – Prepare Your External Load Balancer Domain**

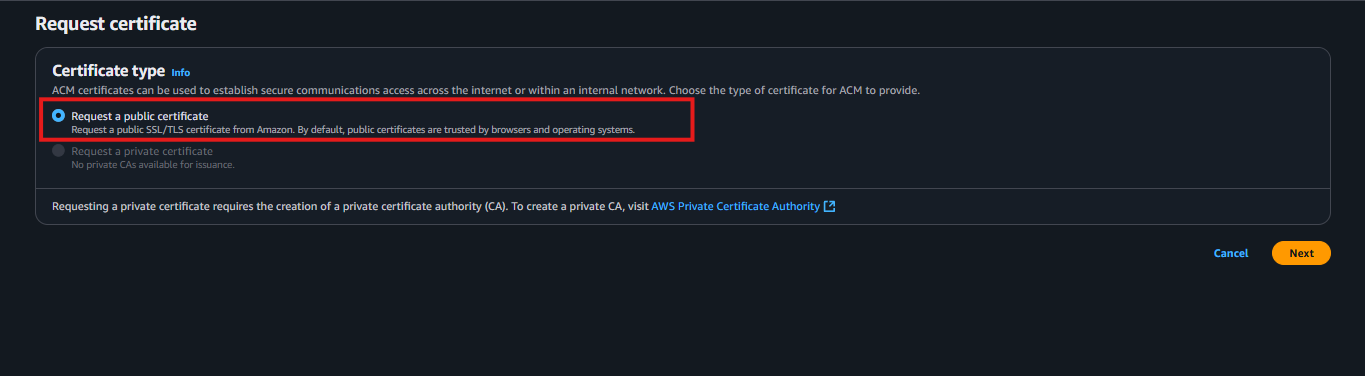
Make sure you have the DNS name of your **External Application Load Balancer** (e.g.  
external-web-alb-1234567890.us-east-1.elb.amazonaws.com).

**Step 2 – Create an SSL Certificate in ACM**

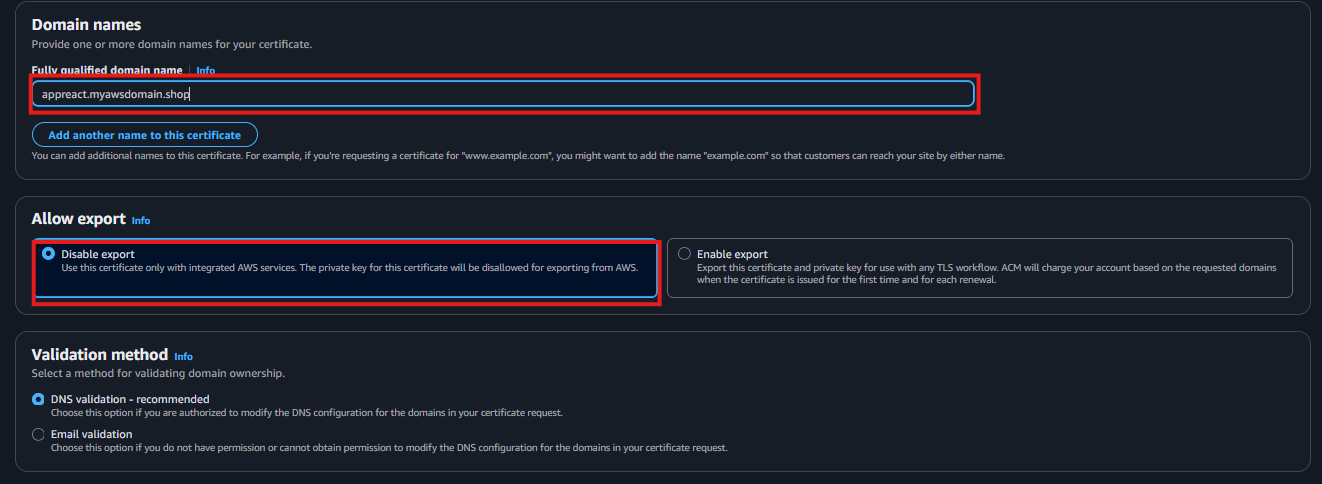
1. Navigate to **AWS Certificate Manager (ACM)** in **N.Virginia where your CloudFront distribution will run** (typically us-east-1).



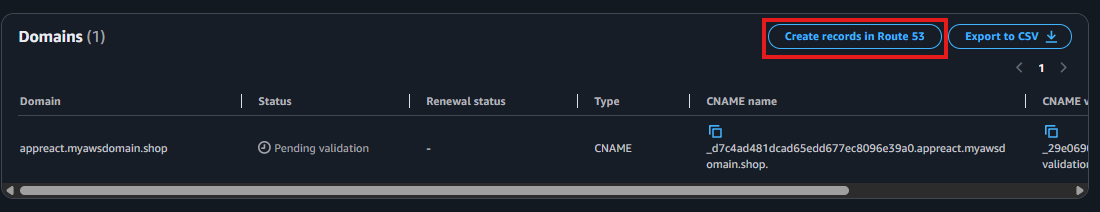
1. Click **Request a certificate → Request a public certificate**

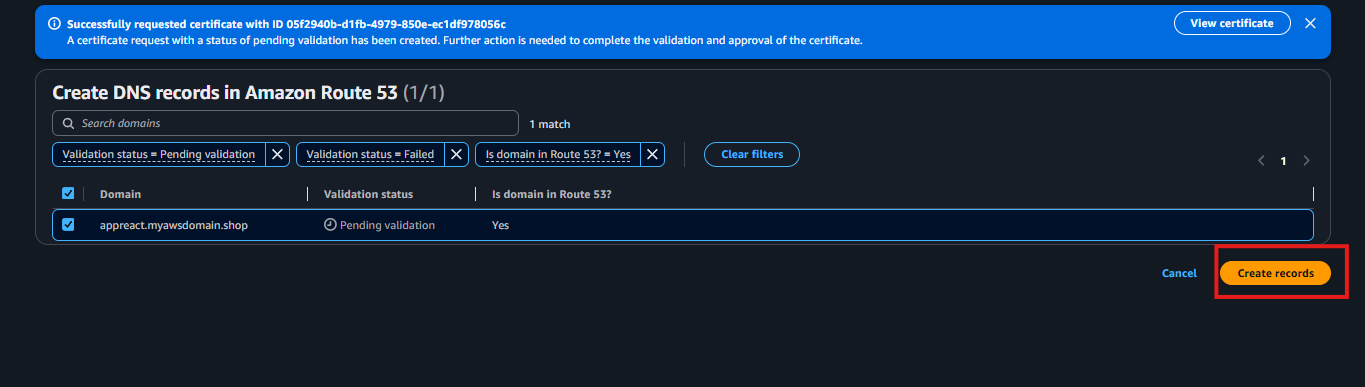


1. Enter your domain name (e.g. www.example.com or app.example.com)



1. Complete the validation by adding the provided **CNAME record** in Route 53

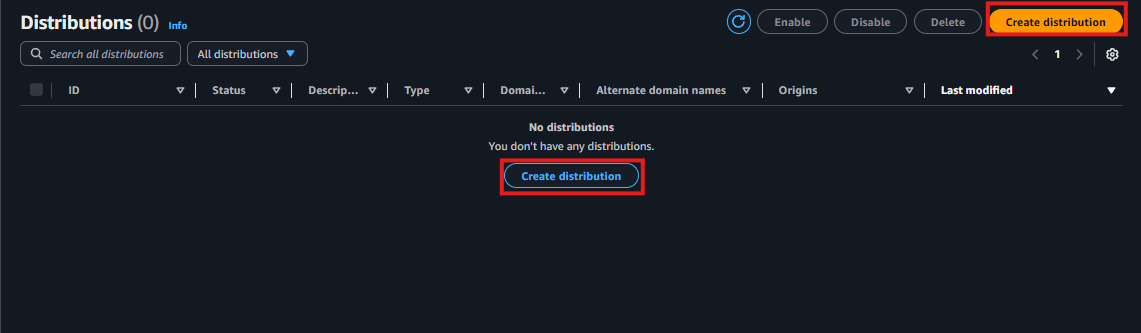


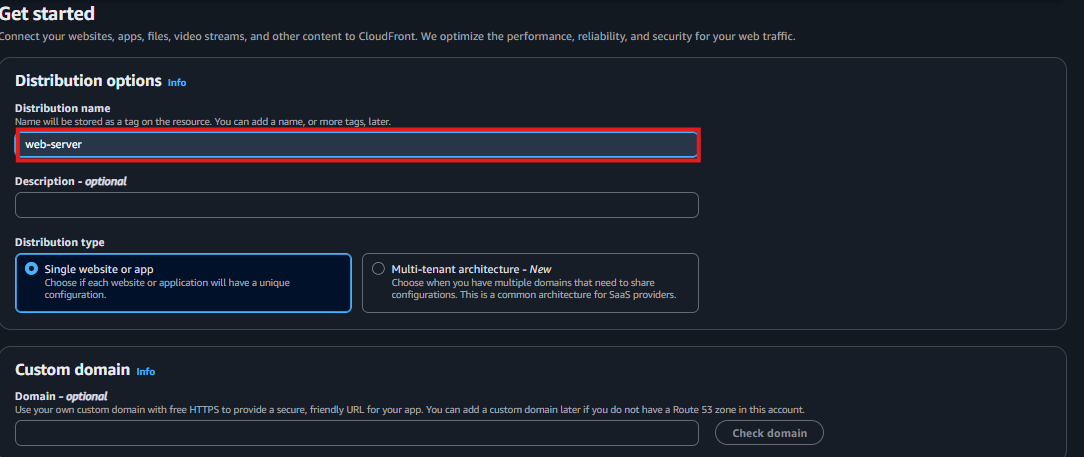


1. After validation, the certificate state becomes **Issued**

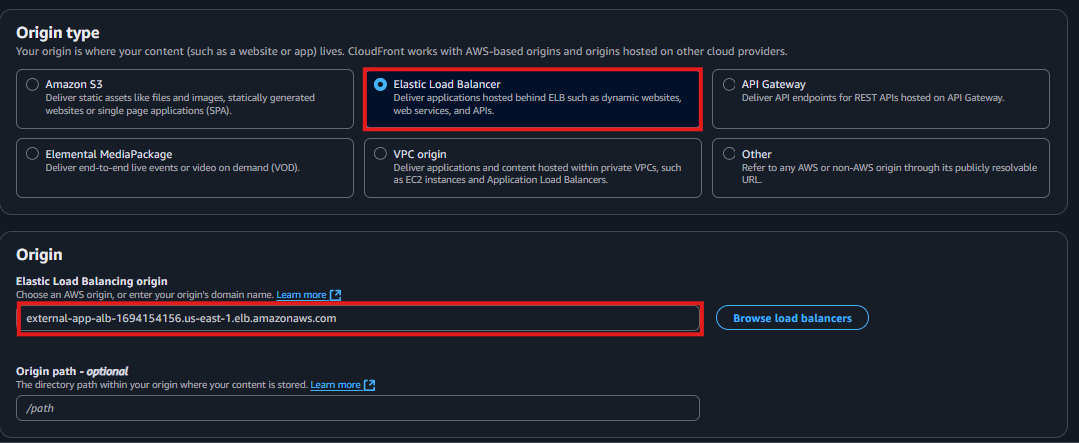
**Step 3 – Create CloudFront Distribution**

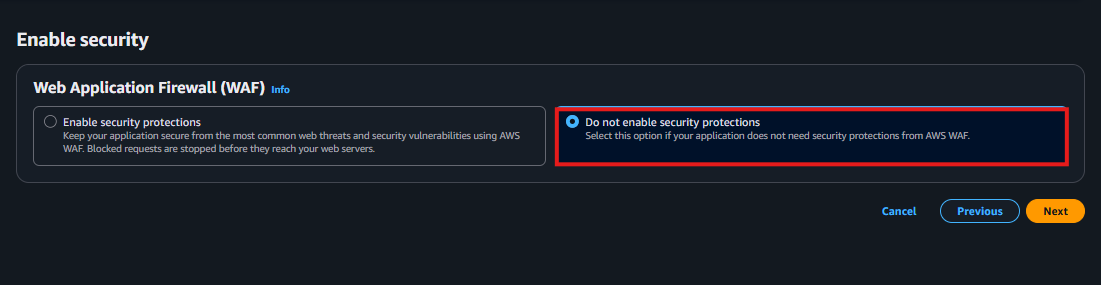
1. Open the **CloudFront console** and click **Create Distribution**



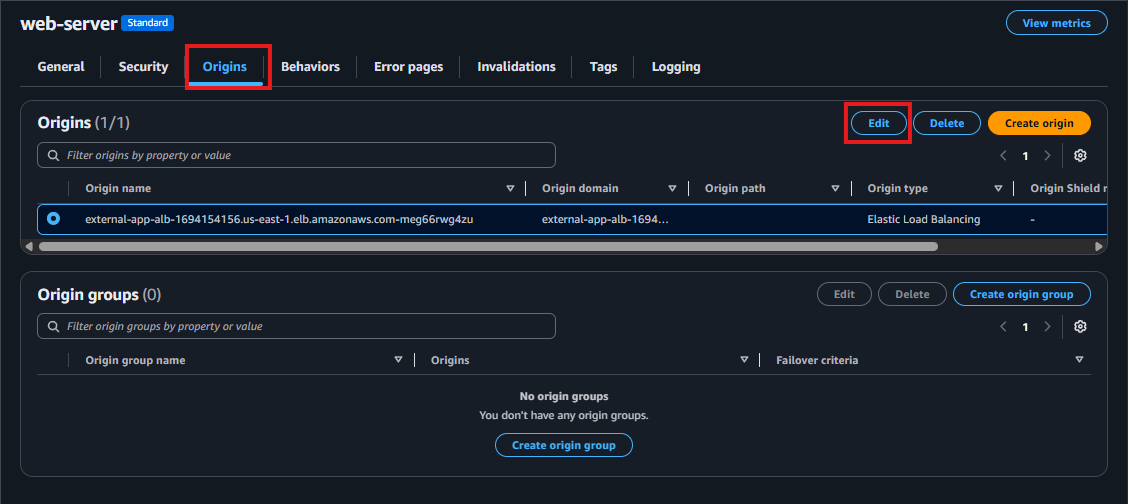


1. Under **Origin** configuration specify:

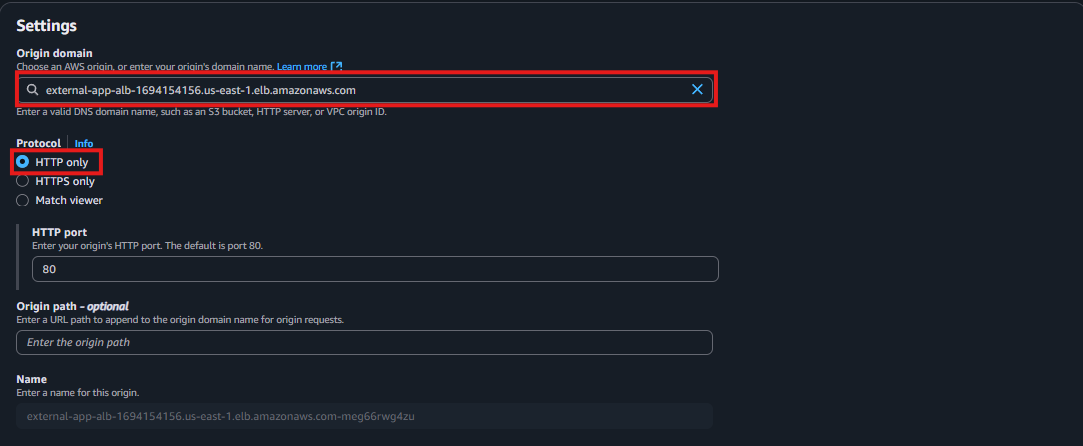




* Create a **Distribution**. Then go to **“Origins”** and click **“Edit”.**



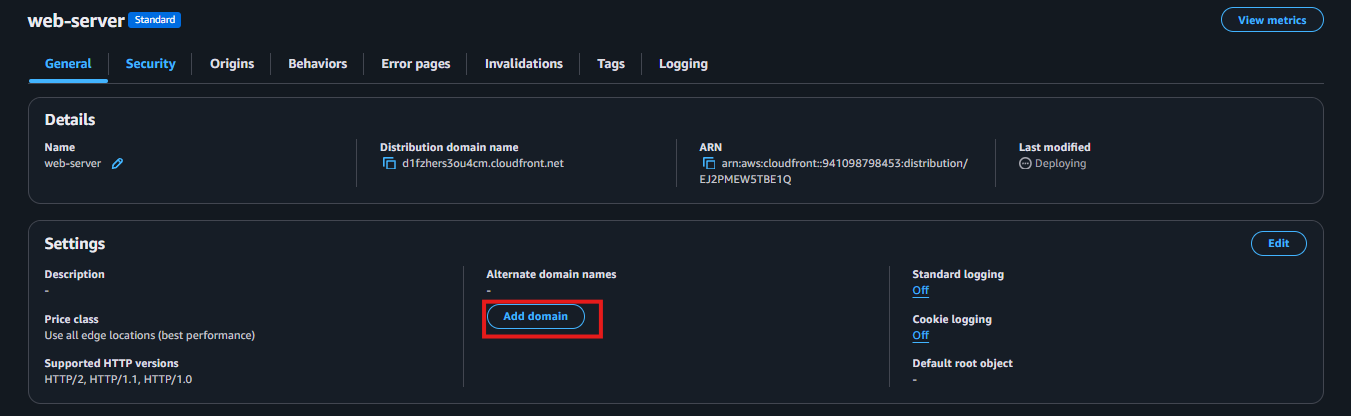
* Select port for the Origin domain in this external load balancer’s .

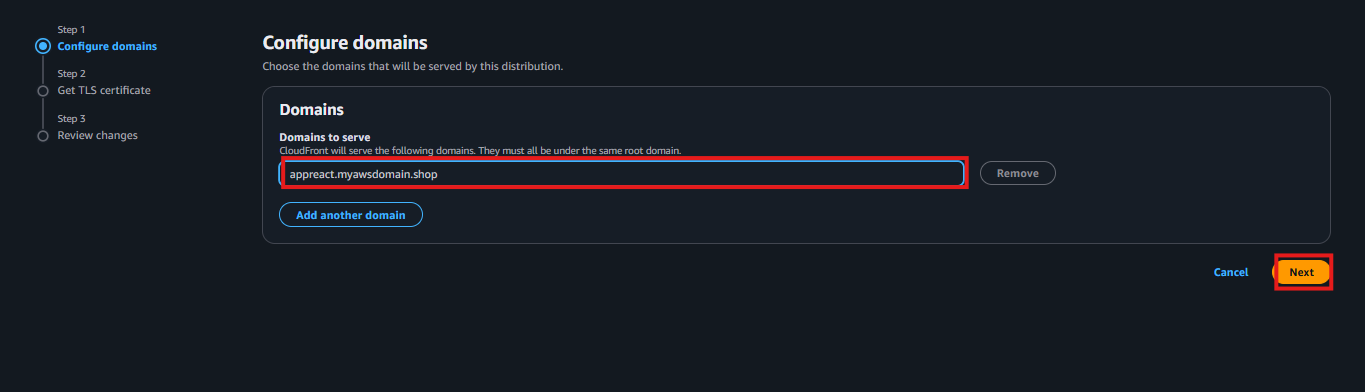


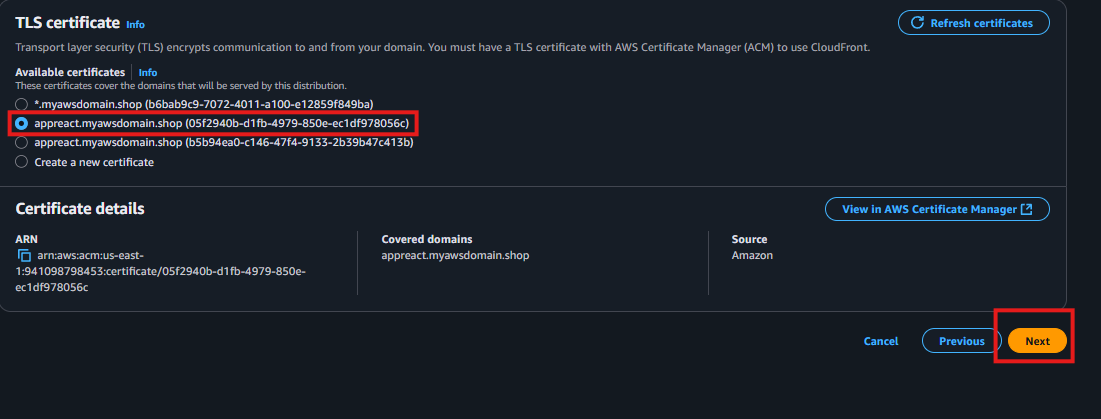
* Finally click on **“Save Changes”.**

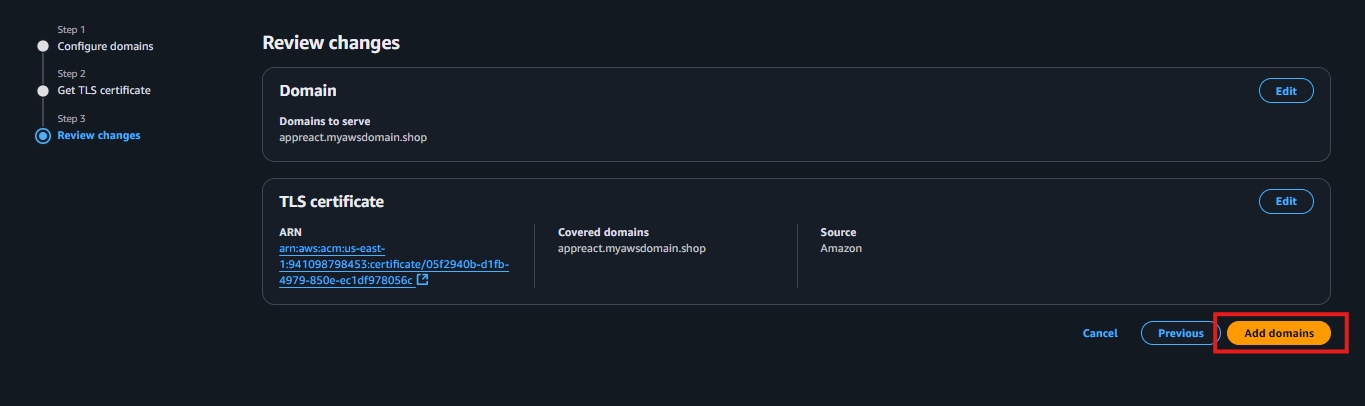
| **Field** | **Value** |
| --- | --- |
| Origin domain | <external-web-alb-dns> |
| Origin protocol | **HTTPS only** (recommended) |
| Origin path | *leave blank* |
| Origin ID | external-alb-origin |

* Once distribution created click on add domain enter your SSL certificate name just before created. Then click next and select your certificate finally press add domain.



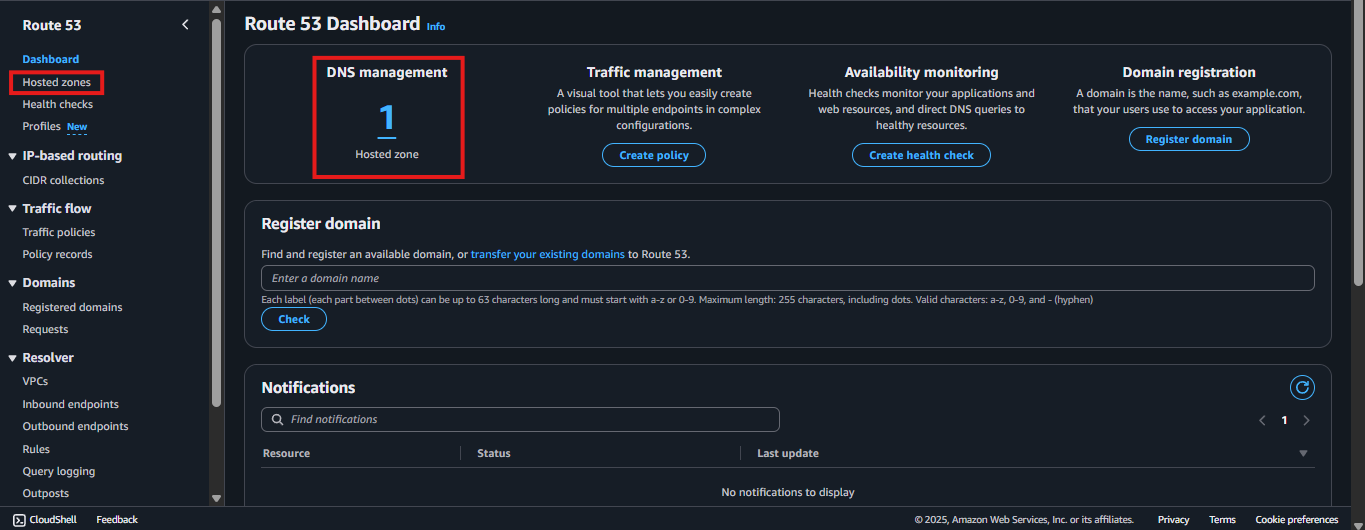




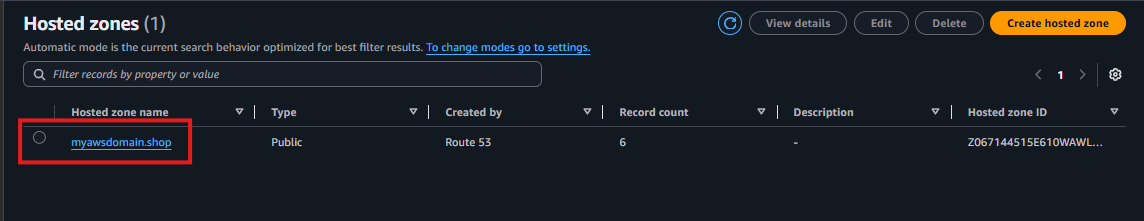


**Step – 14 Final Step – Create DNS Record in Route 53**

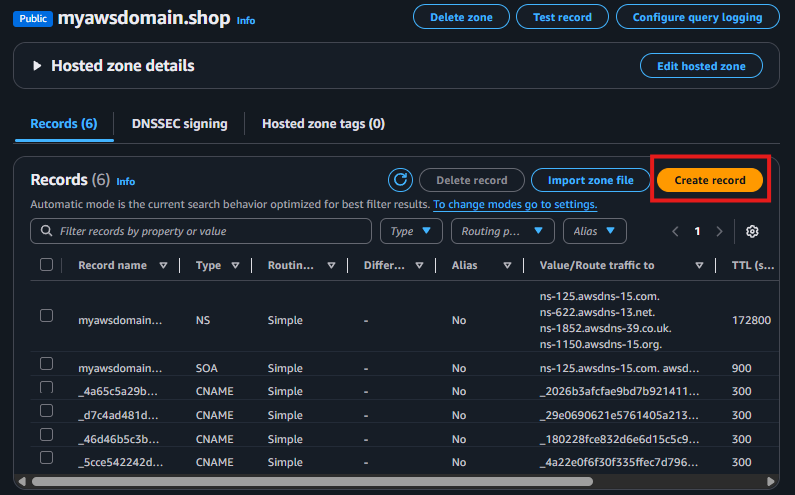
 Open the Route 53 console → Hosted Zones



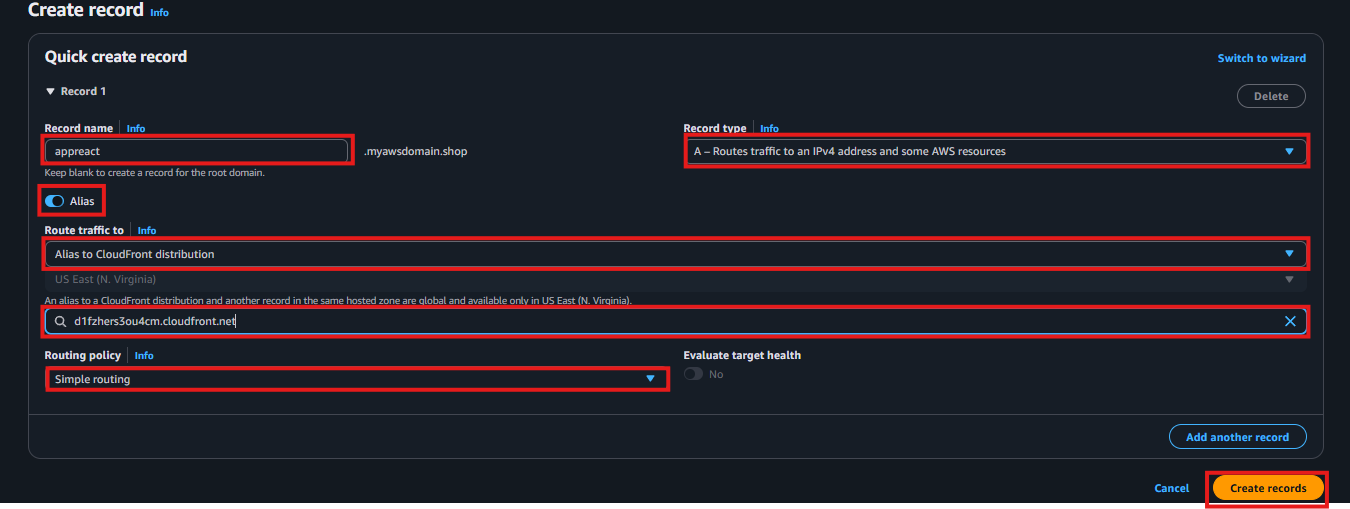
 Select the hosted zone for your domain (e.g. example.com)



 Click Create record



 Enter the domain name (SSL certificate name) and record type, click the Alias button and enter the details finally click **“Create Records”.**



We have successfully deployed a complete **three-tier architecture** on AWS consisting of:

* **Web Tier** – Publicly accessible EC2 instances behind an **External Application Load Balancer**, serving the front-end and forwarding API requests to the application tier.
* **Application Tier** – EC2 instances hosted in private subnets and fronted by an **Internal Application Load Balancer**, processing business logic and API requests.
* **Database Tier** – A **MySQL Amazon RDS** instance running in private subnets, accessible only from the Application tier for enhanced security.

All layers are logically isolated using **custom VPC subnets** and protected with **security groups** following the least-privilege principle.  
Connectivity has been validated end-to-end, and the Web Server successfully retrieves data from the database through the Application Server, confirming that all tiers are working together correctly.

**The following screenshots confirm that the application is fully functional. The web tier successfully renders the front-end UI and retrieves data from the application and database tiers, verifying end-to-end connectivity across the entire three-tier architecture.**

