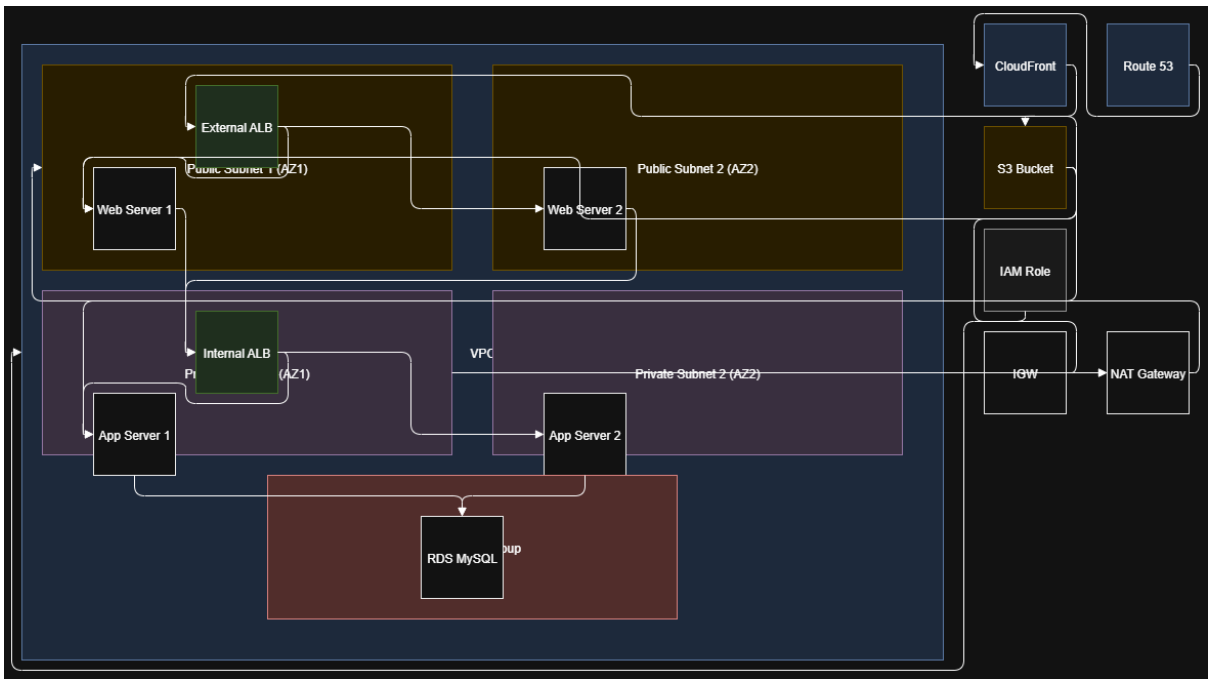
AWS 3-Tier Project (Webtier,Apptier,Dbtier)

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

Service / Tool	Purpose
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 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)

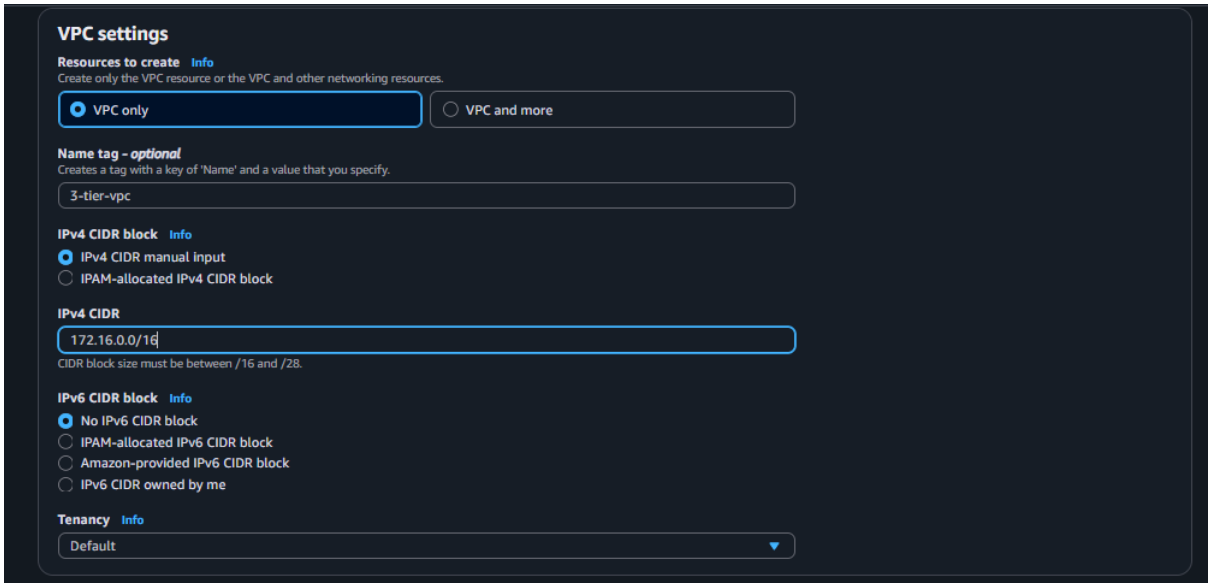
And clone the code from remote by ref below url :
git clone https://github.com/aws-samples/aws-three-tier-web-architecture-workshop.git

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**.



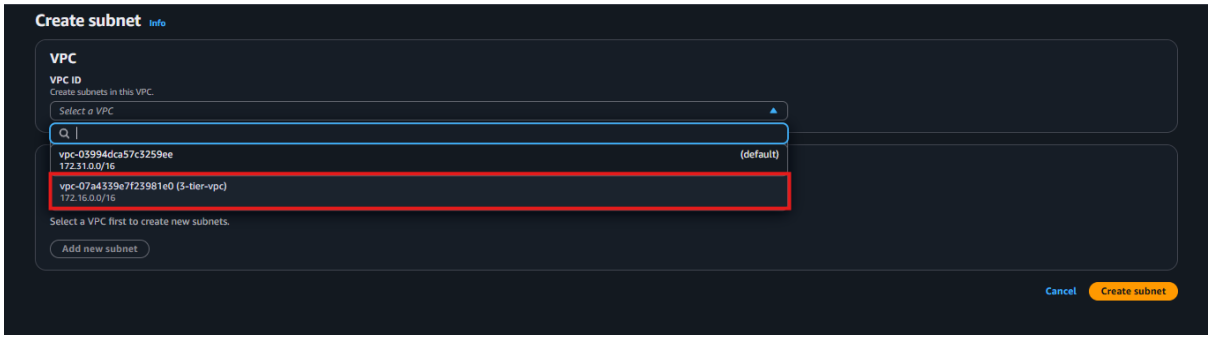
Step 2 – Create Subnets

1. In the VPC Dashboard, select **Subnets** and choose **Create subnet**.

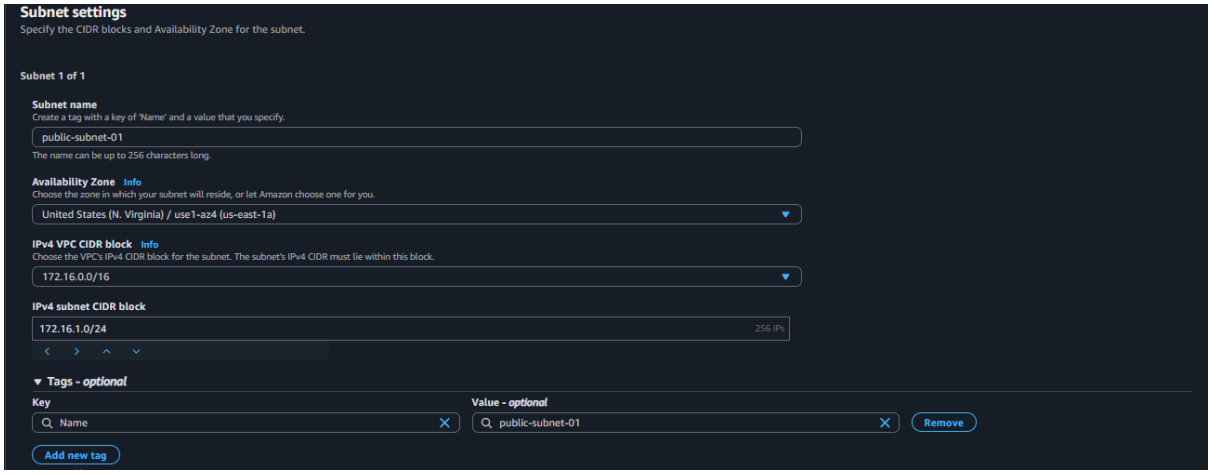
2. Add 2 public subnets (e.g., 172.16.1.0/24 and 172.16.2.0/24) and 2 private subnets (e.g., 172.16.3.0/24 and 172.16.4.0/24).

3. Ensure the subnets are spread across at least 2 Availability Zones.

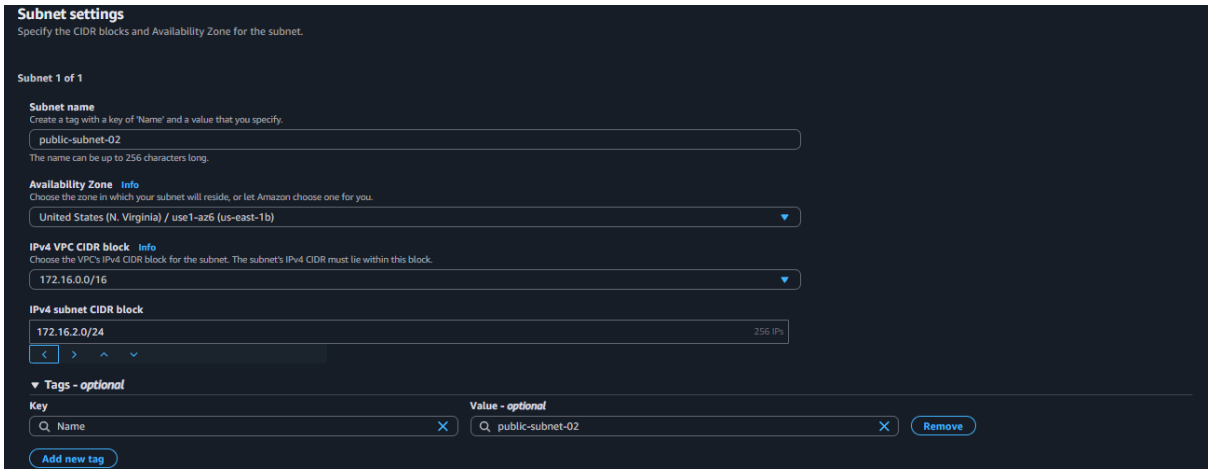
    - Select your custom VPPC



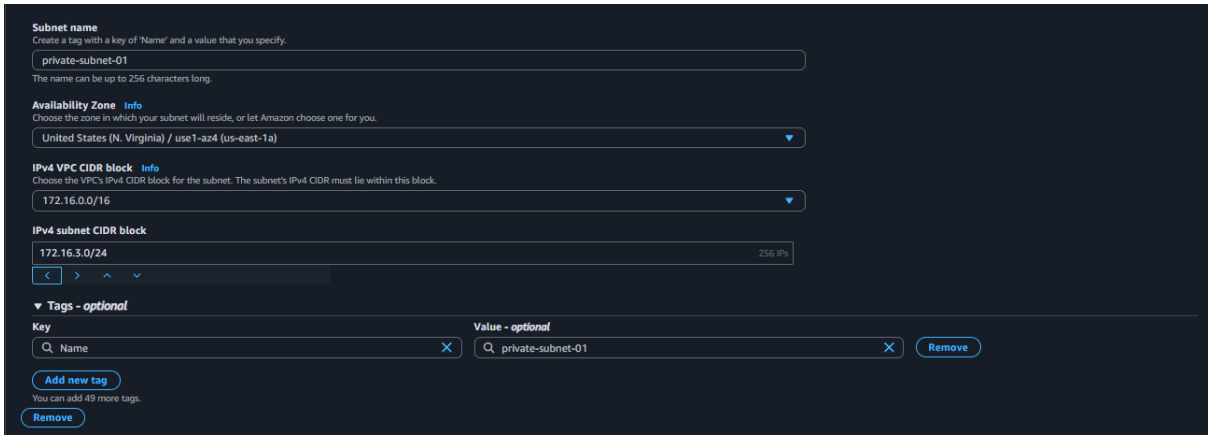
Subnet-1→Public-subnet



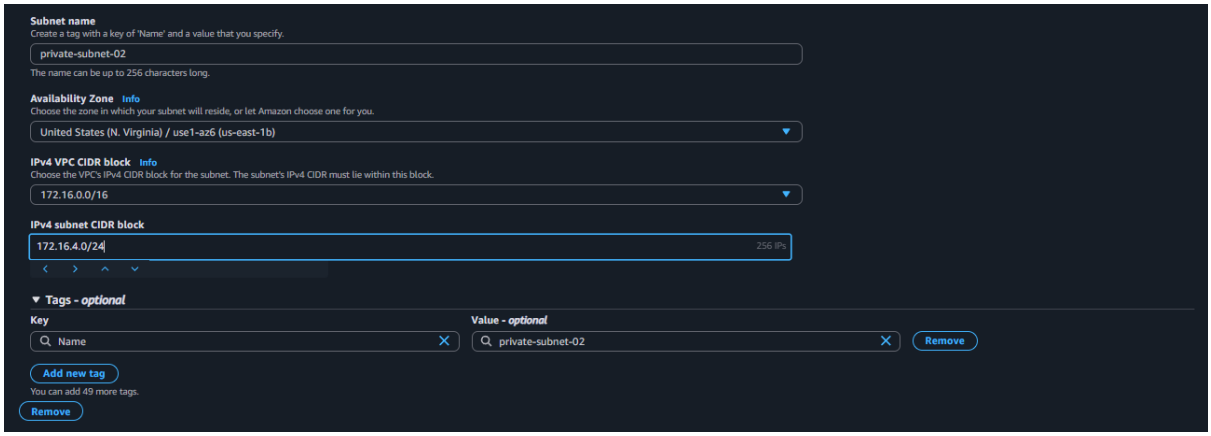
Subnet-2→Public-subnet

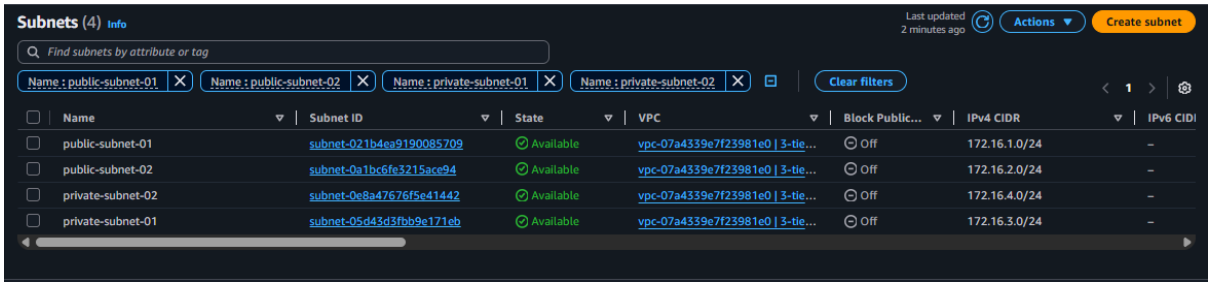


Subnet-3→Private-subnet



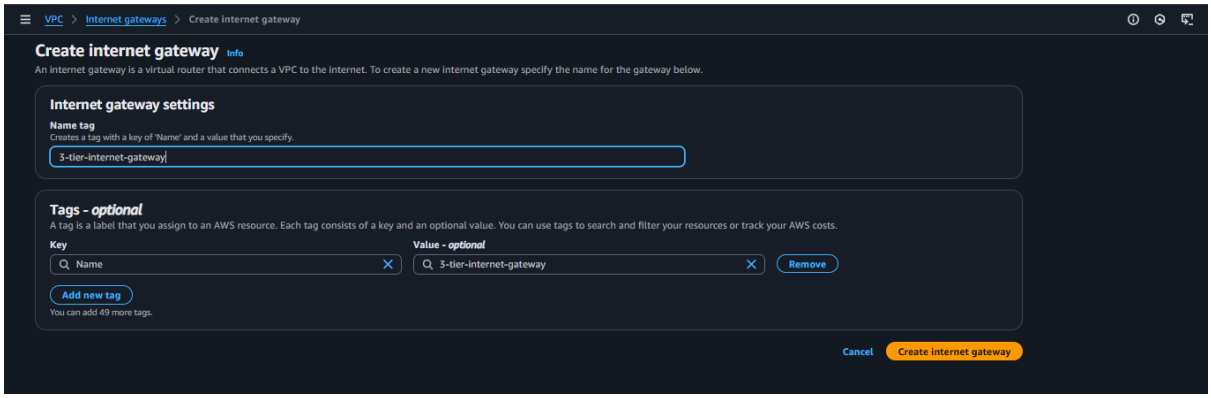
Subnet-4→Private-subnet -





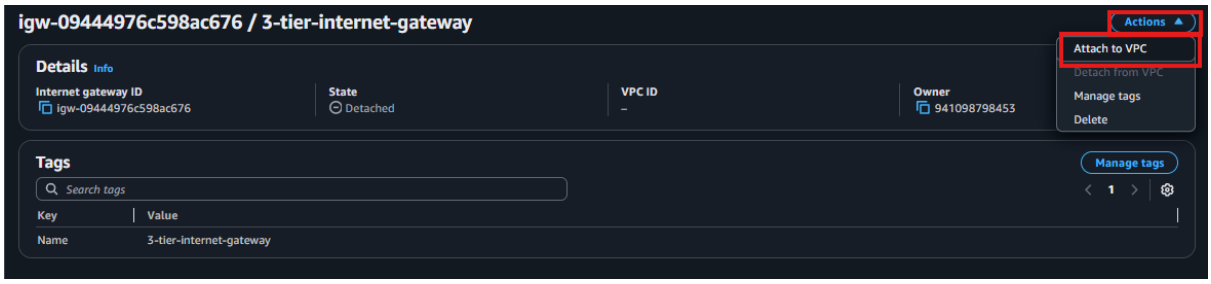
Step 3 – Configure Internet Gateway

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



2. Attach the Internet Gateway to your VPC.

    - In the action button select attach to VPC

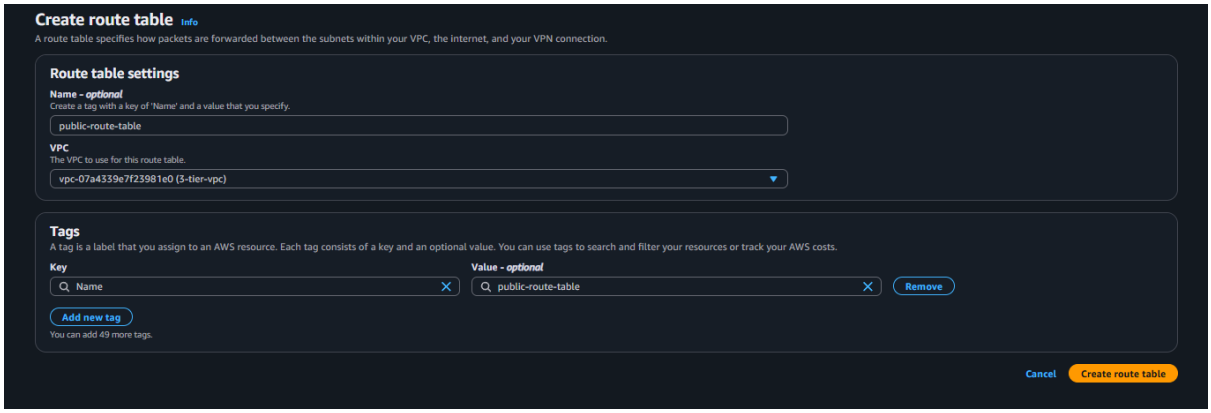


3. Select your custom VPC then click on “Attach internet gateway”

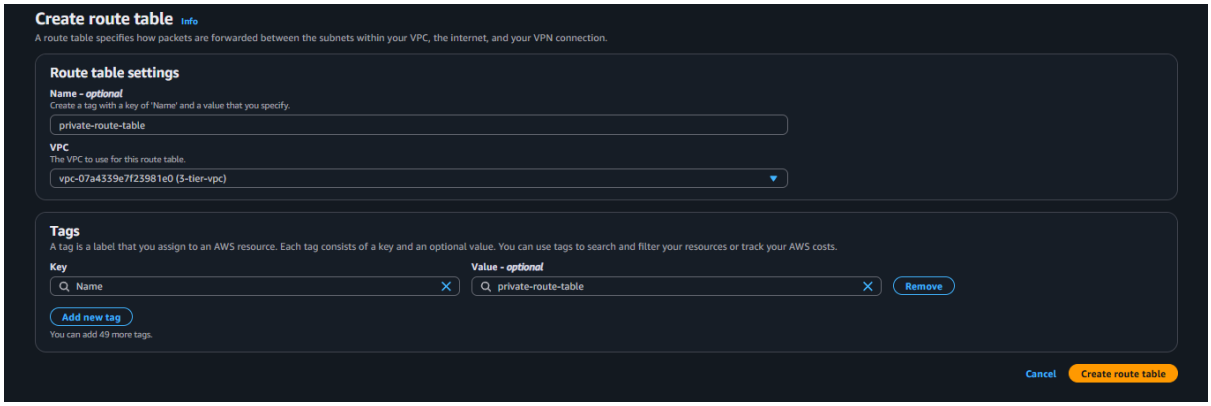
Step 4 – Configure Route Tables

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

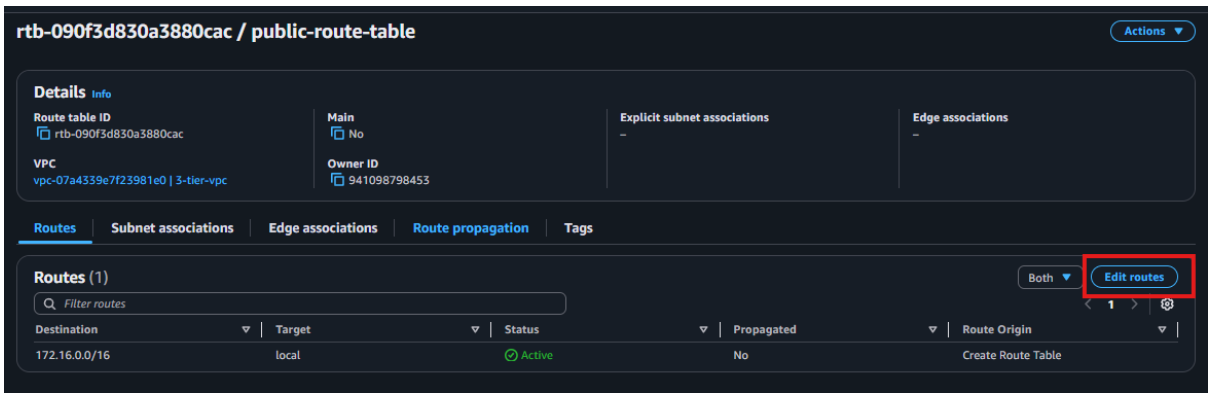
Public-Route-Table:



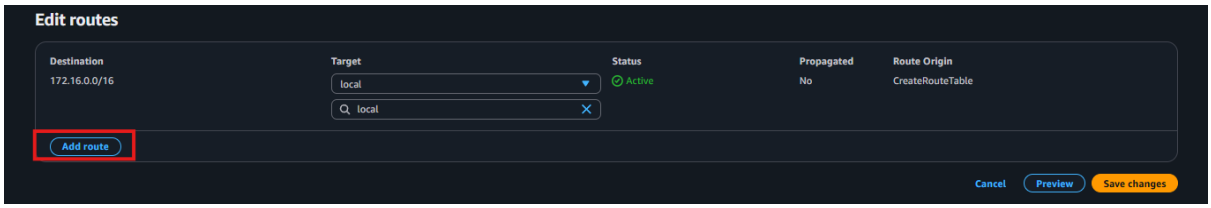
Private-Route-Table:

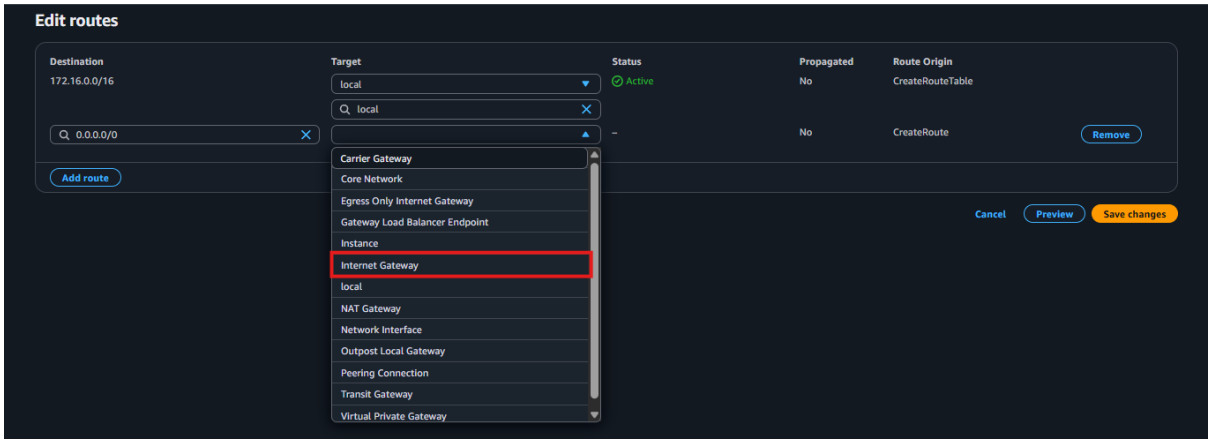


2. Add a route (0.0.0.0/0) to the public route table and target the Internet Gateway. Click on “Edit routes”

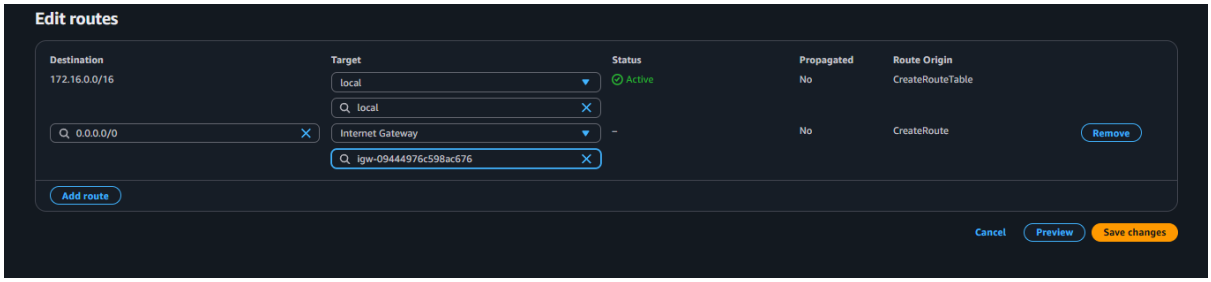


- Click on “Add route” button





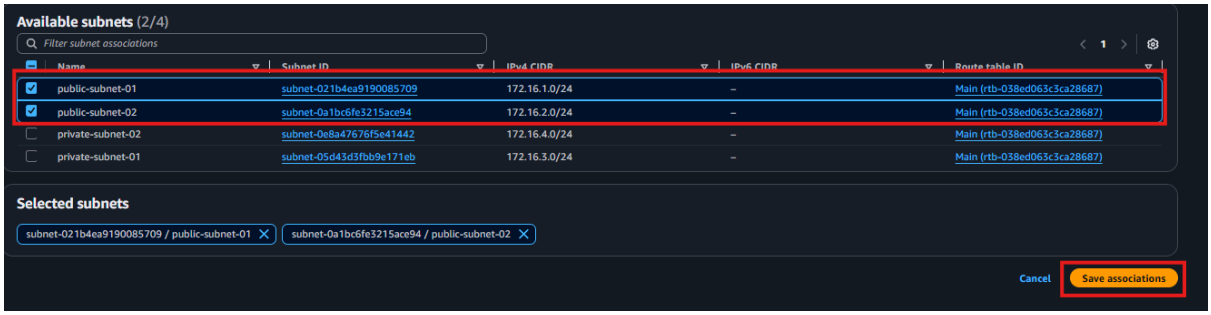
- Select your internet gateway and “Save changes”



3. Associate public subnets with the public route table.

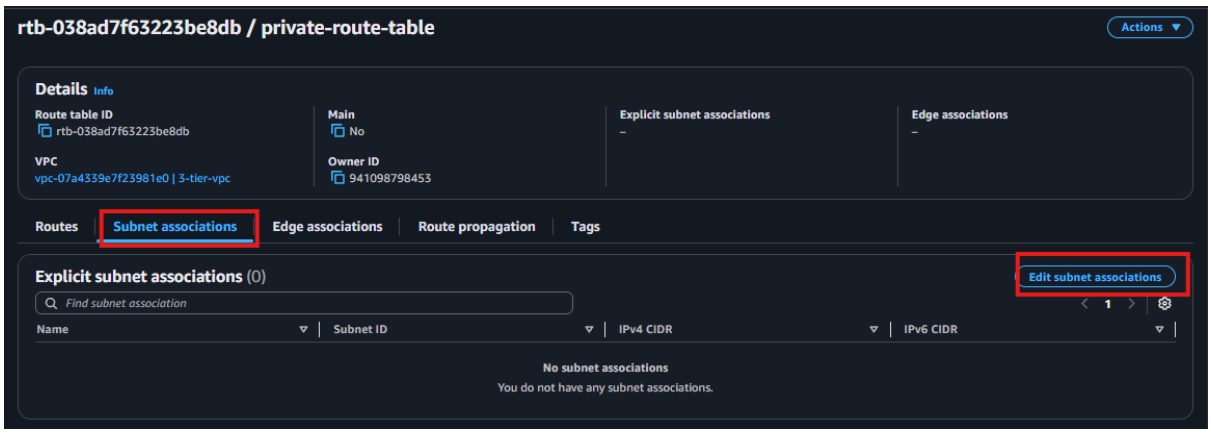
- Click “Subnet associations” then click “Edit subnet associations”

- Select the public subnets and click on “Save associations”

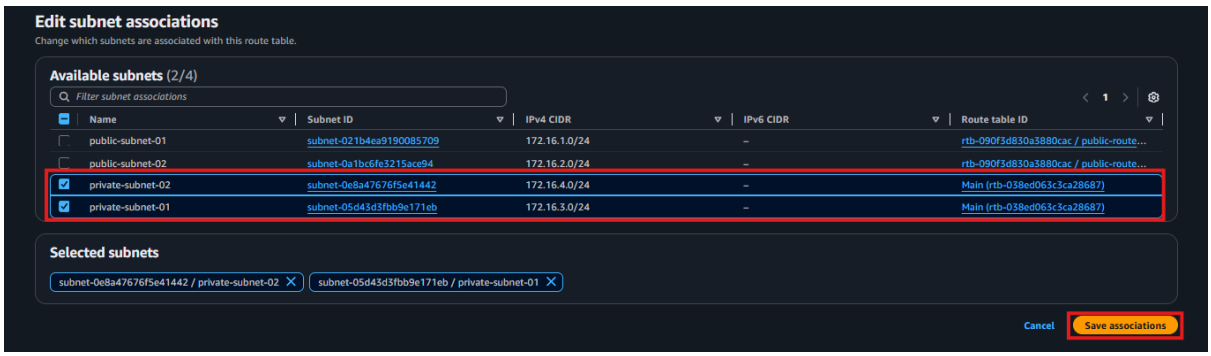


4. Associate private subnets with the private route table.

- Click “Subnet associations” then click “Edit subnet associations”



- Select the private subnets and click on “Save associations”

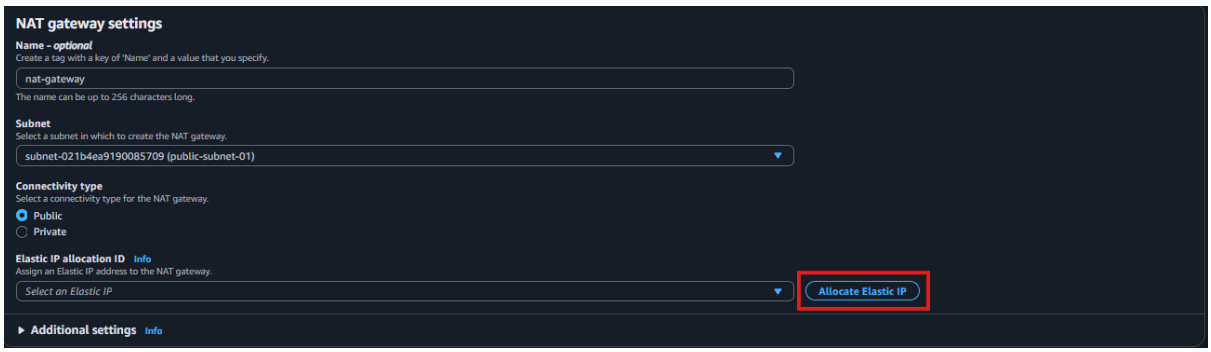


Step 4a – Create NAT Gateway

1. In the VPC Dashboard, select **NAT Gateways** and click **Create NAT Gateway**.

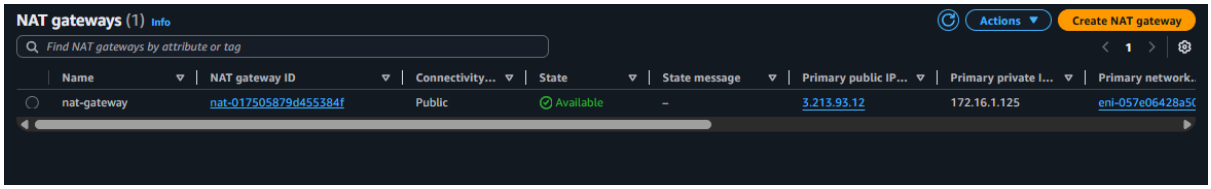
2. Select one of the public subnets and associate an Elastic IP address.

- Then click on “Allocate Elastic IP”

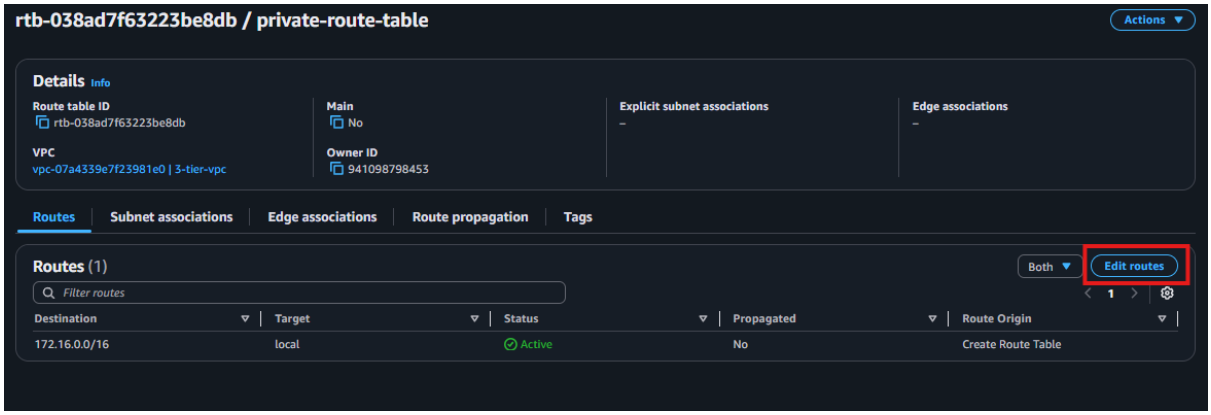


- And click the “Save changes”

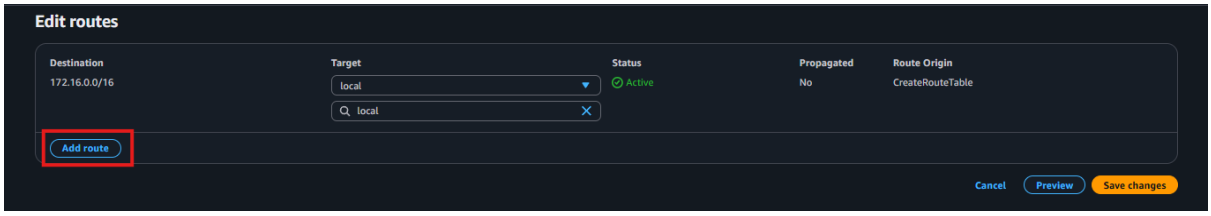
3. Create the NAT Gateway and wait until it becomes available.

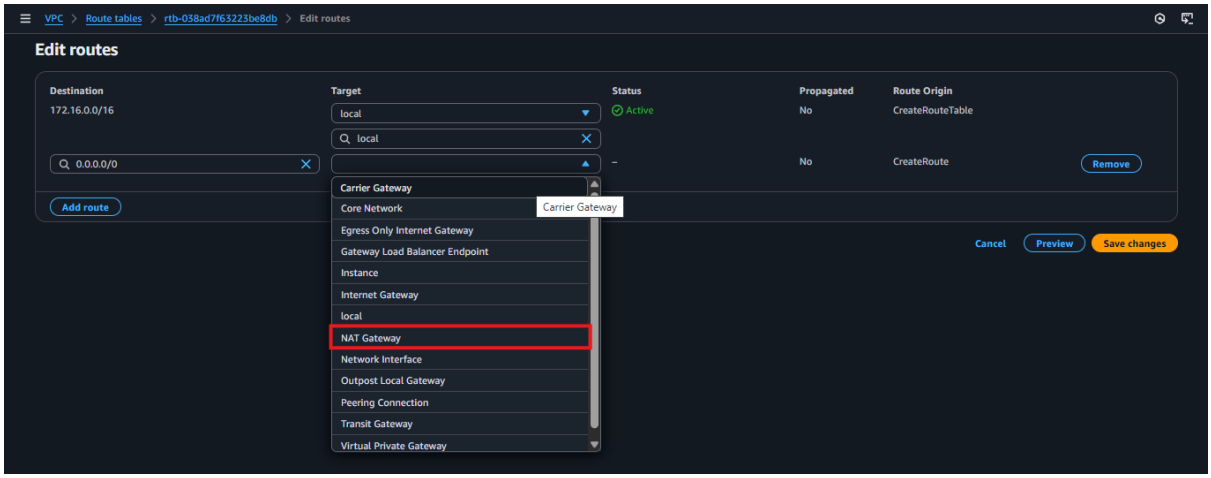


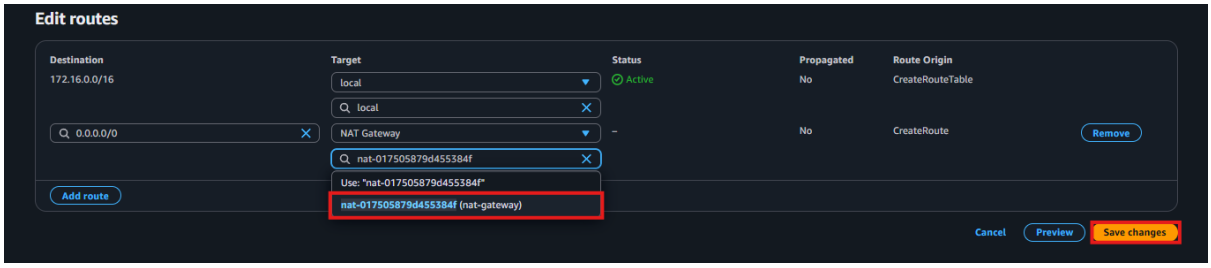
4. In the private route table, add a route (0.0.0.0/0) and target the new NAT Gateway.



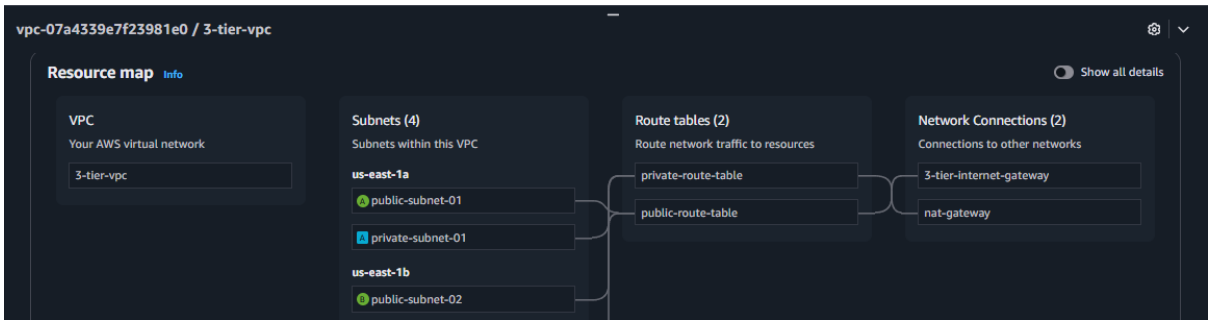
- Click on “Add route” and select “NAT Gateway” then in the drop down select the NAT we have just created. And enter the “Save changes”







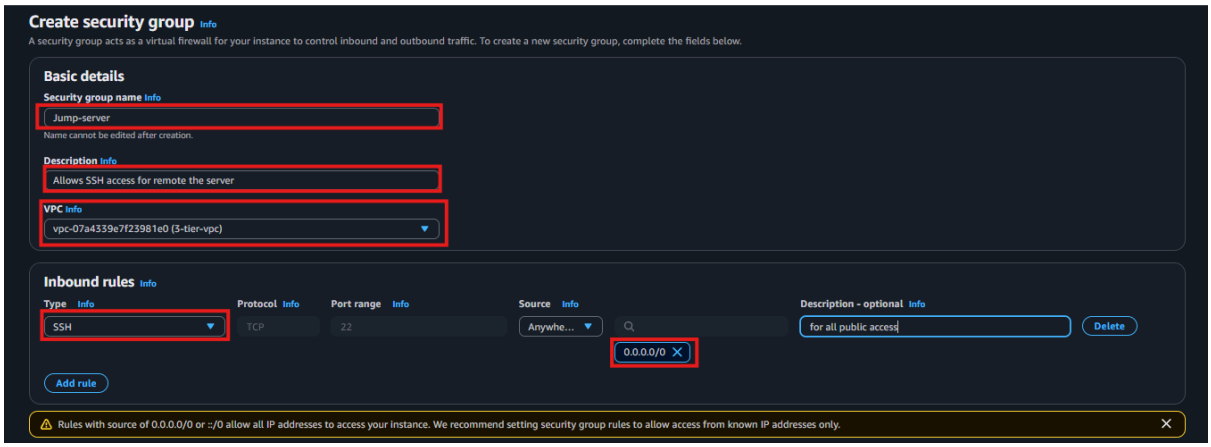
- Finally your VPC → **Resource map** look like this we are done with “Network part”.



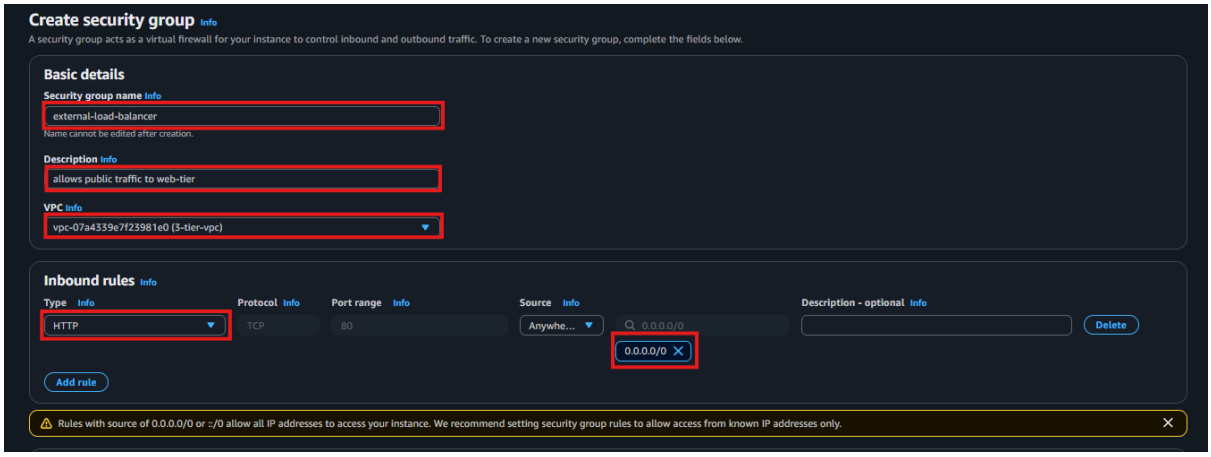
Step 5 – Create Security Groups

Create the following six custom security groups:

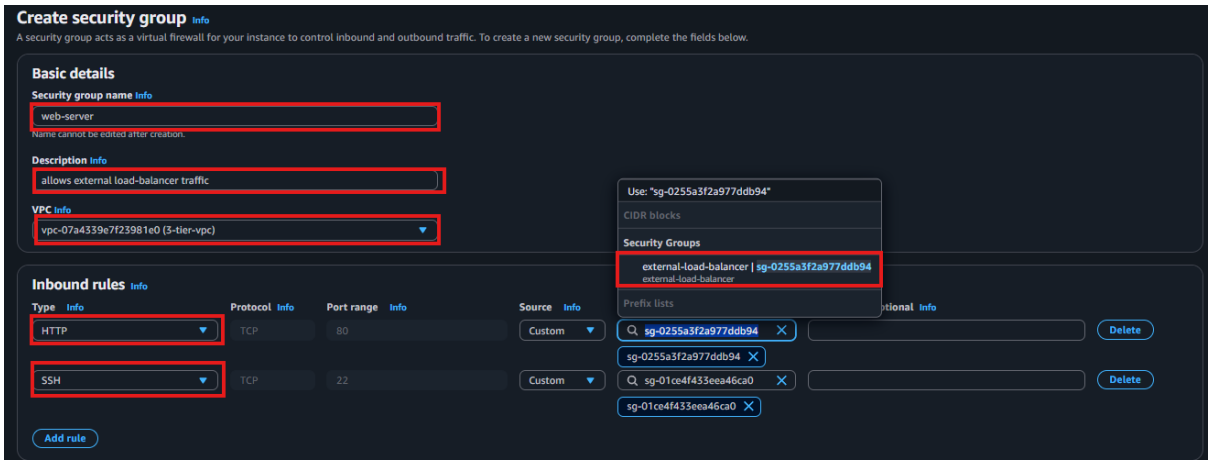
1. Jump Server SG – allows SSH (port 22) from all public IP addresses.



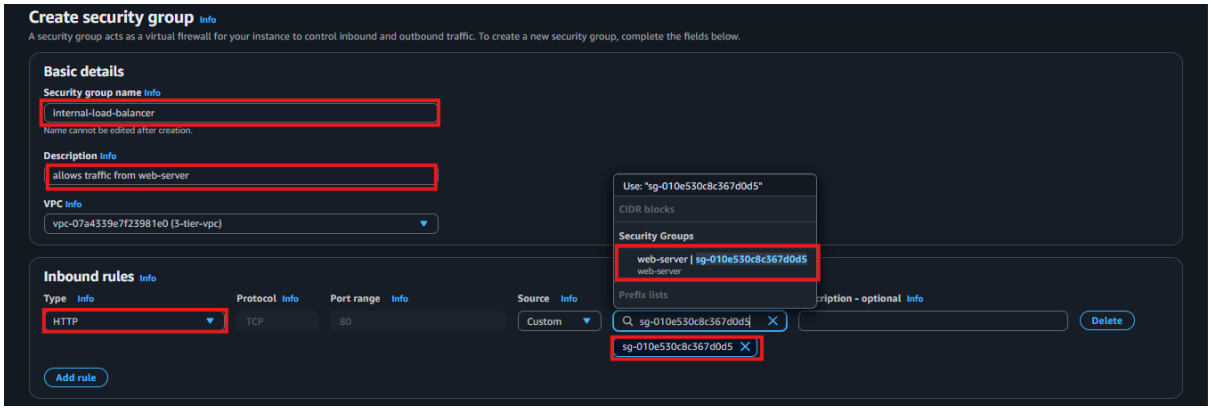
2. External Load Balancer SG – allows HTTP (port 80) from all public IP addresses.



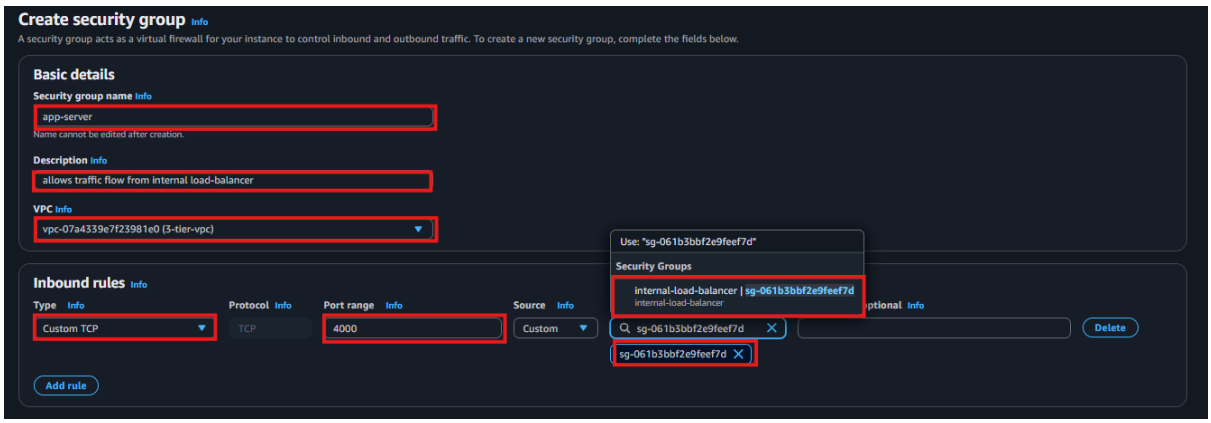
3. Web Server SG – allows HTTP (port 80) from the External Load Balancer SG and SSH (port 22) from the Jump Server SG.

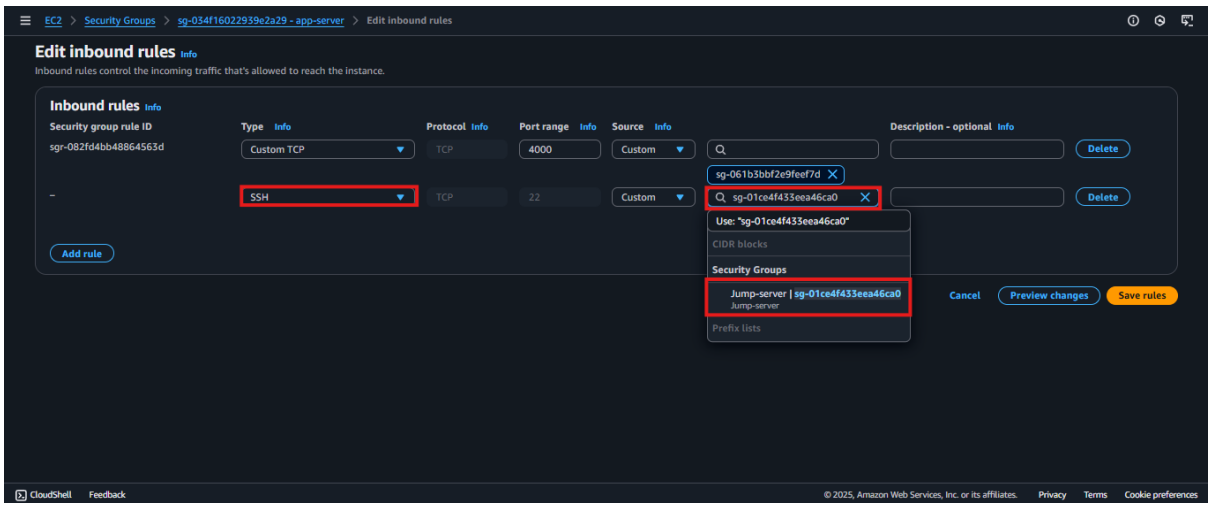


4. Internal Load Balancer SG – allows HTTP (port 80) from the Web Server SG.

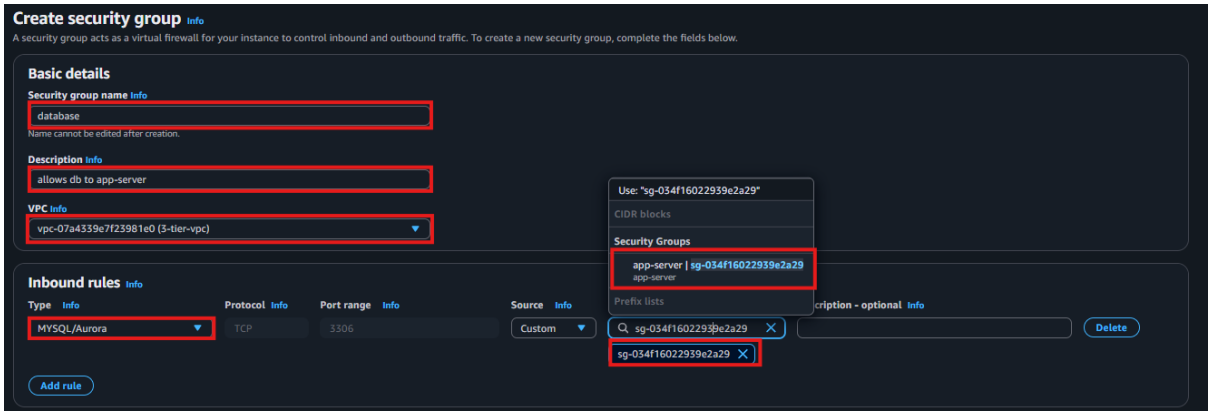


5. App Server SG – allows port 4000 from the Internal Load Balancer SG. And Allow port 22 from the Jump-server SG for remote access from jump-server to app-server





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

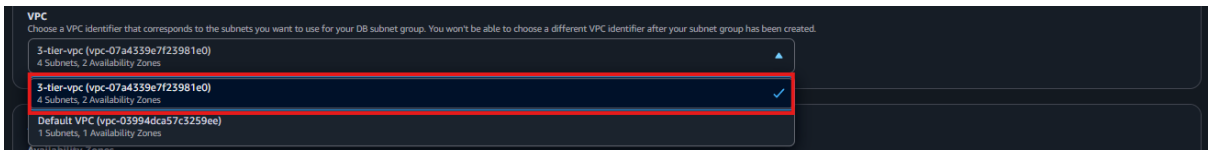


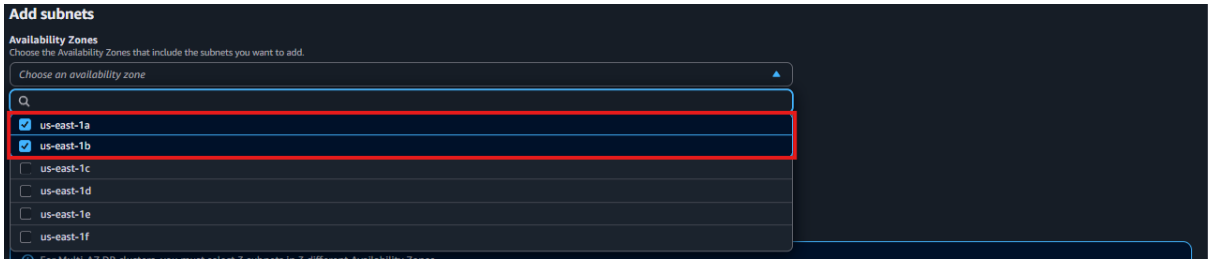
Step 6 – Create RDS Subnet Group

1. In the RDS dashboard, select Subnet groups and click Create DB subnet group.

2. Provide a name and description.

3. 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”.

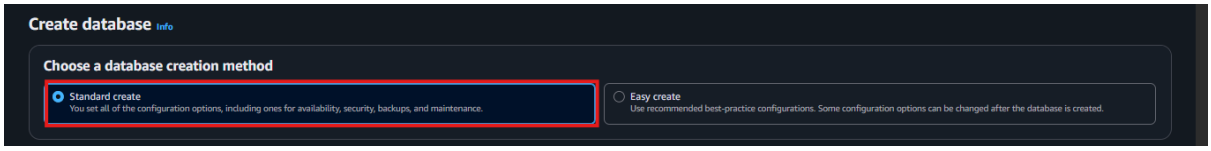




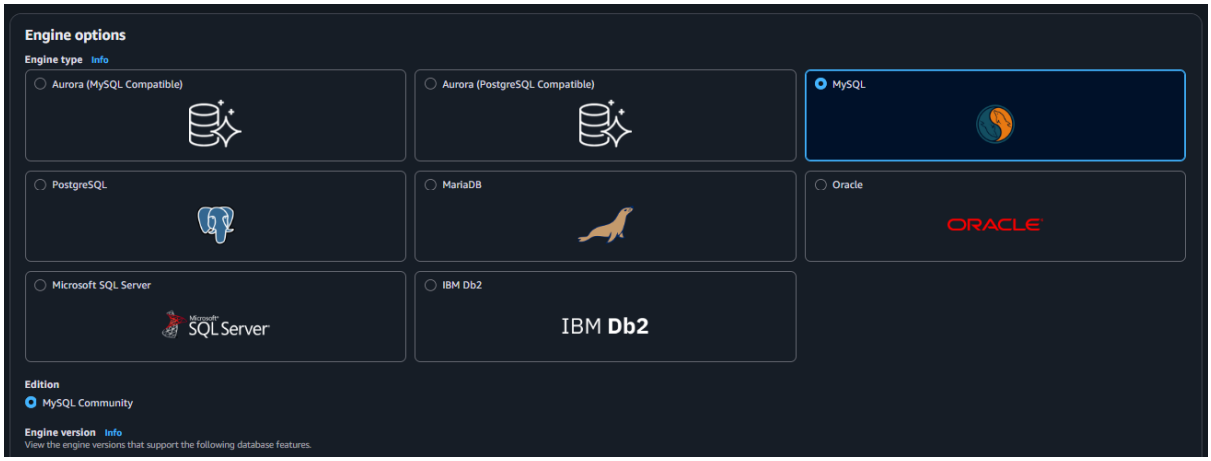
Step 6a – Configure RDS Database

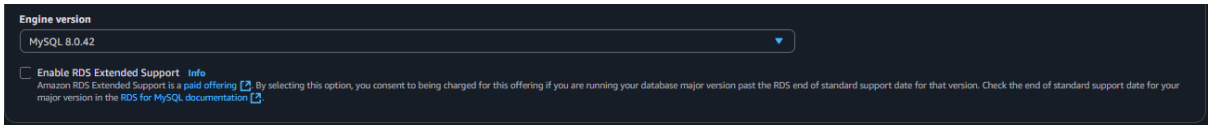
1. In the AWS Console, search for **RDS** and select **Create Database**.

2. Choose Standard Create and select the **Free Tier** options.

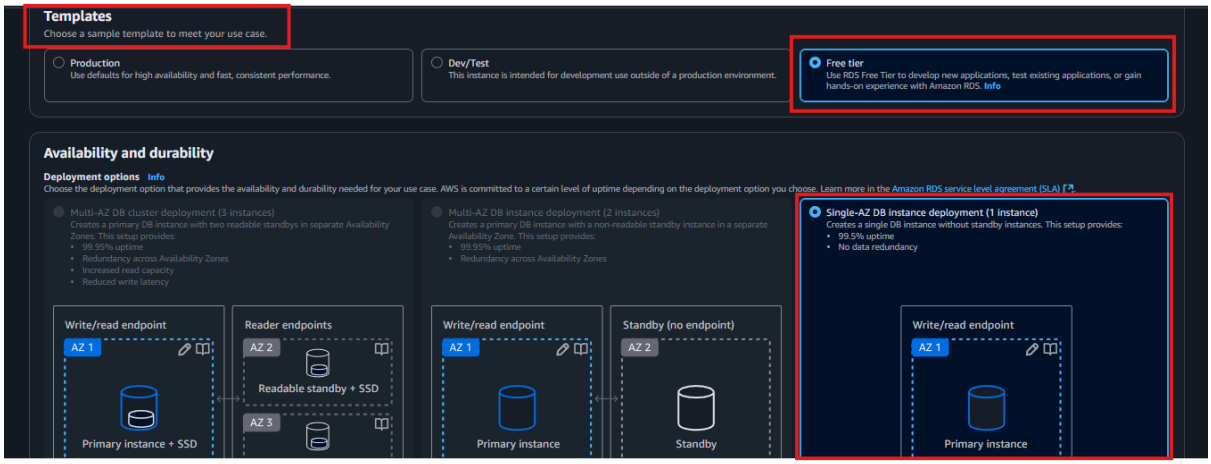


3. Choose **MySQL** as the engine type



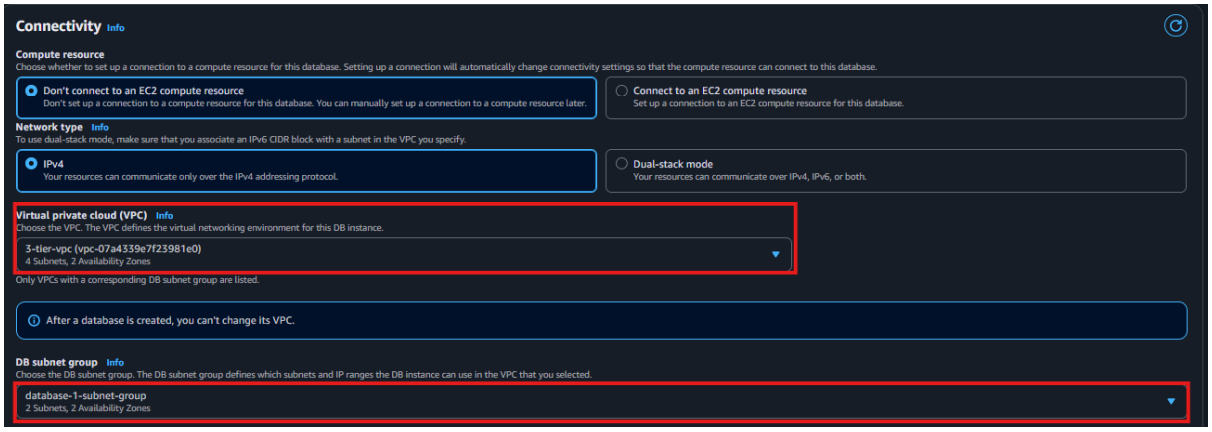


- In the “Templates” chose free tier.

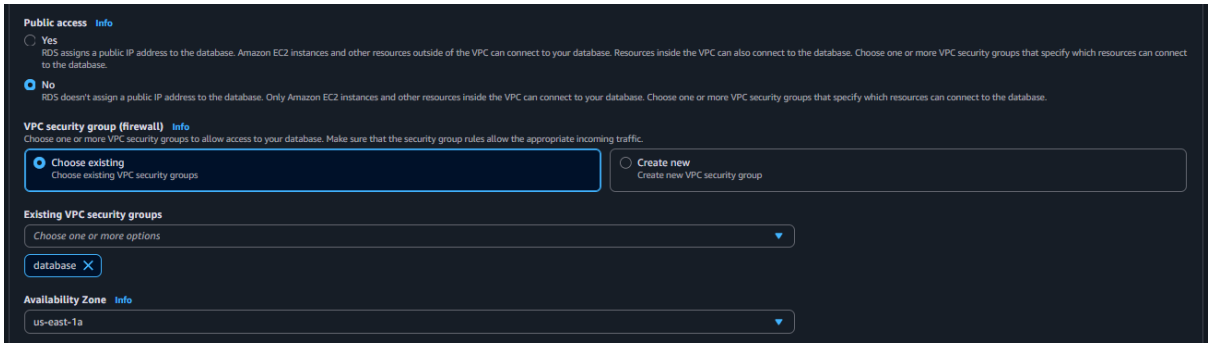


- configure DB name, username, and password.

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



4. 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).

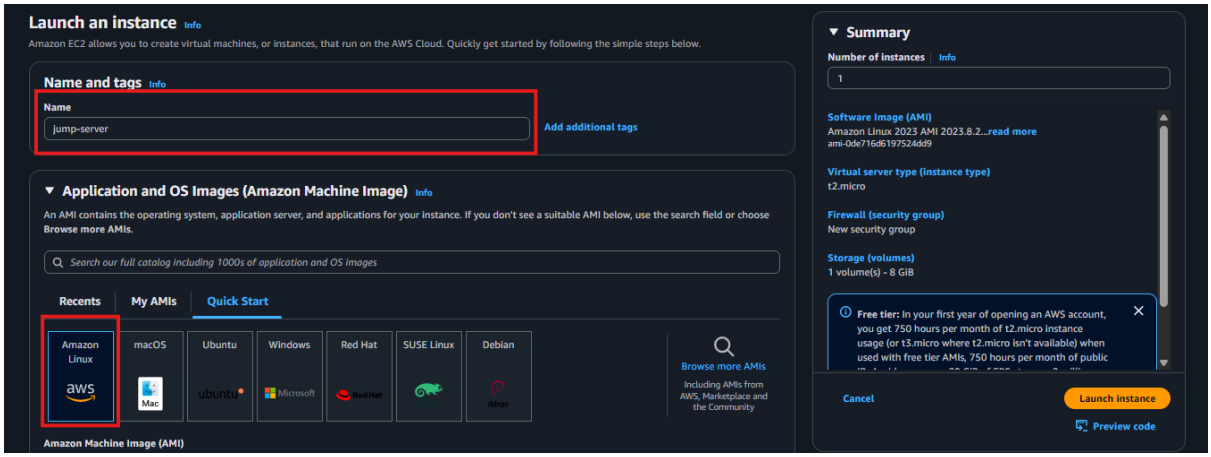


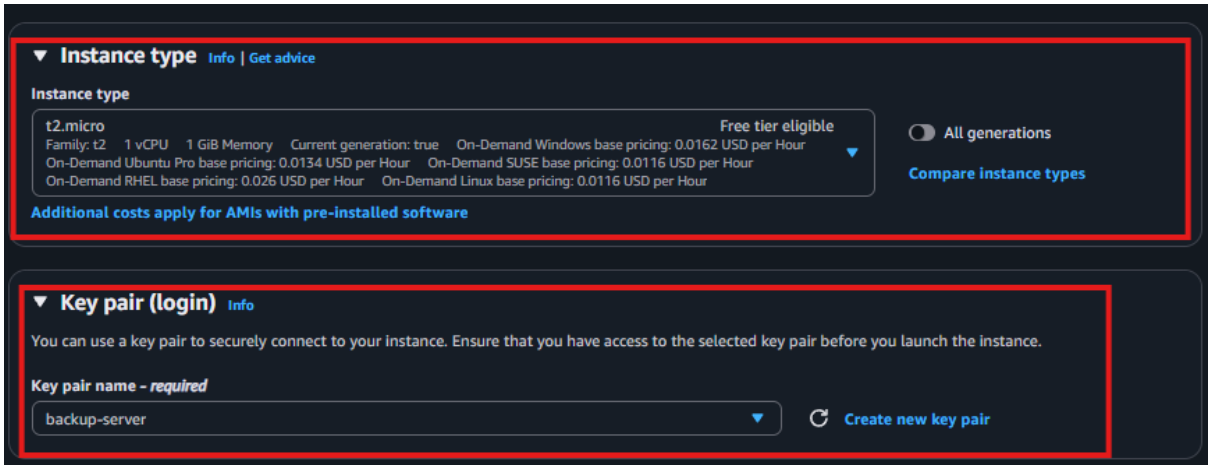
- Then Create database.

Step 7 – Launch Jump Server (Bastion Host)

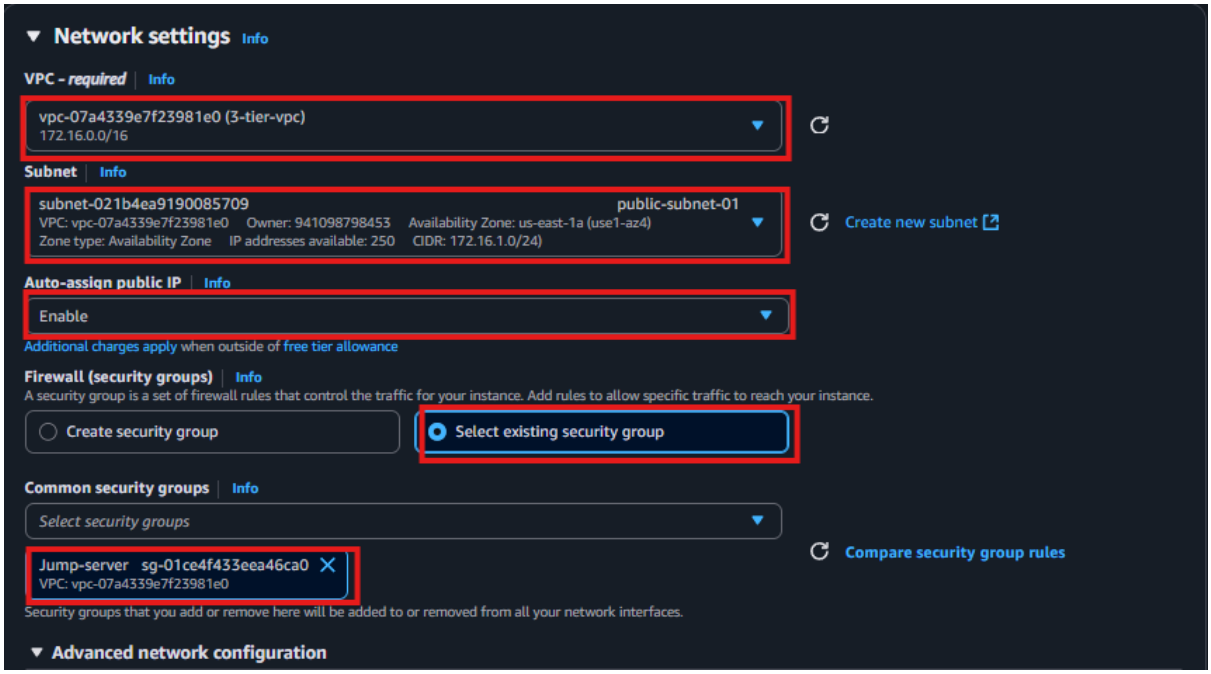
1. Navigate to EC2 and click Launch Instance.

2. Choose an Amazon Linux 2 AMI and t2.micro instance type.





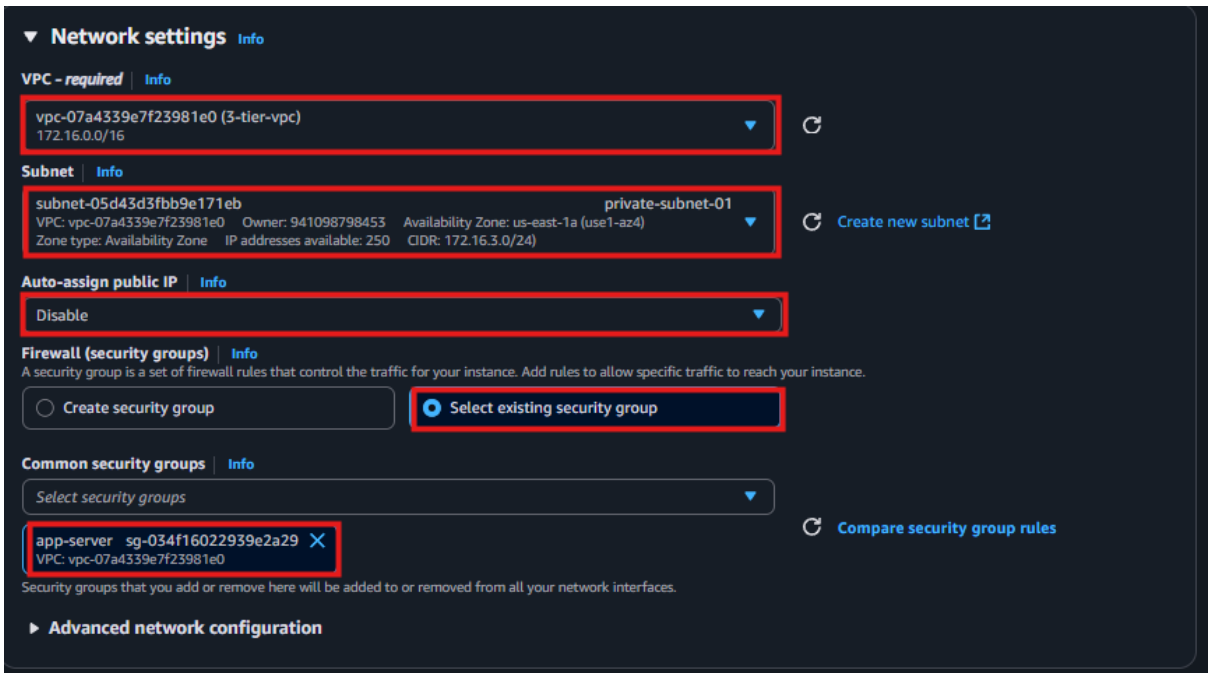
3. Select a public subnet and assign the Jump Server SG security groups



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

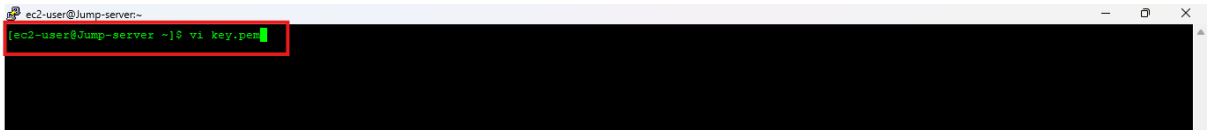
Step 8 – Launch EC2 Instances (Application Tier)

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

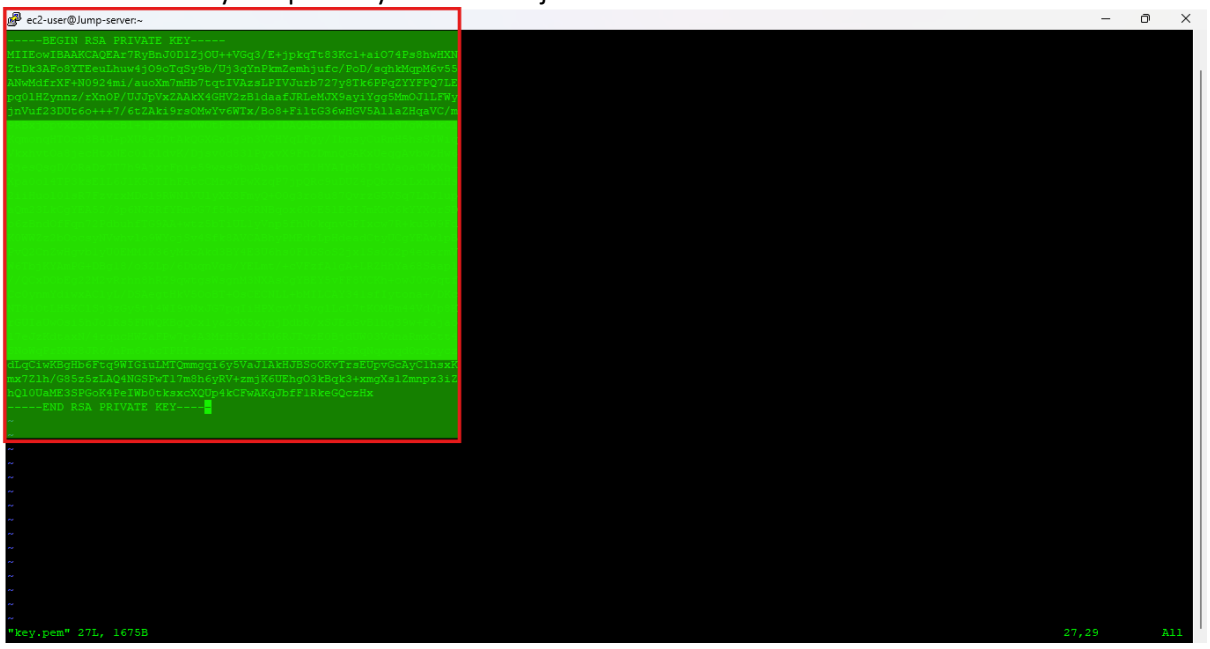


Step 9 - Open the Jump server and take ssh to configure App server and database.

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

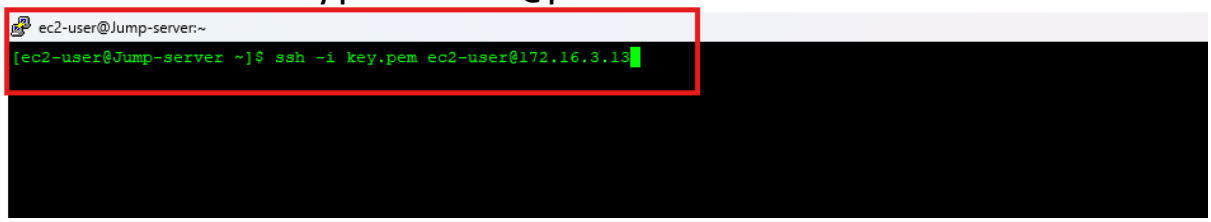


- Paste your .pem key Into the file just created.



- 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

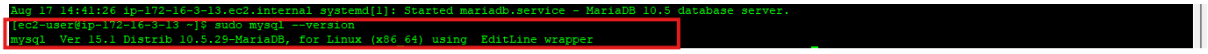
Commands:

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

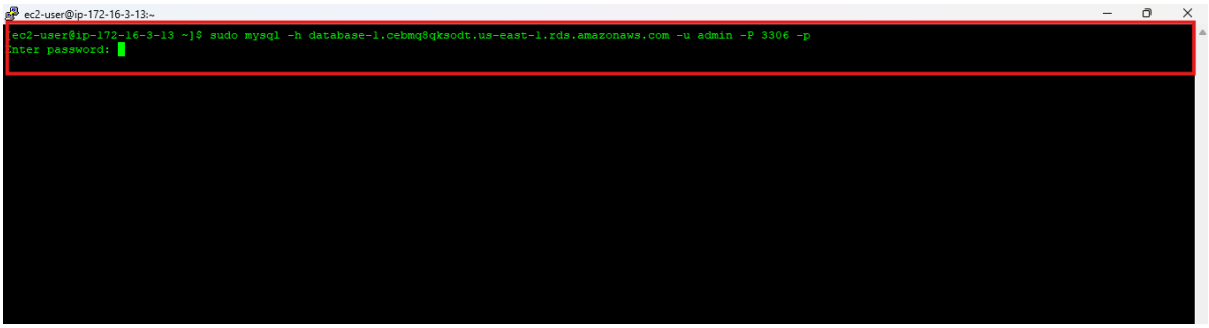
sudo mysql –version

- 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

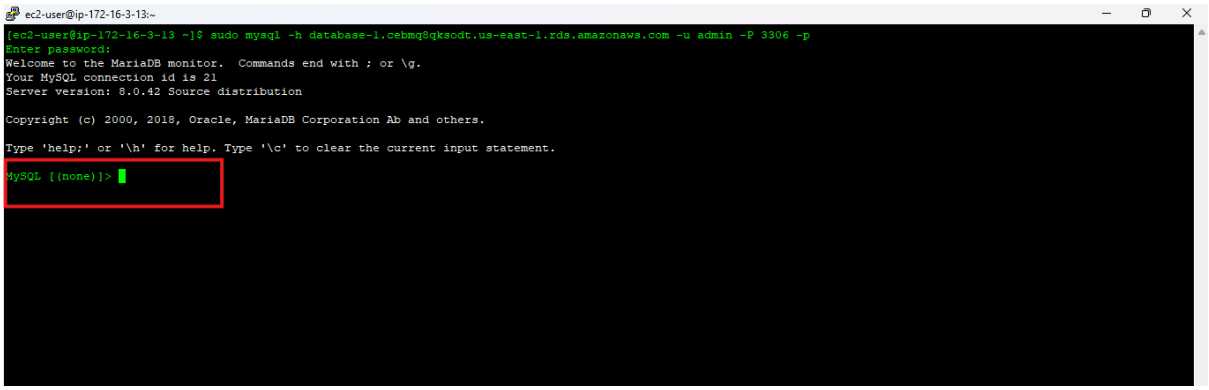


For Ex

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;
```

2. 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;
```

3. When finished, just type exit and hit enter to exit the MySQL client.

Create S3 Bucket & Create IAM Role

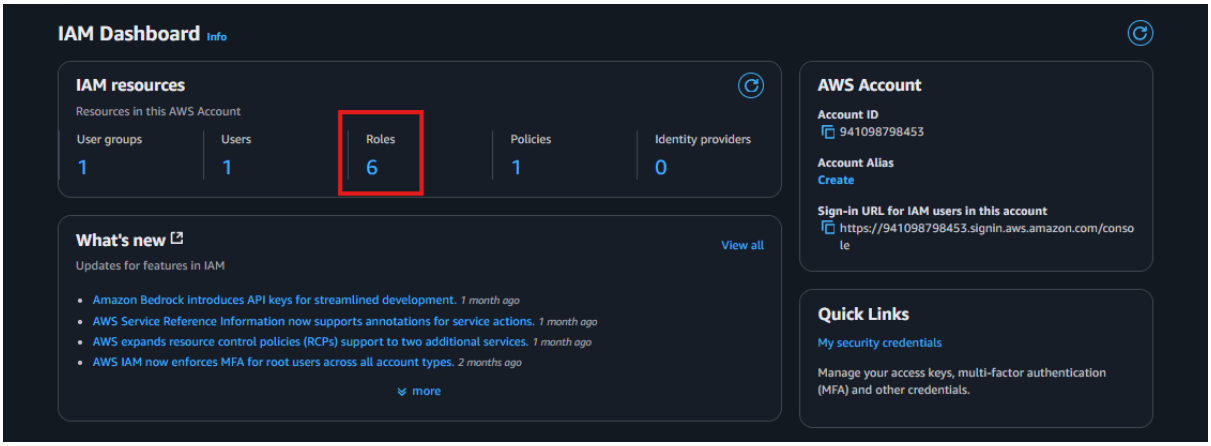
A . Create S3 bucket with default setting
B. Create IAM Role for EC2 ----> the policies needs to be attached

- **AmazonSSMManagedInstanceCore,**
- **AmazonS3ReadOnlyAccess**

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

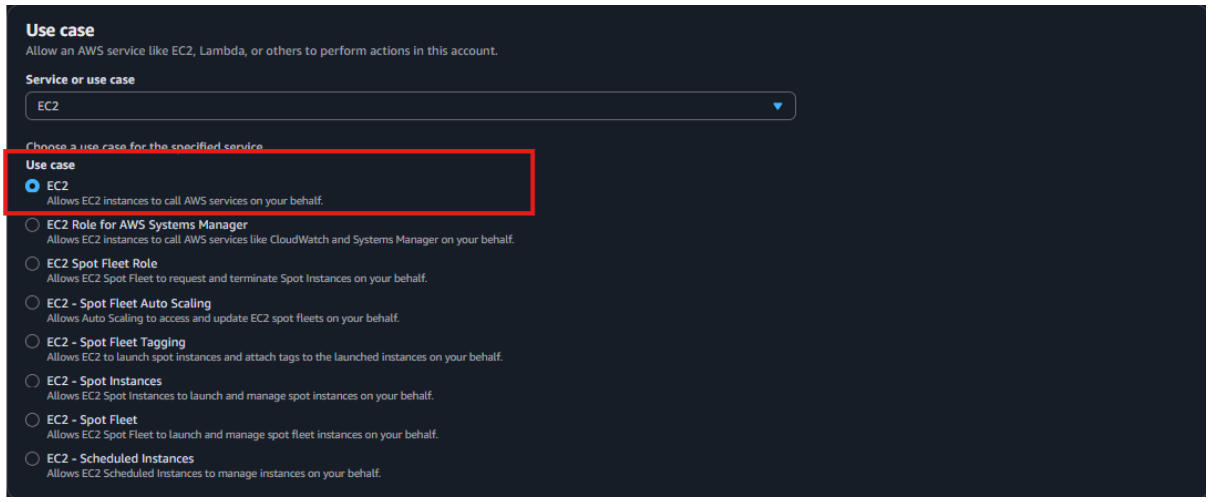
Step 1 – Navigate to IAM

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.

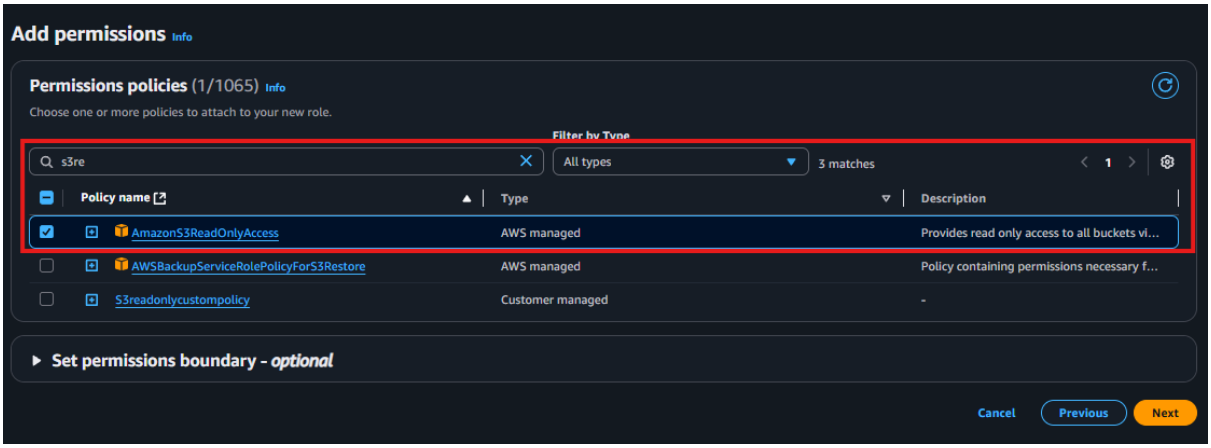


Step 3 – Attach Permission Policy

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.

3. 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
        i. 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
```

E. 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
```

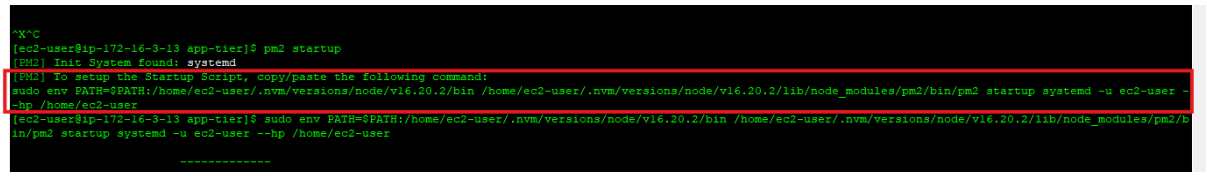
G: you see a status of online, the app is running. If you see errored, then you need to do some troubleshooting. To look at the latest errors, use this command:

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.

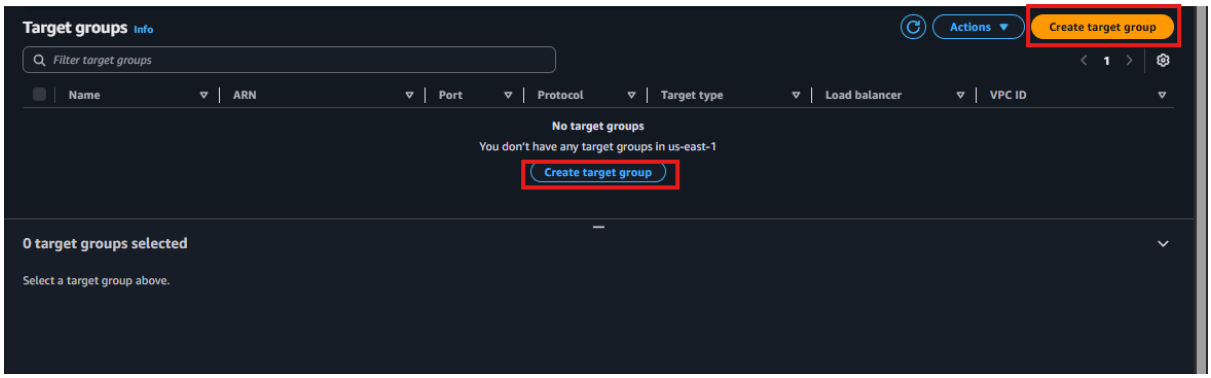
1. Congrats! Your app layer is fully configured and ready to go.

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.

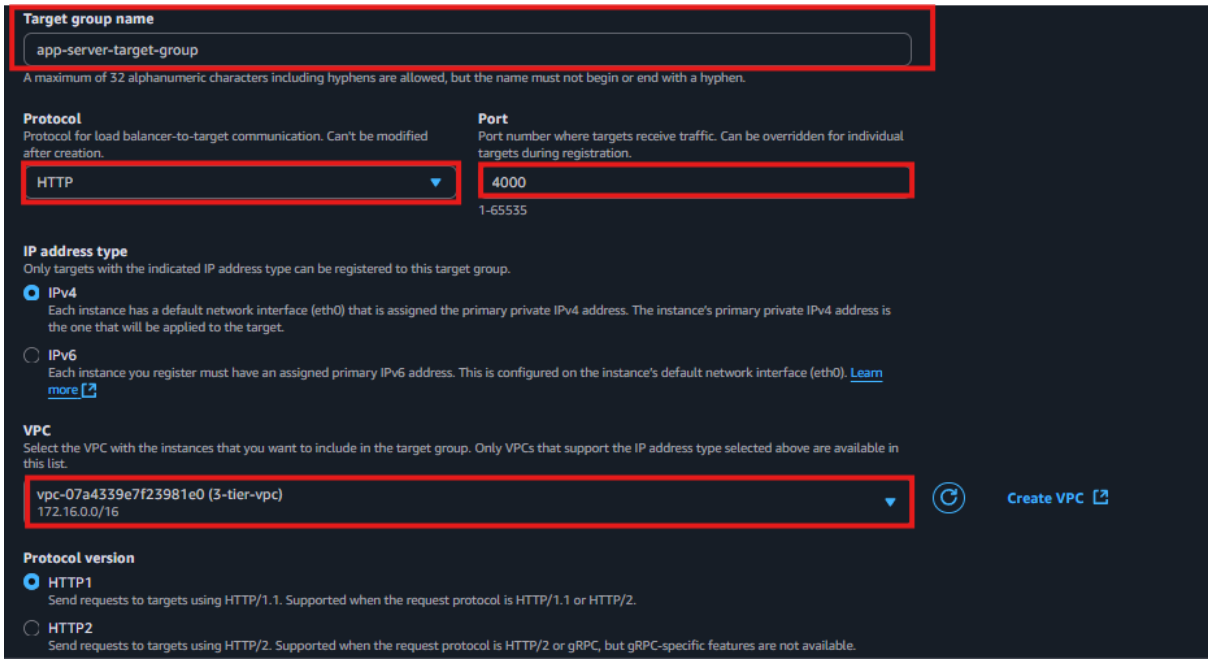


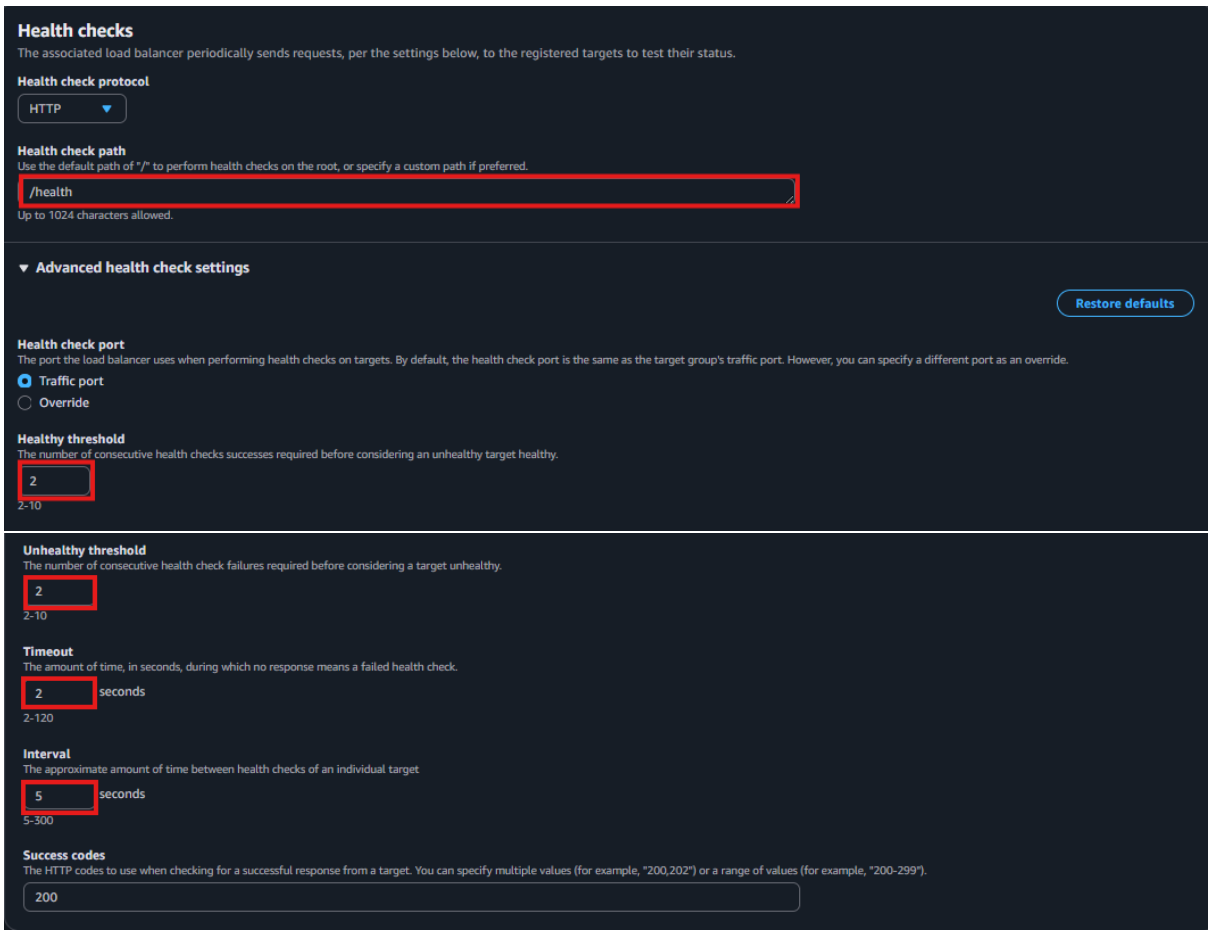
3. Configure the following options:

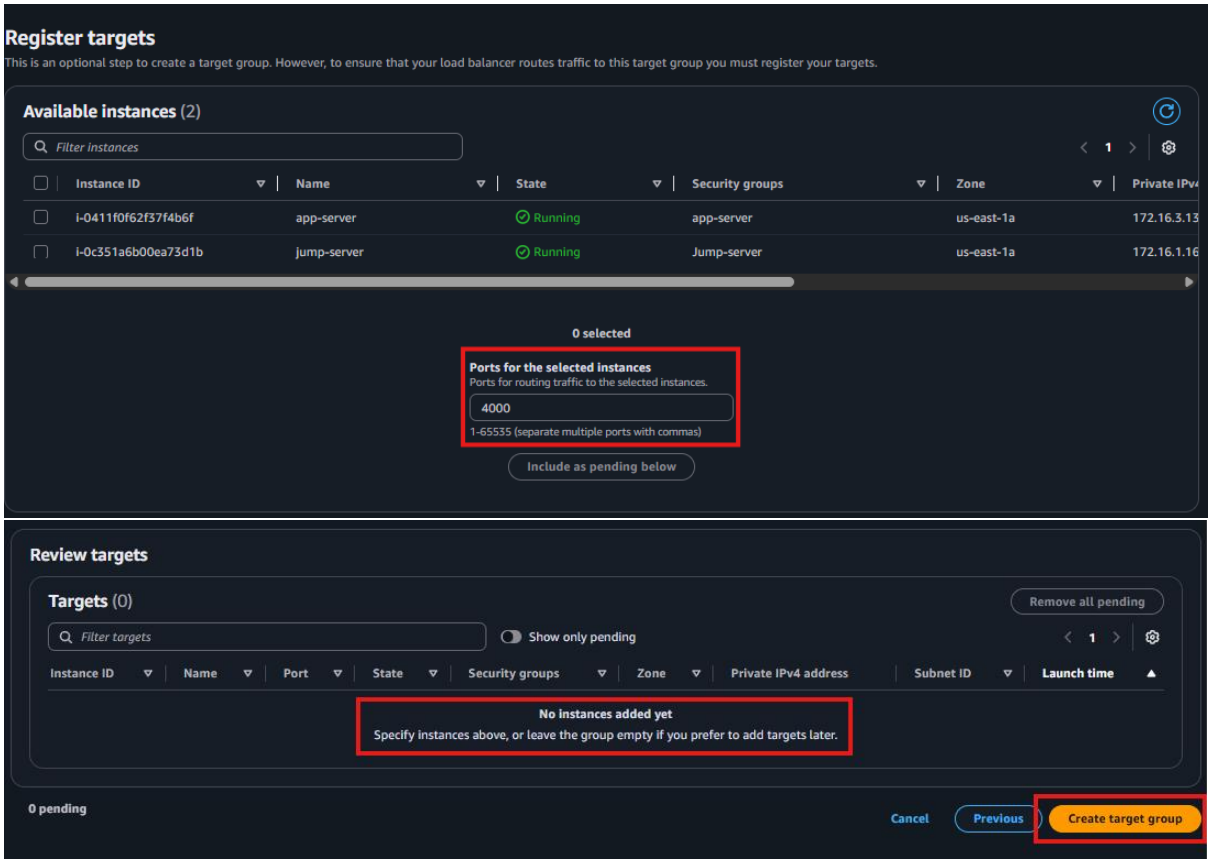
Field	Value
Target type	Instance
Name	app-tier-target-group
Protocol	HTTP
Port	4000
VPC	Select your custom VPC
Health Check Protocol	HTTP
Health Check Path	/health

4. Click Next.

5. Do not register any targets yet (leave it empty).


Target group name
app-server-target-group
A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.
Protocol
Protocol for load balancer-to-target communication. Can't be modified after creation.
HTTP
Port
Port number where targets receive traffic. Can be overridden for individual targets during registration.
4000
1-65535
IP address type
Only targets with the indicated IP address type can be registered to this target group.
IPv4
Each instance has a default network interface (eth0) that is assigned the primary private IPv4 address. The instance's primary private IPv4 address is the one that will be applied to the target.
IPv6
Each instance you register must have an assigned primary IPv6 address. This is configured on the instance's default network interface (eth0). Learn more
VPC
Select the VPC with the instances that you want to include in the target group. Only VPCs that support the IP address type selected above are available in this list.
vpc-07a4339e7f23981e0 (3-tier-vpc)
172.16.0.0/16
Create VPC
Protocol version
HTTP1
Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.
HTTP2
Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.

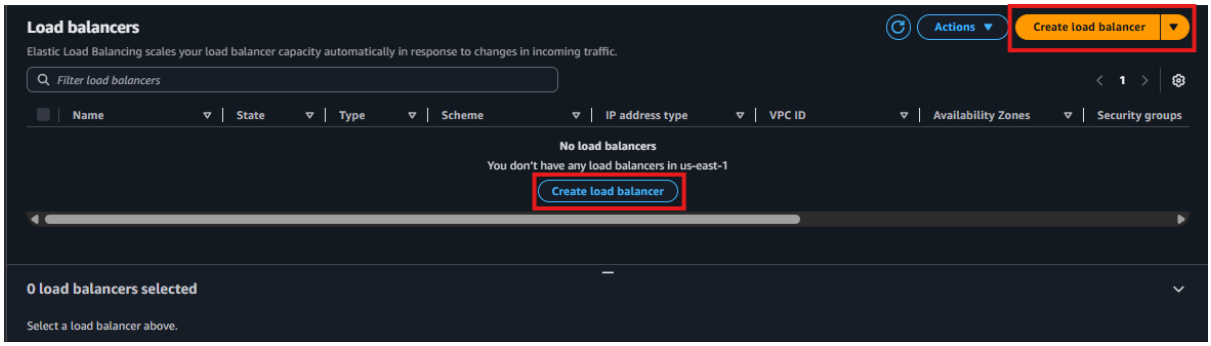

Health checks
The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.
Health check protocol
HTTP
Health check path
Use the default path of "/" to perform health checks on the root, or specify a custom path if preferred.
/health
Up to 1024 characters allowed.
Advanced health check settings
Restore defaults
Health check port
The port the load balancer uses when performing health checks on targets. By default, the health check port is the same as the target group's traffic port. However, you can specify a different port as an override.
Traffic port
Override
Healthy threshold
The number of consecutive health checks successes required before considering an unhealthy target healthy.
2
2-10
Unhealthy threshold
The number of consecutive health check failures required before considering a target unhealthy.
2
2-10
Timeout
The amount of time, in seconds, during which no response means a failed health check.
2
seconds
2-120
Interval
The approximate amount of time between health checks of an individual target
5
seconds
5-300
Success codes
The HTTP codes to use when checking for a successful response from a target. You can specify multiple values (for example, "200,202") or a range of values (for example, "200-299").
200



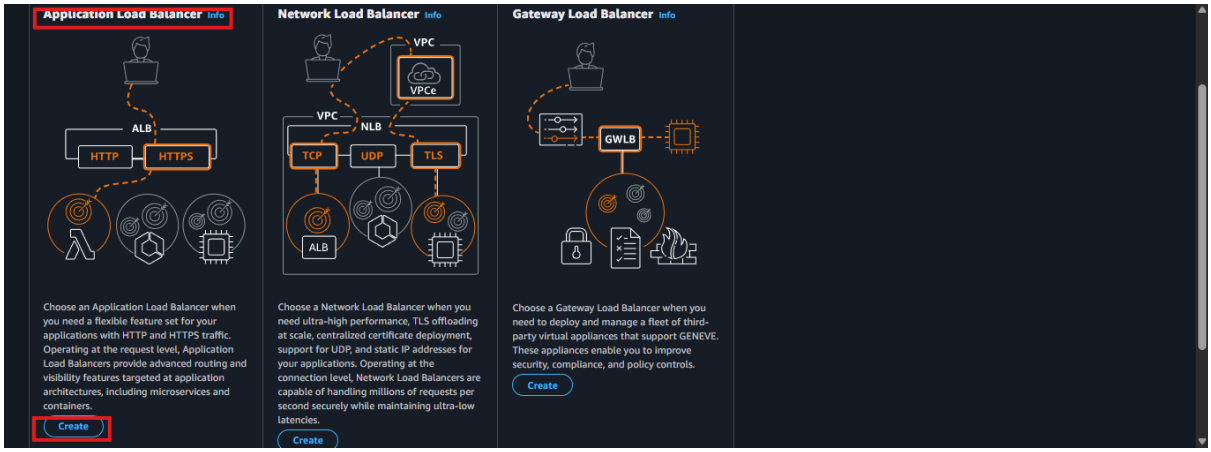
6. Click Create target group.

Step 2 – Create Internal Application Load Balancer

1. Go to EC2 → Load Balancers.



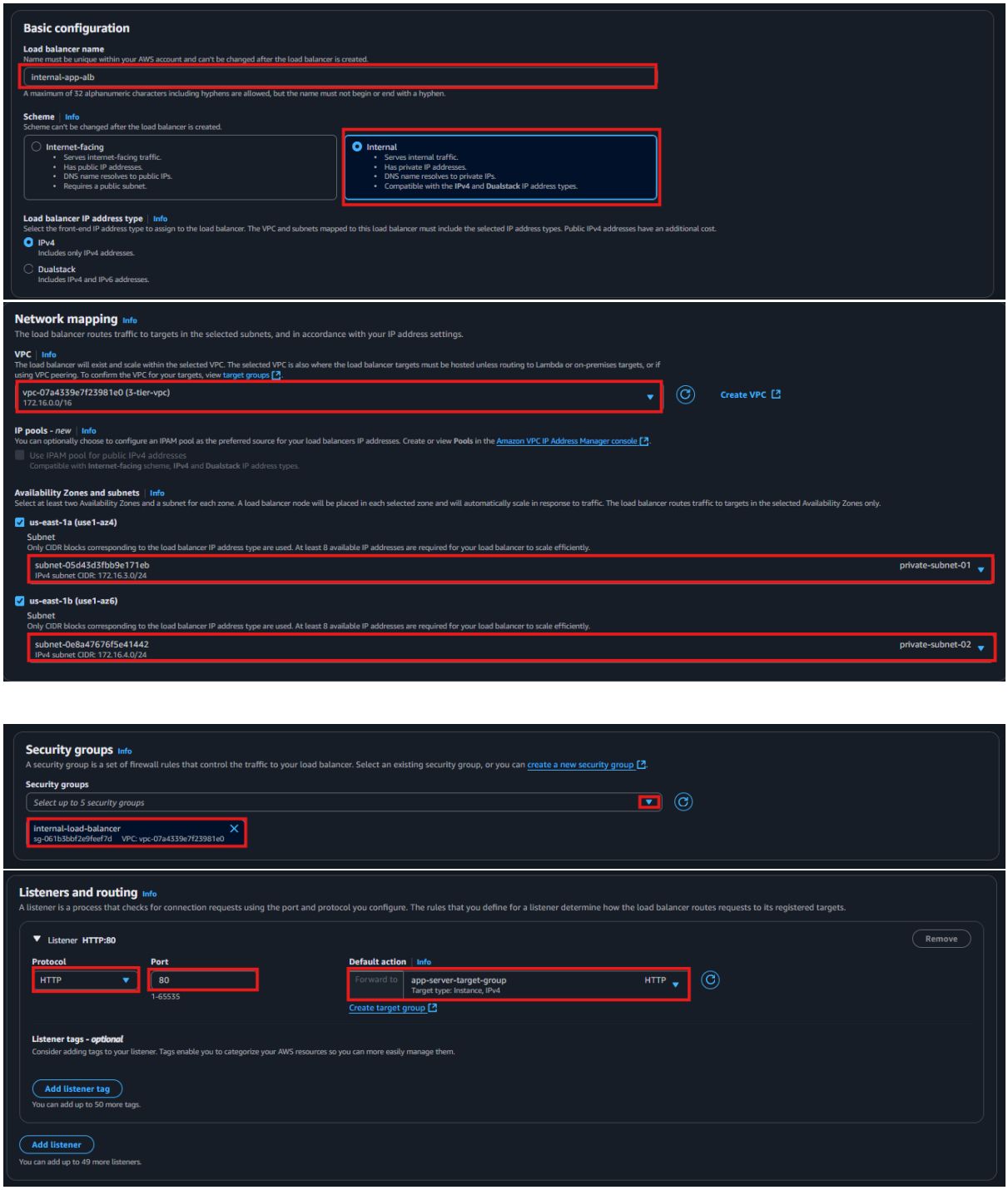
2. Click Create Load Balancer → Application Load Balancer.



3. 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.

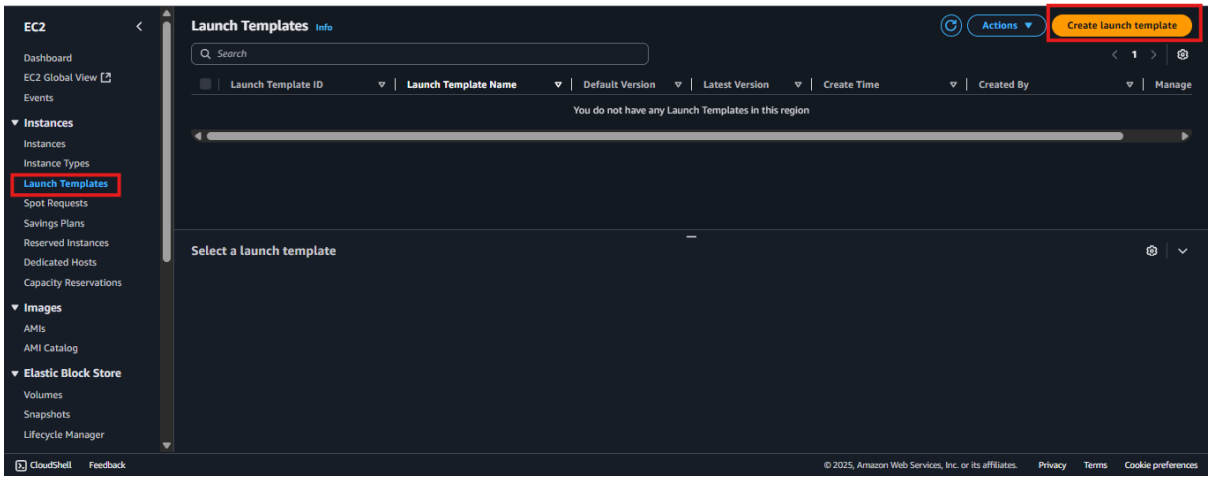
4. Provide a name (e.g. *app-server-ami*) and description.

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

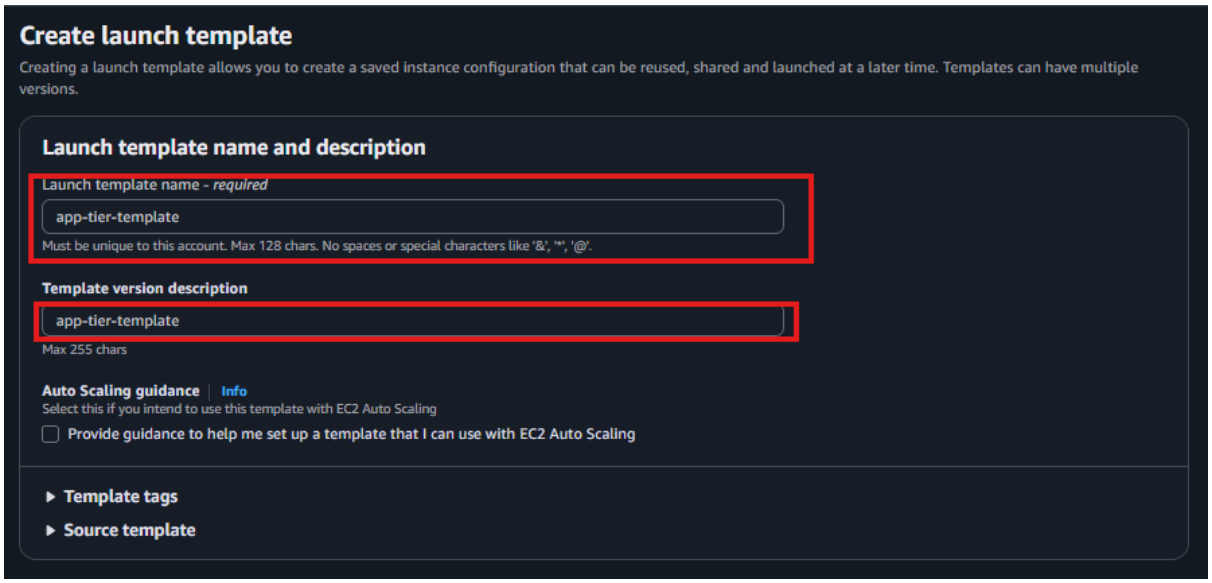
1. Go to EC2 → Launch Templates → Create launch template

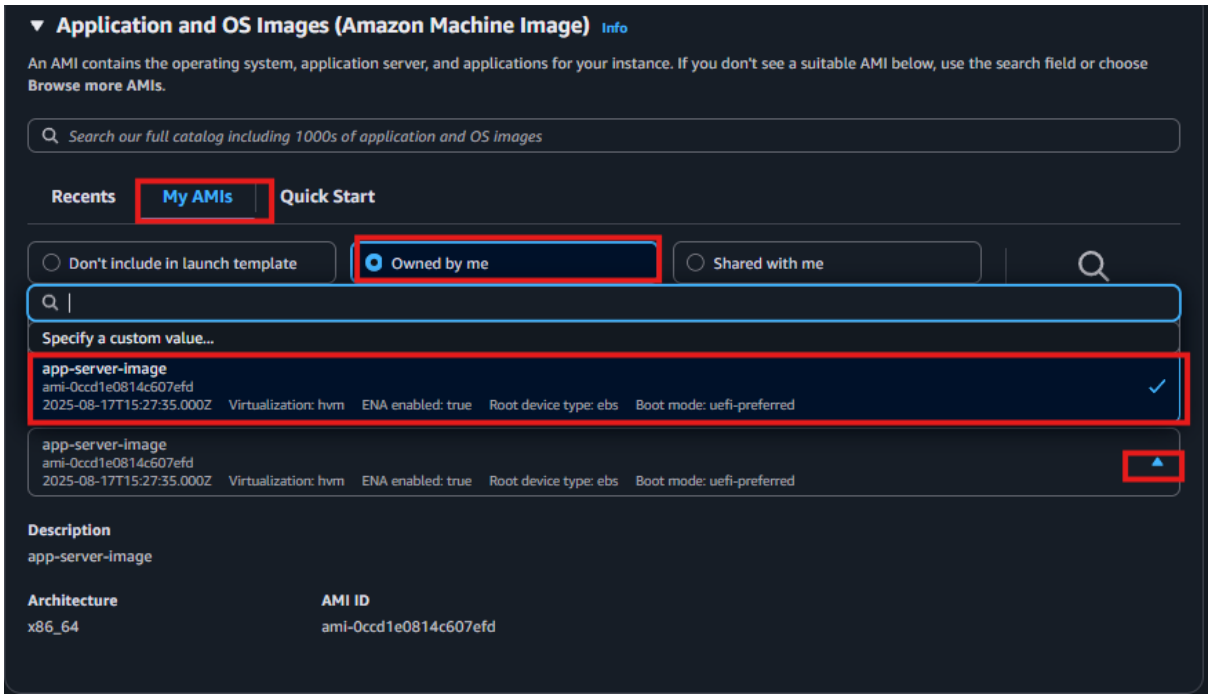


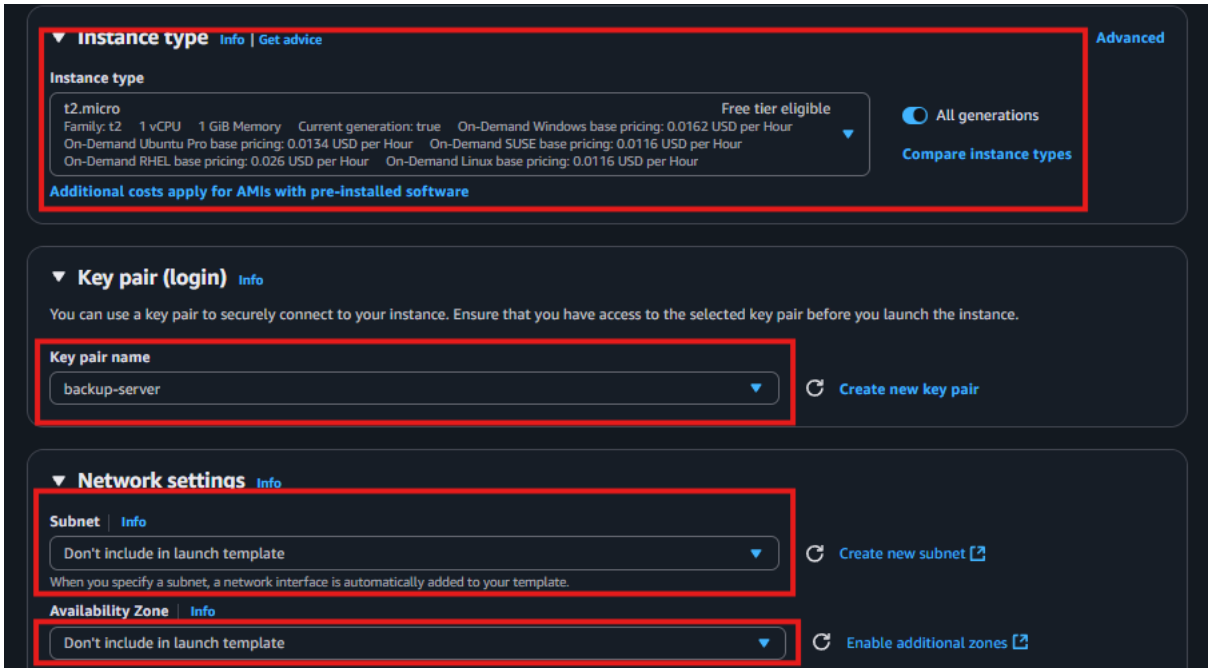
2. Set the following:

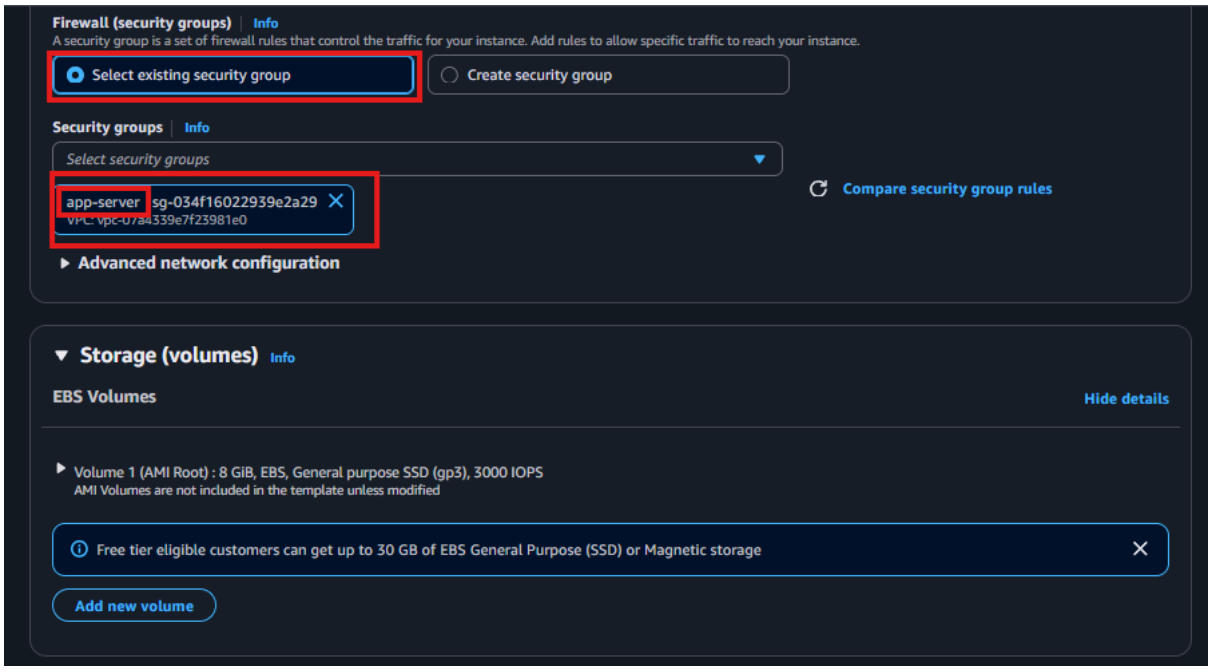
Field	Value
Launch template name	app-tier-template
AMI ID	Select the AMI created in Step 1
Instance type	(same type used for the App Server, e.g. t2.micro)
Key pair	Select the required key pair
Security group	App Server SG (allows port 4000 from the internal ALB)

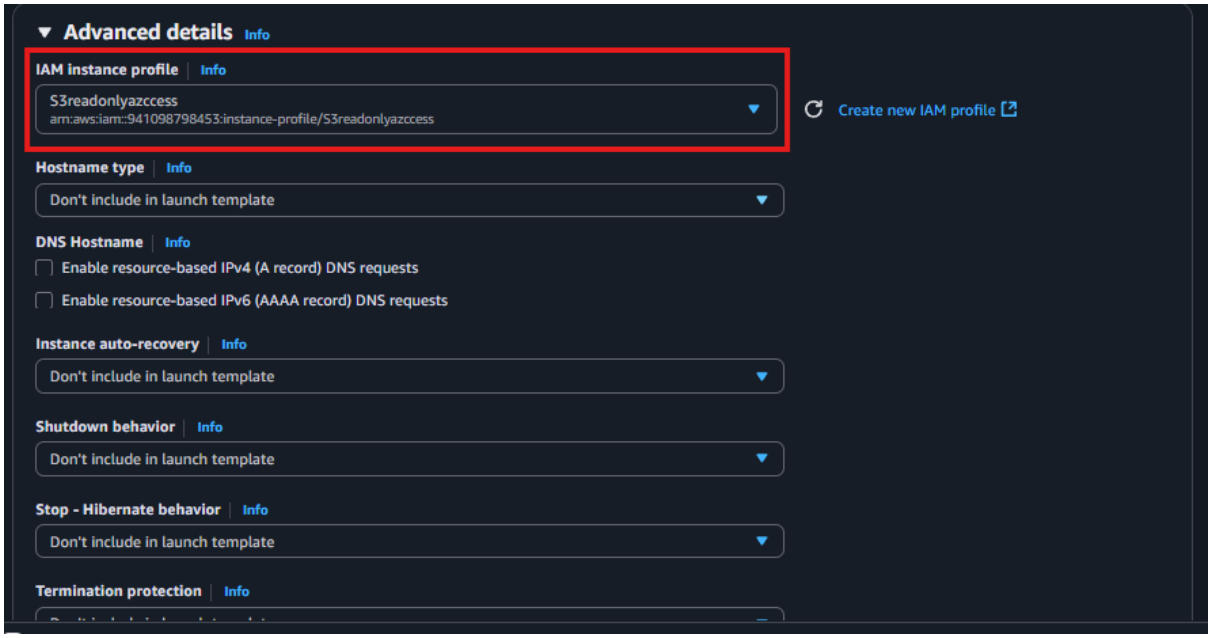
Field	Value


Create launch template
Creating a launch template allows you to create a saved instance configuration that can be reused, shared and launched at a later time. Templates can have multiple versions.
Launch template name and description
Launch template name - required
app-tier-template
Must be unique to this account. Max 128 chars. No spaces or special characters like '&', '*', '@'.
Template version description
app-tier-template
Max 255 chars
Auto Scaling guidance Info
Select this if you intend to use this template with EC2 Auto Scaling
Provide guidance to help me set up a template that I can use with EC2 Auto Scaling
Template tags
Source template


Application and OS Images (Amazon Machine Image) Info
An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose Browse more AMIs.
Search our full catalog including 1000s of application and OS images
Recents
My AMIs
Quick Start
Don't include in launch template
Owned by me
Shared with me
Specify a custom value...
app-server-image
ami-0ccd1e0814c607efd
2025-08-17T15:27:35.000Z Virtualization: hvm ENA enabled: true Root device type: ebs Boot mode: uefi-preferred
app-server-image
ami-0ccd1e0814c607efd
2025-08-17T15:27:35.000Z Virtualization: hvm ENA enabled: true Root device type: ebs Boot mode: uefi-preferred
Description
app-server-image
Architecture
x86_64
AMI ID
ami-0ccd1e0814c607efd


Instance type Info | Get advice
Advanced
Instance type
t2.micro
Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour On-Demand SUSE base pricing: 0.0116 USD per Hour
On-Demand RHEL base pricing: 0.026 USD per Hour On-Demand Linux base pricing: 0.0116 USD per Hour
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name
backup-server
Create new key pair
Network settings Info
Subnet Info
Don't include in launch template
Create new subnet
When you specify a subnet, a network interface is automatically added to your template.
Availability Zone Info
Don't include in launch template
Enable additional zones


Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Select existing security group
Create security group
Security groups Info
Select security groups
app-server sg-034f16022939e2a29
VPC: vpc-07a4339e7f23981e0
Compare security group rules
Advanced network configuration
Storage (volumes) Info
EBS Volumes
Hide details
Volume 1 (AMI Root) : 8 GiB, EBS, General purpose SSD (gp3), 3000 IOPS
AMI Volumes are not included in the template unless modified
Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage
Add new volume

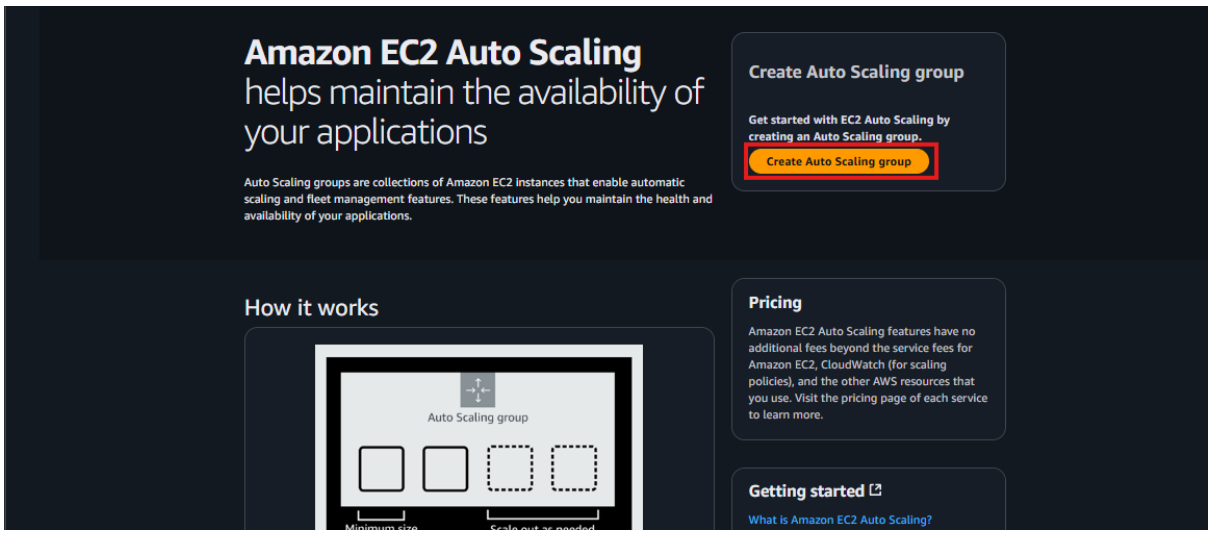


3. Click Create launch template

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

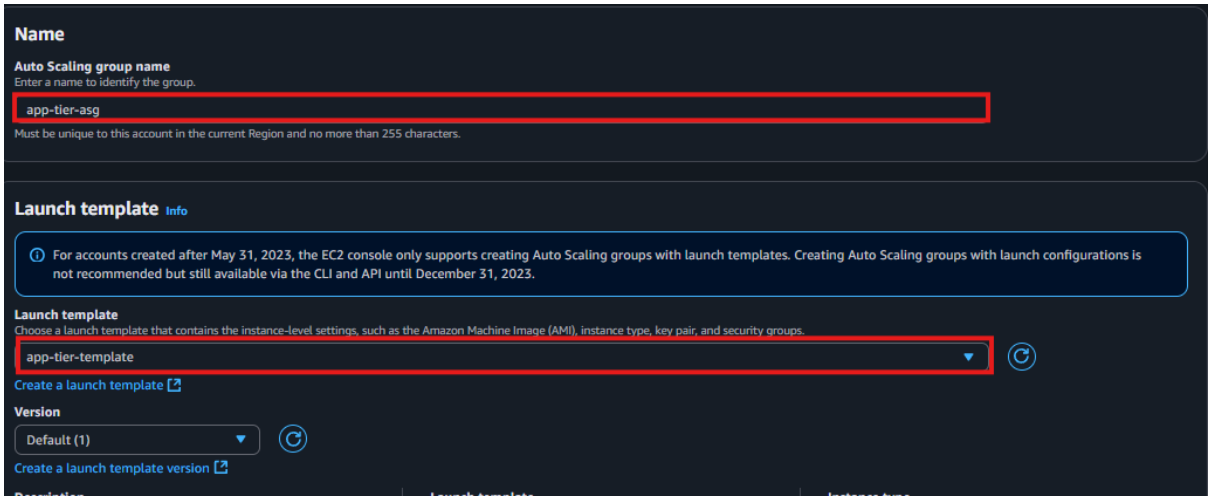
🔄 **Step 3 – Create Auto Scaling Group Based on the Launch Template**

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



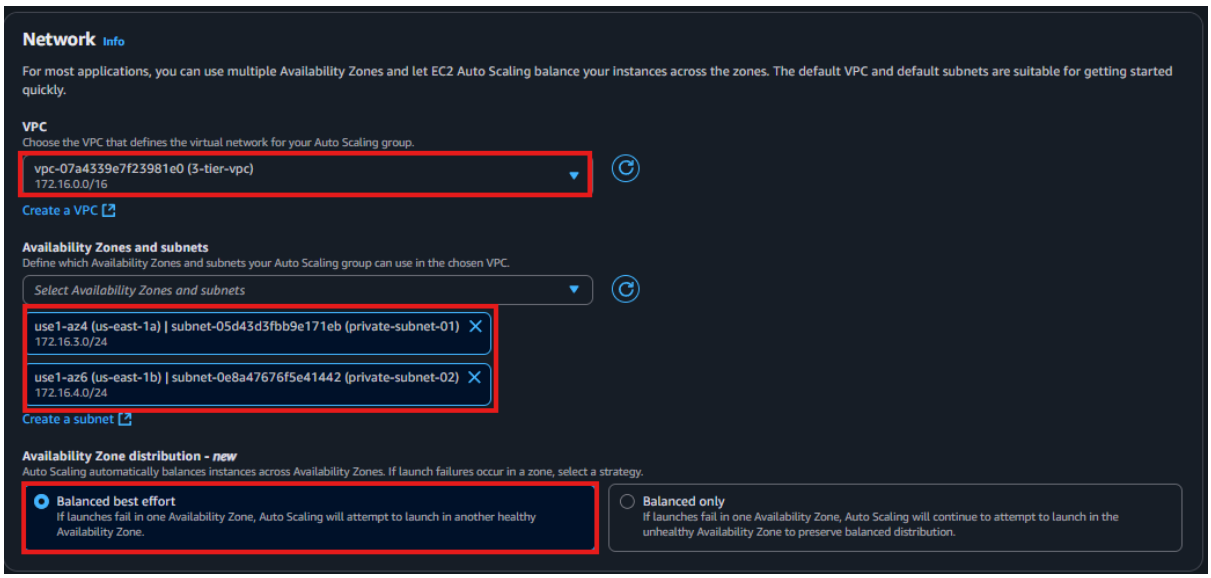
2. **Auto Scaling group name**: app-tier-asg

3. **Launch template**: select app-tier-template



4. **VPC**: Select the custom VPC

5. **Subnets**: Select **private subnets**



Click **Next**

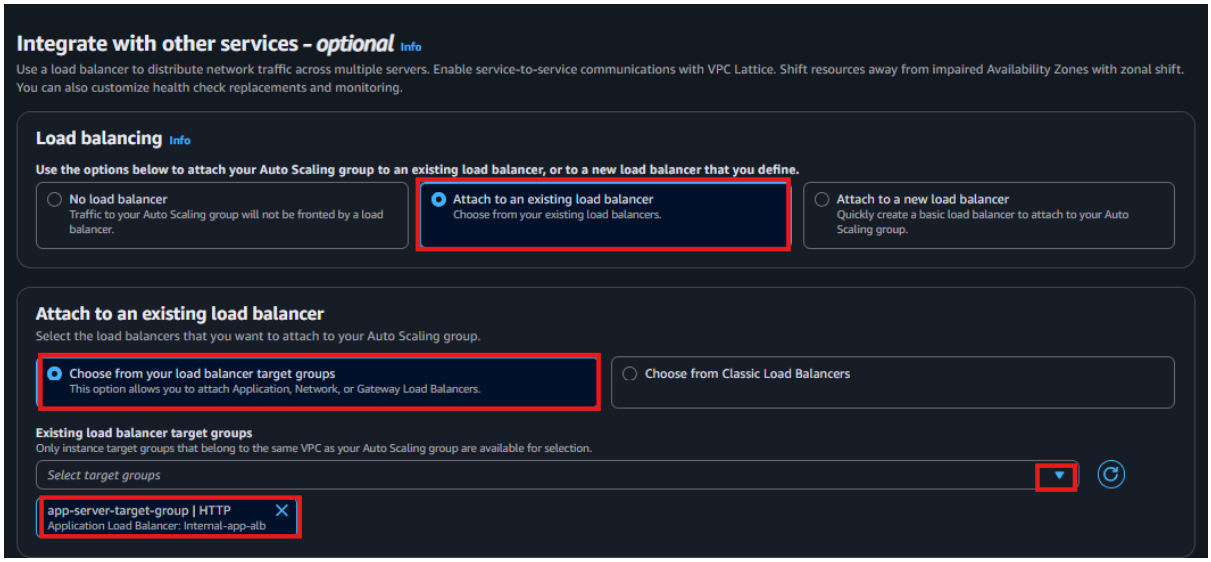
🌐 Step 4 – Attach to the Internal Load Balancer

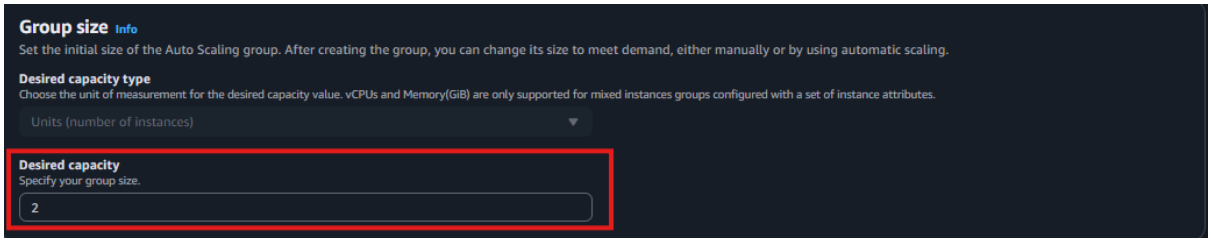
1. Select **Attach to an existing load balancer**

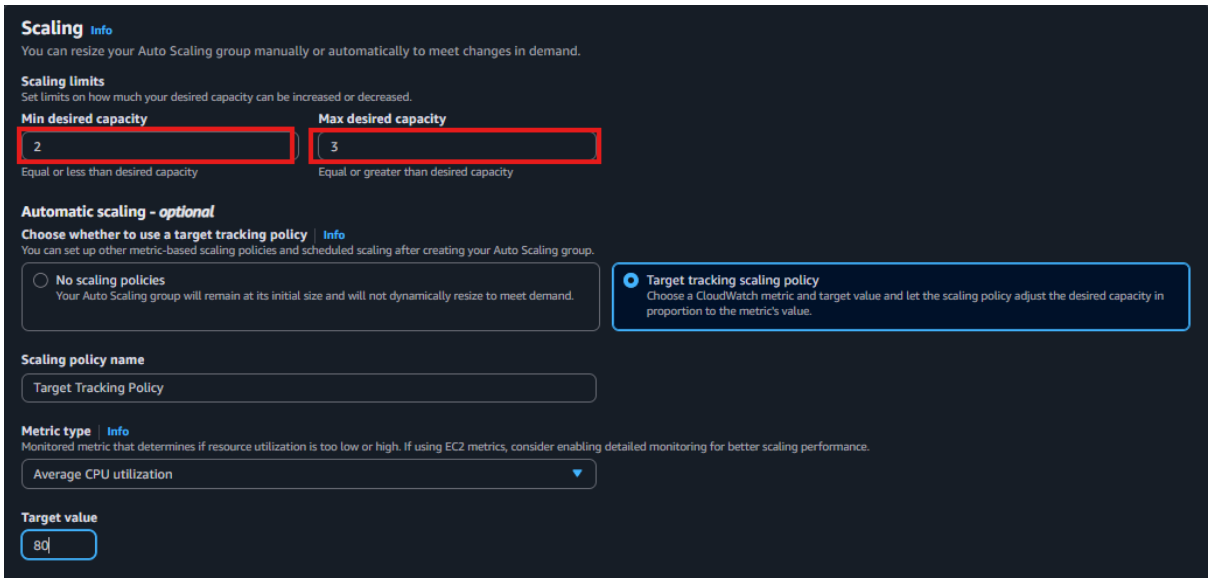
2. Choose **Application Load Balancer**

3. Select **internal-app-alb**

4. Select the **target group** you created for the App Tier (e.g. app-target-group)



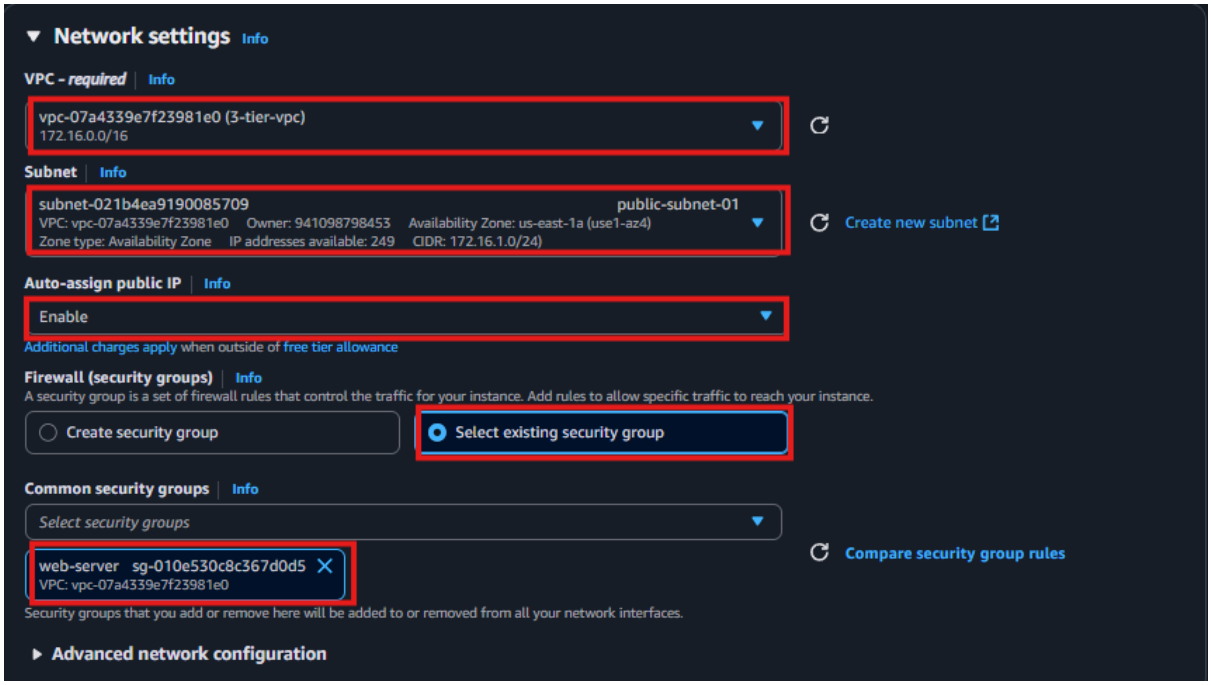


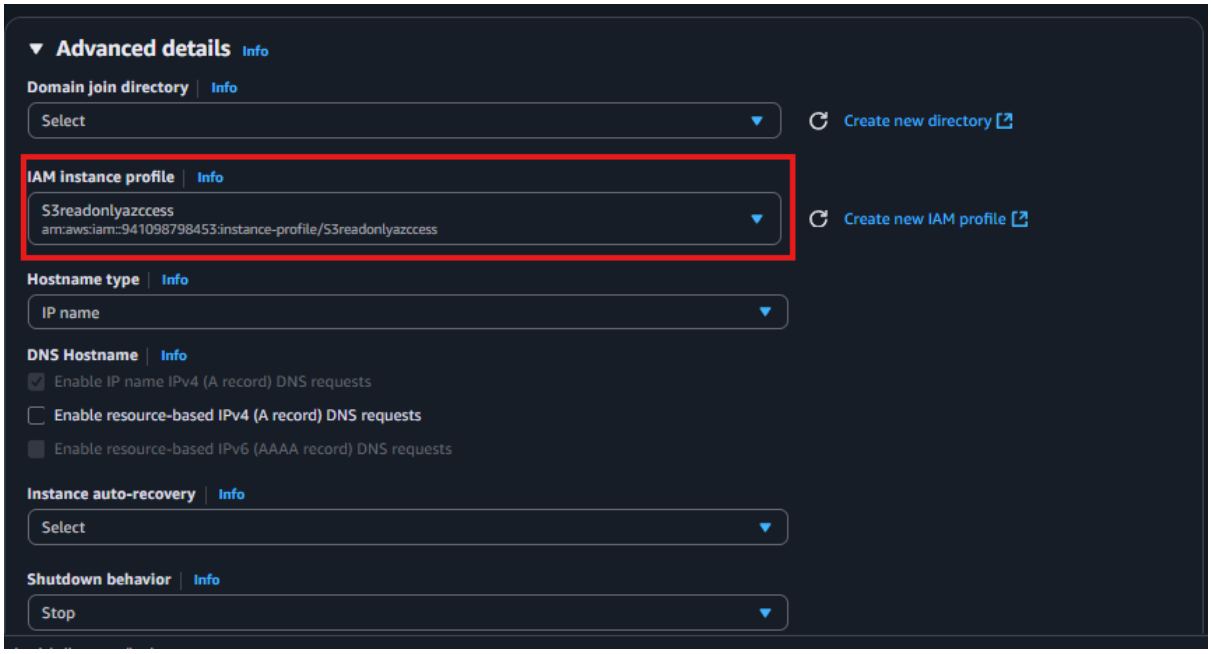


Click **Next** and **finish** the creation of auto-scaling group

Step 11 – Launch EC2 Instances (Web-Tier)

2. 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.

```
        listen       [::]:80;
        server_name  _;

        #health check
        location /health {
        default_type text/html;
        return 200 "<!DOCTYPE html><p>Web Tier Health Check</p>\n";
        }

        #react app and front end files
        location / {
        root    /home/ec2-user/web-tier/build;
        index index.html index.htm
        try_files $uri /index.html;
        }

        #proxy for internal lb
        location /api/{
                proxy_pass http://[REPLACE-WITH-INTERNAL-LB-DNS]:80/;
        }


    }

# Settings for a TLS enabled server.
#
#    server {
#        listen       443 ssl http2;
#        listen       [::]:443 ssl http2;
#        server_name  _;
#        root         /usr/share/nginx/html;
#
```

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 :

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
source ~/.bashrc
nvm install 16
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:

```
cd ~/web-tier
npm install
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.

```
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:

```
cd /etc/nginx
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
```

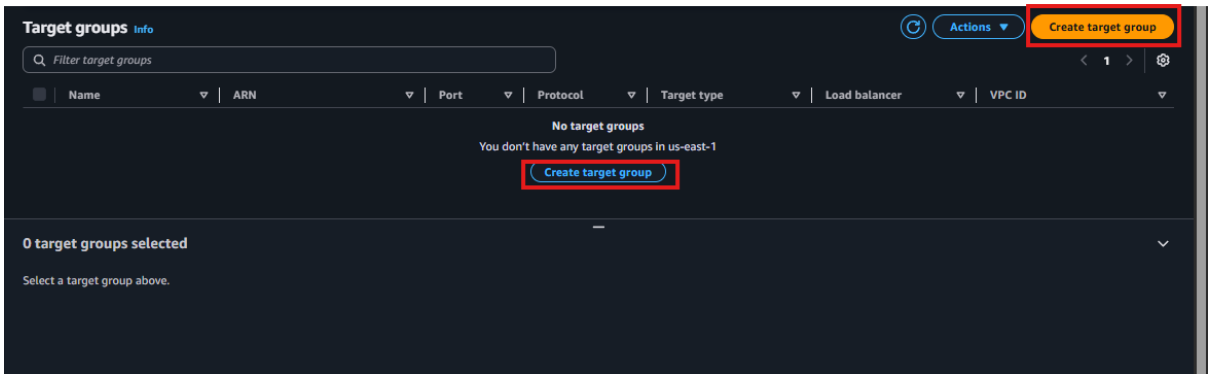
Now when you plug in the public IP of your web tier instance, you should see your website, which you can find on the Instance details page on the EC2 dashboard. If you have the database connected and working correctly, then you will also see the database working. You'll be able to add data. Careful with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

4. Go to EC2 → Target Groups in the AWS Console.

5. Click Create target group.



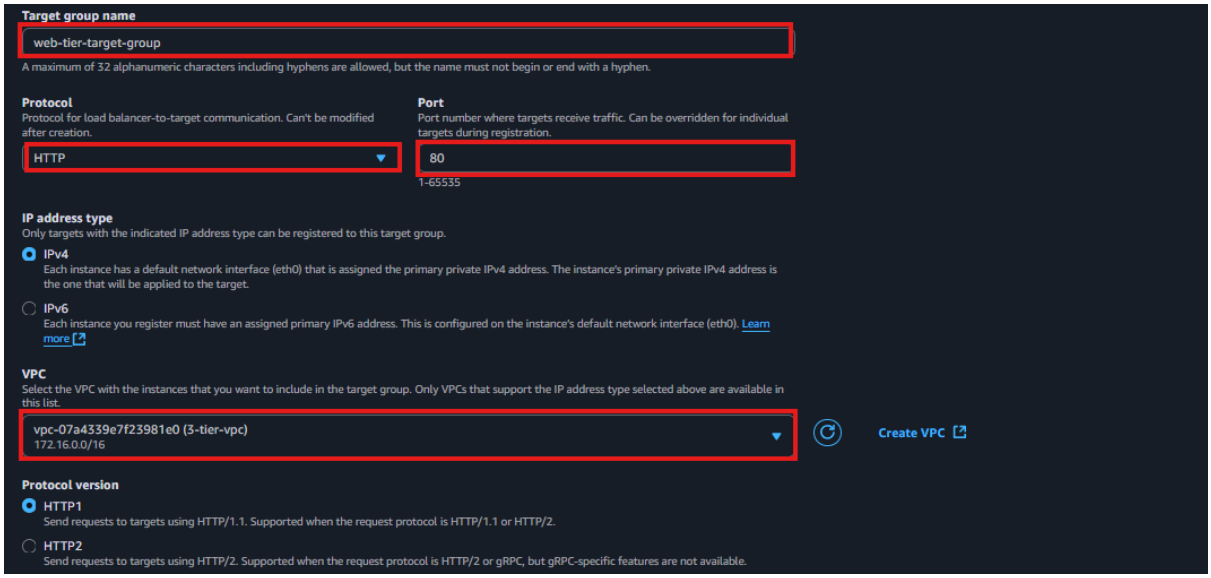
6. Configure the following options:

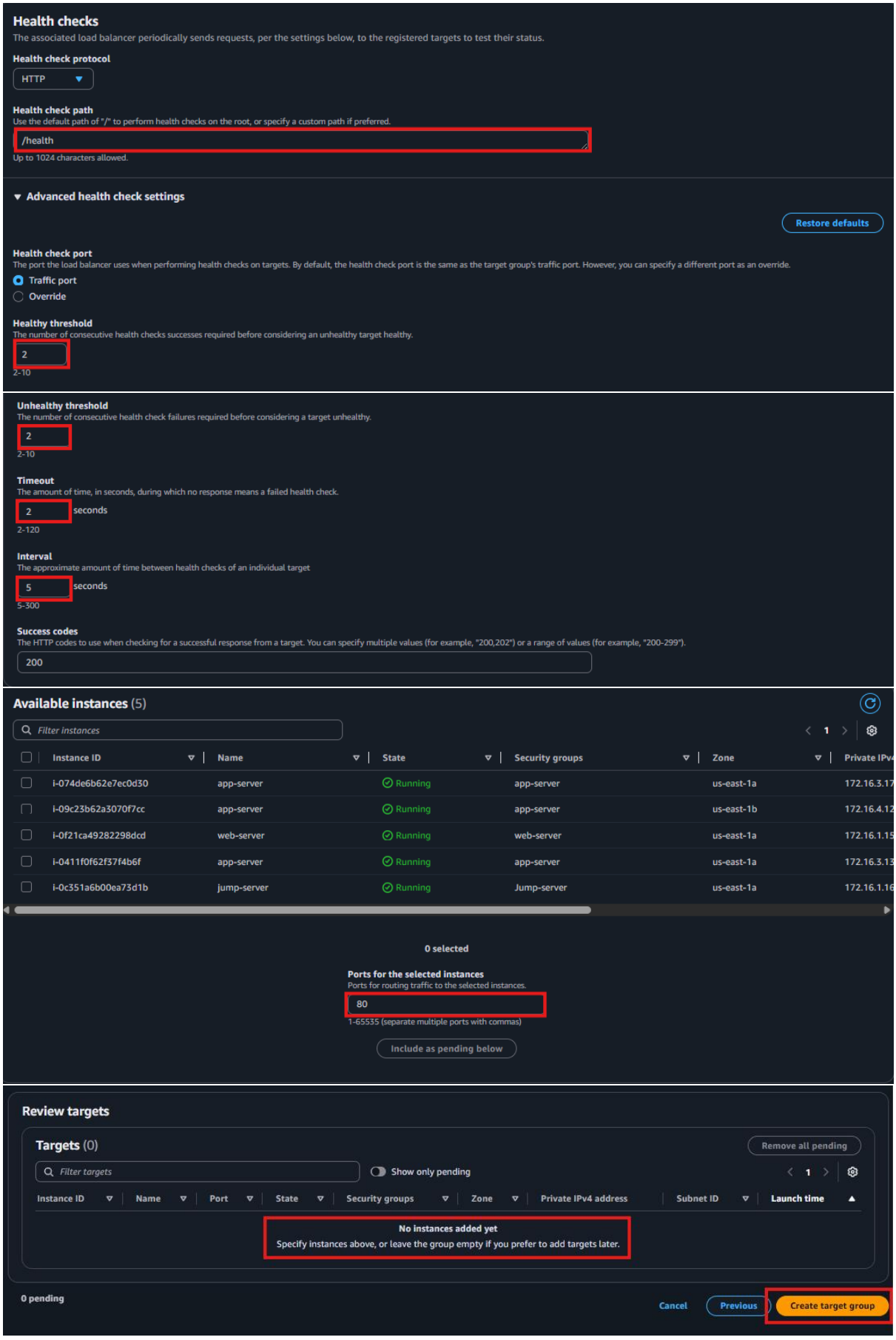
Field	Value
Target type	Instance
Name	web-tier-target-group

Field	Value
Protocol	HTTP
Port	80
VPC	Select your custom VPC
Health Check Protocol	HTTP
Health Check Path	/health

7. Click Next.

8. Do not register any targets yet (leave it empty).

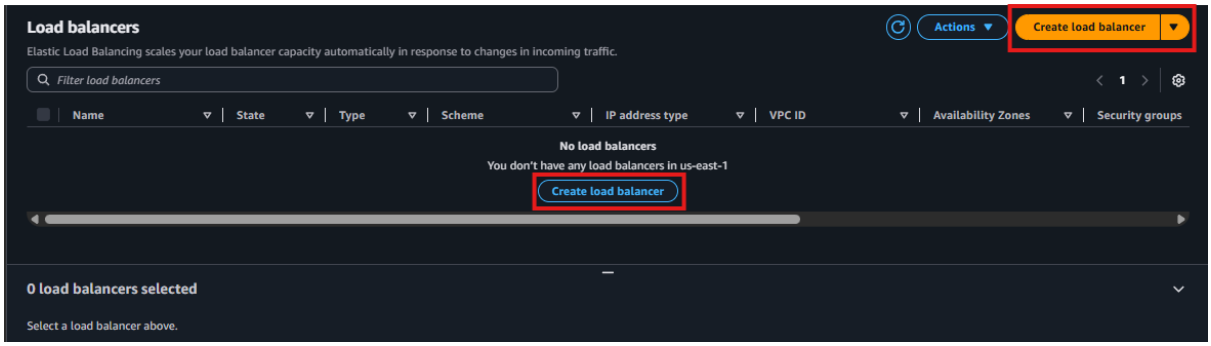



Health checks
The associated load balancer periodically sends requests, per the settings below, to the registered targets to test their status.
Health check protocol
HTTP
Health check path
Use the default path of "/" to perform health checks on the root, or specify a custom path if preferred.
/health
Up to 1024 characters allowed.
Advanced health check settings
Restore defaults
Health check port
The port the load balancer uses when performing health checks on targets. By default, the health check port is the same as the target group's traffic port. However, you can specify a different port as an override.
Traffic port
Override
Healthy threshold
The number of consecutive health checks successes required before considering an unhealthy target healthy.
2
2-10
Unhealthy threshold
The number of consecutive health check failures required before considering a target unhealthy.
2
2-10
Timeout
The amount of time, in seconds, during which no response means a failed health check.
2
seconds
2-120
Interval
The approximate amount of time between health checks of an individual target
5
seconds
5-300
Success codes
The HTTP codes to use when checking for a successful response from a target. You can specify multiple values (for example, "200,202") or a range of values (for example, "200-299").
200
Available instances (5)
Filter instances
Instance ID
Name
State
Security groups
Zone
Private IPv4
i-074de6b62e7ec0d30 app-server Running app-server us-east-1a 172.16.3.17
i-09c23b62a3070f7cc app-server Running app-server us-east-1b 172.16.4.12
i-0f21ca49282298dcd web-server Running web-server us-east-1a 172.16.1.15
i-0411f0f62f37f4b6f app-server Running app-server us-east-1a 172.16.3.13
i-0c351a6b00ea73d1b jump-server Running Jump-server us-east-1a 172.16.1.16
0 selected
Ports for the selected instances
Ports for routing traffic to the selected instances.
80
1-65535 (separate multiple ports with commas)
Include as pending below
Review targets
Targets (0)
Remove all pending
Filter targets
Show only pending
Instance ID
Name
Port
State
Security groups
Zone
Private IPv4 address
Subnet ID
Launch time
No instances added yet
Specify instances above, or leave the group empty if you prefer to add targets later.
0 pending
Cancel
Previous
Create target group

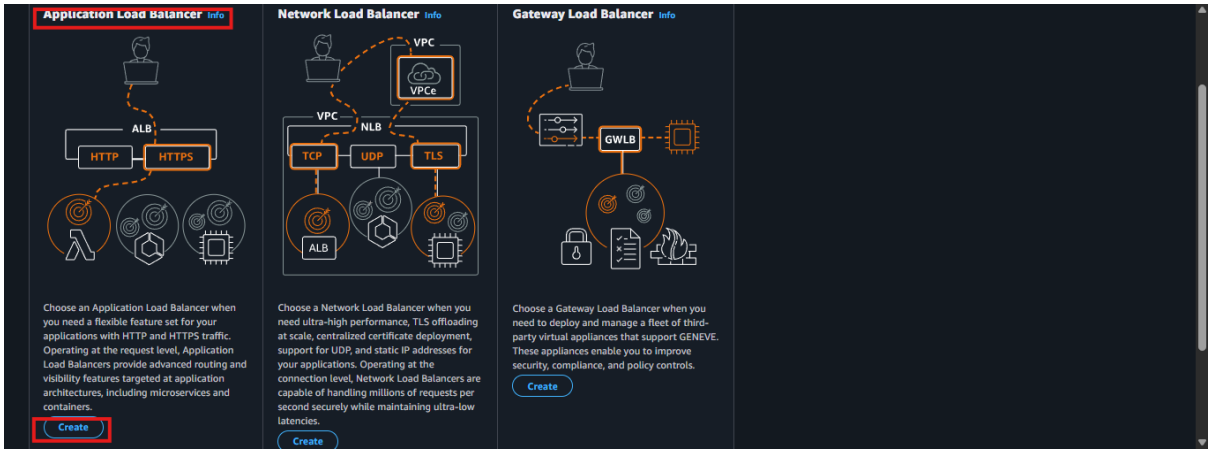
9. Click Create target group.

Step 2 – Create Internal Application Load Balancer

4. Go to EC2 → Load Balancers.



5. Click Create Load Balancer → Application Load Balancer.

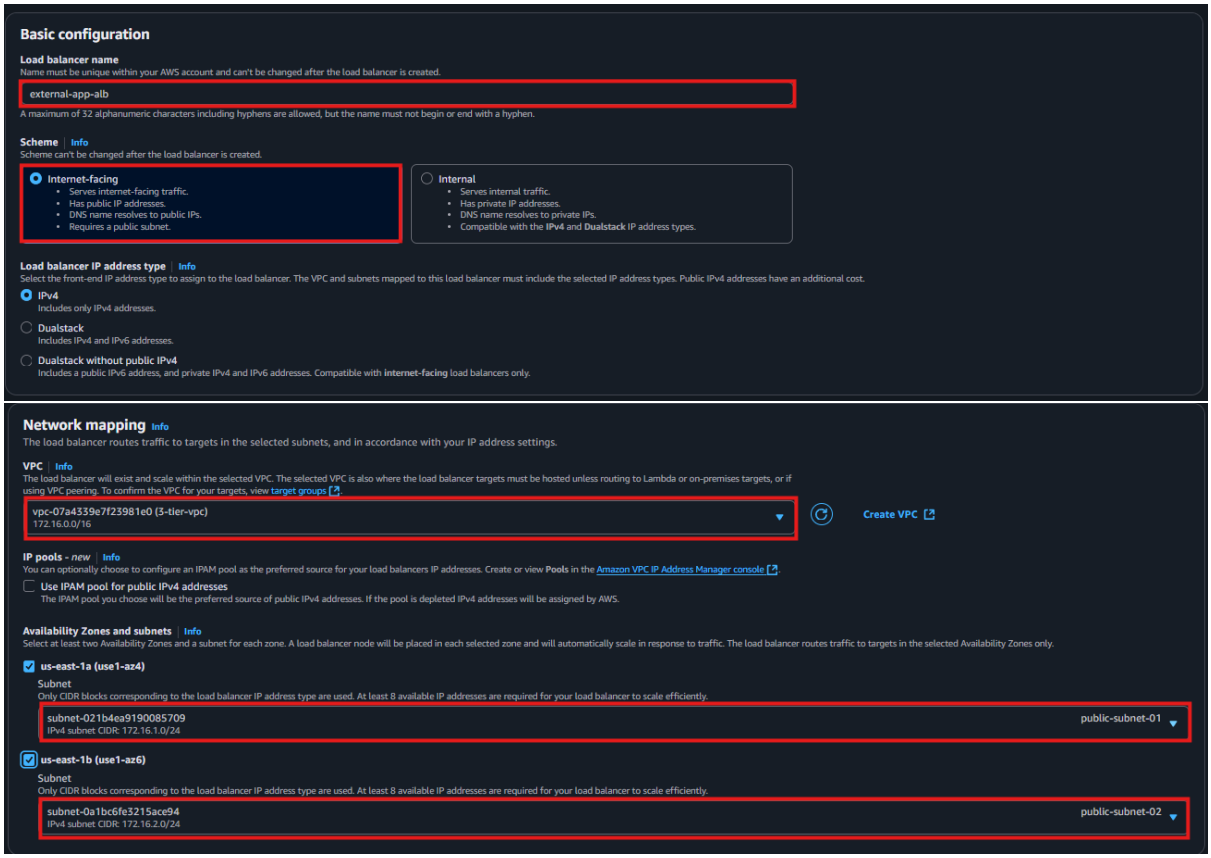


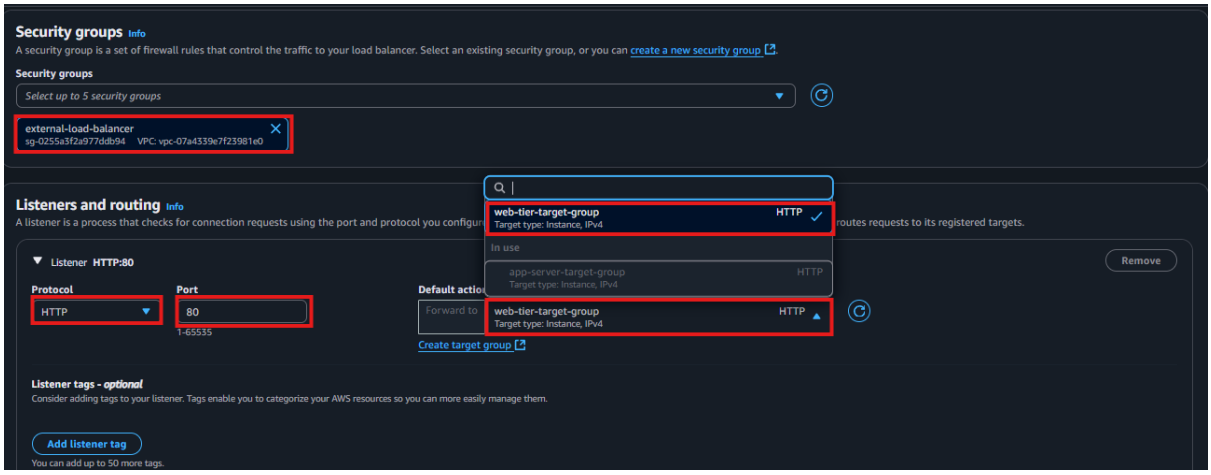
6. Configure the following:

Field	Value
Name	external-app-alb
Scheme	Internet-facing
IP address type	IPv4

Field	Value
VPC	Select your custom VPC
Availability Zones	Select Public subnets (Web subnets)

Click Next: Configure Security Settings (leave default).





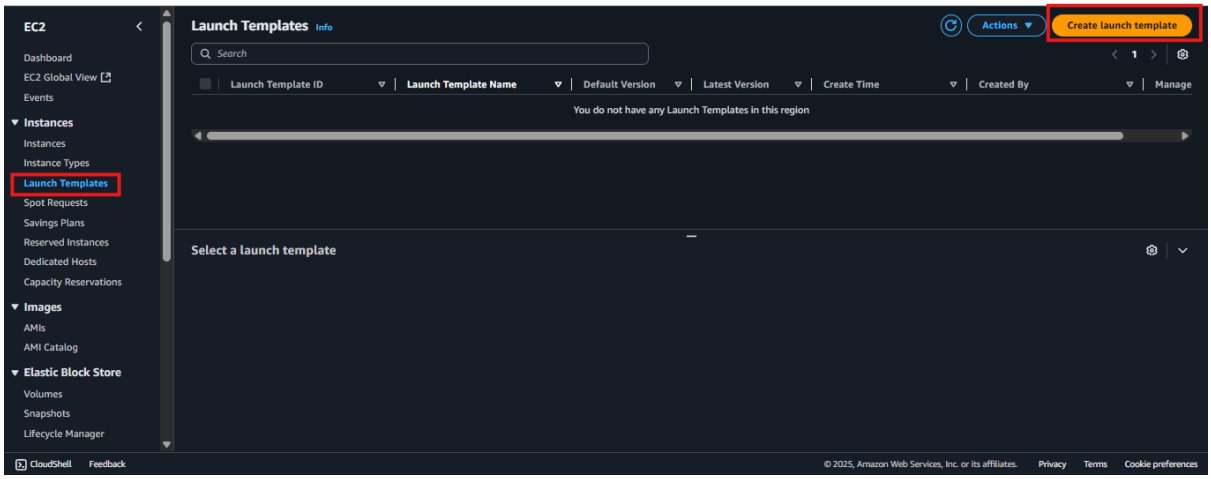
Click **Next** and **Create Load Balancer.**

🔧 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).

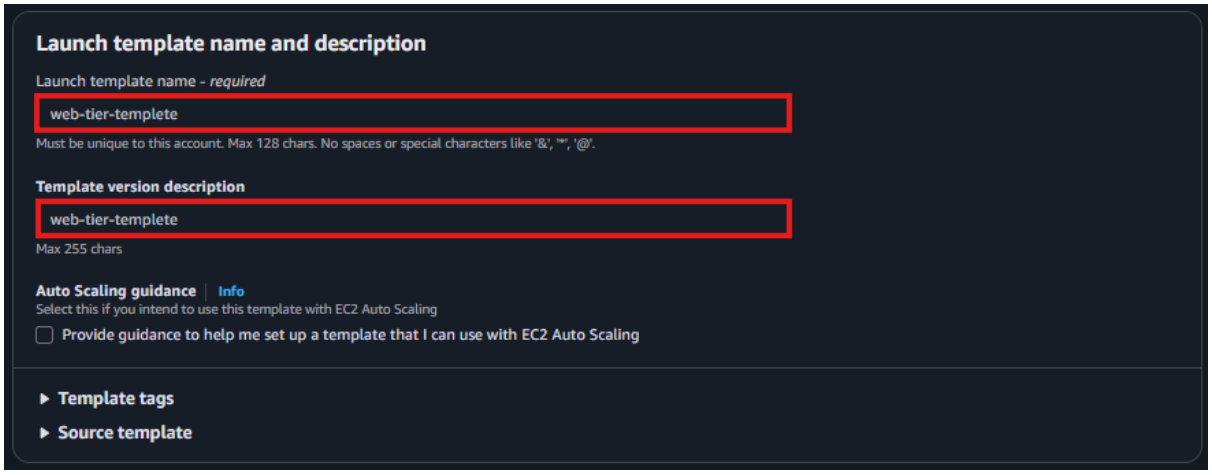
🔧 Step 2 – Create a Launch Template Using the AMI

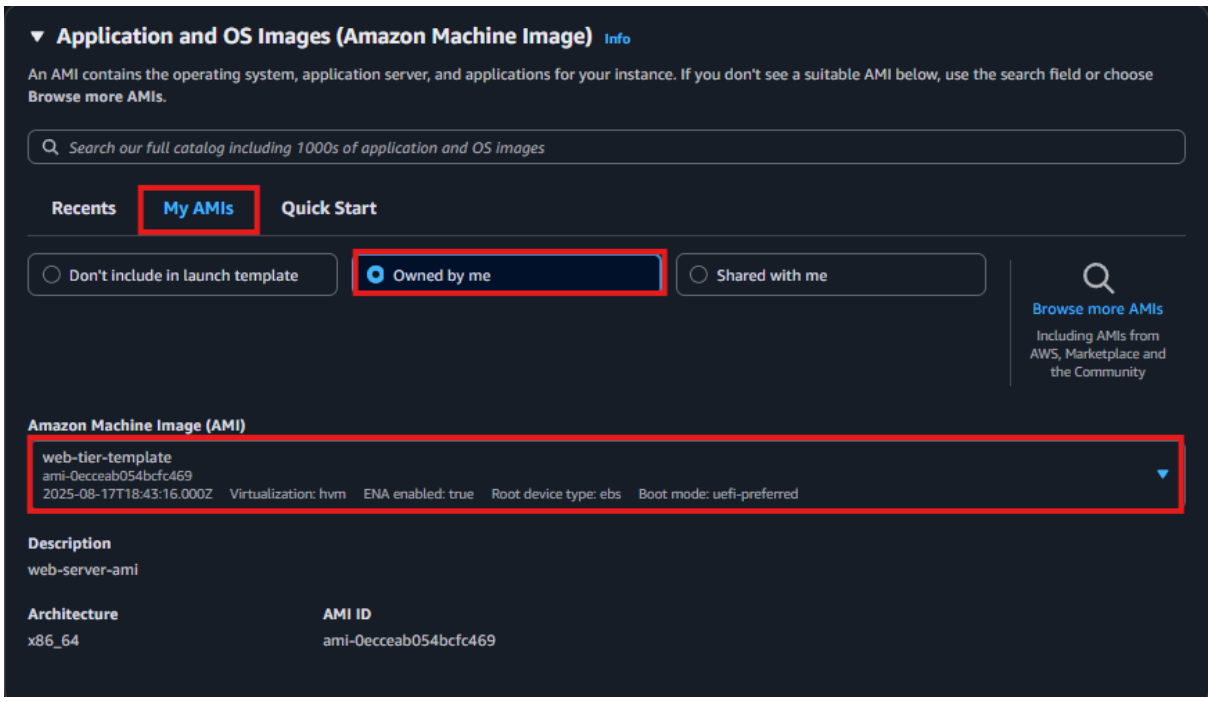
4. Go to EC2 → Launch Templates → Create launch template

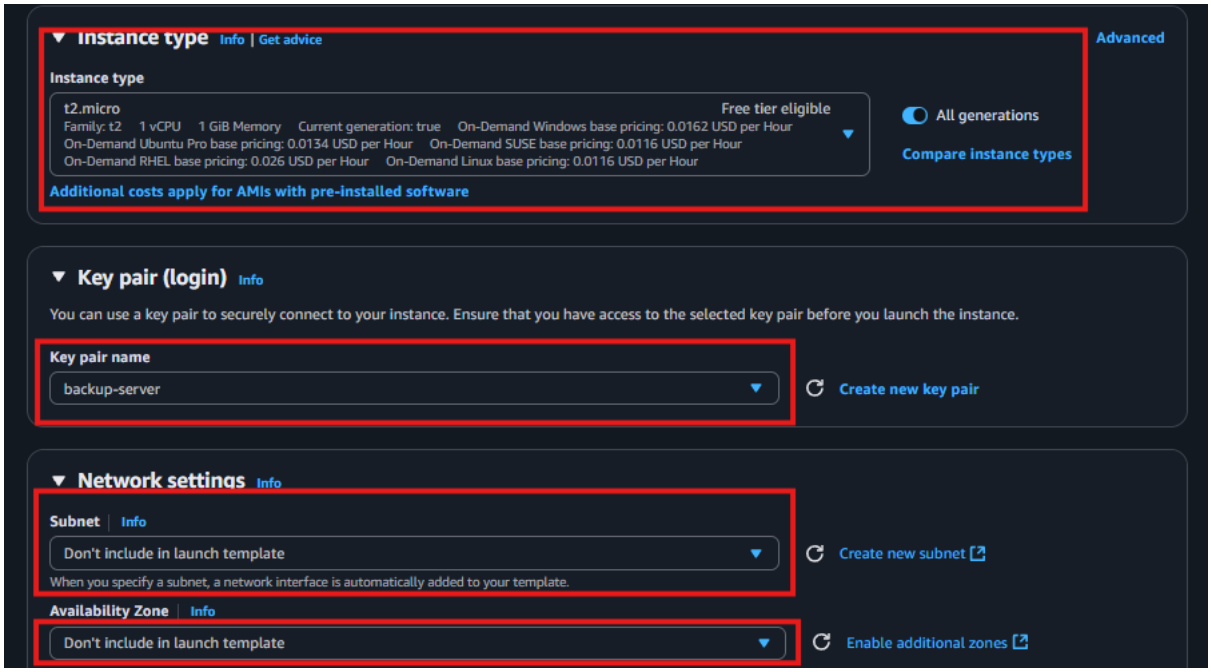


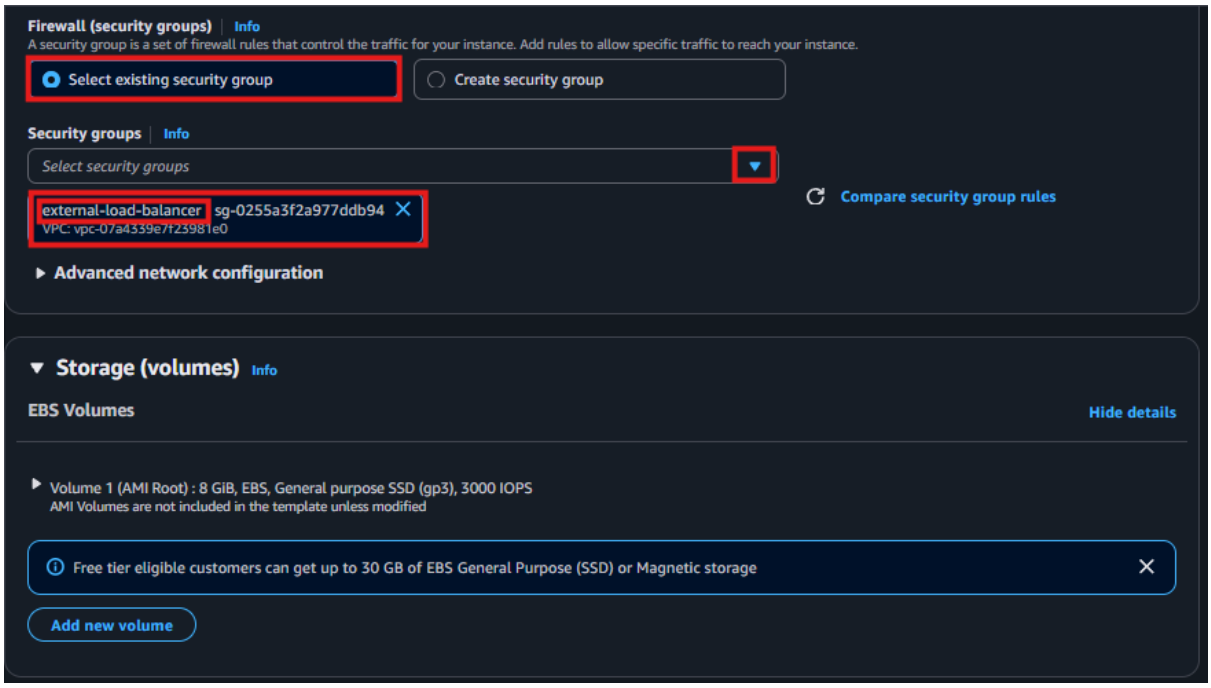
5. Set the following:

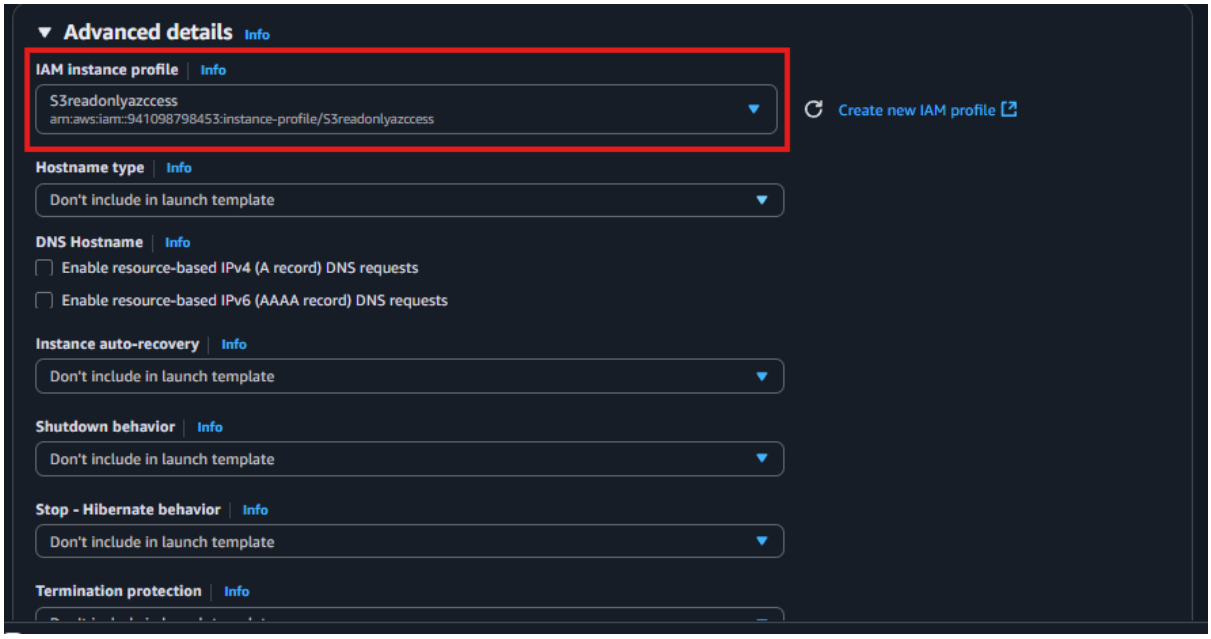
Field	Value
Launch template name	web-tier-template
AMI ID	Select the AMI created in Step 1
Instance type	(same type used for the web Server, e.g. t2.micro)
Key pair	Select the required key pair
Security group	web Server SG (allows port 80 from the External ALB)


Launch template name and description
Launch template name - required
web-tier-templete
Must be unique to this account. Max 128 chars. No spaces or special characters like '&', '*', '@'.
Template version description
web-tier-templete
Max 255 chars
Auto Scaling guidance | Info
Select this if you intend to use this template with EC2 Auto Scaling
Provide guidance to help me set up a template that I can use with EC2 Auto Scaling
Template tags
Source template


Application and OS Images (Amazon Machine Image) Info
An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose Browse more AMIs.
Search our full catalog including 1000s of application and OS images
Recents
My AMIs
Quick Start
Don't include in launch template
Owned by me
Shared with me
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
web-tier-template
ami-0ecceab054bcfc469
2025-08-17T18:43:16.000Z Virtualization: hvm ENA enabled: true Root device type: ebs Boot mode: uefi-preferred
Description
web-server-ami
Architecture
x86_64
AMI ID
ami-0ecceab054bcfc469


Instance type Info | Get advice
Advanced
Instance type
t2.micro
Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour On-Demand SUSE base pricing: 0.0116 USD per Hour
On-Demand RHEL base pricing: 0.026 USD per Hour On-Demand Linux base pricing: 0.0116 USD per Hour
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login) Info
You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.
Key pair name
backup-server
Create new key pair
Network settings Info
Subnet Info
Don't include in launch template
When you specify a subnet, a network interface is automatically added to your template.
Create new subnet
Availability Zone Info
Don't include in launch template
Enable additional zones

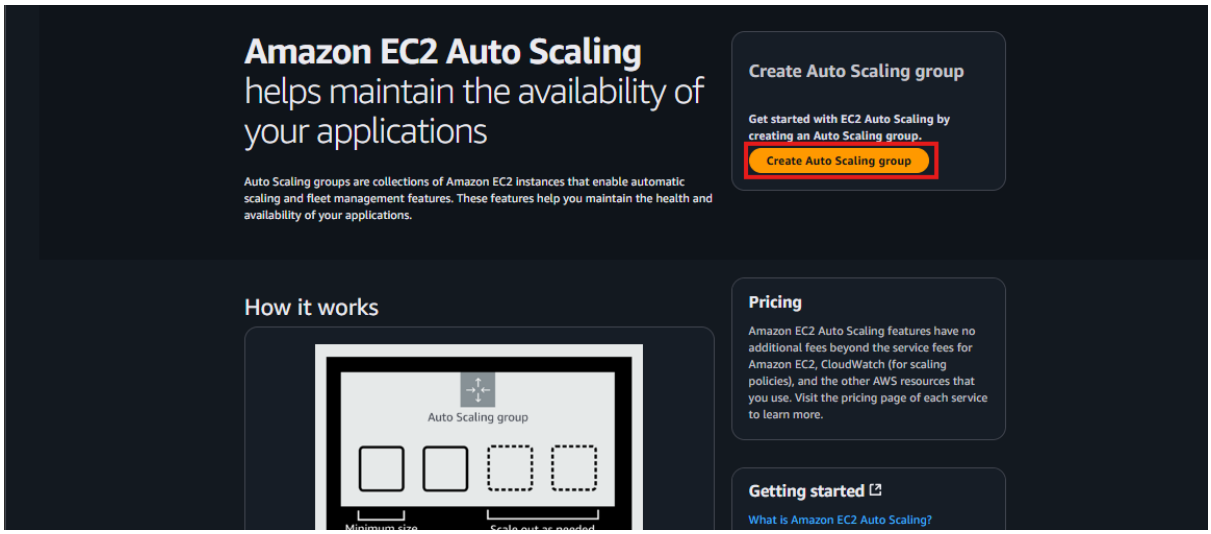

Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.
Select existing security group
Create security group
Security groups Info
Select security groups
external-load-balancer sg-0255a3f2a977ddb94
VPC: vpc-07a4339e7f23981e0
Compare security group rules
Advanced network configuration
Storage (volumes) Info
EBS Volumes
Hide details
Volume 1 (AMI Root) : 8 GiB, EBS, General purpose SSD (gp3), 3000 IOPS
AMI Volumes are not included in the template unless modified
Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage
Add new volume



6. Click Create launch template

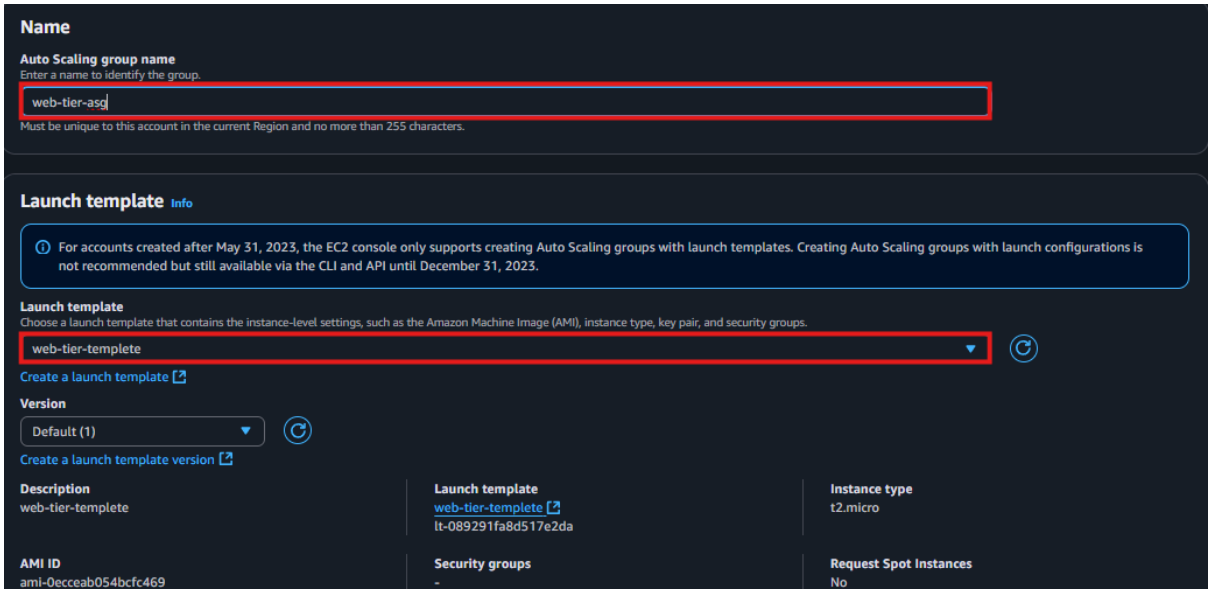
🔄 Step 3 – Create Auto Scaling Group Based on the Launch Template

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



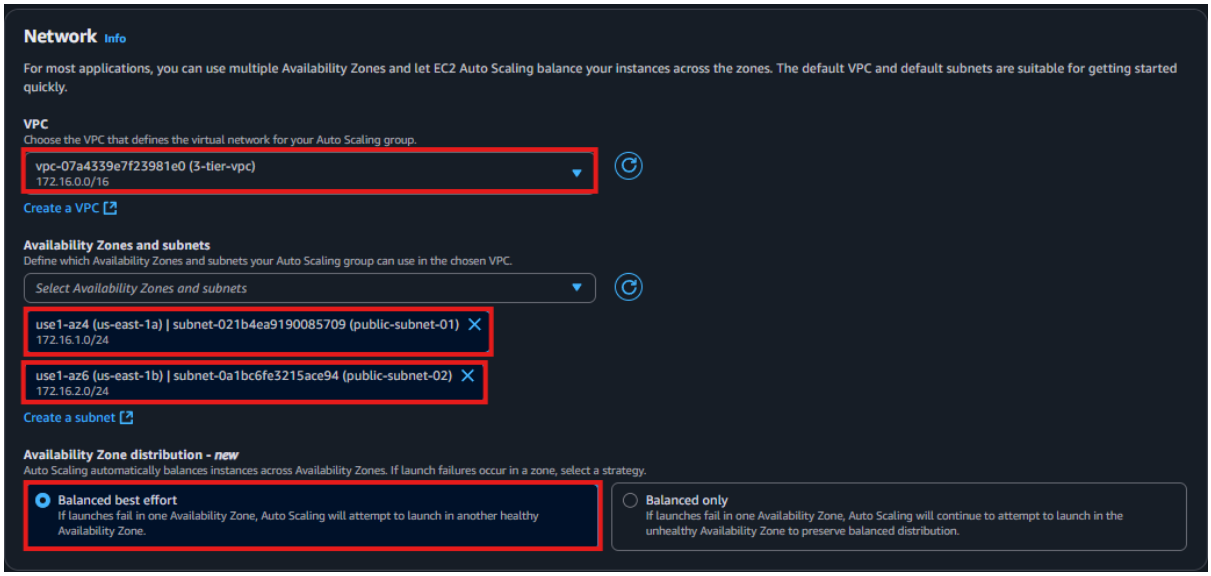
7. **Auto Scaling group name**: web-tier-asg

8. **Launch template**: select web-tier-template



9. **VPC**: Select the custom VPC

10. **Subnets**: Select **Public subnets**



Click **Next**

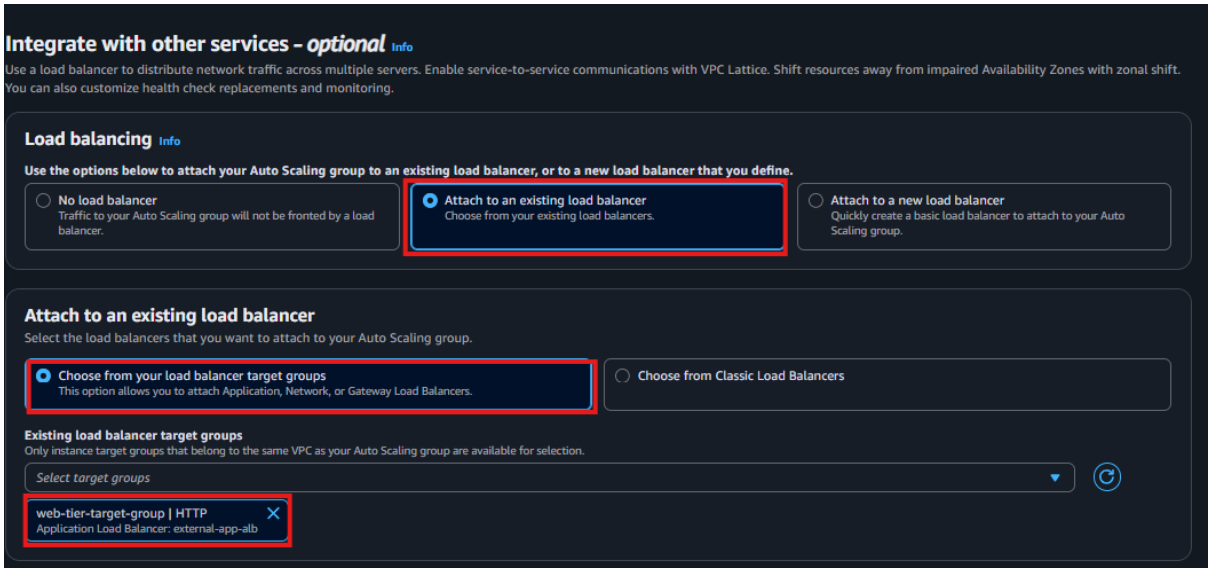
🌐 Step 4 – Attach to the External Load Balancer

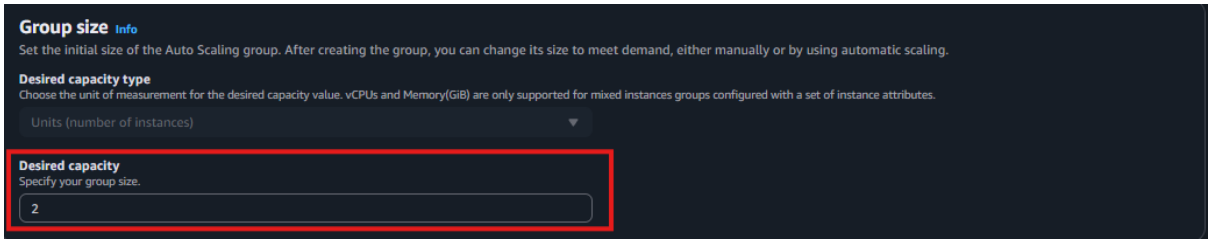
5. Select **Attach to an existing load balancer**

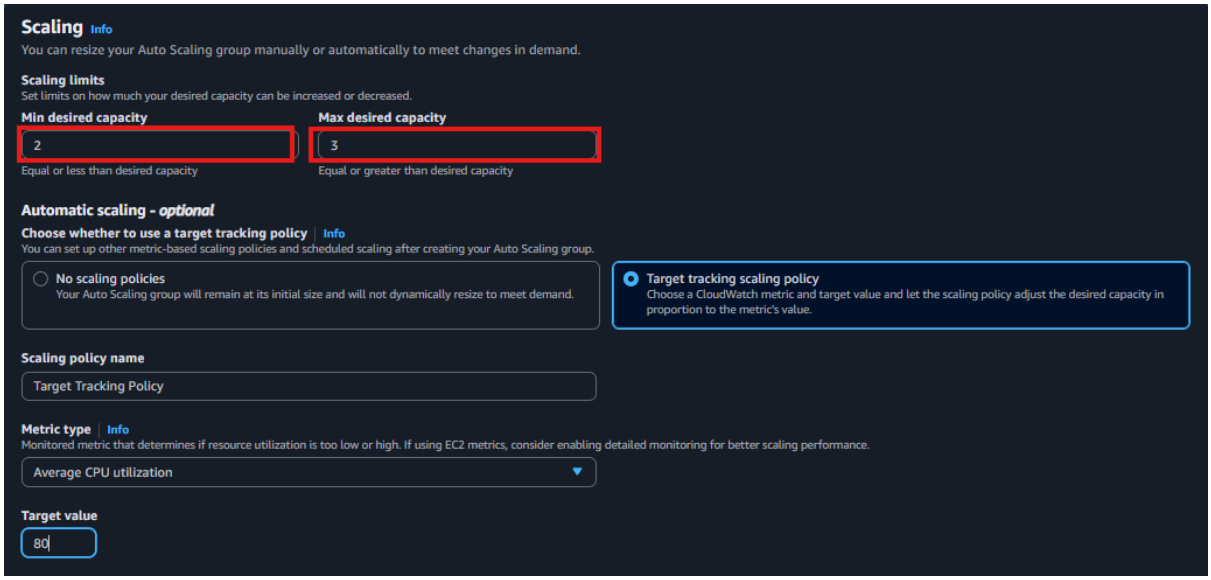
6. Choose **Application Load Balancer**

7. Select **external-app-alb**

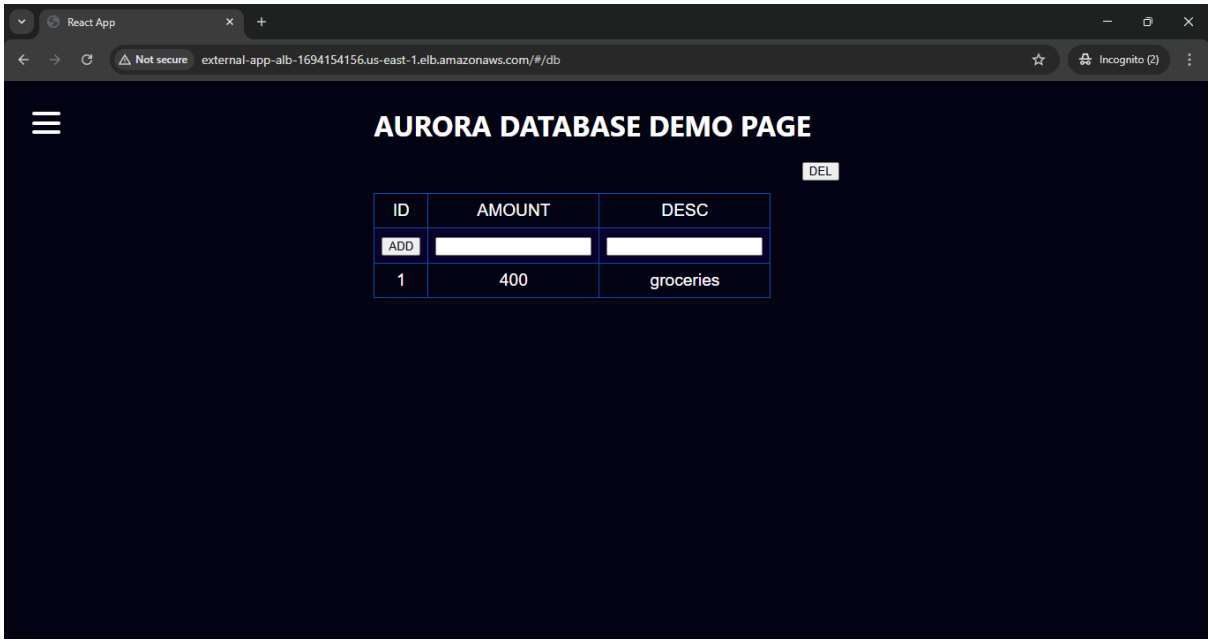
8. Select the **target group** you created for the Web Tier (e.g. Web-target-group)







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.

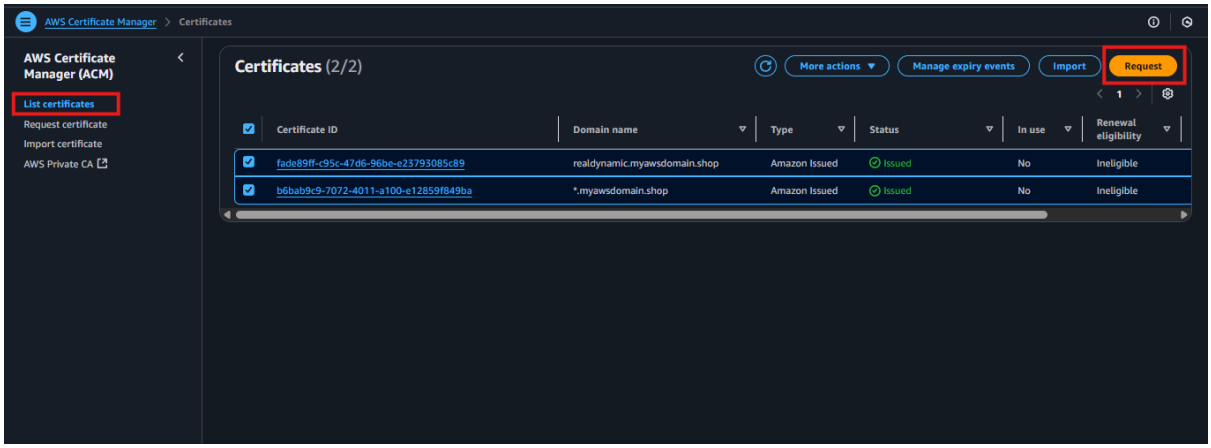
🌐 Step - 13 Configure CloudFront → External Load Balancer (with SSL)

Step 1 – Prepare Your External Load Balancer Domain

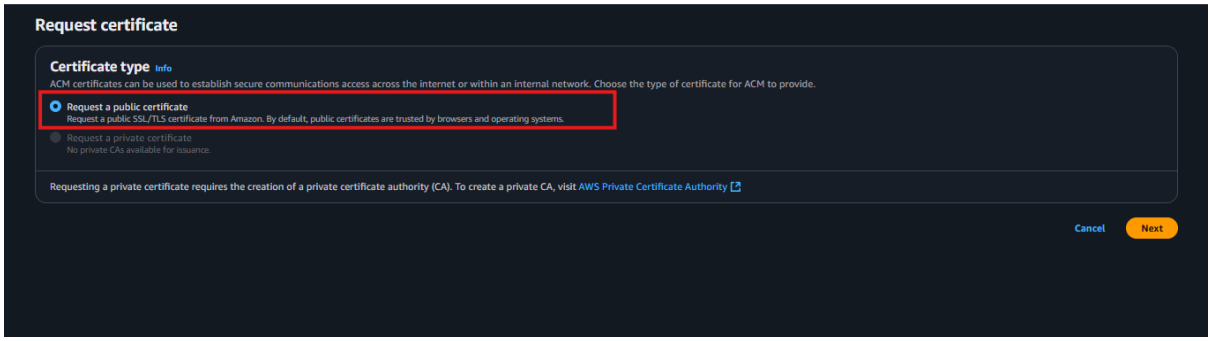
Make sure you have the DNS name of your **External Application Load Balancer** (e.g. external-web-alb-1234567890.us-east-1.elb.amazonaws.com).

Step 2 – Create an SSL Certificate in ACM

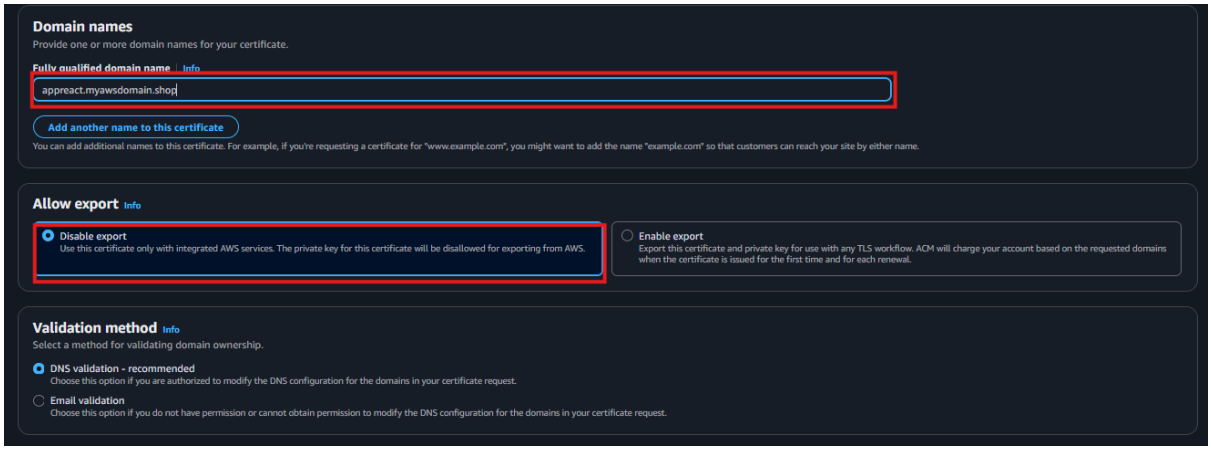
1. Navigate to **AWS Certificate Manager (ACM)** in **N.Virginia where your CloudFront distribution will run** (typically us-east-1).



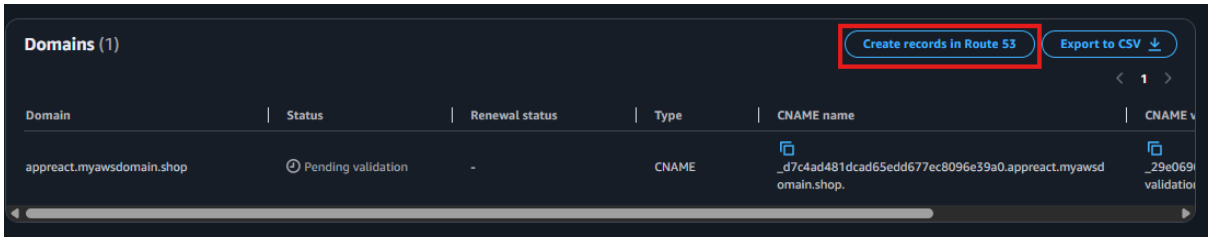
2. Click **Request a certificate → Request a public certificate**

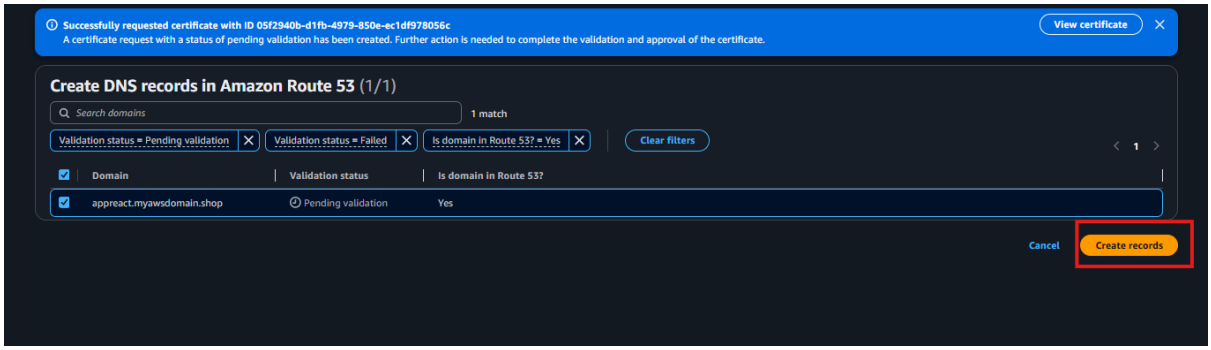


3. Enter your domain name (e.g. www.example.com or app.example.com)



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

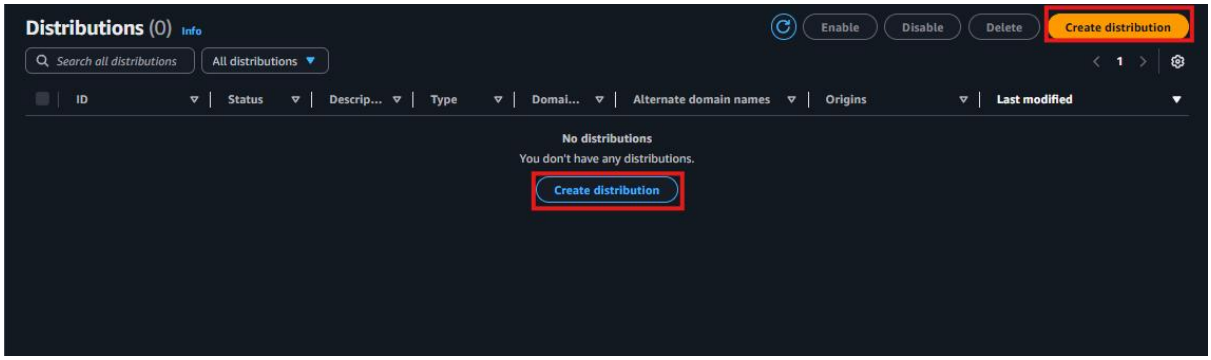




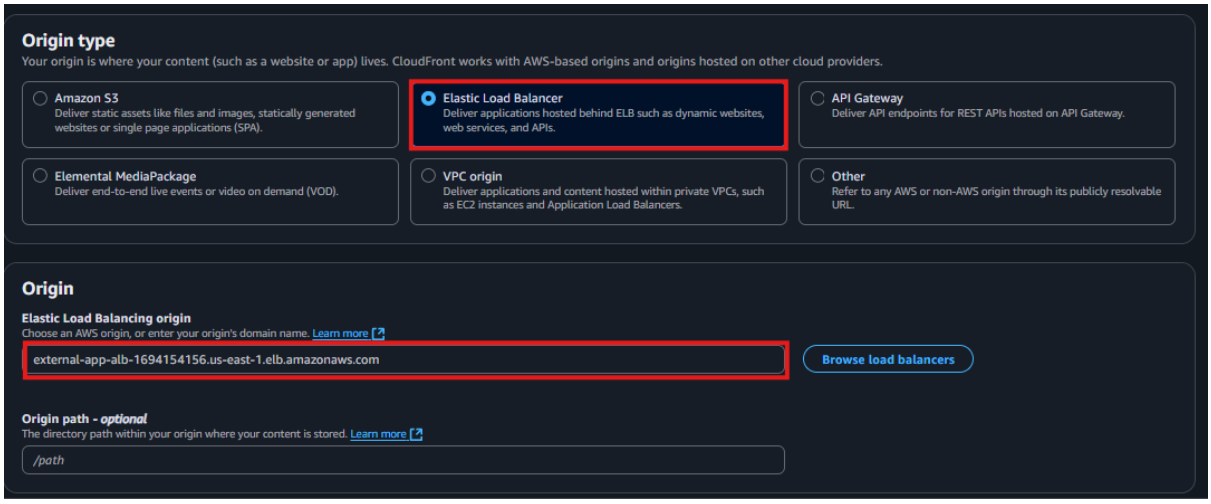
5. After validation, the certificate state becomes **Issued**

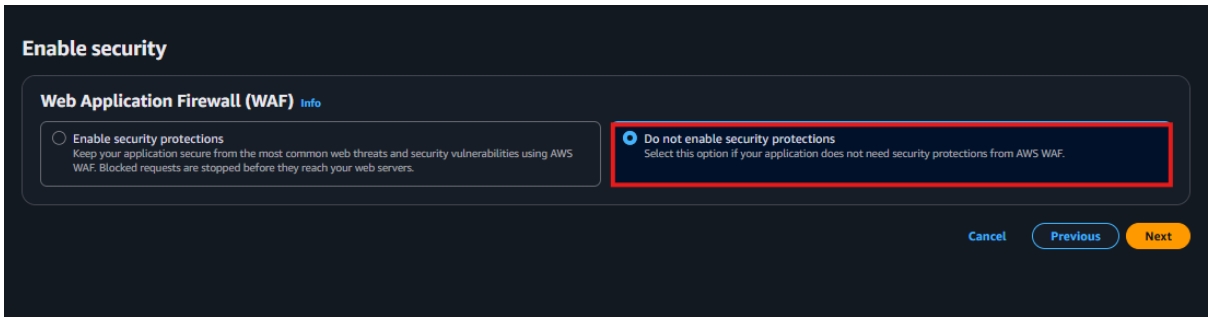
Step 3 – Create CloudFront Distribution

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

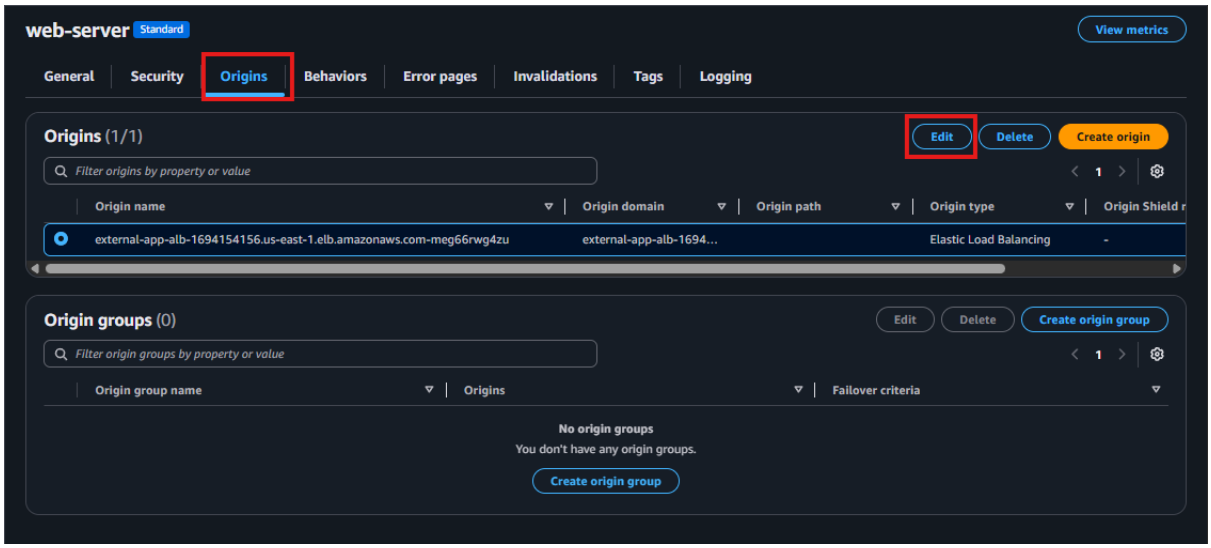


2. Under **Origin** configuration specify:

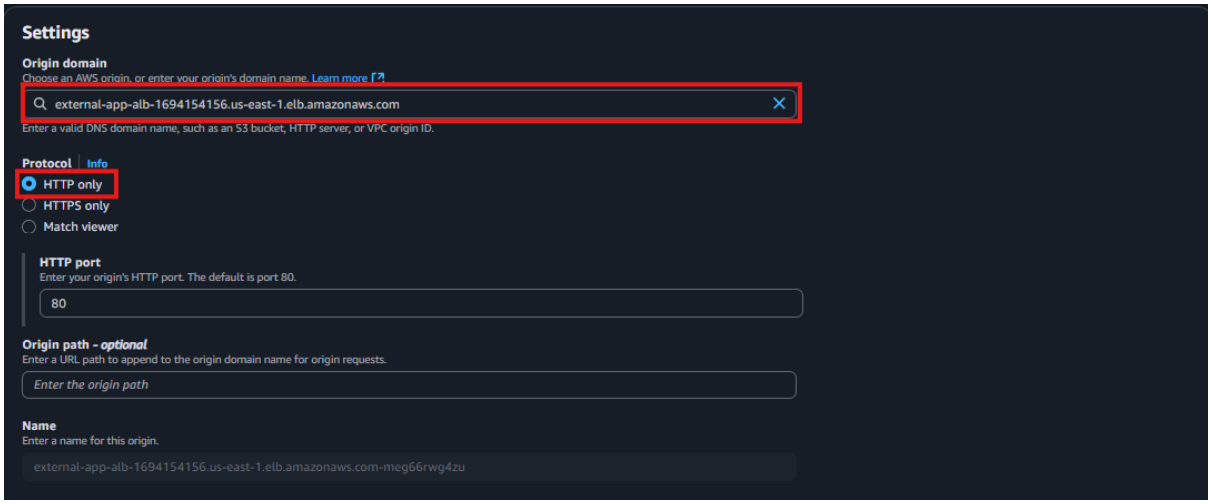




- Create a **Distribution**. Then go to **“Origins”** and click **“Edit”.**



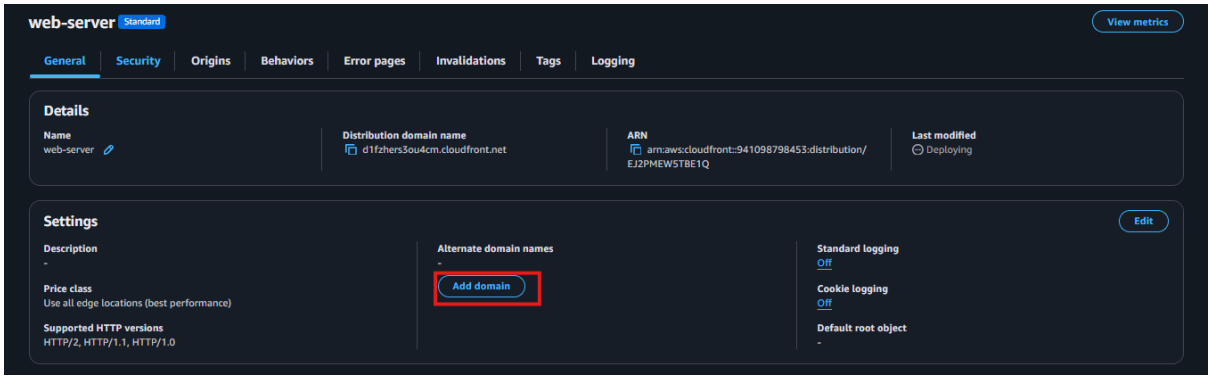
- Select port for the Origin domain in this external load balancer’s .

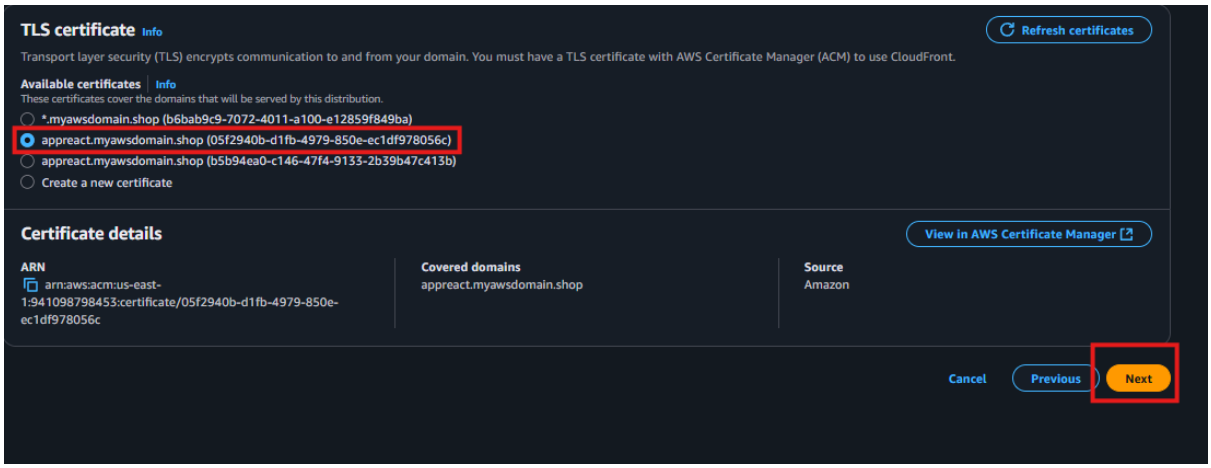


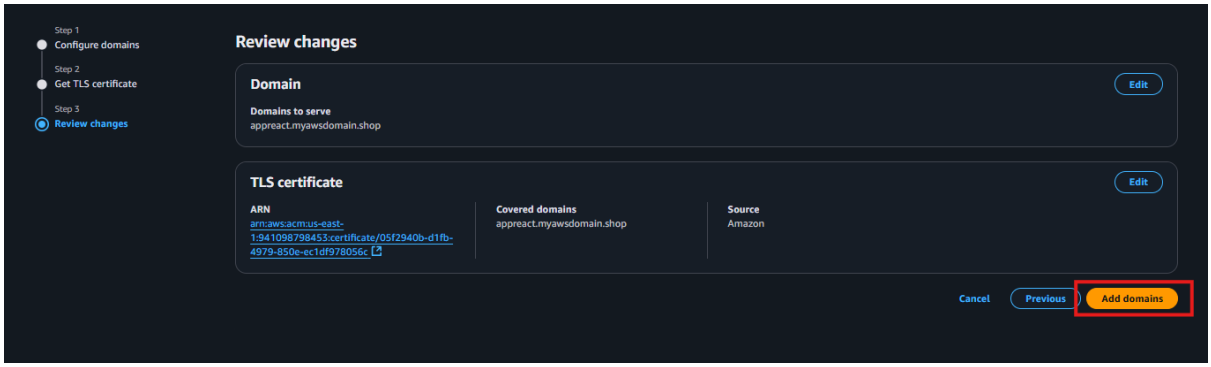
- Finally click on **"Save Changes".**

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.

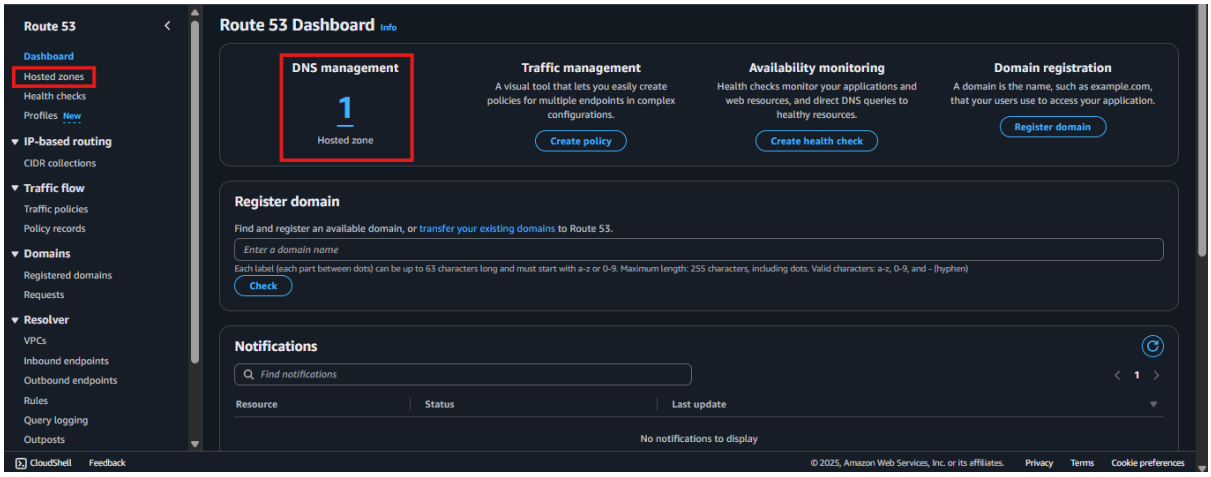




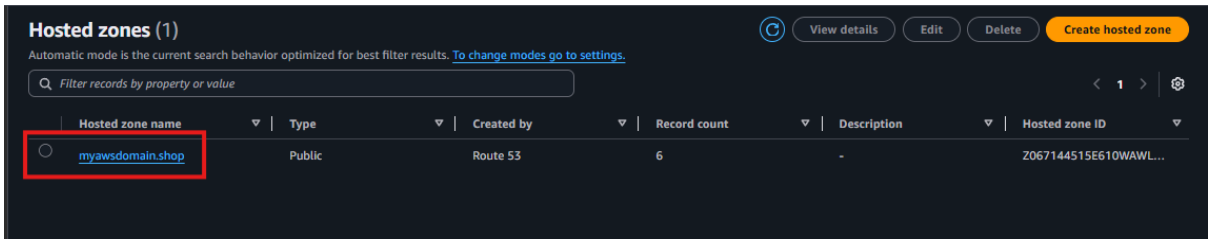


Step – 14 Final Step – Create DNS Record in Route 53

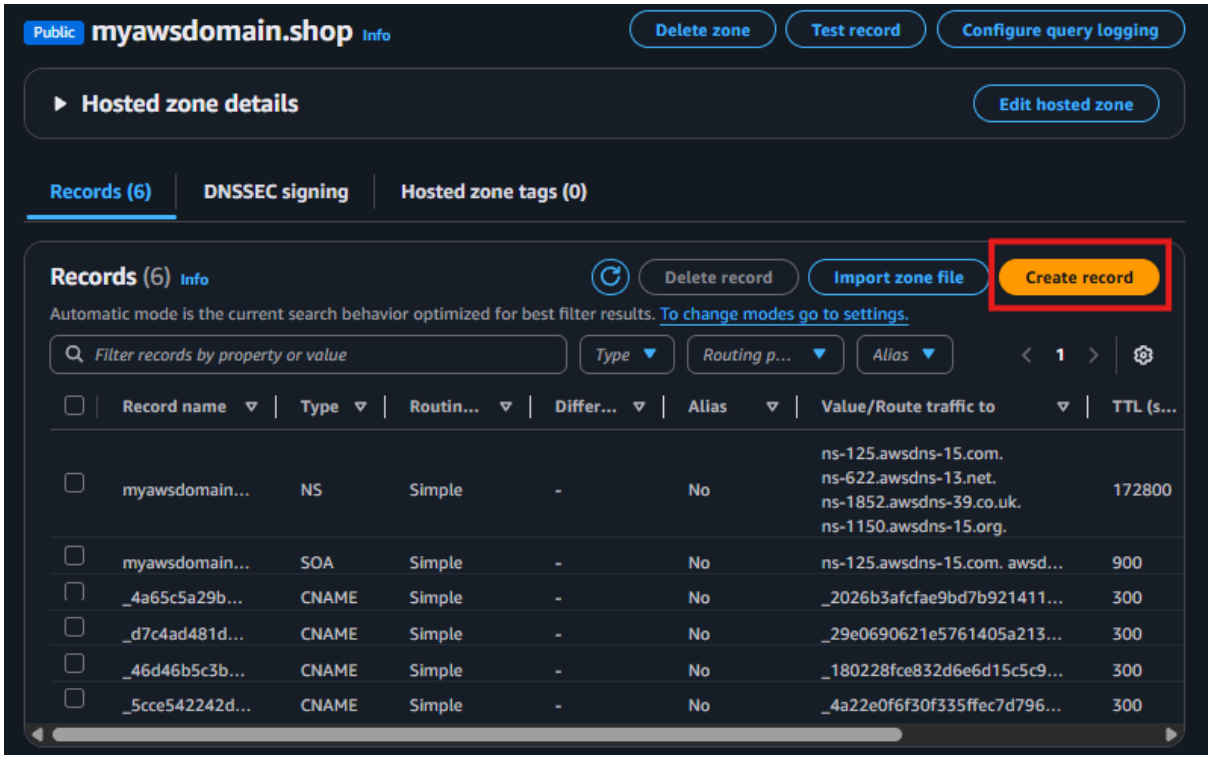
 Open the Route 53 console → Hosted Zones



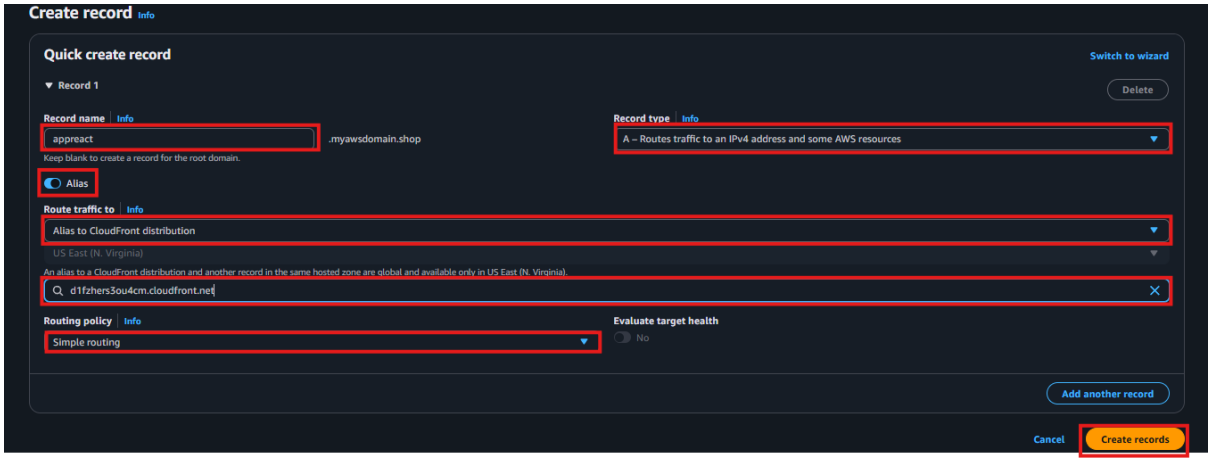
 Select the hosted zone for your domain (e.g. example.com)



 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:

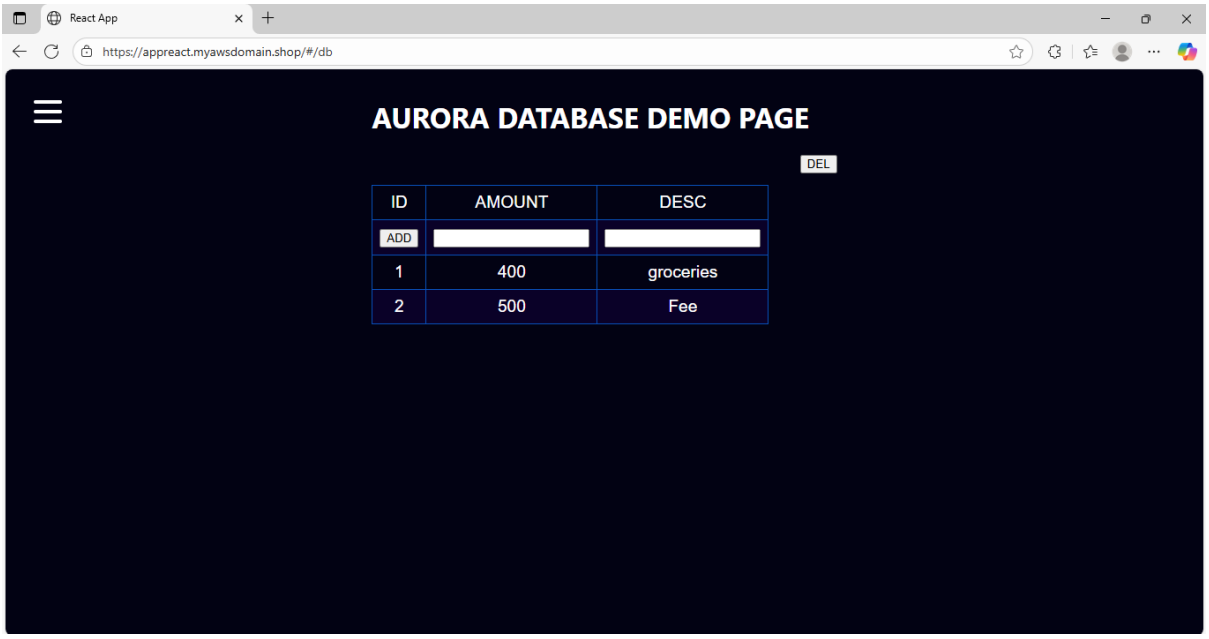
- **Web Tier** – Publicly accessible EC2 instances behind an **External Application Load Balancer**, serving the front-end and forwarding API requests to the application tier.

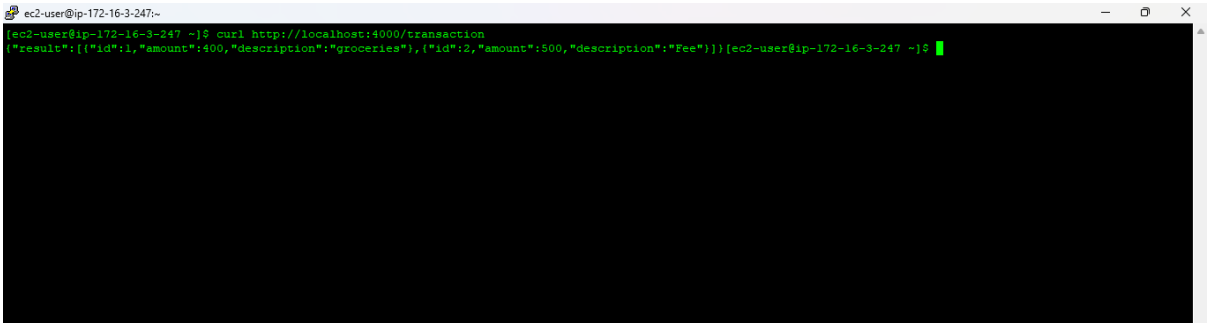
- **Application Tier** – EC2 instances hosted in private subnets and fronted by an **Internal Application Load Balancer**, processing business logic and API requests.

- **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.


React App
https://appreact.myawsdomain.shop/#/db
AURORA DATABASE DEMO PAGE
DEL
ID
AMOUNT
DESC
ADD
1
400
groceries
2
500
Fee


ec2-user@ip-172-16-3-247:~
[ec2-user@ip-172-16-3-247 ~]$ curl http://localhost:4000/transaction
{"result":[{"id":1,"amount":400,"description":"groceries"},{"id":2,"amount":500,"description":"Fee"}]}[ec2-user@ip-172-16-3-247 ~]$