AWS Three-Tier Architecture Project

- AWS Three-tier architecture is used to separate the application into three distinct layers namely Web-tier, App-tier and Database-Tier.

Web-Tier:

This Layer is responsible for displaying the user-interface. It handles user input, displays information, and manages the user experience.

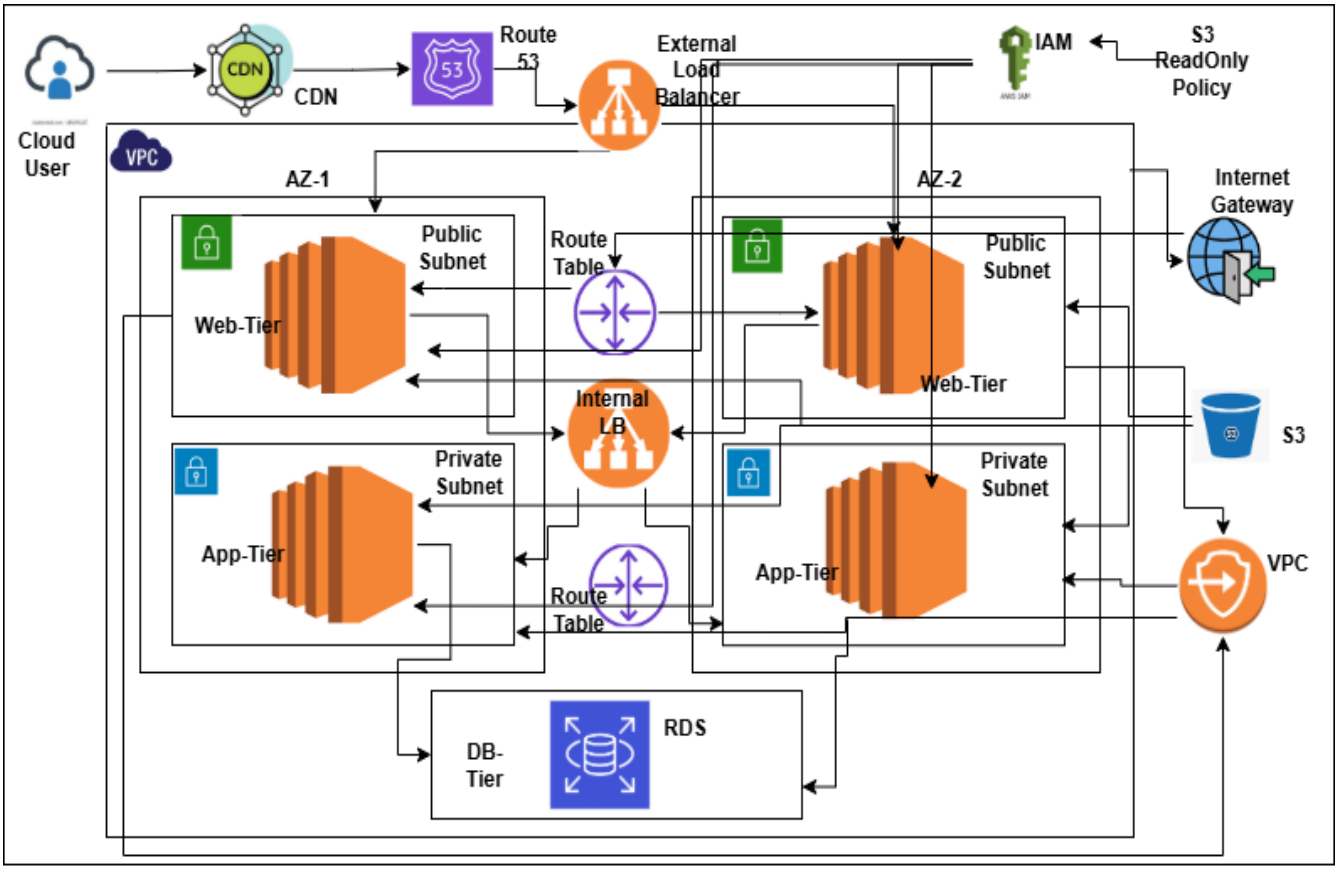
App-Tier:

This Layer contains the business logic of the application, it implements business rules, and interact with database tier to manage and retrieve data.

Database-Tier:

This Layer handles database operation, ensuring data integrity and security.

3-Tier Architecture Diagram

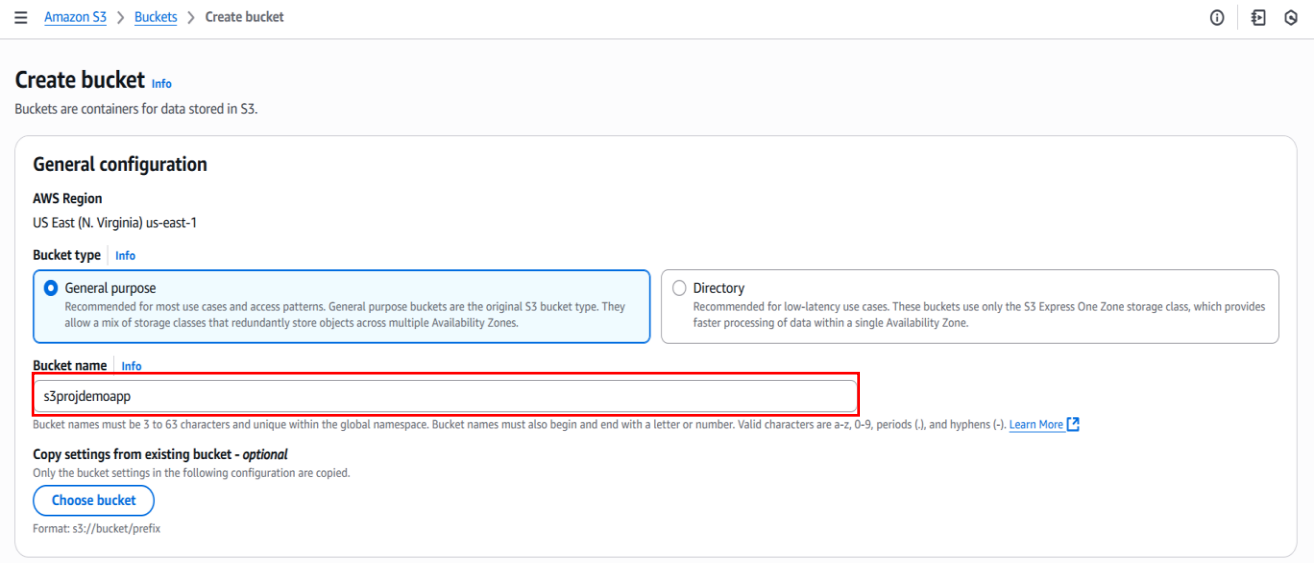


Project Flow:

Step1: S3 bucket Creation

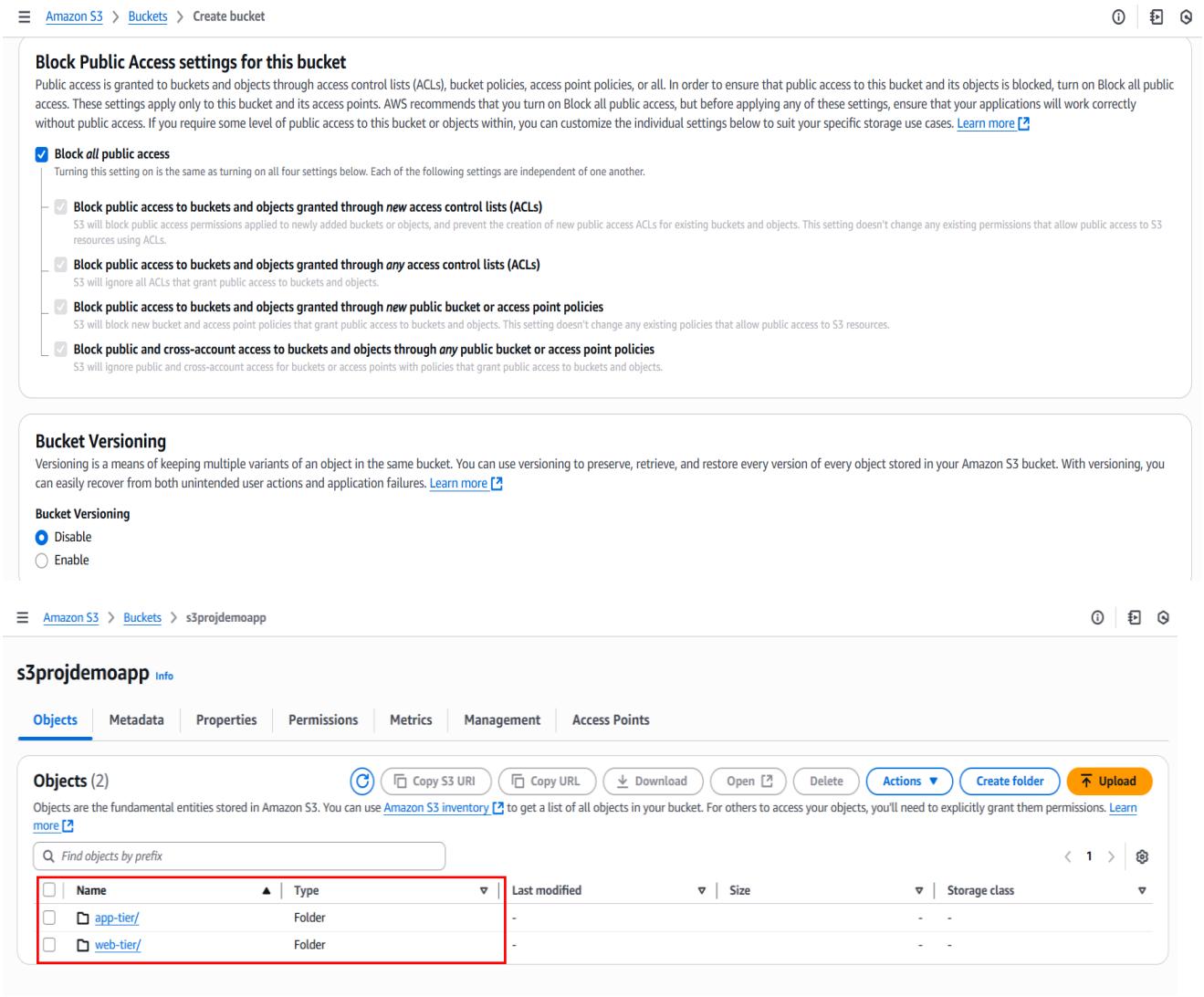
An Amazon S3 bucket is a fundamental storage resource within AWS S3. It functions as a container for storing objects, which are essentially files (like images, documents, or application data) along with any associated metadata.

- Download the code from Github Repository into your local machine.
- Create the Unique S3 Bucket.
- Node JS Application contains Web-tier and App-tier folder.
- Uploading the web-tier and app-tier folder inside the S3 Bucket.



→ By default, S3 bucket is Private, Bucket versioning is disabled.

→ If we want we can enabling the bucket versioning to recover the previous files.



→ S3 Bucket created and uploaded the folders.

Step2: IAM Role Creation

IAM is an AWS security service that provides secure control over access to AWS resources. It enables administrators to manage users, groups, roles, and permissions, dictating who can be authenticated (signed in) and authorized (granted permissions) to use AWS services and resources.

Create the **IAM** role for **AmazonS3ReadOnlyAccess policy** to access the S3 bucket from EC2 instance.

AmazonS3ReadOnlyAccess

Provides read only access to all buckets via the AWS Management Console.

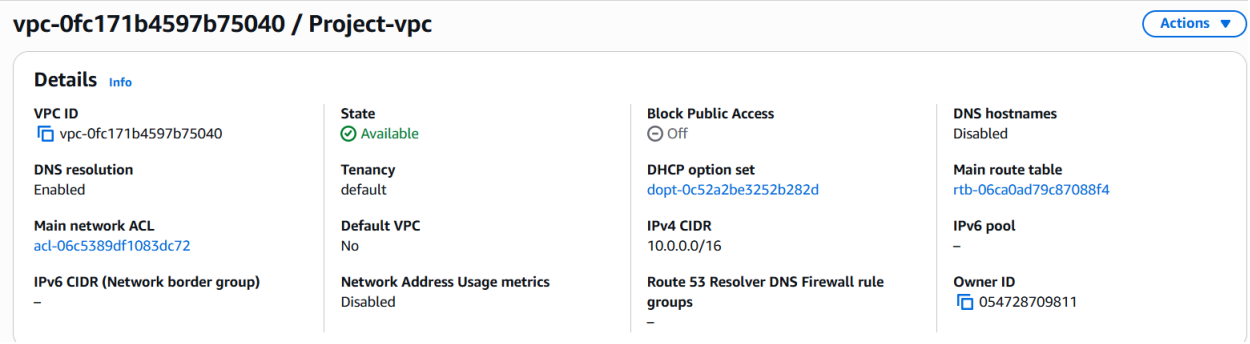
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*",
                "s3:List*",
                "s3:Describe*",
                "s3-object-lambda:Get*",
                "s3-object-lambda:List*"
            ],
            "Resource": "*"
        }
    ]
}
```

Step3: VPC Creation

VPC is a logically isolated section of the AWS cloud that allows you to launch AWS resources in a virtual network that you define.

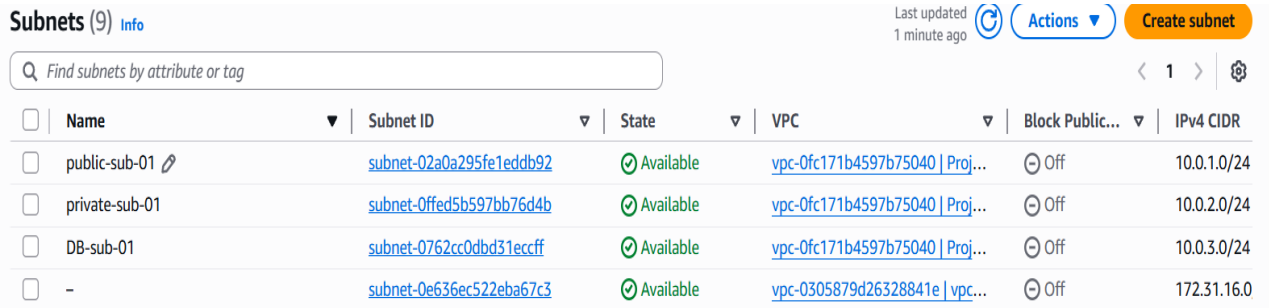
VPC Creation

Creating the **VPC** to customize the network configuration within AWS.



- Creating the **public** and **private Subnets** with multiple Available Zone within the VPC.

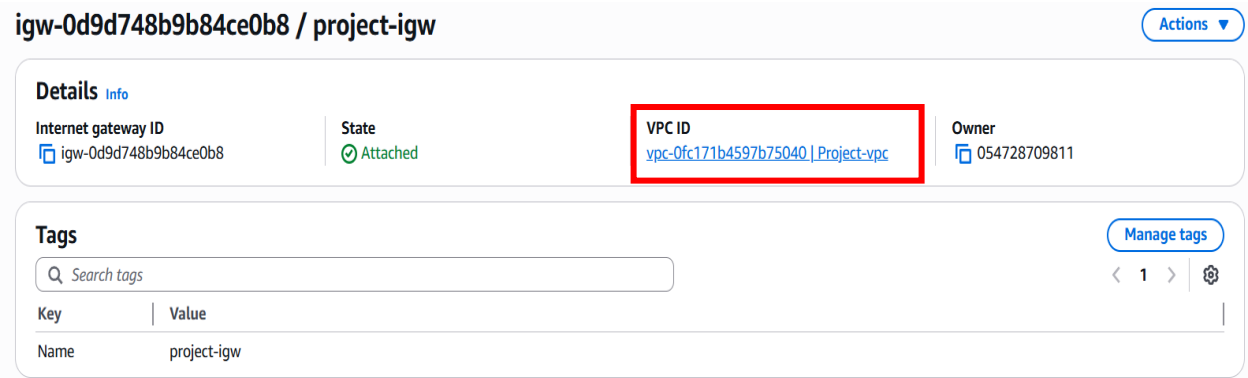
Subnets:



Creating Internet Gateway

- An AWS Internet Gateway (IGW) is a horizontally scaled, redundant, and highly available VPC component that provides a connection between your Virtual Private Cloud (VPC) and the internet.
- Internet gateway supports both inbound and outbound rules.

→ Creating Internet gateway and attach with VPC.



Creating NAT Gateway

- NAT Gateway in AWS is a managed Network Address Translation (NAT) service that allows instances in a private subnet to access the internet or other VPCs without being directly exposed to the public internet.
- It acts as a bridge, enabling outbound traffic from private instances while preventing unsolicited inbound connections.

→ Creating NAT gateway and attach with VPC.

→ Make the Connectivity type as Public to allows instances within a private subnet to access the internet.

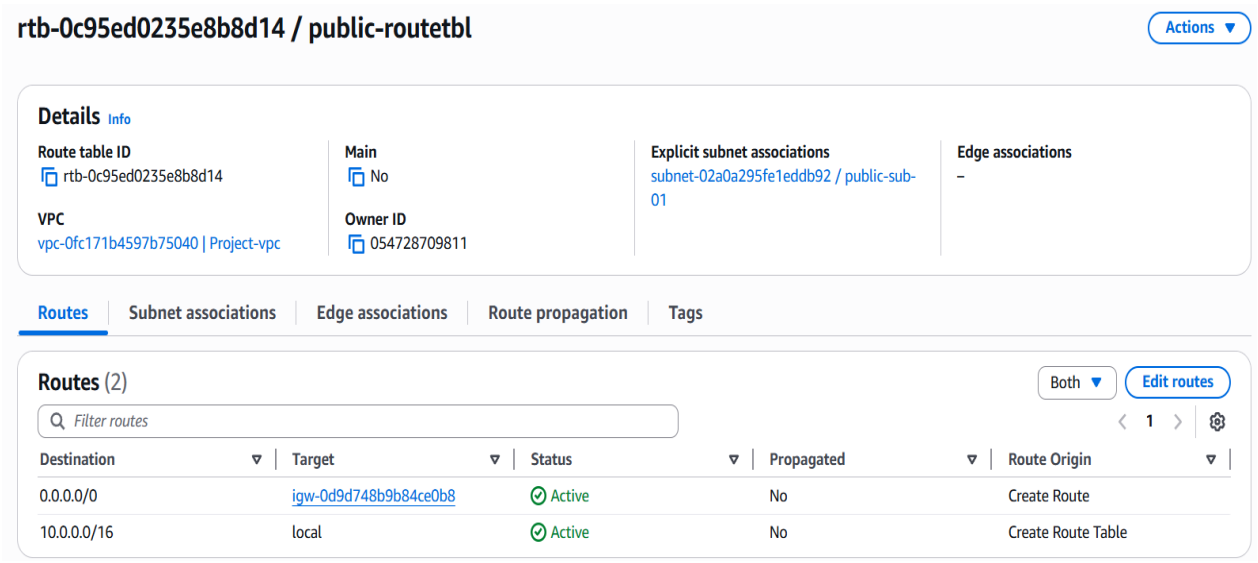
→ Allocate Elastic IP Address, allow to enable instances in a private subnet to connect to the internet while preventing inbound connections from the internet. The EIP acts as a static, public IPv4 address that is associated with the NAT Gateway, allowing it to initiate outbound connections and translate private IP addresses to public ones.

Route Tables

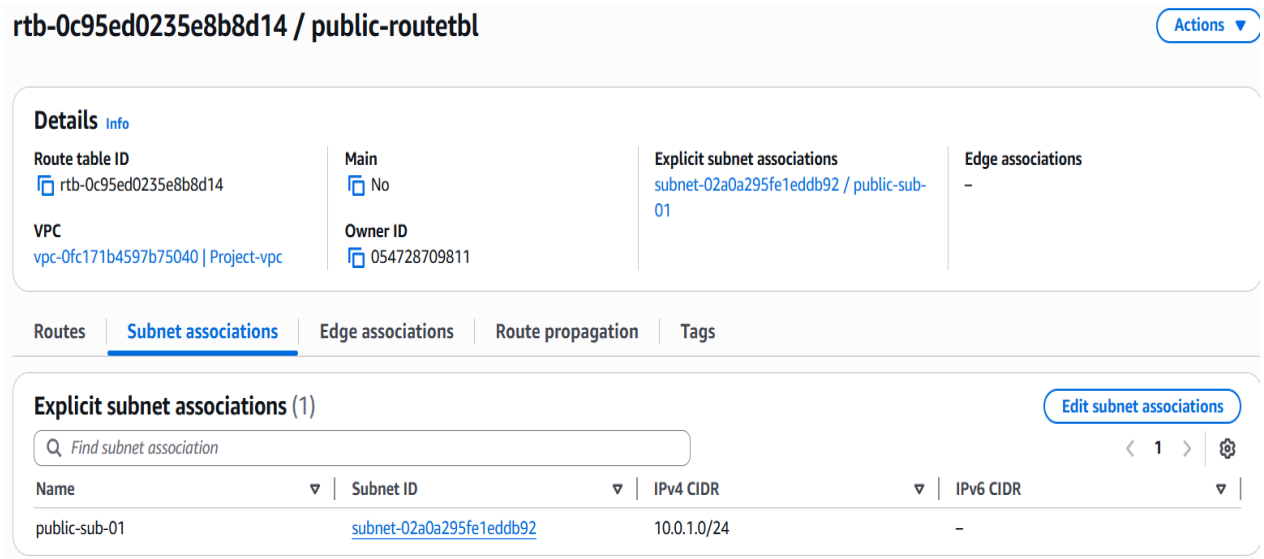
- Create public and private **Route tables** and associate subnets respectively and attach **Internet Gateway** in public route table and attach the **NAT gateway** in public subnet as well as private route table to provide internet access for private subnets. NAT gateway is used to allow only outbound rules.

Public Route Tables:

→ Edit Routes – Adding Internet Gateway in public route table.

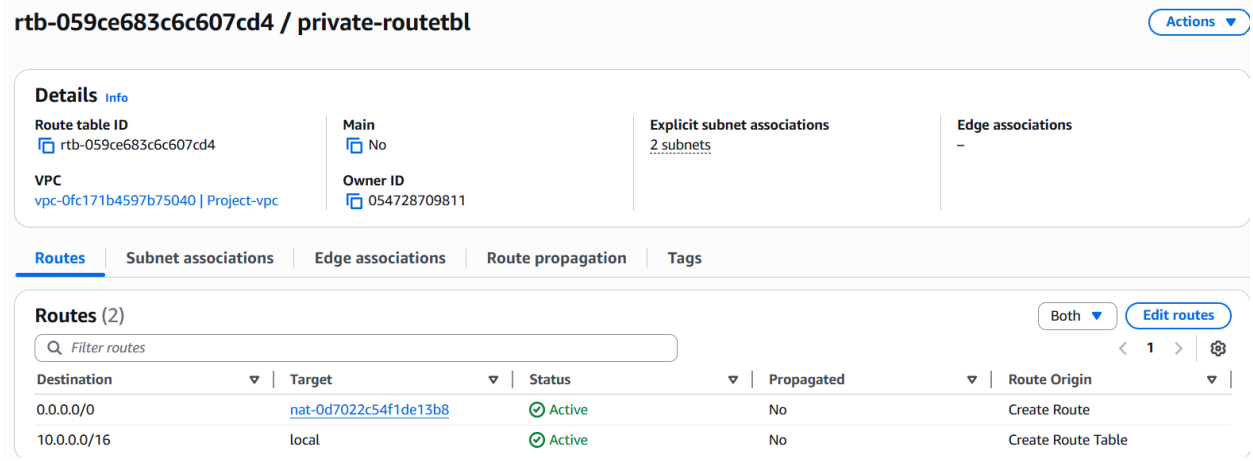


→ Subnet associations – Adding Public subnets in public route table.

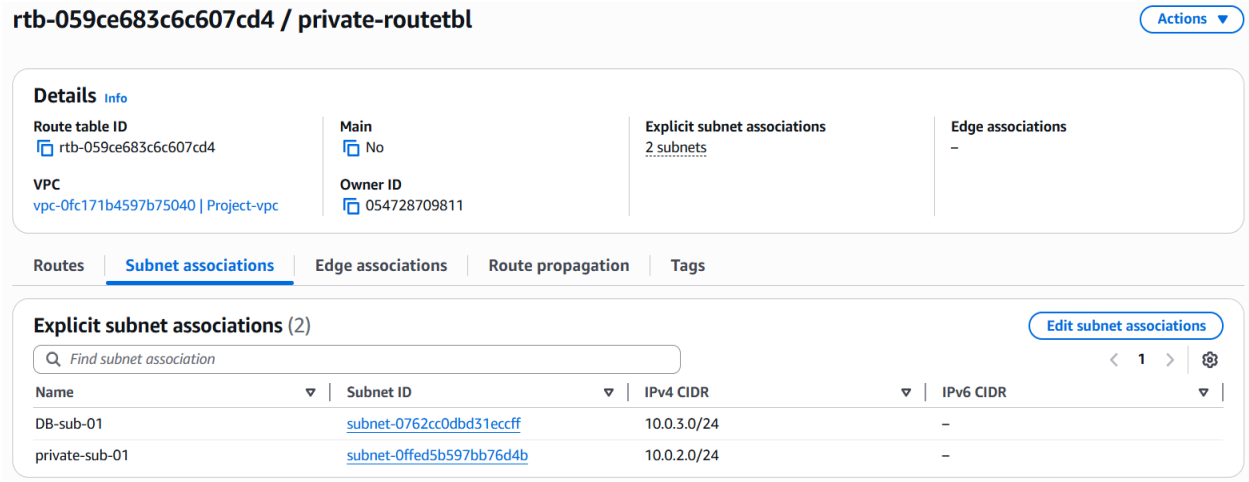


Private Route Tables:

→ Edit Routes – Adding NAT Gateway in private route table.



→ Subnet associations – Adding Private subnets in private route table.



Step4: Security Group Creation

Creating **Security group**, it acts as a firewall that controls the inbound and outbound traffics.

We need to create six Security group each for different purposes.

External Load Balancer: Allow Http (port 80) to anyone.

Jump Server: Allow SSH (port 22) to anyone.

Web Server: Http (port 80) to External Load Balancer,

Allow SSH to Jump Server.

Internal Load Balancer: Allow Http (port 80) to Web Server.

App Server: Allow TCP port 4000 to Internal Load Balancer,

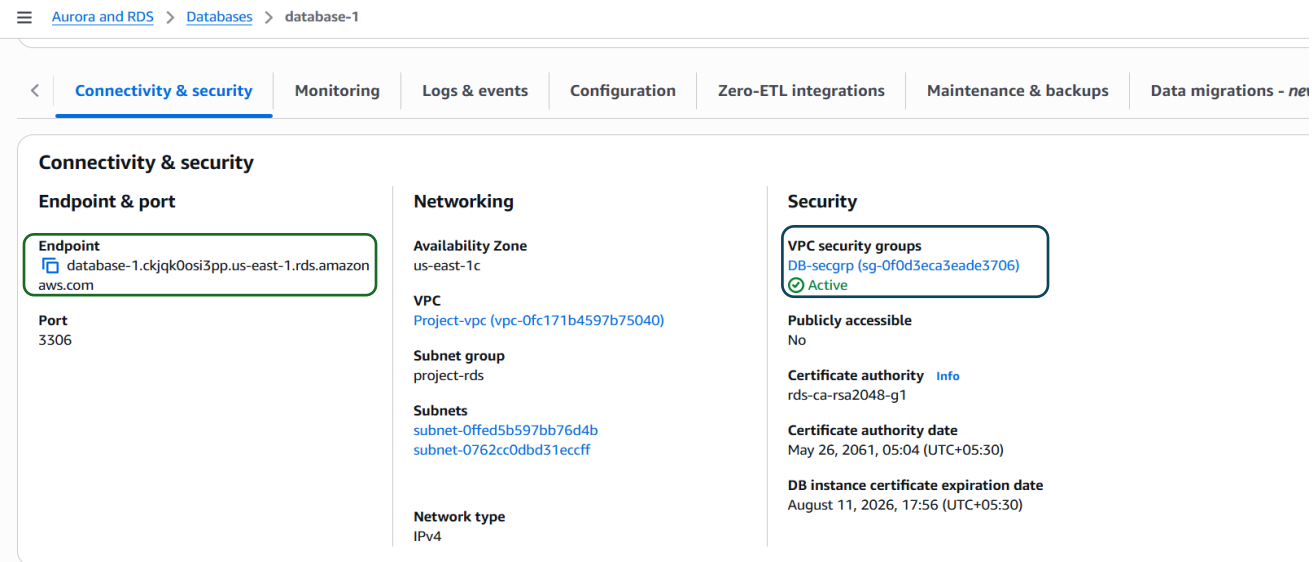
SSH (port 22) to Jump Server.

Database Server: Allow MySQL/aurora port 3306 to App Server.

Security group ID	Security group name	VPC ID	Description
sg-0d9dcf19241100275	Internal-LB	vpc-0fc171b4597b75040	Internal-LB
sg-0832c71b2686856a0	Webserver	vpc-0fc171b4597b75040	Webserver
sg-010137275c5aa8165	Jump-Server	vpc-0fc171b4597b75040	Jump-Server
sg-08ddfcb80763deffd	App-Server	vpc-0fc171b4597b75040	App-Server
sg-0f0d3eca3eade3706	DB-secgrp	vpc-0fc171b4597b75040	DB-secgrp
sg-0c18959cb40d2fe74	Public-LB	vpc-0fc171b4597b75040	Public-LB

Step5: Database-Tier Configuration

- Goto RDS → Create DB instances → Create Database
- Select Engine Option: MySQL → Edition: MySQL Community
- Deployment Option: Select Availability Zone type.
- Settings → Credentials management : Self-Managed, set username and password
- Select custom VPC you have created
- Select Security group → choose existing : Database security group
- Select Availability Zone
- Additional settings like Log Reports (optional)



Database created successfully.

Step6: Procedure to Create the EC2 instance

EC2 (Elastic Compute Cloud) instance is a virtual server in the cloud, providing resizable compute capacity that allows user to run applications.

→ Launce Instance → Name → AMI (Application Machine Image) browse AMI → Provides pre-configured template to launch Ec2 instance.

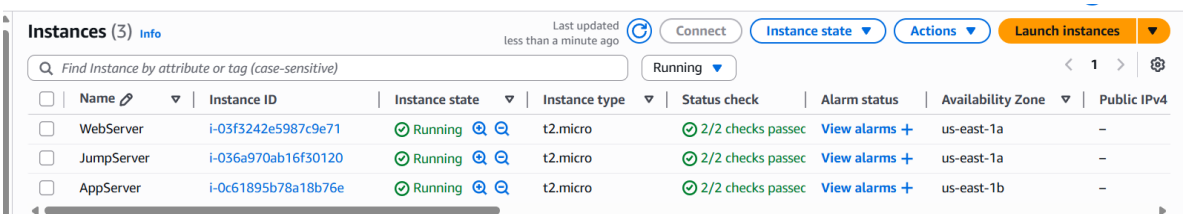
→ Select Instance Type : t2.micro (which is free)

→ Create Key pair → Key pair name, key pair type - RSA (which is encryped private and public key pair → Select Private key file format → .pem Private Key file format types:

- .pem Private Enhanced Mail (for openssh)
- .ppk PuTTy Private key (for putty)

→ Network Settings: choose VPC, Subnet, Available zone, Auto-assign public ip, Security group, Configure Inbound security rules.
→ Advanced Networking configurations : We can assign Primary Ip , Secondary Ip.
→ User data (optional): we can run bash scripting which helps to run the script during Ec2 server creation.

- Create Ec2 instance for Jump Server → select VPC, select **Public** subnet with multiple AZ, auto assign public Ip: **enable**, Select Security group: Jump server security group, set primary Ip and secondary Ip (optional) Eg. 10.0.1.5.
- Create Ec2 instance for Web Server → select VPC, select **Public** subnet with multiple AZ, auto assign public Ip: **enable**, Select Security group: Web server security group, set primary Ip and secondary Ip (optional) Eg. 10.0.1.15.
- Create Ec2 instance for App Server → select VPC, select **Private** subnet with multiple AZ, auto assign public Ip: **disable**, Select Security group: App server security group, set primary Ip and secondary Ip (optional) Eg. 10.0.2.5.
- Select Ec2 instance → Goto Action → Security → Modify IAM and Attach the IAM role for Jump Server, Web Server, App Server.
- Jump Server: It is responsible for safely bypassing firewalls that isolate the public and private networks. It acts as a bridge between different security zones. Every cloud user should communicate with Application only through the Jump server.



Step7: App-Tier Configuration

7a. Goto app server

➡ Open jump server ➡ Create the pem key and provide read permission
sudo chmod 400 key.pem

➡ Goto app server through SSH credentials
ssh –i <keypair> <username>@ip i.e., ssh –i key.pem ec2-user@10.0.2.5

7b. Install MySQL Client and Connect

Commands to 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

Master Username: admin
password: Admin123Pwd
endpoint (hostname): database-1.ckjqk0osi3pp.us-east-1.rds.amazonaws.com
Command to Login MySQL DB Engine:
mysql -h <Host_Name> -P 3306 -u <User_Name> -p

mysql -h database-1.ckjqk0osi3pp.us-east-1.rds.amazonaws.com -P 3306 -u admin –p

Enter the password:

```
[root@ip-10-0-2-6 ec2-user]# mysql -h database-1.ckjqk0osi3pp.us-east-1.rds.amazonaws.com -P 3306 -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 36
Server version: 8.0.39 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| webappdb           |
+--------------------+
5 rows in set (0.002 sec)

MySQL [(none)]> use webappdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

➡Your database is now connected...If not connected, then check your credentials and Security group.

7c. Create Database, table and Insert the record in db.

SQL commands:

Create the database	create database <database_name>; Create database webappdb;
Show database list	Show database;
Create table in database	Use webappdb; CREATE TABLE IF NOT EXISTS transactions (id INT NOT NULL AUTO_INCREMENT, amount DECIMAL(10,2), description VARCHAR(100), PRIMARY KEY (id));
Show table list	SHOW TABLES;
Insert record	INSERT INTO transactions (amount, description) VALUES ('400', 'groceries');
Select record	SELECT * FROM transactions;

```
MySQL [(none)]> create database webappdb;
Query OK, 1 row affected (0.021 sec)

MySQL [(none)]> use webappdb;
Database changed
MySQL [webappdb]> create table if not exist transactions (
    -> id int not null auto_increment,
    -> amount decimal(10,2),
    -> description varchar(100),
    -> primary key (id)
    ->
    -> )
    -> ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'exist transactions (
id int not null auto_increment,
amount decimal(10,2),
descr' at line 1
MySQL [webappdb]> create table if not exists transactions ( id int not null auto_increment, amount decimal(10,2), description varchar(100), primary key (id));
Query OK, 0 rows affected (0.054 sec)

MySQL [webappdb]> show tables
    -> ;
+--------------------+
| Tables_in_webappdb |
+--------------------+
| transactions       |
+--------------------+
1 row in set (0.002 sec)

MySQL [webappdb]> insert into transactions (amount,description) values (1000,'diarys');
Query OK, 1 row affected (0.011 sec)

MySQL [webappdb]> select * from transactions
    -> ;
+----+---------+-------------+
| id | amount  | description |
+----+---------+-------------+
|  1 | 1000.00 | diarys      |
+----+---------+-------------+
1 row in set (0.001 sec)
```

```
| Tables_in_webappdb |
+--------------------+
| transactions       |
+--------------------+
1 row in set (0.002 sec)

MySQL [webappdb]> select * from transactions;
+----+---------+-------------+
| id | amount  | description |
+----+---------+-------------+
| 18 |  100.00 | fruits      |
| 19 |  200.00 | Almond      |
| 20 | 7868.00 | uynyi       |
+----+---------+-------------+
3 rows in set (0.001 sec)

MySQL [webappdb]> insert into transactions (amount,description) values (500,'Ghee');
Query OK, 1 row affected (0.005 sec)

MySQL [webappdb]> select * from transactions;
+----+---------+-------------+
| id | amount  | description |
+----+---------+-------------+
| 18 |  100.00 | fruits      |
| 19 |  200.00 | Almond      |
| 20 | 7868.00 | uynyi       |
| 21 |  500.00 | Ghee        |
```

→ after inserting the records, then Type Exit

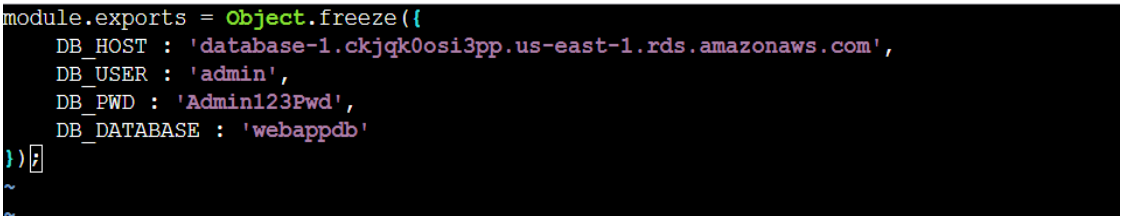
7d. Copy the S3 bucket App-tier folder into App server

In App server, make directory for App-tier folder content,
sudo mkdir app-tier
cd app-tier
Command to copy s3 bucket folder into app server:
aws s3 cp s3://s3projdemoapp/app-tier/ . --recursive
aws s3 ls → to list all content inside s3 bucket

Update the hostname, username, password, database in DbConfig.js file.

Open → app-tier → DbConfig.js

```
module.exports = Object.freeze({
DB_HOST : ' database-1.ckjqk0osi3pp.us-east-1.rds.amazonaws.com',
DB_USER : 'admin',
DB_PWD : 'Admin123Pwd',
DB_DATABASE : 'webappdb'
});
```



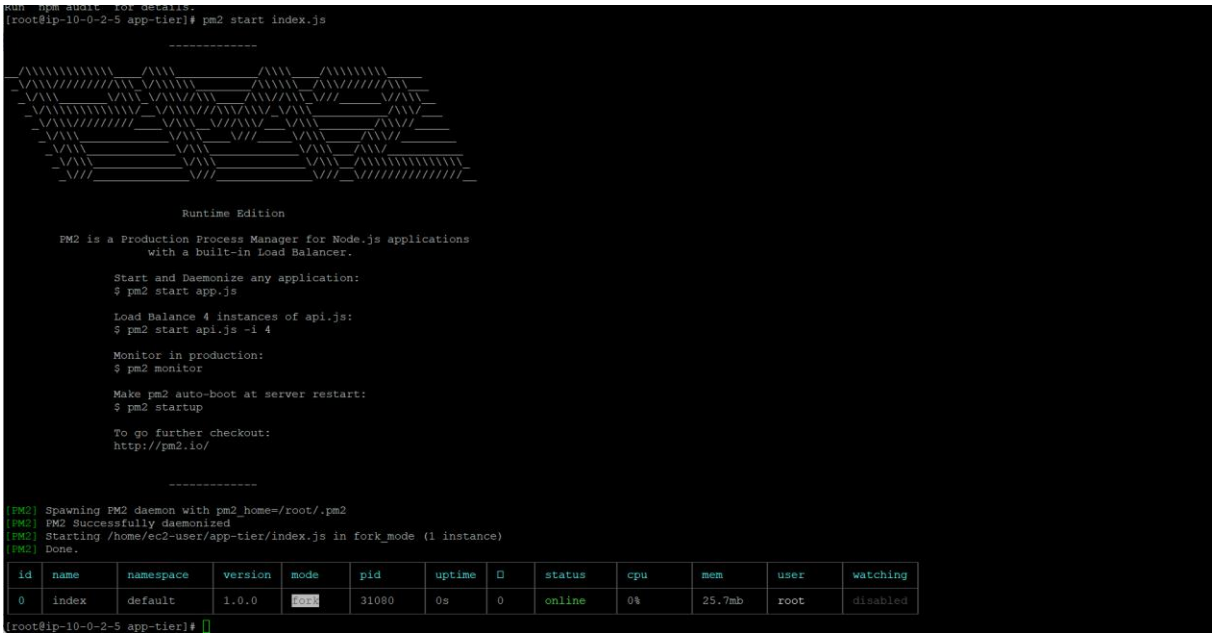
7e: Installing NVM (node version manager).
Run the command:
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
source ~/.bashrc

7f: Next, install a compatible version of Node.js and make sure it's being used

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.

Run the below commands to install Pm2 process manager

```
nvm install 16
nvm use 16
npm install -g pm2
npm install
pm2 start index.js
```



➡ To make sure the app is running correctly run the following command:
pm2 list

➡ you will find status as online; the app is running. If you find any error occurred, then you need to do some troubleshooting. To look at the latest errors, use this command:

pm2 logs
pm2 startup

After running this pm2 starup command, you will get the following script to run the command to save the process done till now, if you don't run the script given below, then node process will not be save.

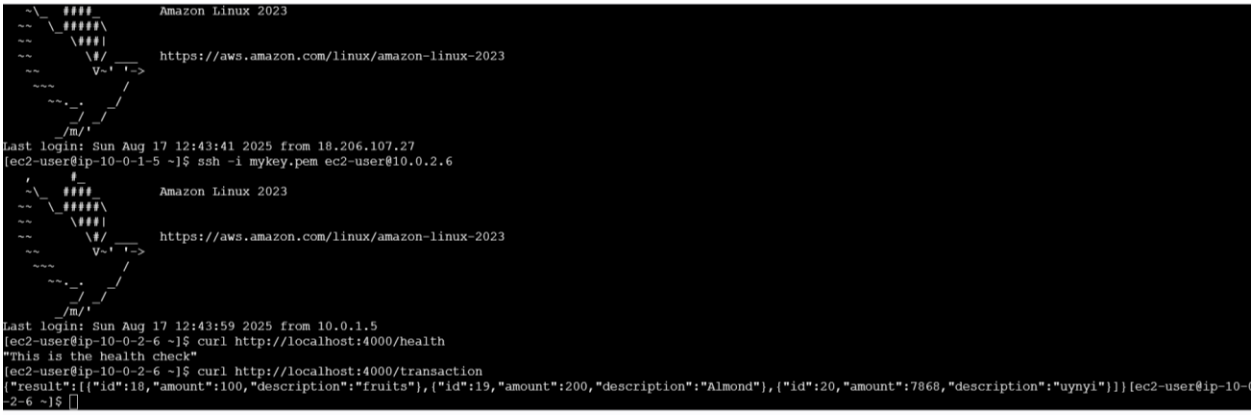
The command to save the process is starts with ...
Sudo env PATH=$PATH:/home/ec2-user/.nvm/version/node/v16.20.....

Hint: after running pm2 startup, then you will get the below instructions,
To setup the Startup Script, copy/paste the following command:
Sudo env PATH=$PATH:/home/ec2-user/.nvm/version/node/v16.20.....

pm2 save

Testing the App-Tier configurations:

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, curl http://localhost:4000/health
To test App to connected with database, curl http://localhost:4000/transaction



Before Creating AMI, make sure everything is working fine even after the server is restarted.

Now, everything is fine, then take AMI for Auto Scaling process for Internal Load Balancer

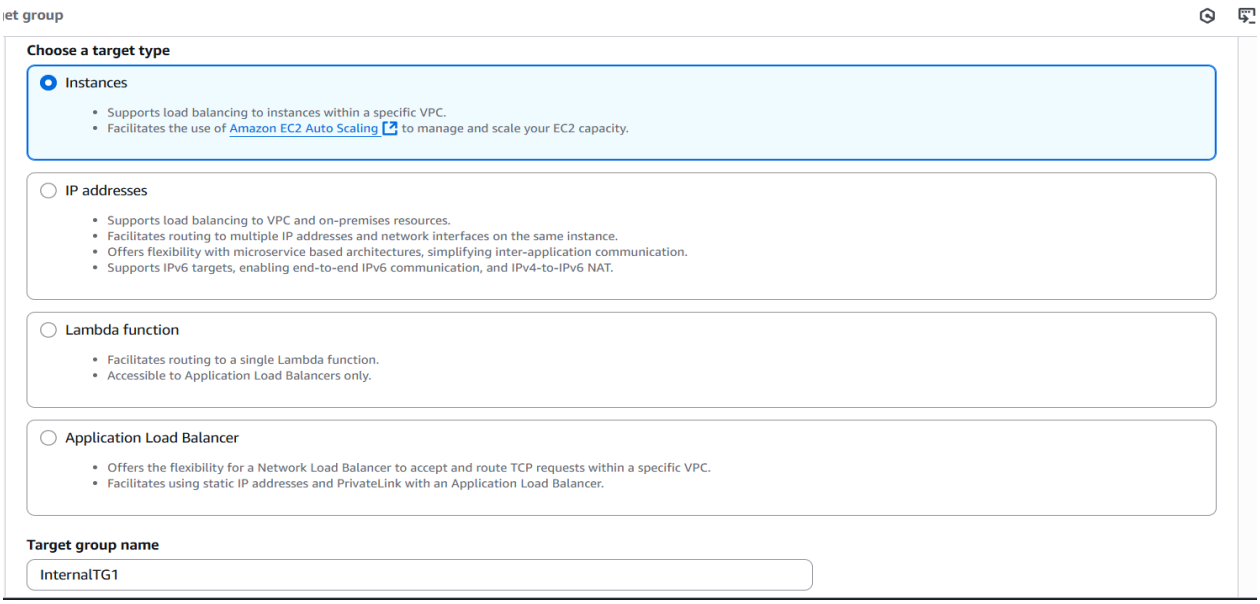
Step8: Creating Internal Application Load Balancer for App-tier configuration

- Elastic Load Balancer (ELB) service, is a mechanism that automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, within one or more Availability Zones.
- Application Load Balancer works based on Path based routing policy, it operates in L7 of OSI Modal. It handles http, https traffics.

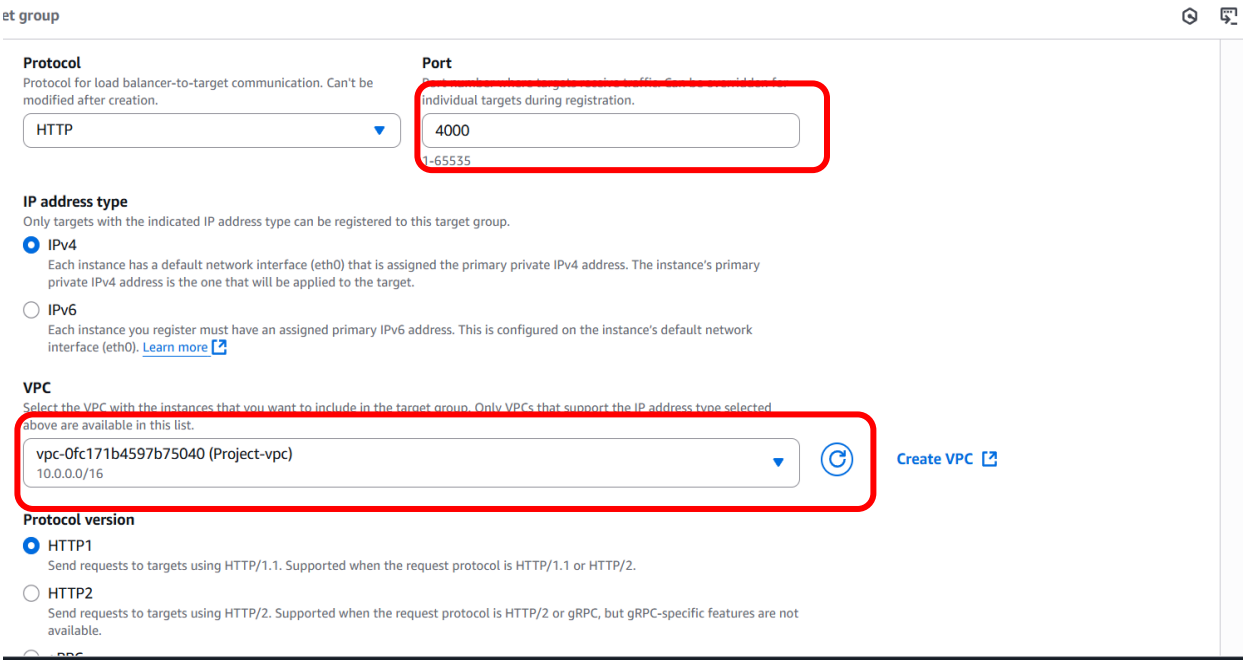
Create Internal Load Balancer without creating instance inside the target group.

Procedure to Create Internal ALB:

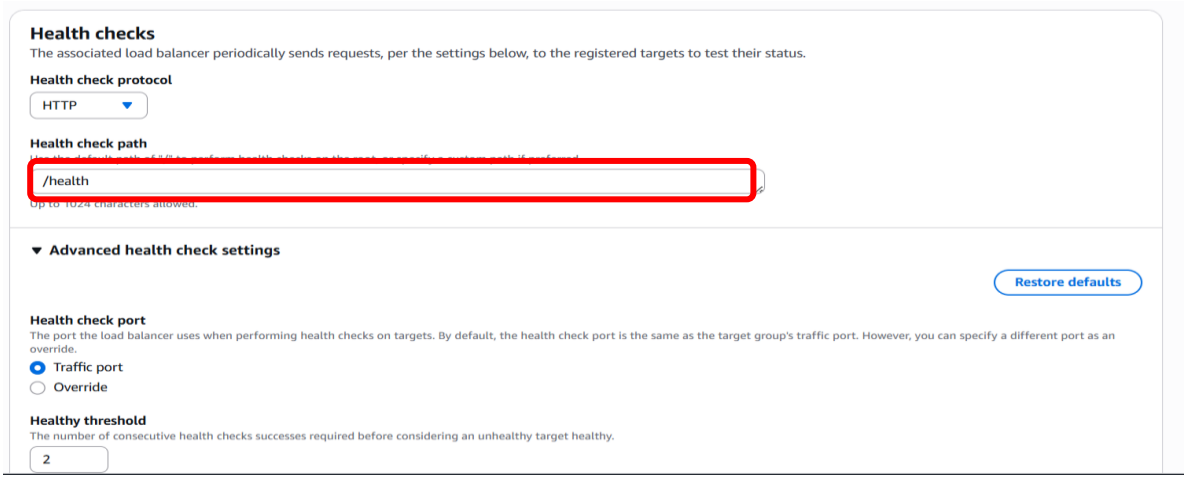
→ Create the Target group



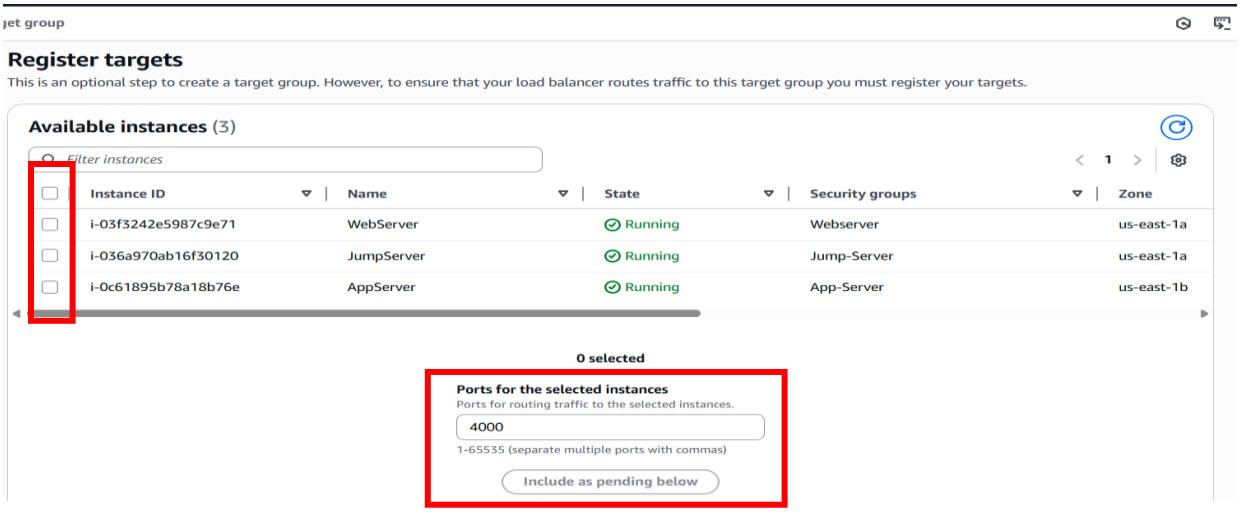
→ Select port number 4000 which is configured in backend Application and select the VPC we created.



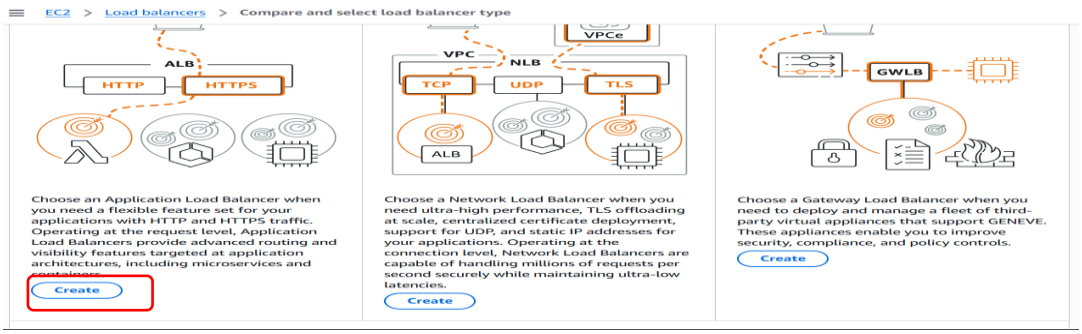
→ Health Check Path is **'health'**, which is configured in backend process.

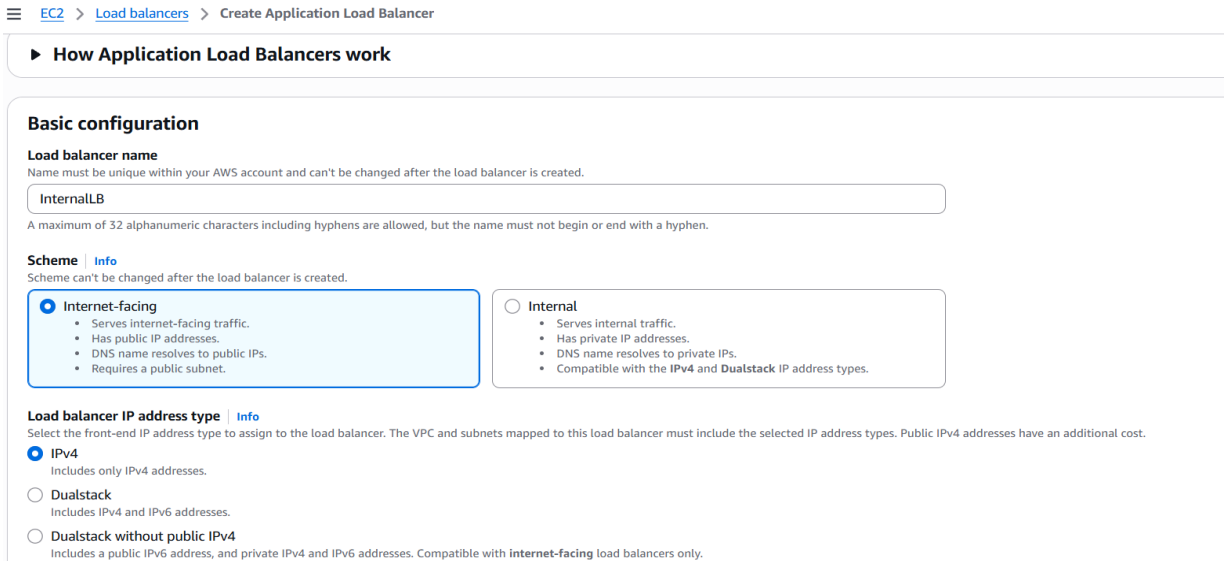


→ Creating LB without registering the EC2 instance inside the target group.

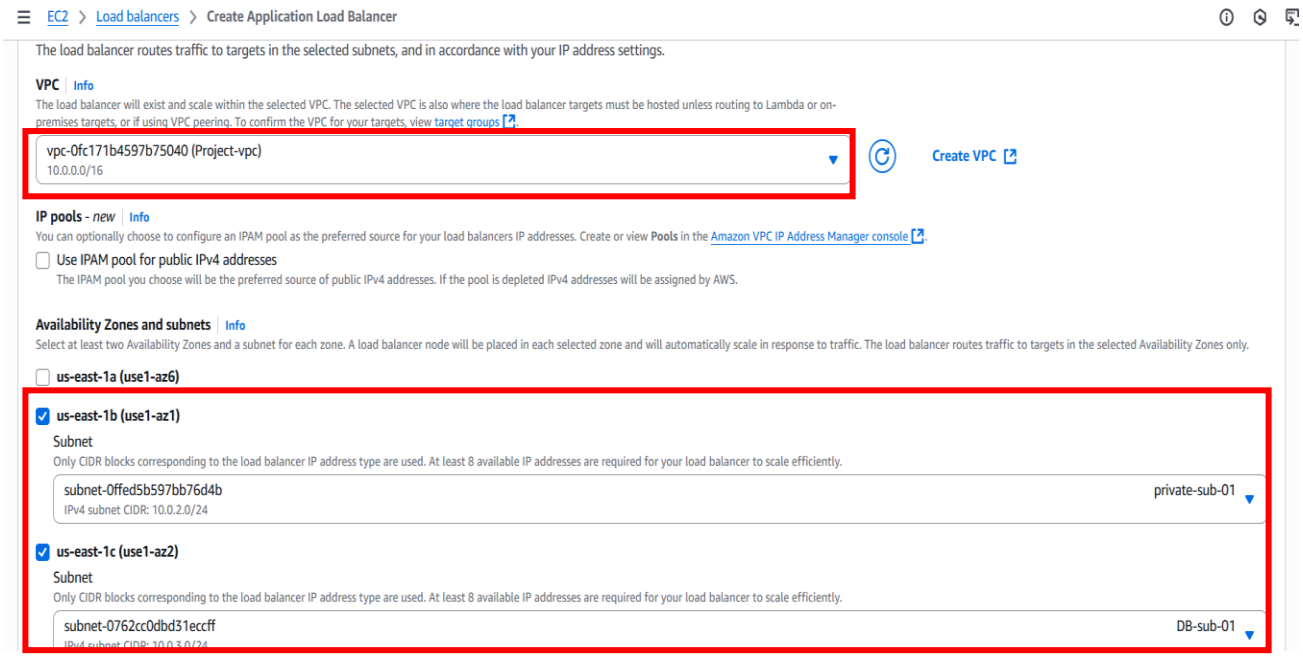


→ Create Application Load Balancer and add the Target group inside the ALB.



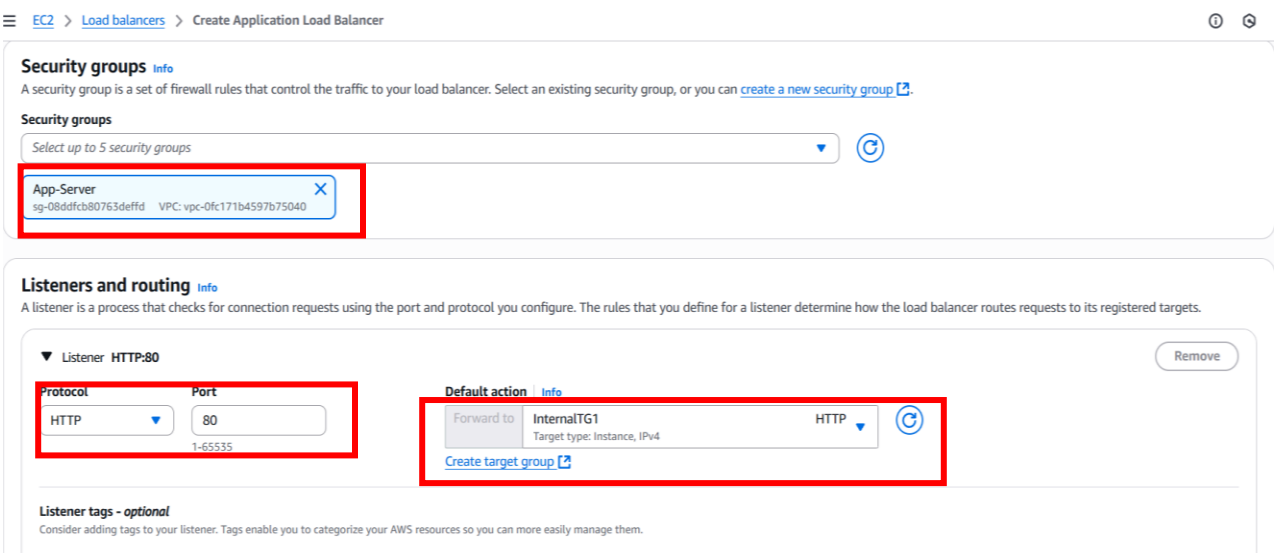


→ Select VPC and Private subnets for multiple Available Zones.



→ Select the App Server Security group

→ Select the Internal Target group inside the Internal Load Balancer.



Load Balancer is created successfully.

Step9: Creating Auto Scaling for Internal Application Load Balancer

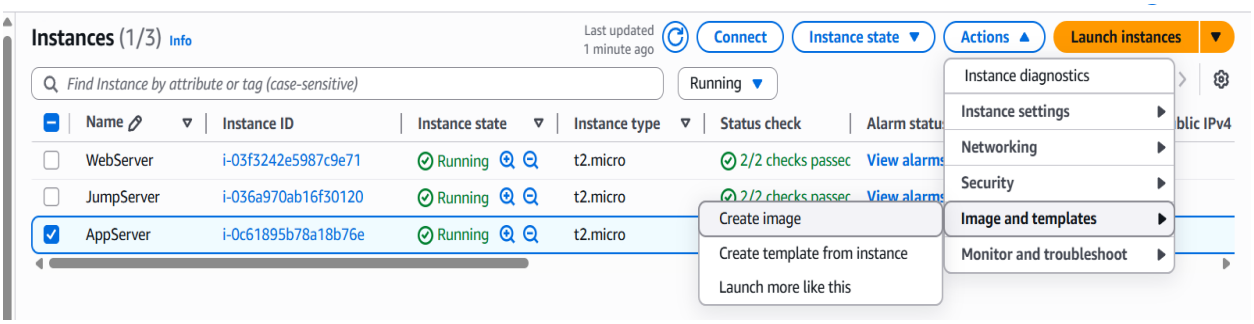
AWS Auto Scaling allows you to automatically adjust the number of EC2 instances capacity based on the real-time demand.

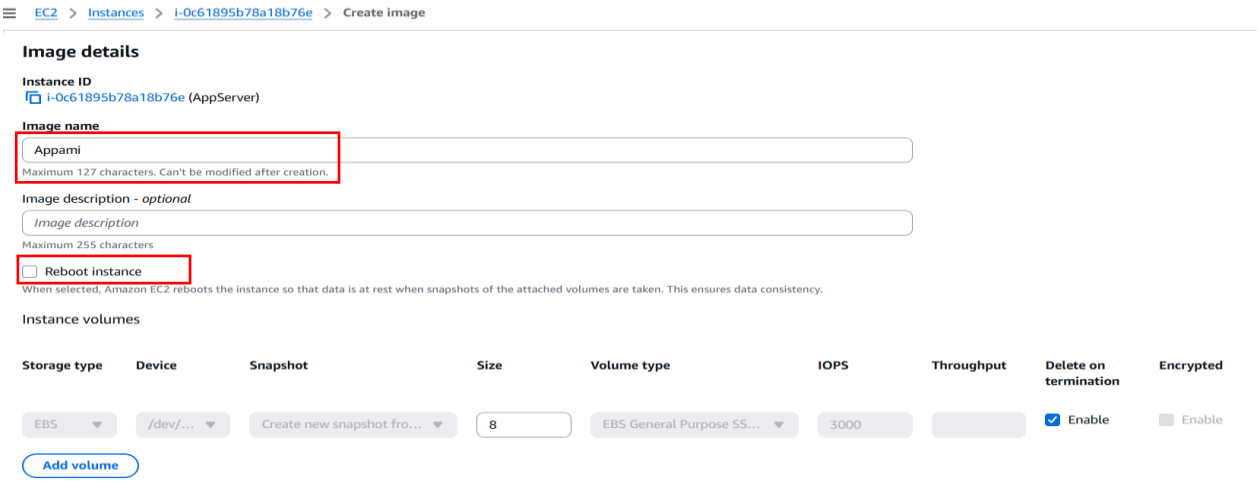
- Generate Auto Scaling for Internal ALB and Select the (Private) App-tier configuration Ec2 instance as the Template.

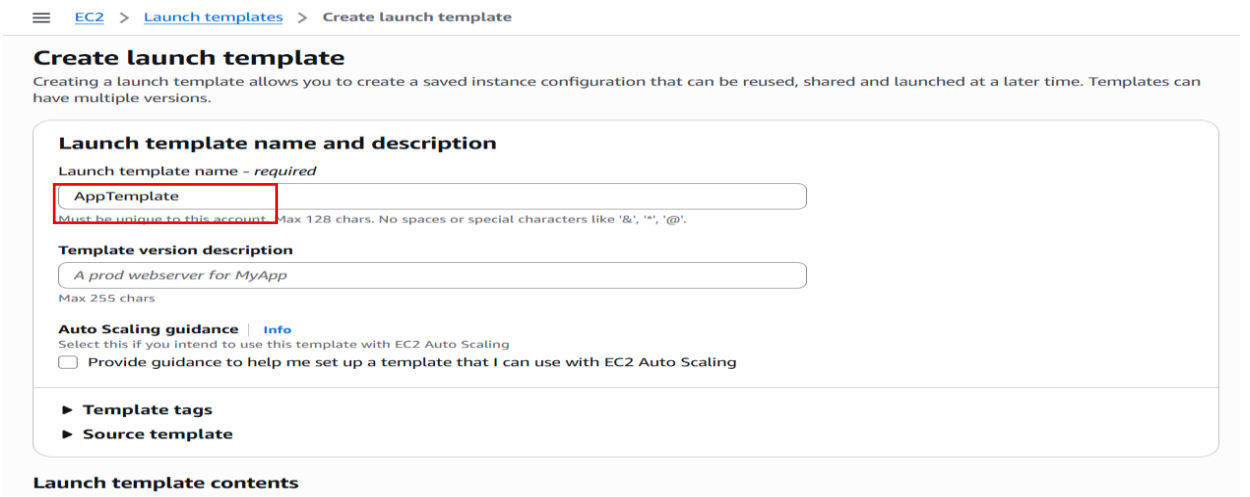
Procedure to Create Auto Scaling:

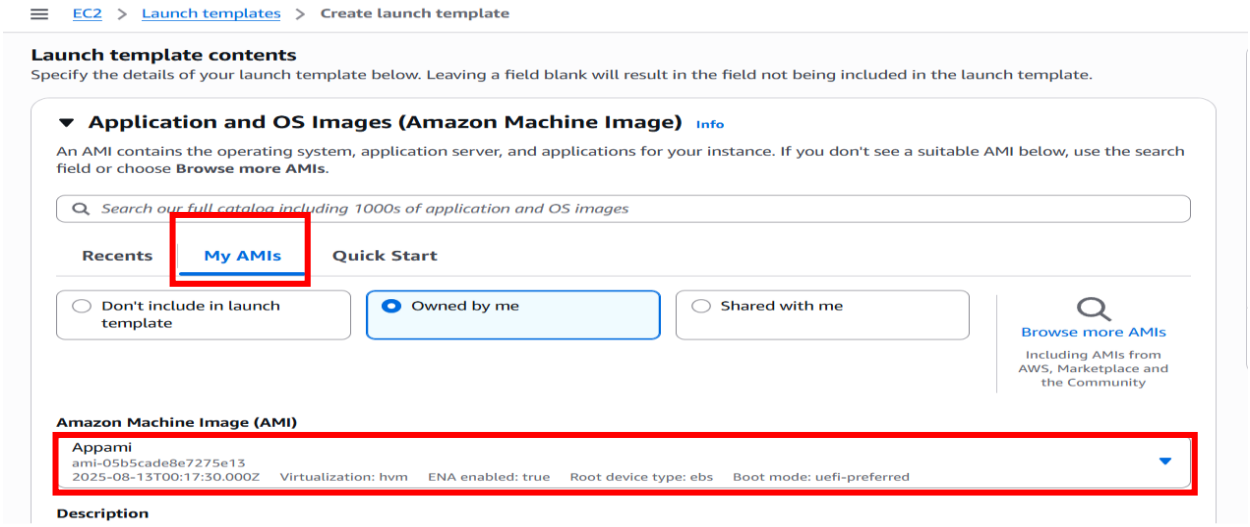
Create AMI Template.

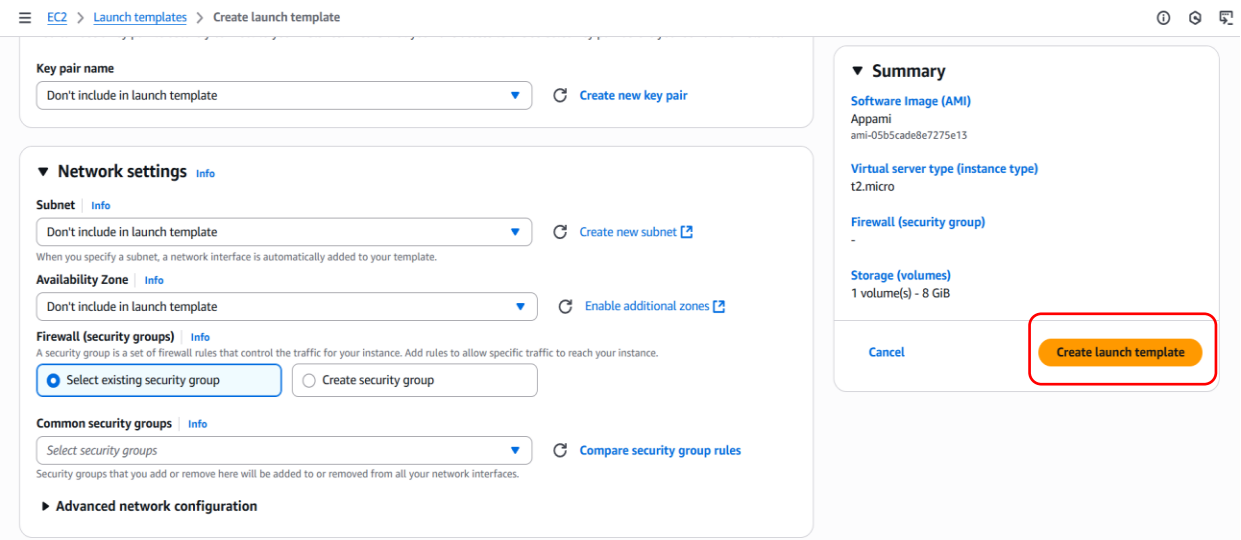
Amazon Machine Image (AMI) is a pre-configured template for launching EC2 instance.






EC2 > Launch templates > Create launch template
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
AppTemplate
Must be unique to this account. Max 128 chars. No spaces or special characters like '&', '*', '@'.
Template version description
A prod webserver for MyApp
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
Launch template contents

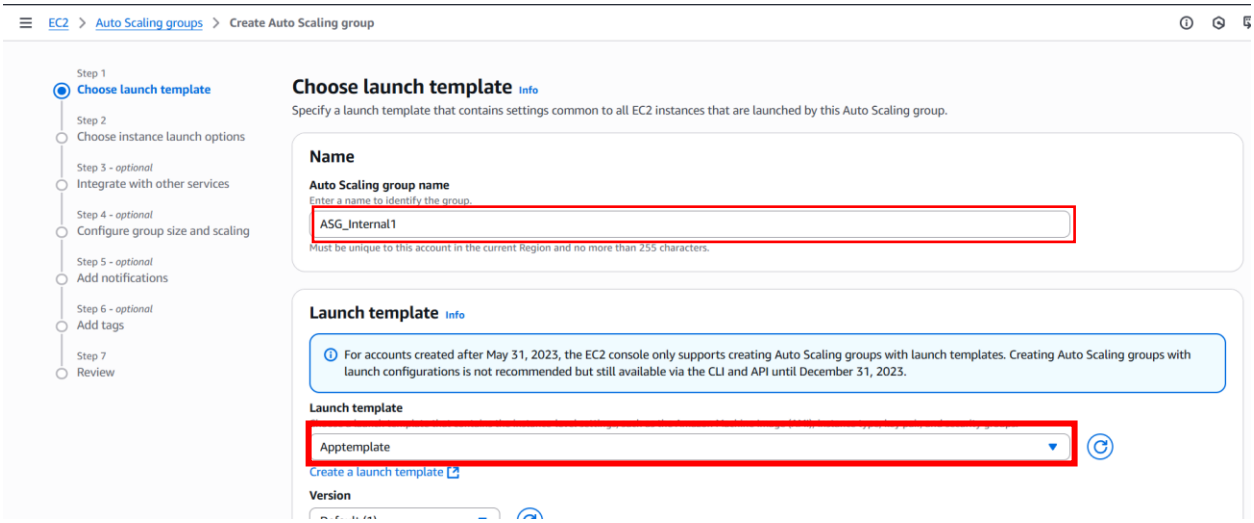

EC2 > Launch templates > Create launch template
Launch template contents
Specify the details of your launch template below. Leaving a field blank will result in the field not being included in the launch template.
Application and OS Images (Amazon Machine Image) Info
An AMI contains the operating system, 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)
Appami
ami-05b5cade8e7275e13
2025-08-13T00:17:30.000Z Virtualization: hvm ENA enabled: true Root device type: ebs Boot mode: uefi-preferred
Description



Add AMI Template inside the Auto Scaling group.

→ Choose the App Server AMI Template inside the Auto Scaling group.

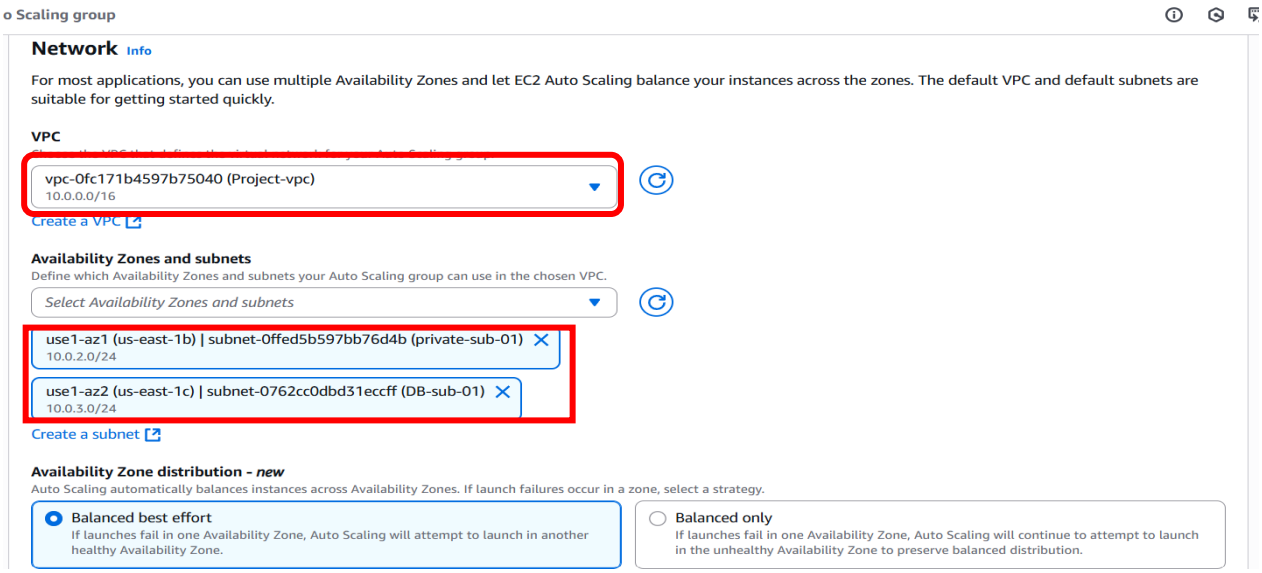
→ EC2 instance will be automatically assigned inside the target group.

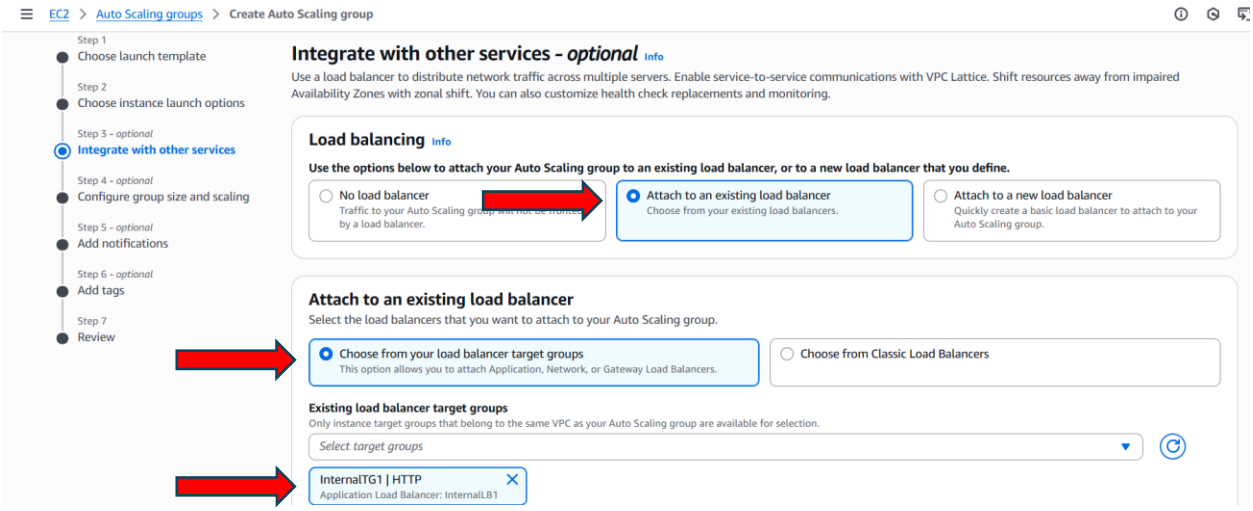


→ Select VPC and Private subnets for multiple Available Zones.

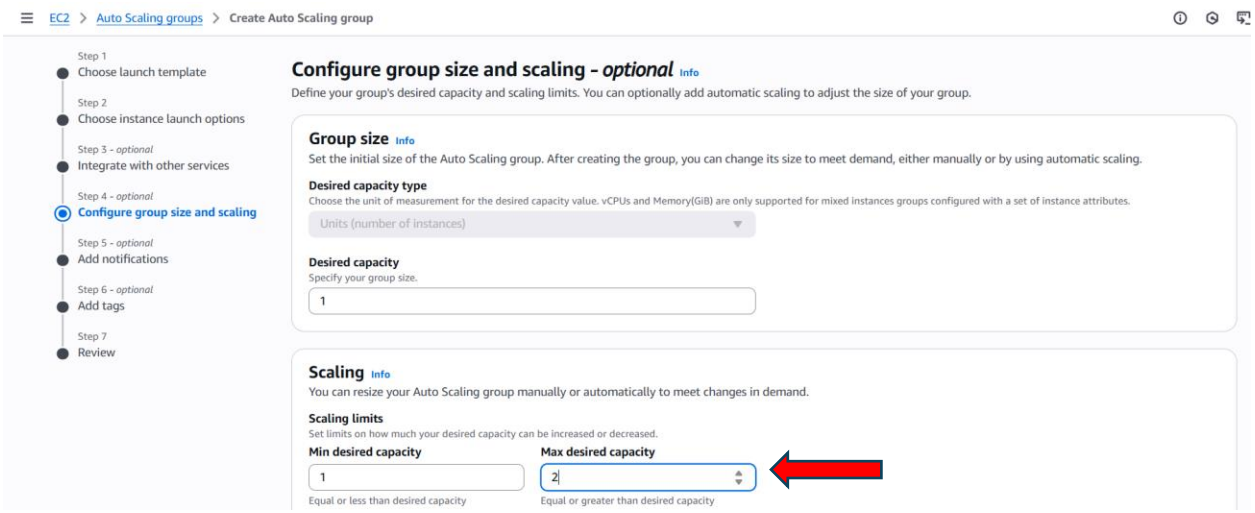
→ Attach to an Existing Load Balancer.

→ Select the Internal Load Balancer.

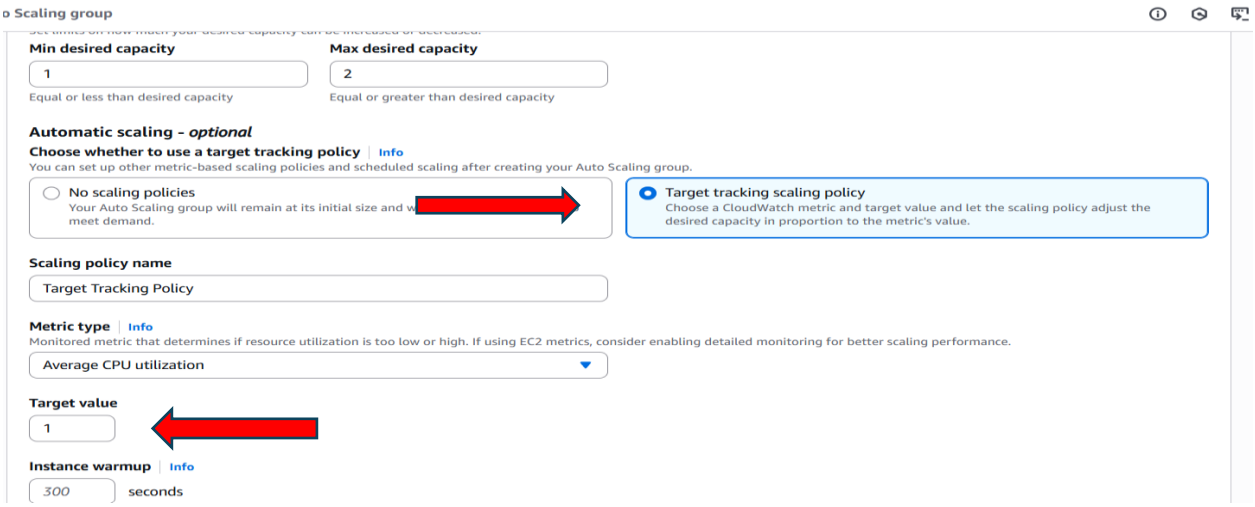


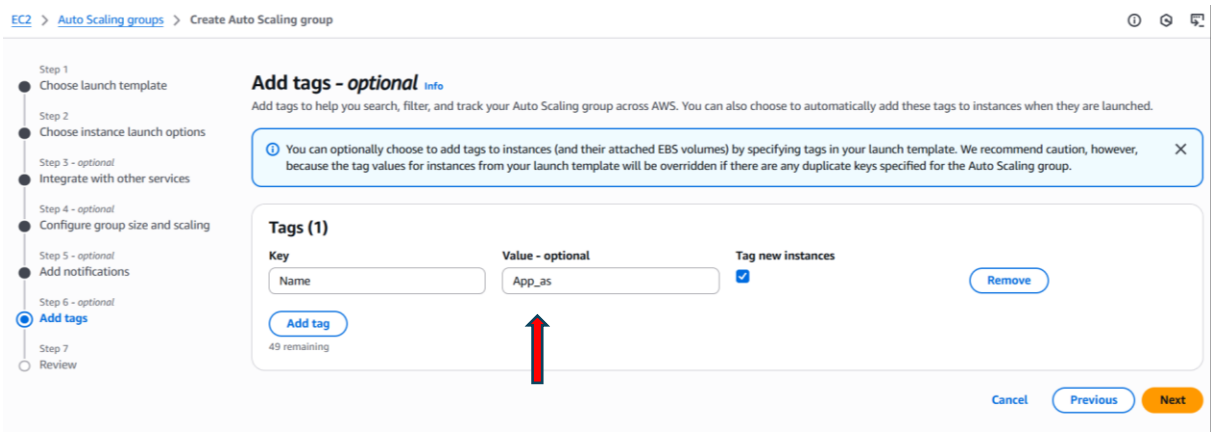


Fixing the desired capacity based on the real-time demand.



Here I'm fixing the Target Value as 1 for testing purpose.





Auto Scaling for External Load Balancer is created successfully.

Step10: Web-Tier Configuration:

10a. Goto web server

➡ Open jump server
➡ Goto app server through SSH credentials

ssh –i <keypair> <username>@ip i.e., ssh –i key.pem ec2-user@10.0.1.15

10b. Copy the S3 bucket web-tier folder into web server

In web server, make directory for web-tier folder content,
sudo mkdir web-tier
cd web-tier

Command to copy s3 bucket folder into web server:
aws s3 cp s3://s3projdemoapp/web-tier/ . --recursive
aws s3 ls ➡ to list all content inside s3 bucket

10c. Installing NVM (Node Version Manager)

> NVM is a command-line tool that facilitates the installation, management, and switching between multiple versions of Node.js on a single system.

Run the following command:
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash source ~/.bashrc

Next, install a compatible version of Node.js and make sure it's being used
nvm install 16
nvm use 16

Install NPM (Node Package Manager)

> NPM is the key tool in JavaScript development, allowing developers to install, share, and manage dependencies in their projects.

npm install ➡ it will create the node_module folder

npm run build → it will create the build folder, which contains index.html file

10d. Connecting Frontend with backend application by Nginx server

Nginx is the powerful web server that can do reverse proxy, load balancer, mail proxy, and http cache.

Add Internal Load balancer URL in nginx.config file by installing nginx

sudo yum install nginx –y

- You will have nginx.conf file in application-code folder, which will be provided by the developer.
- Copy the internal Load Balancer url and paste inside the nginx file. (in proxy for internal load balancer).
- And copy that nginx.config file content, replace it with the /etc/nginx/nginx.conf file.

cd /etc/nginx
ls → you will find nginx.conf
sudo vi nginx.conf

```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Mon Aug 18 11:25:02 2025 from 18.206.107.29
[ec2-user@ip-10-0-1-5 ~]$ ssh -i mykey.pem ec2-user@10.0.1.15
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Mon Aug 18 11:25:29 2025 from 10.0.1.5
[ec2-user@ip-10-0-1-15 ~]$ cd web-tier
[ec2-user@ip-10-0-1-15 web-tier]$ ls
README.md  build  node_modules  package-lock.json  package.json  public  src
[ec2-user@ip-10-0-1-15 web-tier]$
```

```
[ec2-user@ip-10-0-1-15 ~]$ cd /etc/nginx
[ec2-user@ip-10-0-1-15 nginx]$ ls
conf.d      fastcgi.conf          fastcgi_params          koi-utf  mime.types          nginx.conf          scgi_params          uwsgi_params          win-utf
default.d  fastcgi.conf.default  fastcgi_params.default  koi-win  mime.types.default  nginx.conf.default  scgi_params.default  uwsgi_params.default
[ec2-user@ip-10-0-1-15 nginx]$ sudo vi nginx.conf
[ec2-user@ip-10-0-1-15 nginx]$
```

```
    server {
        listen       80;
        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://InternalLB-160122618.us-east-1.elb.amazonaws.com:80/;
        }


    }
-- INSERT --
```

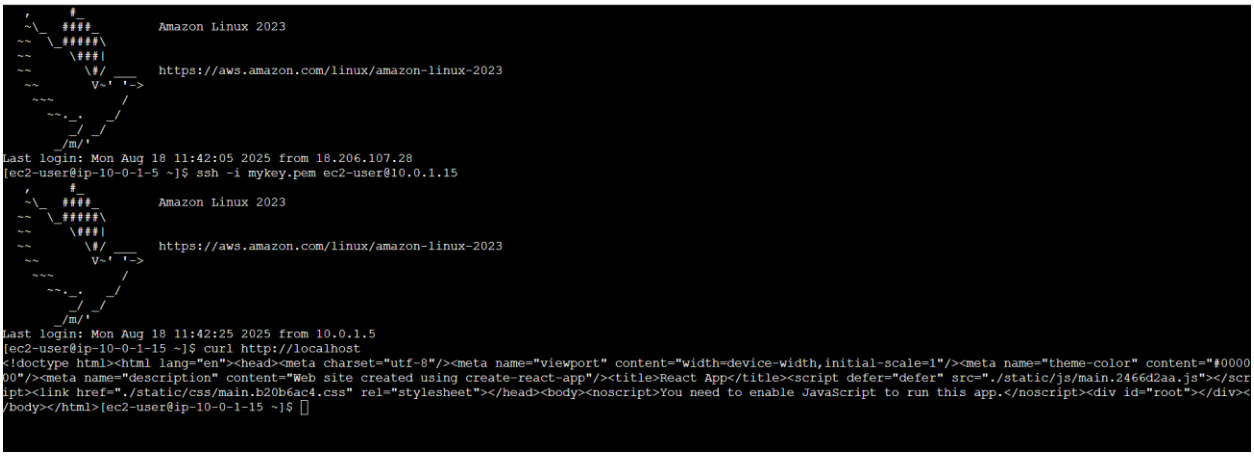
providing permission for home directory

sudo chmod –R 755 /home/ec2-user ➡ providing permission for nginx to access the files

sudo service nginx restart

sudo chkconfig nginx on → this command is used to start the service even after the system is rebooted. Make sure that both frontend and backend database is inserting the records correctly

curl http://localhost → Checking Web server Locally,



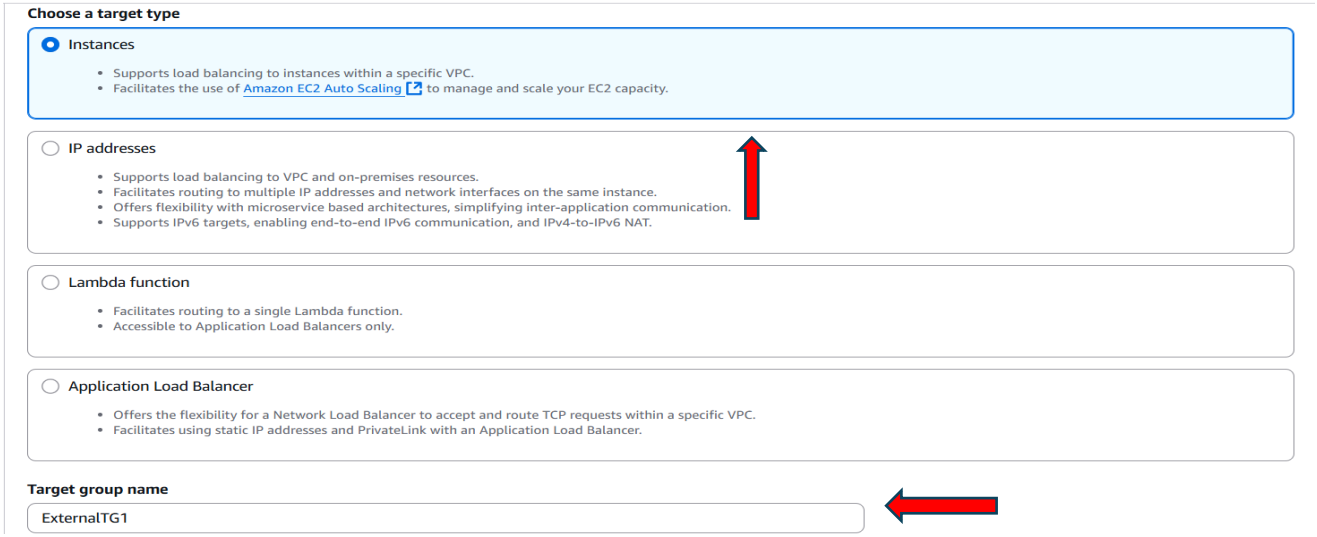
Now, everything is fine, then take AMI for Auto Scaling process for External Load Balancer.

Step 11: Creating External Application Load Balancer for Web-tier configuration

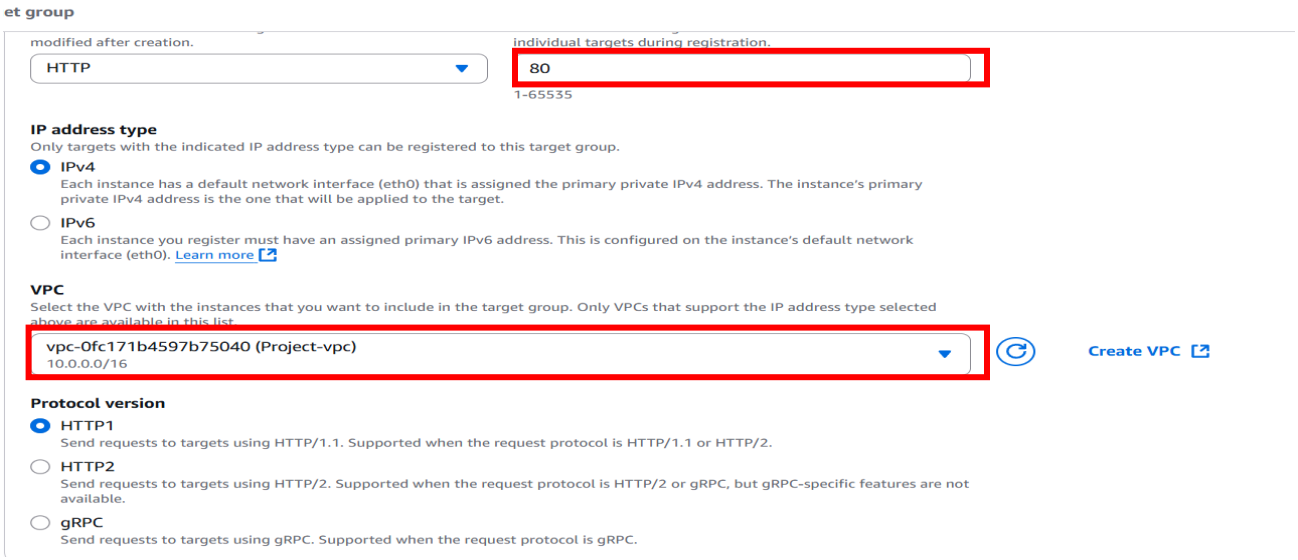
- Create External Load Balancer without creating instance inside the target group.

Procedure to Create External ALB:

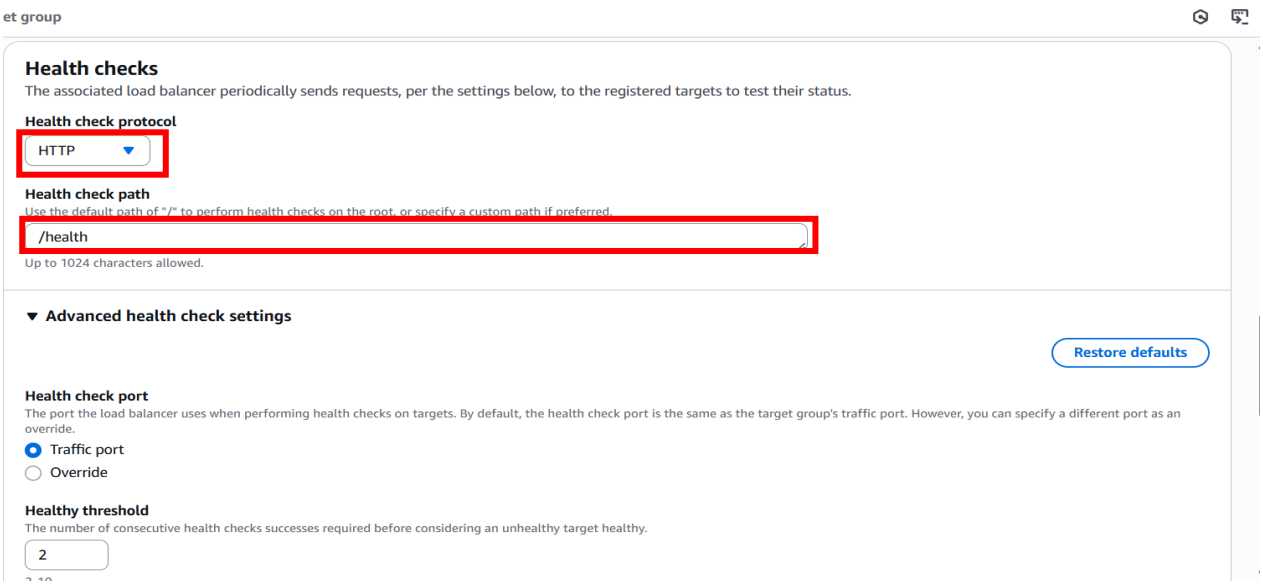
→ Create the Target group



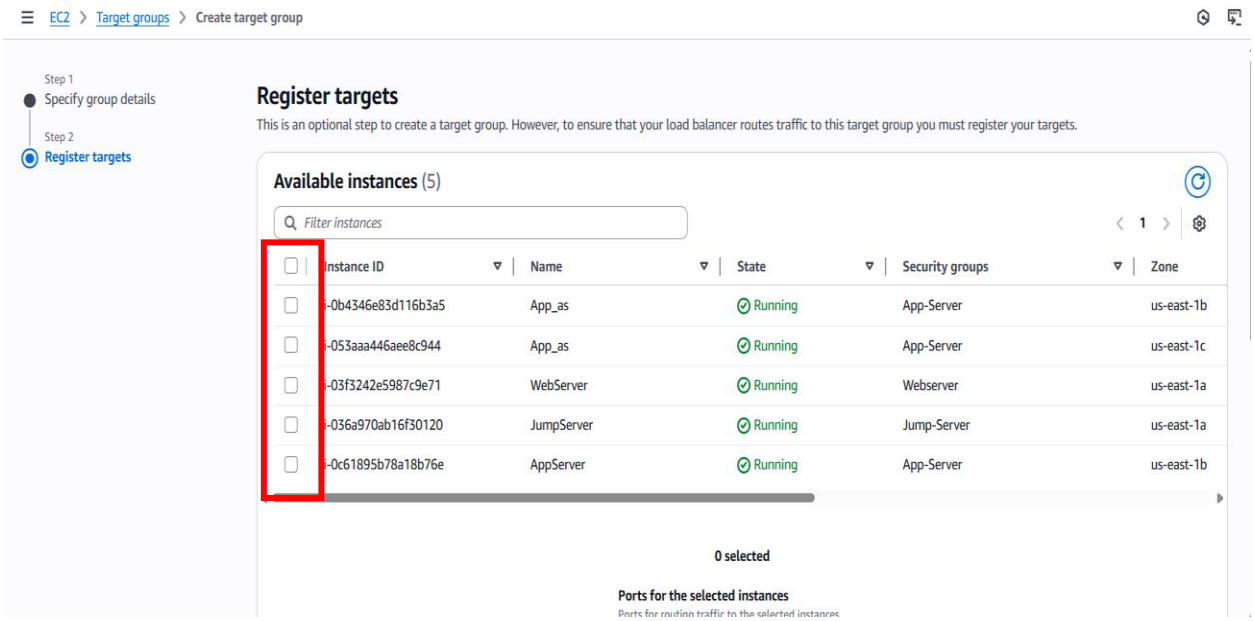
→ Select port number 80 and select the VPC we created.

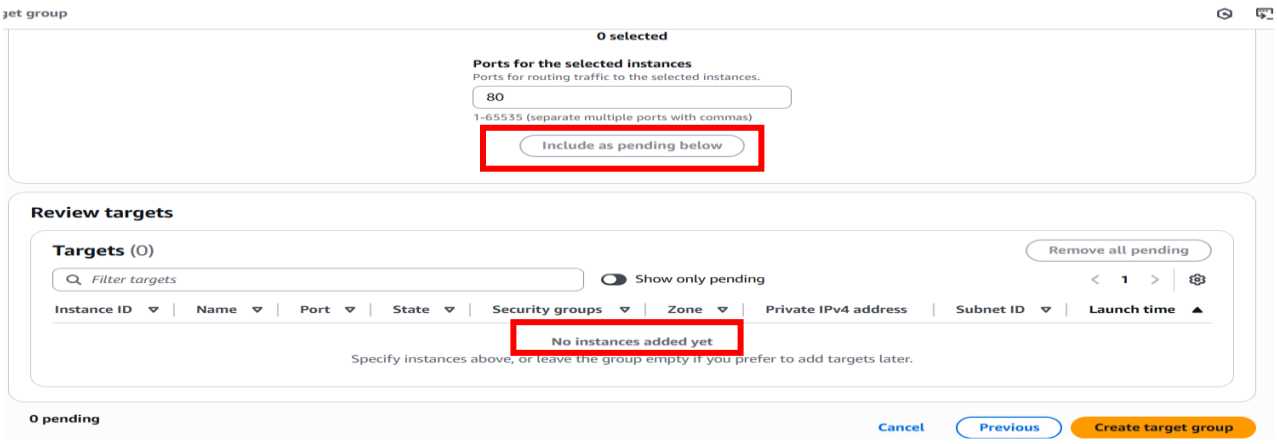


→ Health Check Path is 'health', which is configured in backend process.

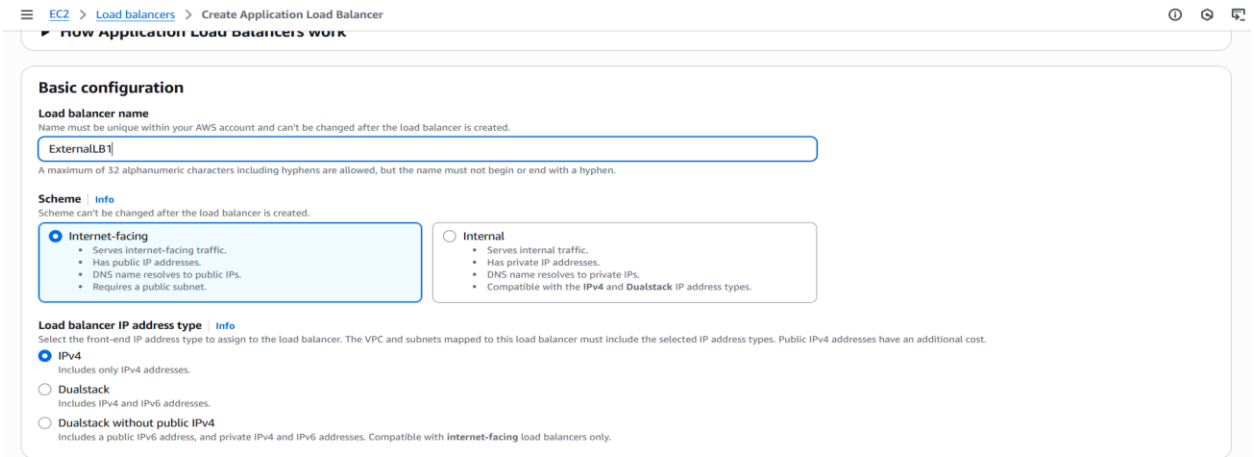


→ Creating LB without registering the EC2 instance inside the target group.

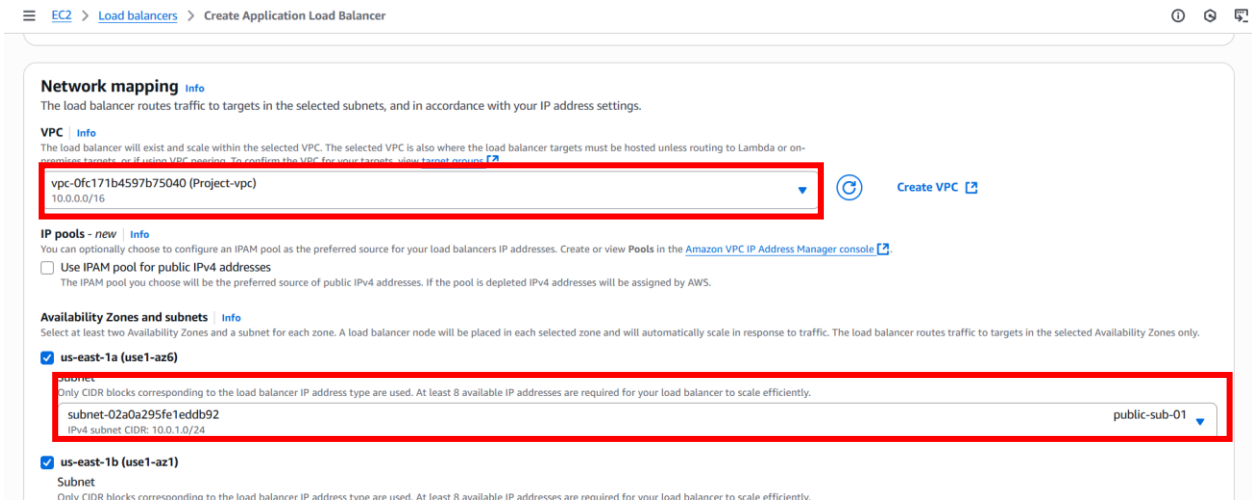




→ Create Application Load Balancer and add the Target group inside the ALB.

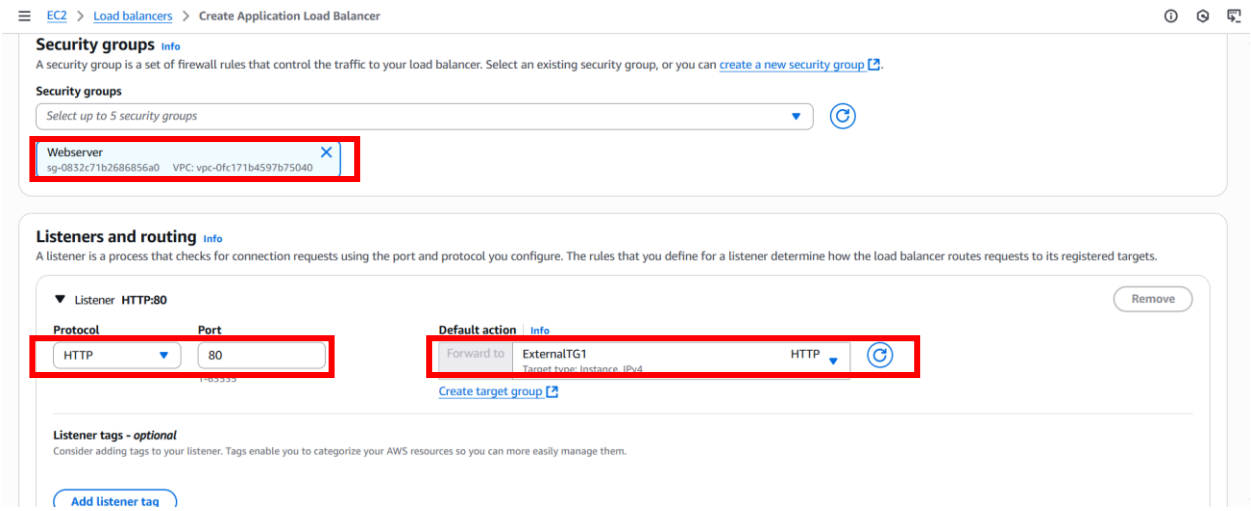


→ Select VPC and Public subnets for multiple Available Zones.



→ Select the Web Server Security group

→ Select the External Target group inside the External Load Balancer.



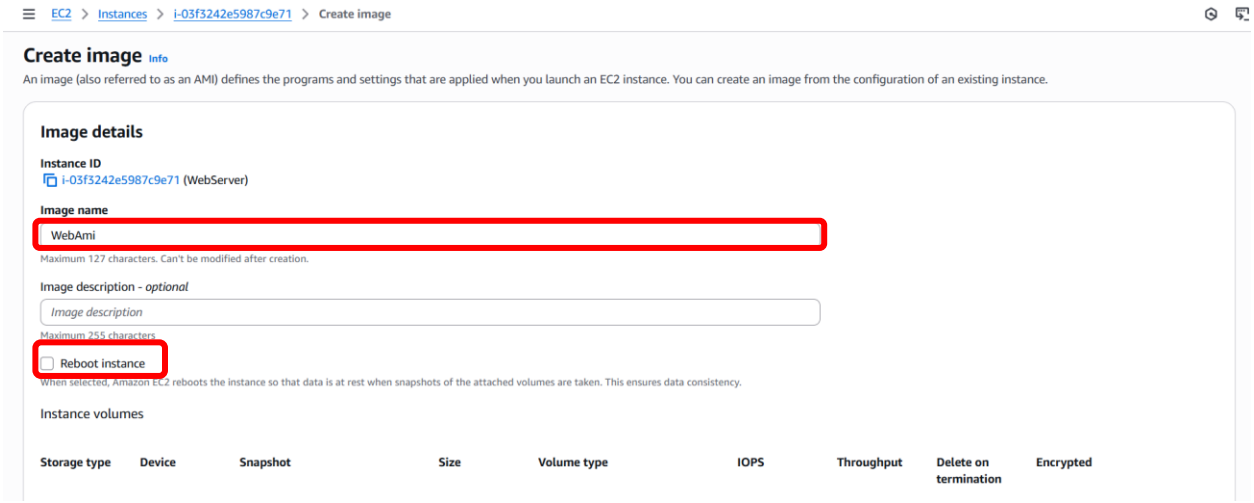
External Load Balancer is created successfully.

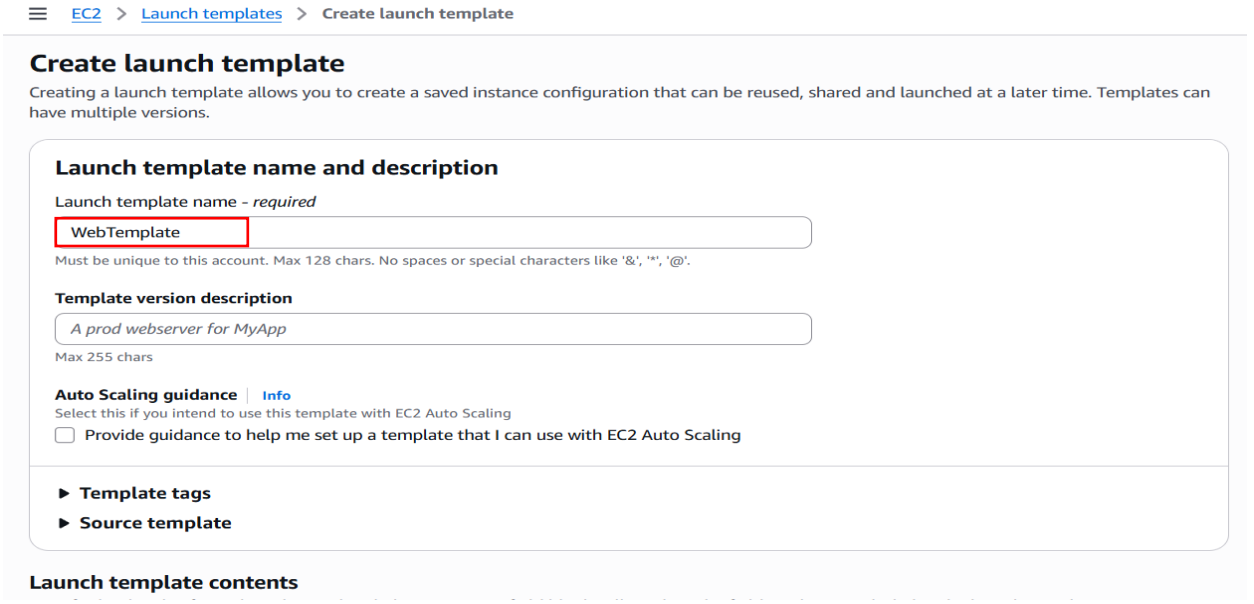
Step12: Creating Auto Scaling for External Application Load Balancer

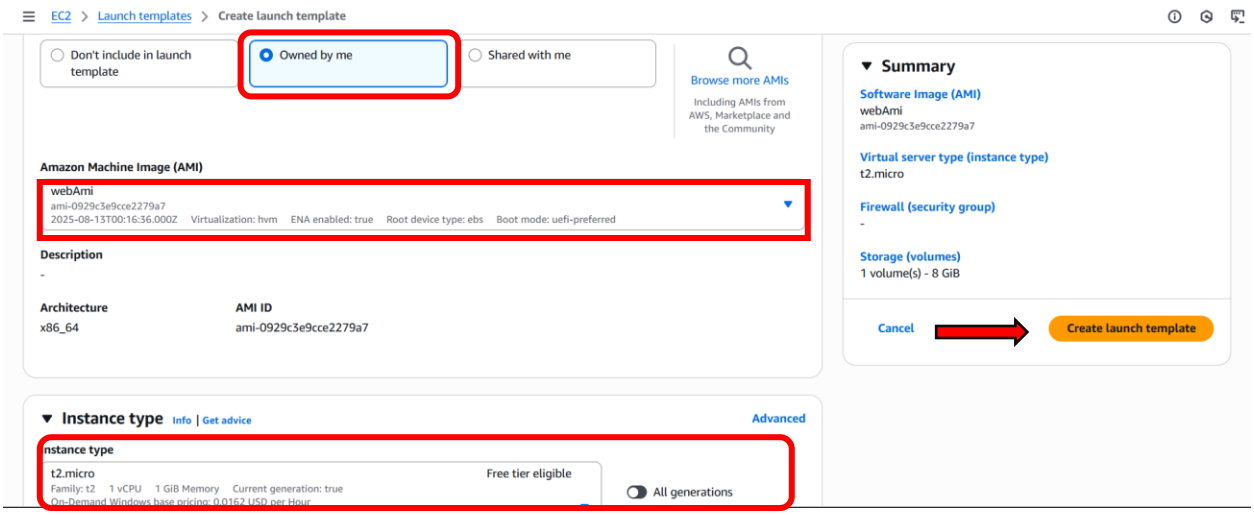
- Generate Auto Scaling for External ALB and Select the (Public) Web-tier configuration Ec2 instance as the Template.

Procedure to Create Auto Scaling:

Create AMI Template



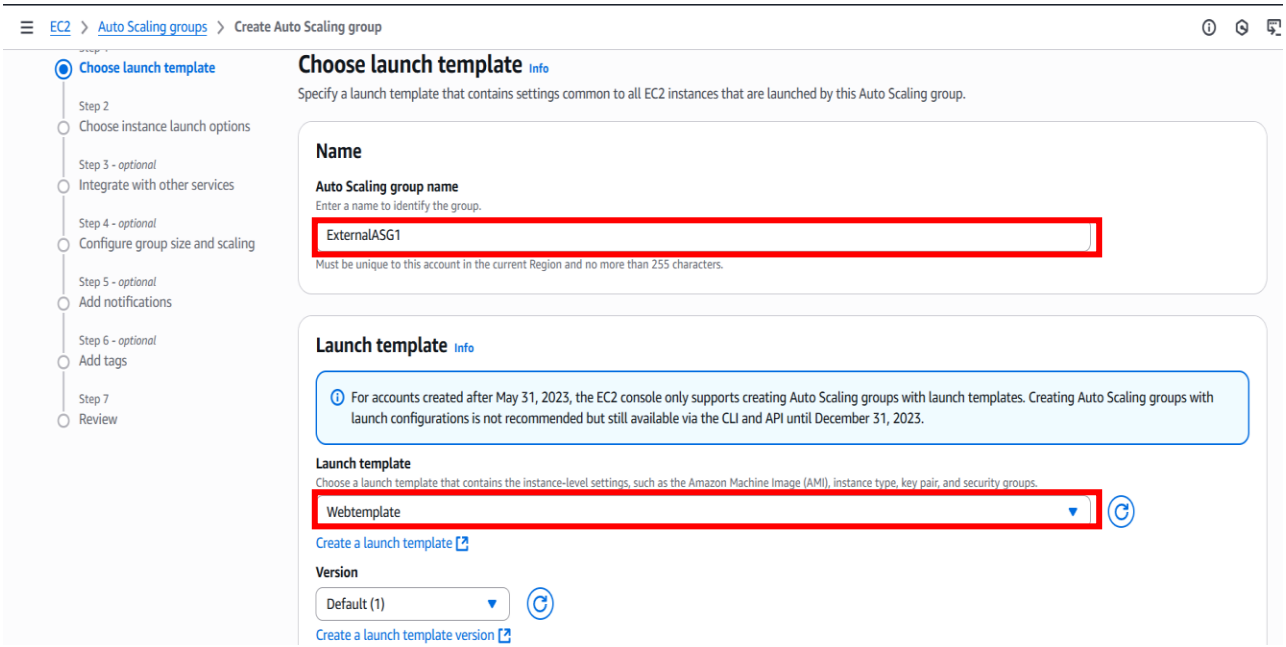




Add AMI Template inside the Auto Scaling group.

→ Choose the Web Server AMI Template inside the Auto Scaling group.

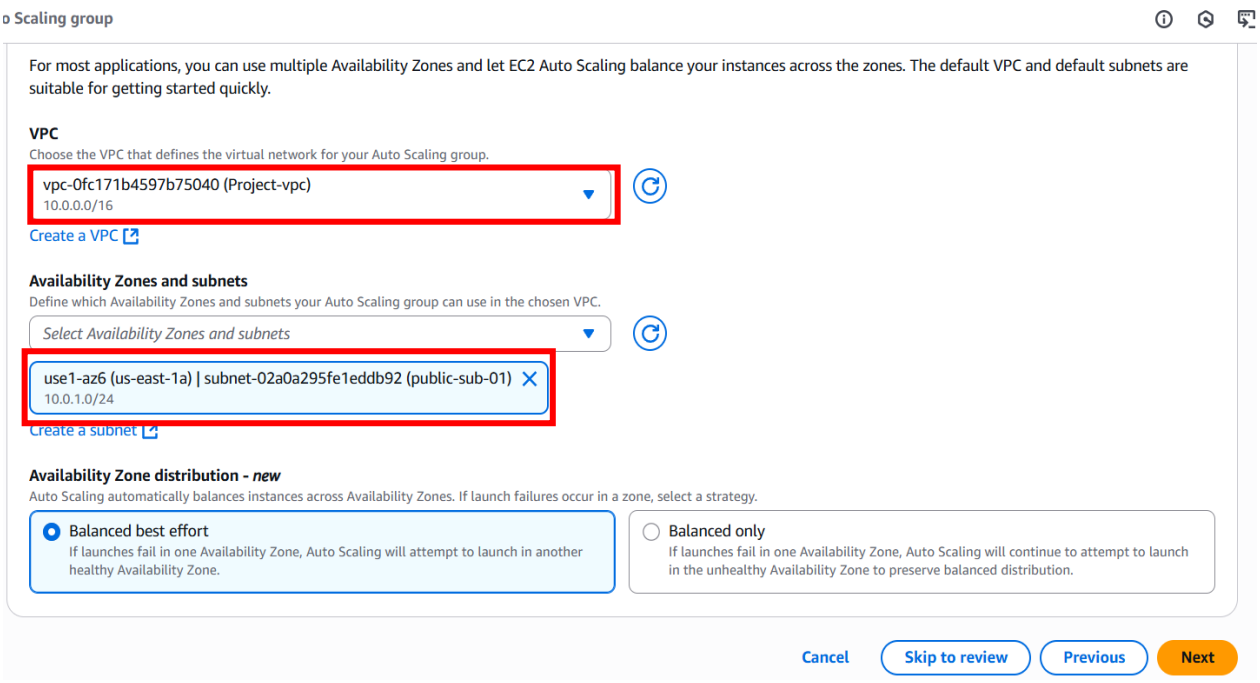
→ EC2 instance will be automatically assigned inside the target group.

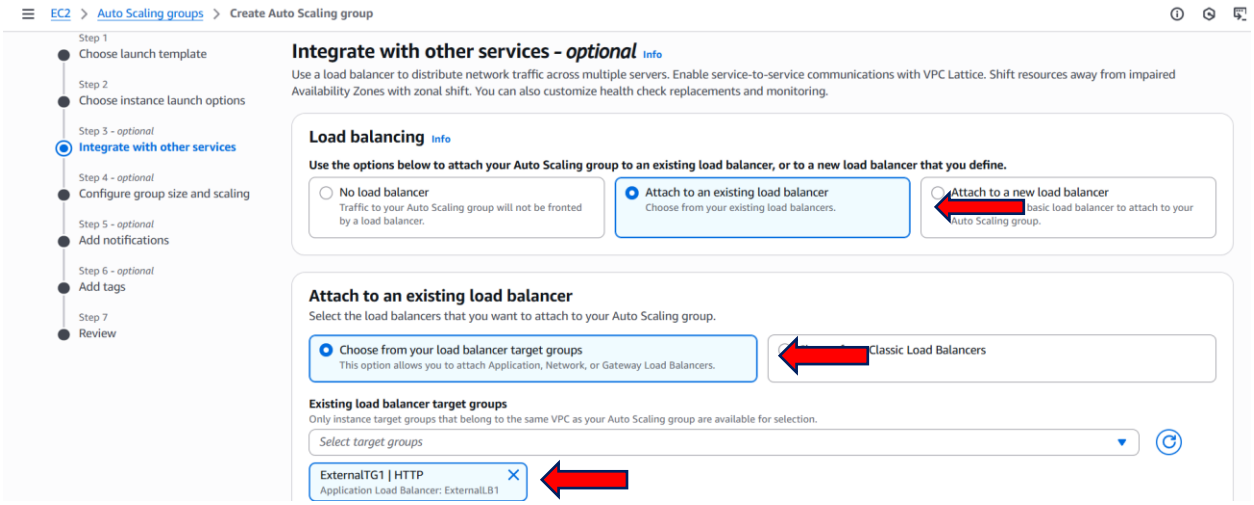


→ Select VPC and Public subnets for multiple Available Zones.

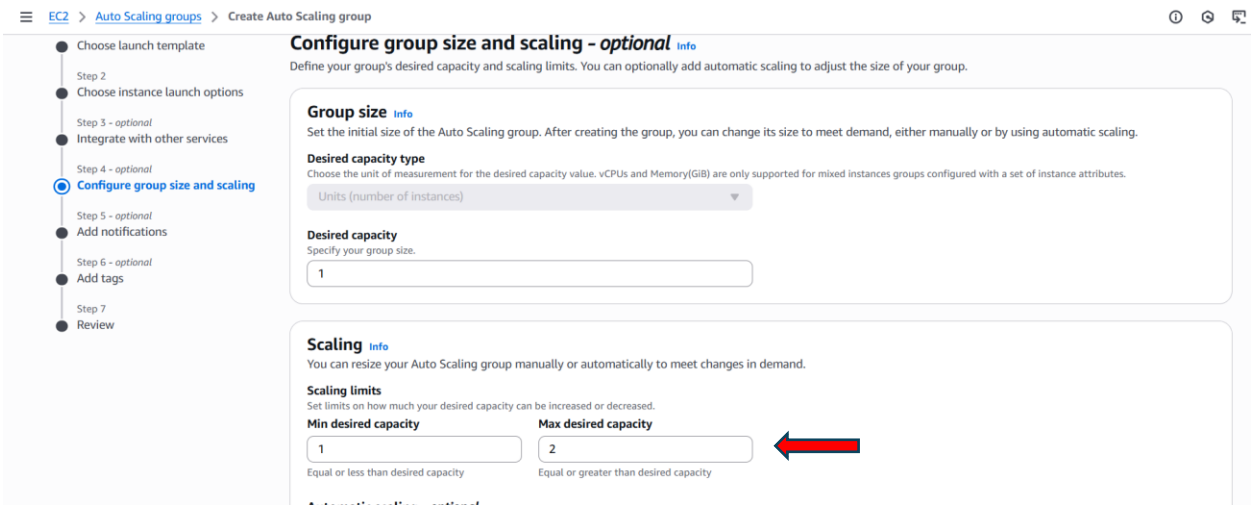
→ Attach to an Existing Load Balancer.

→ Select the External Load Balancer.

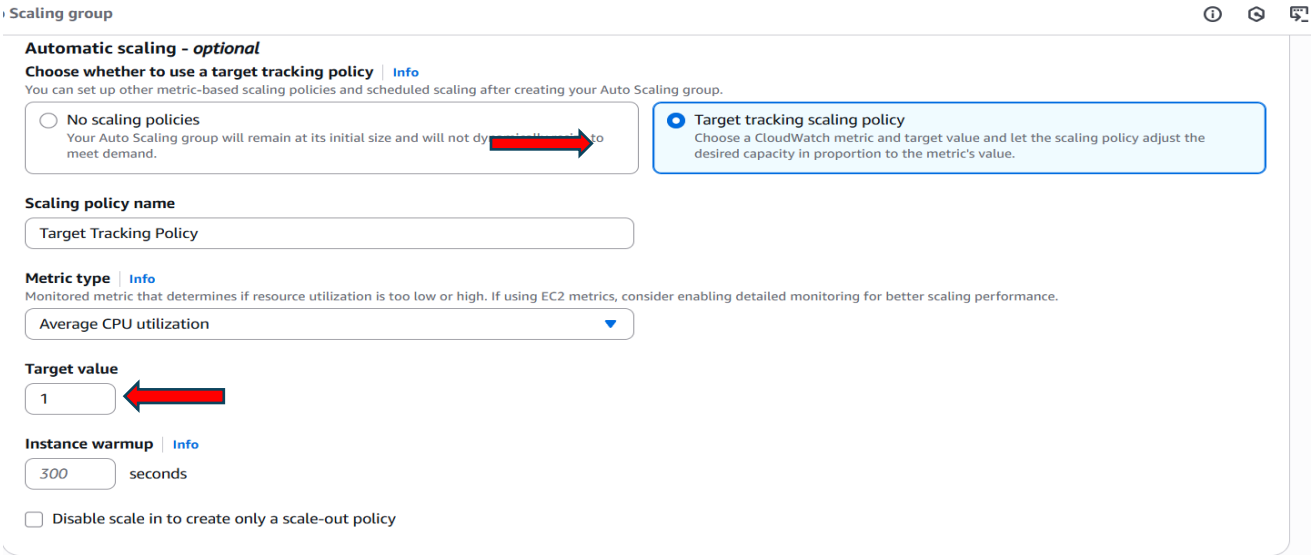


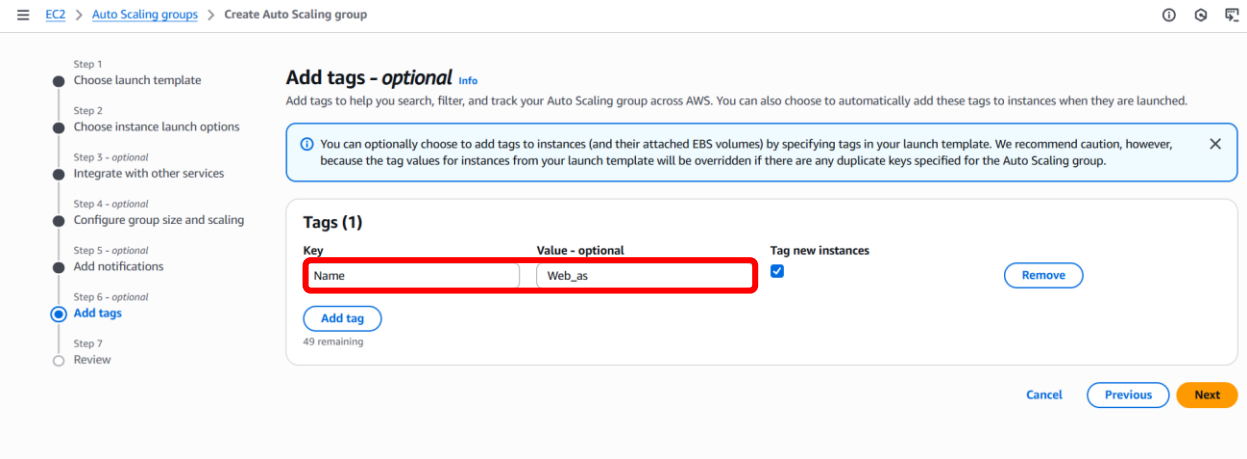


Fixing the desired capacity based on the real-time demand.



Here I'm fixing the Target Value as 1 for testing purpose.

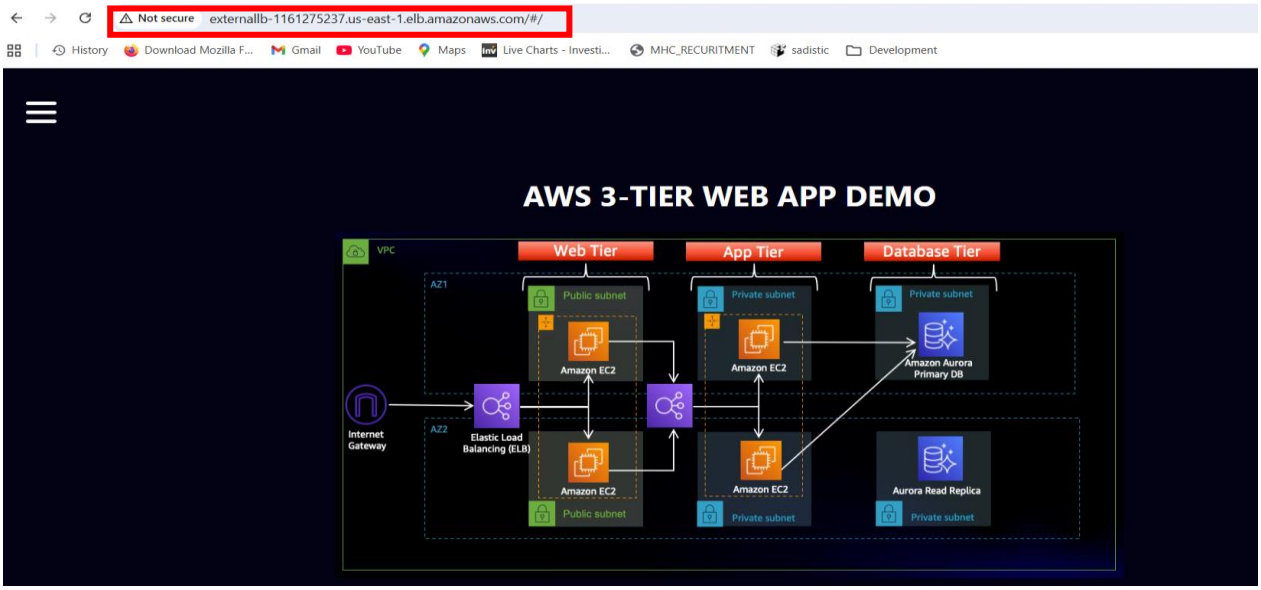




Auto Scaling for External Load Balancer is created successfully.

Step13: Testing Application Locally

Copy the External Load Balancer URL and run in browser



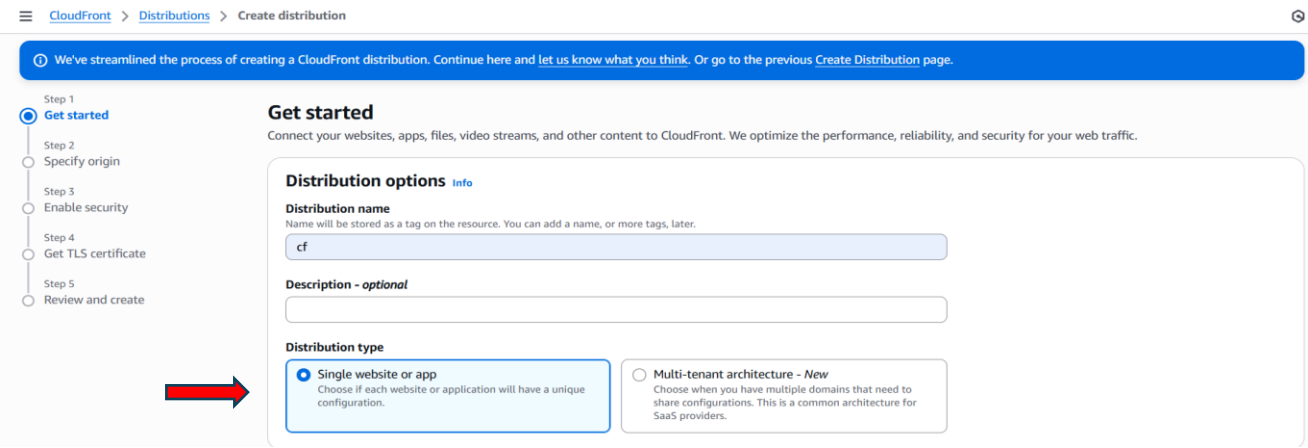
Our Application is working fine.

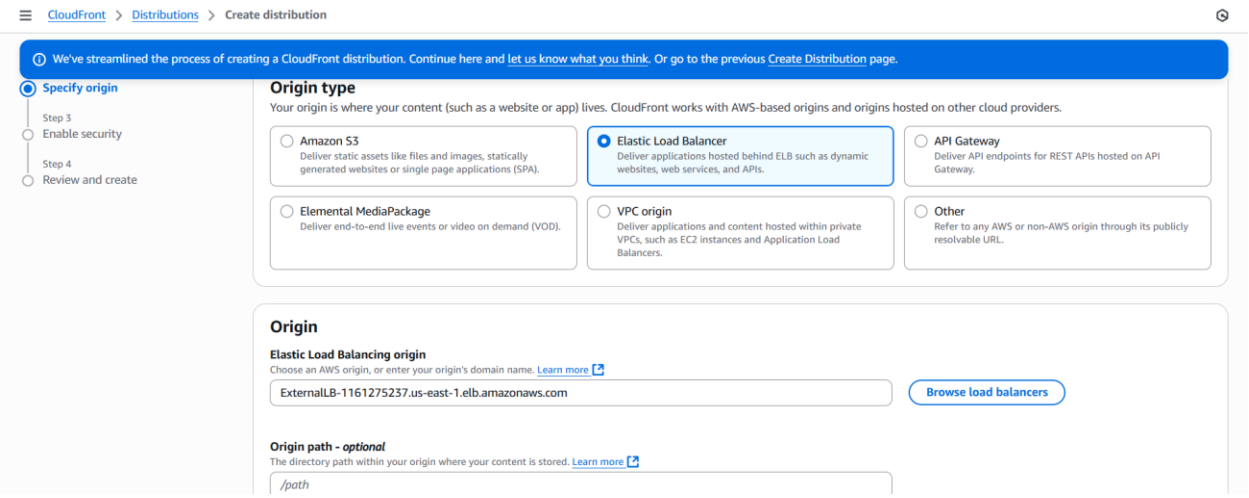
Step14: Enabling CloudFront Distribution (CDN)

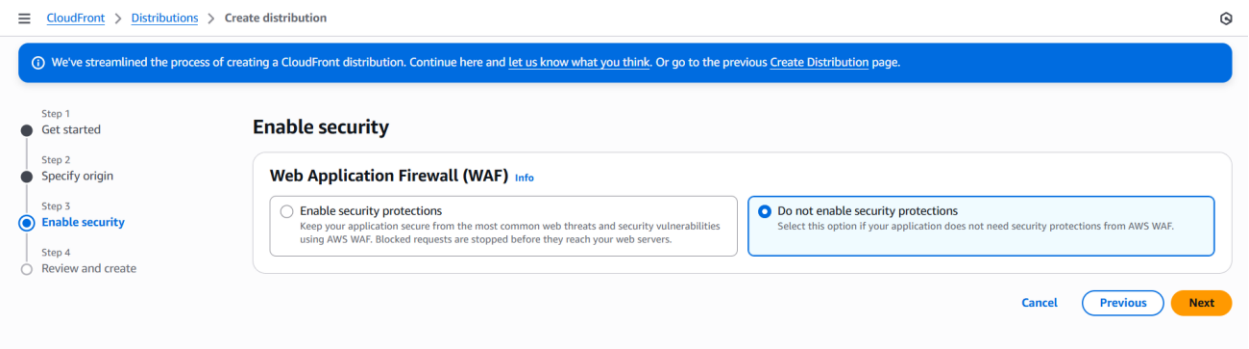
- Amazon CloudFront is a content delivery network service that accelerates the delivery of your static and dynamic web content to users globally.

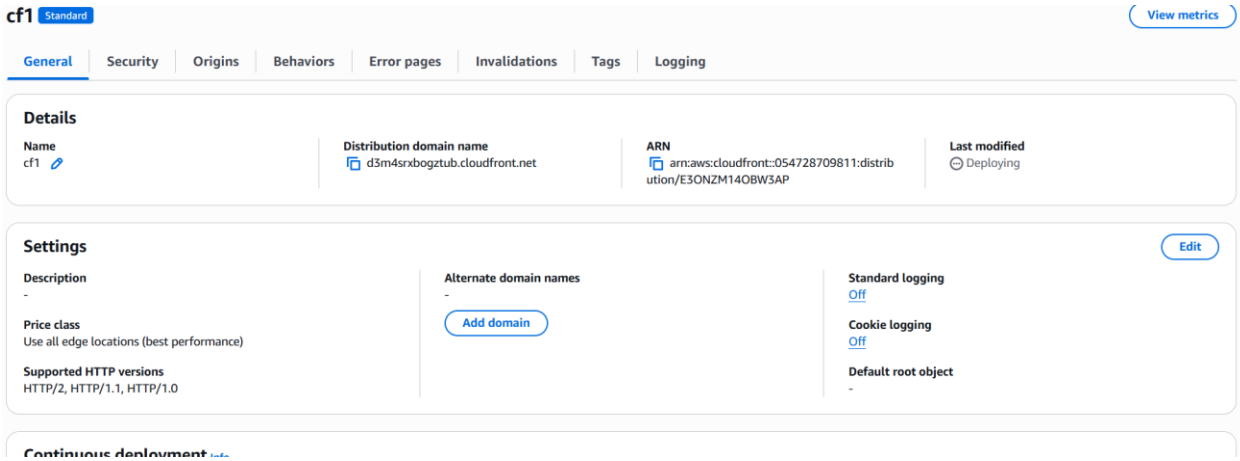
- It achieves this by caching content at edge locations (data centers) around the world, bringing the content closer to your users and reducing latency.

→ Attach the External Load Balancer inside the CloudFront.









Run CDN through Http. Copy the domain and run in browser,

Creating Hosted Zone

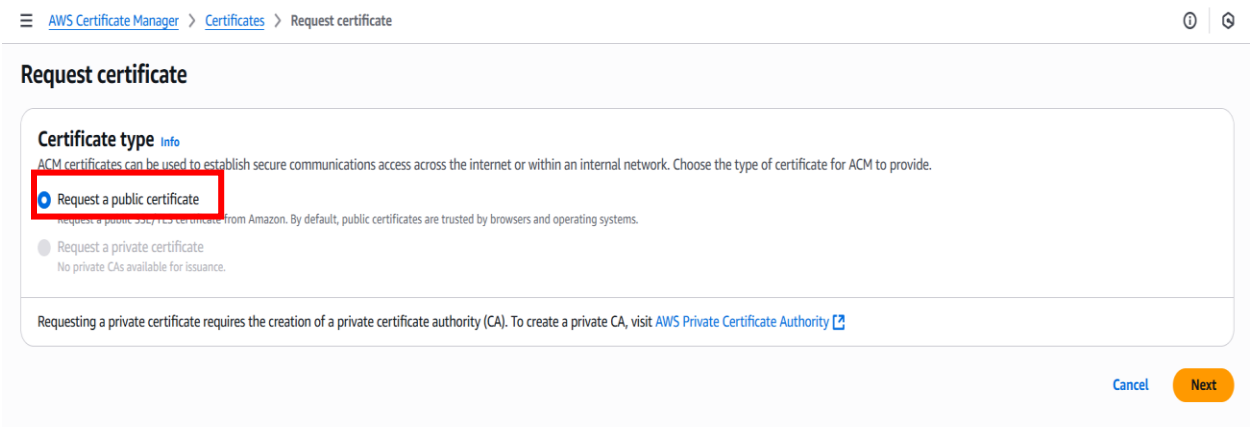
→ Purchase the Domain in GoDaddy website and Create the Hosted Zone with the same Domain name in Route 53 .

→ Automatically Namespaces will be created inside the Hosted Zone.

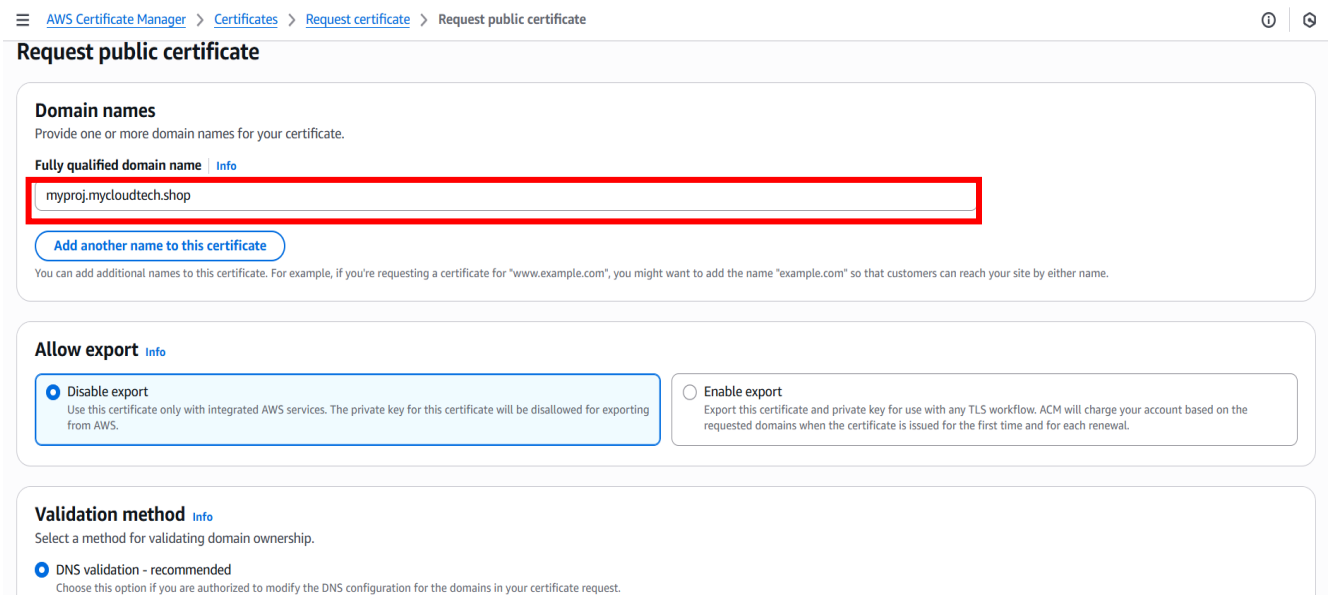
→ Copy the namespaces and paste inside the GoDaddy Domain website.

Enabling SSL Certificate (ACM)

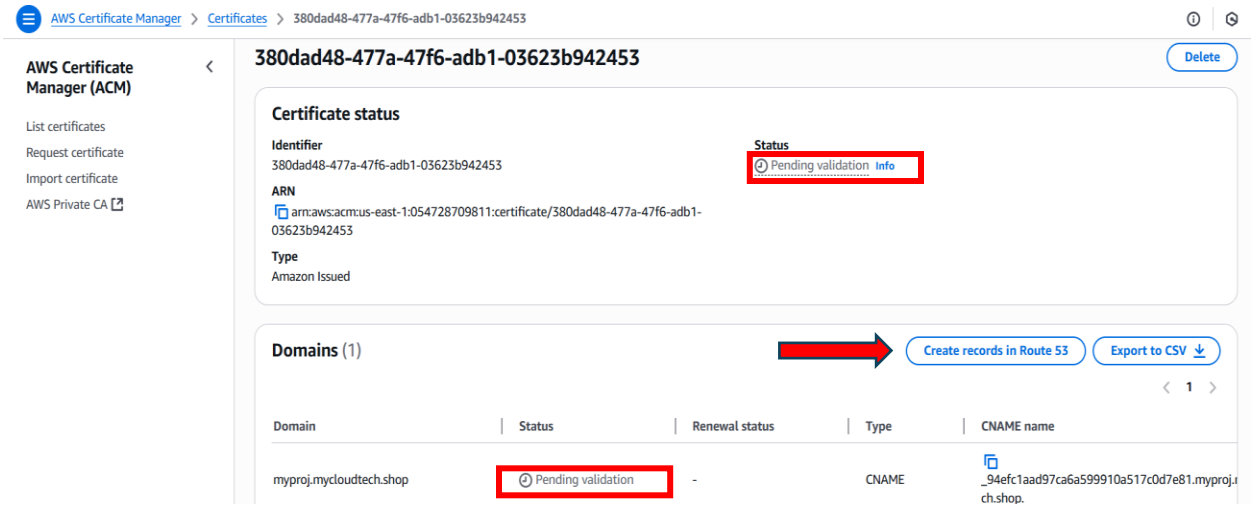
➞ Request a Public Certificate

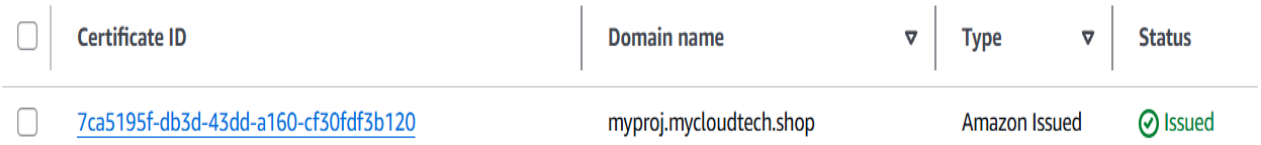


➞ Provide the fully qualified Domain Name (contains sub domain and main domain)



➡ Create the DNS record in Route 53.

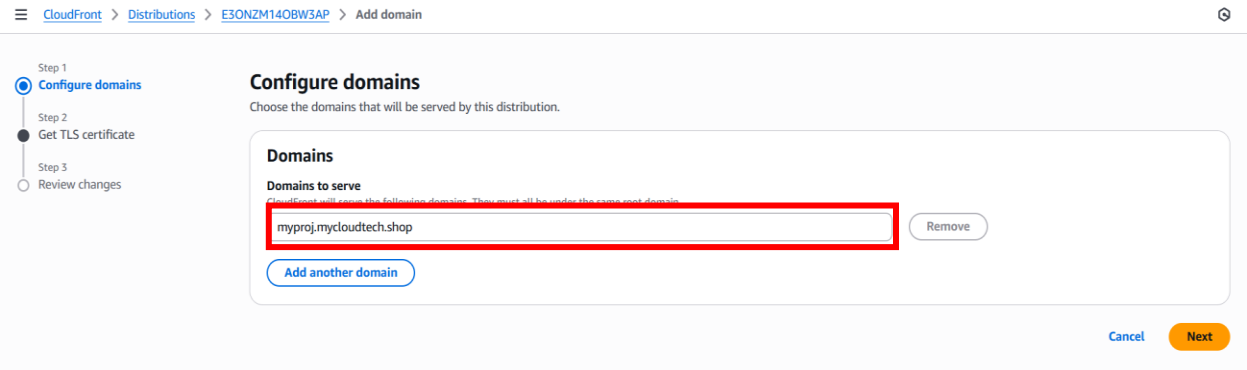




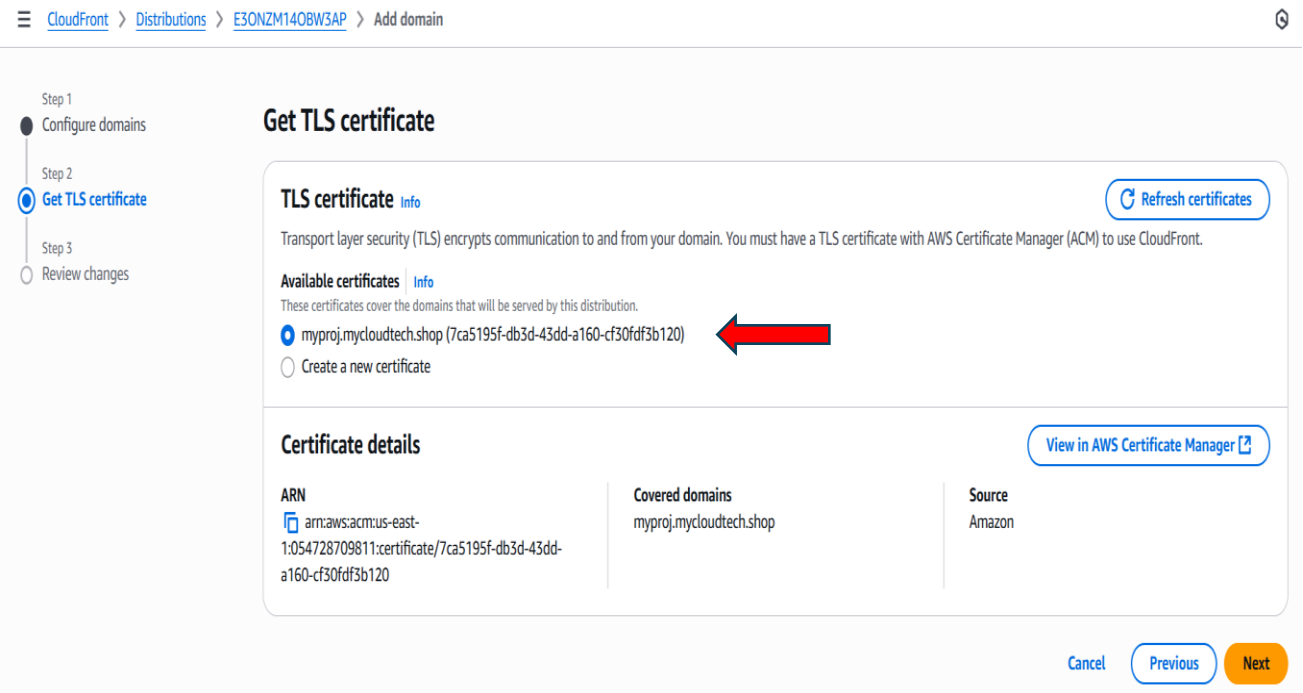
DNS Record is Created in Route 53 and Request for public certificate (SSL) is Issued.

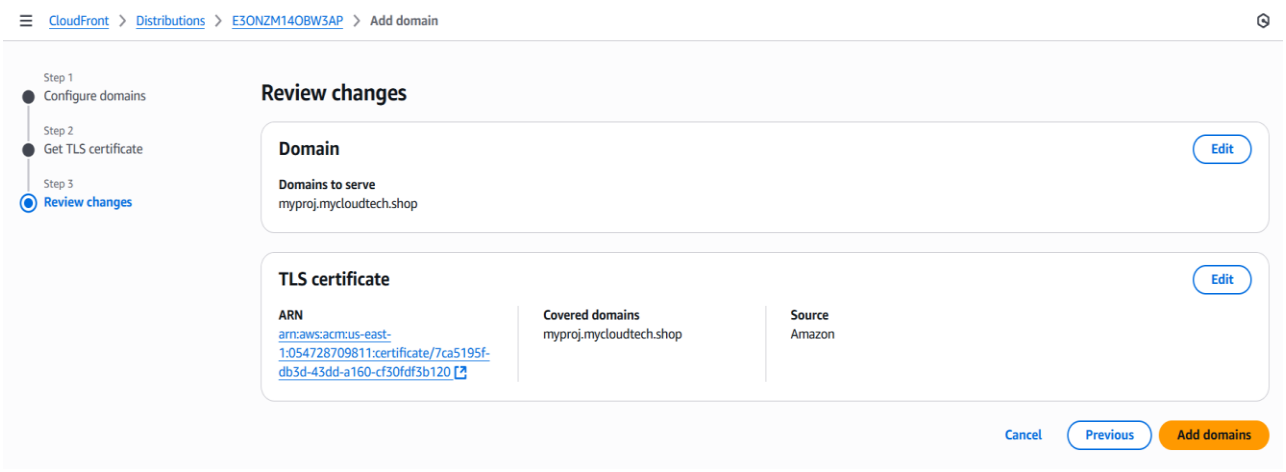
Adding Domain in CloudFront

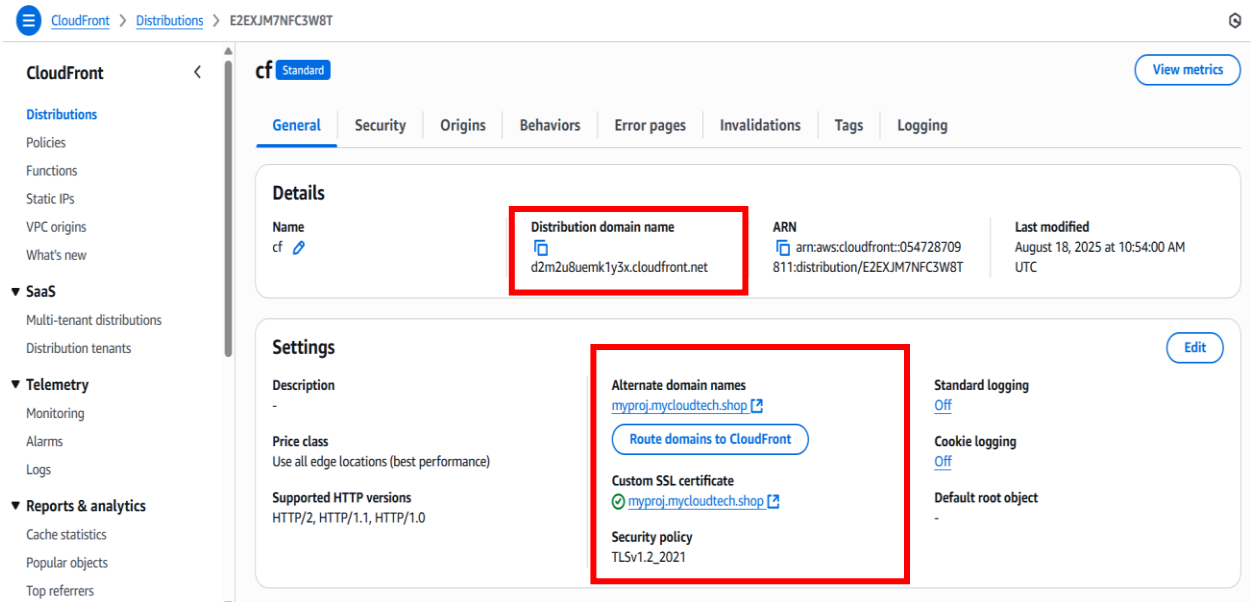
→ Enter Fully qualified Domain Name which we have created for SSL.



Now, Certificate is selected, Add the Domain.





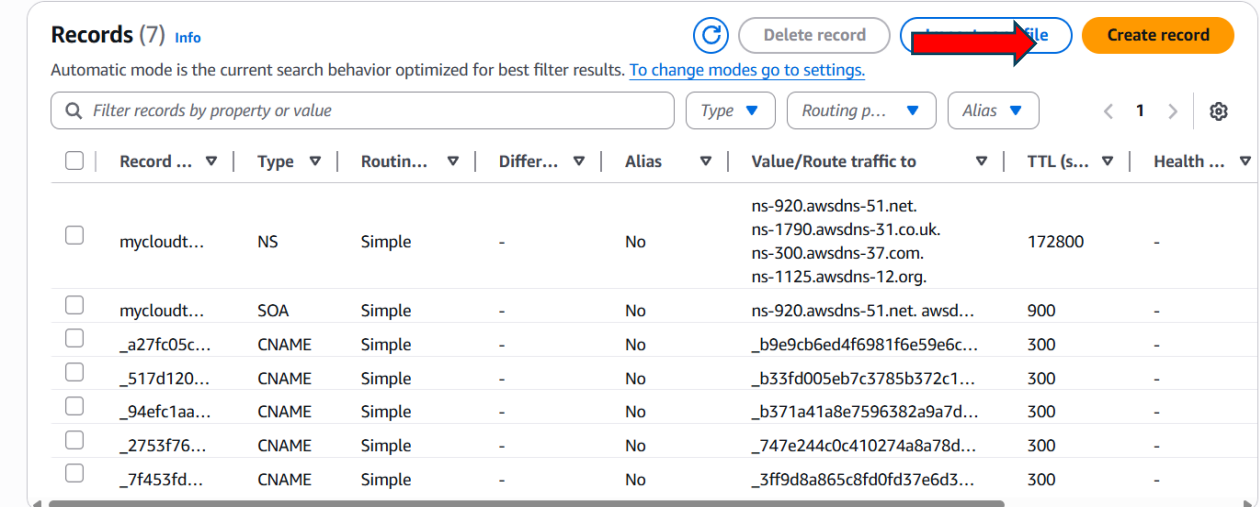


→ SSL Certificate is enabled for CloudFront.

→ Now, Route the CloudFront Distribution in Route 53.

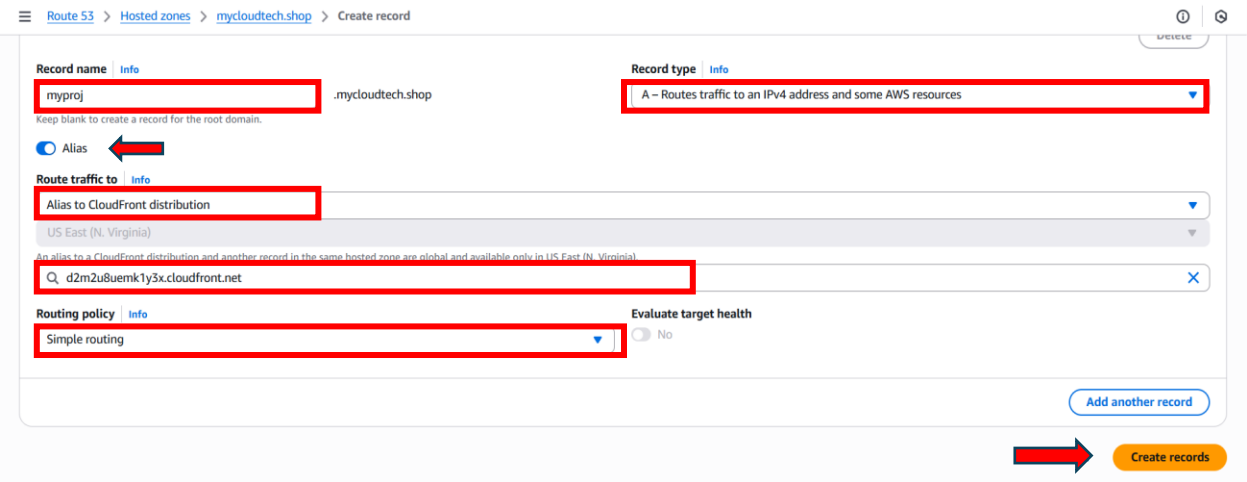
Step15: Hosting Application in Route53

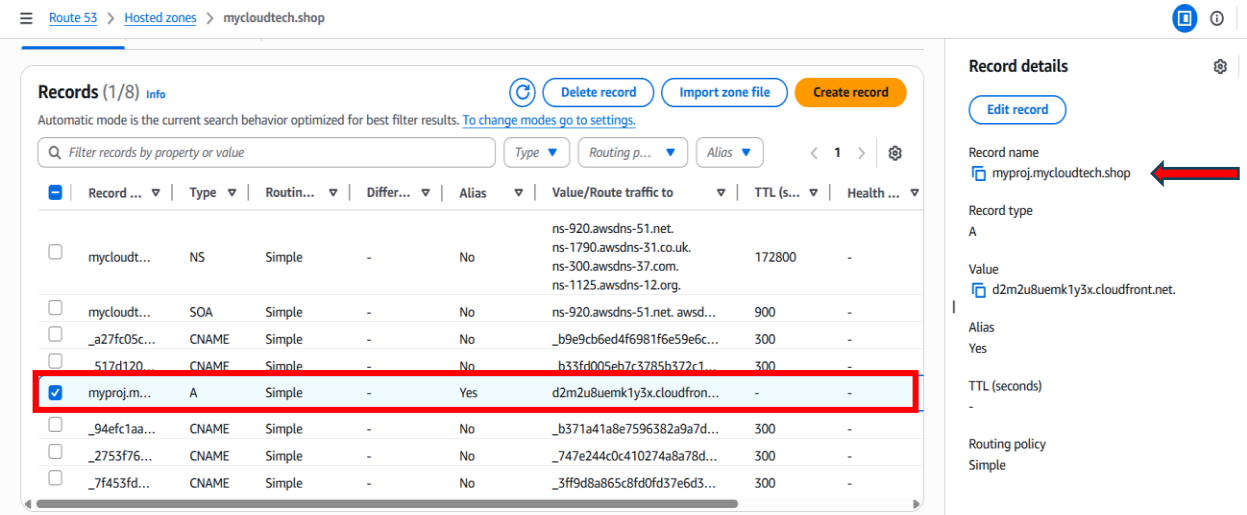
- Amazon Route 53 is to act as a highly available and scalable Domain Name System (DNS) web service for routing internet traffic to applications and resources
- It translates human-readable domain names (like "www.example.com") into the numerical IP addresses that computers use to connect. Additionally, Route 53 offers domain registration and health checking capabilities.



Create the Record in Hosted Zone:

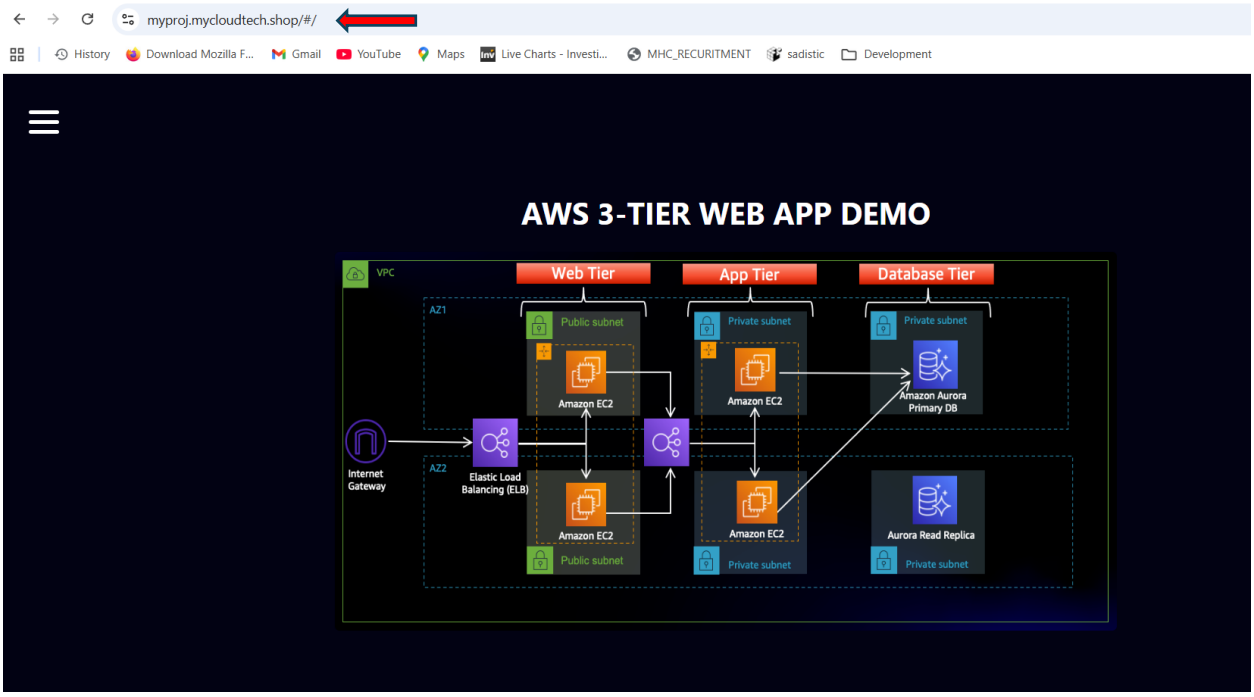
→ Select the Record type
→ Enable Alias
→ Route the traffic : Alias to CloudFront distribution
→ Selecting Simple routing policy.

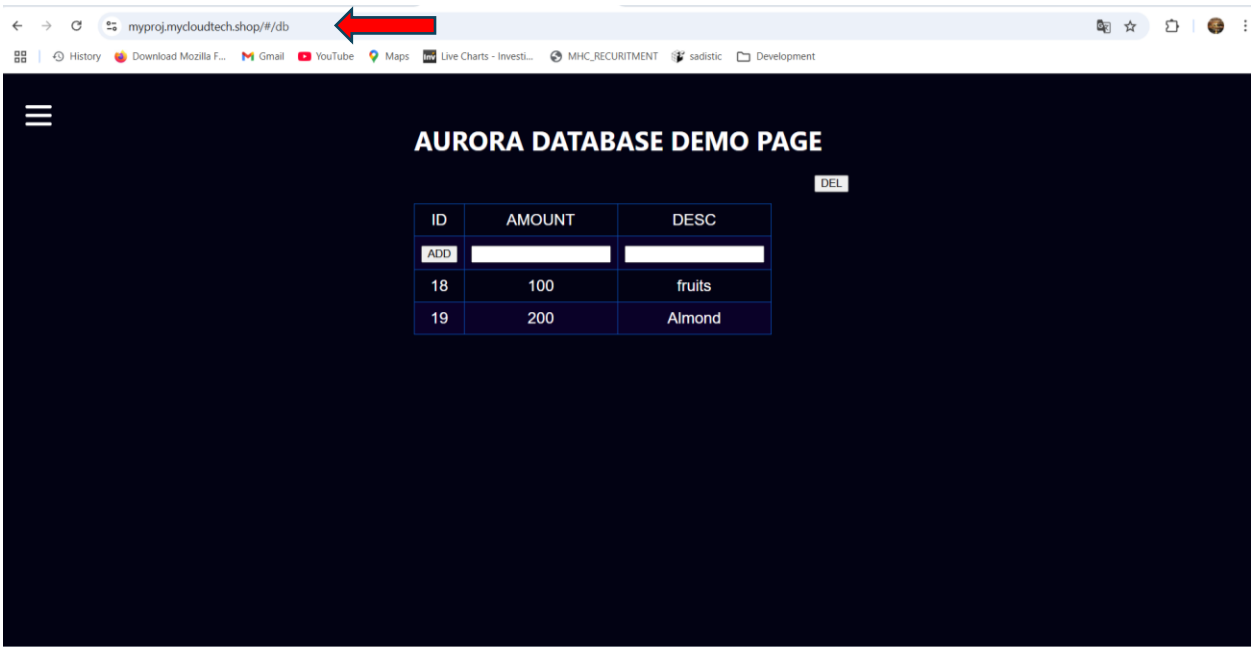




Now our application is hosted in Route 53, lets run the application.

Run: myproj.mycloudtech.shop





AWS Three-Tier Architecture Project has deployed Successfully.