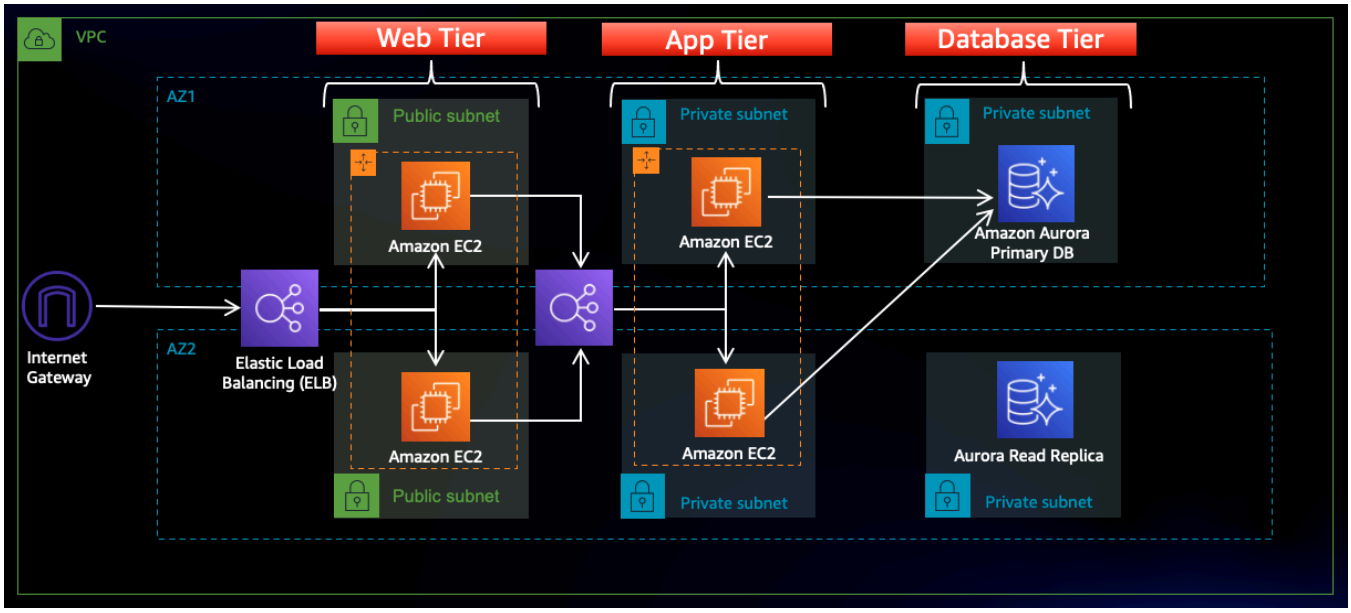
Build a Three Tier Architecture



In this architecture, a public-facing Application Load Balancer forwards client traffic to our web tier EC2 instances. The web tier is running Nginx web servers that are configured to serve a React.js website and redirects our API calls to the application tier's internal facing load balancer. The internal facing load balancer then forwards that traffic to the application tier, which is written in Node.js. The application tier manipulates data in an Aurora MySQL multi-AZ database and returns it to our web tier. Load balancing, health checks and autoscaling groups are created at each layer to maintain the availability of this architecture.

What is a three-tier architecture?

A three-tier application is a type of software architecture that divides an application into three logical layers: presentation, application (or logic), and data. Each layer serves a specific purpose and communicates with the other layers to perform various functions within the application.

Parts of a three-tier architecture

Presentation Tier (Front-end)

This layer is responsible for handling user interactions and displaying information to users.

It includes user interfaces, such as web pages, mobile apps, or desktop interfaces. The presentation tier communicates with the application tier to request and receive data, but it doesn't directly interact with the data storage.

Application Tier (Middle-tier or Logic Tier)

This layer contains the core logic and functionality of the application.

It processes user requests from the presentation tier, executes business logic, and interacts with the data tier to retrieve or update data.

The application tier acts as an intermediary between the presentation tier and the data tier, ensuring separation of concerns and modularity.

Data Tier (Back-end)

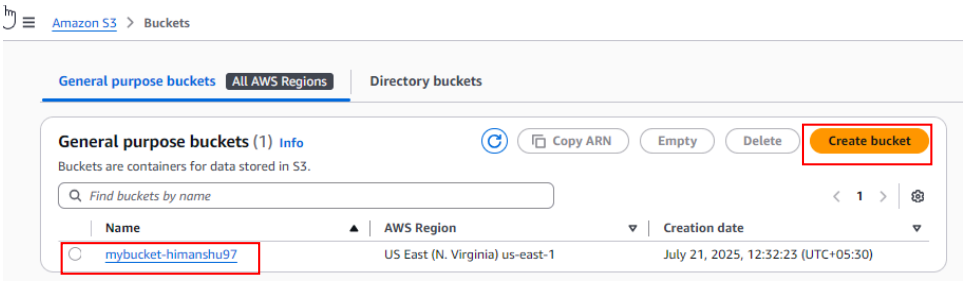
This layer manages the storage and retrieval of data used by the application.

It typically consists of databases or other data storage systems.

The data tier stores and organizes data, handles data retrieval and manipulation requests from the application tier, and ensures data integrity and security.

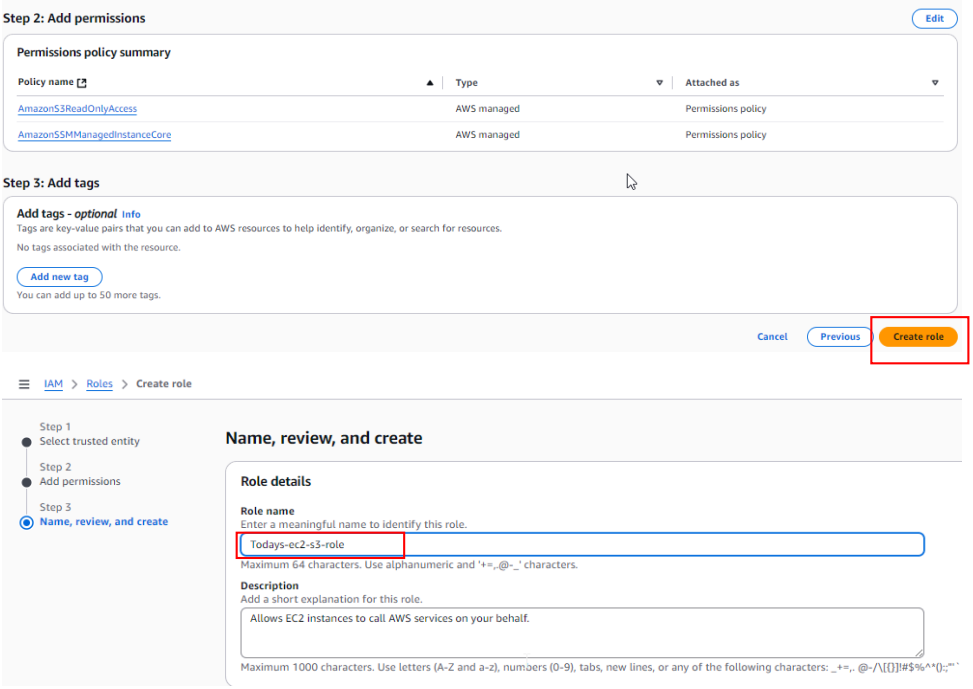
Navigate to the S3 service in the AWS console and create a new S3 bucket.

Give it a name → Create Bucket Give the Bucket Name → Select Region → Enabled ACLs → Enable Bucket Versioning → Click on Create



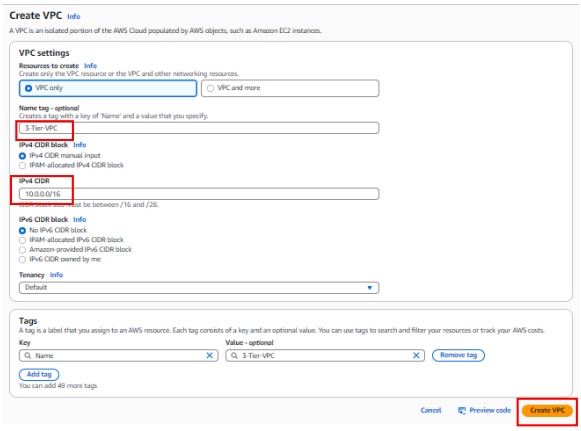
Navigate to the IAM dashboard in the AWS console and create an EC2 role.

Click on IAM → Roles → Create Roles → Selected Trusted Entity → AWS service → use case (EC2) → Add Permission → we need two policies AmazonSSMManagedInstanceCore AmazonS3ReadOnlyAccess → Click on Create Role



In this section we will be building out the VPC networking components as well as security groups that will add a layer of protection around our EC2 instances, Aurora databases, and Elastic Load Balancers.

Create VPC → Define IPv4 CIDR → and click on Create VPC



We will need six subnets across two availability zones. That means that three subnets will be in one availability zone, and three subnets will be in another zone. Each subnet in one availability zone will correspond to one layer of our three tier architecture. Create each of the 6 subnets by specifying the VPC we created in part 1 and then choose a name, availability zone, and appropriate CIDR range for each of the subnets.

Public-Web-Subnet-AZ-1 -10.0.0.0/24

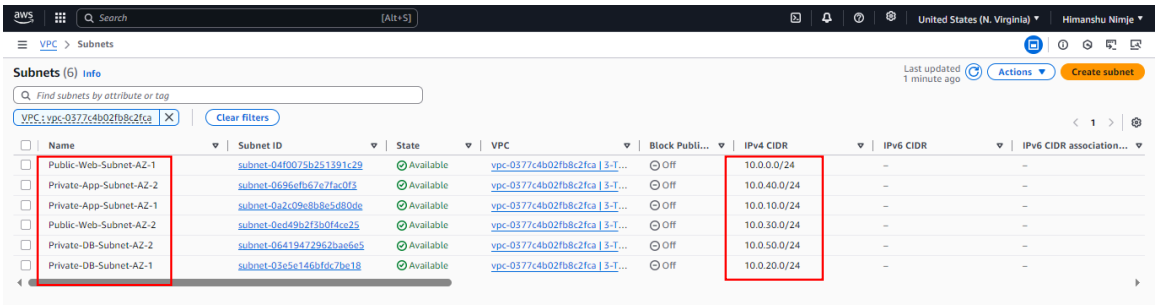
Private-App-Subnet-AZ-1 -10.0.10.0/24

Private-DB-Subnet-AZ-1 -10.0.20.0/24

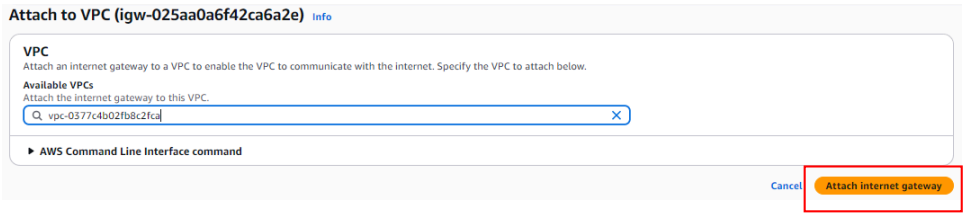
Public-Web-Subnet-AZ-2 -10.0.30.0/24

Private-App-Subnet-AZ-2 -10.0.40.0/24

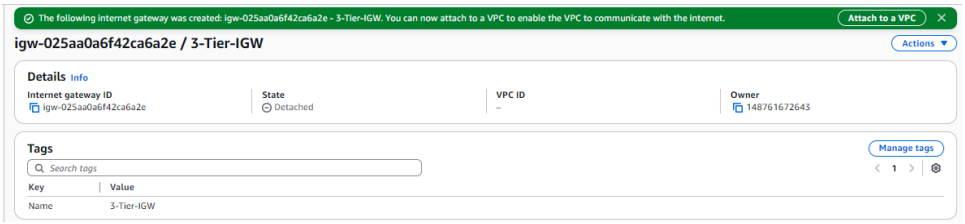
Private-DB-Subnet-AZ-2 -10.0.50.0/24



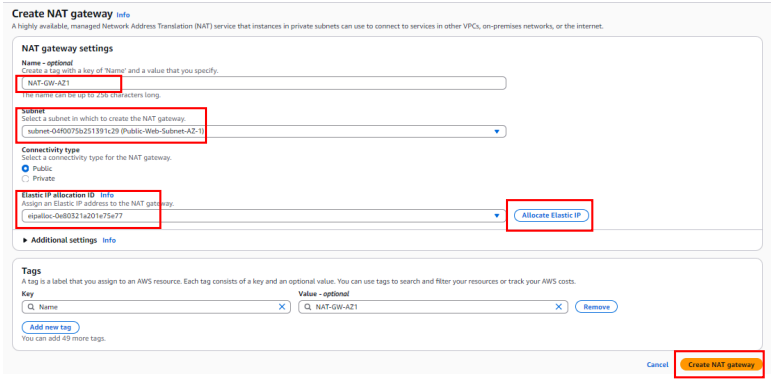
Create Internet Gateway



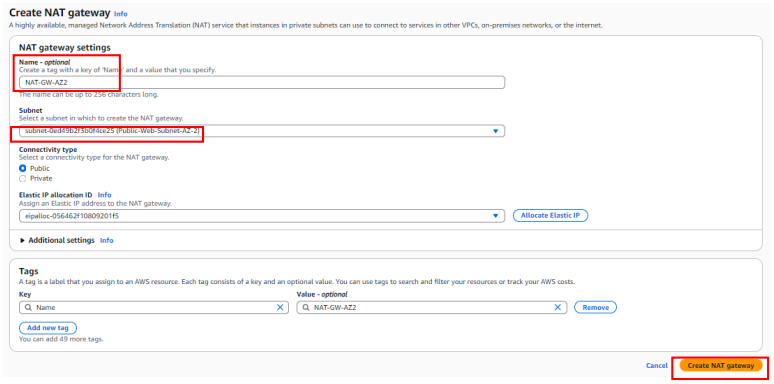
And Attach to Custom VPC that we made



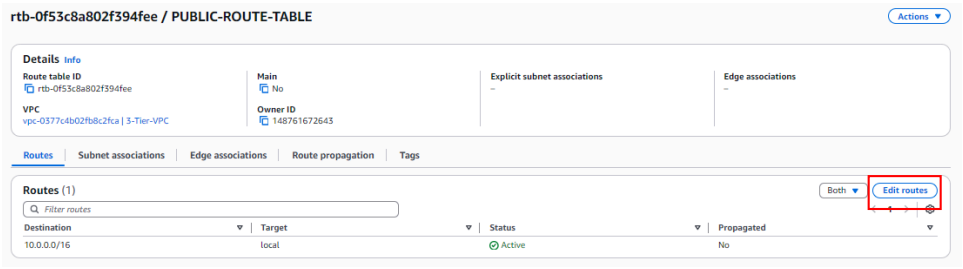
In order for our instances in the app layer private subnet to be able to access the internet they will need to go through a NAT Gateway. For high availability, you'll deploy one NAT gateway in each of your public subnets. Navigate to NAT Gateways on the left side of the current dashboard and click Create NAT Gateway.

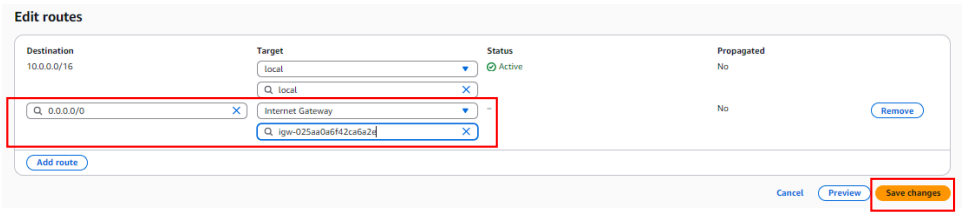


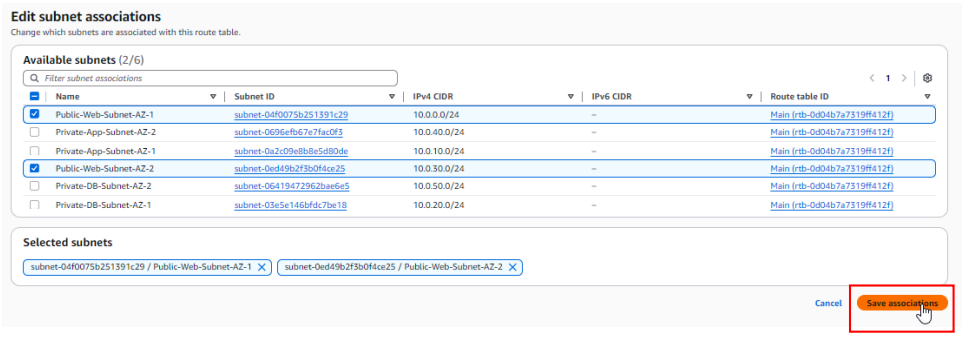
Fill in the Name, choose one of the public subnets you created, and then allocate an Elastic IP. Click Create NAT gateway.



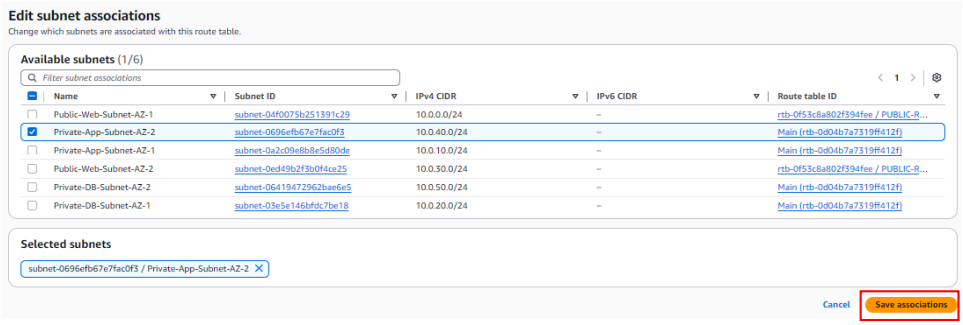
Here we need 3 Route Table
One is public Route Table
And Two PrivateRoute Table for AZ1 and AZ2
Click on create route → give it name (Public Route Table) → select custom VPC → after create route table → got to edit routes → add routes 0.0.0.0 → target → Internet gateway after edit routes → click on subnet association → select both → public Subnet AZ1 and AZ2 → and Click on Save Changes

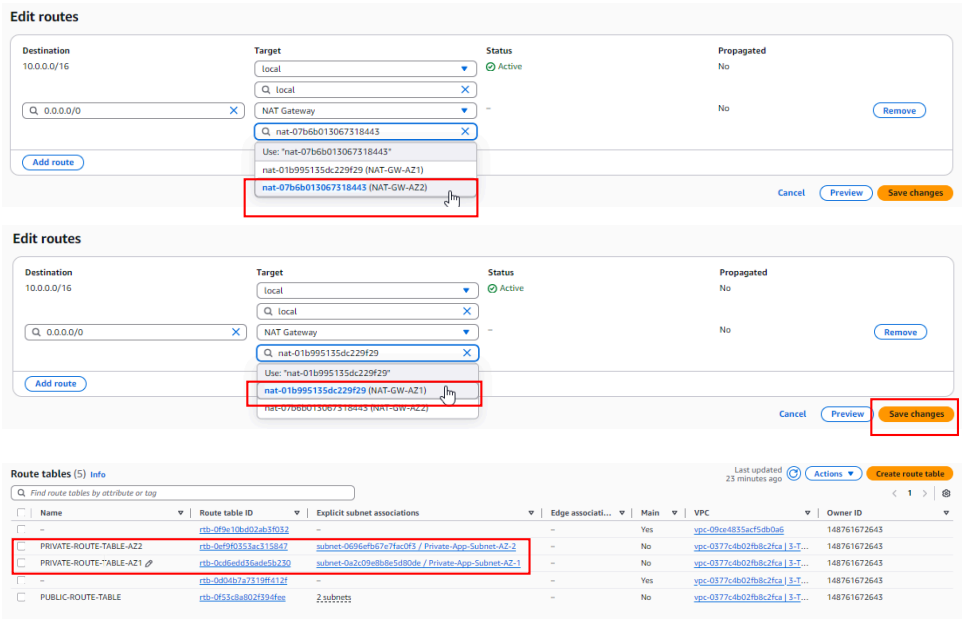






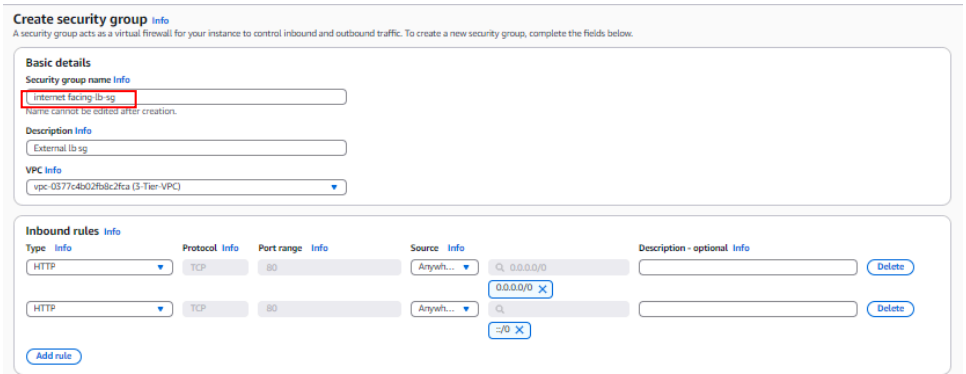
Create Private route table for AZ1 → select private app subnet AZ1
After creating private route table 1 → edit route -> edit route 0.0.0.0 → Source Target is → NAT gateway AZ1 (same step for private route table 2)



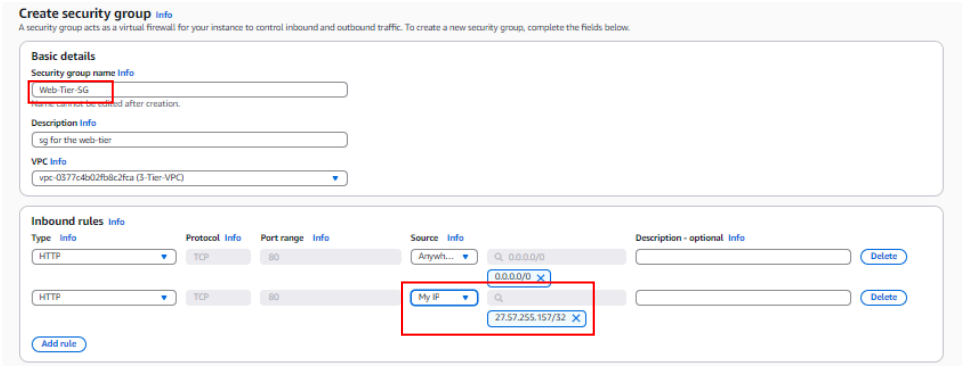


Security groups will tighten the rules around which traffic will be allowed to our Elastic Load Balancers and EC2 instances.

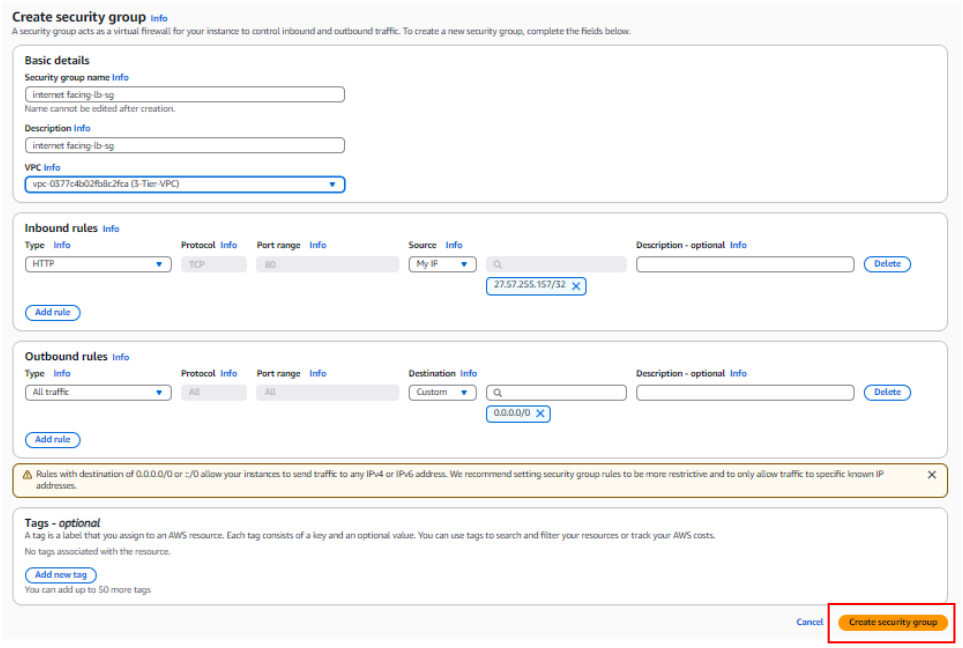
The first security group you'll create is for the public, internet facing load balancer. After typing a name and description, add an inbound rule to allow HTTP type traffic for your IP.

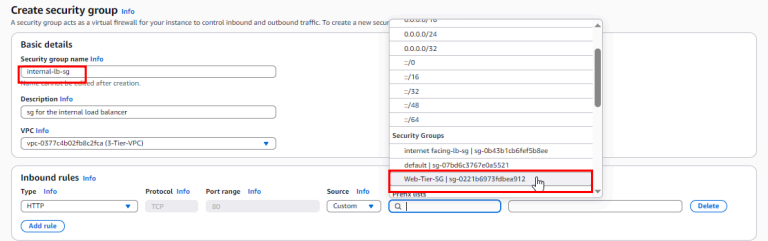


The second security group you'll create is for the public instances in the web tier. After typing a name and description, add an inbound rule that allows HTTP type traffic from your internet facing load balancer security group you created in the previous step. This will allow traffic from your public facing load balancer to hit your instances. Then, add an additional rule that will allow HTTP type traffic for your IP. This will allow you to access your instance when we test.

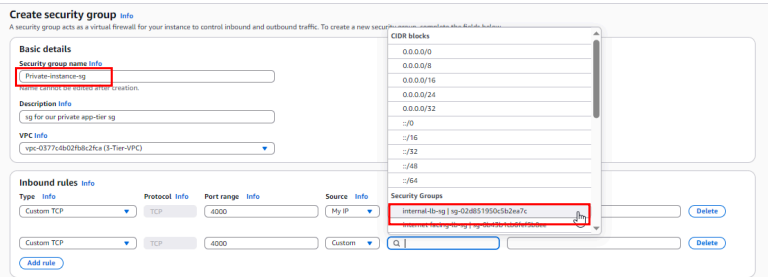


The third security group will be for our internal load balancer. Create this new security group and add an inbound rule that allows HTTP type traffic from your public instance security group. This will allow traffic from your web tier instances to hit your internal load balancer.

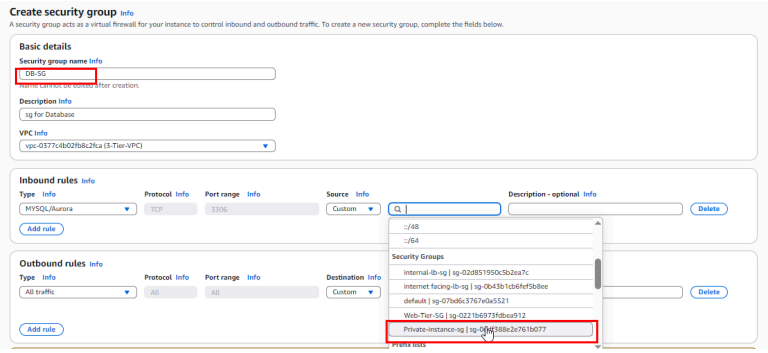


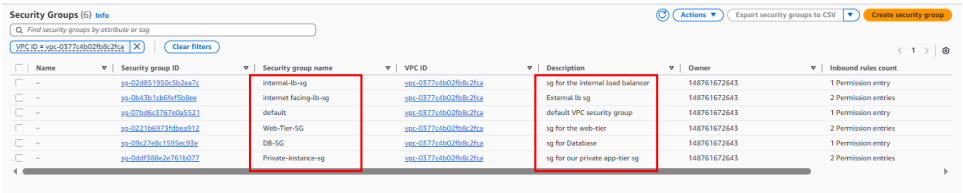


The fourth security group we'll configure is for our private instances. After typing a name and description, add an inbound rule that will allow TCP type traffic on port 4000 from the internal load balancer security group you created in the previous step. This is the port our app tier application is running on and allows our internal load balancer to forward traffic on this port to our private instances. You should also add another route for port 4000 that allows your IP for testing.



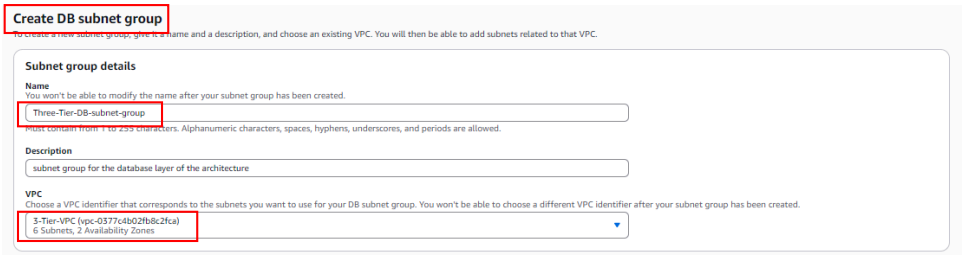
The fifth security group we'll configure protects our private database instances. For this security group, add an inbound rule that will allow traffic from the private instance security group to the MYSQL/Aurora port (3306).

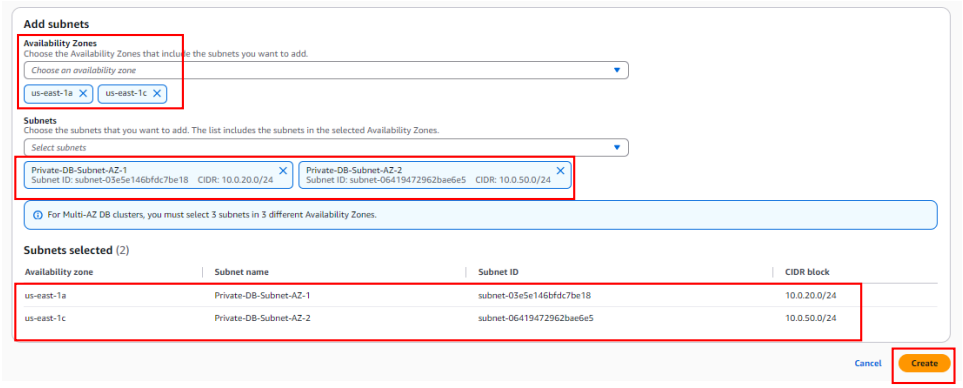




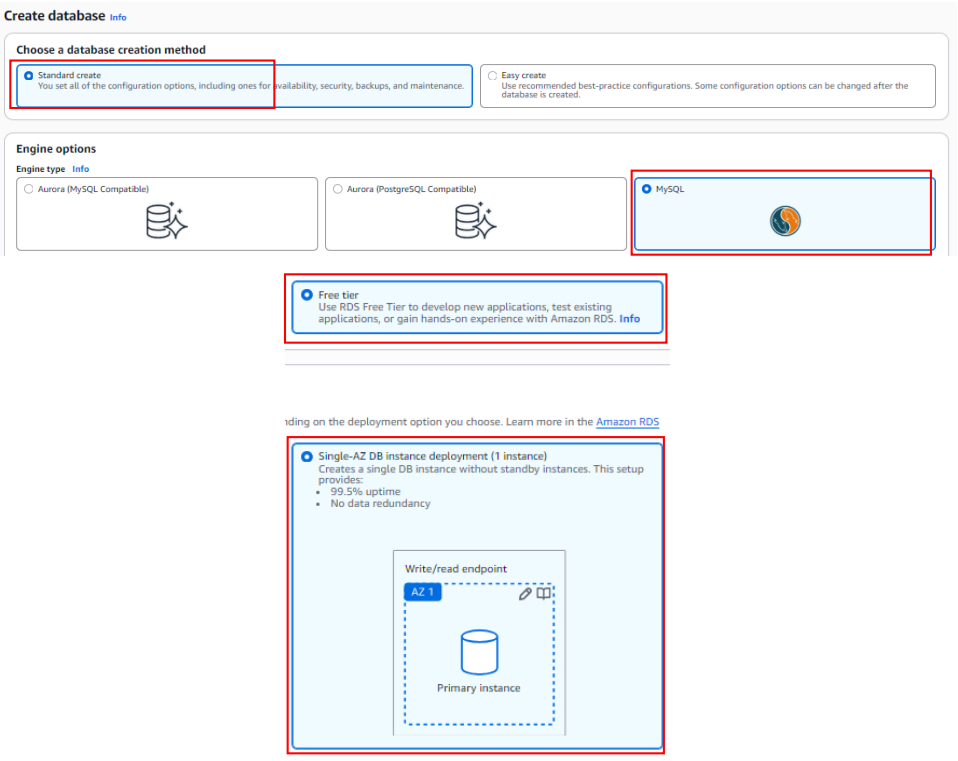
Database Deployment

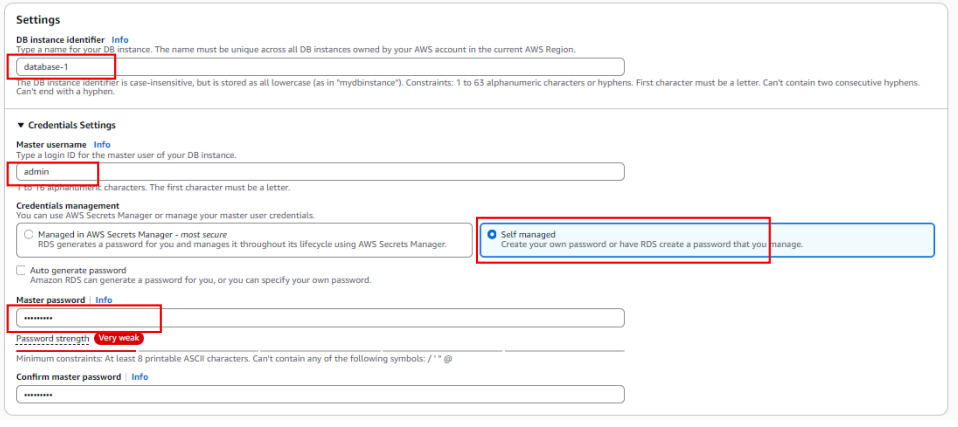
First we create a subnet group → give it a name → give the description → select custom vpc → select AZ → Add subnet → click on create subnet

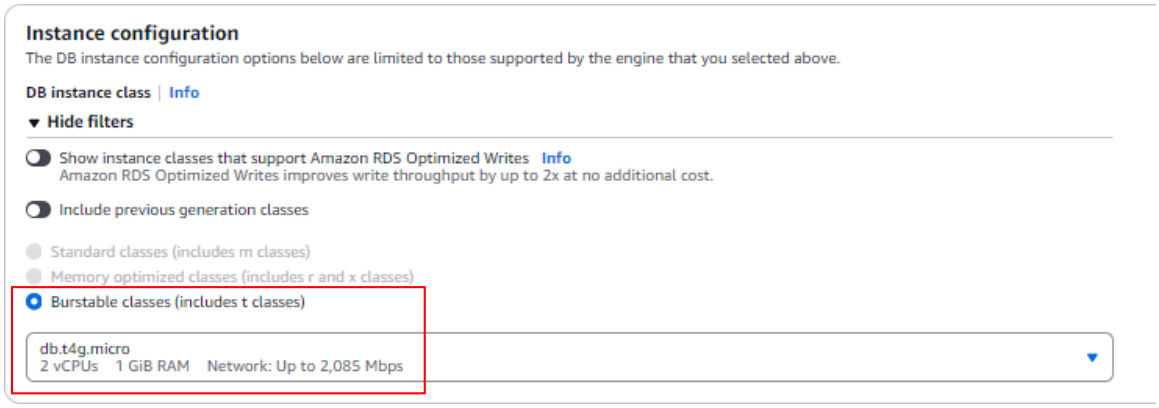


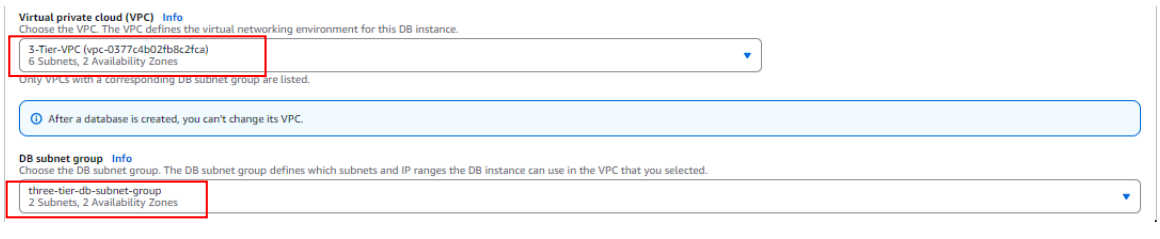


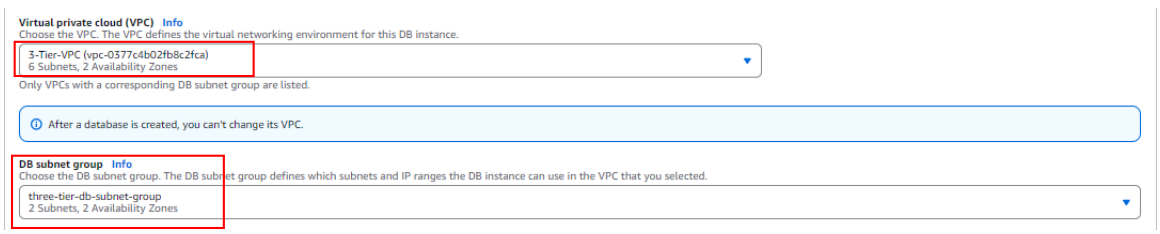
After creating subnet group → we create Database → click on Database → select creating method (Standard) → select engine (mysql) → free tier → Master user name → admin → give it a password → instance configuration (default) → select custom VPC → select Db subnet group → select security group (DB-SG) → Click on Create Database

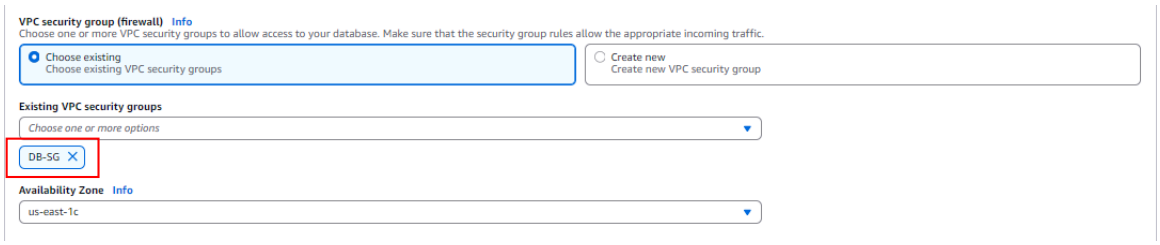


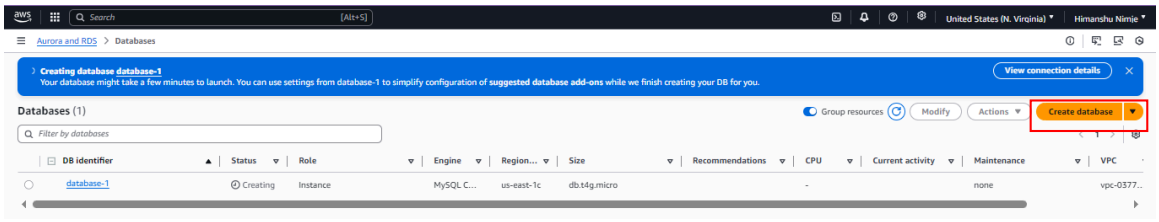


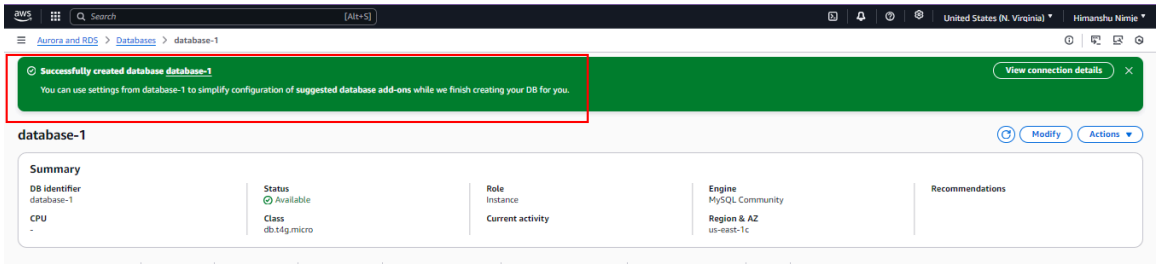

Instance configuration
The DB instance configuration options below are limited to those supported by the engine that you selected above.
DB instance class | Info
Hide filters
Show instance classes that support Amazon RDS Optimized Writes Info
Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.
Include previous generation classes
Standard classes (includes m classes)
Memory optimized classes (includes r and x classes)
Burstable classes (includes t classes)
db.t4g.micro
2 vCPUs 1 GiB RAM Network: Up to 2,085 Mbps


Virtual private cloud (VPC) Info
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.
3-Tier-VPC (vpc-0377c4b02fb8c2fca)
6 Subnets, 2 Availability Zones
Only VPCs with a corresponding DB subnet group are listed.
After a database is created, you can't change its VPC.
DB subnet group Info
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.
three-tier-db-subnet-group
2 Subnets, 2 Availability Zones


Virtual private cloud (VPC) Info
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.
3-Tier-VPC (vpc-0377c4b02fb8c2fca)
6 Subnets, 2 Availability Zones
Only VPCs with a corresponding DB subnet group are listed.
After a database is created, you can't change its VPC.
DB subnet group Info
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.
three-tier-db-subnet-group
2 Subnets, 2 Availability Zones

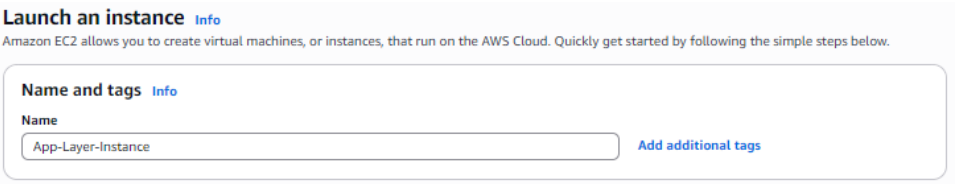

VPC security group (firewall) Info
Choose one or more VPC security groups to allow access to your database. Make sure that the security group rules allow the appropriate incoming traffic.
Choose existing
Choose existing VPC security groups
Create new
Create new VPC security group
Existing VPC security groups
Choose one or more options
DB-SG
Availability Zone Info
us-east-1c


Creating database database-1
Your database might take a few minutes to launch. You can use settings from database-1 to simplify configuration of suggested database add-ons while we finish creating your DB for you.
View connection details
Databases (1)
Group resources
Modify
Actions
Create database
DB identifier
Status
Role
Engine
Region...
Size
Recommendations
CPU
Current activity
Maintenance
VPC
database-1
Creating
Instance
MySQL C...
us-east-1c
db.t4g.micro
none

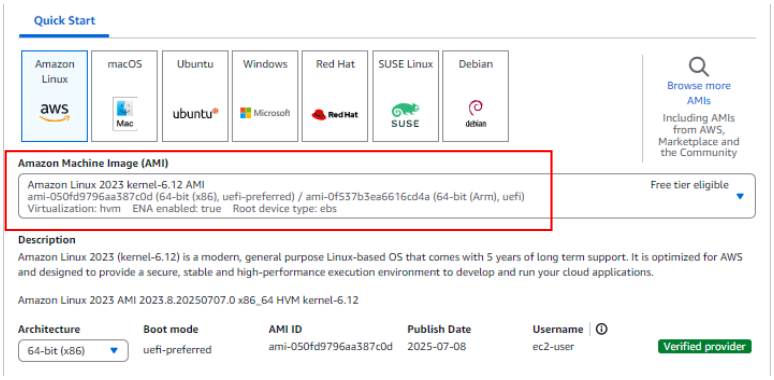

Successfully created database database-1
You can use settings from database-1 to simplify configuration of suggested database add-ons while we finish creating your DB for you.
View connection details
database-1
Modify
Actions
Summary
DB identifier
database-1
Status
Available
Role
Instance
Engine
MySQL Community
Recommendations
CPU
Class
db.t4g.micro
Current activity
Region & AZ
us-east-1c

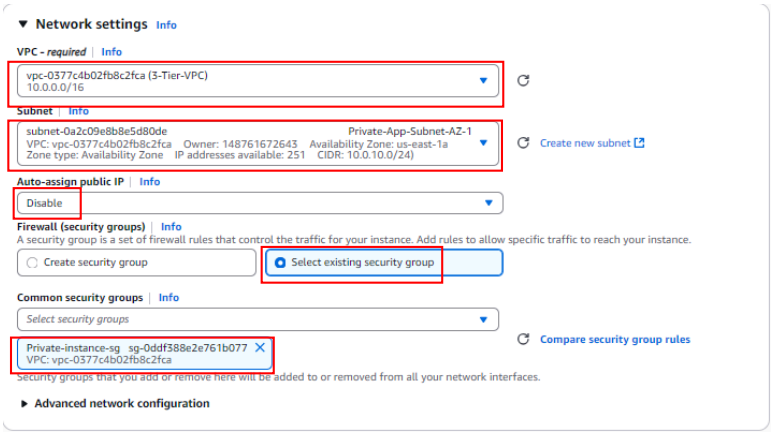
App Tier Instance Deployment

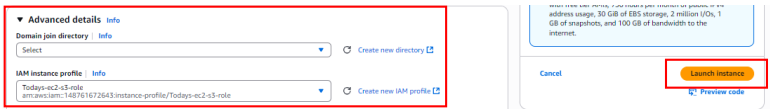
In this section of our project we will create an EC2 instance for our app layer and make all necessary software configurations so that the app can run. The app layer consists of a Node.js application that will run on port 4000. We will also configure our database with some data and tables.



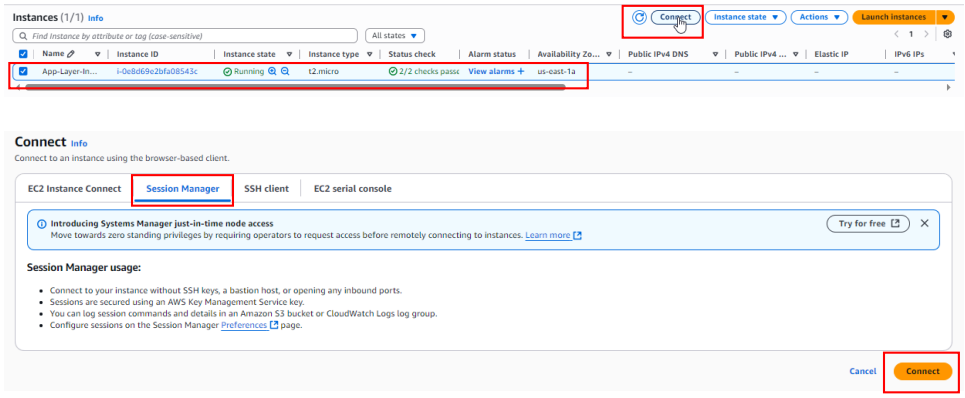
Click on launch instance → give it name → select Amazon machine image → instance type (t2-micro) → processed without key pair → select vpc → subnet (select private App subnet AZ1) → public ip disable → in security group (select private instance sg) → click on Advanced Details → IAM profile (select ec2 and s3 role that we created) → click on launch instance







Connect to instances



`sudo -su ec2-user`

Let's take this moment to make sure that we are able to reach the internet via our NAT gateways.

ping 8.8.8.8

```
sh-5.2$ sudo -su ec2-user
[ec2-user@ip-10-0-10-133 bin]$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=2.01 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=1.33 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=1.39 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=1.31 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=116 time=1.32 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=116 time=1.26 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=116 time=1.29 ms
^C
--- 8.8.8.8 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6008ms
rtt min/avg/max/mdev = 1.264/1.415/2.011/0.245 ms
[ec2-user@ip-10-0-10-133 bin]$
```

```
[ec2-user@ip-10-0-10-133 bin]$ sudo yum install mysql -y
Amazon Linux 2023 Kernel Livepatch repository                    159 kB/s |  17 kB     00:00
No match for argument: mysql
Error: Unable to find a match: mysql
[ec2-user@ip-10-0-10-133 bin]$
```

`sudo wget https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm`

```
[ec2-user@ip-10-0-10-133 bin]$ wget https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm
```

sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022

```
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
```

sudo yum install mysql80-community-release-el9-1.noarch.rpm

```
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
[ec2-user@ip-10-0-10-133 bin]$ sudo yum install mysql80-community-release-el9-1.noarch.rpm
```

sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022

```
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
[ec2-user@ip-10-0-10-133 bin]$ sudo yum install mysql
```

```
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm -e gpg-pubkey-3a79bd29-*
```

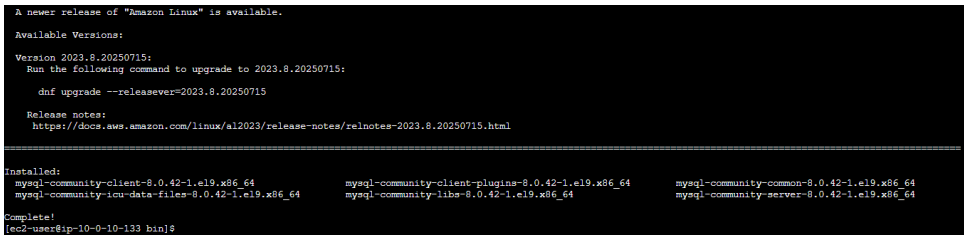
```
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm -e gpg-pubkey-3a79bd29-*
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
```

YOU FOUND THIS ERROR MESSAGE → error: GPG check FAILED
SIMPLY TYPE THIS COMMAND ↓
sudo dnf install mysql-server --nogpgcheck -y

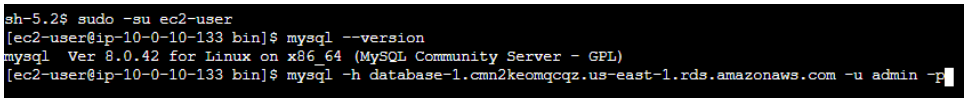
```
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm -e gpg-pubkey-3a79bd29-*
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
[ec2-user@ip-10-0-10-133 bin]$ sudo dnf install mysql-server --nogpgcheck -y
```

```
Error: GPG check FAILED
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm -e gpg-pubkey-3a79bd29-*
[ec2-user@ip-10-0-10-133 bin]$ sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
[ec2-user@ip-10-0-10-133 bin]$ sudo dnf install mysql-server --nogpgcheck -y
Last metadata expiration check: 0:03:23 ago on Mon Jul 21 15:42:50 2025.
Dependencies resolved.
================================================================================
 Package                          Architecture    Version              Repository                  Size
================================================================================
Installing:
 mysql-community-server           x86_64          8.0.42-1.el9         mysql80-community           50 M
Installing dependencies:
 mysql-community-client           x86_64          8.0.42-1.el9         mysql80-community          3.4 M
 mysql-community-client-plugins   x86_64          8.0.42-1.el9         mysql80-community          1.4 M
 mysql-community-common           x86_64          8.0.42-1.el9         mysql80-community          555 k
 mysql-community-icu-data-files   x86_64          8.0.42-1.el9         mysql80-community          2.3 M
 mysql-community-libs             x86_64          8.0.42-1.el9         mysql80-community          1.5 M

Transaction Summary
================================================================================
Install  6 Packages

Total size: 59 M
Total download size: 52 M
Installed size: 337 M
Downloading Packages:
[SKIPPED] mysql-community-client-8.0.42-1.el9.x86_64.rpm: Already downloaded
[SKIPPED] mysql-community-client-plugins-8.0.42-1.el9.x86_64.rpm: Already downloaded
[SKIPPED] mysql-community-common-8.0.42-1.el9.x86_64.rpm: Already downloaded
[SKIPPED] mysql-community-libs-8.0.42-1.el9.x86_64.rpm: Already downloaded
(5/6): mysql-community-icu-data-files-8.0.42-1.el9.x86_64.rpm  45 MB/s | 2.3 MB     00:00
(6/6): mysql-community-server-8.0.42-1.el9.x86_64.rpm          84 MB/s |  50 MB     00:00
--------------------------------------------------------------------------------
Total                                                          88 MB/s |  52 MB     00:00
Running transaction check
```

```
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : mysql-community-common-8.0.42-1.el9.x86_64             1/6
  Installing       : mysql-community-client-plugins-8.0.42-1.el9.x86_64     2/6
  Installing       : mysql-community-libs-8.0.42-1.el9.x86_64               3/6
  Running scriptlet: mysql-community-libs-8.0.42-1.el9.x86_64               3/6
  Installing       : mysql-community-client-8.0.42-1.el9.x86_64             4/6
  Installing       : mysql-community-icu-data-files-8.0.42-1.el9.x86_64     5/6
  Running scriptlet: mysql-community-server-8.0.42-1.el9.x86_64             6/6
  Installing       : mysql-community-server-8.0.42-1.el9.x86_64             6/6
  Running scriptlet: mysql-community-server-8.0.42-1.el9.x86_64             6/6
  Verifying        : mysql-community-client-8.0.42-1.el9.x86_64             1/6
  Verifying        : mysql-community-client-plugins-8.0.42-1.el9.x86_64     2/6
  Verifying        : mysql-community-common-8.0.42-1.el9.x86_64             3/6
  Verifying        : mysql-community-icu-data-files-8.0.42-1.el9.x86_64     4/6
  Verifying        : mysql-community-libs-8.0.42-1.el9.x86_64               5/6
  Verifying        : mysql-community-server-8.0.42-1.el9.x86_64             6/6
================================================================================
WARNING:
  A newer release of "Amazon Linux" is available.
```

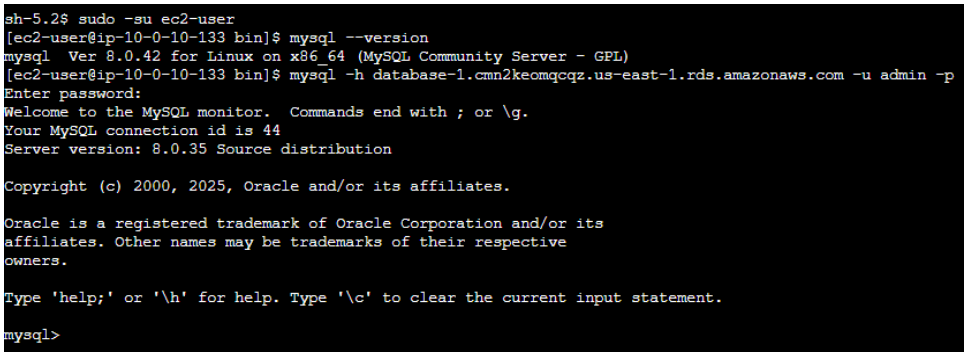


mysql --version



Initiate your DB connection with your Aurora RDS writer endpoint. In the following command, replace the RDS writer endpoint and the username, and then execute it in the browser terminal:

mysql -h CHANGE-TO-YOUR-RDS-ENDPOINT -u CHANGE-TO-USER-NAME -p



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

CREATE DATABASE webappdb;

```
sh-5.2$ sudo -su ec2-user
[ec2-user@ip-10-0-10-133 bin]$ mysql --version
mysql  Ver 8.0.42 for Linux on x86_64 (MySQL Community Server - GPL)
[ec2-user@ip-10-0-10-133 bin]$ mysql -h database-1.cmn2keomqcqz.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 44
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> CREATE DATABASE webappdb;
Query OK, 1 row affected (0.01 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| webappdb           |
+--------------------+
5 rows in set (0.00 sec)

mysql> USE webappdb;
Database changed
mysql>
```

You can verify that it was created correctly with the following command:
SHOW DATABASES;

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| webappdb           |
+--------------------+
5 rows in set (0.00 sec)

mysql> USE webappdb;
Database changed
mysql> 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 (0.05 sec)

mysql> INSERT INTO transactions (amount,description) VALUES ('400','groceries');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM transactions;
+----+--------+-------------+
| id | amount | description |
+----+--------+-------------+
|  1 | 400.00 | groceries   |
+----+--------+-------------+
1 row in set (0.00 sec)

mysql>
```

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));

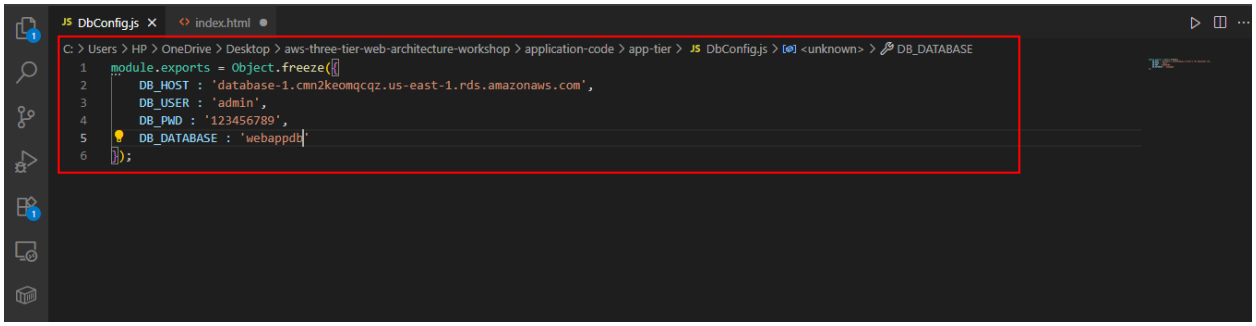
Verify the table was created:
SHOW TABLES;

Insert data into table for use/testing later:
INSERT INTO transactions (amount,description) VALUES ('400','groceries');

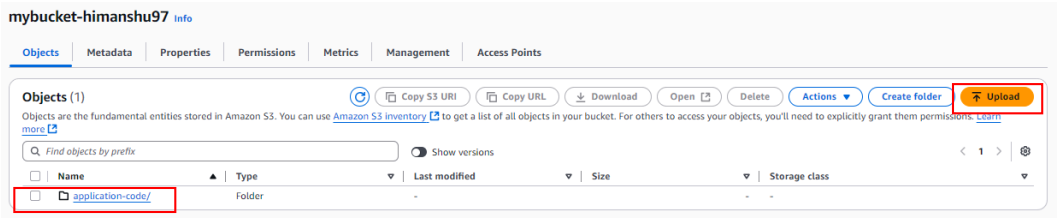
Verify that your data was added by executing the following command:
SELECT * FROM transactions;

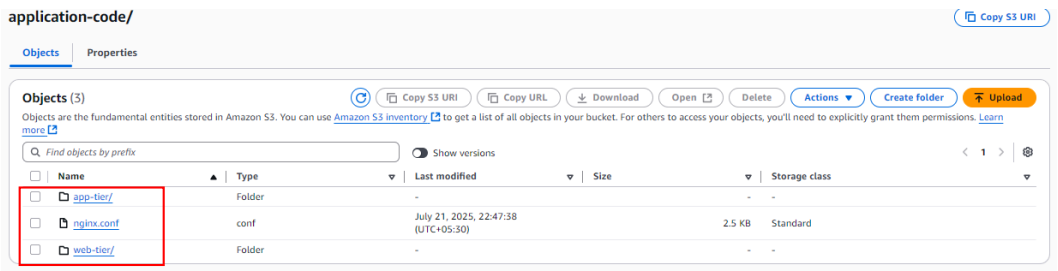
```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| webappdb           |
+--------------------+
5 rows in set (0.00 sec)

mysql> USE webappdb;
Database changed
mysql> 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 (0.05 sec)

mysql> INSERT INTO transactions (amount,description) VALUES ('400','groceries');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM transactions;
+----+--------+-------------+
| id | amount | description |
+----+--------+-------------+
|  1 | 400.00 | groceries   |
+----+--------+-------------+
1 row in set (0.00 sec)

mysql> exit
```

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.



Upload the app-tier folder to the S3 bucket





Now go back to our SSM session. Now we need to install all of the necessary components to run our backend application. Start by installing NVM (node version manager).

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

source ~/.bashrc

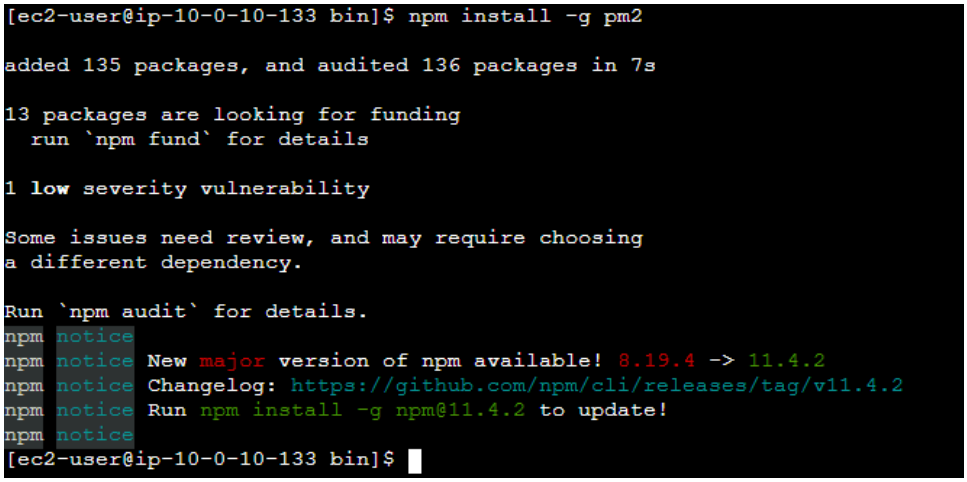
```
sh-5.2$ sudo -su ec2-user
[ec2-user@ip-10-0-10-133 bin]$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
```

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

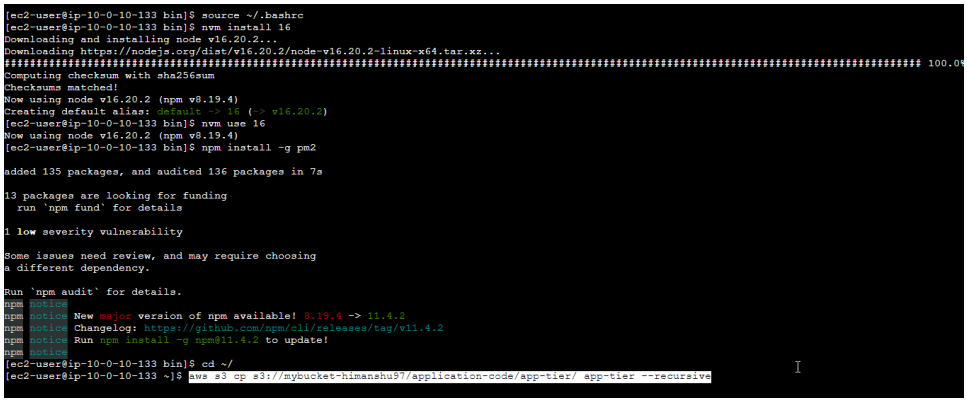
nvm use 16

```
sh-5.2$ sudo -su ec2-user
[ec2-user@ip-10-0-10-133 bin]$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 14926  100 14926    0     0   690k      0 --:--:-- --:--:-- --:--:--  728k
=> Downloading nvm as script to '/home/ec2-user/.nvm'

=> Appending nvm source string to /home/ec2-user/.bashrc
=> Appending bash_completion source string to /home/ec2-user/.bashrc
=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This loads nvm bash_completion
[ec2-user@ip-10-0-10-133 bin]$ source ~/.bashrc
[ec2-user@ip-10-0-10-133 bin]$ nvm install 16
Downloading and installing node v16.20.2...
Downloading https://nodejs.org/dist/v16.20.2/node-v16.20.2-linux-x64.tar.xz...
######################################################################################################################################################################## 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v16.20.2 (npm v8.19.4)
Creating default alias: default -> 16 (-> v16.20.2)
[ec2-user@ip-10-0-10-133 bin]$ nvm use 16
Now using node v16.20.2 (npm v8.19.4)
[ec2-user@ip-10-0-10-133 bin]$
```

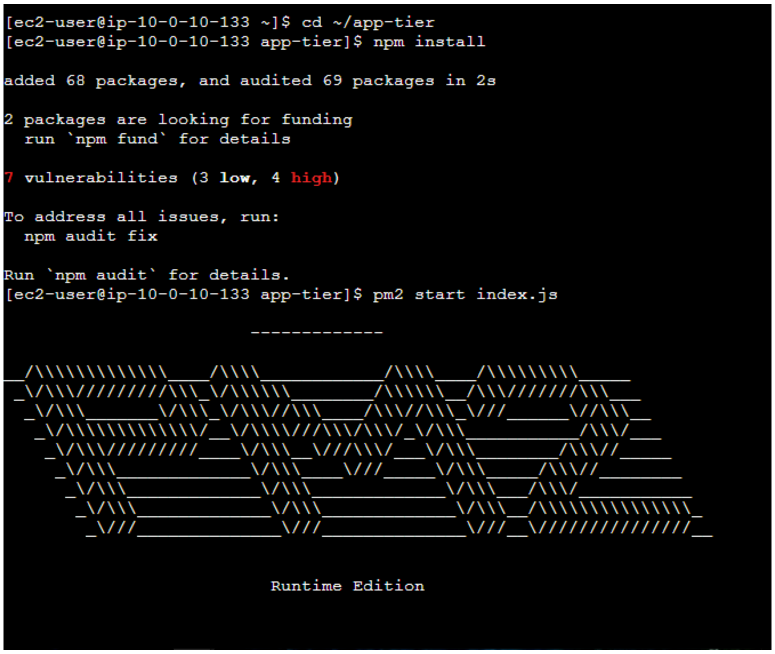
PM2 is a daemon process manager that will keep our node.js app running when we exit the instance or if it is rebooted. Install that as well.
npm install -g pm2



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



Navigate to the app directory, install dependencies, and start the app with pm2.
cd ~/app-tier
npm install
pm2 start index.js

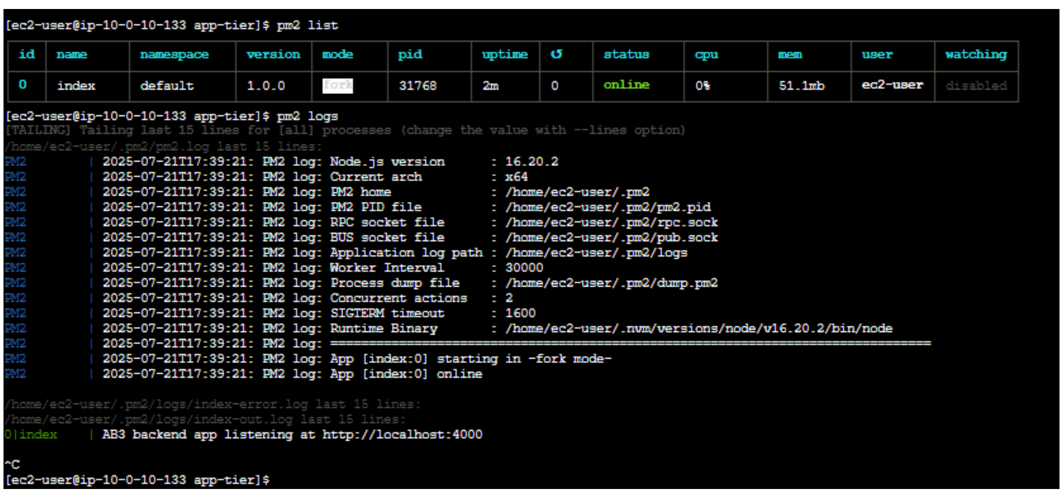

[ec2-user@ip-10-0-10-133 ~]$ cd ~/app-tier
[ec2-user@ip-10-0-10-133 app-tier]$ npm install
added 68 packages, and audited 69 packages in 2s
2 packages are looking for funding
run `npm fund` for details
7 vulnerabilities (3 low, 4 high)
To address all issues, run:
npm audit fix
Run `npm audit` for details.
[ec2-user@ip-10-0-10-133 app-tier]$ pm2 start index.js
Runtime Edition

To make sure the app is running correctly run the following:

pm2 list

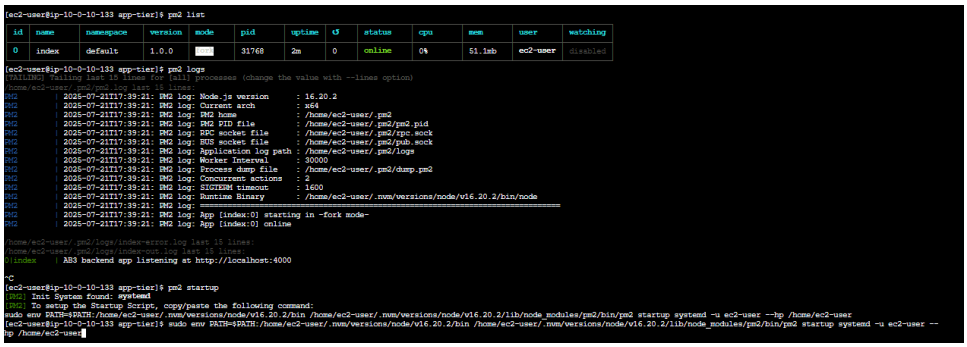
If you see a status of online, the app is running. If you see errored, then you need to do some troubleshooting. To look at the latest errors, use this command:

pm2 logs


[ec2-user@ip-10-0-10-133 app-tier]$ pm2 list
id | name | namespace | version | mode | pid | uptime | ↺ | status | cpu | mem | user | watching
0 | index | default | 1.0.0 | fork | 31768 | 2m | 0 | online | 0% | 51.1mb | ec2-user | disabled
[ec2-user@ip-10-0-10-133 app-tier]$ pm2 logs
[TAILING] Tailing last 15 lines for [all] processes (change the value with --lines option)
/home/ec2-user/.pm2/pm2.log last 15 lines:
PM2 | 2025-07-21T17:39:21: PM2 log: Node.js version : 16.20.2
PM2 | 2025-07-21T17:39:21: PM2 log: Current arch : x64
PM2 | 2025-07-21T17:39:21: PM2 log: PM2 home : /home/ec2-user/.pm2
PM2 | 2025-07-21T17:39:21: PM2 log: PM2 PID file : /home/ec2-user/.pm2/pm2.pid
PM2 | 2025-07-21T17:39:21: PM2 log: RPC socket file : /home/ec2-user/.pm2/rpc.sock
PM2 | 2025-07-21T17:39:21: PM2 log: BUS socket file : /home/ec2-user/.pm2/pub.sock
PM2 | 2025-07-21T17:39:21: PM2 log: Application log path : /home/ec2-user/.pm2/logs
PM2 | 2025-07-21T17:39:21: PM2 log: Worker Interval : 30000
PM2 | 2025-07-21T17:39:21: PM2 log: Process dump file : /home/ec2-user/.pm2/dump.pm2
PM2 | 2025-07-21T17:39:21: PM2 log: Concurrent actions : 2
PM2 | 2025-07-21T17:39:21: PM2 log: SIGTERM timeout : 1600
PM2 | 2025-07-21T17:39:21: PM2 log: Runtime Binary : /home/ec2-user/.nvm/versions/node/v16.20.2/bin/node
PM2 | 2025-07-21T17:39:21: PM2 log: ===
PM2 | 2025-07-21T17:39:21: PM2 log: App [index:0] starting in -fork mode-
PM2 | 2025-07-21T17:39:21: PM2 log: App [index:0] online
/home/ec2-user/.pm2/logs/index-error.log last 15 lines:
/home/ec2-user/.pm2/logs/index-out.log last 15 lines:
0|index | AB3 backend app listening at http://localhost:4000
^C
[ec2-user@ip-10-0-10-133 app-tier]$

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



After running this you will see a message similar to this.
[PM2] To setup the Startup Script, copy/paste the following command: sudo env PATH=$PATH:/home/ec2-user/.nvm/versions/node/v16.0.0/bin /home/ec2-user/.nvm/versions/node/v16.0.0/lib/node_modules/pm2/bin/pm2 startup systemd -u ec2-user —hp /home/ec2-user

DO NOT run the above command, rather you should copy and past the command in the output you see in your own 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.

1

```
curl http://localhost:4000/health
```

The response should looks like the following:

2

```
"This is the health check"
```

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

3

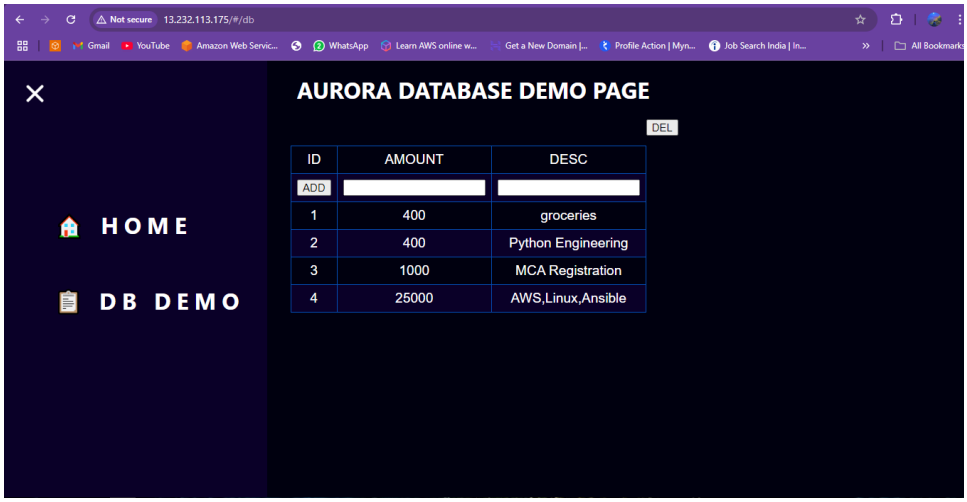
```
curl http://localhost:4000/transaction
```

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

4

```
{"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"}]}
```

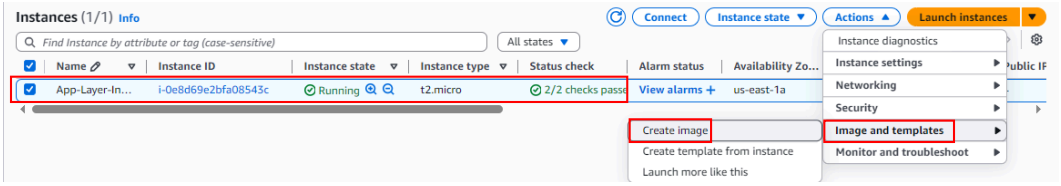
If you see both of these responses, then your networking, security, database and app configurations are correct.



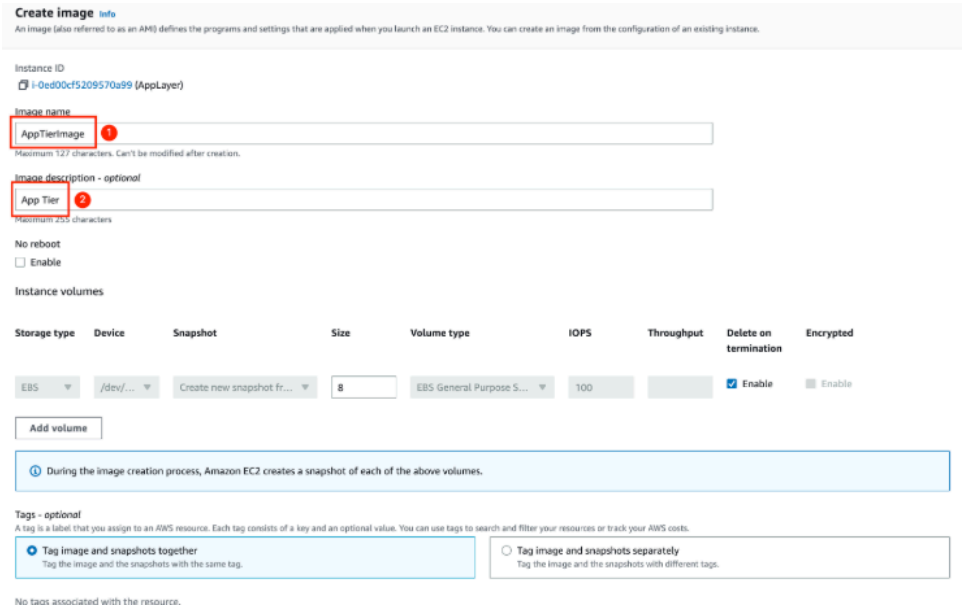
Internal Load Balancing and Auto Scaling

Navigate to Instances on the left hand side of the EC2 dashboard. Select the app tier instance we created and under Actions select Image and templates. Click Create Image.

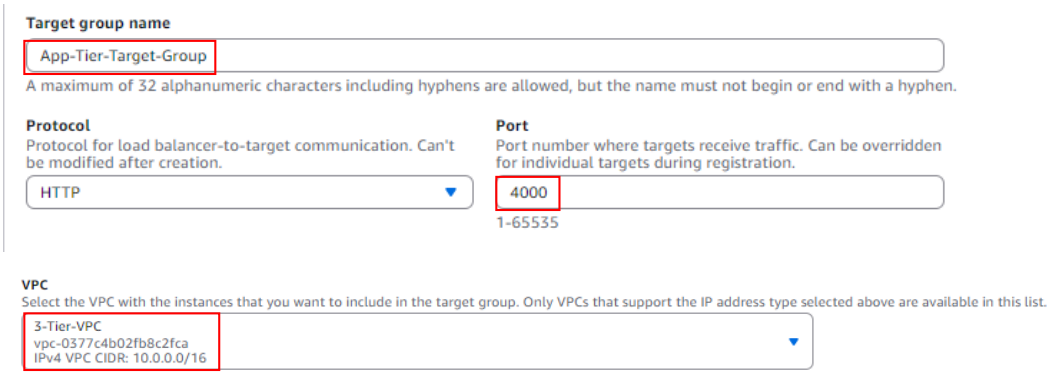
Give it a name App-Tier-image → description (Apptier) → click on create image

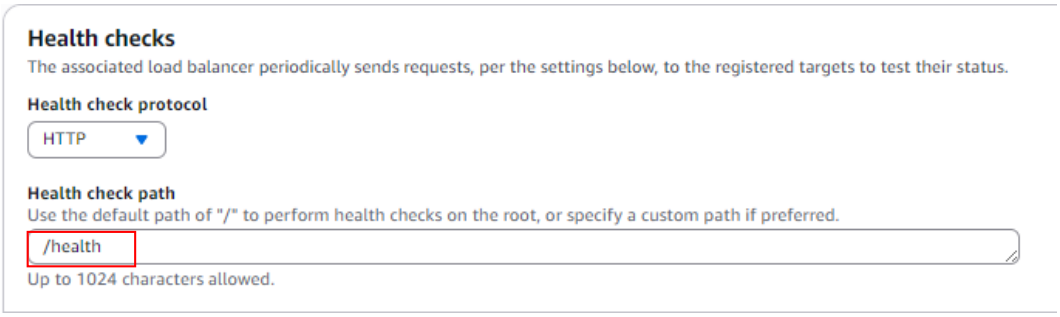


After create image we need Target group

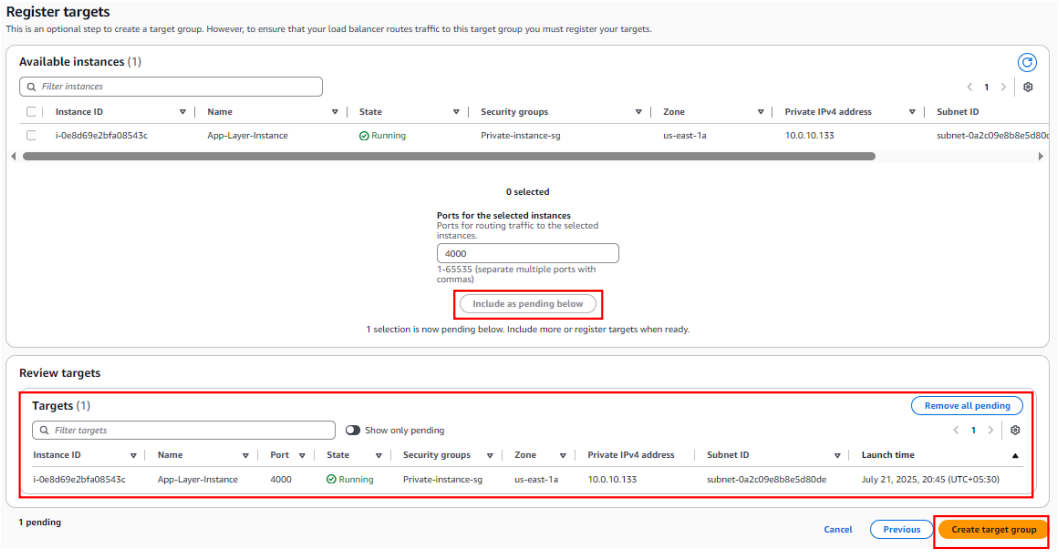


Now we create Target Group → click on create TG → choose to target type → instance → give it a TG name (App Tier target group) → http (4000) → select custom VPC → in the health check path → /health → click on next

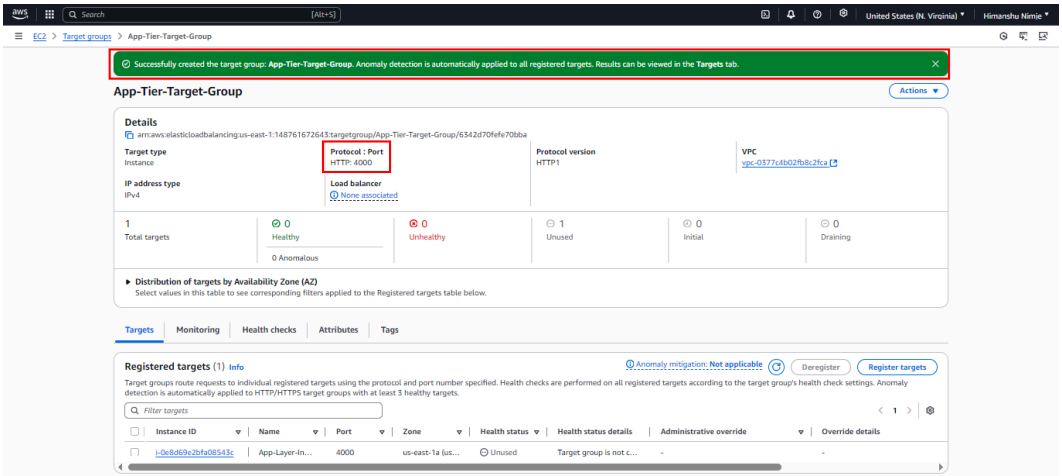




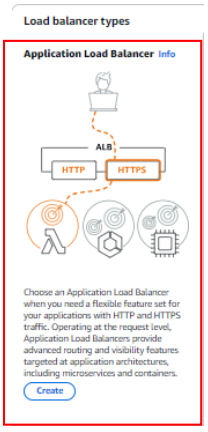
Now register targets → enter the port value is 4000Check mark to target group and include as pending below → click on target group



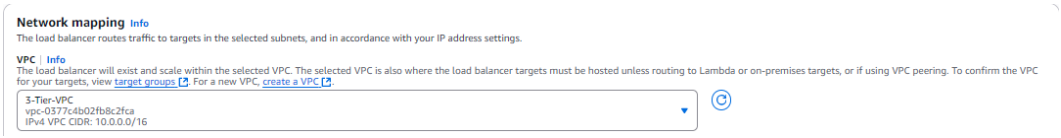
Now Target Group is Created



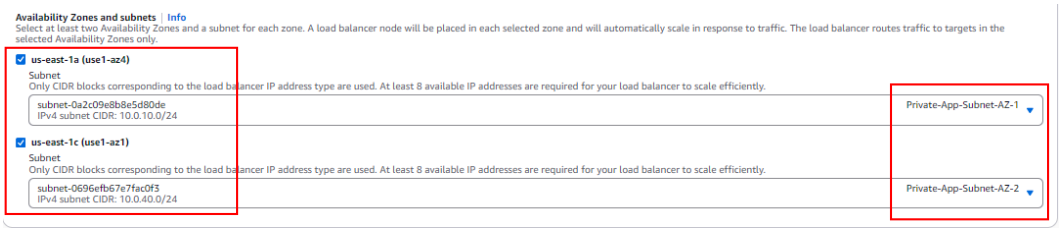
After creating target group → we create Application Load Balancer



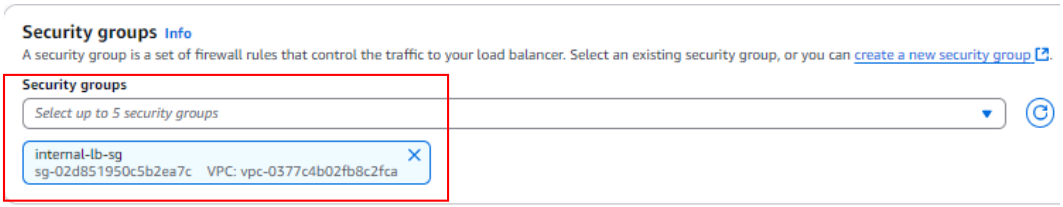
Click on create Load Balancer → Select (ALB) → Give the name (App-tier internal LB) → scheme select (instance) → select custom VPC



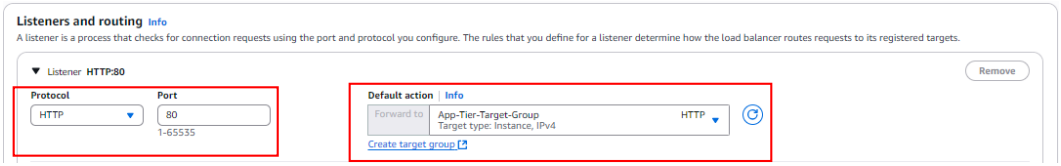
Select subnet and AZ (select Private App subnet AZ1 and AZ2) →



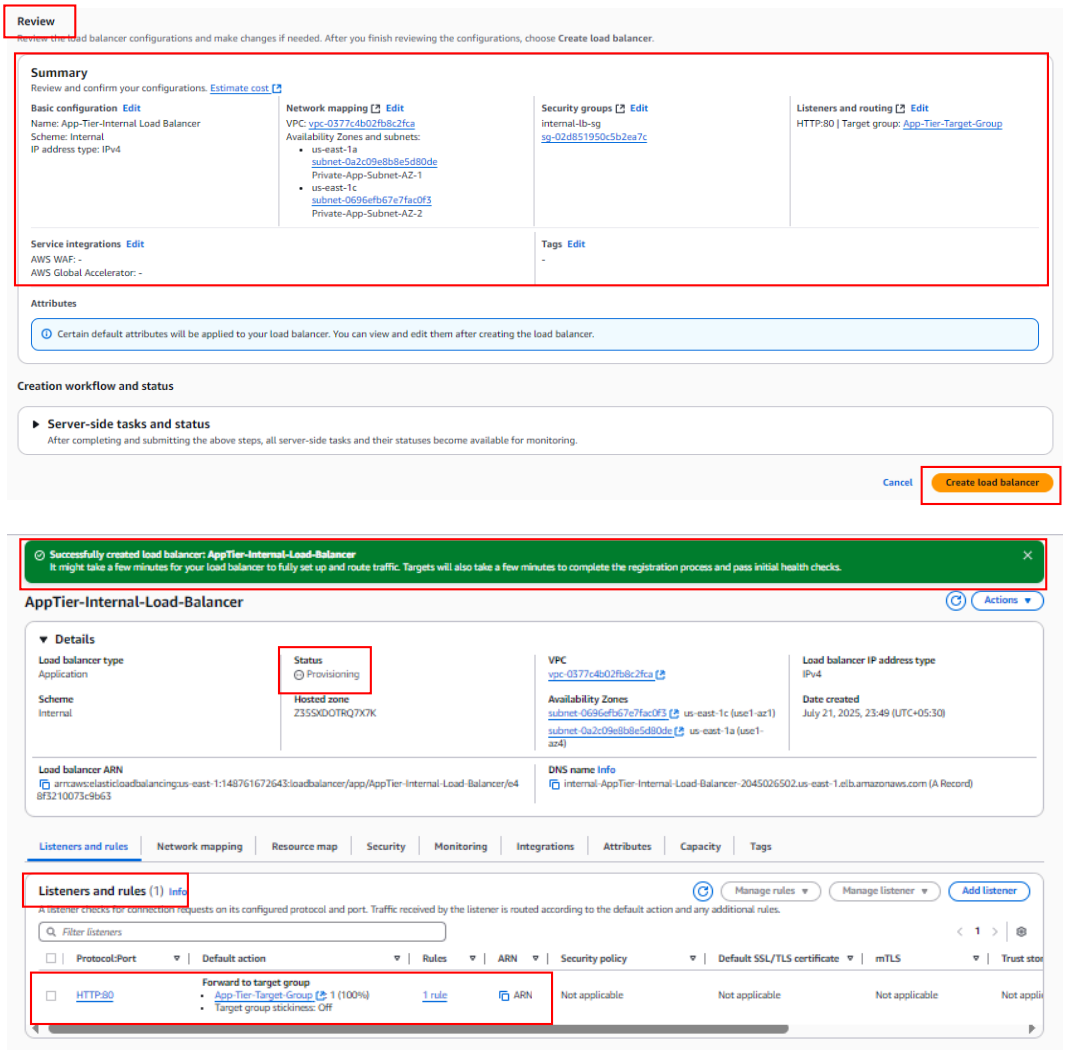
→ In security Group (internal-lb-sg)



Here Listener and Routing → select protocol http 80 → add (App-Tier-TG)

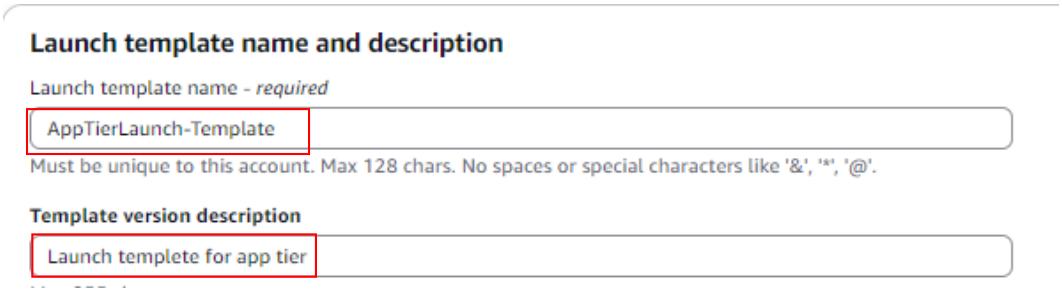


Review and Create Load Balancer ↴

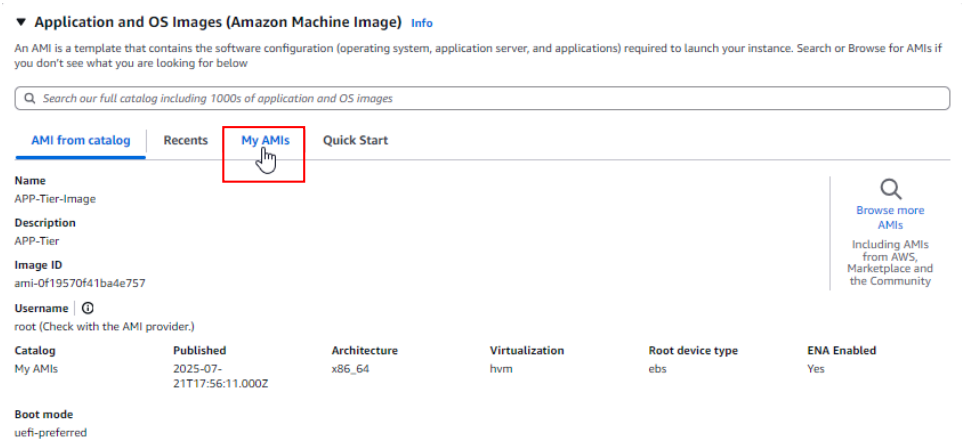


Before we configure Auto Scaling, we need to create a Launch template with the AMI we created earlier. On the left side of the EC2 dashboard navigate to Launch Template under Instances and click Create Launch Template.

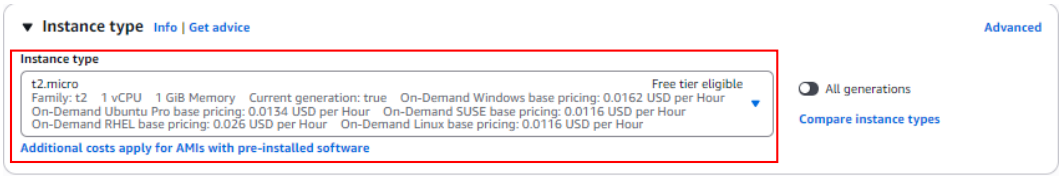
Click on create launch → give the name → (app tier launch template) →



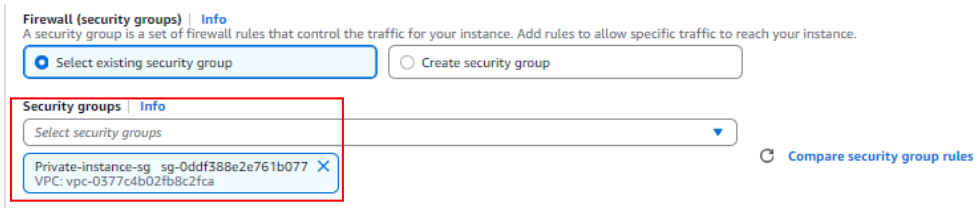
Select Custom AMI (click on my ami) → select (App-tier Image)



Select instance type (t2micro)
Processed without keypair
Don't include subnet

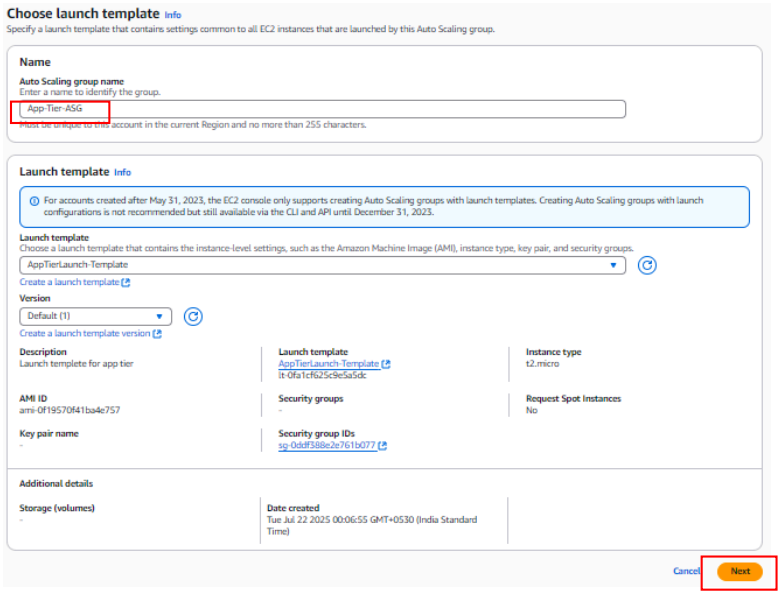


Select Security Group (Private-instance-SG)

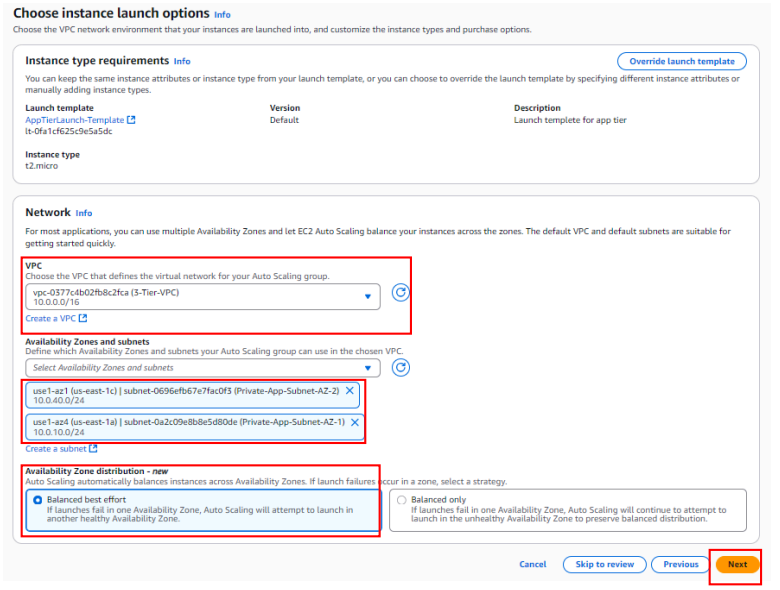


Click on Advanced Details
Click on IAM instance profile (select ec2-s3 role we created in the beginning)
Click on launch template

We will now create the Auto Scaling Group for our app instances.
Click on create auto scaling group → give the name (App-tier-ASG)
Select template (App-tier-template)

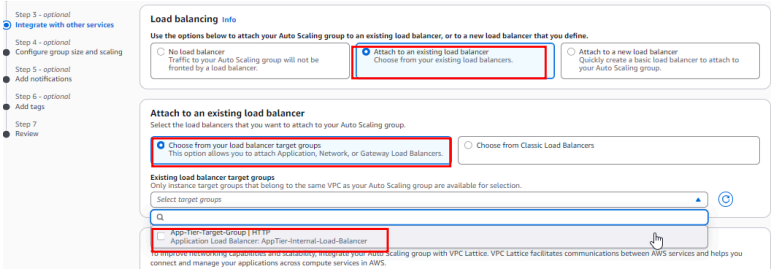


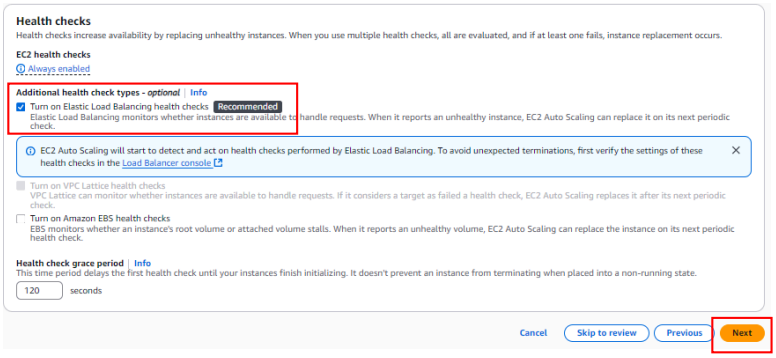
Select custom vpc → select AZ and Subnet (Private app subnet AZ1 and AZ2)



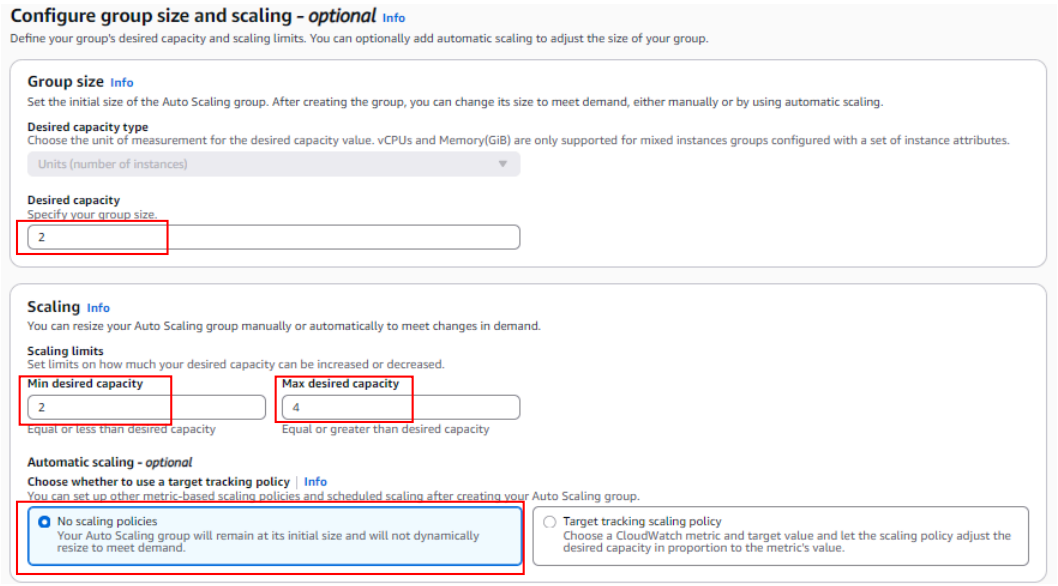
Click on load balancing → selected (Attach to an existing load balancer)

Select (App-Tier-Target Group)



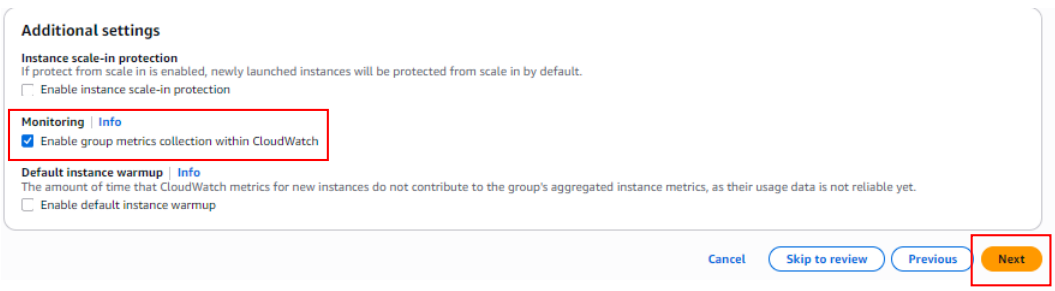


Select the group size and scaling
Desired Capacity 2
Minimum capacity 2
Maximum Capacity 4
Select (no scaling policies)



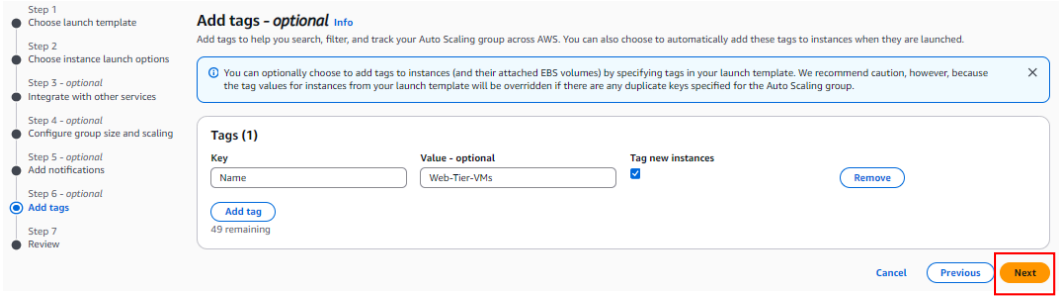
In additional setting

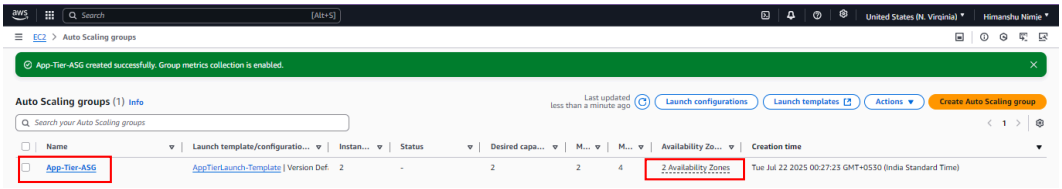
Enable Monitoring



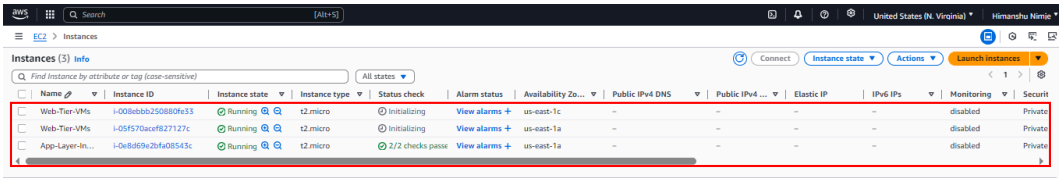
Add Tags

Review and Create Auto Scaling Group





Here is 2 instances automatically created



Web Tier Instance Deployment

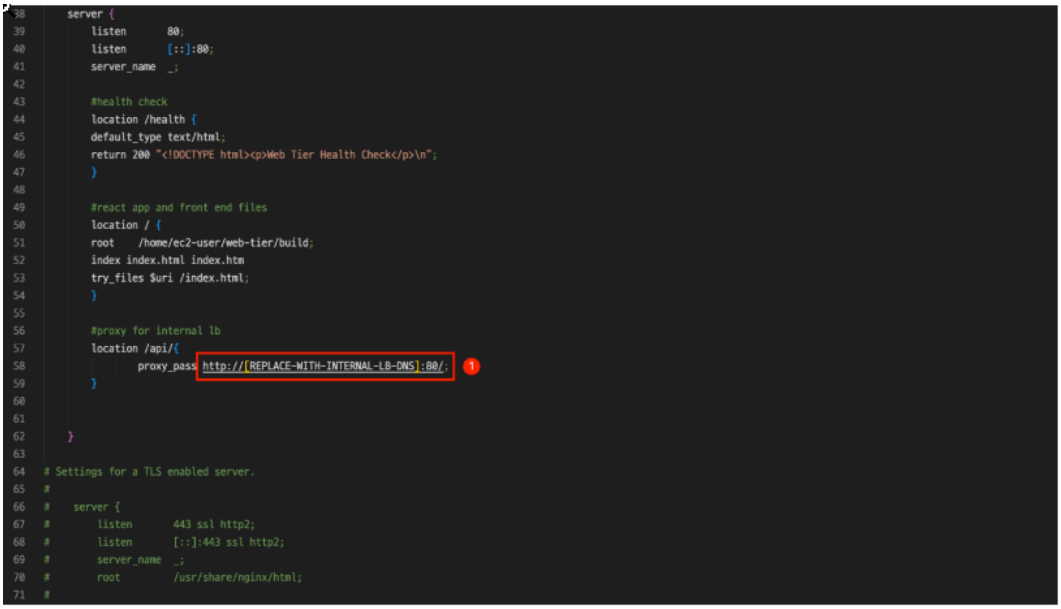
We will deploy an EC2 instance for the web tier and make all necessary software configurations for the NGINX web server and React.js website.

Update NGINX Configuration Files

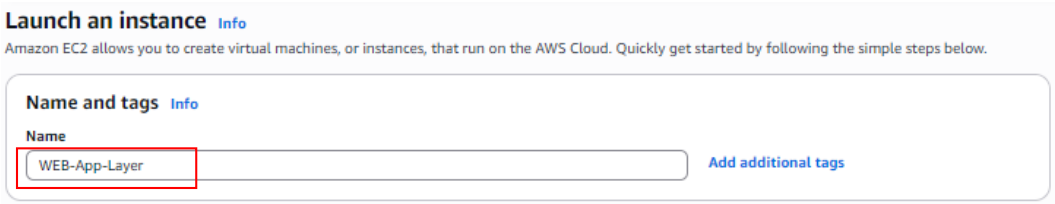
Update Config File

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.

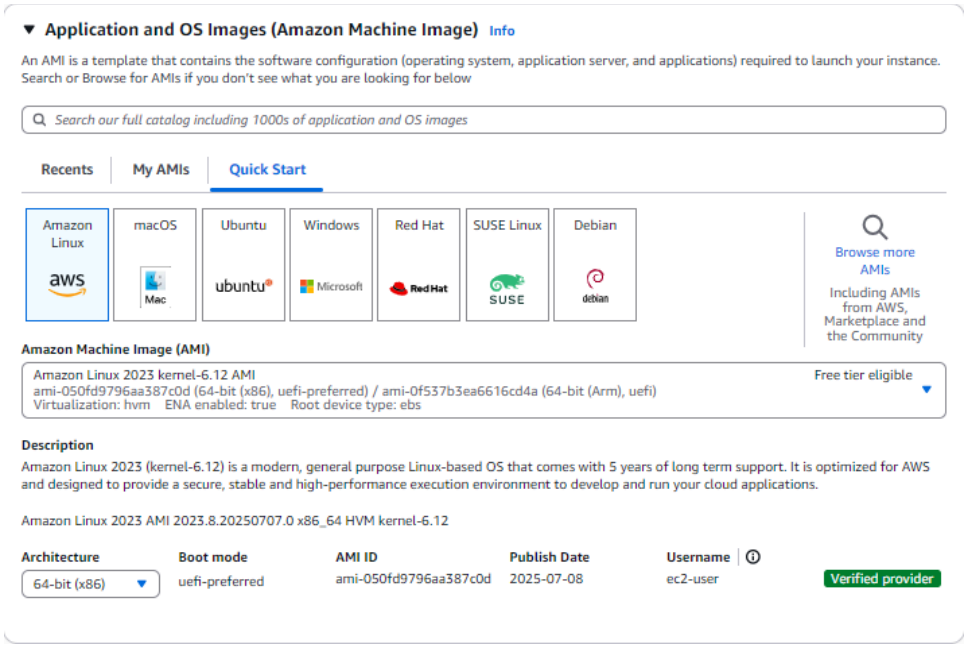
Then, upload this file and the application-code/web-tier folder to the s3 bucket you created for this lab.



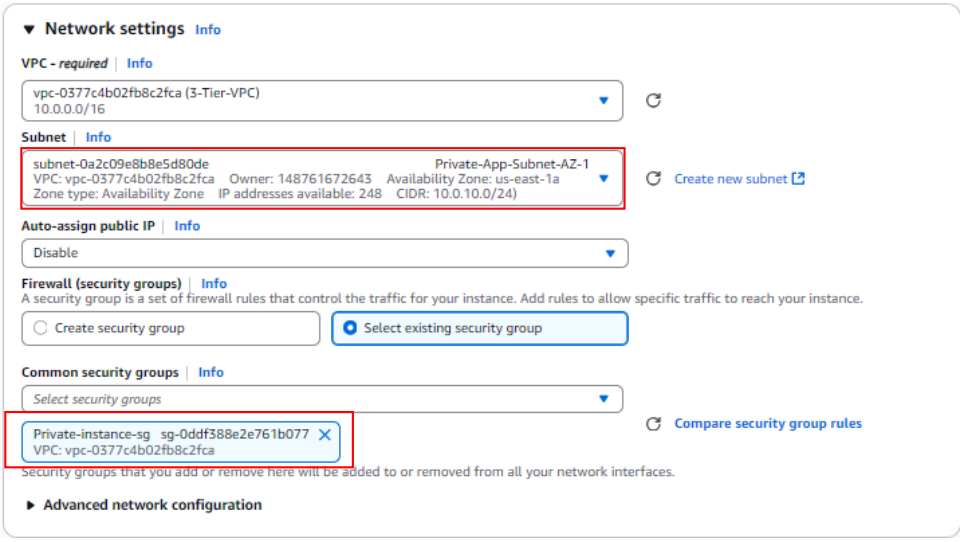
Launch New Instance the Name is Web-App-Layer



Select Amazon Machine

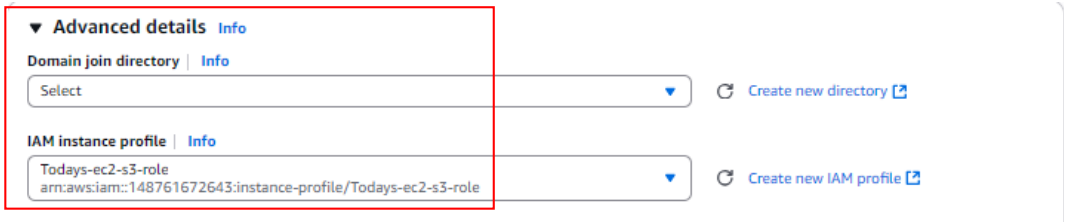


Select custom VPC
Select subnet (Private-App-Subnet-AZ1)
Select Security Groups (Private-Instance SG)

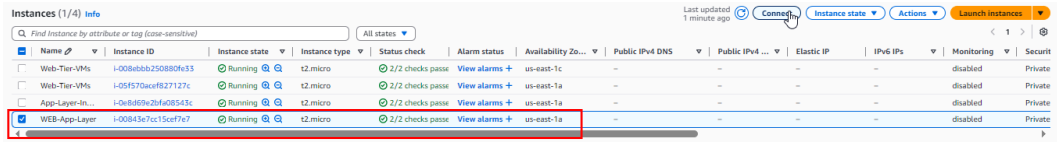


Click on Advanced Details (in the IAM Profile Select ec2-s3-role)

Launch Instance



After Launch instance connect to Session Manager



Enter command

Sudo -su ec2-user

Configure Web Instance

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

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

2. Now we need to download our web tier code from our s3 bucket:

```
cd ~/
aws s3 cp s3://BUCKET_NAME/web-tier/ web-tier --recursive
```

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

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

3. 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 amazon-linux-extras install nginx1 -y
```

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

```
1 cd /etc/nginx
2 ls
```

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

```
1 sudo rm nginx.conf
2 sudo aws s3 cp s3://BUCKET_NAME/nginx.conf .
```

Then, restart Nginx with the following command:

```
sudo service nginx restart
```

To make sure Nginx has permission to access our files execute this command:

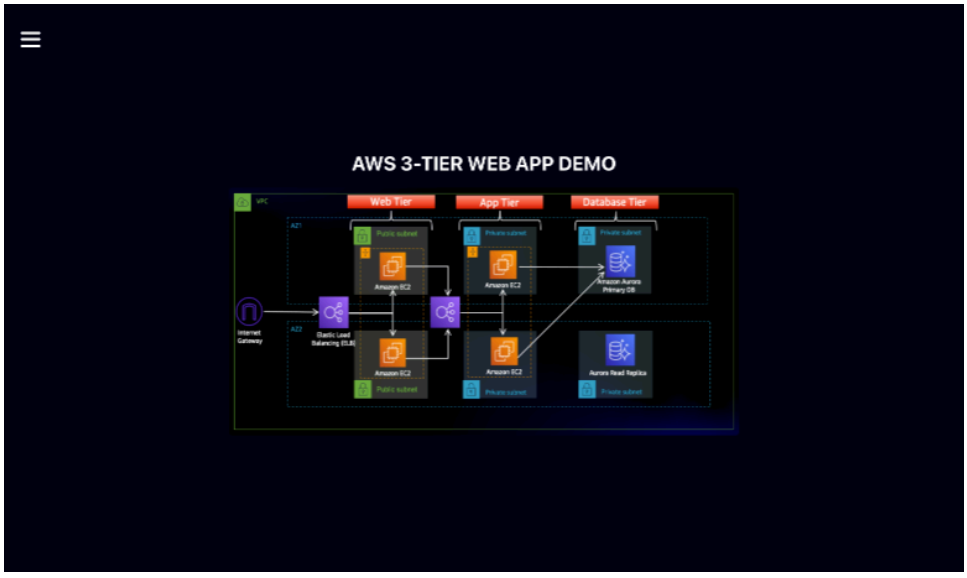
```
chmod -R 755 /home/ec2-user
```

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

```
sudo chkconfig nginx on
```

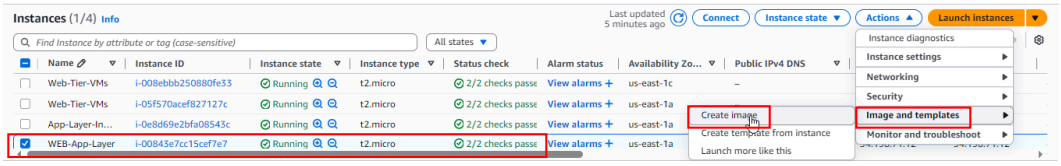
5. Now when you plug in the public IP of your web tier instance, you should see your website, which you can find on the Instance details page on the EC2 dashboard. If you have the database connected and working correctly, then you will also see the database working. You'll be able to add data. Careful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

```
sh-5.2$ sudo -su ec2-user
[ec2-user@ip-10-0-10-184 bin]$ cd ~/web-tier
[ec2-user@ip-10-0-10-184 web-tier]$ npm run build

> aws-3tier-web-layer@0.1.0 build
> react-scripts build

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  74.86 kB  build/static/js/main.9f58f4bc.js
  1.78 kB   build/static/js/453.5846ab7c.chunk.js
  493 B     build/static/css/main.b20b6ac4.css

The project was built assuming it is hosted at ./.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.

Find out more about deployment here:

  https://cra.link/deployment

[ec2-user@ip-10-0-10-184 web-tier]$
```

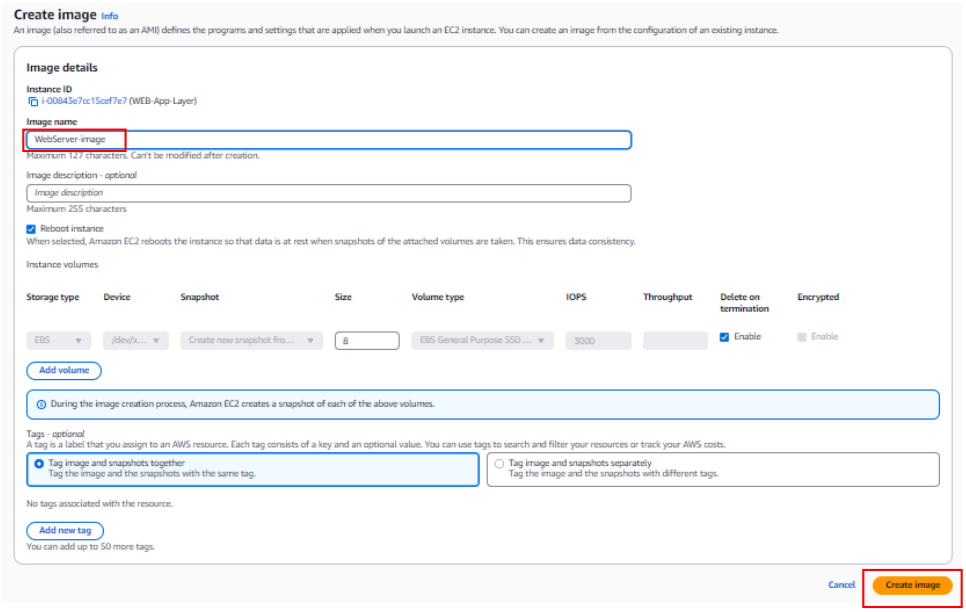


External Load Balancer and Auto Scaling

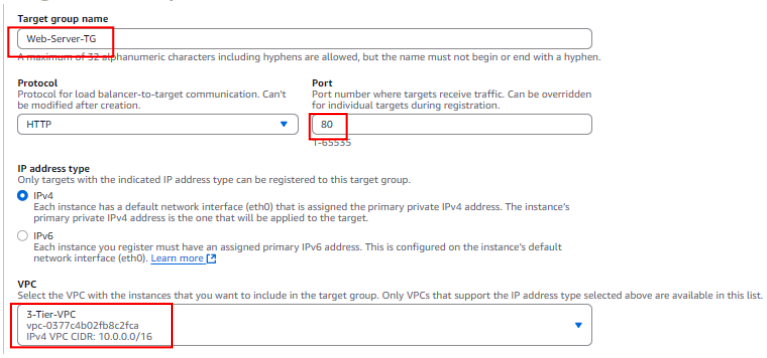
We will create an Amazon Machine Image (AMI) of the web tier instance we just created, and use that to set up autoscaling with an external facing load balancer in order to make this tier highly available.



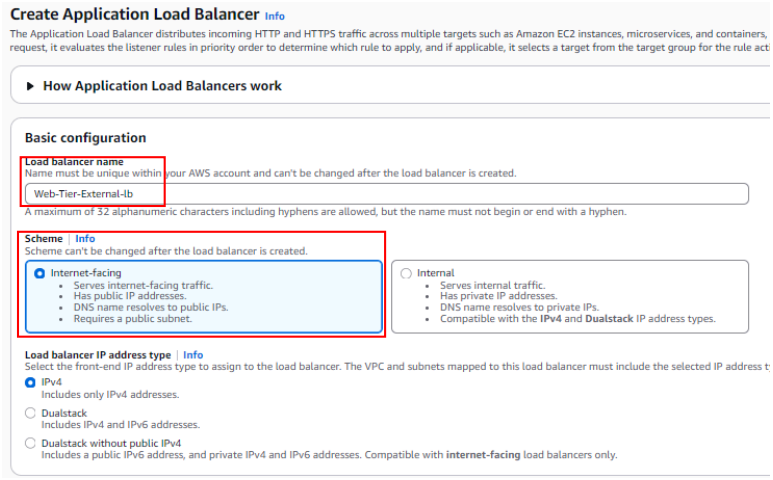
Create Webserver-Image



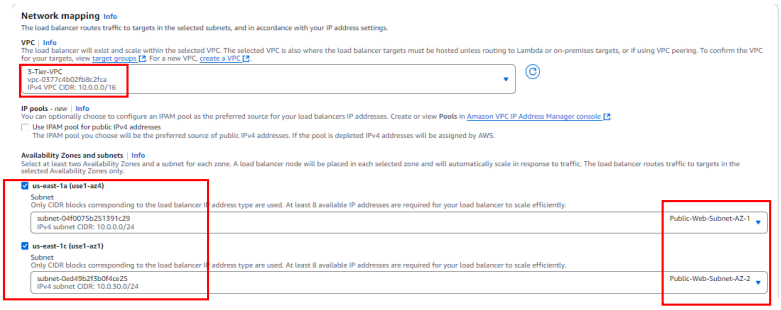
After create Image we need Target Group
Define Ports
Select custom vpc
Click on create Target Group



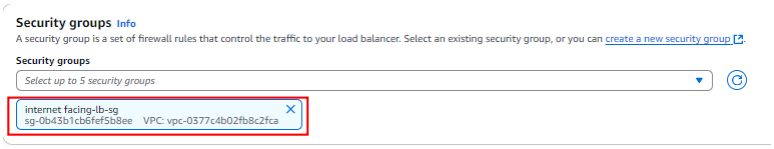
Create Application Load Balancer (Give it a name) Web-Tier-External-lb
Select scheme (Now at this time we select internet-facing)



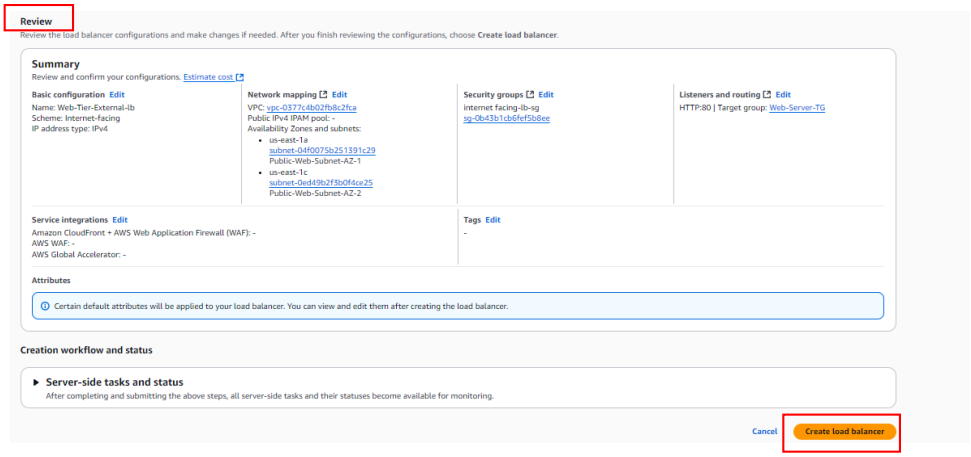
Select custom vpc
Select subnet (public web subnet AZ1 and AZ2)



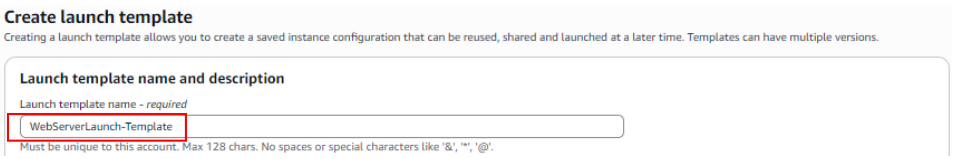
Security Group (internet facing lb-sg)

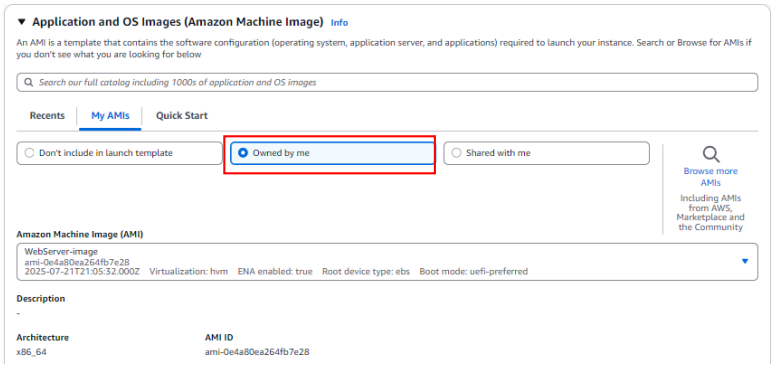


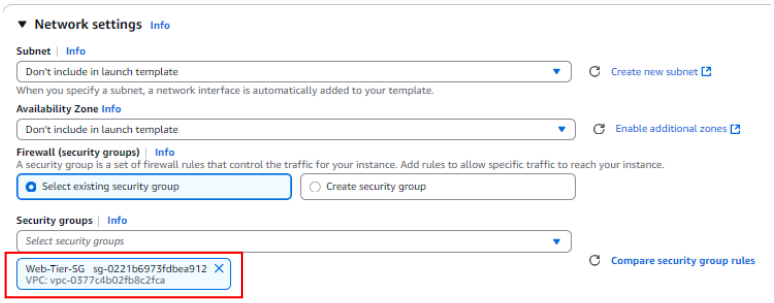
Review and Create load balancer

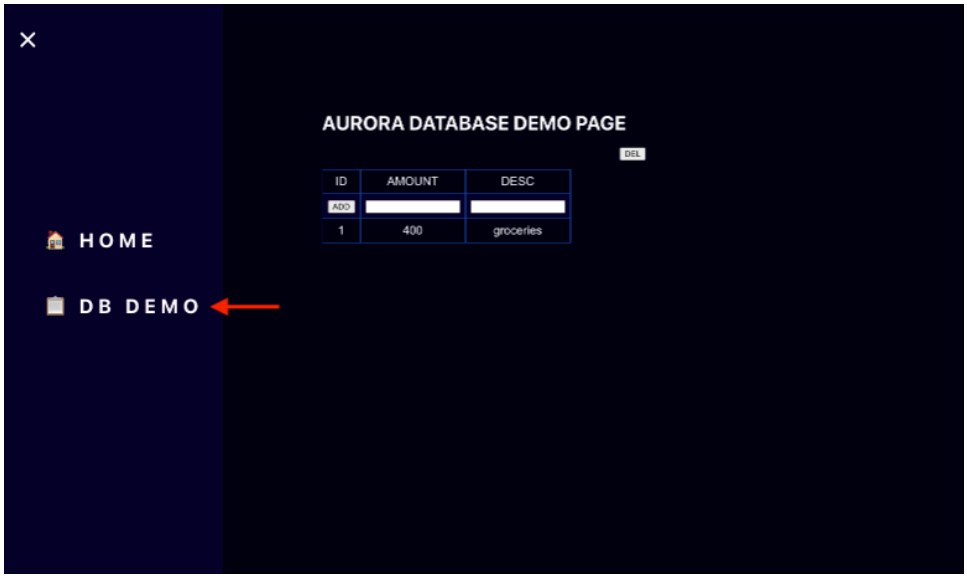


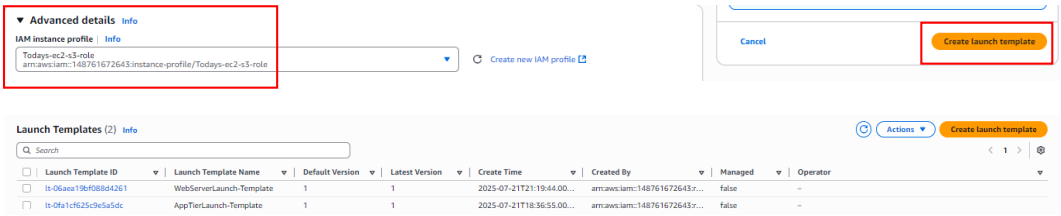
Create template for Auto-scaling Group



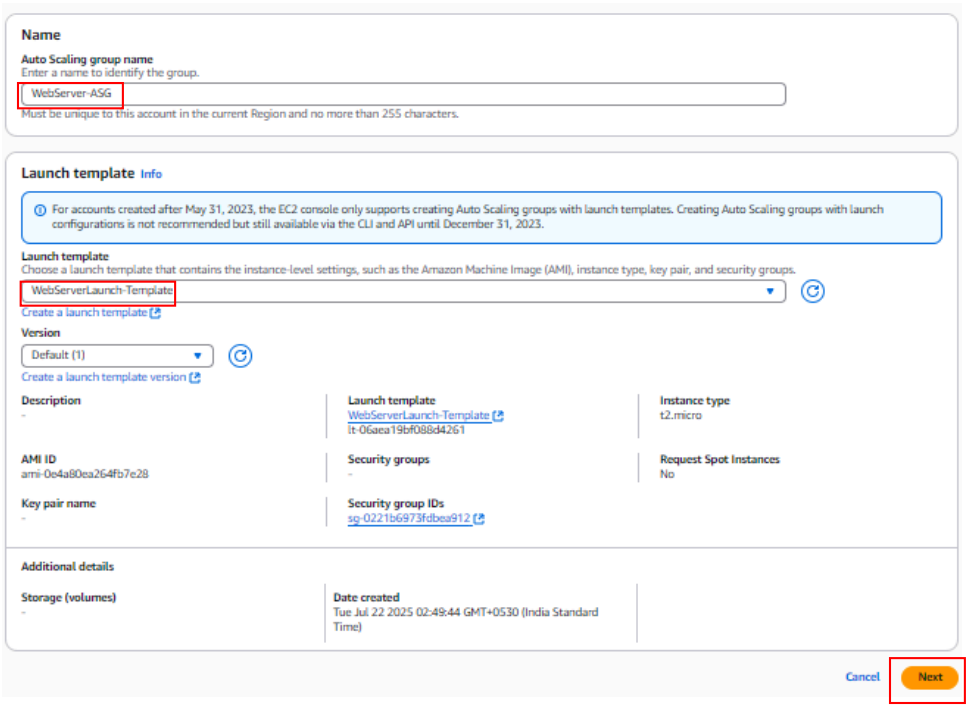






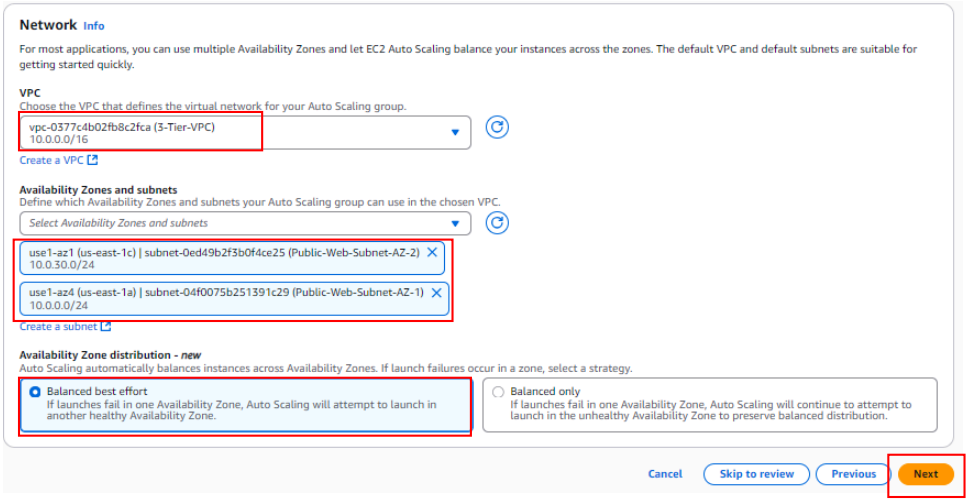


Create Auto-scaling group → give the name → webserver -ASG → select Template (WebserverLaunch-Template)

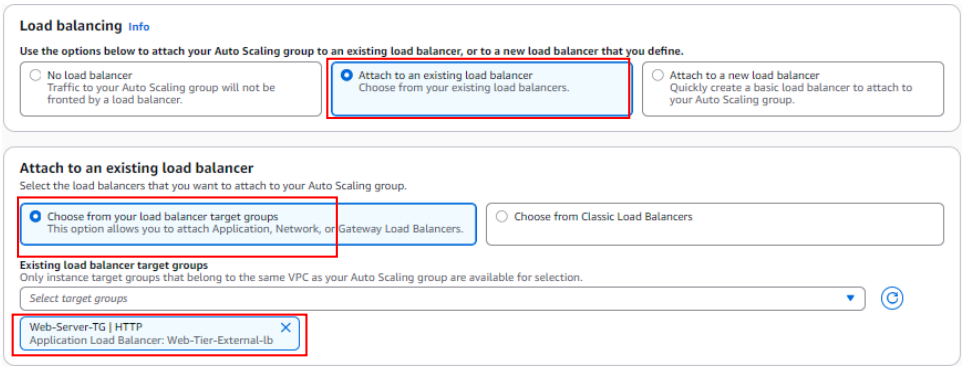


Select VPC

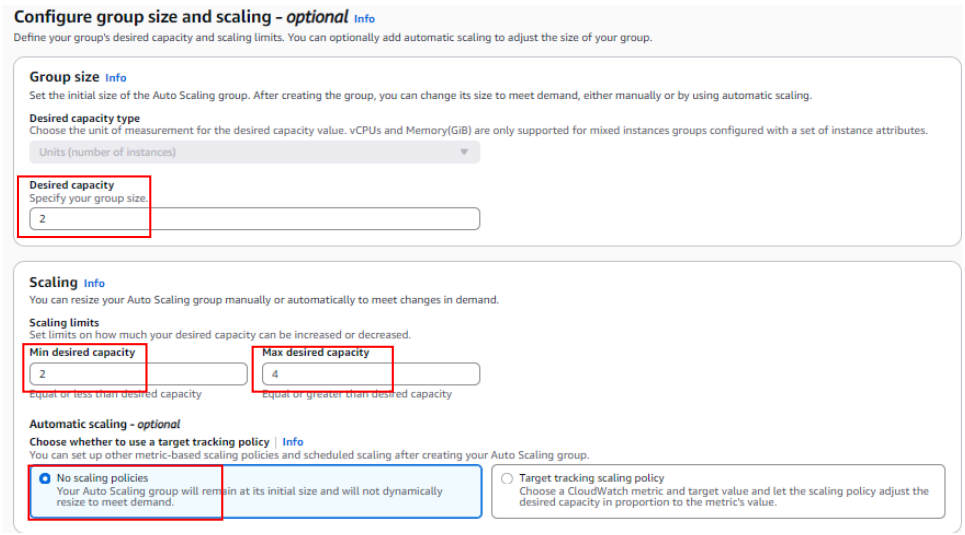
Select AZ and Subnet (public-web-subnet-AZ1 and AZ2)



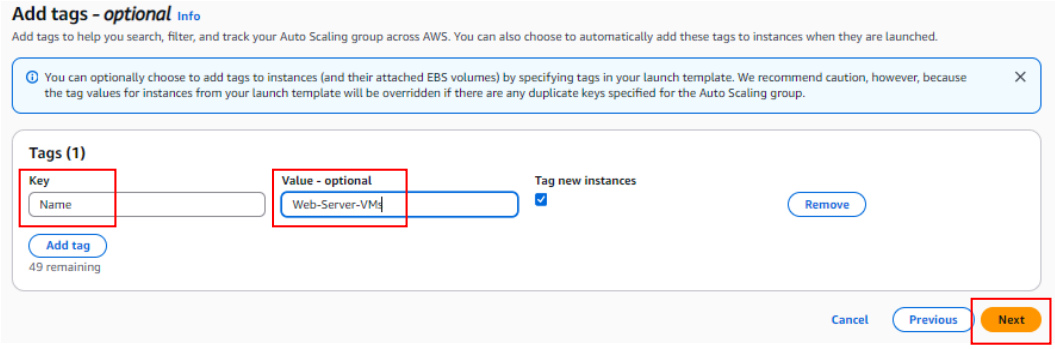
In the load balancing → select existing load balancer that we created (Web-server-TG)



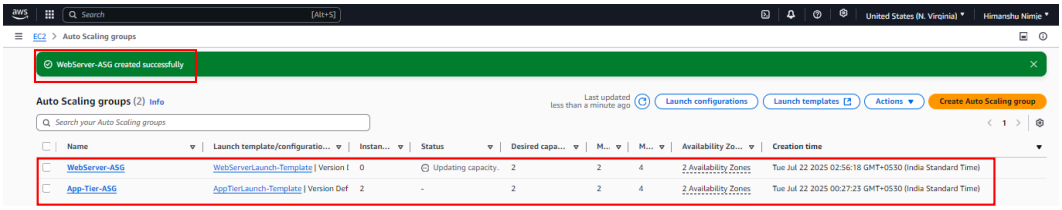
Configure group and scaling size
Desired capacity 2
Minimum Capacity 2
And maximum capacity 4



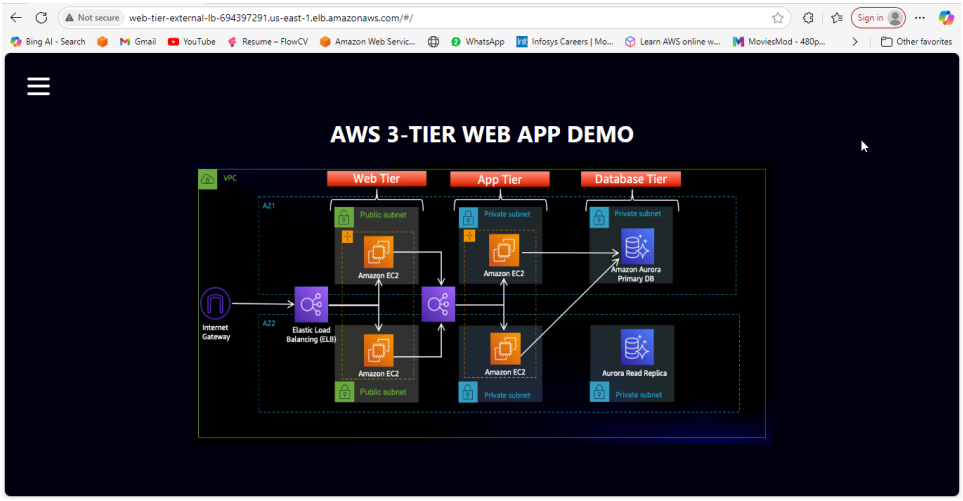
Give the Tag
And create Auto scaling

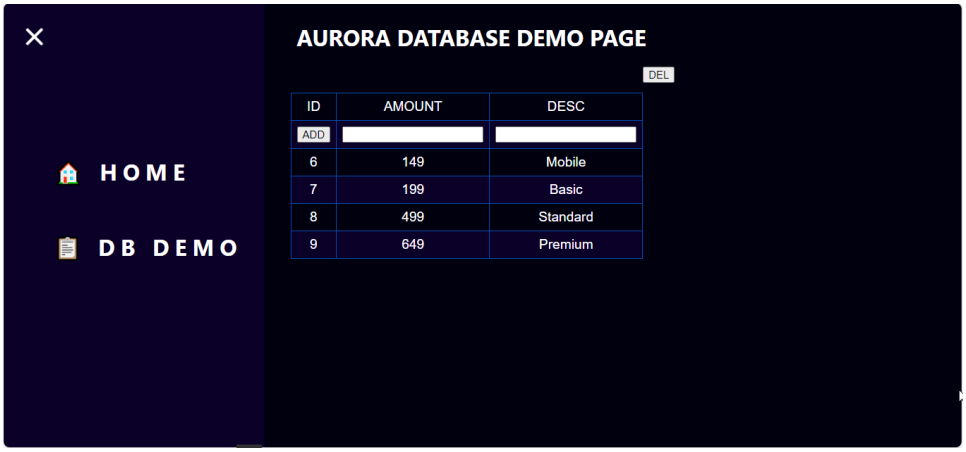


After creating Auto-scaling group
Go to load balancer → there is a Web-server-external-LB → copy the DNS Name and paste it into new browser



You should now have your external load balancer and autoscaling group configured correctly. You should see the autoscaling group spinning up 2 new web tier instances. If you want to test if this is working correctly, you can delete one of your new instances manually and wait to see if a new instance is booted up to replace it. To test if your entire architecture is working, navigate to your external facing loadbalancer, and plug in the DNS name into your browser.





Successfully I've Implemented a 3 Tier Web Architecture!