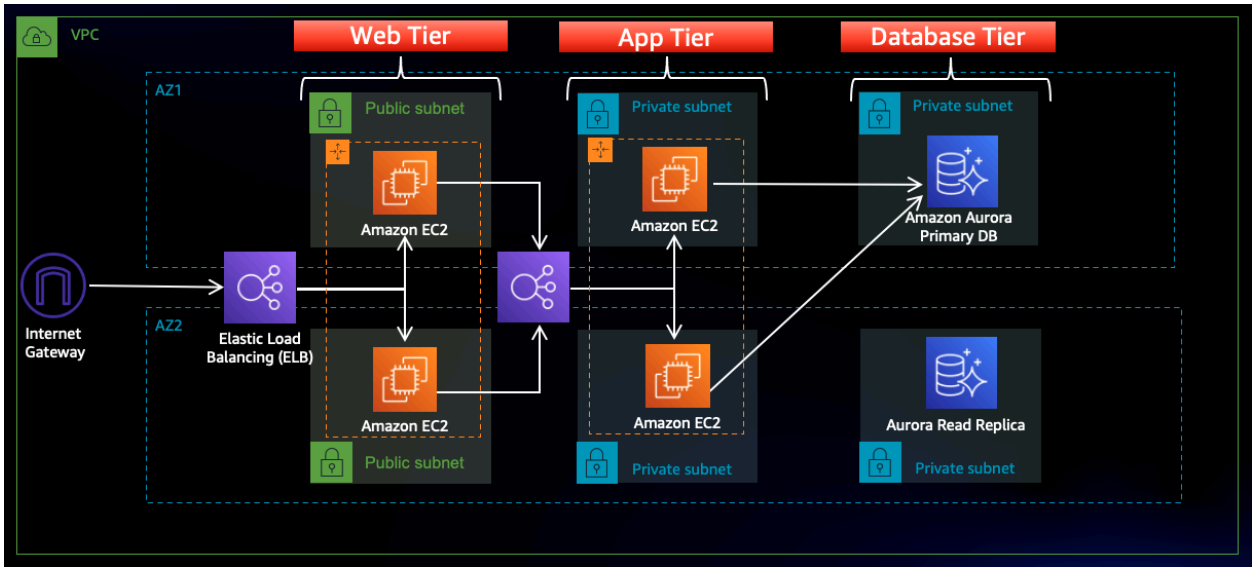
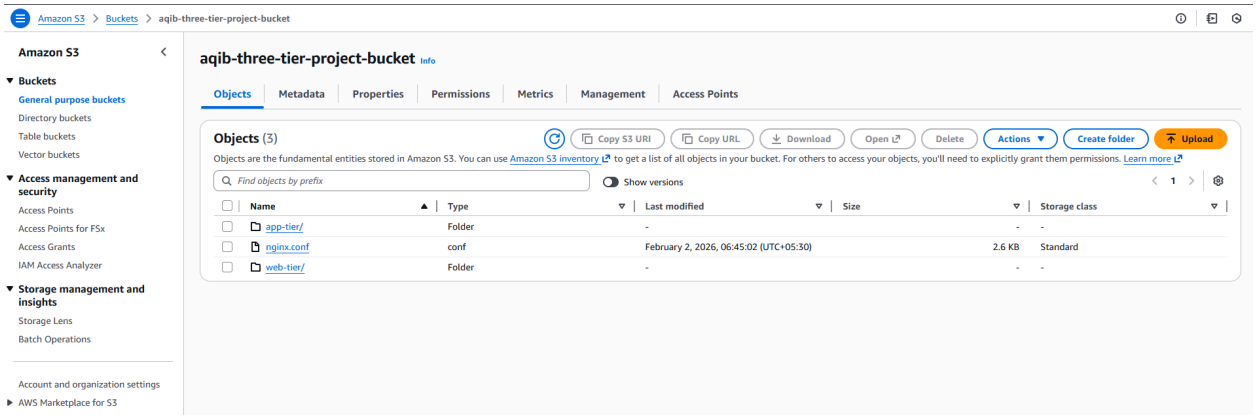
AWS Three-Tier Web Architecture Project

In this architecture, a public-facing Application Load Balancer forwards client traffic to our web tier EC2 instances. The web tier is running Nginx webservers 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.

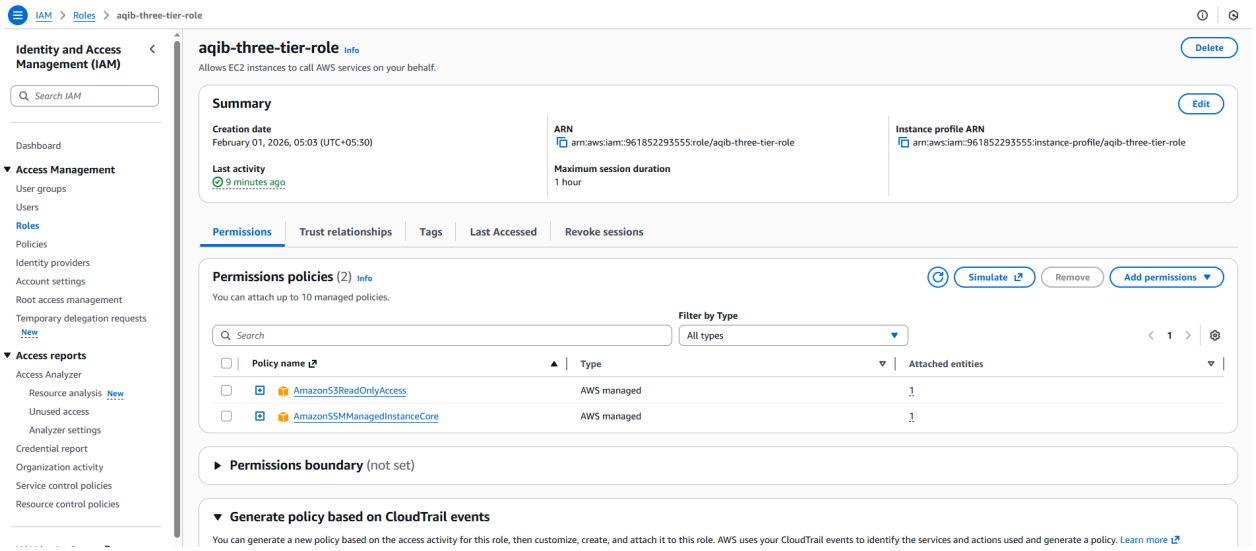


Part 1 Setup

1.1 - S3 Bucket

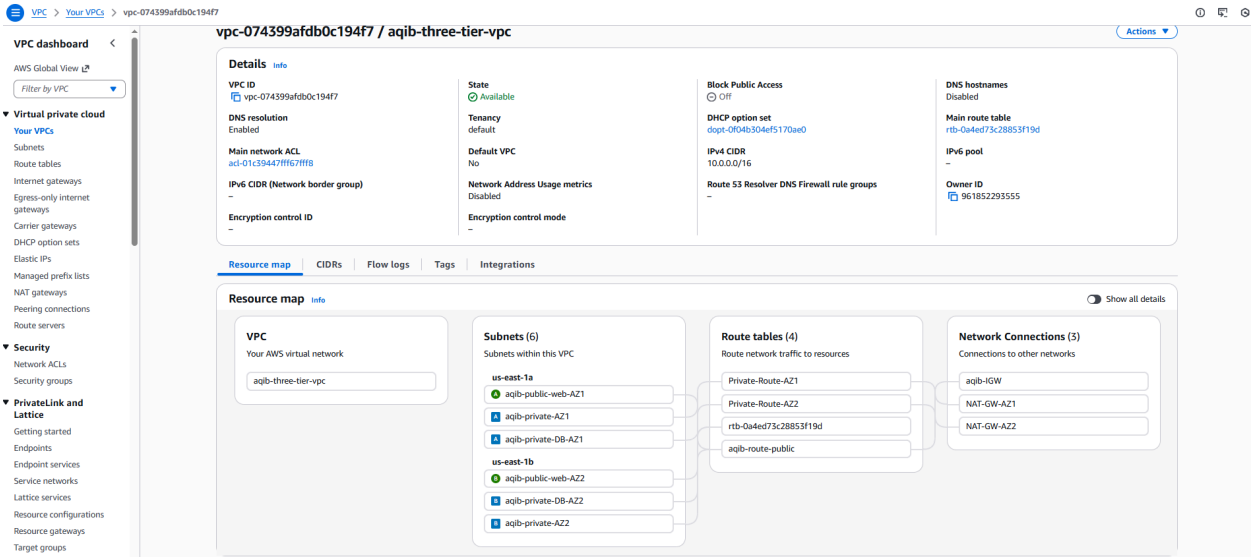


1.2- IAM

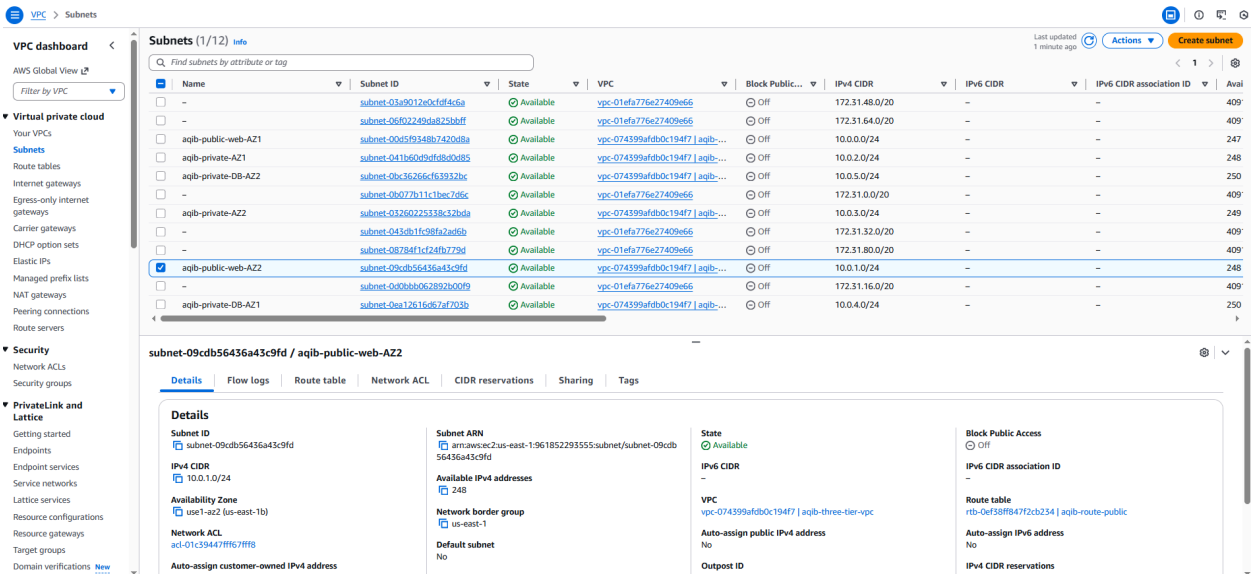


Part 2 - Networking and Security

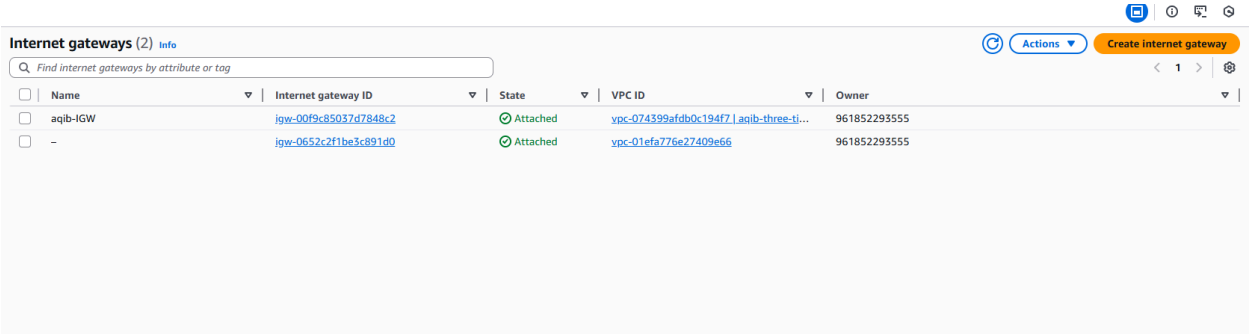
1.1- VPC & Subnets



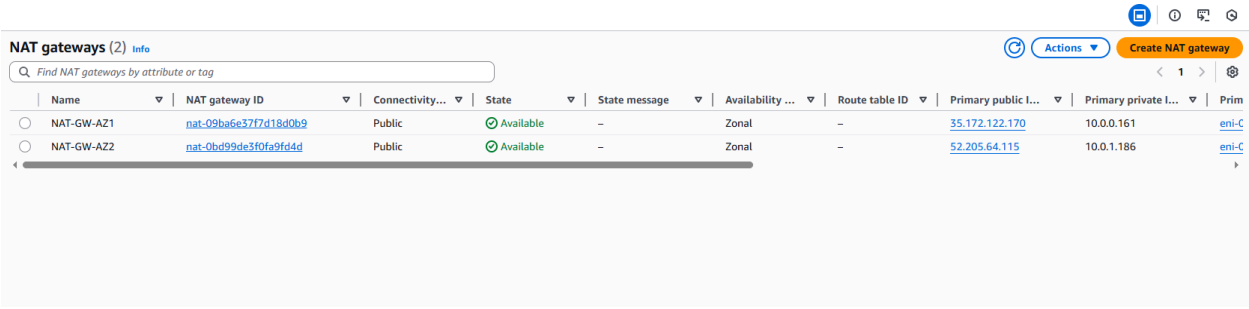
1.2 - Subnets



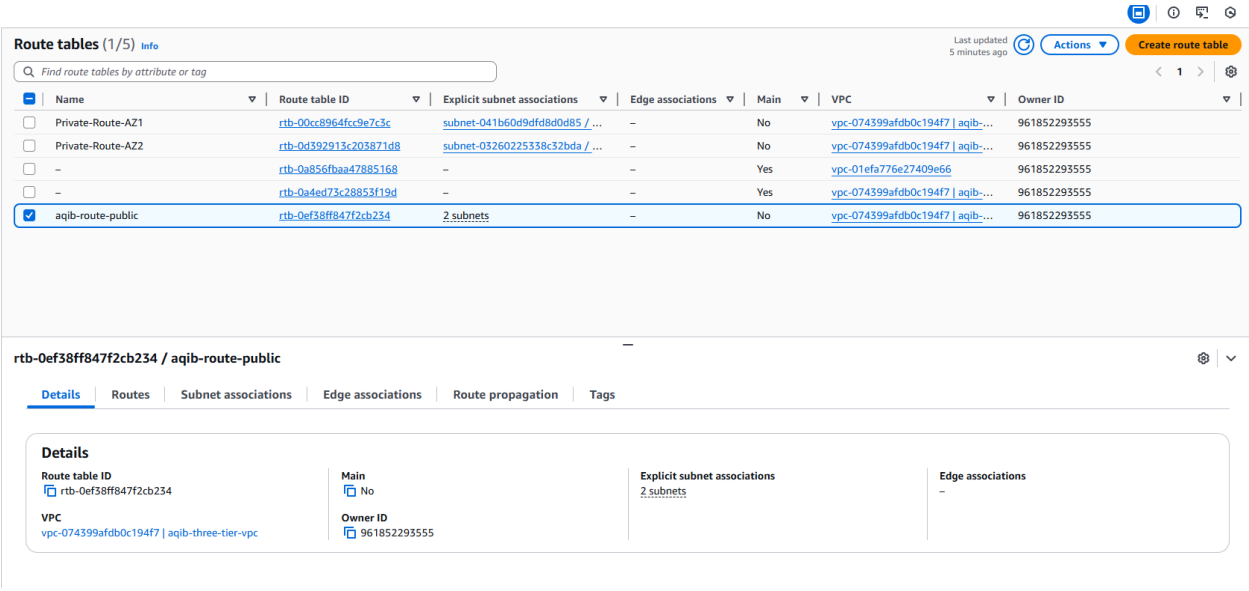
2.1 - IGW & NAT Gateway

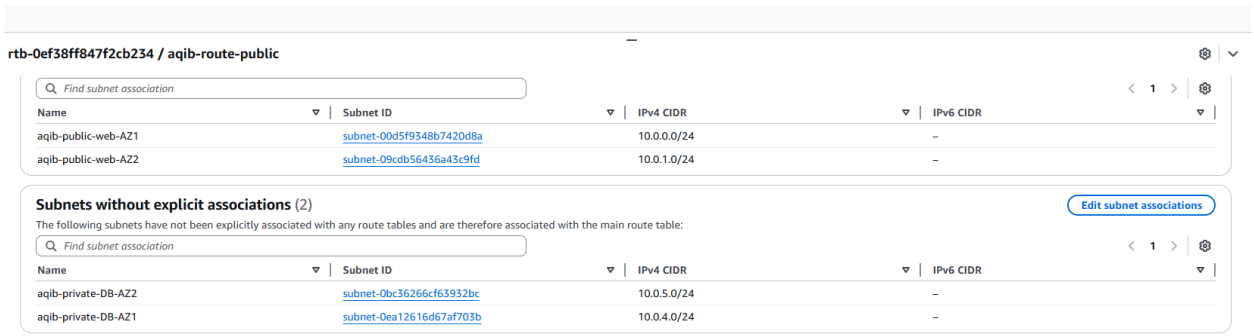


2.2 - NAT

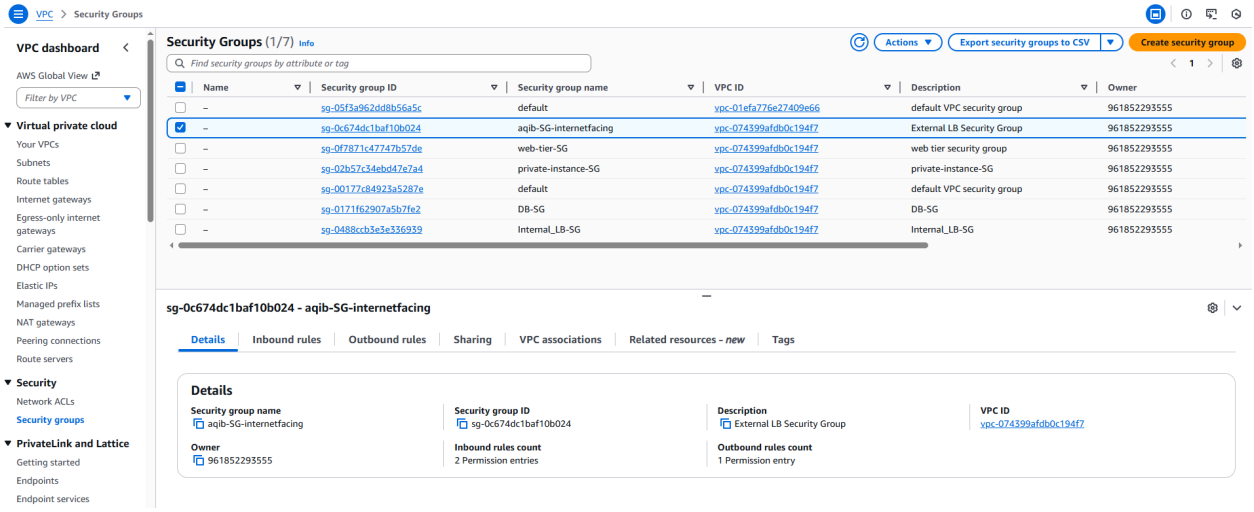


2.3 - Route Tables



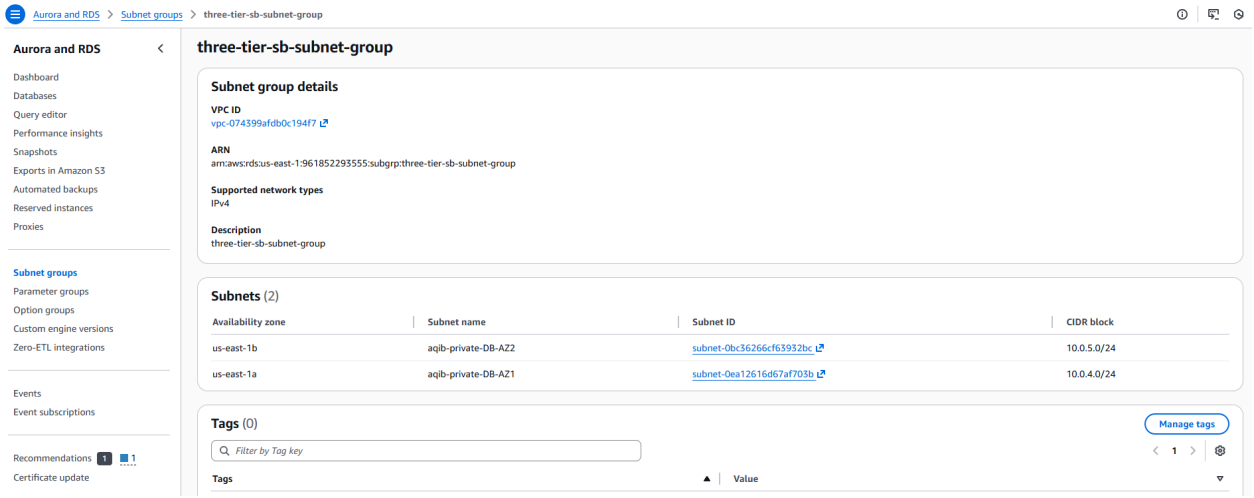


2.4- Security groups

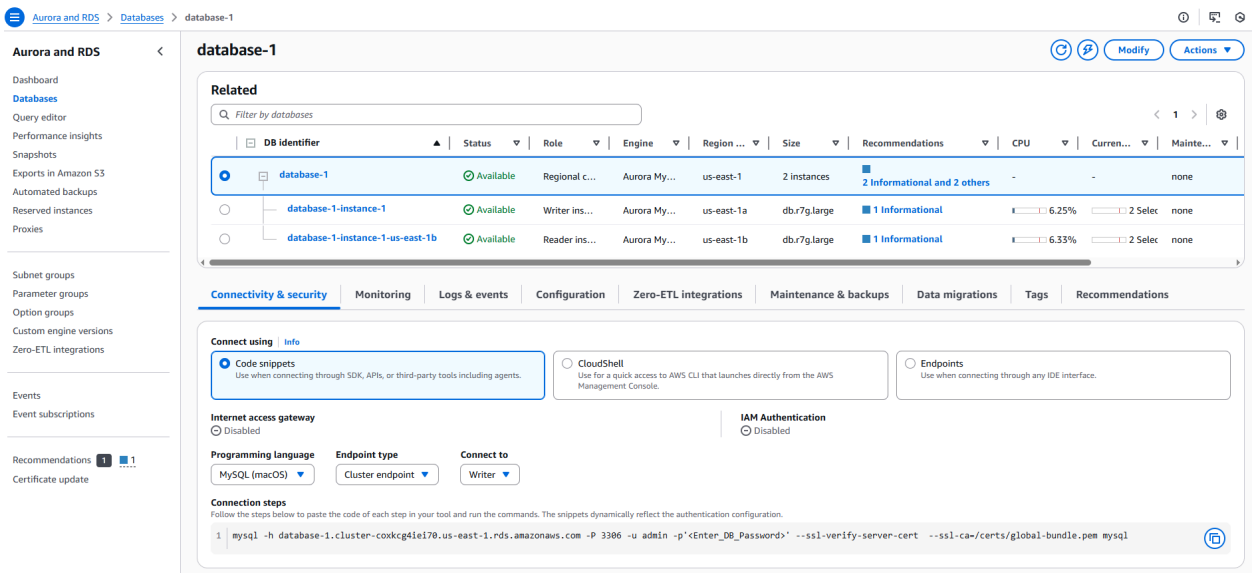


Part 3 - DB Deployment

3.1 -Subnet groups

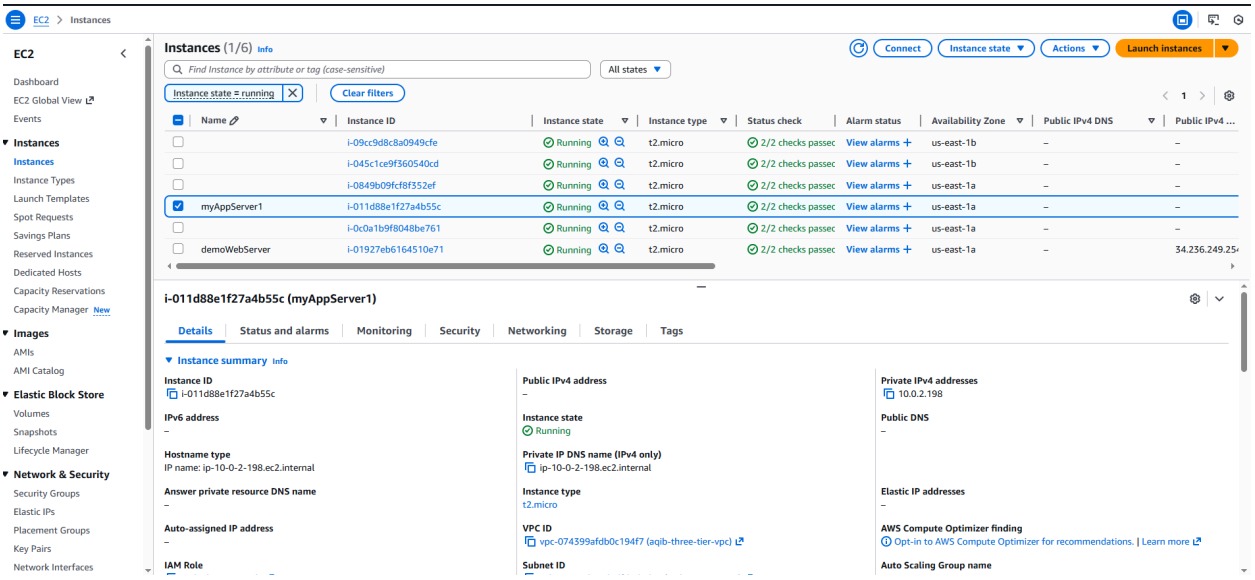


3.2 -DB

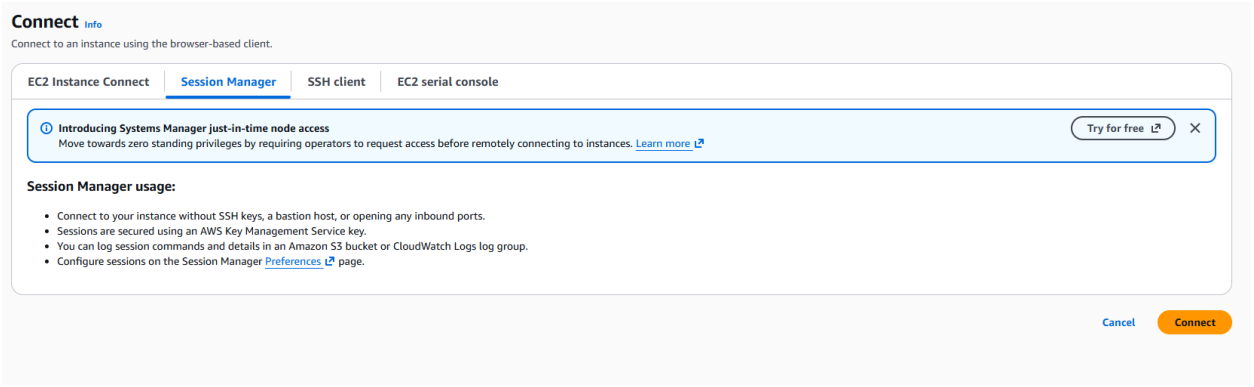


Part 4 - App tier EC2 deployment

4.1 - EC2



Connect to instance



4.2 - Configure DB

1. Start by downloading the MySQL CLI:

   Run

   ```
   sudo wget
   https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm
   ```

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

sudo yum install
https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm

sudo yum install mysql -y
```

2. 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
```

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

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

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

```
CREATE DATABASE webappdb;
```

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

```
SHOW DATABASES;
```

4. 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;
```

5. 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:

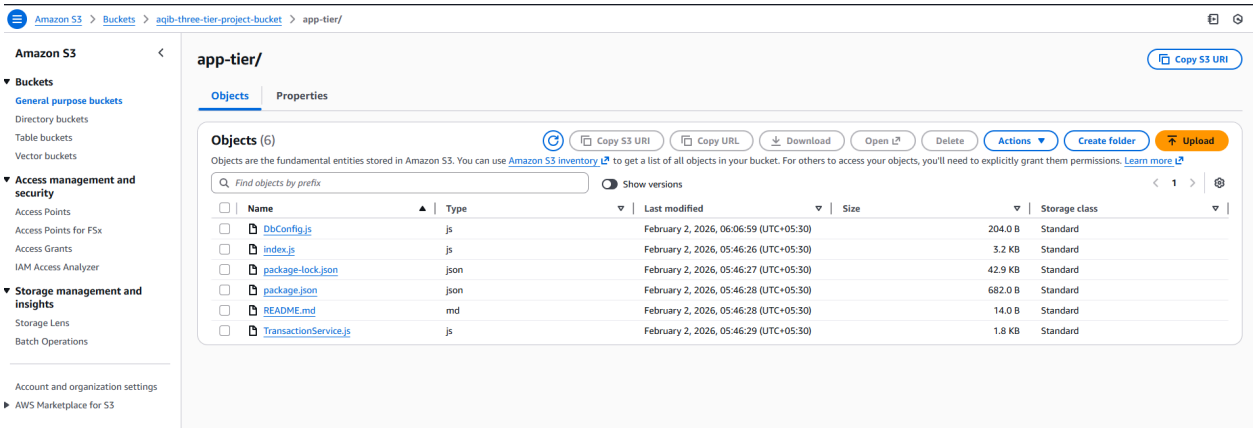
```
SELECT * FROM transactions;
```

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

4.3 - Configure app Instance

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.

NOTE: This is NOT considered a best practice, and is done for the simplicity of the lab. Moving these credentials to a more suitable place like Secrets Manager is left as an extension for this workshop.



Go back to your 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
```

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

```
nvm install 16
```

```
nvm use 16
```

5. *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
```

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

```
cd ~/
```

```
aws s3 cp s3://BUCKET_NAME/app-tier/ app-tier --recursive
```

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

```
cd ~/app-tier
```

```
npm install
```

```
pm2 start index.js
```

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
```

NOTE: If you're having issues, check your configuration file for any typos, and double check that you have followed all installation commands till now.

8. *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
```

4.4- Test App Tier

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

To hit out health check endpoint, copy this command into your SSM terminal. This is our simple health check endpoint that tells us if the app is simply running.

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

The response should looks like the following:

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

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

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

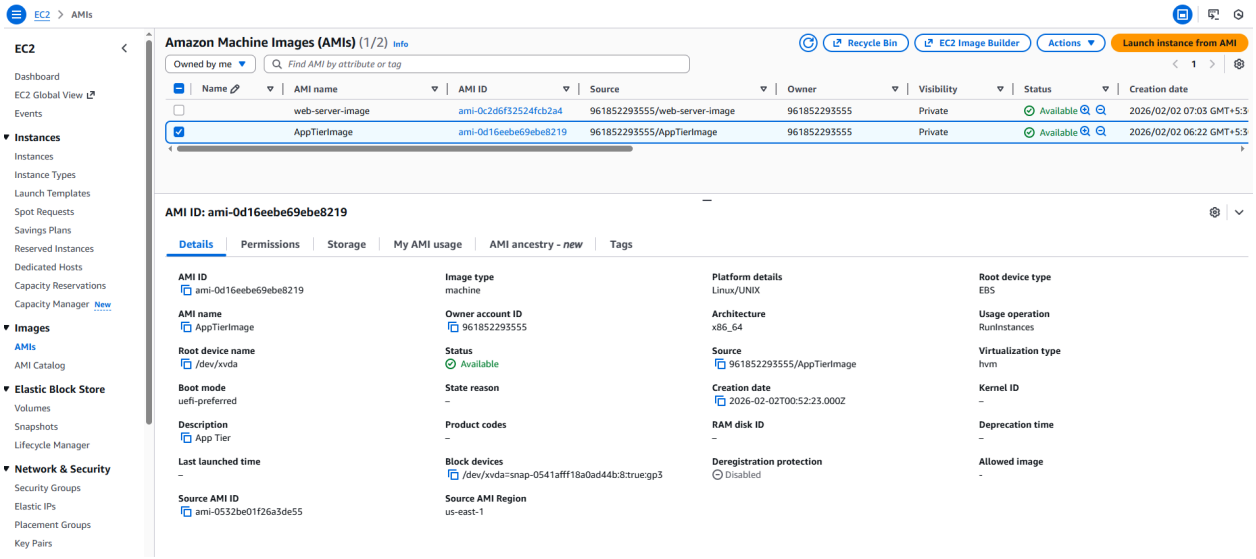
You should see a response containing the test data we added earlier:

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

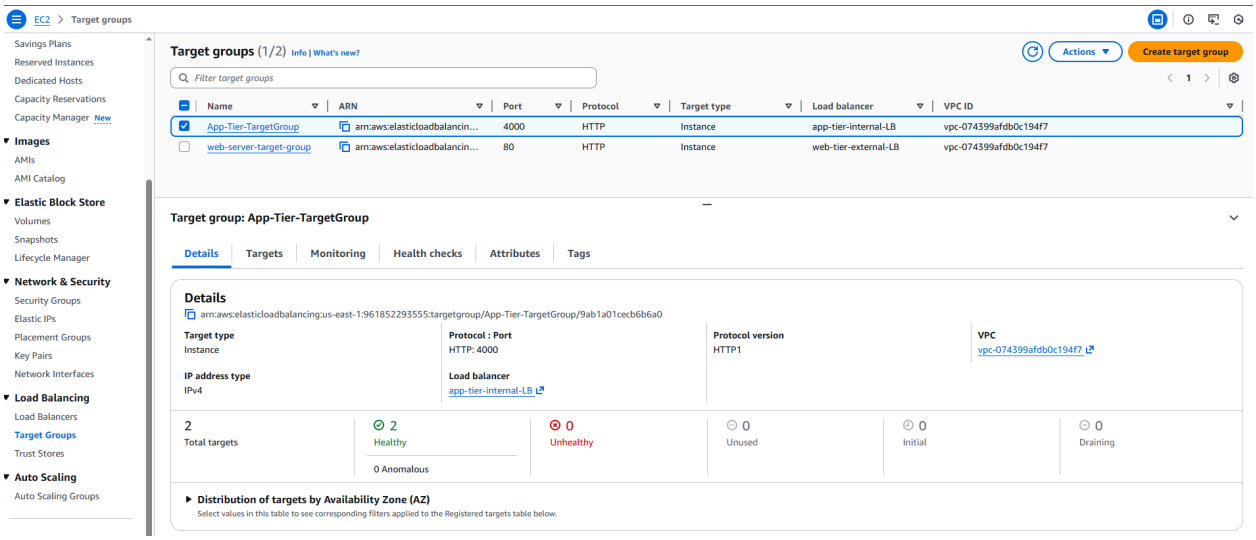
If you see both of these responses, then your networking, security, database and app configurations are correct.

Part 5 - Internal Load Balancing and Auto Scaling

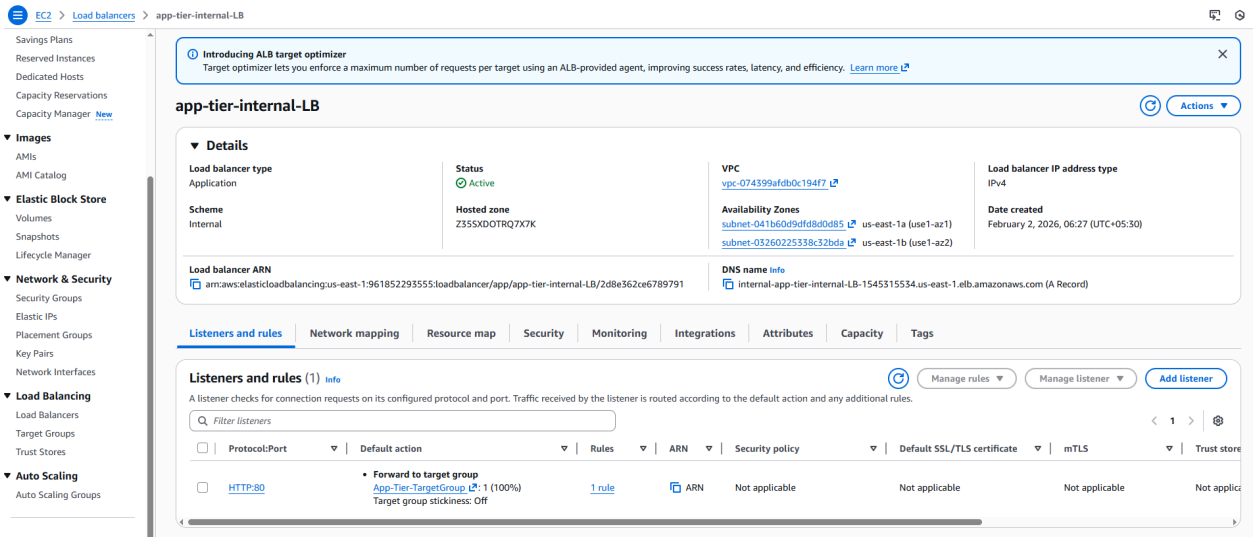
5.1 - App Tier AMI



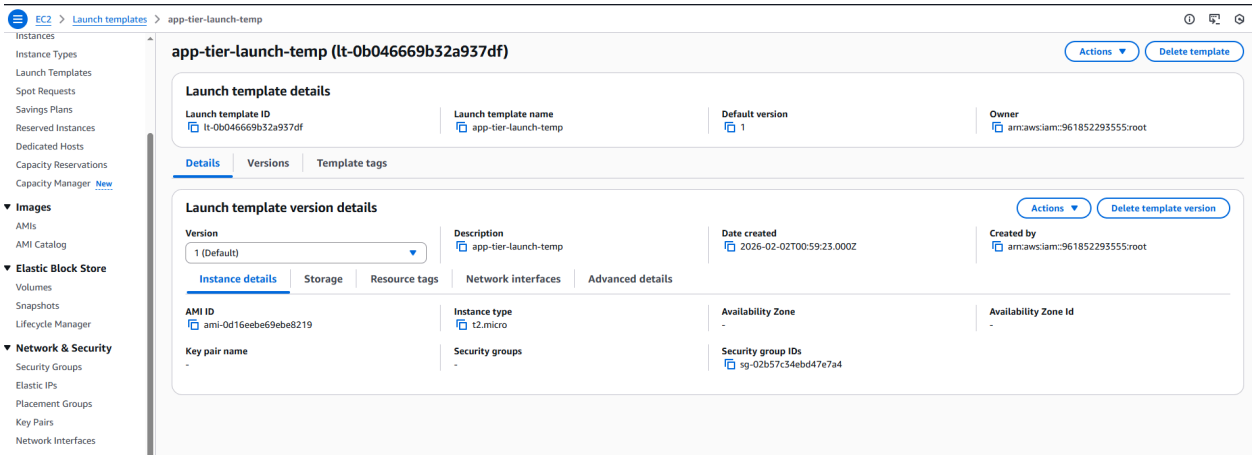
5.2 - Target Group



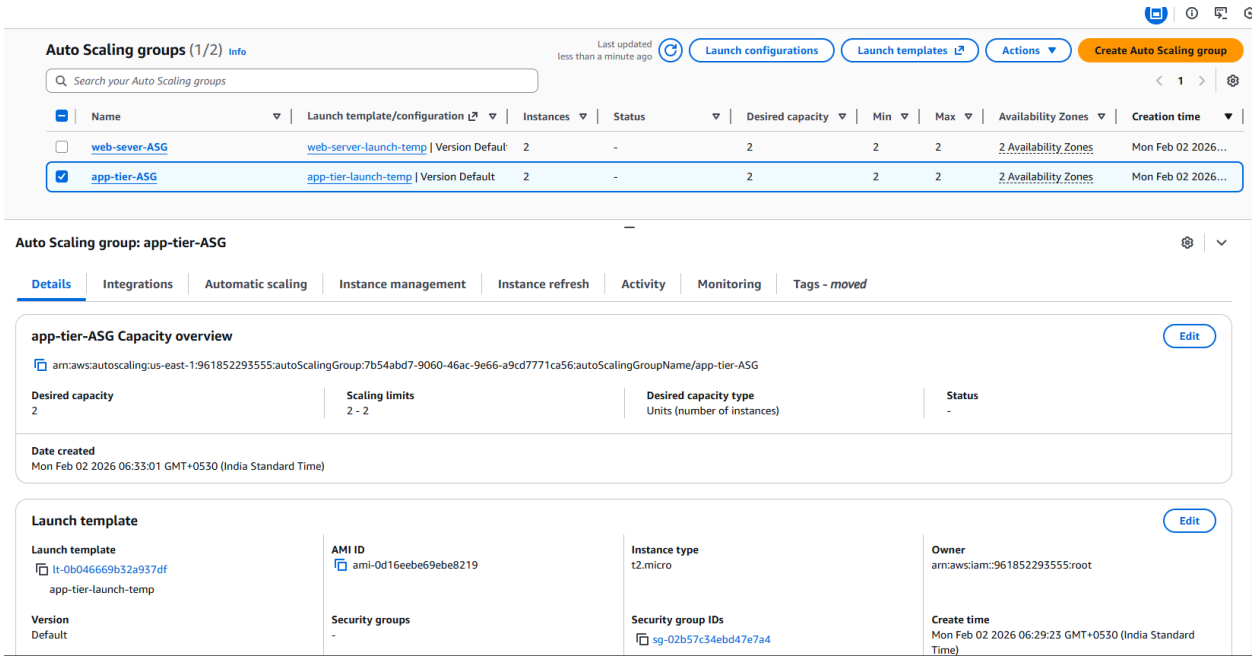
5.3 - Internal LB



5.4- Launch Template



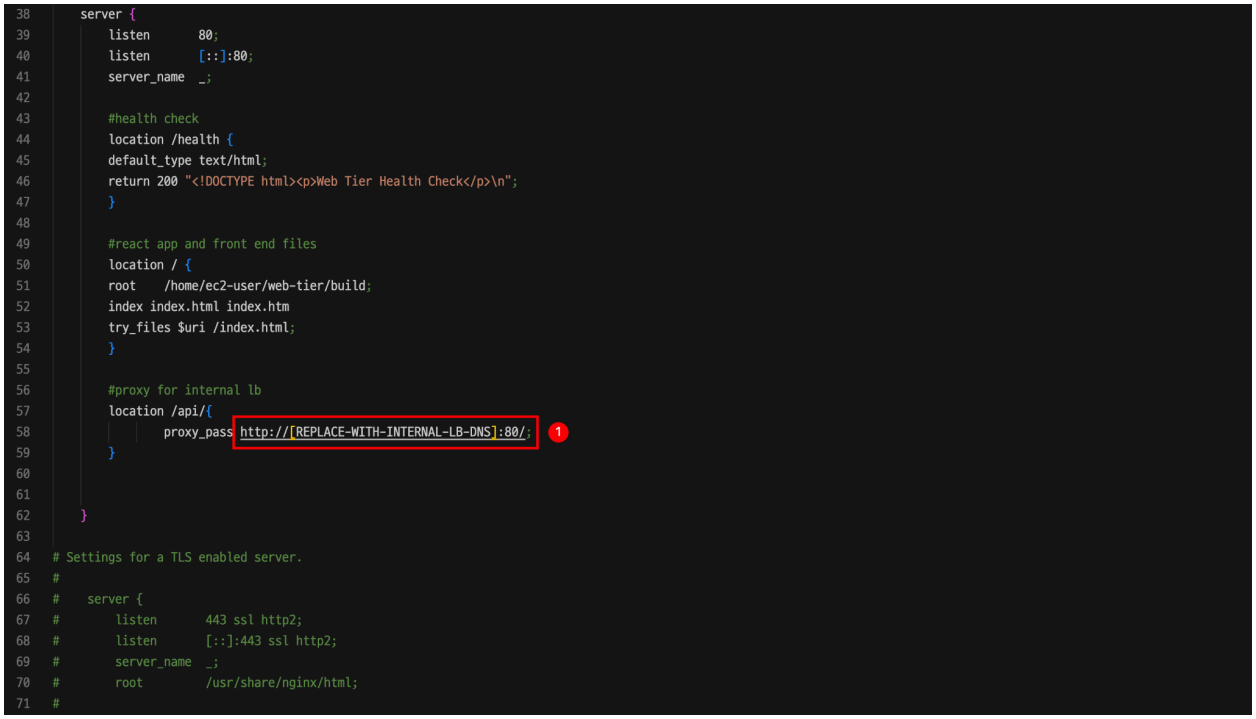
5.5 - ASG App-Tier



Part 6: Web Tier Instance Deployment

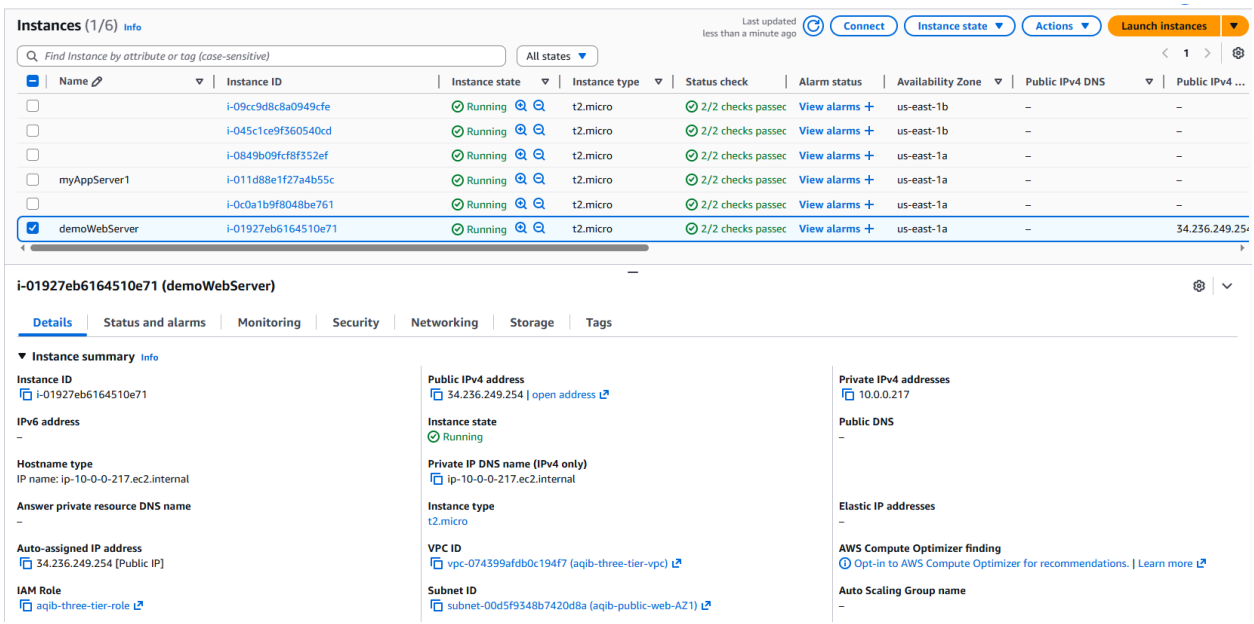
6.1 - Update Config File

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.

6.2 - Web Instance Deployment & connect to Instance



6.3 - Configure Web Instance

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

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

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:

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

```
cd /etc/nginx
ls
```

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

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

Then, restart Nginx with the following command:

```
sudo service nginx restart
```

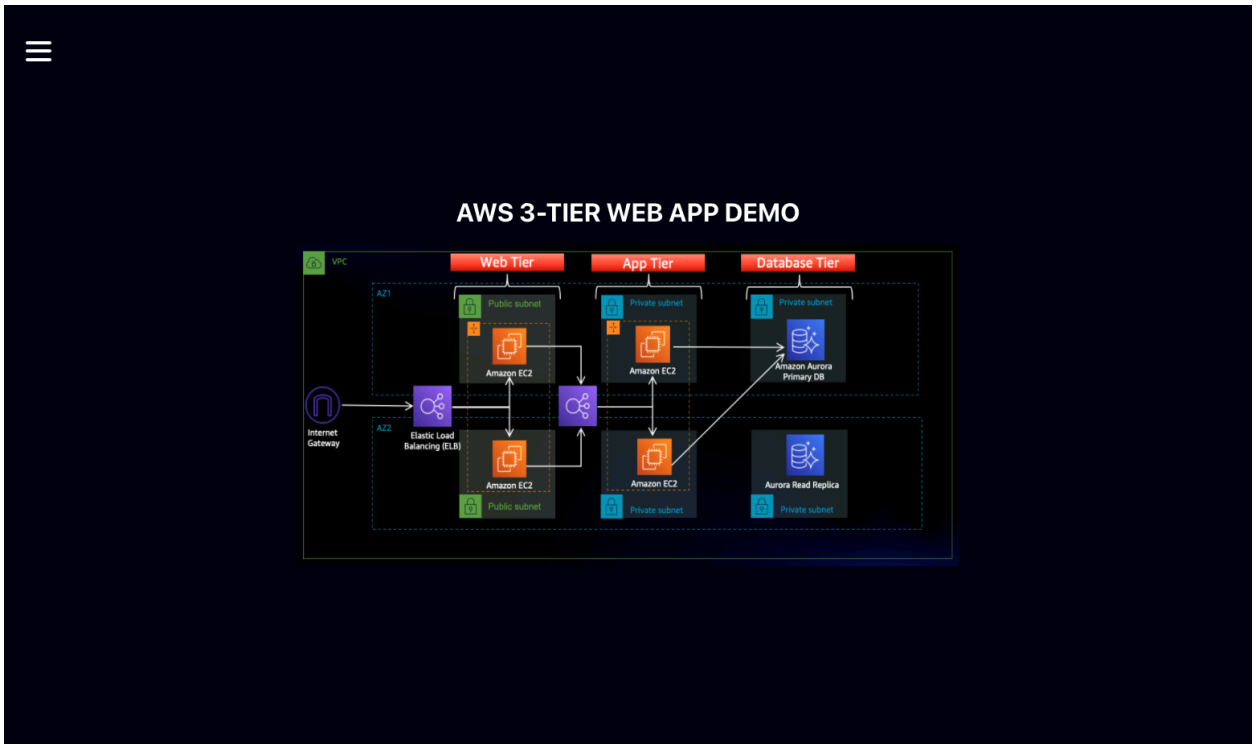
To make sure Nginx has permission to access our files execute this command:

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

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

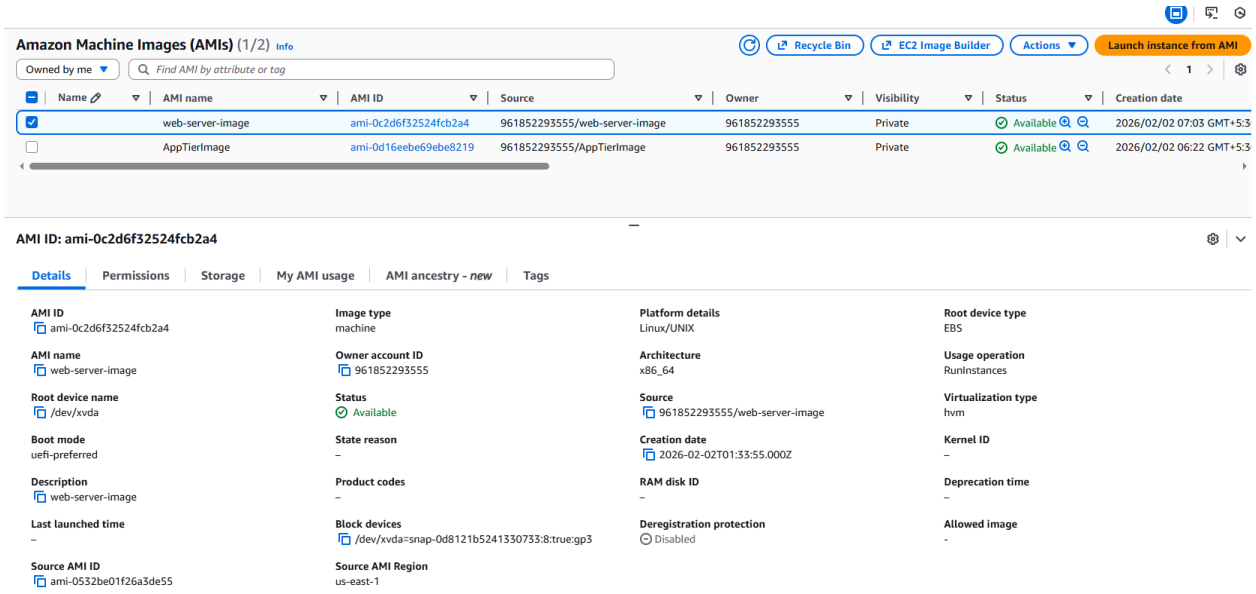
```
sudo chkconfig nginx on
```

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.

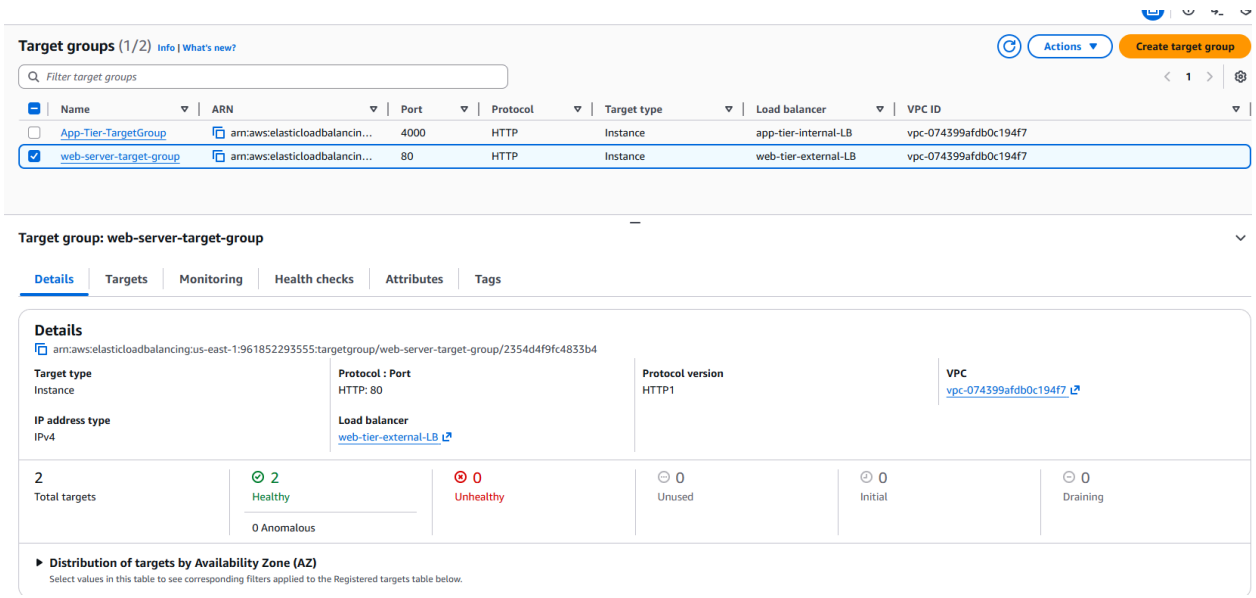


Part 7: External Load Balancer and Auto Scaling

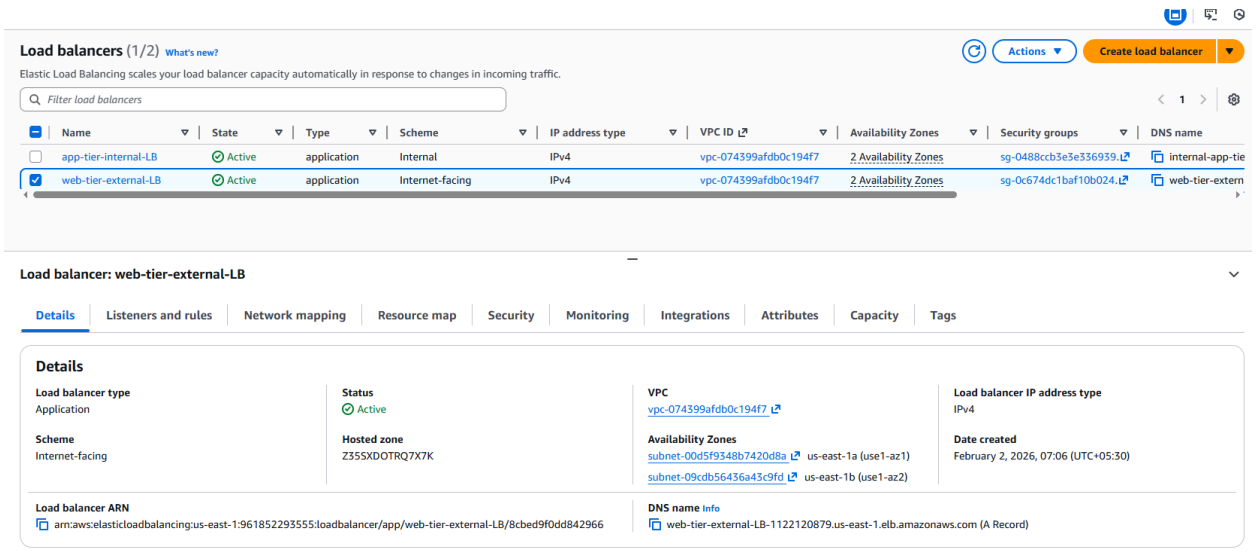
7.1 - Web Tier AMI



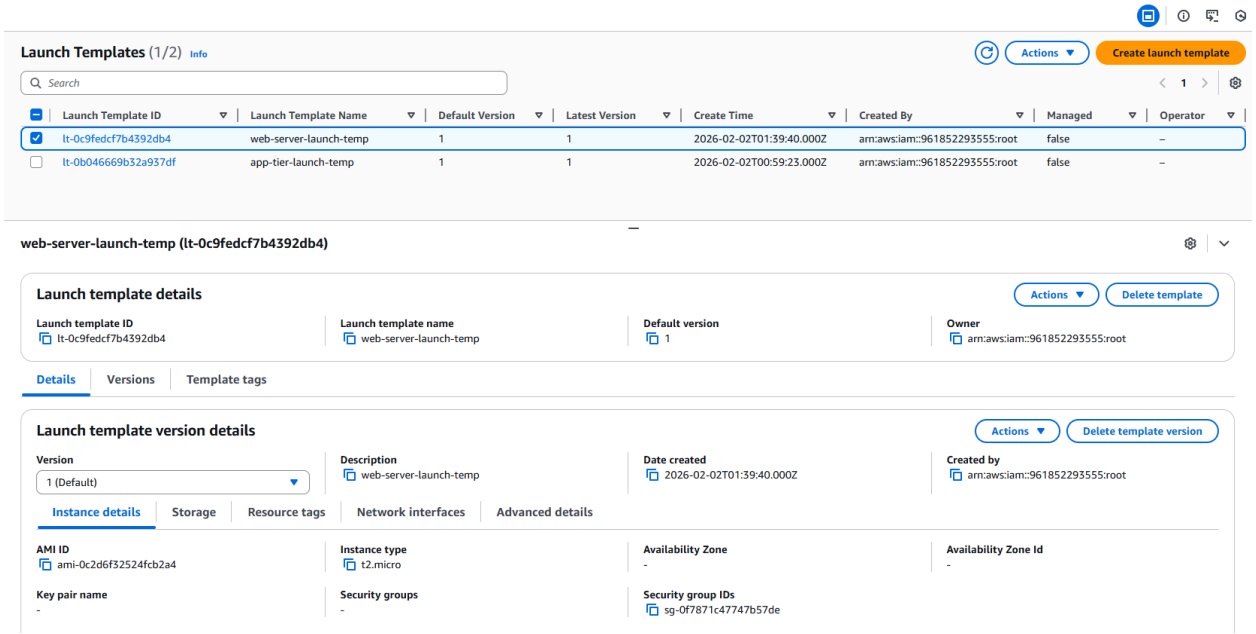
7.2 - TG



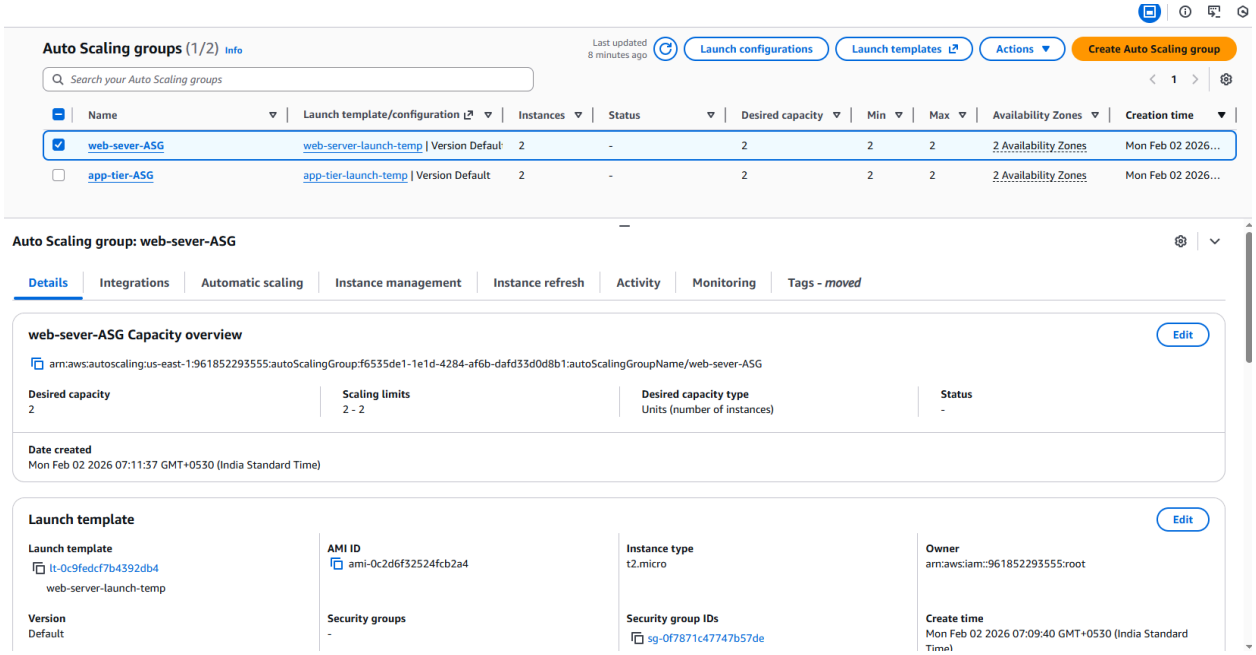
7.3 - Internet Facing LB



7.4 - Launch Template



7.5 - ASG


Auto Scaling groups (1/2) Info
Last updated 8 minutes ago
Launch configurations
Launch templates
Actions
Create Auto Scaling group
Search your Auto Scaling groups
Name
Launch template/configuration
Instances
Status
Desired capacity
Min
Max
Availability Zones
Creation time
web-sever-ASG
web-server-launch-temp | Version Default
2
-
2
2
2
2 Availability Zones
Mon Feb 02 2026...
app-tier-ASG
app-tier-launch-temp | Version Default
2
-
2
2
2
2 Availability Zones
Mon Feb 02 2026...
Auto Scaling group: web-sever-ASG
Details
Integrations
Automatic scaling
Instance management
Instance refresh
Activity
Monitoring
Tags - moved
web-sever-ASG Capacity overview
Edit
arn:aws:autoscaling:us-east-1:961852293555:autoScalingGroup:f6535de1-1e1d-4284-af6b-dafd33d0d8b1:autoScalingGroupName/web-sever-ASG
Desired capacity
2
Scaling limits
2 - 2
Desired capacity type
Units (number of instances)
Status
-
Date created
Mon Feb 02 2026 07:11:37 GMT+0530 (India Standard Time)
Launch template
Edit
Launch template
lt-0c9fedcf7b4392db4
web-server-launch-temp
AMI ID
ami-0c2d6f32524fcb2a4
Instance type
t2.micro
Owner
arn:aws:iam::961852293555:root
Version
Default
Security groups
-
Security group IDs
sg-0f7871c47747b57de
Create time
Mon Feb 02 2026 07:09:40 GMT+0530 (India Standard Time)

Load Balancer URL working fine the website is up and running