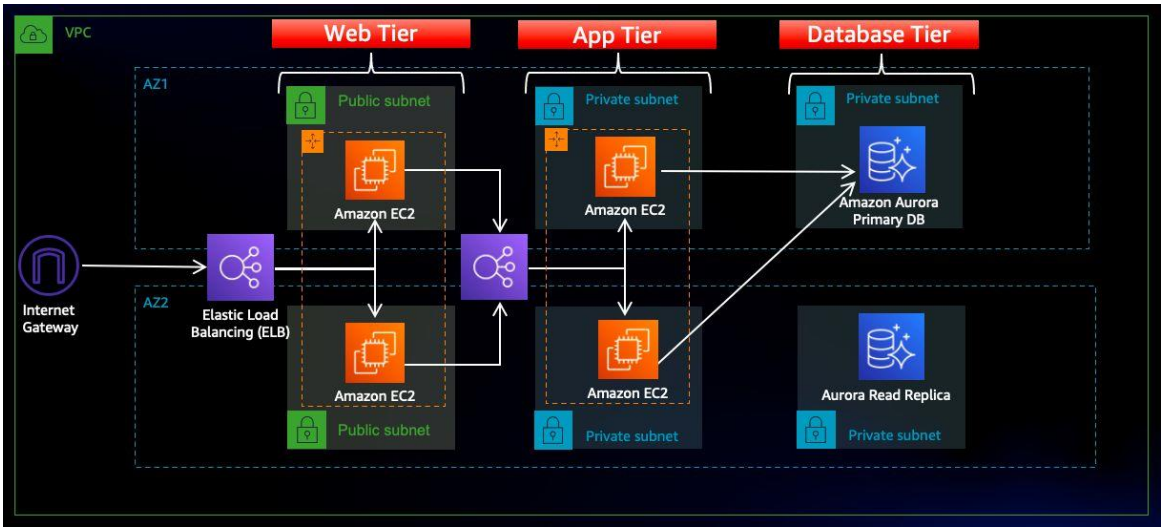
AWS Three Tier Web Architecture

Project hands on ***by P.Lokesh(mech)***

Introduction :

Architecture Overview :



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.

It contains total 6parts as below :
Part 0 : Set up
Part 1 : Networking & security
Part 2 : Database Deployment
Part 3 : App Tier Instance Deployment
Part 4 : Internal Load Balancing and Auto Scaling
Part 5 : Web Tier Instance Deployment
Part 6: : External Load Balancer and Auto Scaling

Part 0 : Set up

For this workshop, we will be downloading the code from Github and upload it to S3 so our instances can access it. We will also create an AWS Identity and Access Management EC2 role so we can use AWS Systems Manager Session Manager to connect to our instances securely and without needing to create SSH key pairs.

Learning Objectives:

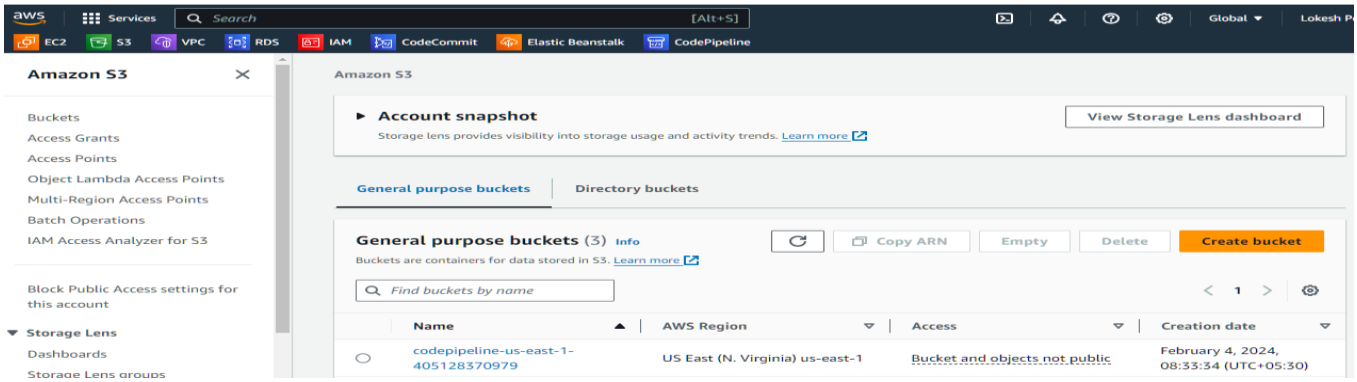
- S3 Bucket Creation.
- IAM EC2 Instance Role Creation.
- Download Code from Github Repository.

Download Code from Github :

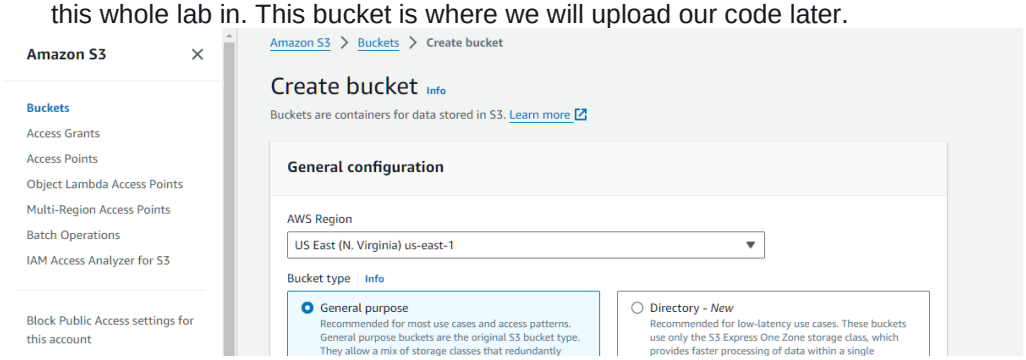
Download the code from [this repository](https://github.com/aws-samples/aws-three-tier-web-architecture-workshop.git) into your local environment by running the command below. If you don't have git installed, you can just download the zip. Save it somewhere you can easily access.
git clone https://github.com/aws-samples/aws-three-tier-web-architecture-workshop.g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

S3 Bucket Creation:

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

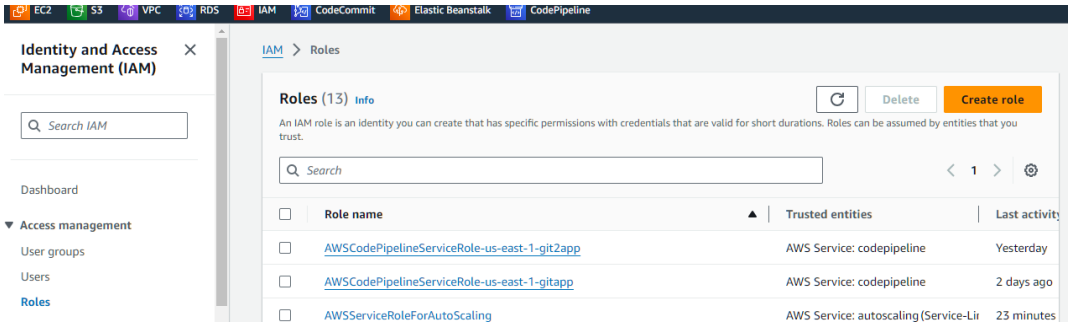


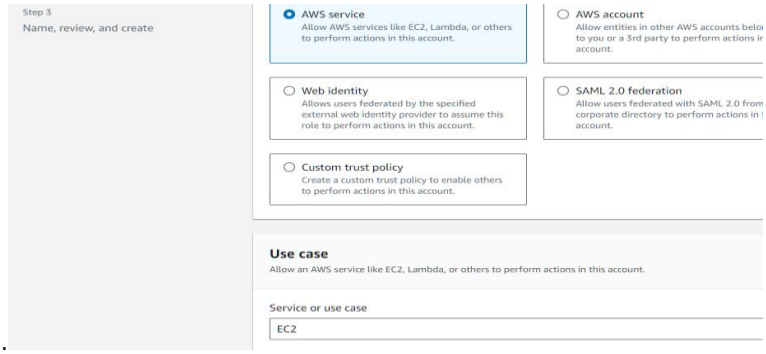
2) Give it a unique name, and then leave all the defaults as in. Make sure to select the region that you intend to run this whole lab in. This bucket is where we will upload our code later.



IAM EC2 Instance Role Creation:

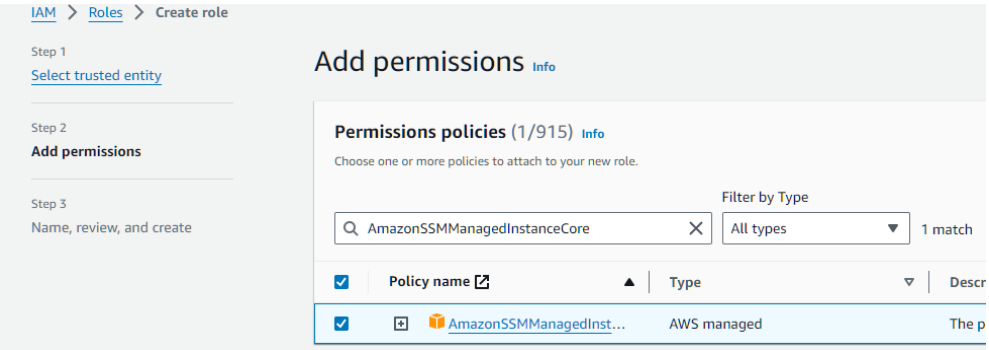
1) Navigate to the IAM dashboard in the AWS console and create an EC2 role.





2) Select EC2 as the trusted entity .
3) When adding permissions, include the following AWS managed policies. You can search for them and select them. These policies will allow our instances to download our code from S3 and use Systems Manager Session Manager to securely connect to our instances without SSH keys through the AWS console.

- **AmazonSSMManagedInstanceCore**
- **AmazonS3ReadOnlyAccess.**



4) Give your role a name, and then click **Create Role**.

Part 1: Networking and Security

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

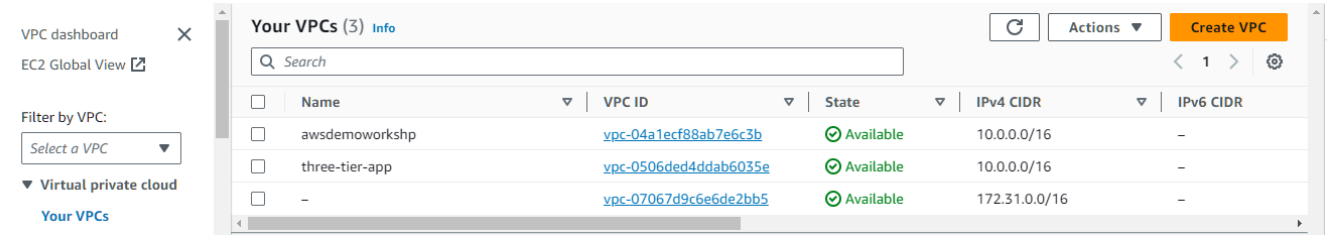
Learning Objectives:

- Create an isolated network with the following components:
  - VPC
  - Subnets
  - Route Tables
  - Internet Gateway
  - NAT gateway
  - Security Groups

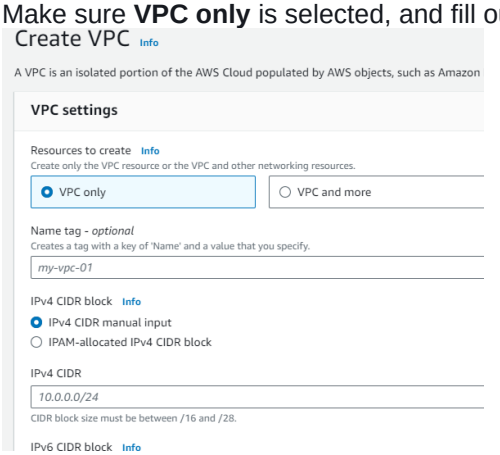
VPC and Subnets

1) **VPC Creation :**

Navigate to the VPC dashboard in the AWS console and navigate to **Your VPCs** on the left hand side



2) Make sure **VPC only** is selected, and fill out the VPC Settings with a **Name tag** and a **CIDR range** of your choice.

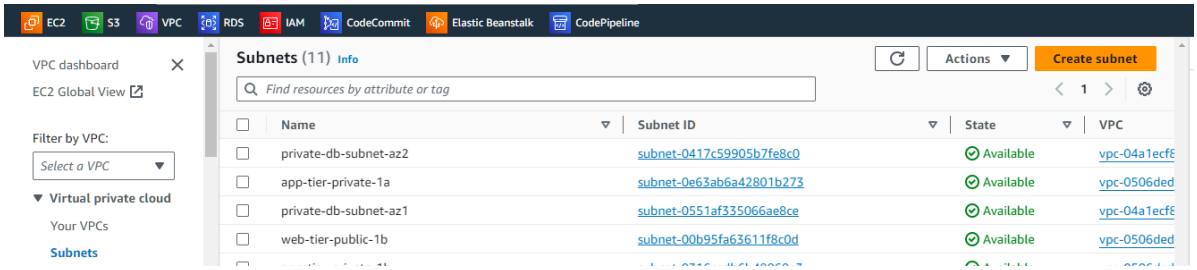


NOTE: Make sure you pay attention to the region you're deploying all your resources in. You'll want to stay consistent for this workshop.

NOTE: Choose a CIDR range that will allow you to create at least 6 subnets.

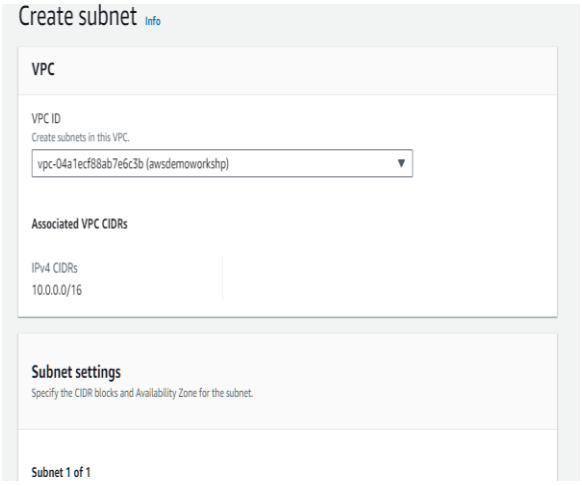
Subnet Creation:

1. Next, create your subnets by navigating to **Subnets** on the left side of the dashboard and clicking **Create subnet**.

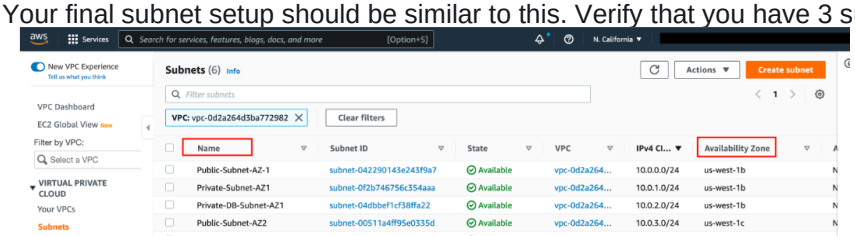


2. We will need six subnets across two availability zones. That means that three subnets will be in one availability zone, and three subnets will be in another zone. Each subnet in one availability zone will correspond to one layer of our three tier architecture. Create each of the 6 subnets by specifying the VPC we created in part 1 and then choose a name, availability zone, and appropriate CIDR range for each of the subnets.
NOTE*: It may be helpful to have a naming convention that will help you remember what each subnet is for. For example in one* ***AZ*** *you might have the following****: Public-Web-Subnet-AZ-1, Private-App-Subnet-AZ-1, Private-DB-Subnet-AZ-1.***

NOTE*: Remember, your CIDR range for the subnets will be subsets of your VPC CIDR range.*



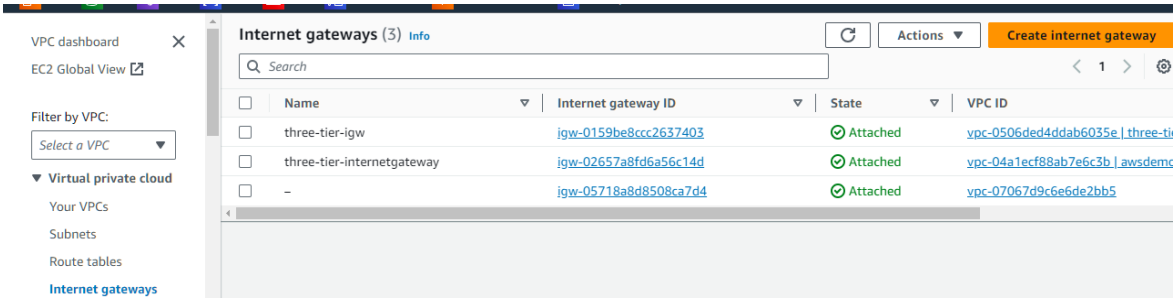
Your final subnet setup should be similar to this. Verify that you have 3 subnets across 2 different availability zones.



Internet Connectivity:

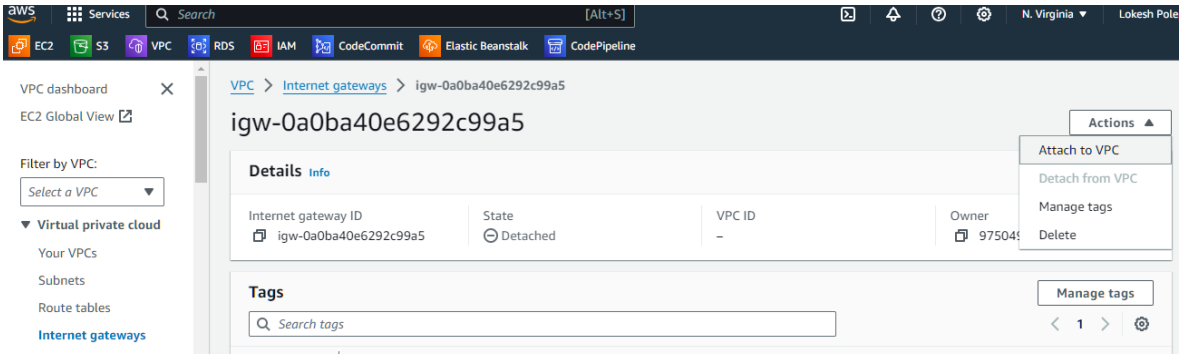
Internet Gateway

1. In order to give the public subnets in our VPC internet access we will have to create and attach an Internet Gateway. On the left hand side of the VPC dashboard, select **Internet Gateway**.

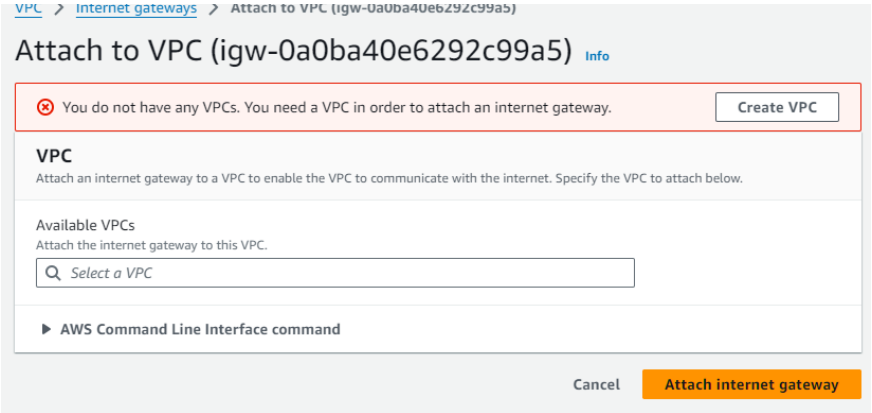


2. Create your internet gateway by simply giving it a name and clicking **Create internet gateway**.

3. After creating the internet gateway, attach it to your VPC that you create in the **VPC and Subnet Creation** step of the workshop. You have a couple options on how to do this, either with the creation success message or the **Actions** drop down.

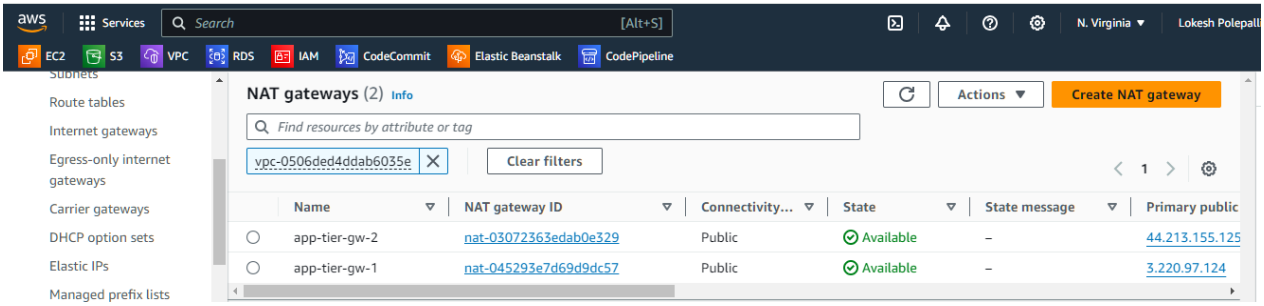


Then, select the correct VPC and click **Attach internet gateway**.

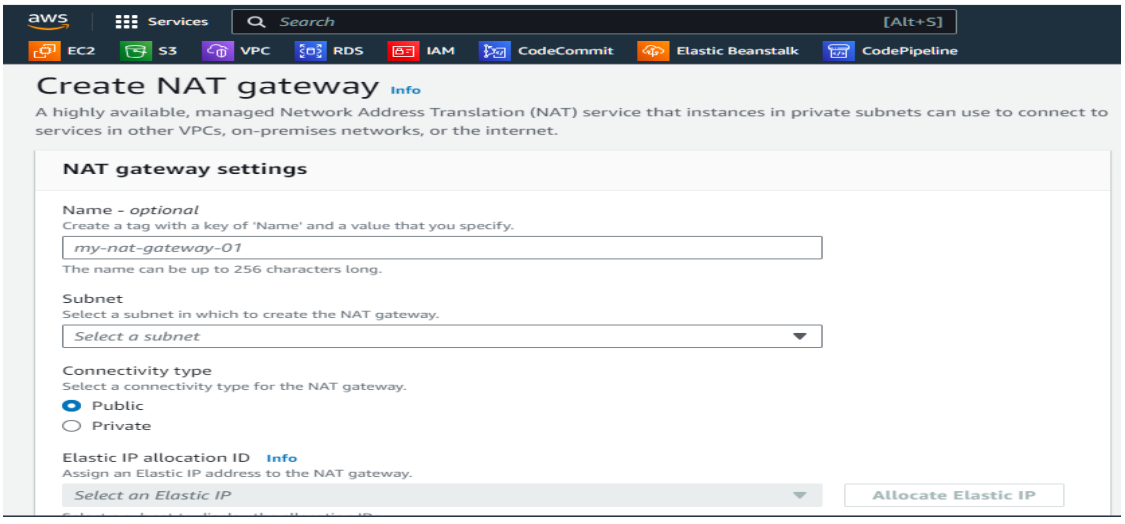


NAT Gateway

1. 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**.



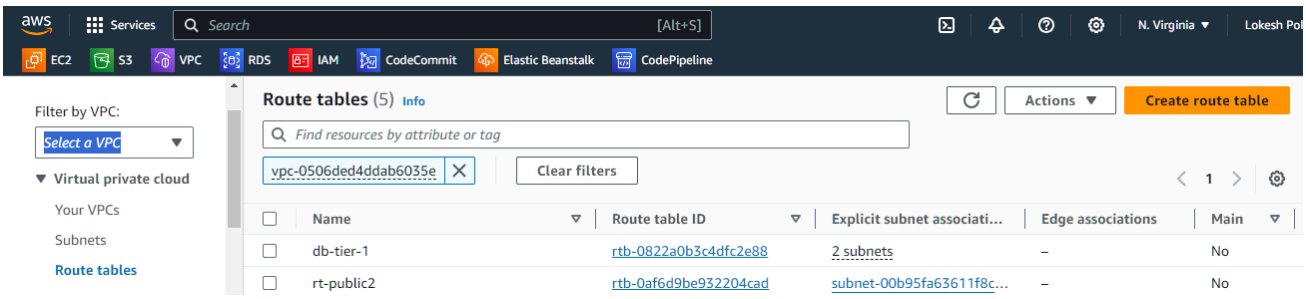
2. Fill in the **Name**, choose one of the **public subnets** you created in part 2, and then allocate an Elastic IP. Click **Create NAT gateway**.

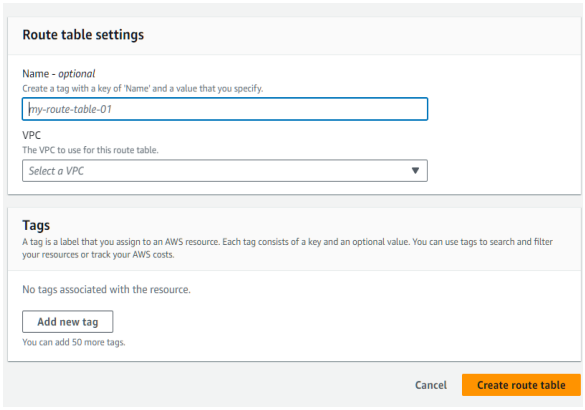


3. Repeat step 1 and 2 for the other subnet.

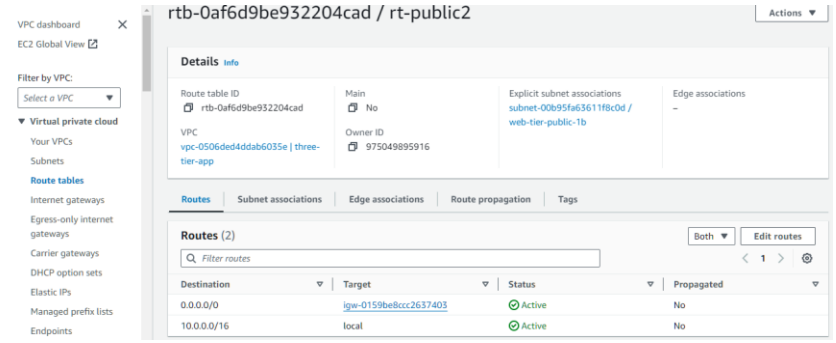
Routing Configuration:

1. Navigate to Route Tables on the left side of the VPC dashboard and click Create route table First, let's create one route table for the web layer ***public subnets*** and name it accordingly.

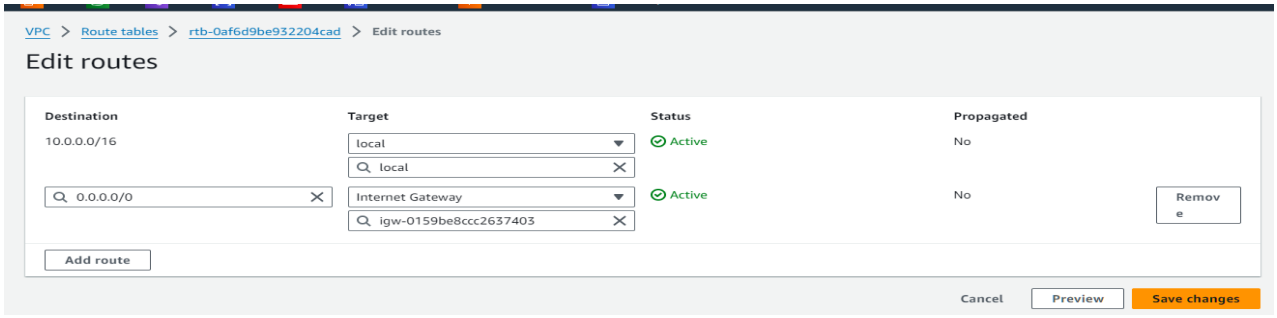




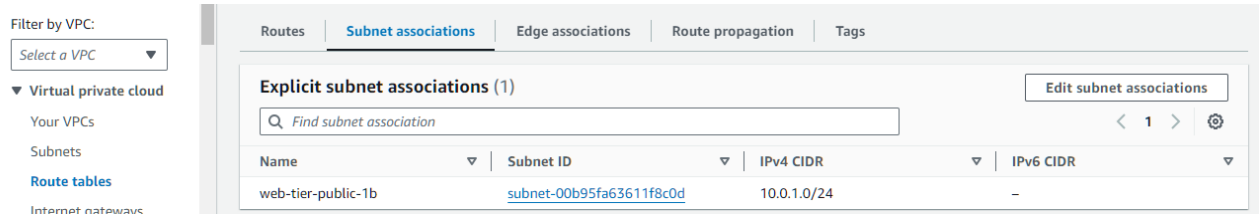
2. After creating the route table, you'll automatically be taken to the details page. Scroll down and click on the **Routes tab** and **Edit routes**.



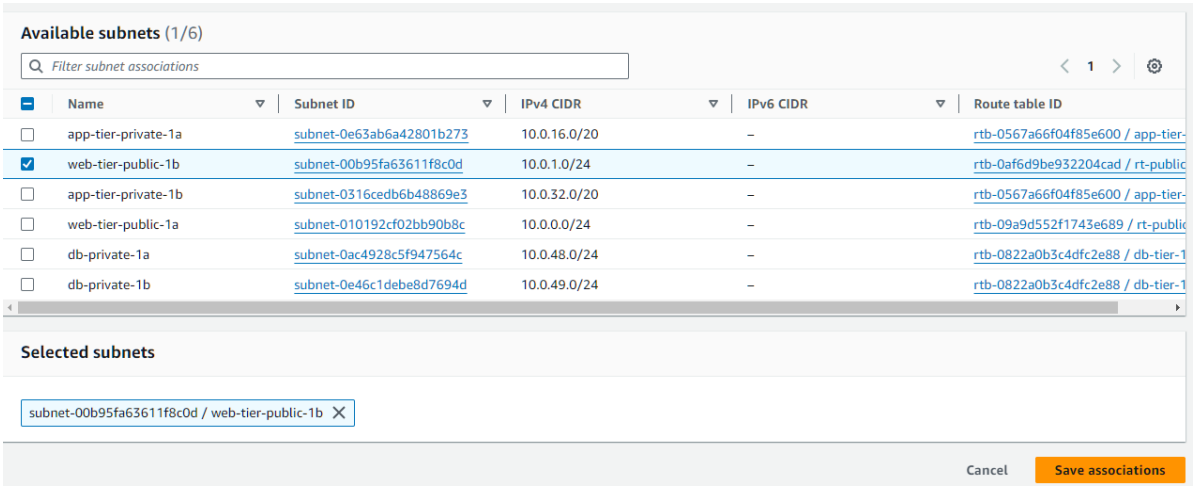
3. Add a route that directs traffic from the VPC to the internet gateway. In other words, for all traffic *destined* for IPs outside the VPC CDIR range, add an entry that directs it to the internet gateway as a *target*. Save the changes.



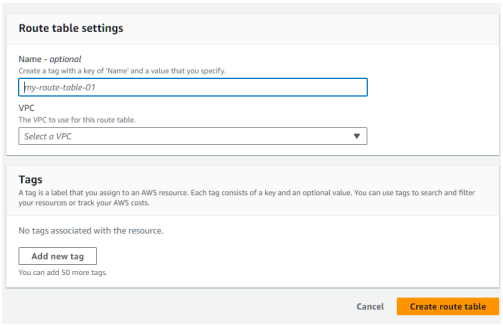
4. Edit the *Explicit Subnet Associations* of the route table by navigating to the route table details again. Select **Subnet Associations** and click **Edit subnet associations**.

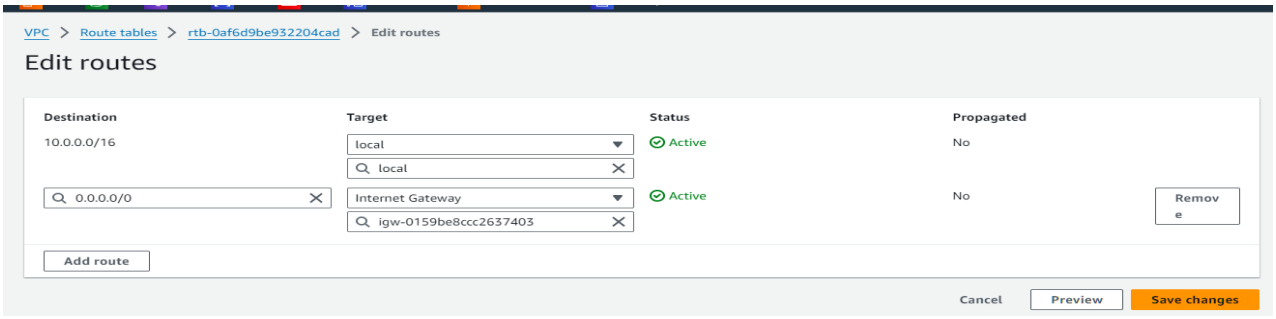


Select the two web layer public subnets you created eariler and click **Save associations**.

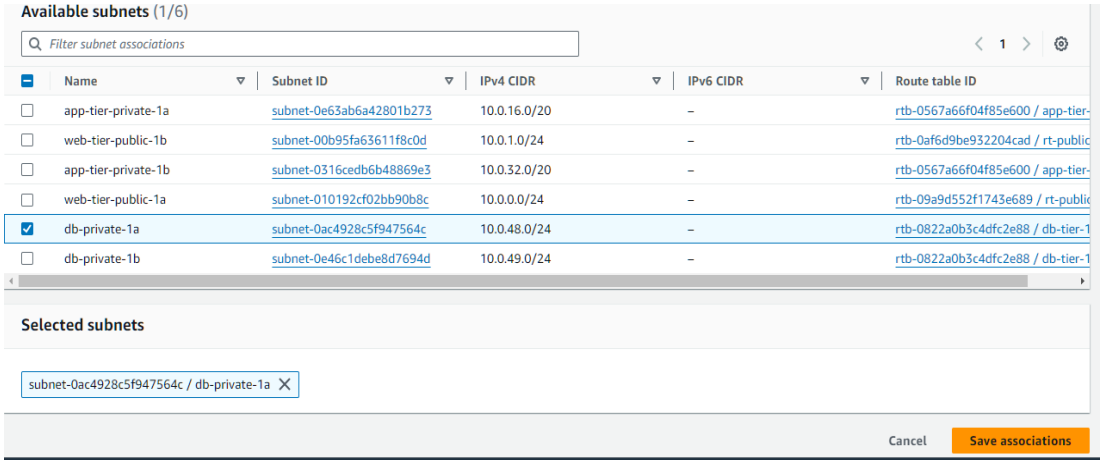


4. Now create 2 more route tables, one for each app layer private subnet in each availability zone. These route tables will route app layer traffic destined for outside the VPC to the NAT gateway in the respective availability zone, so add the appropriate routes for that.



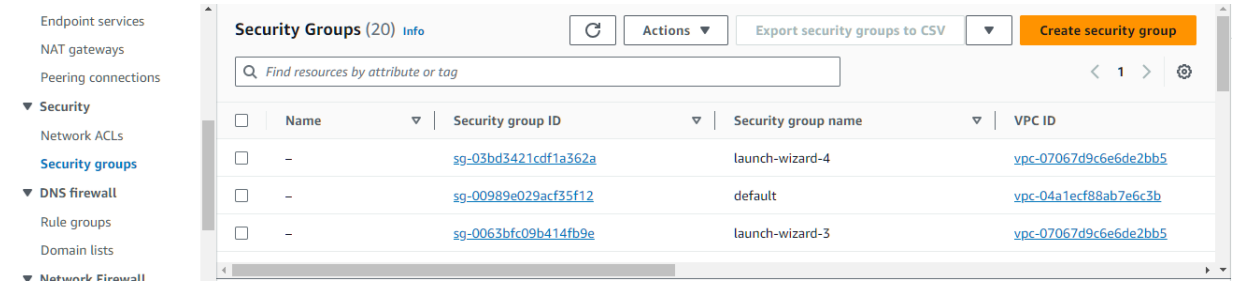


Once the route tables are created and routes added, add the appropriate subnet associations for each of the app layer private subnets.

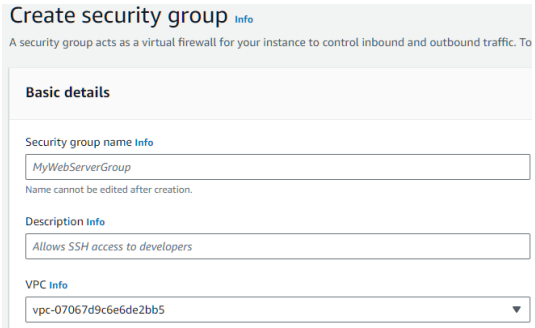


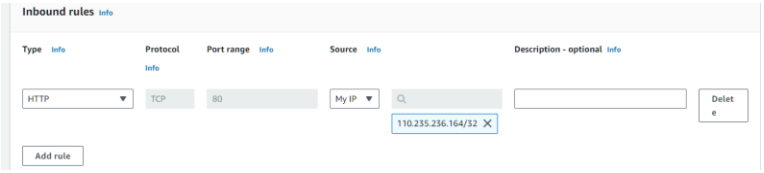
Security Groups:

1. Security groups will tighten the rules around which traffic will be allowed to our Elastic Load Balancers and EC2 instances. Navigate to **Security Groups** on the left side of the VPC dashboard, under **Security**.

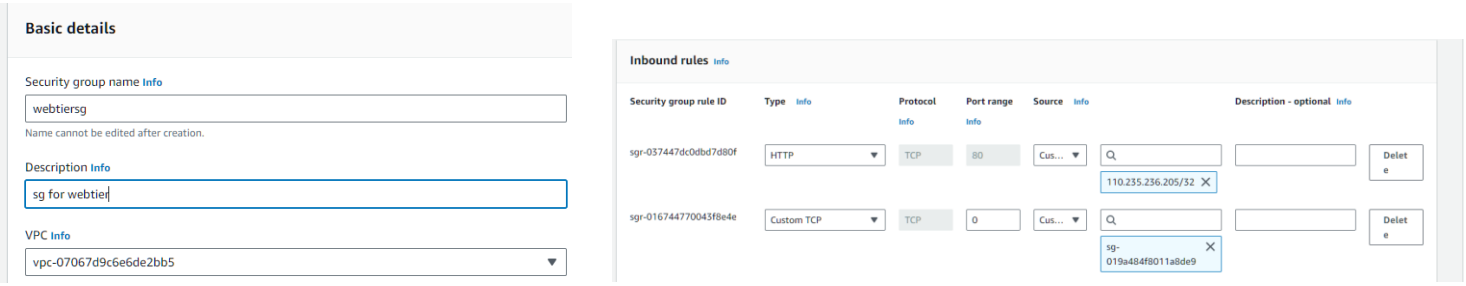


2. The first security group you'll create is for the public, **internet facing** load balancer. After typing a name and description, add an inbound rule to allow **HTTP** type traffic for your **IP**.

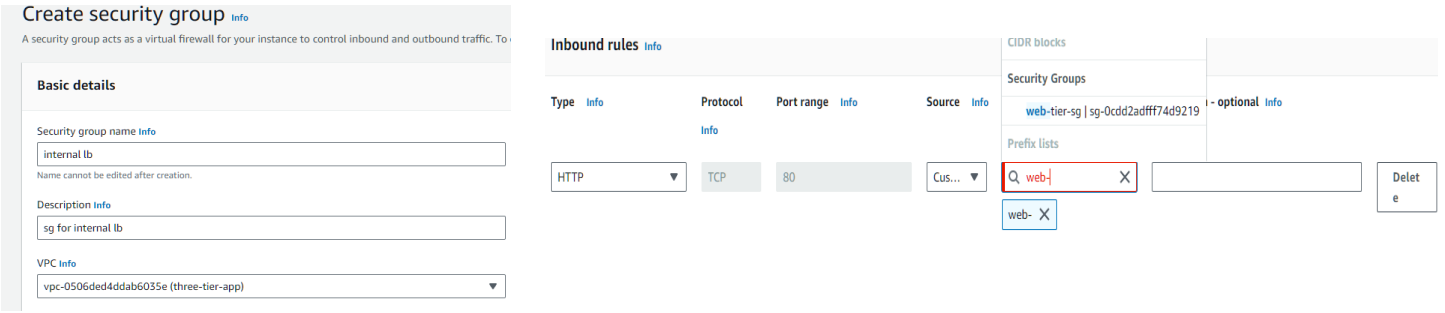




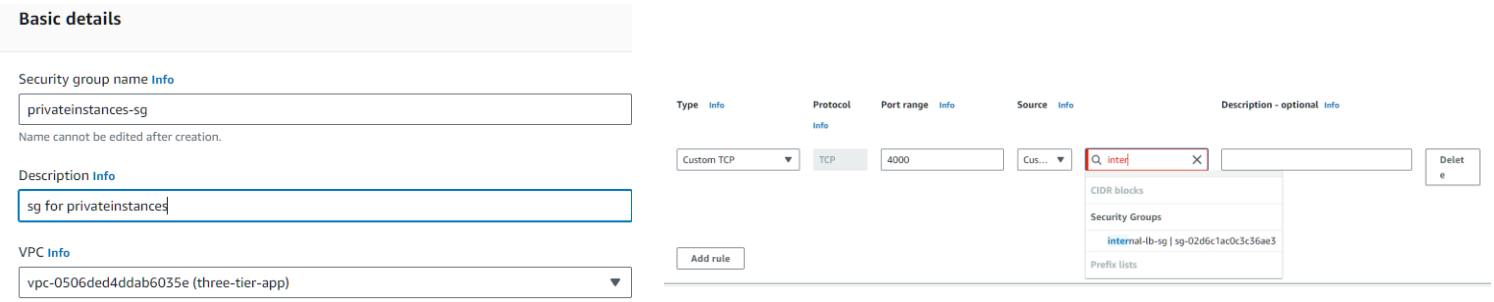
3. The second security group you'll create is for the public instances in the web tier. After typing a name and description, add an inbound rule that allows **HTTP** type traffic from your internet facing load balancer security group you created in the previous step. This will allow traffic from your 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.



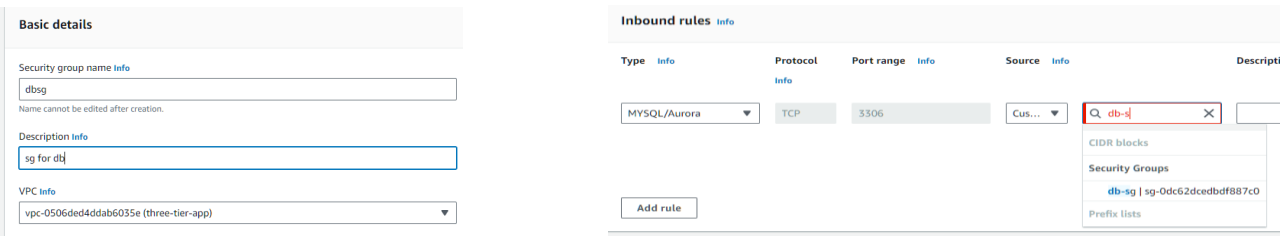
4. The third security group will be for our internal load balancer. Create this new security group and add an inbound rule that allows **HTTP** type traffic from your public instance security group. This will allow traffic from your web tier instances to hit your internal load balancer.



5. 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.



6. 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).



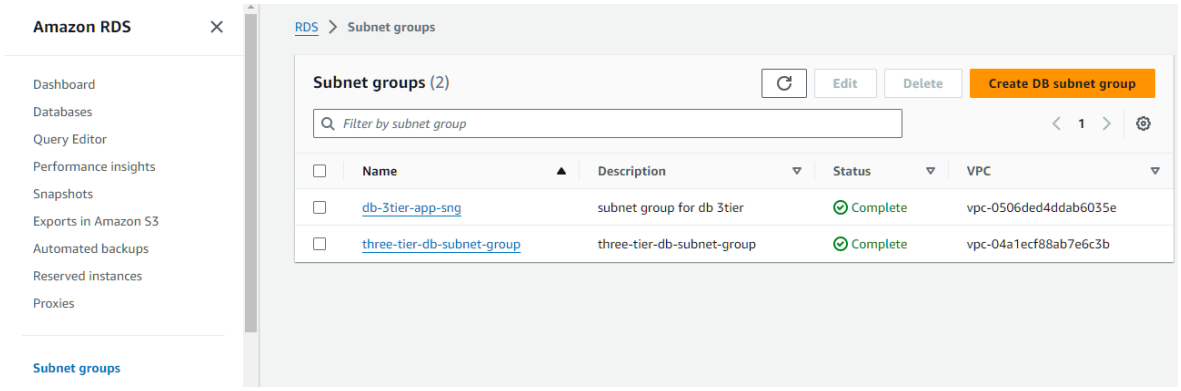
Part 2: Database Deployment

This section of the workshop will walk you through deploying the database layer of the three tier architecture.
Learning Objectives:

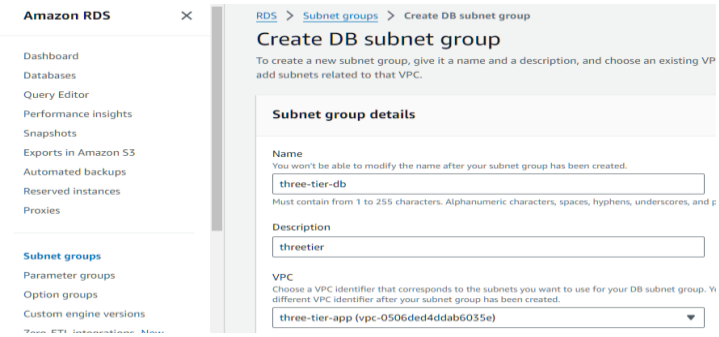
- Deploy Database Layer
    - Subnet Groups
    - Multi-AZ Database

Subnet Groups:

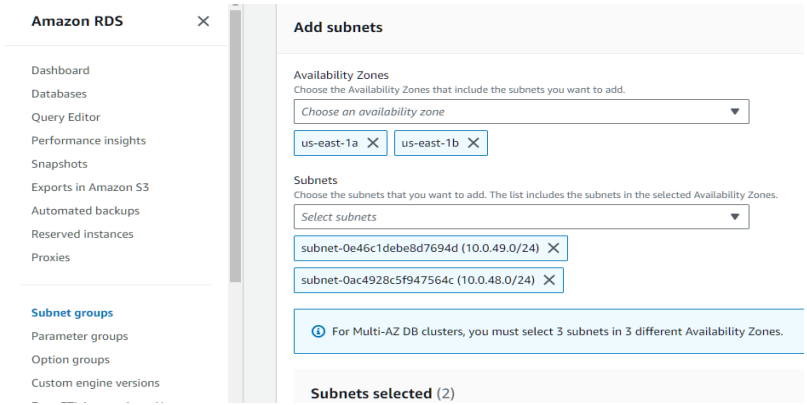
1. Navigate to the RDS dashboard in the AWS console and click on **Subnet groups** on the left hand side. Click **Create DB subnet group**.



2. Give your subnet group a name, description, and choose the VPC we created.

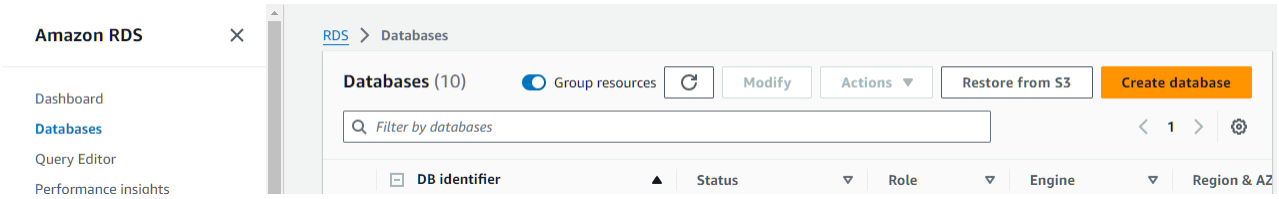


3. When adding subnets, make sure to add the subnets we created in each availability zone specificaly for our database layer. You may have to navigate back to the VPC dashboard and check to make sure you're selecting the correct subnet IDs.

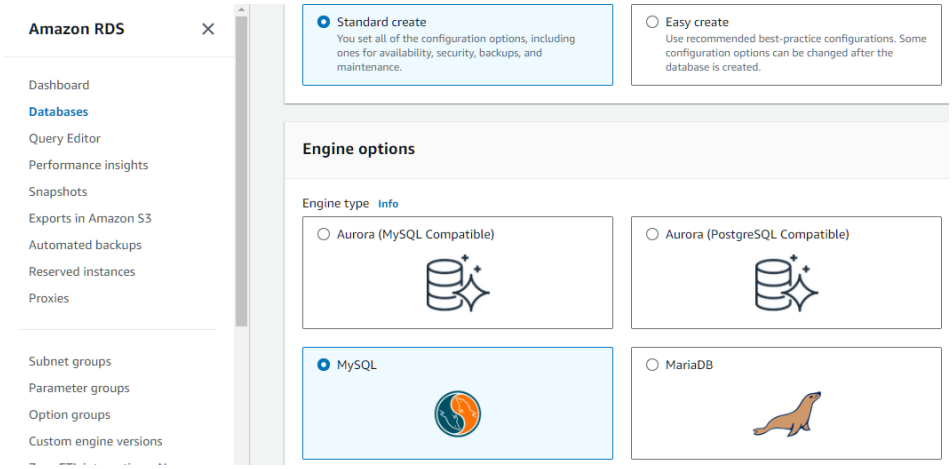


Database Deployment:

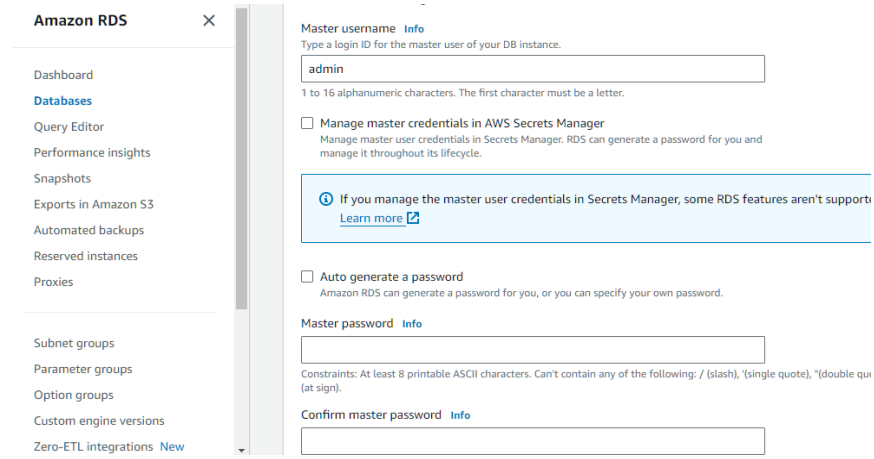
1. Navigate to **Databases** on the left hand side of the RDS dashboard and click **Create database**.



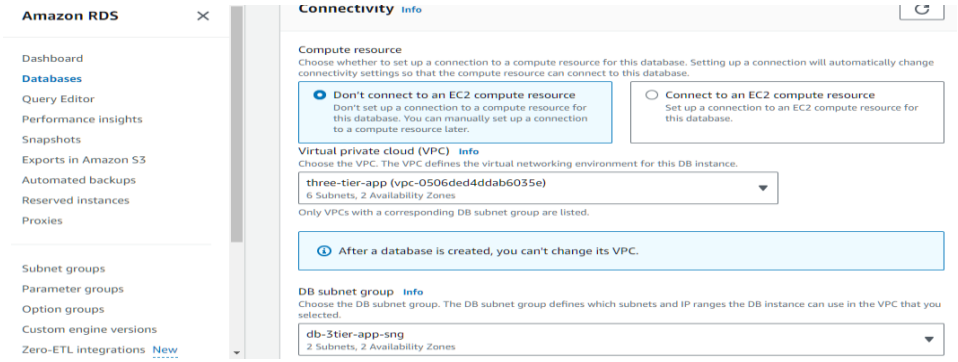
2. We'll now go through several configuration steps. Start with a **Standard create** for this **MySQL (FREE TIER)** database. Leave the rest of the defaults in the **Engine options** as default.



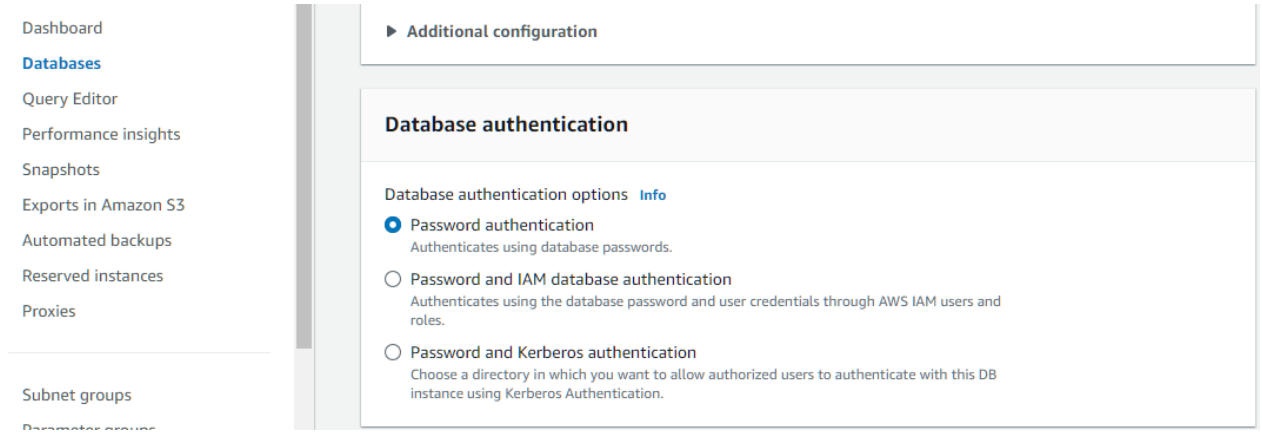
3. Under the **Templates** section choose **FREE TIER** since this isn't being used for production at the moment. Under **Settings** set a username and password of your choice and note them down since we'll be using password authentication to access our database.



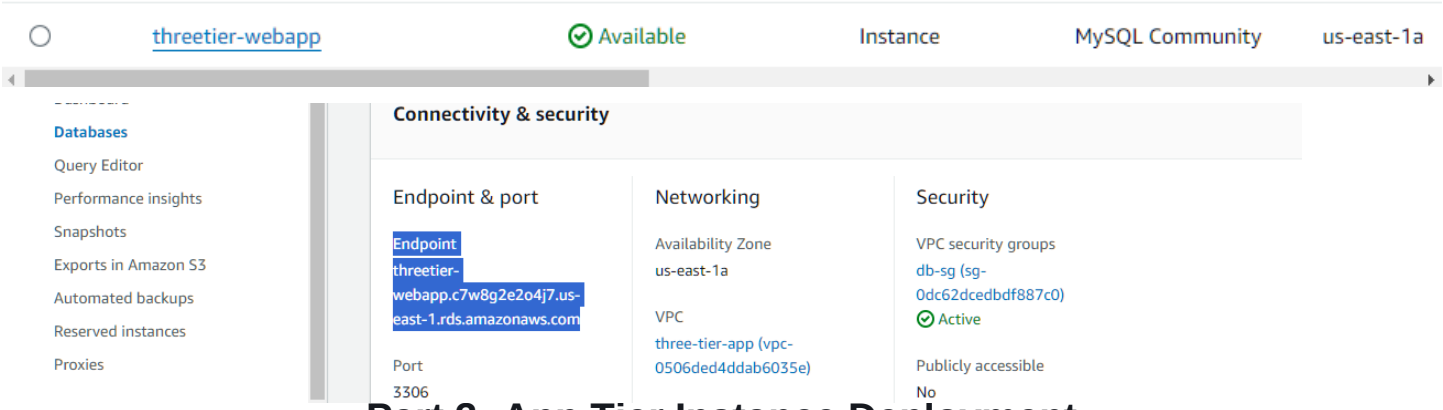
4. Under **Connectivity**, set the VPC, choose the subnet group we created earlier, and select no for public access



Set the security group we created for the database layer, make sure **password authentication** is selected as our authentication choice, and create the database.



3. When your database is provisioned, you should see your database available. Note down the writer endpoint for your database for later use.



Part 3: App Tier Instance Deployment

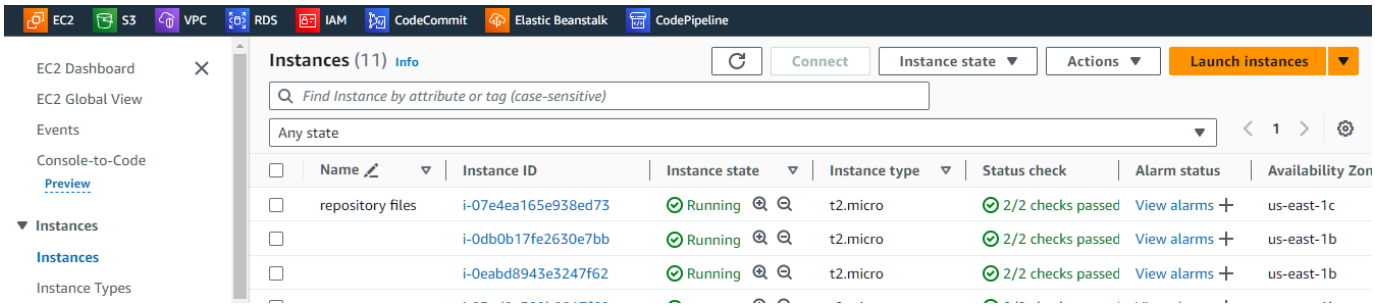
In this section of our workshop we will create an EC2 instance for our app layer and make all necessary software configurations so that the app can run. The app layer consists of a Node.js application that will run on port 4000. We will also configure our database with some data and tables.

Learning Objectives:

- Create App Tier Instance
- Configure Software Stack
- Configure Database Schema
- Test DB connectivity

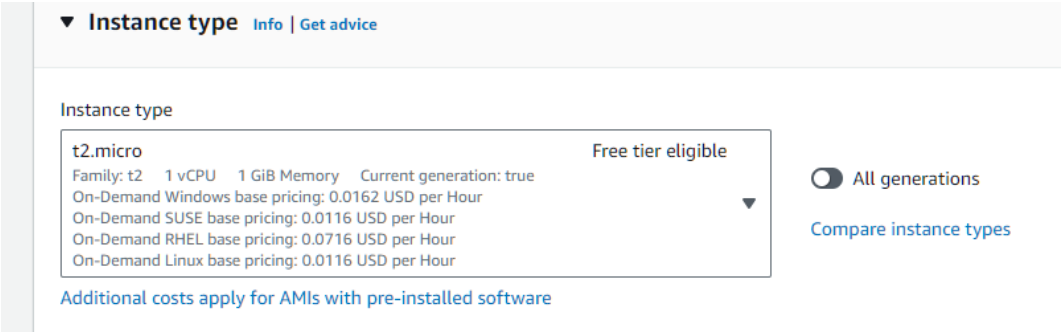
App Instance Deployment:

1. Navigate to the EC2 service dashboard and click on **Instances** on the left hand side. Then, click **Launch Instances**.

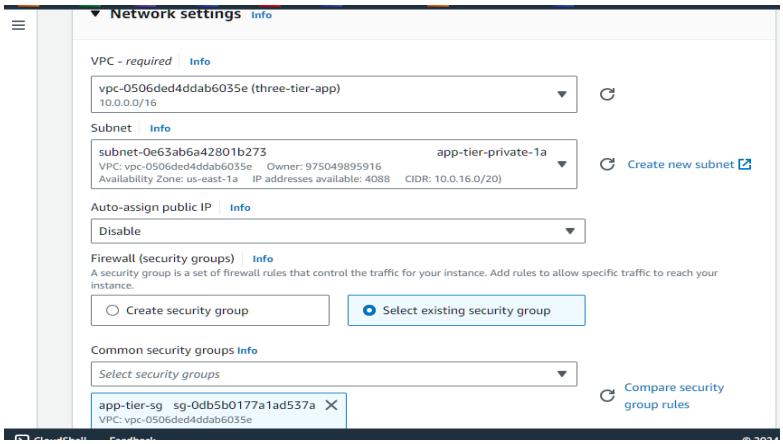


2. Select the first **Amazon Linux 2 AMI**

3. We'll be using the free tier eligible **T.2 micro** instance type. Select that and click **Next: Configure Instance Details**.

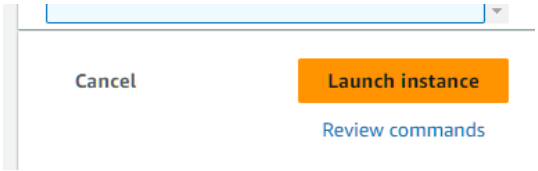


4. When configuring the instance details, make sure to select to correct **Network**, **subnet**, and **select existing group** we created. Note that this is the app layer, so use one of the private subnets we created for this layer.



5. We have to configure **IAM INSTANCE PROFILEROLE** in advanced settings & then hit create Instance.

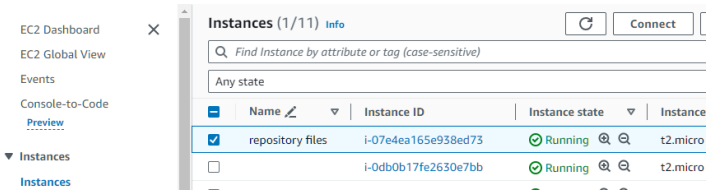
You'll be taken to a page where you can click launch instance, and you'll see the instance you just launched.

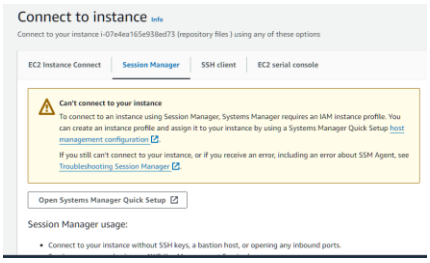


Connect to Instance:

1. Navigate to your list of running EC2 Instances by clicking on **Instances** on the left hand side of the EC2 dashboard. When the instance state is running, connect to your instance by clicking the checkmark box to the left of the instance, and click the connect button on the top right corner of the dashboard.Select the Session Manager tab, and click connect. This will open a new browser tab for you.

NOTE: If you get a message saying that you cannot connect via session manager, then check that your instances can route to your NAT gateways and verify that you gave the necessary permissions on the IAM role for the Ec2 instance.





2. When you first connect to your instance like this, you will be logged in as ssm-user which is the default user. Switch to ec2-user by executing the following command in the browser terminal:

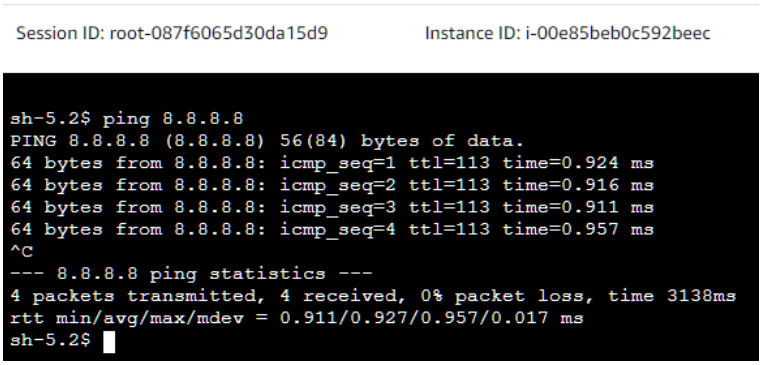
   sudo -su ec2-user

3. Let's take this moment to make sure that we are able to reach the internet via our NAT gateways. If your network is configured correctly up till this point, you should be able to ping the google DNS servers:

   ping 8.8.8.8

You should see a transmission of packets. Stop it by pressing cntrl c.

NOTE: If you can't reach the internet then you need to double check your route tables and subnet associations to verify if traffic is being routed to your NAT gateway!



Configure Database:

1. To configure the database of **Mysql** , firstly have to download package by using following **wget & rpm** commands **:**

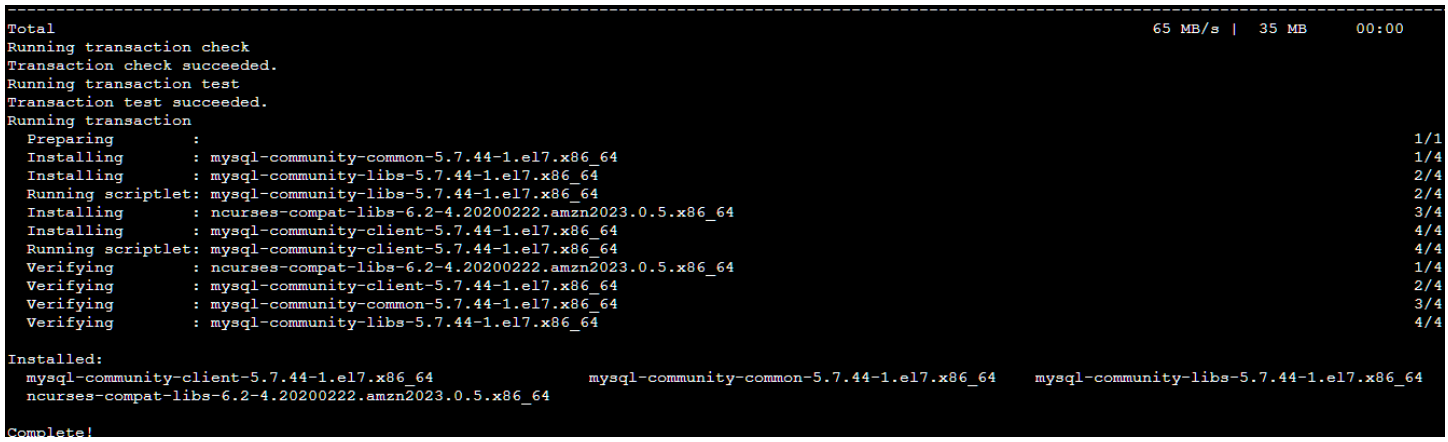
   steps for installing mysql :

   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



1. Start by downloading the MySQL CLI:

```
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:

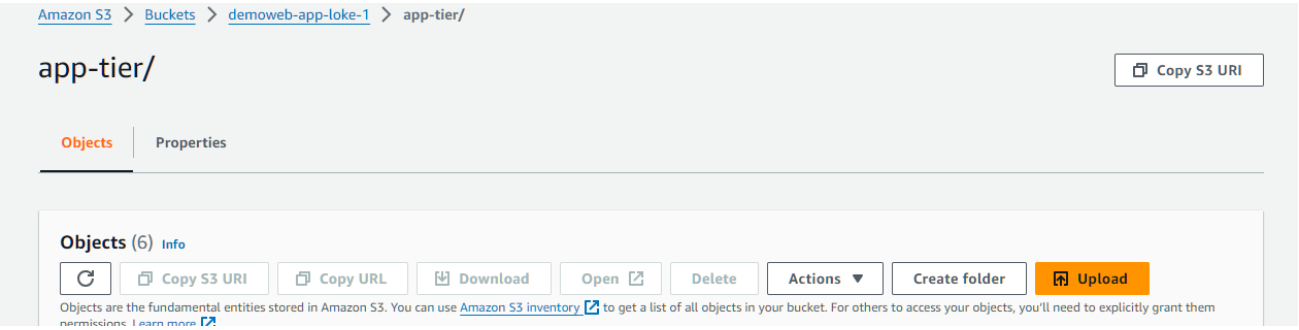
SELECT * FROM transactions;

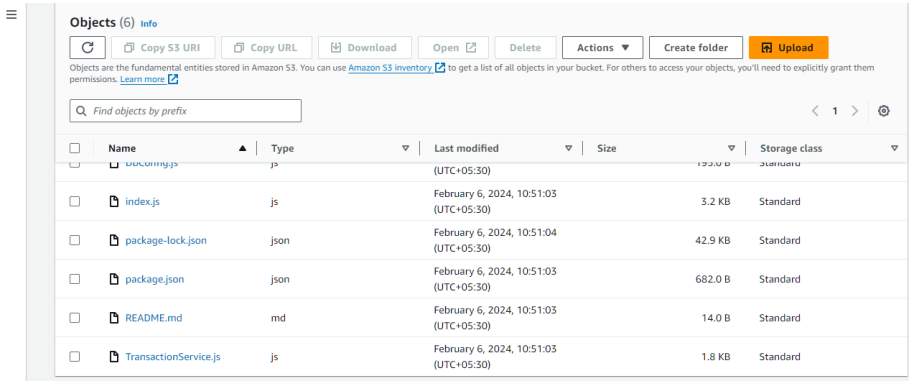
6. When finished, just type **exit** and hit enter to exit the MySQL client.

Configure App Instance:

1. 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.

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.





2. Upload the **app-tier** folder to the S3 bucket that you created in part 0.

3. 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

```
Run `npm audit` for details.
[ec2-user@ip-10-0-0-79 app-tier]$ pm2 start index.js
[PM2] Starting /home/ec2-user/app-tier/index.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	index	default	1.0.0	fork	20302	0s	0	online	0%	24.4mb	ec2-user	disabled

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

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

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

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

Part 4: Internal Load Balancing and Auto Scaling

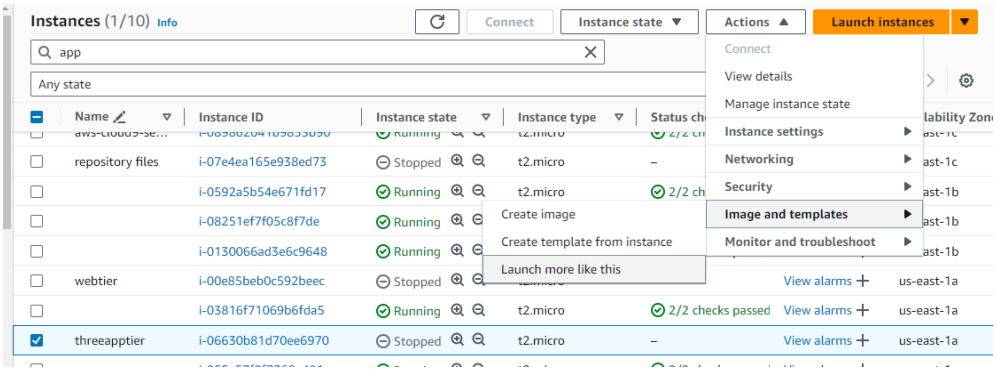
In this section of the workshop we will create an Amazon Machine Image (AMI) of the app tier instance we just created, and use that to set up autoscaling with a load balancer in order to make this tier highly available.

Learning Objectives:

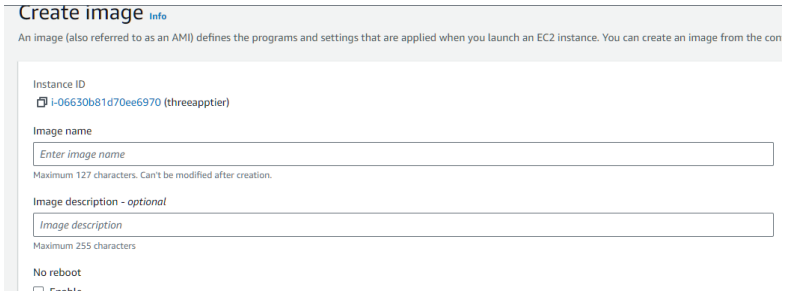
- Create an AMI of our App Tier
- Create a Launch Template
- Configure Autoscaling
- Deploy Internal Load Balancer

App Tier AMI

1. 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**.

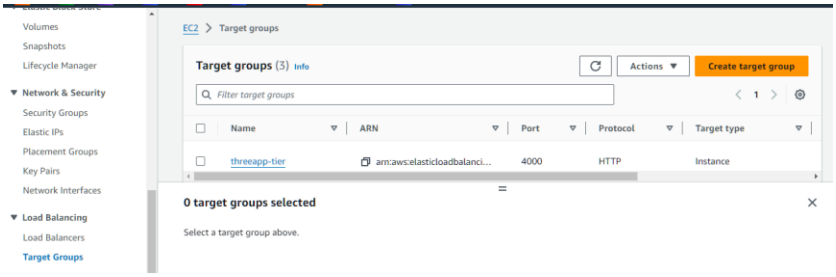


2. Give the image a name and description and then click **Create image**. This will take a few minutes, but if you want to monitor the status of image creation you can see it by clicking **AMIs** under **Images** on the left hand navigation panel of the EC2 dashboard.

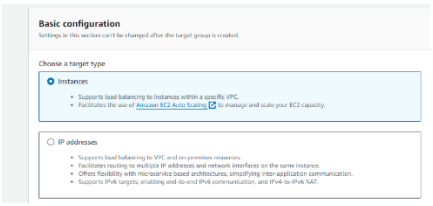


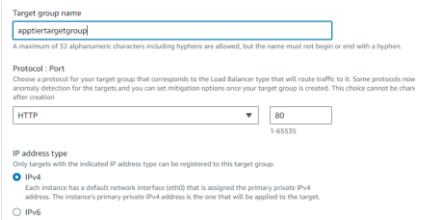
Target Group

1. While the AMI is being created, we can go ahead and create our target group to use with the load balancer. On the EC2 dashboard navigate to **Target Groups** under **Load Balancing** on the left hand side. Click on **Create Target Group**.

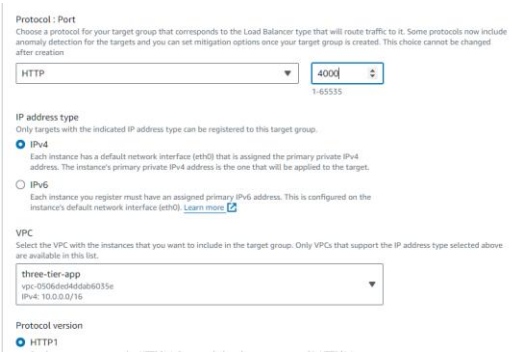


2. The purpose of forming this target group is to use with our load blancer so it may balance traffic across our private app tier instances. Select Instances as the target type and give it a name.

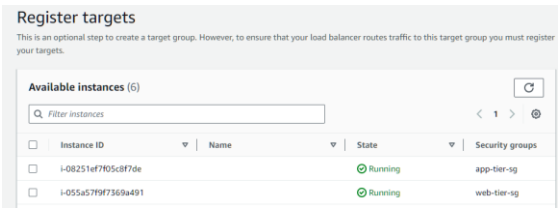


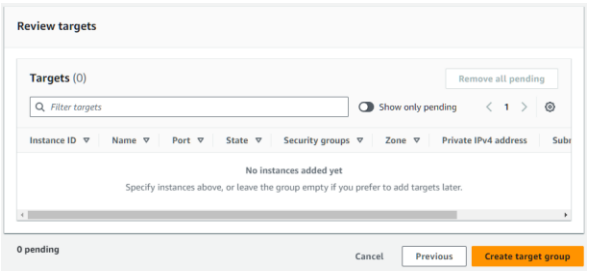


Then, set the protocol to **HTTP** and the port to 4000. Remember that this is the port our Node.ja app is running on. Select the VPC we've been using thus far, and then change the health check path to be **/health**. This is the health check endpoint of our app. Click **Next**.



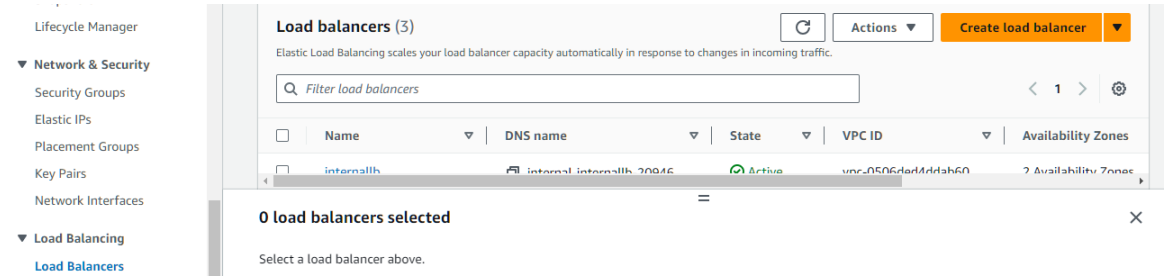
3. We are **NOT** going to register any targets for now, so just skip that step and create the target group.





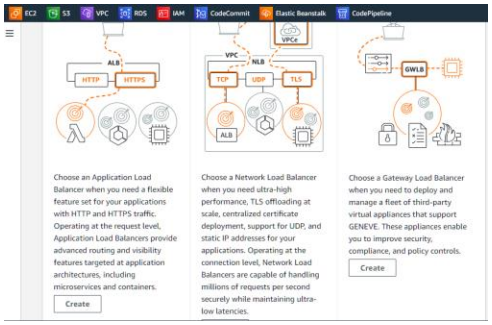
Internal Load Balancer:

1. On the left hand side of the EC2 dashboard select **Load Balancers** under **Load Balancing** and click **Create Load Balancer**.

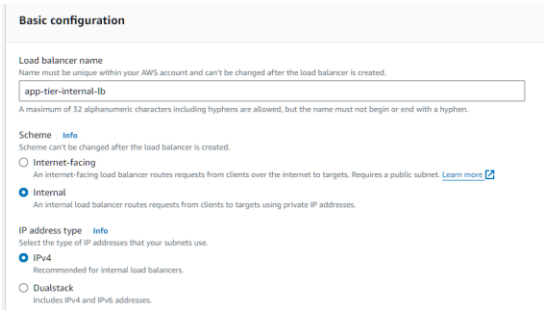


2. We'll be using an **Application Load Balancer** for our **HTTP** traffic so click the create button for that option.

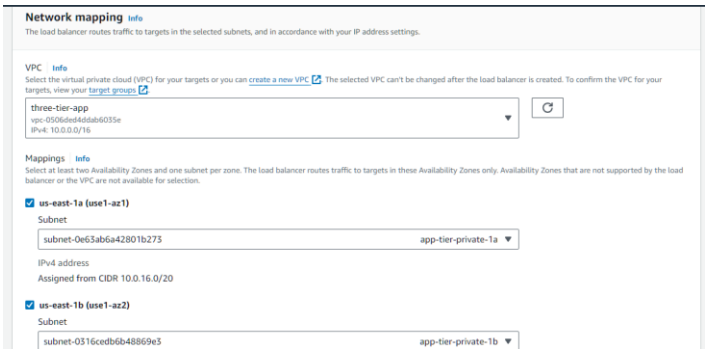
\



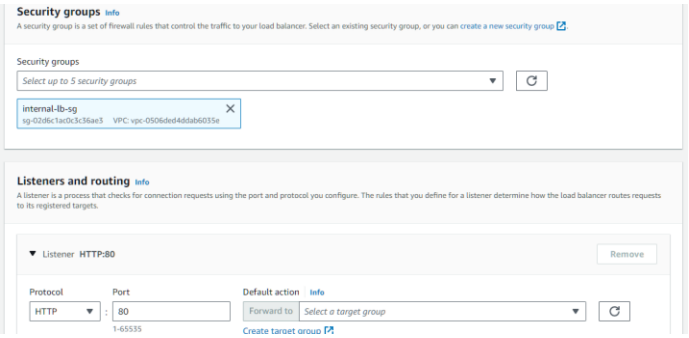
3. After giving the load balancer a name, be sure to select **internal** since this one will not be public facing, but rather it will route traffic from our web tier to the app tier.



Select the correct network configuration for VPC and private subnets

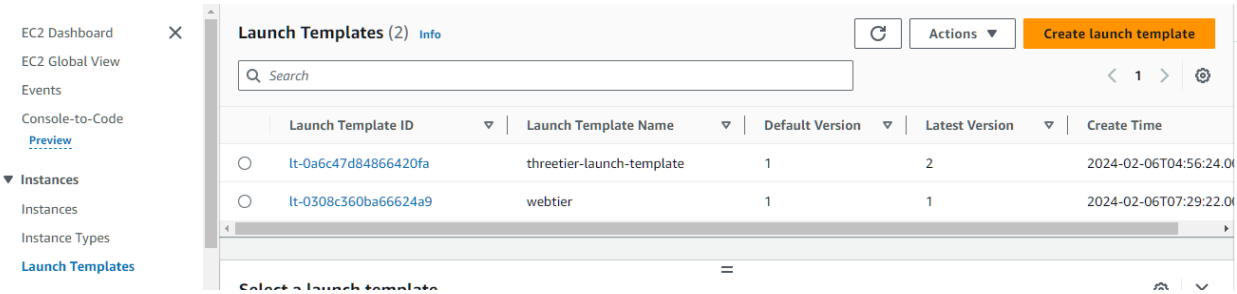


Select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our **target group** that we just created, so select it from the dropdown, and create the load balancer

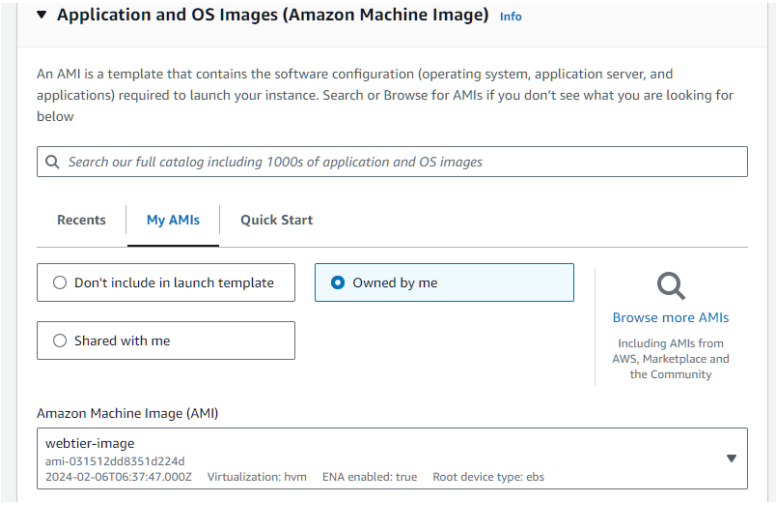


Launch Template

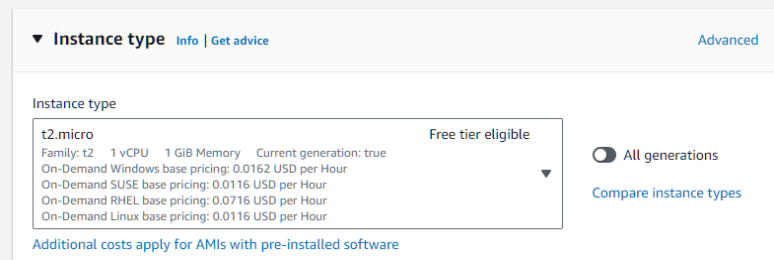
1. Before we configure Auto Scaling, we need to create a Launch template with the AMI we created earlier. On the left side of the EC2 dashboard navigate to **Launch Template** under **Instances** and click **Create Launch Template**.

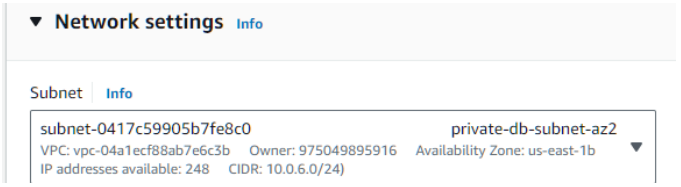


2. Name the Launch Template, and then under **Application and OS Images** include the app tier AMI you created.

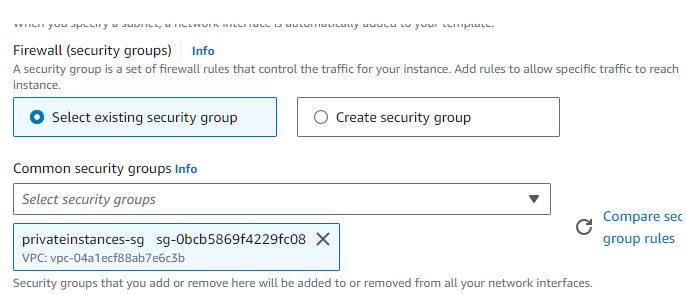


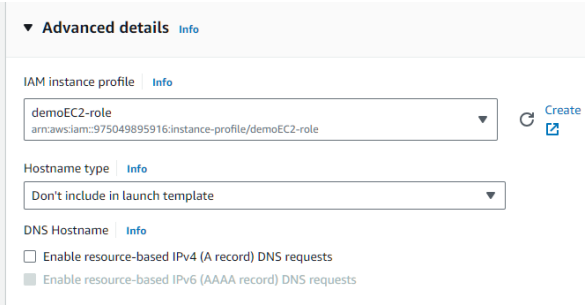
Under **Instance Type** select t2.micro. For **Key pair** and **Network Settings** don't include it in the template. We don't need a key pair to access our instances and we'll be setting the network information in the autoscaling group.





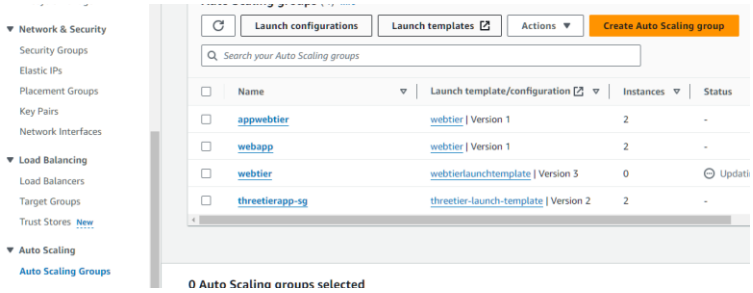
Set the correct security group for our app tier, and then under **Advanced details** use the same IAM instance profile we have been using for our EC2 instances.



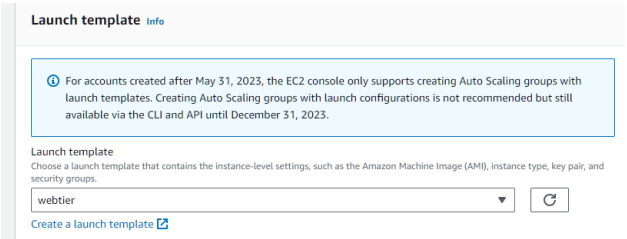


Auto Scaling

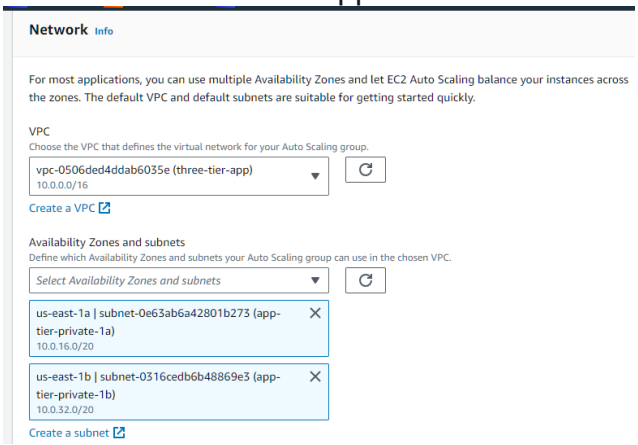
1. We will now create the Auto Scaling Group for our app instances. On the left side of the EC2 dashboard navigate to **Auto Scaling Groups** under **Auto Scaling** and click **Create Auto Scaling group**.



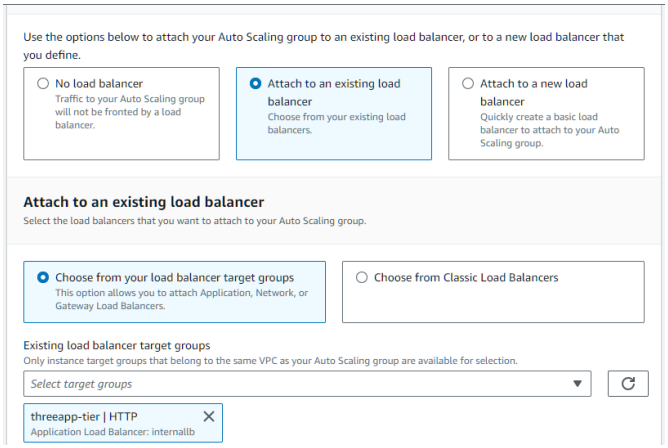
2. Give your Auto Scaling group a name, and then select the Launch Template we just created and click next.



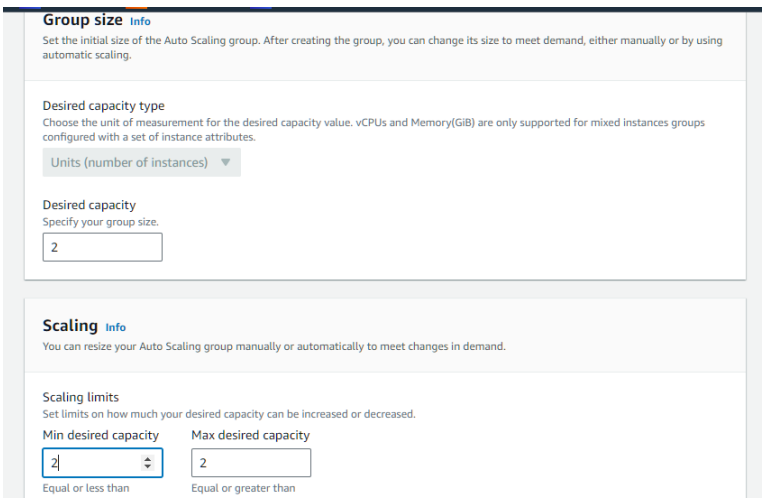
3. On the **Choose instance launch options** page set your VPC, and the private instance subnets for the app tier and continue to step 3.



For this next step, attach this Auto Scaling Group to the Load Balancer we just created by selecting the existing load balancer's target group from the dropdown. Then, click next



5. For **Configure group size and scaling policies**, set desired, minimum and maximum capacity to **2**. Click skip to review and then Create Auto Scaling Group.



You should now have your internal load balancer and autoscaling group configured correctly. You should see the autoscaling group spinning up 2 new app tier instances. If you wanted 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.

NOTE: Your original app tier instance is excluded from the ASG so you will see 3 instances in the EC2 dashboard. You can delete your original instance that you used to generate the app tier AMI but it's recommended to keep it around for troubleshooting purposes.

Part 5: Web Tier Instance Deployment

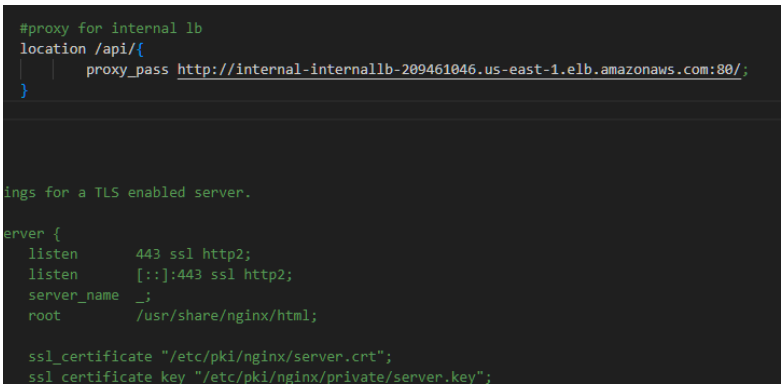
In this section we will deploy an EC2 instance for the web tier and make all necessary software configurations for the NGINX web server and React.js website.

Learning Objectives

- Update NGINX Configuration Files
- Create Web Tier Instance
- Configure Software Stack

Update Config File

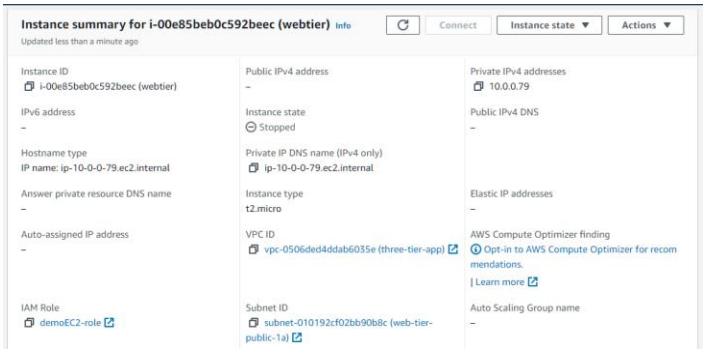
Before we create and configure the web instances, open up the **application-code/nginx.conf** file from the repo we downloaded. Scroll down to **line 58** and replace [INTERNAL-LOADBALANCER-DNS] with your internal load balancer's DNS entry. You can find this by navigating to your internal load balancer's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

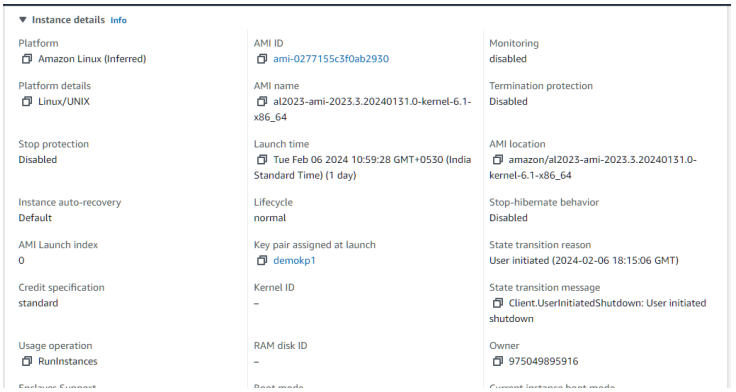


Then, upload this file **and** the **application-code/web-tier** folder to the s3 bucket you created for this lab.

Web Instance Deployment

1. Follow the same instance creation instructions we used for the App Tier instance in **Part 3: App Tier Instance Deployment**, with the exception of the subnet. We will be provisioning this instance in one of our **public subnets**. Make sure to select the correct network components, security group, and IAM role. **This time, auto-assign a public ip** on the **Configure Instance Details page**. Remember to tag the instance with a name so we can identify it more easily.





Connect to Instance

1. Follow the same steps you used to connect to the app instance and change the user to **ec2-user**. Test connectivity here via ping as well since this instance should have internet connectivity:

```
sudo -su ec2-user

ping 8.8.8.8
```

Note: If you don't see a transfer of packets then you'll need to verify your route tables attached to the subnet that your instance is deployed in.

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

Installation instructions for Amazon Linux 2023

- Create the /etc/ssl/nginx/ directory:
  - **sudo mkdir -p /etc/ssl/nginx**
- Install prerequisite packages:
  - **sudo yum install ca-certificates**
- Copy the above two files to the Amazon Linux server into `/etc/ssl/nginx/` directory. Use your SCP client or other secure file transfer tools.
- Add NGINX Plus repository by downloading the file plus-amazonlinux2023.repo to /etc/yum.repos.d:
  - **sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/plus-amazonlinux2023.repo**
- If you have modsecurity subscription, add modsecurity repository by downloading the file modsecurity-amazonlinux2023.repo to /etc/yum.repos.d:
  - **sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/modsecurity-amazonlinux2023.repo**

Update the Package Repository

Before installing Nginx, it's a good practice to update the package repository on your Amazon Linux **instance. To do this, run the following command:**

```
sudo dnf update -y
```

Install Nginx

Next, we need to install Nginx on Amazon Linux 2023. To do this, run the following command:

Copy Codesudo dnf install nginx -y

Start Nginx Once the installation is complete, start the Nginx service using the following command:

```
sudo systemctl status nginx
```

If Nginx is running correctly, you will see output similar to the following:

```
[ec2-user@ip-10-0-0-79 nginx]$ sudo systemctl status nginx
● nginx.service - The nginx HTTP and reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: disabled)
     Active: active (running) since Wed 2024-02-07 05:41:07 UTC; 29min ago
    Process: 2132 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
    Process: 2144 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
    Process: 2149 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
   Main PID: 2150 (nginx)
      Tasks: 2 (limit: 1114)
     Memory: 4.8M
        CPU: 62ms
     CGroup: /system.slice/nginx.service
             ├─2150 "nginx: master process /usr/sbin/nginx"
             └─2151 "nginx: worker process"

Feb 07 05:41:07 ip-10-0-0-79.ec2.internal systemd[1]: Starting nginx.service - The nginx HTTP and reverse proxy server...
Feb 07 05:41:07 ip-10-0-0-79.ec2.internal nginx[2144]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
Feb 07 05:41:07 ip-10-0-0-79.ec2.internal nginx[2144]: nginx: configuration file /etc/nginx/nginx.conf test is successful
Feb 07 05:41:07 ip-10-0-0-79.ec2.internal systemd[1]: Started nginx.service - The nginx HTTP and reverse proxy server.
```

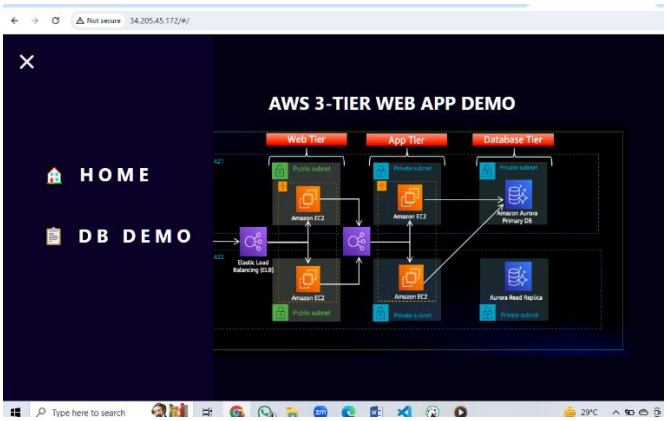
To make sure Nginx has permission to access our files execute this command:

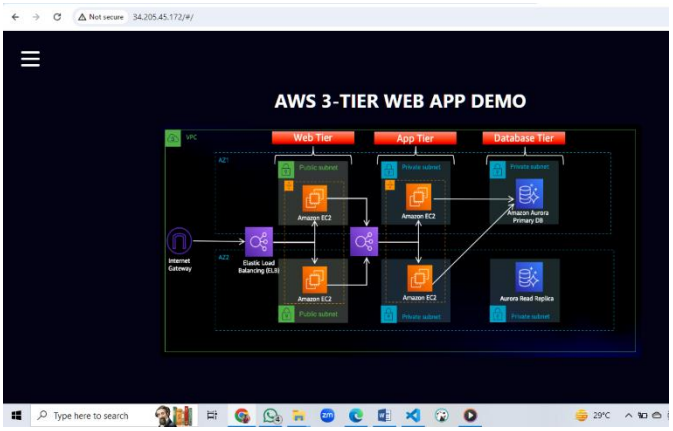
```
chmod -R 755 /home/ec2-user
```

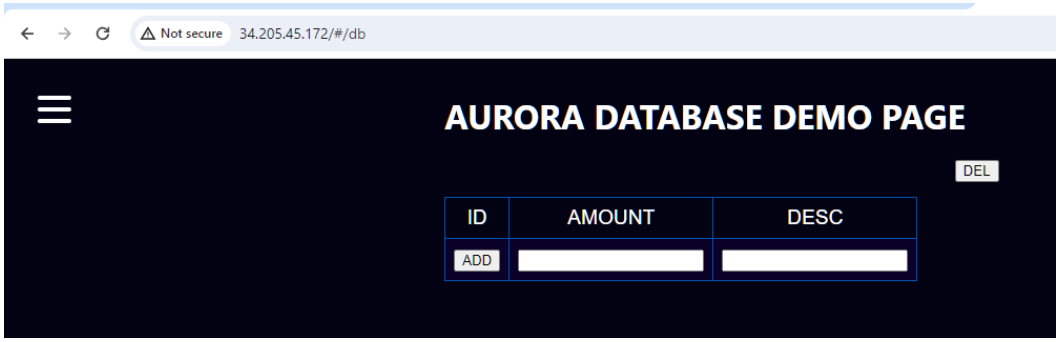
And then to make sure the service starts on boot, run this command:

```
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 6: External Load Balancer and Auto Scaling

In this section of the workshop 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.

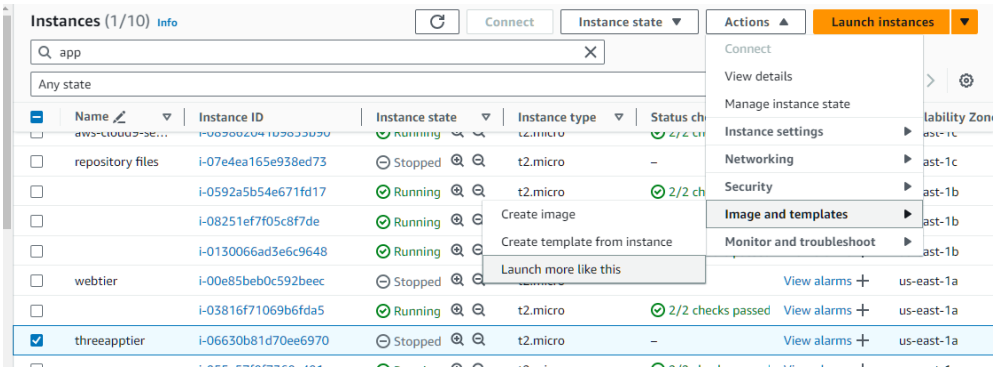
Learning Objectives:

- Create an AMI of our Web Tier
- Create a Launch Template
- Configure Auto Scaling
- Deploy External Load Balancer

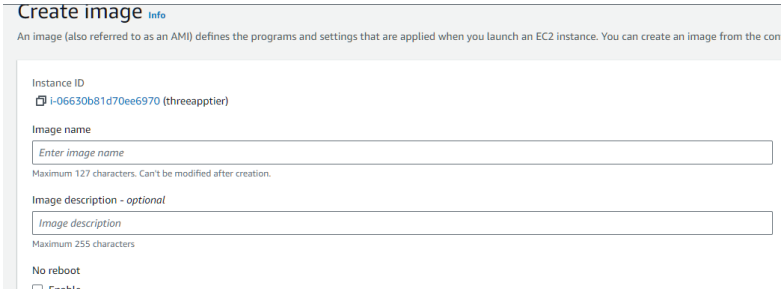
Web Tier AMI

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

2. we created and under **Actions** select **Image and templates**. Click **Create Image**.

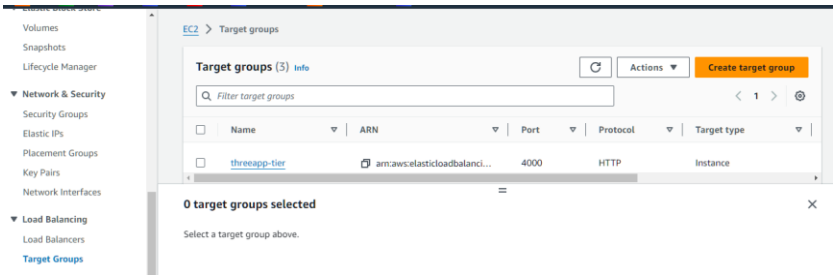


3. Give the image a name and description and then click **Create image**. This will take a few minutes, but if you want to monitor the status of image creation you can see it by clicking **AMIs** under **Images** on the left hand navigation panel of the EC2 dashboard.

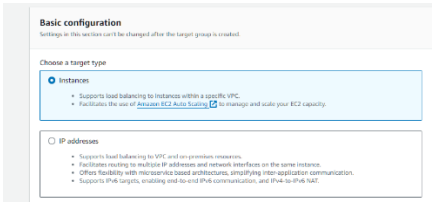


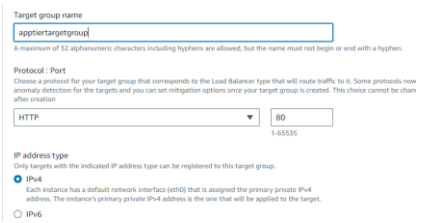
Target Group

2. While the AMI is being created, we can go ahead and create our target group to use with the load balancer. On the EC2 dashboard navigate to **Target Groups** under **Load Balancing** on the left hand side. Click on **Create Target Group**.

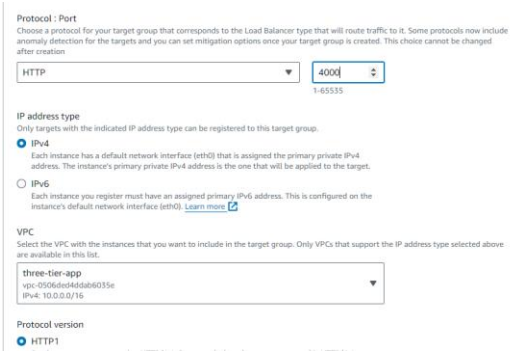


3. The purpose of forming this target group is to use with our load blancer so it may balance traffic across our private app tier instances. Select Instances as the target type and give it a name.

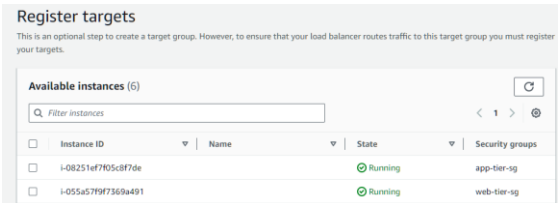


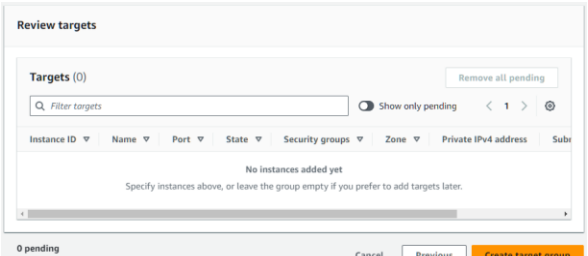


Then, set the protocol to **HTTP** and the port to 4000. Remember that this is the port our Node.ja app is running on. Select the VPC we've been using thus far, and then change the health check path to be **/health**. This is the health check endpoint of our app. Click **Next**.



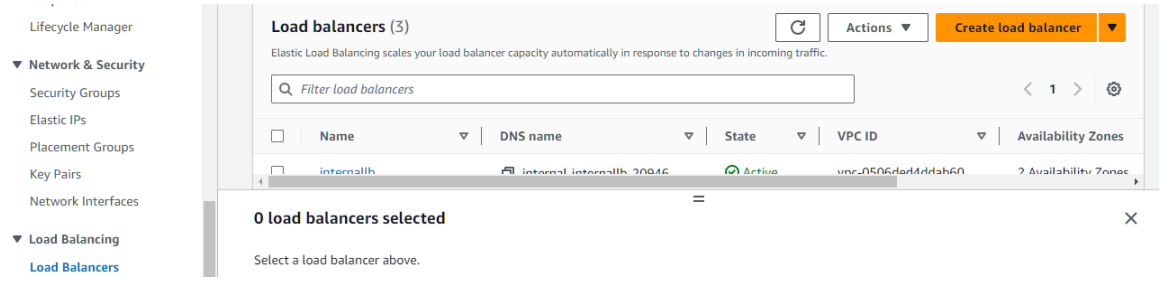
4. We are **NOT** going to register any targets for now, so just skip that step and create the target group.





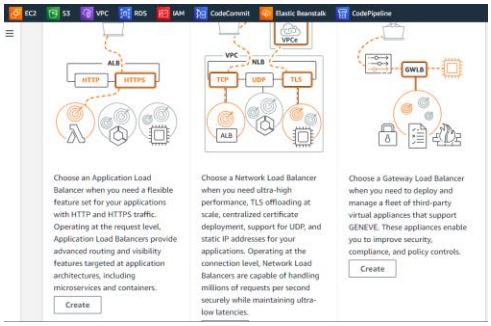
Internal Load Balancer:

2. On the left hand side of the EC2 dashboard select **Load Balancers** under **Load Balancing** and click **Create Load Balancer**.

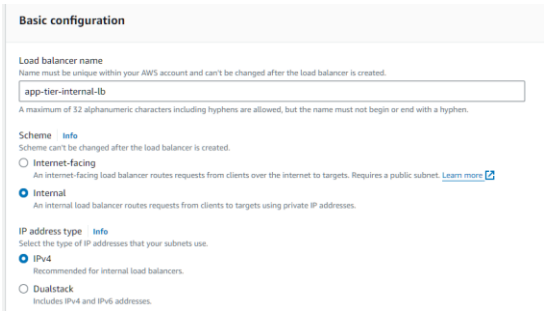


3. We'll be using an **Application Load Balancer** for our **HTTP** traffic so click the create button for that option.

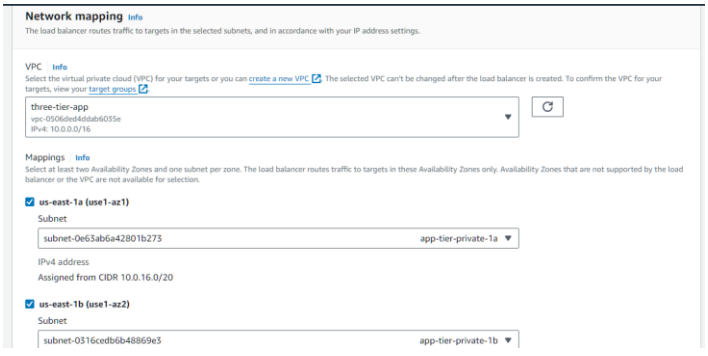
\



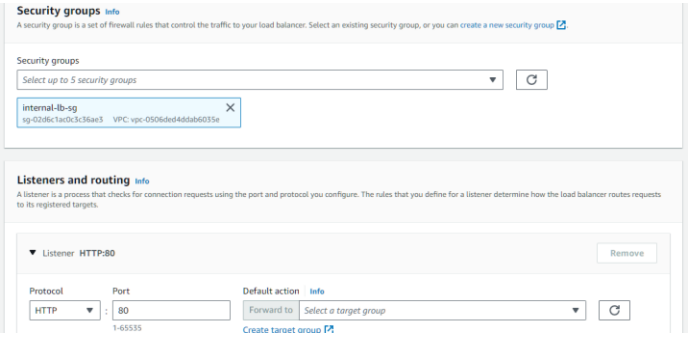
4. After giving the load balancer a name, be sure to select **internal** since this one will not be public facing, but rather it will route traffic from our web tier to the app tier.



Select the correct network configuration for VPC and private subnets

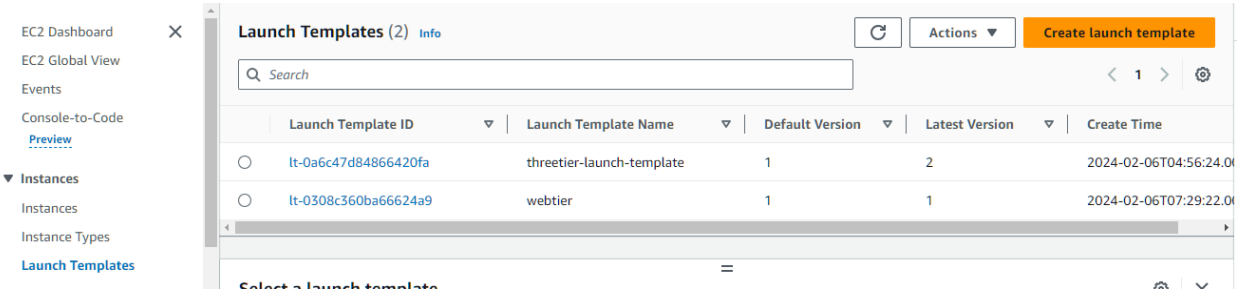


Select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our **target group** that we just created, so select it from the dropdown, and create the load balancer

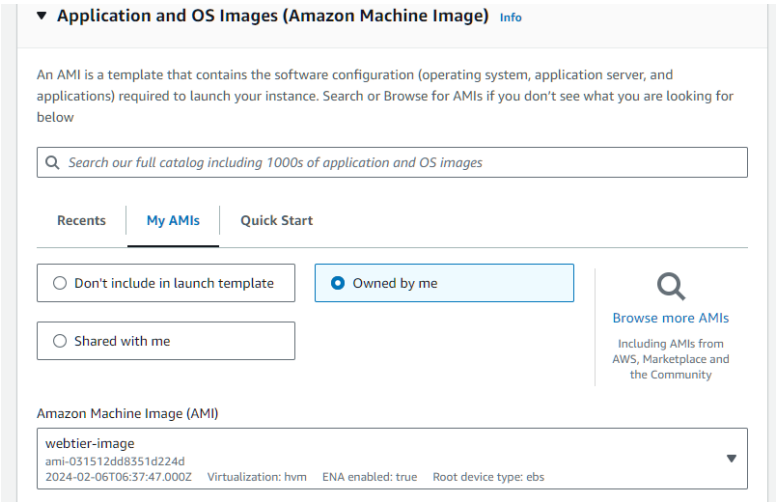


Launch Template

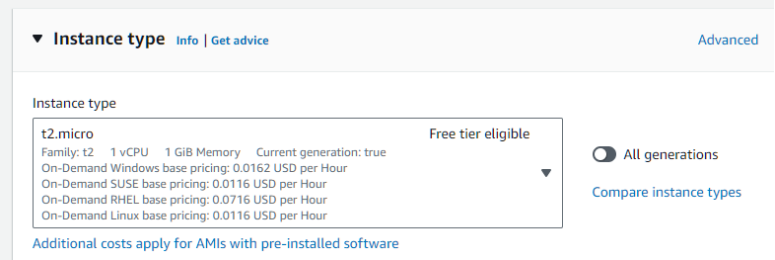
2. Before we configure Auto Scaling, we need to create a Launch template with the AMI we created earlier. On the left side of the EC2 dashboard navigate to **Launch Template** under **Instances** and click **Create Launch Template**.

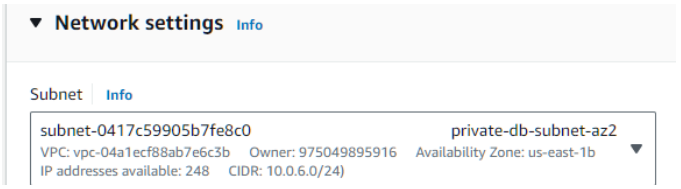


3. Name the Launch Template, and then under **Application and OS Images** include the app tier AMI you created.

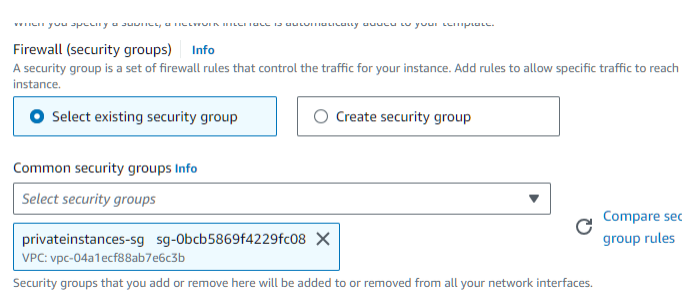


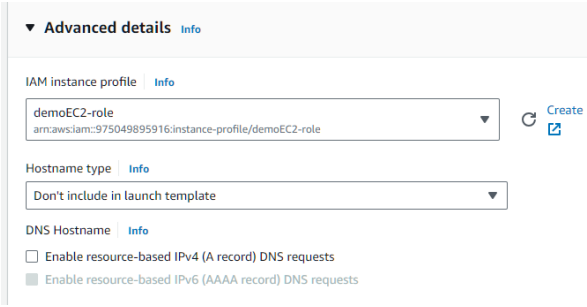
Under **Instance Type** select t2.micro. For **Key pair** and **Network Settings** don't include it in the template. We don't need a key pair to access our instances and we'll be setting the network information in the autoscaling group.





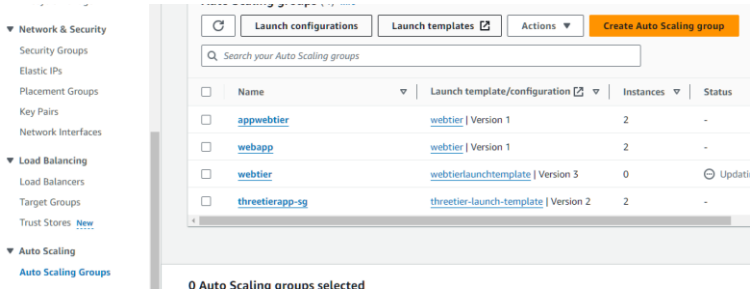
Set the correct security group for our app tier, and then under **Advanced details** use the same IAM instance profile we have been using for our EC2 instances.



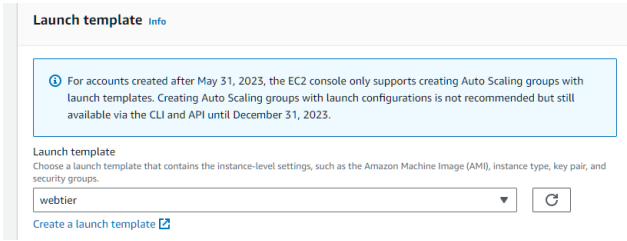


Auto Scaling

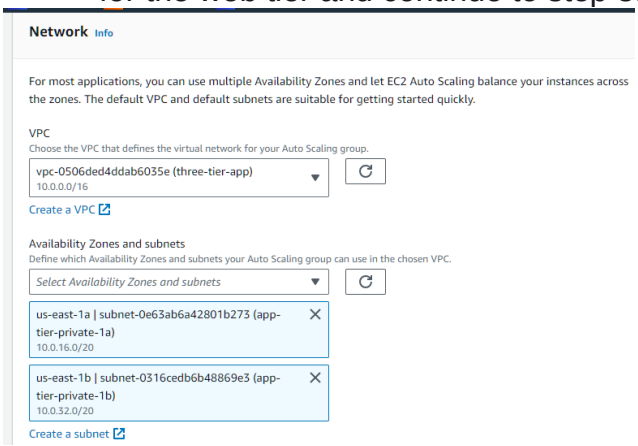
2. We will now create the Auto Scaling Group for our app instances. On the left side of the EC2 dashboard navigate to **Auto Scaling Groups** under **Auto Scaling** and click **Create Auto Scaling group**.



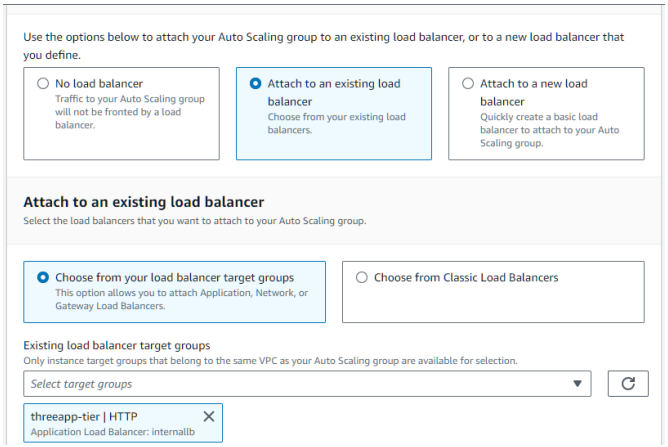
3. Give your Auto Scaling group a name, and then select the Launch Template we just created and click next.



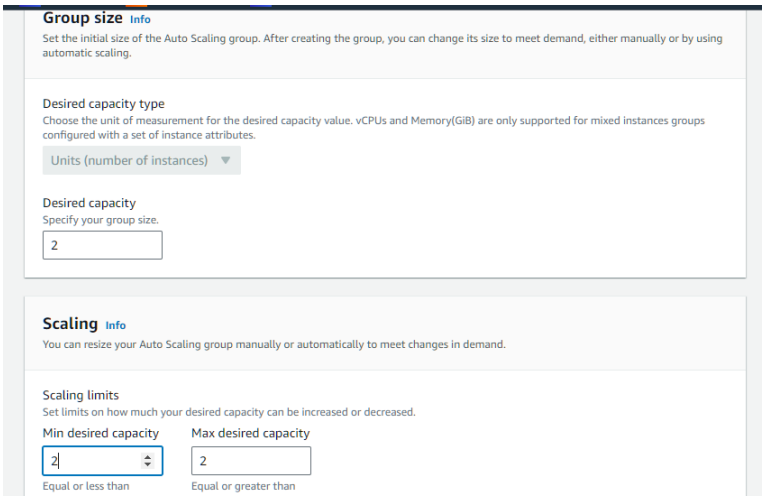
5. On the **Choose instance launch options** page set your VPC, and the public instance subnets for the web tier and continue to step 3.



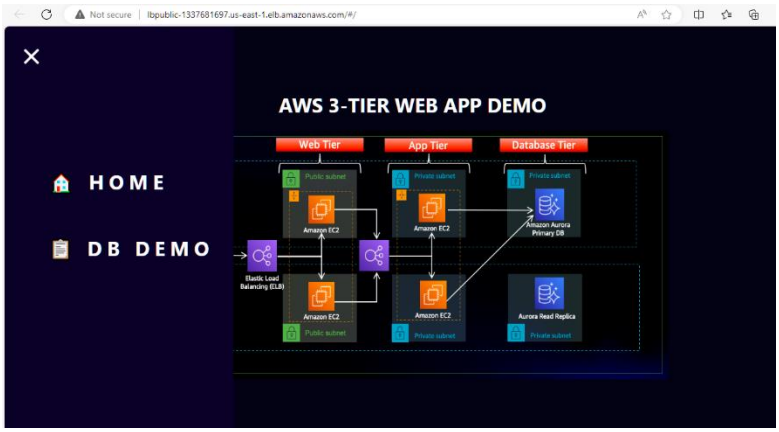
For this next step, attach this Auto Scaling Group to the Load Balancer we just created by selecting the existing load balancer's target group from the dropdown. Then, click next

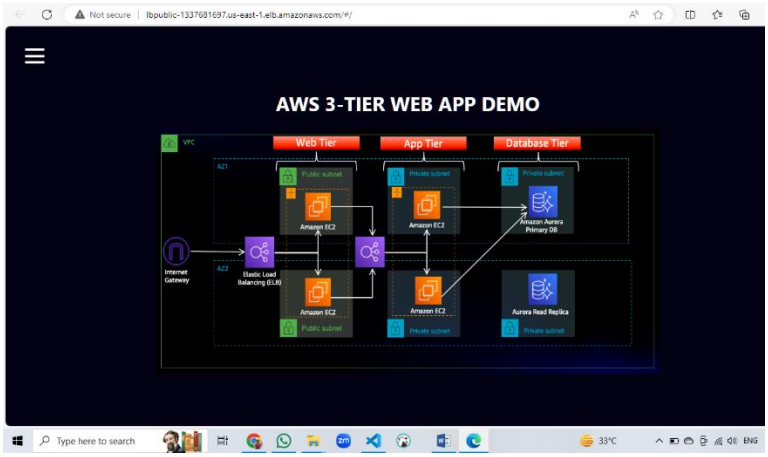


6. For **Configure group size and scaling policies**, set desired, minimum and maximum capacity to **2**. Click skip to review and then Create Auto Scaling Group.

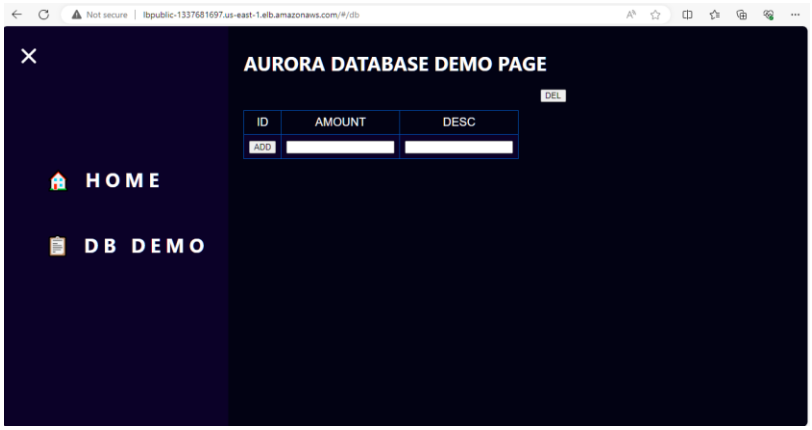


You should now have your external load balancer and autoscaling group configured correctly. You should see the autoscaling group spinning up 2 new web tier instances. If you wanted to test if this is working correctly, you can delete one of your new instances manually and wait to see if a new instance 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.





\



NOTE: Again, your original web tier instance is excluded from the ASG so you will see 3 instances in the EC2 dashboard. You can delete your original instance that you used to generate the web tier AMI but it's recommended to keep it around for troubleshooting purposes.

Congrats! You've Implemented a 3 Tier Web Architecture!

Appendix

DOCUMENTATION:

- NVM (Node Version Manager)
  - https://github.com/nvm-sh/nvm
- Node.js
  - https://nodejs.org
- PM2
  - https://pm2.keymetrics.io/docs/usage/pm2-doc-single-page/
- NGINX
  - https://docs.nginx.com/?_ga=2.4975283.1227355032.1619285306-501395053.1619109877
- Amazon Linux Extras
  - https://aws.amazon.com/amazon-linux-2/faqs/