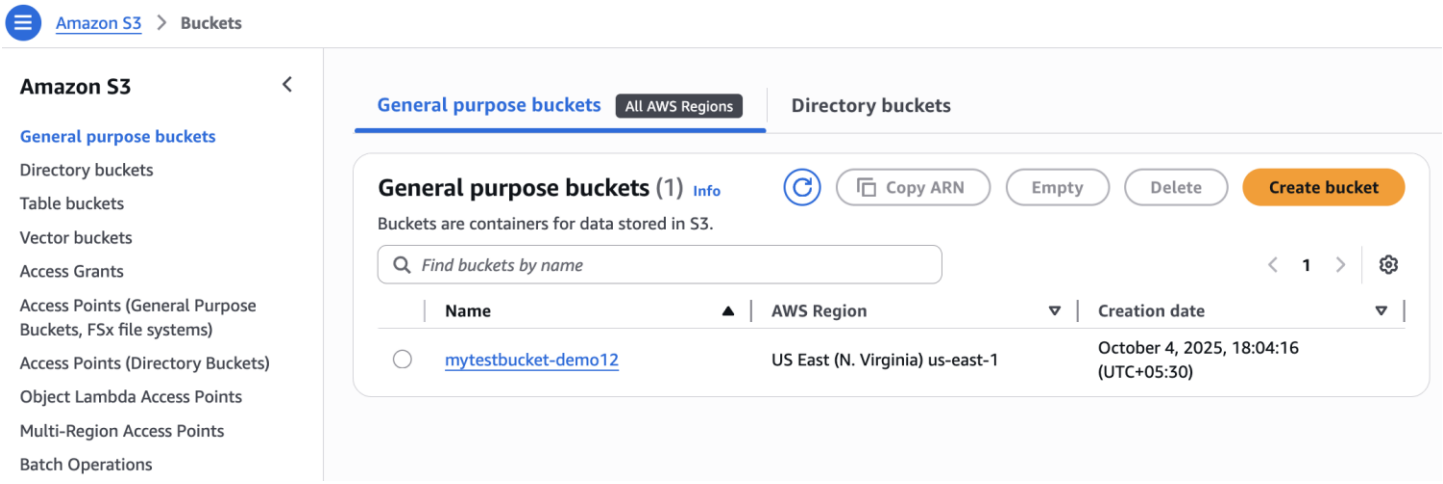
AWS 3 Tier Web Architecture Practical

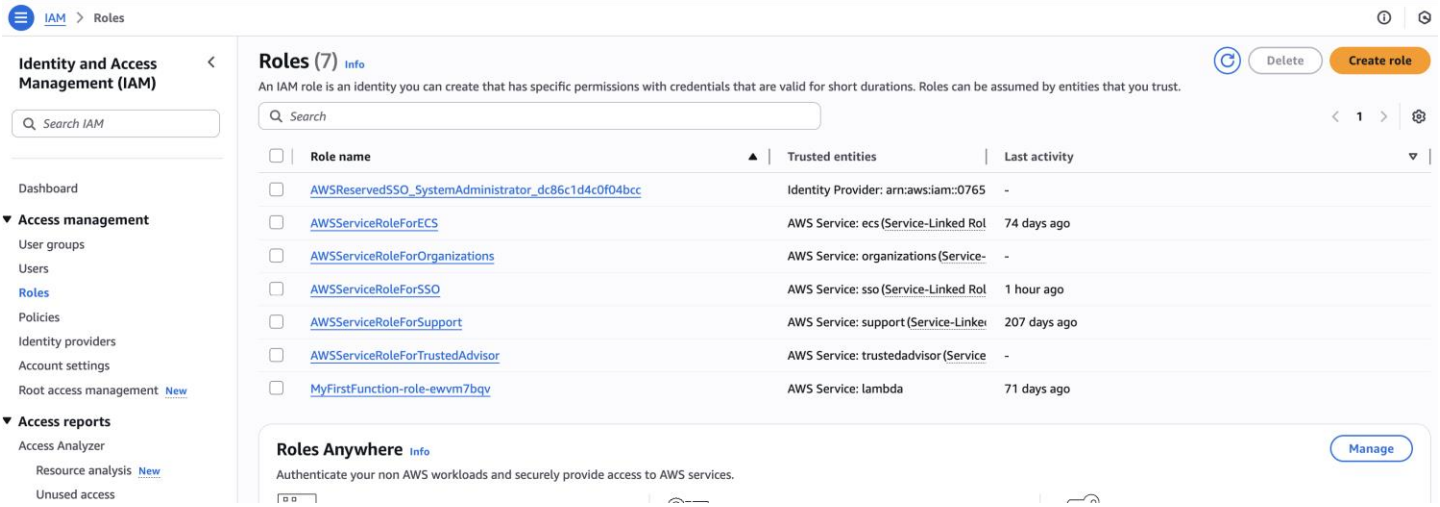
Step-1: Create S3 bucket

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

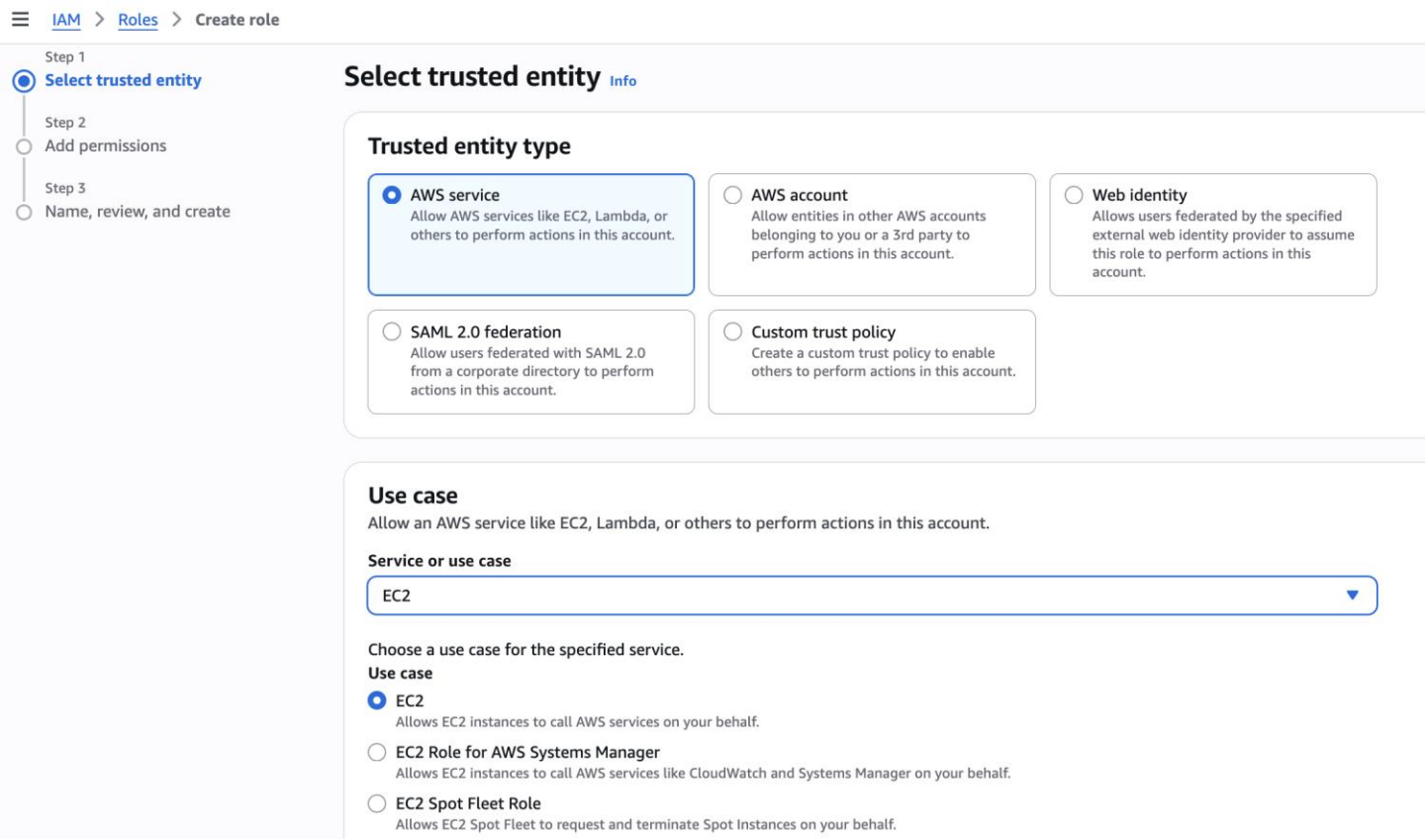


Step-2: IAM EC2 Instance Role Creation

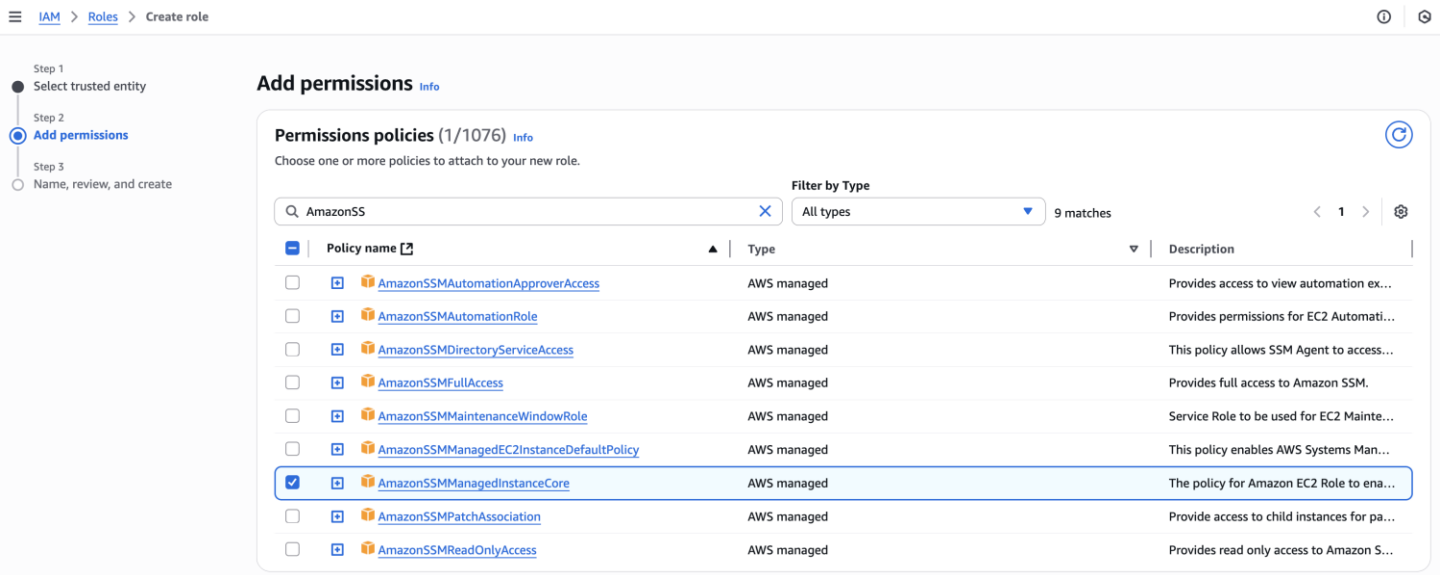
- Navigate to the IAM dashboard in the AWS console and create an EC2 role.

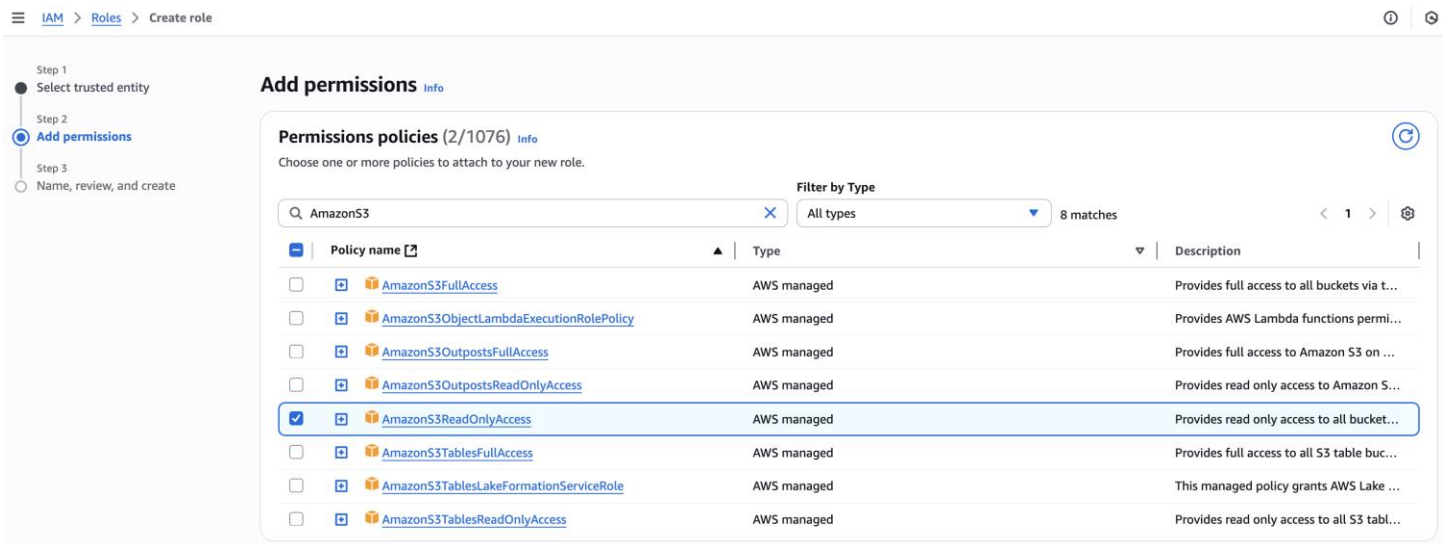


- Select EC2 as the trusted entity and click on next.



- 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**





IAM > Roles > Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

Myec2demo-role

Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.

Allows EC2 instances to call AWS services on your behalf.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^*();"'`

Step 1: Select trusted entities

Trust policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
```

Step 2: Add permissions

Edit

Permissions policy summary

Policy name	Type	Attached as
AmazonS3ReadOnlyAccess	AWS managed	Permissions policy
AmazonSSMManagedInstanceCore	AWS managed	Permissions policy

Step 3: Add tags

Add tags - *optional* Info

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

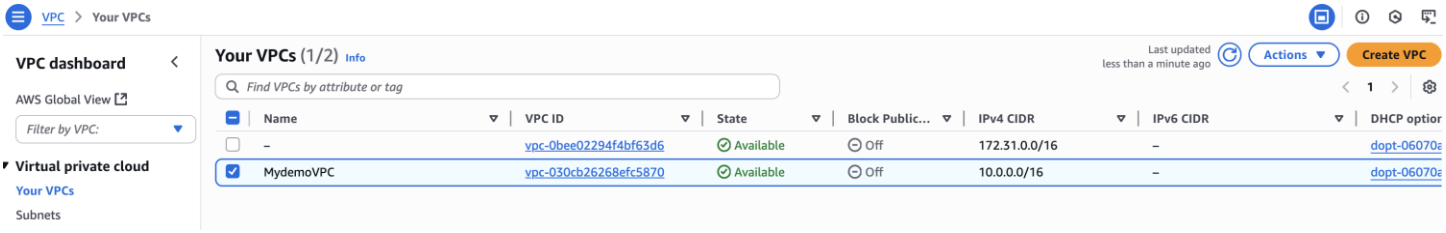
Add new tag

You can add up to 50 more tags.

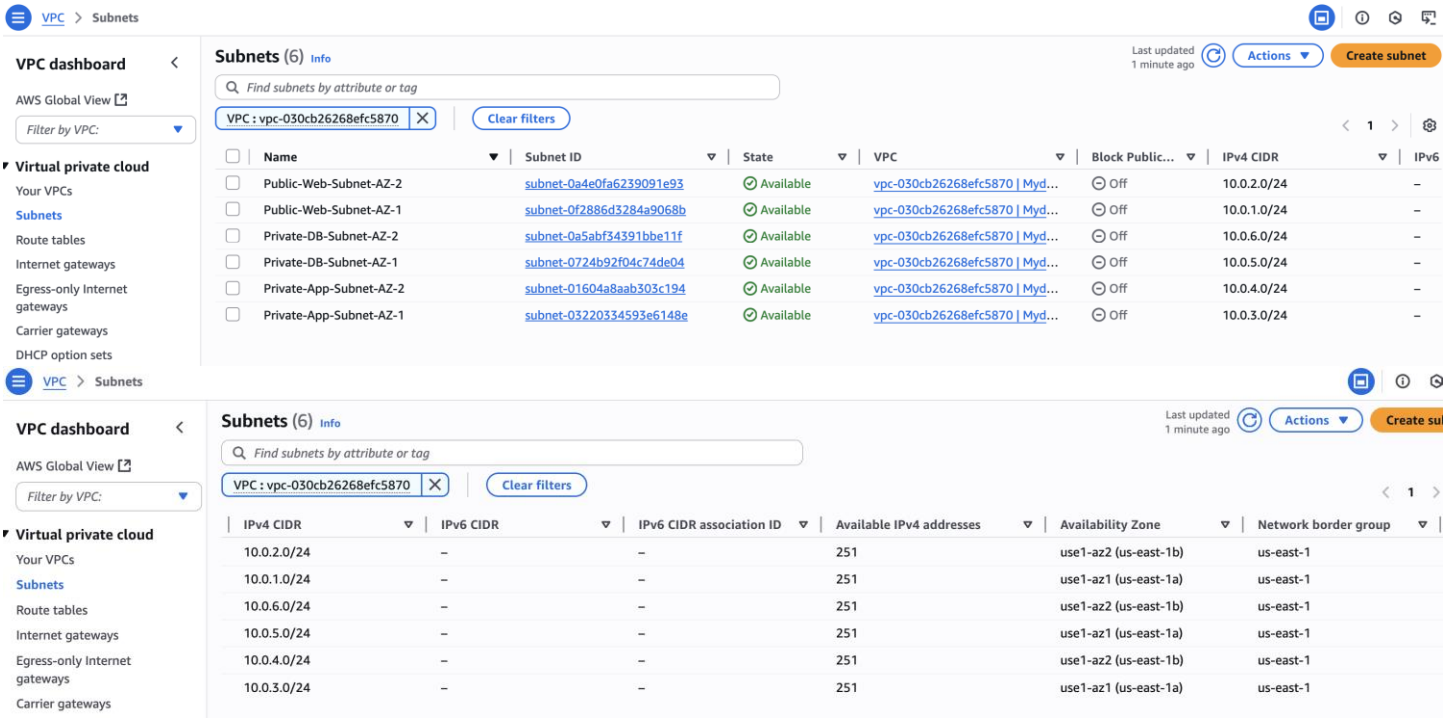
Cancel Previous Create role

Step-3: Create VPC & Subnets

- Navigate to the VPC dashboard in the AWS console and navigate to Your VPCs on the left side.

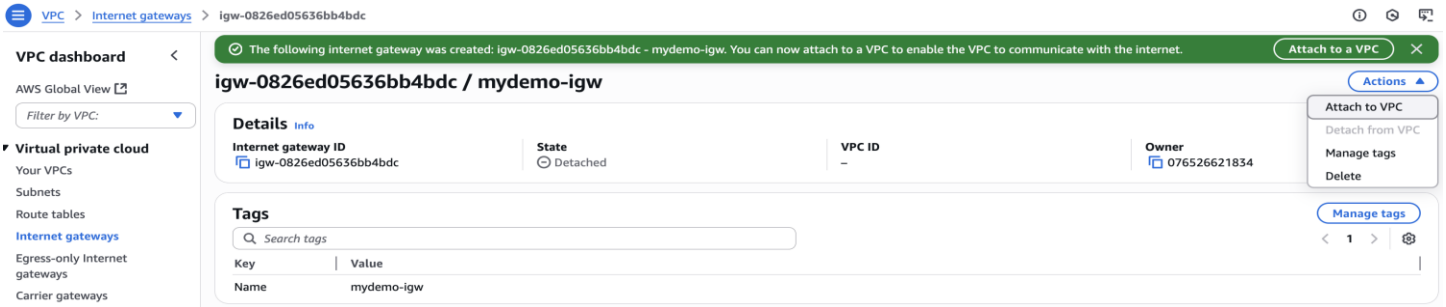


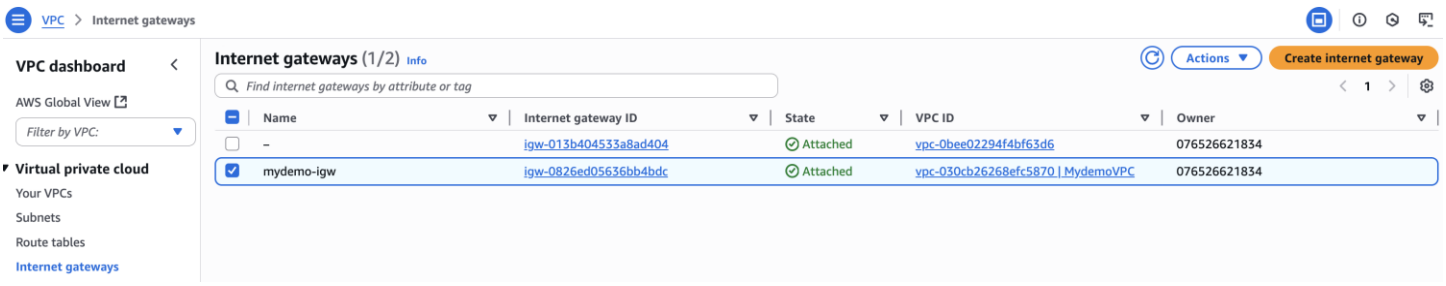
- Next, create your subnets by navigating to Subnets on the left side of the dashboard and clicking Create subnet. We will need six subnets across two availability zones. That means that three subnets will be in one availability zone, and three subnets will be in another zone.



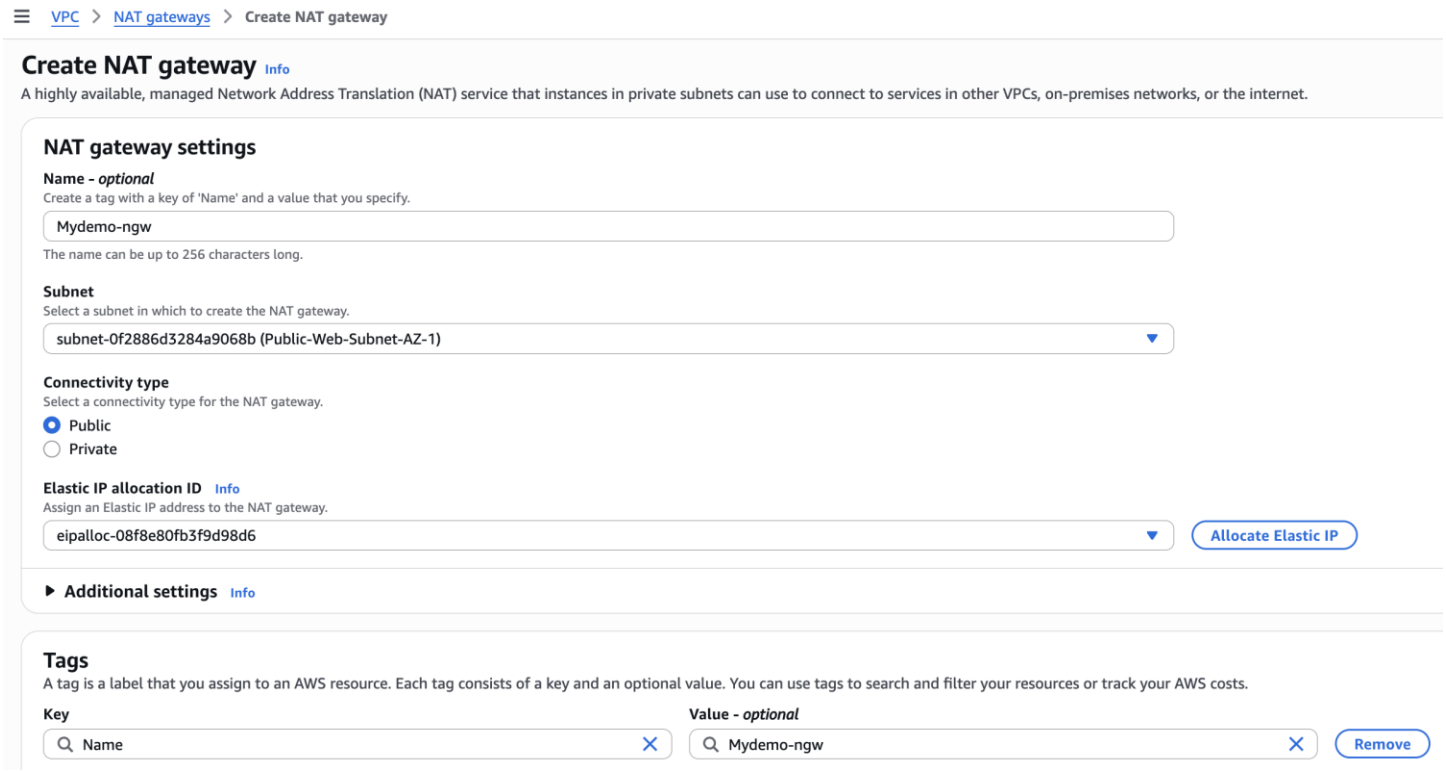
Setp-4: Create Internet Gateway and configure routing

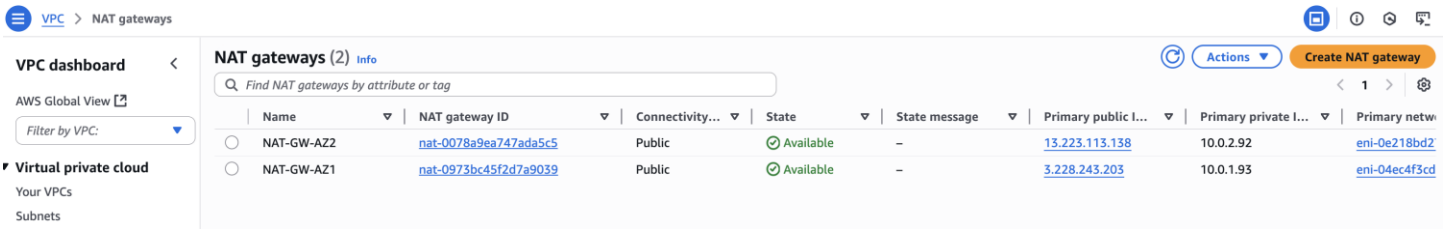
- To give the public subnets in our VPC internet access we will have to create and attach an Internet Gateway. On the left-hand side of the VPC dashboard, select Internet Gateway. Create your internet gateway by simply giving it a name and clicking Create internet gateway. After creating the internet gateway, attach it to your VPC.



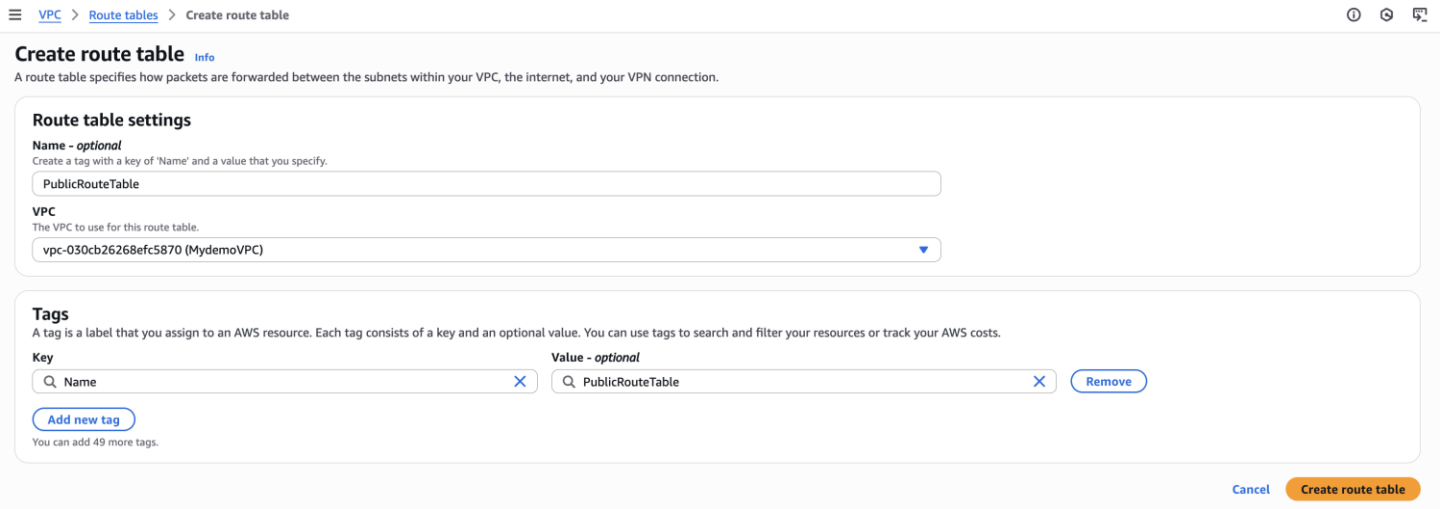


- For our instances in the app layer private subnet to be able to access the internet they will need to go through a NAT Gateway. For high availability, you'll deploy one NAT gateway in each of your public subnets. Navigate to NAT Gateways on the left side of the current dashboard and click Create NAT Gateway. Fill in the Name, choose one of the public subnets you created, and then allocate an Elastic IP. Click Create NAT gateway. Do the same for both public subnets.

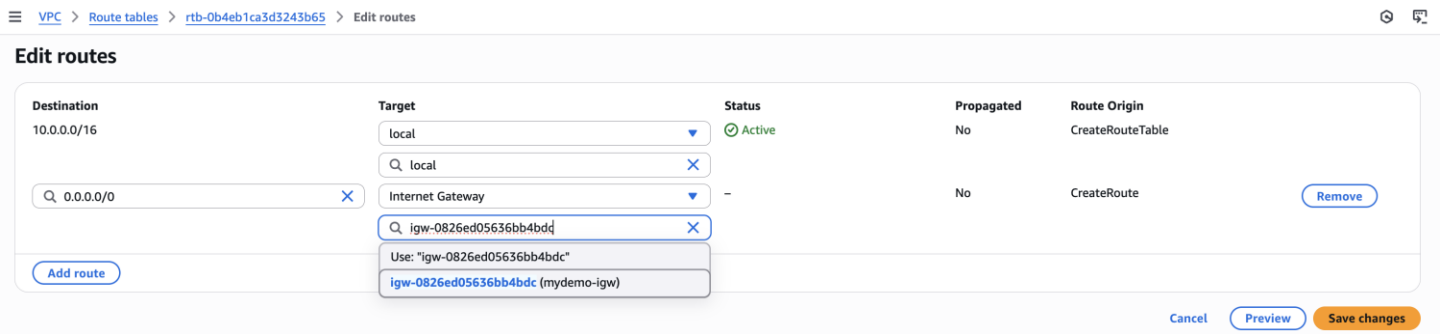




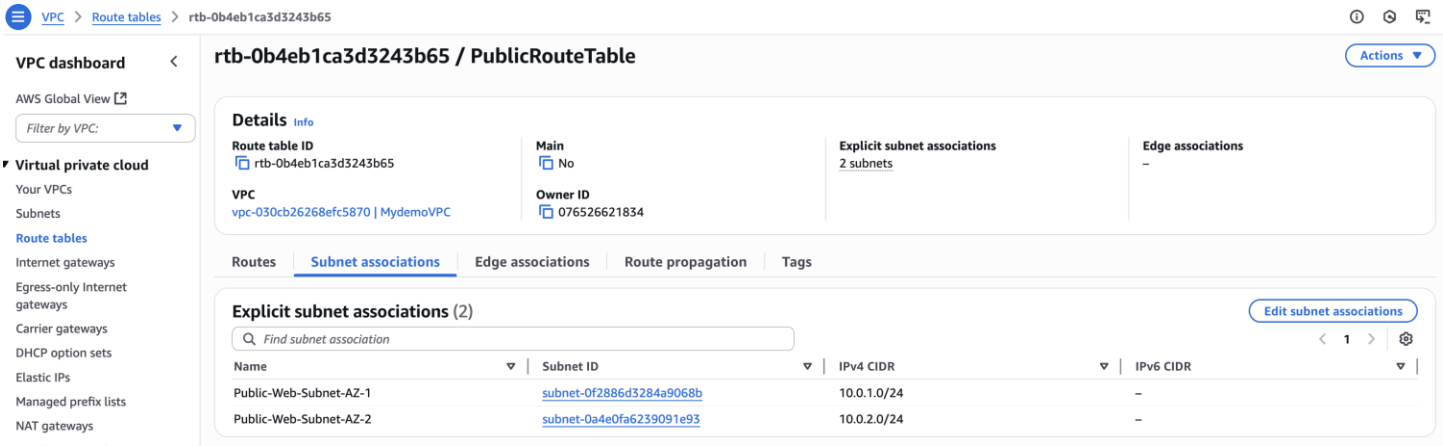
- Navigate to Route Tables on the left side of the VPC dashboard and click Create route table First, let's create one route table for the web layer public subnets and name it accordingly.



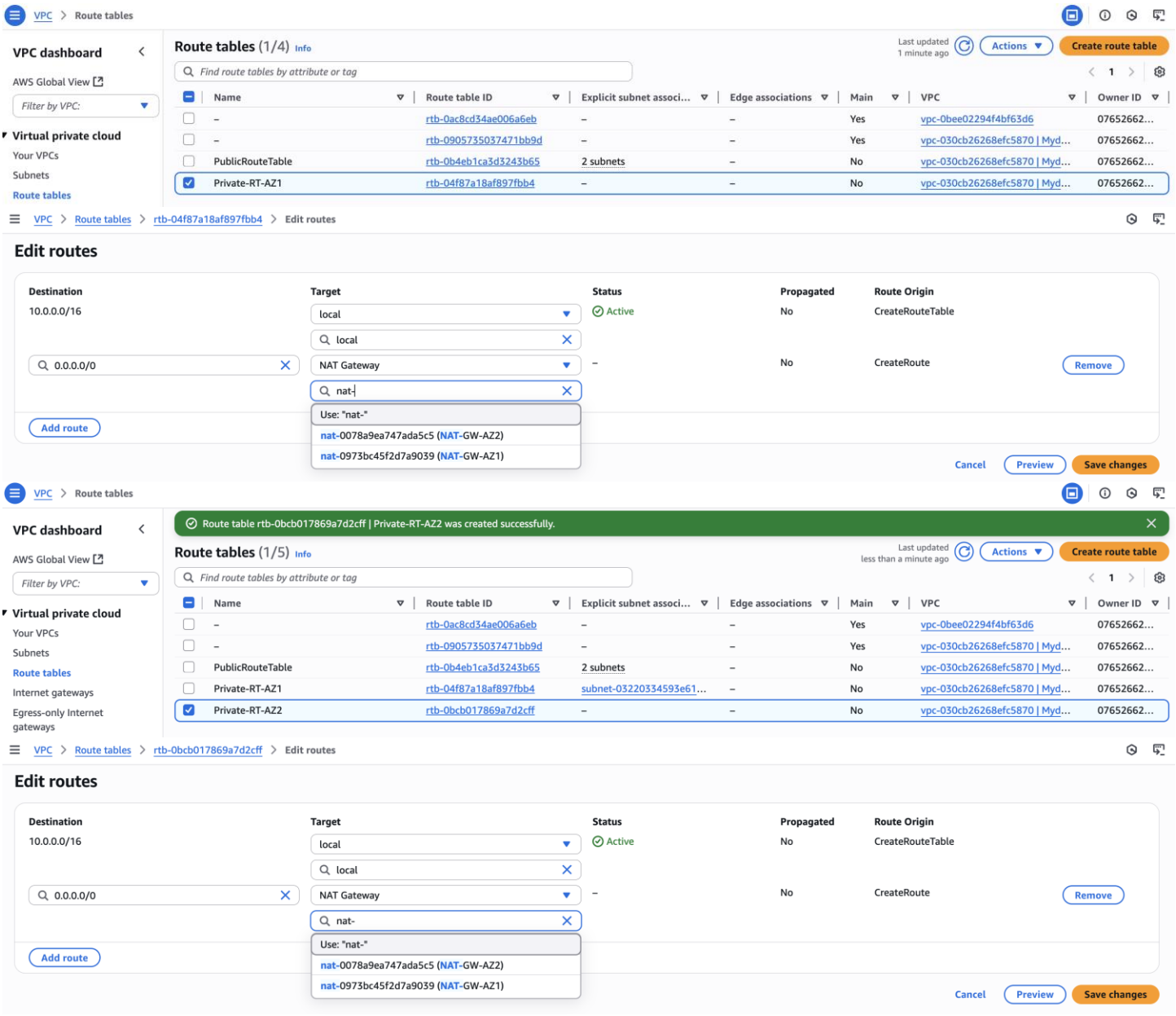
- After creating the route table, you'll automatically be taken to the details page. Scroll down and click on the Routes tab and Edit routes. Add a route that directs traffic from the VPC to the internet gateway. In other words, for all traffic destined for IPs outside the VPC CDIR range, add an entry that directs it to the internet gateway as a target. Save the changes.



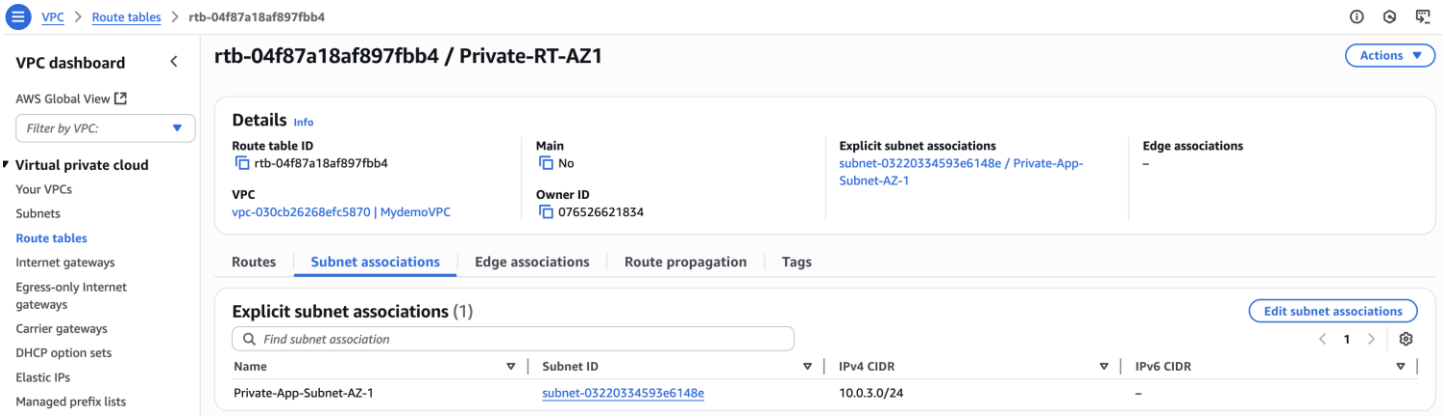
- Edit the Explicit Subnet Associations of the route table by navigating to the route table details again. Select Subnet Associations and click Edit subnet associations. Select the two-web layer public subnets you created earlier and click Save associations.

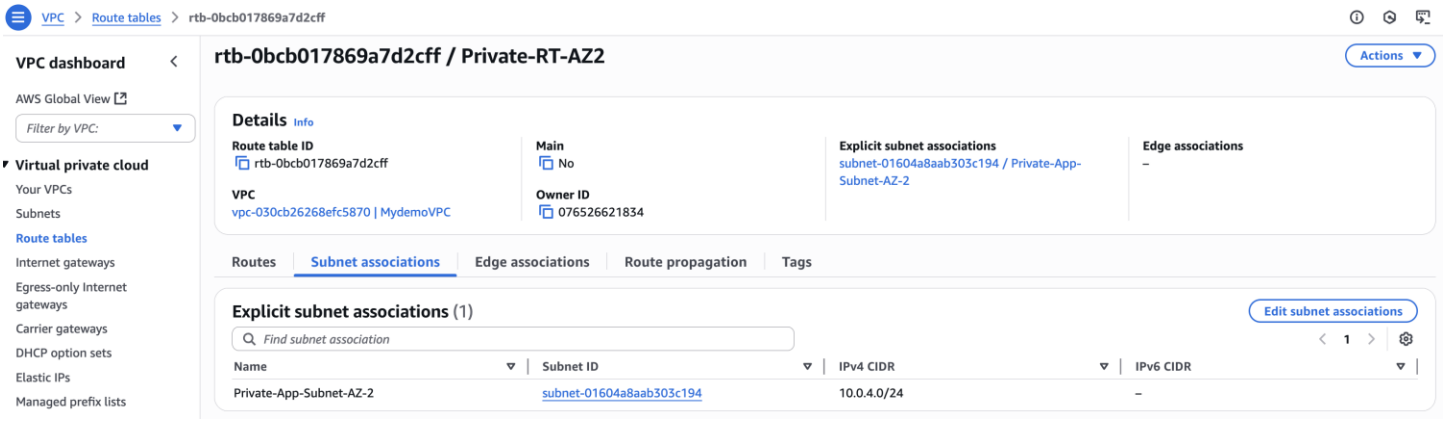


- Now create 2 more route tables, one for each app layer private subnet in each availability zone. These route tables will route app layer traffic destined for outside the VPC to the NAT gateway in the respective availability zone, so add the appropriate routes for that.



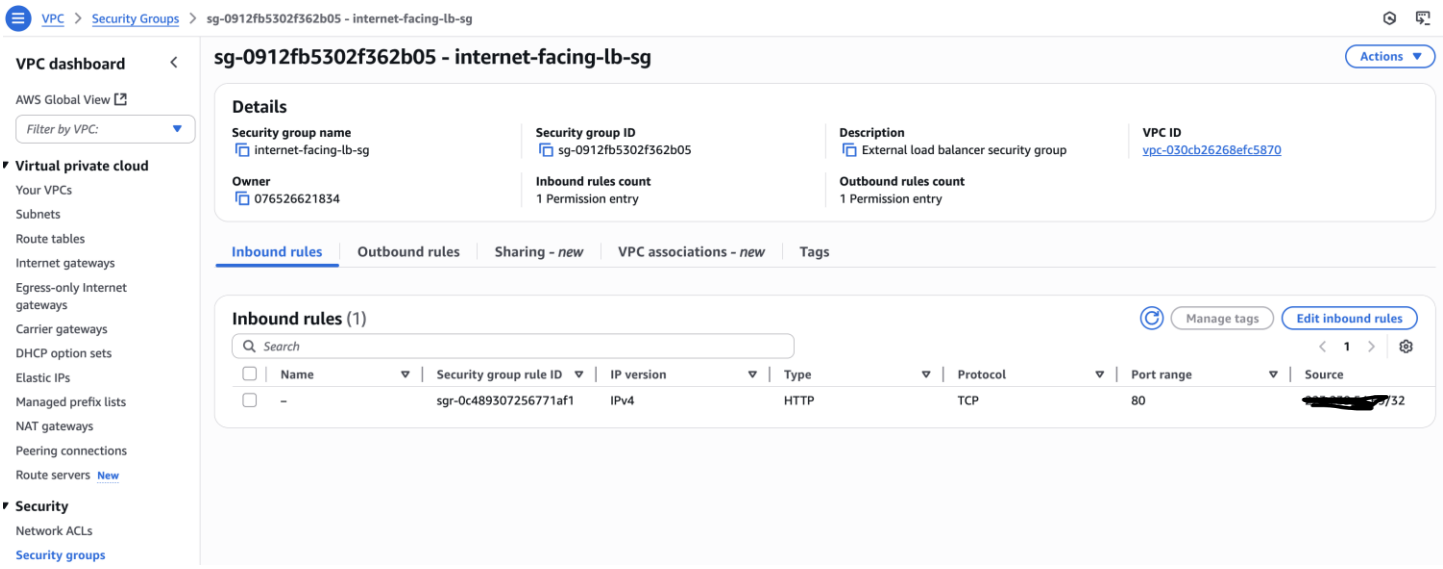
- Once the route tables are created and routes added, add the appropriate subnet associations for each of the app layer private subnets.



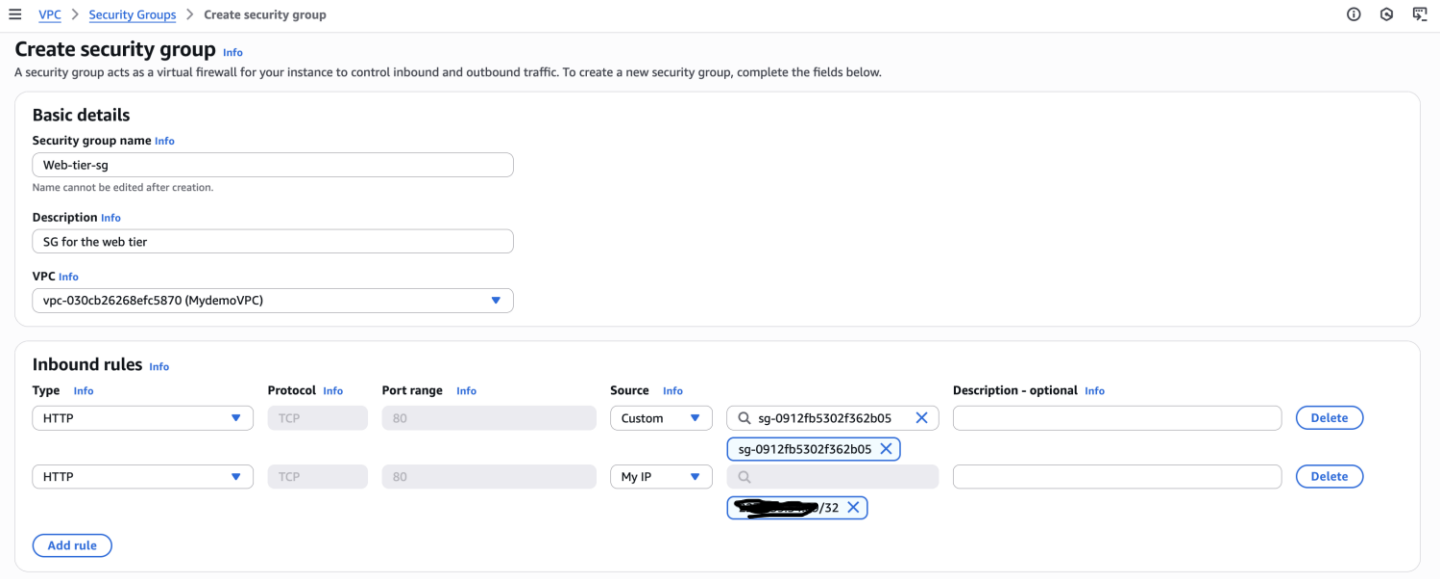


Step-5: Create Security Groups

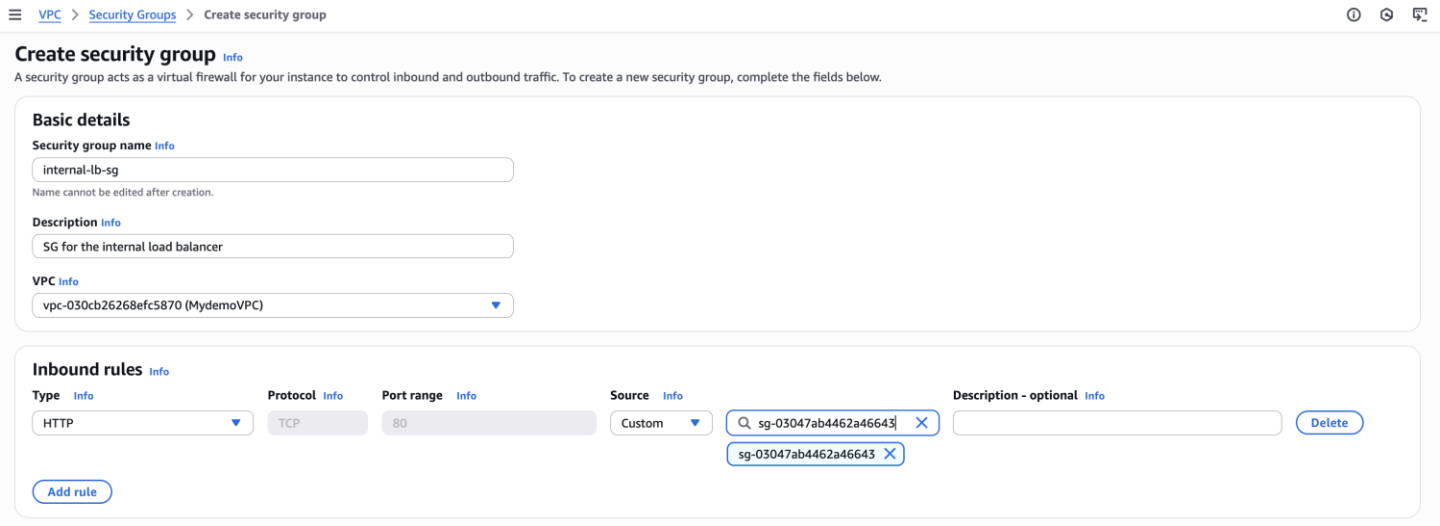
- Security groups will tighten the rules around which traffic will be allowed to our Elastic Load Balancers and EC2 instances. Navigate to Security Groups on the left side of the VPC dashboard, under Security. The first security group you'll create is for the public, internet facing load balancer. After typing a name and description, add an inbound rule to allow HTTP type traffic for your IP.



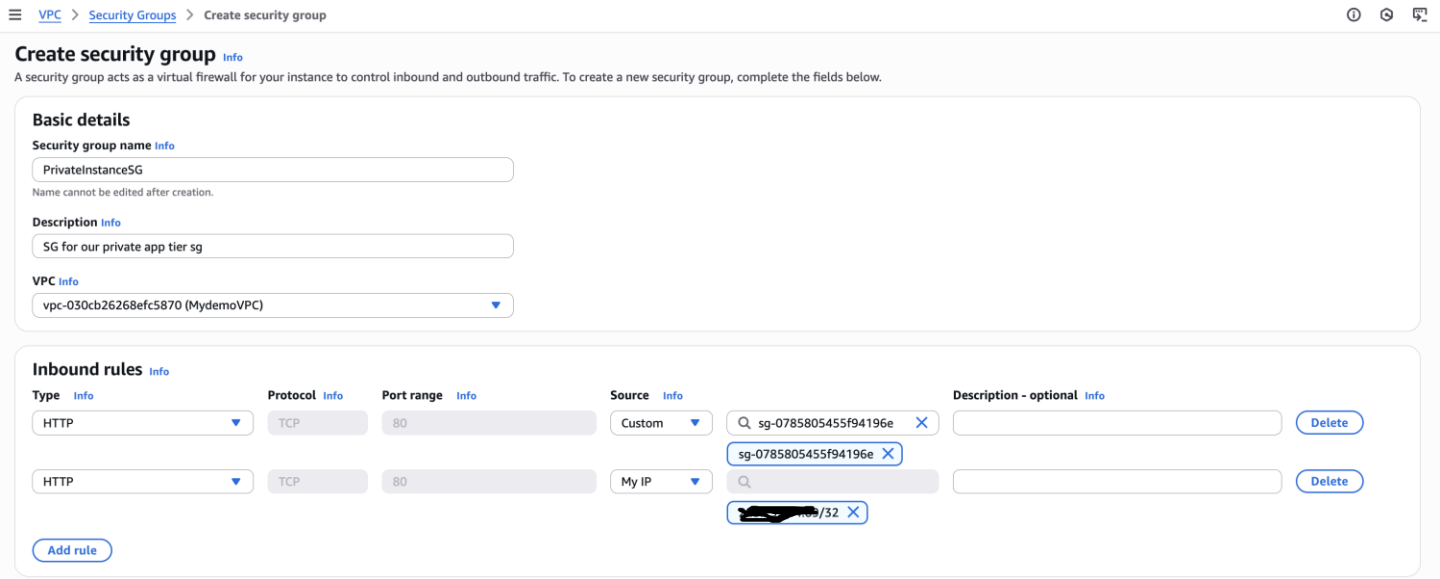
- The second security group you'll create is for the public instances in the web tier. After typing a name and description, add an inbound rule that allows HTTP type traffic from your internet facing load balancer security group you created in the previous step. This will allow traffic from your public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



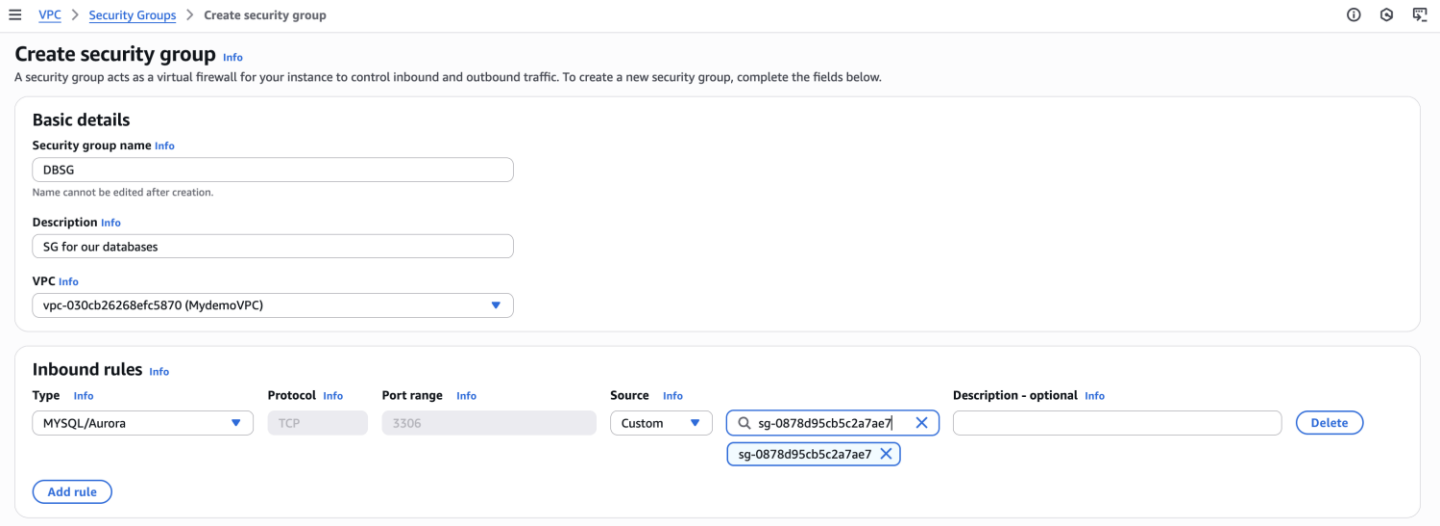
- The third security group will be for our internal load balancer. Create this new security group and add an inbound rule that allows HTTP type traffic from your public instance security group. This will allow traffic from your web tier instances to hit your internal load balancer.

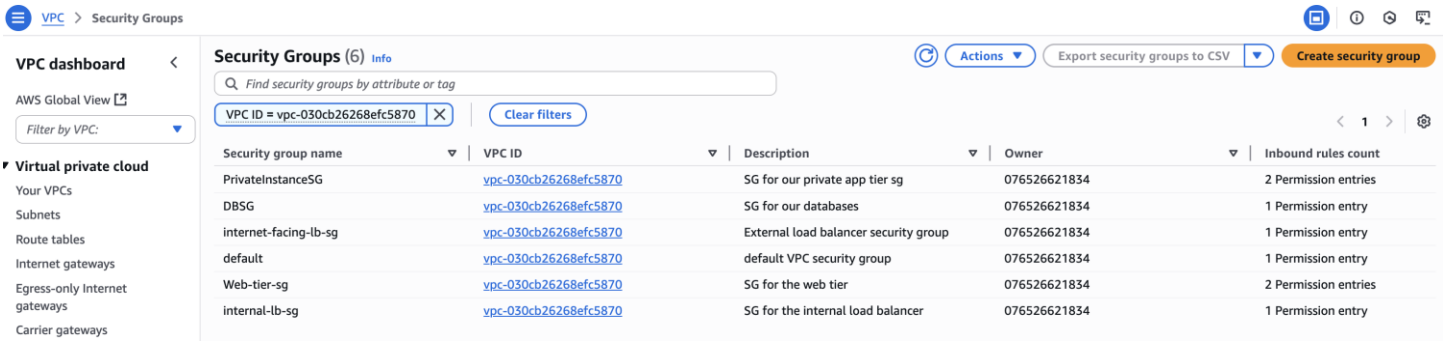


- The fourth security group we'll configure is for our private instances. After typing a name and description, add an inbound rule that will allow TCP type traffic on port 4000 from the internal load balancer security group you created in the previous step. This is the port our app tier application is 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).

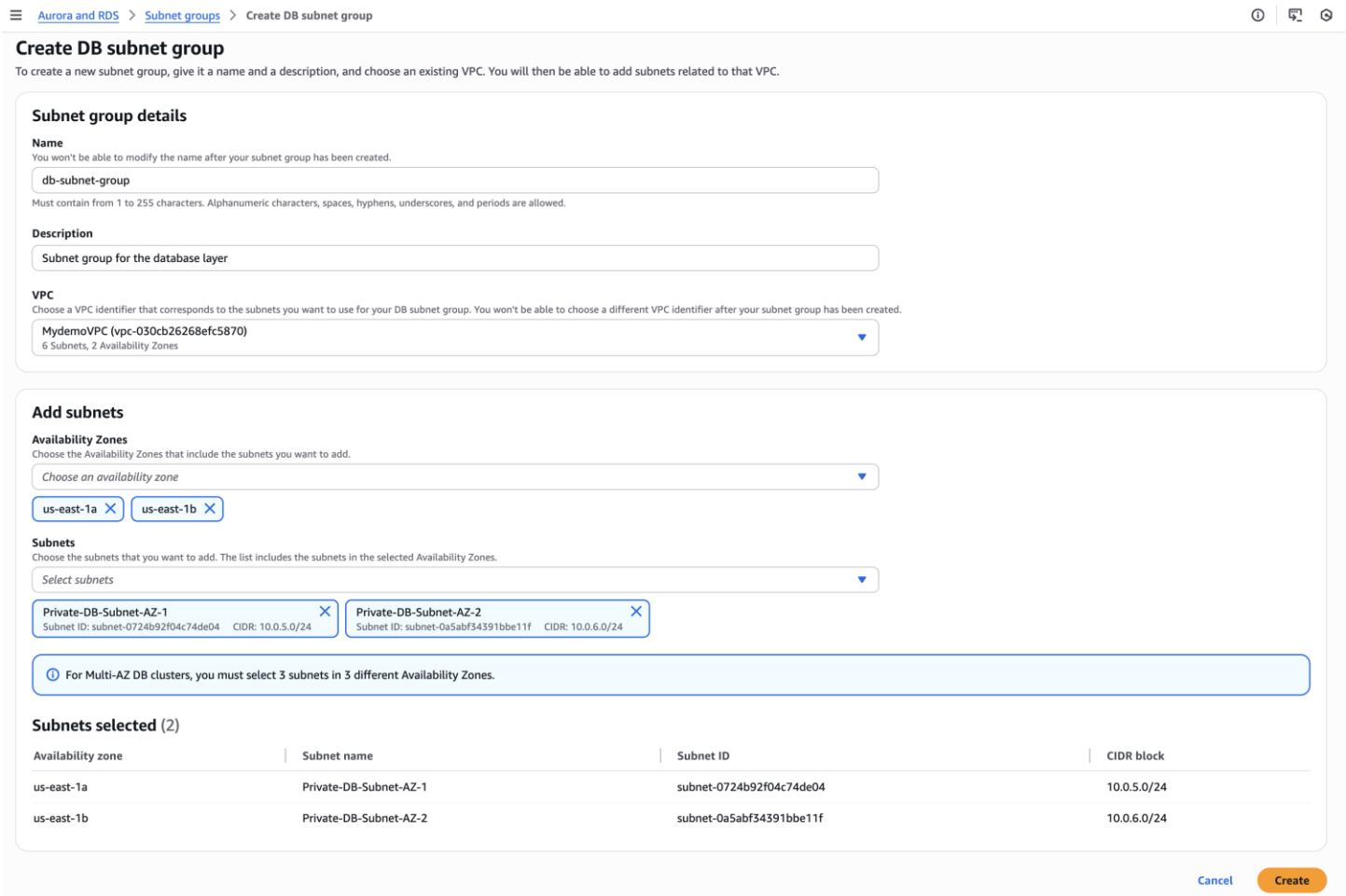


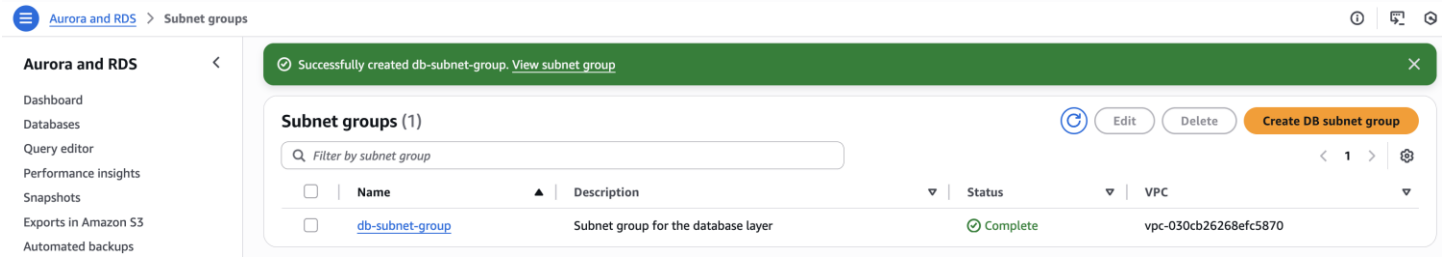


Step-6: Create Database Subnet Groups and deploy database

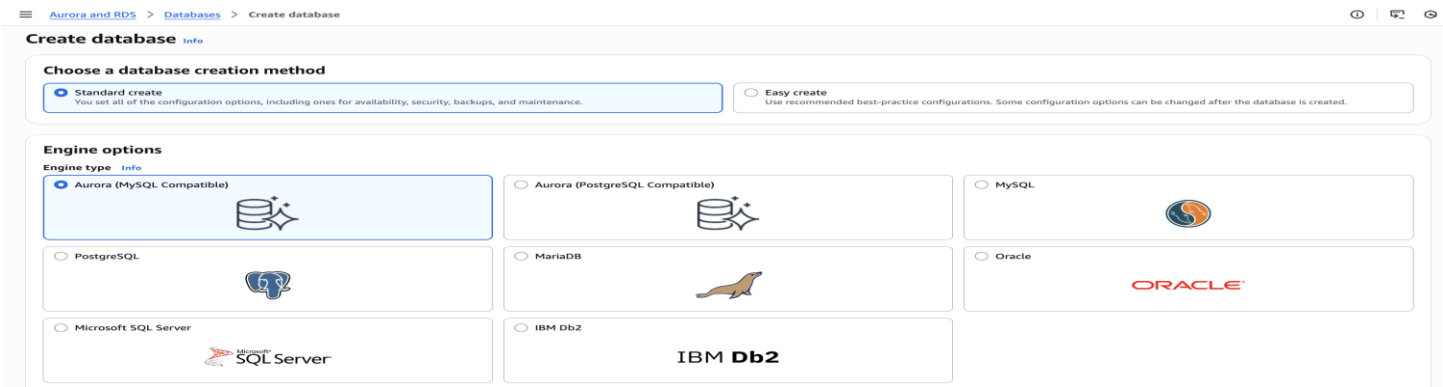
- Navigate to the RDS dashboard in the AWS console and click on Subnet groups on the left hand side. Click Create DB subnet group. Give your subnet group a name, description, and choose the VPC we created.

When adding subnets, make sure to add the subnets we created in each availability zone specifically for our database layer, make sure you're selecting the correct subnet IDs.

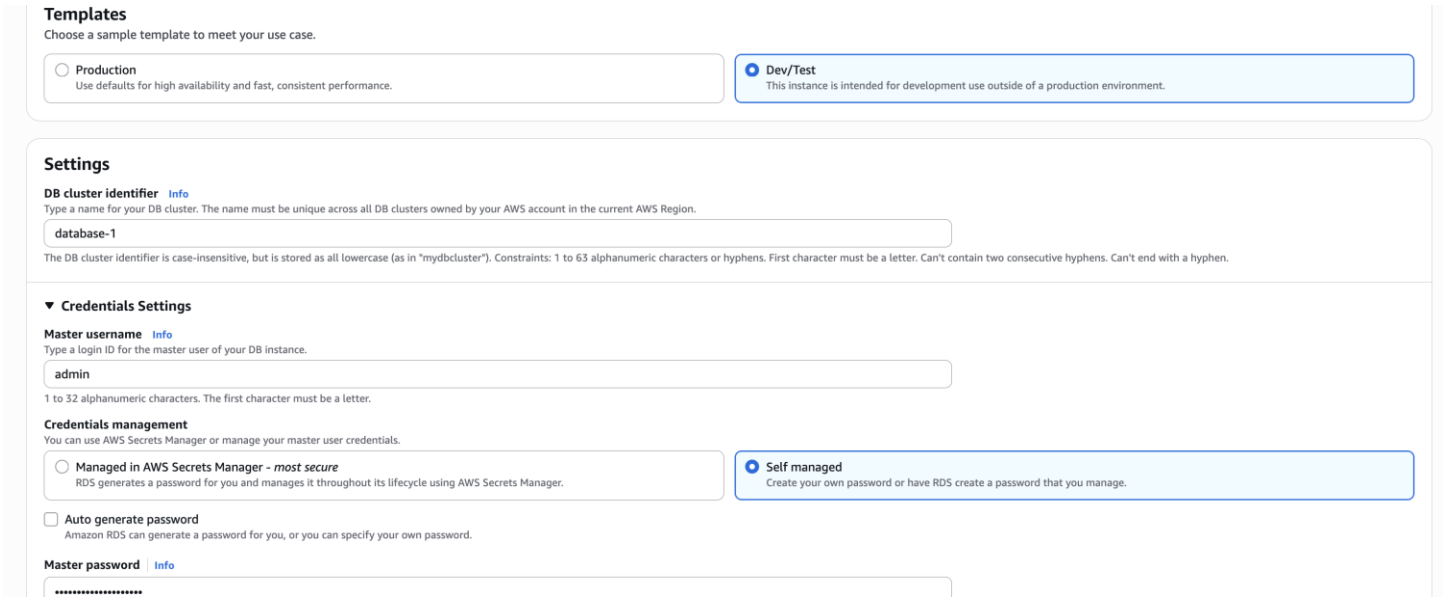




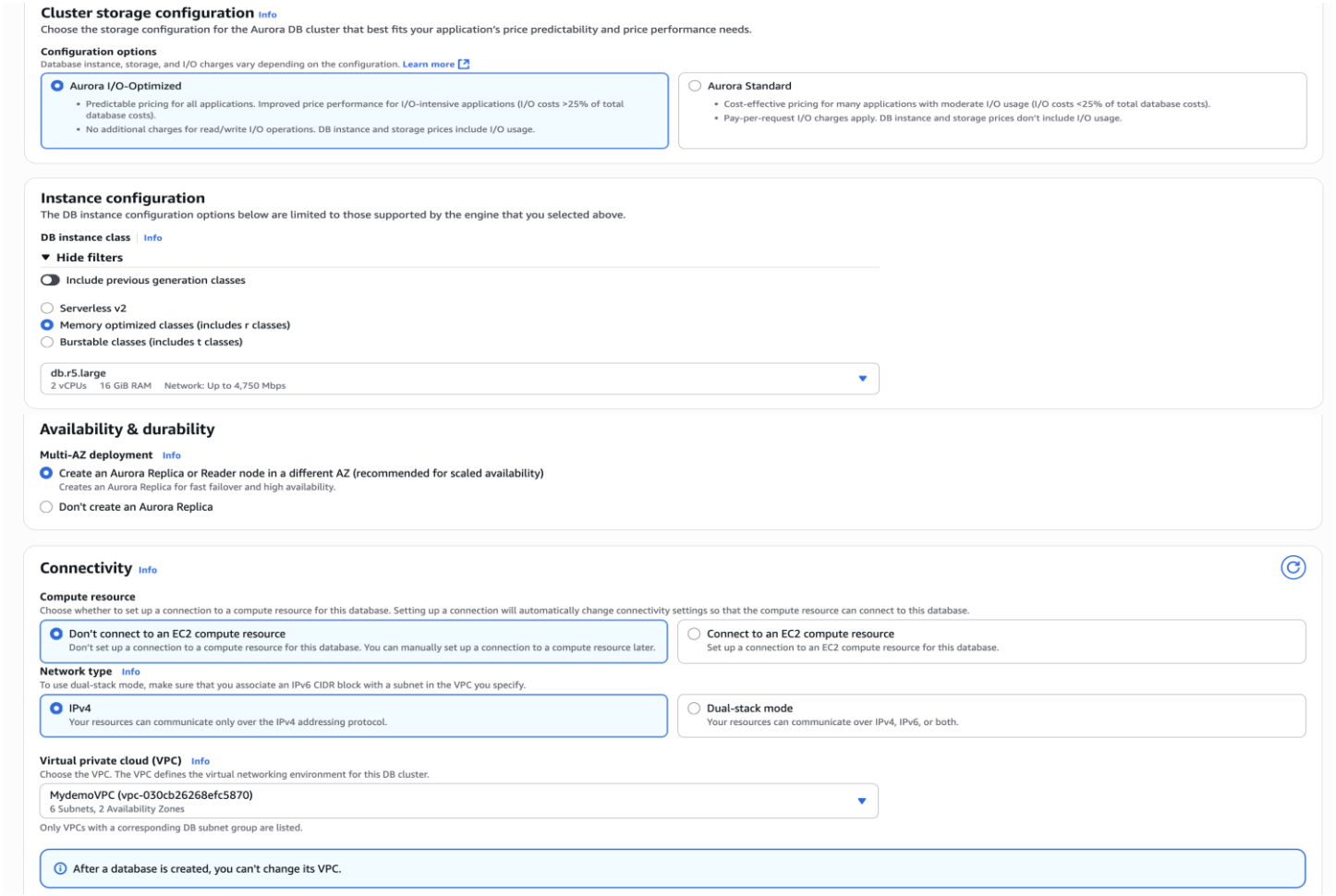
- Navigate to Databases on the left hand side of the RDS dashboard and click Create database. Start with a Standard create for this MySQL-Compatible Amazon Aurora database. Leave the rest of the defaults in the Engine options as default.

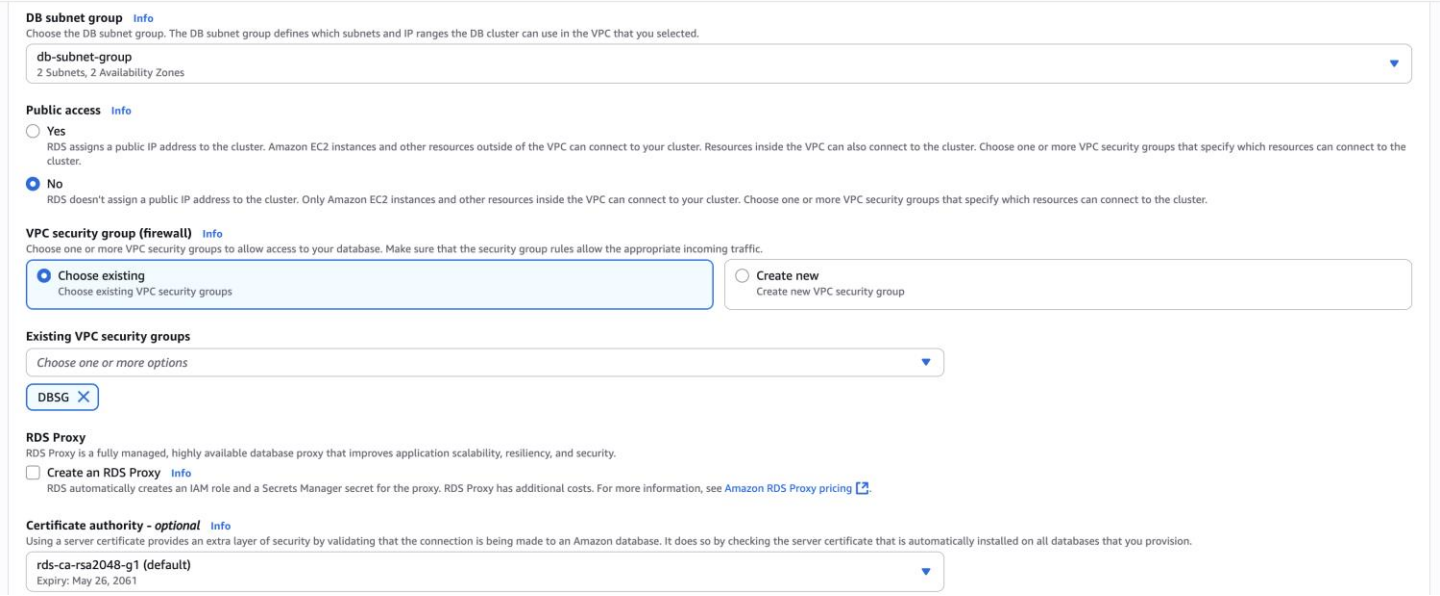


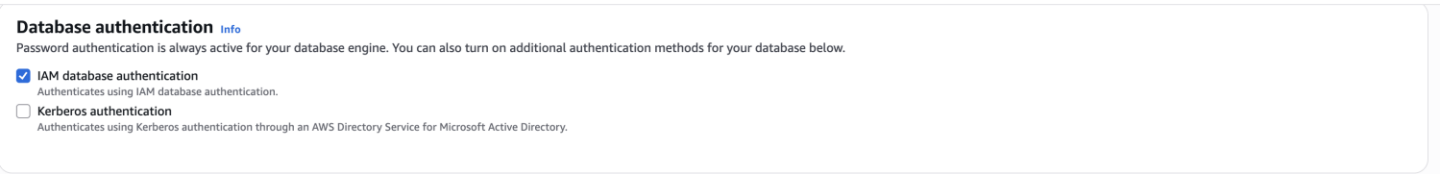
- Under the Templates section choose Dev/Test since this isn't being used for production now. Under Settings set a username and password of your choice and note them down since we'll be using password authentication to access our database.

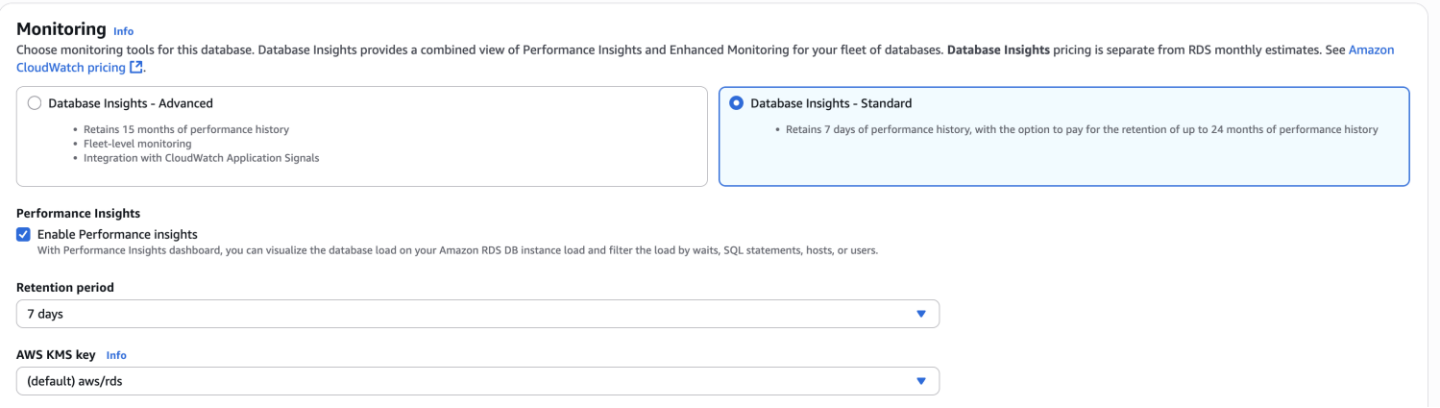


- Next, under Availability and durability change the option to create an Aurora Replica or reader node in a different availability zone. Under Connectivity, set the VPC, choose the subnet group we created earlier, and select no for public access.





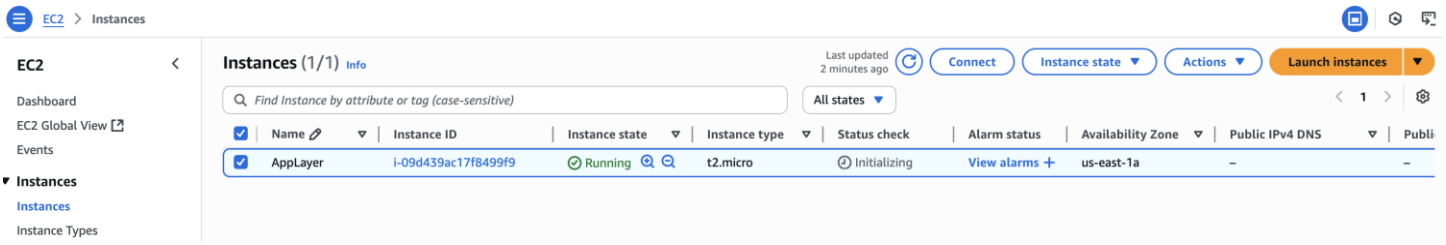




- When your database is provisioned, you should see a reader and writer instance in the database subnets of each availability zone. Note down the writer endpoint for your database for later use.

Step-7: App Tier Instance Deployment

- Navigate to the EC2 service dashboard and click on Instances on the left hand side. Then, click Launch Instances.
- Set the Name and Tag. Select Amazon Linux for the OS and Amazon Linux 2 AMI (HVM) - Kernel 5.10 for the AMI. For Instance Type, select t2.micro. (Select Proceed without a key pair.) For Network settings, click Edit.
- Select a VPC and a subnet. For Subnet, select Private-App-Subnet. For Security Group, select PrivateInstanceSG, which you created in the previous step. For Auto-assign public IP, select Disable. click Advanced details.
- In the IAM instance profile field, select the IAM Role you created in the previous step. (In our example, we created the EC2andS3Role.) Finally, click Launch instance to create the instance.



- When the instance state is running, connect to your instance. witch to ec2-user by executing the following command in the browser terminal:
    - sudo -su ec2-user

  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.

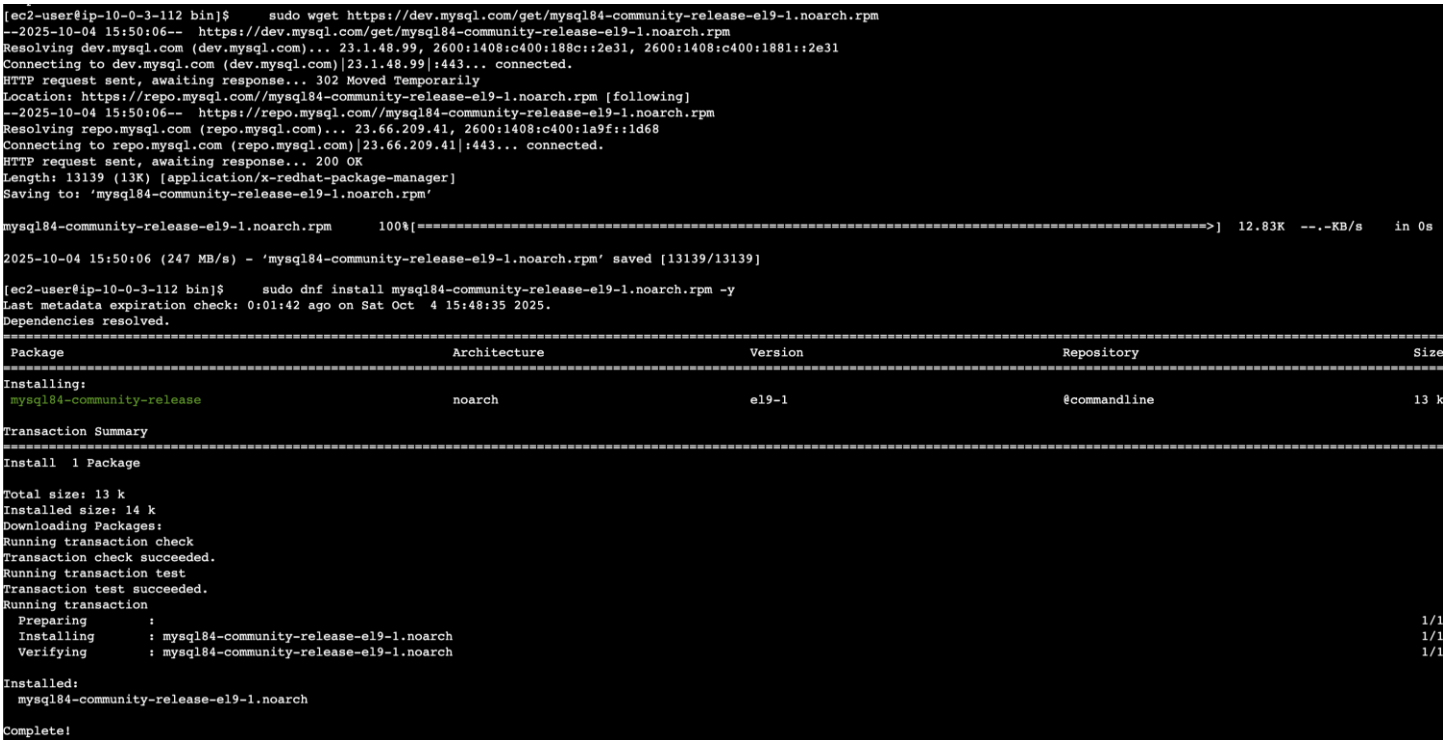
Session ID: root-i2yngjb882nqouqqd5ye3suju8 | Shortcuts | Instance ID: i-09d439ac17f8499f9

```
sh-5.2$ sudo -su ec2-user
[ec2-user@ip-10-0-3-112 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=112 time=2.17 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=112 time=1.80 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=112 time=1.81 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=112 time=1.77 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=112 time=1.85 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 1.770/1.878/2.168/0.147 ms
[ec2-user@ip-10-0-3-112 bin]$
```

- Now configure database. Start by downloading the MySQL CLI.
    - sudo dnf update -y
    - sudo wget https://dev.mysql.com/get/mysql84-community-release-el9-1.noarch.rpm
    - sudo dnf install mysql84-community-release-el9-1.noarch.rpm -y
    - sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql
    - sudo dnf install mysql-community-server -y
    - sudo systemctl enable --now mysqld
- 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.
- 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;
- 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;
- When finished, just type exit and hit enter to exit the MySQL client.



```
[ec2-user@ip-10-0-3-112 bin]$     sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql
[ec2-user@ip-10-0-3-112 bin]$     sudo dnf install mysql-community-server -y
MySQL 8.4 LTS Community Server                                              8.6 MB/s | 1.2 MB     00:00
MySQL Connectors Community                                                  1.6 MB/s |  90 kB     00:00
MySQL Tools 8.4 LTS Community                                               6.7 MB/s | 659 kB     00:00
Dependencies resolved.
==========================================================================================================
 Package                            Architecture     Version          Repository                     Size
==========================================================================================================
Installing:
 mysql-community-server             x86_64           8.4.6-1.el9      mysql-8.4-lts-community        50 M
Installing dependencies:
 mysql-community-client             x86_64           8.4.6-1.el9      mysql-8.4-lts-community       3.1 M
 mysql-community-client-plugins     x86_64           8.4.6-1.el9      mysql-8.4-lts-community       1.5 M
 mysql-community-common             x86_64           8.4.6-1.el9      mysql-8.4-lts-community       578 k
 mysql-community-icu-data-files     x86_64           8.4.6-1.el9      mysql-8.4-lts-community       2.3 M
 mysql-community-libs               x86_64           8.4.6-1.el9      mysql-8.4-lts-community       1.5 M

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

Total download size: 59 M
Installed size: 331 M
Downloading Packages:
(1/6): mysql-community-common-8.4.6-1.el9.x86_64.rpm                        6.5 MB/s | 578 kB     00:00
(2/6): mysql-community-client-8.4.6-1.el9.x86_64.rpm                         28 MB/s | 3.1 MB     00:00
(3/6): mysql-community-libs-8.4.6-1.el9.x86_64.rpm                           55 MB/s | 1.5 MB     00:00
(4/6): mysql-community-icu-data-files-8.4.6-1.el9.x86_64.rpm                 32 MB/s | 2.3 MB     00:00
(5/6): mysql-community-client-plugins-8.4.6-1.el9.x86_64.rpm                6.4 MB/s | 1.5 MB     00:00
(6/6): mysql-community-server-8.4.6-1.el9.x86_64.rpm                         79 MB/s |  50 MB     00:00
----------------------------------------------------------------------------------------------------------
Total                                                                        76 MB/s |  59 MB     00:00
MySQL 8.4 LTS Community Server                                              3.0 MB/s | 3.1 kB     00:00
Importing GPG key 0xA8D3785C:
[ec2-user@ip-10-0-3-112 bin]$     sudo systemctl enable --now mysqld
[ec2-user@ip-10-0-3-112 bin]$ mysql -h database-1.cluster-cclmg2aewqne.us-east-1.rds.amazonaws.com -u admin -p taKdew-3qegwe-cikwak
Enter password:
ERROR 1049 (42000): Unknown database 'taKdew-3qegwe-cikwak'
[ec2-user@ip-10-0-3-112 bin]$ mysql -h database-1.cluster-cclmg2aewqne.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 155
Server version: 8.0.39 8bc99e28

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 (0.00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| webappdb           |
+--------------------+
5 rows in set (0.01 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 (0.02 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_webappdb |
+--------------------+
| transactions       |
+--------------------+
1 row in set (0.00 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 (0.01 sec)

mysql> exit
```

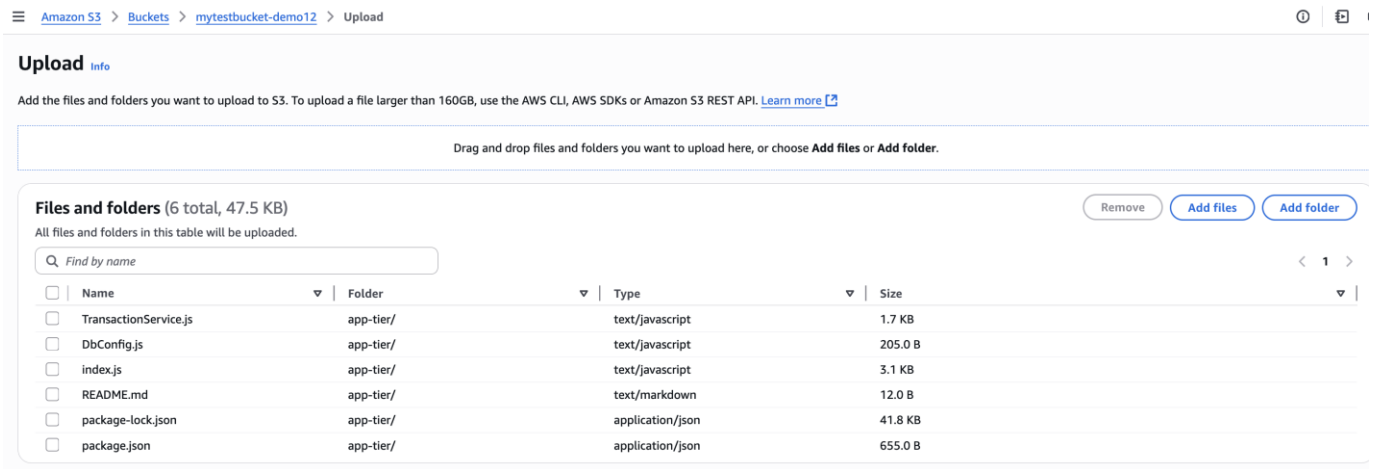
Step-8: Configure App Instance

- 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.

DbConfig.js

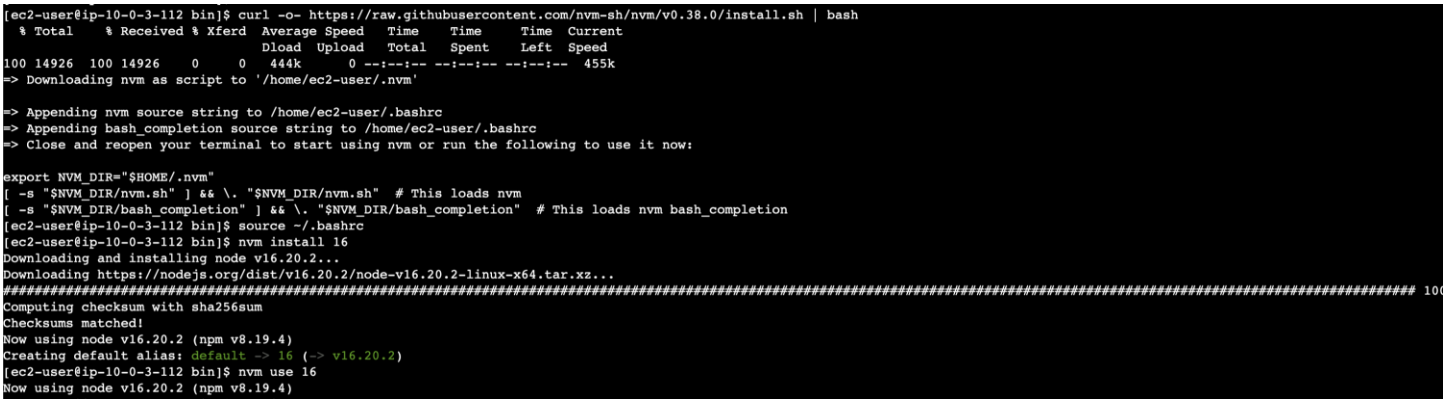
```
module.exports = Object.freeze({
    DB_HOST : 'database-1-instance-1.cclmq2aewqne.us-east-1.rds.amazonaws.com',
    DB_USER : 'admin',
    DB_PWD : 'taKdew-3qegwe-cikwak',
    DB_DATABASE : 'webappdb'
});
```

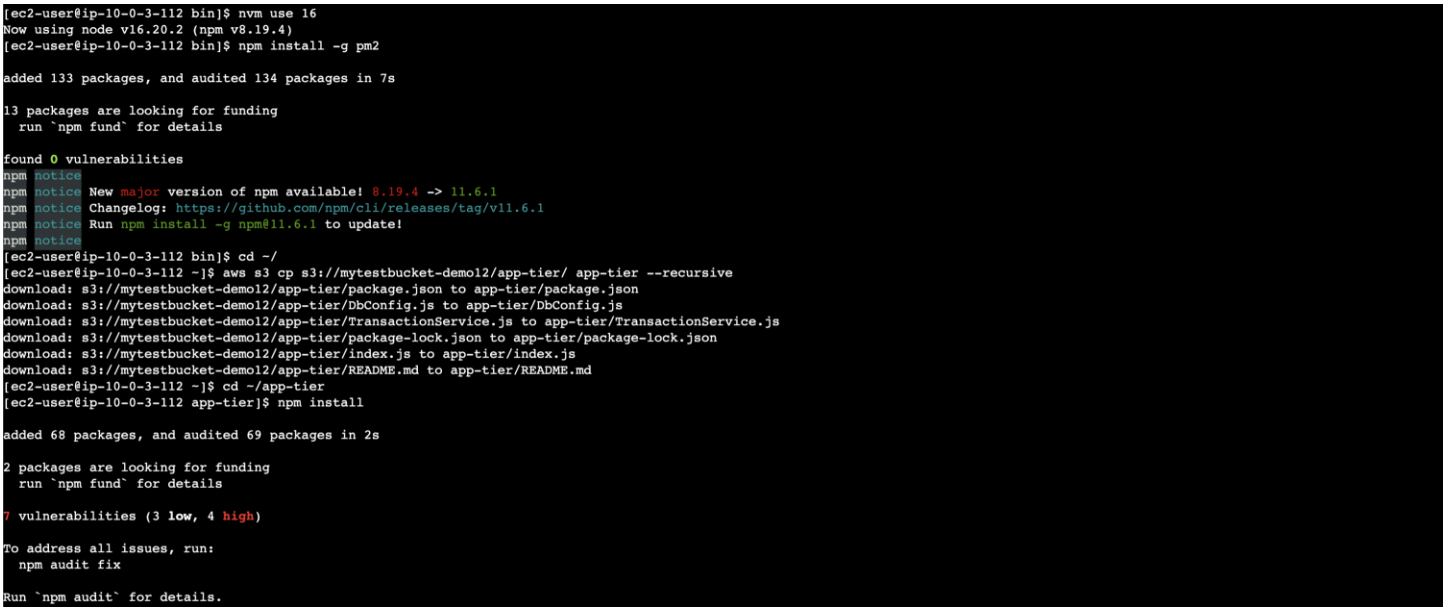
- Upload the app-tier folder to the S3 bucket that you created in part 0.

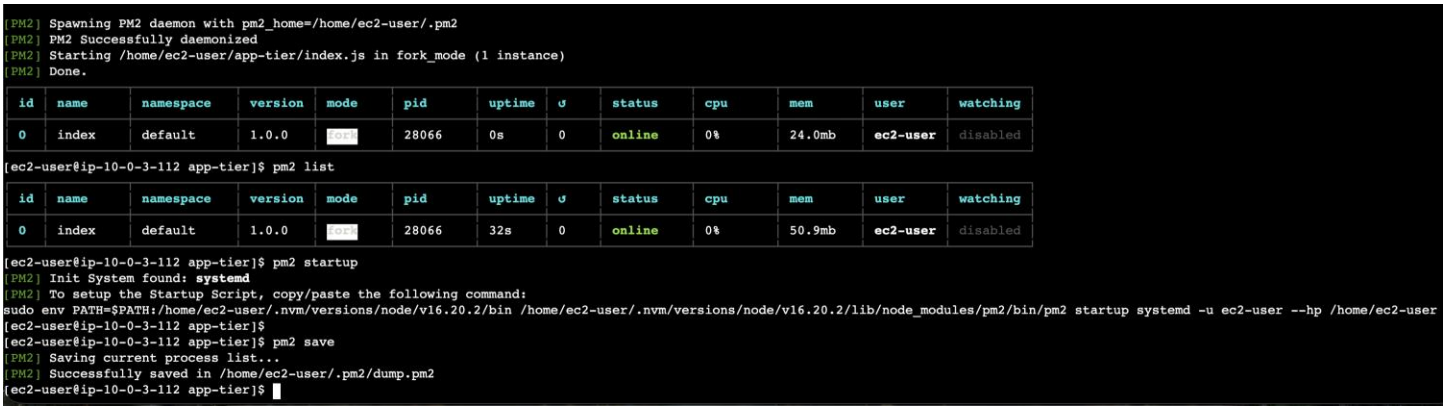


- Now we need to install all of the necessary components to run our backend application. Start by installing NVM (node version manager).
    - curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
    - source ~/.bashrc
- Next, install a compatible version of Node.js and make sure it's being used
    - nvm install 16
    - nvm use 16
- PM2 is a daemon process manager that will keep our node.js app running when we exit the instance or if it is rebooted. Install that as well.
    - npm install -g pm2
- Now we need to download our code from our s3 buckets onto our instance. In the command below, replace BUCKET_NAME with the name of the bucket you uploaded the app-tier folder to:
    - Cd ~/
    - aws s3 cp s3:// mytestbucket-demo12/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
- 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.
- 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.
    - To setup the Startup Script, copy/paste the following command:
      sudo env PATH=$PATH:/home/ec2-user/.nvm/versions/node/v16.20.2/bin /home/ec2-user/.nvm/versions/node/v16.20.2/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





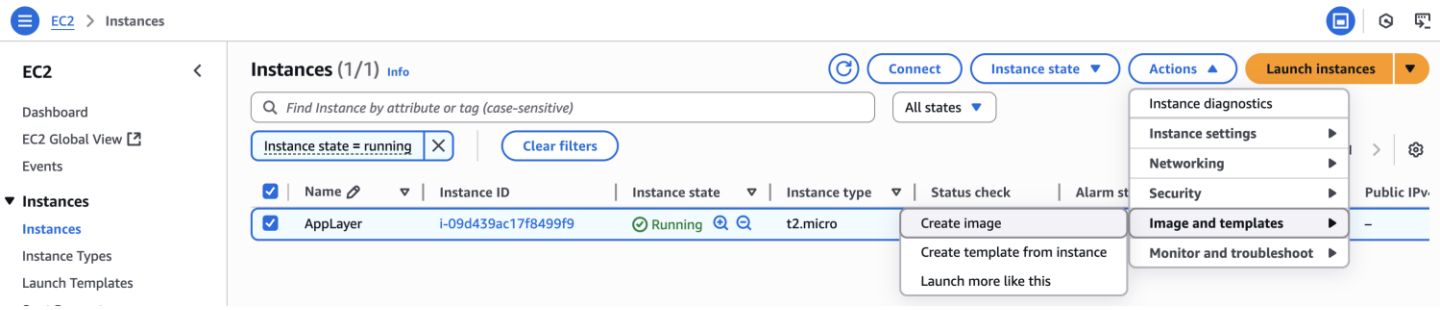


Step-9: Test App Tier

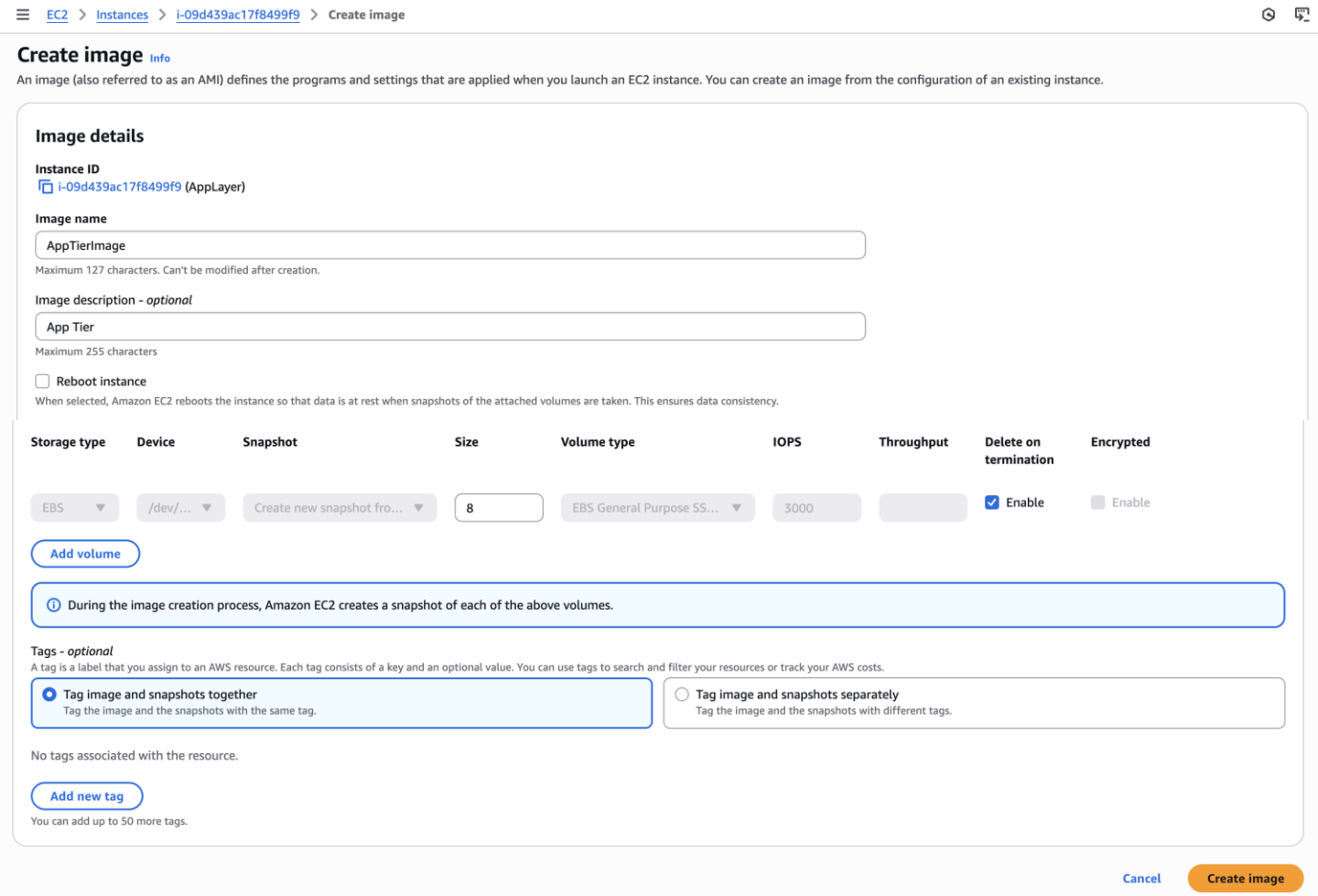
- Now let's run a couple tests to see if our app is configured correctly and can retrieve data from the database. To hit out health check endpoint, copy this command into your SSM terminal. This is our simple health check endpoint that tells us if the app is simply running.
    - curl http://localhost:4000/health
- 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"}]}
      If you see both of these responses, then your networking, security, database and app configurations are correct. Congrats! Your app layer is fully configured and ready to go.

Step-10: Internal Load Balancing and Auto Scaling

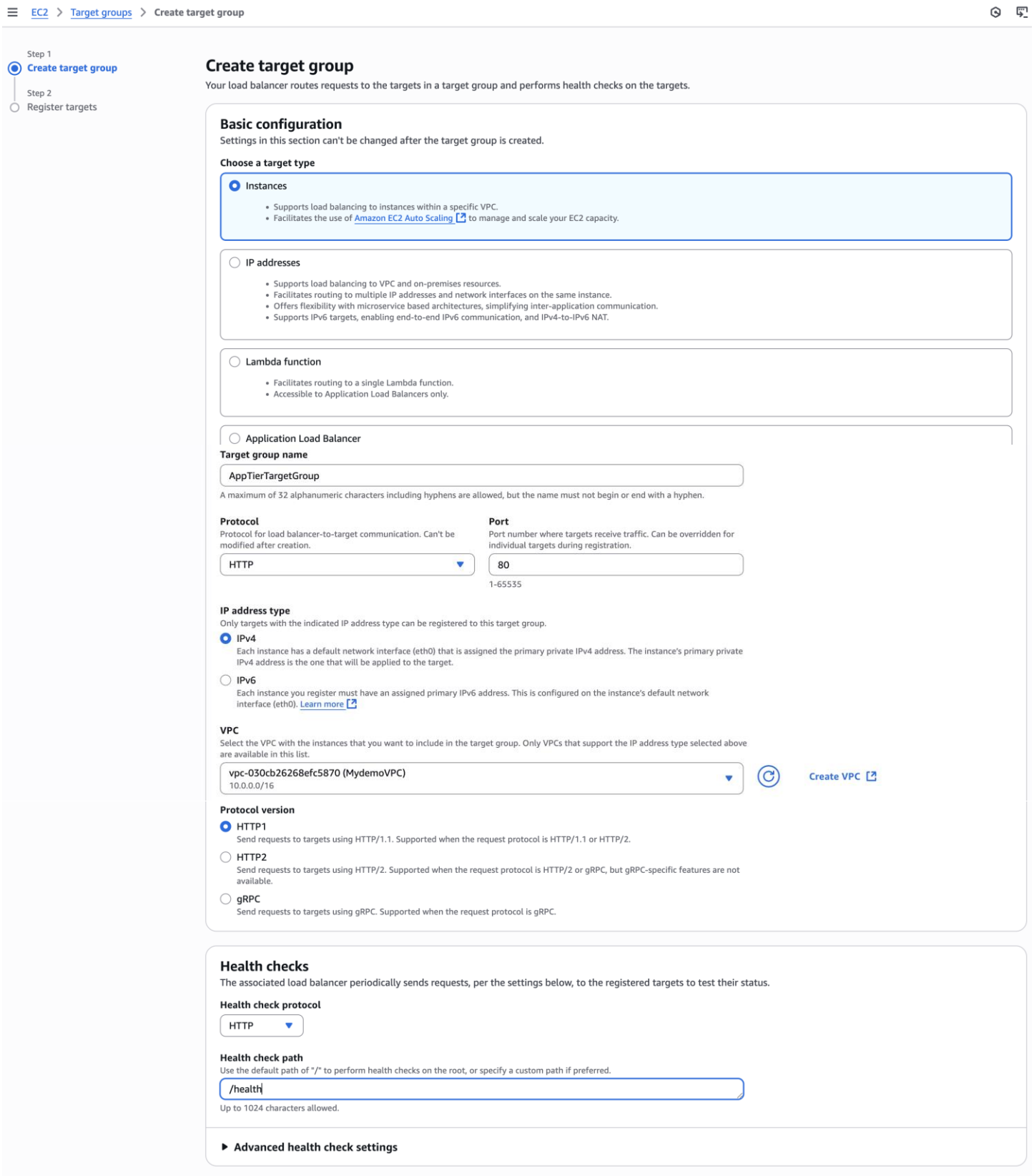
- Navigate to Instances on the left hand side of the EC2 dashboard. Select the app tier instance we created and under Actions select Image and templates. Click Create Image.



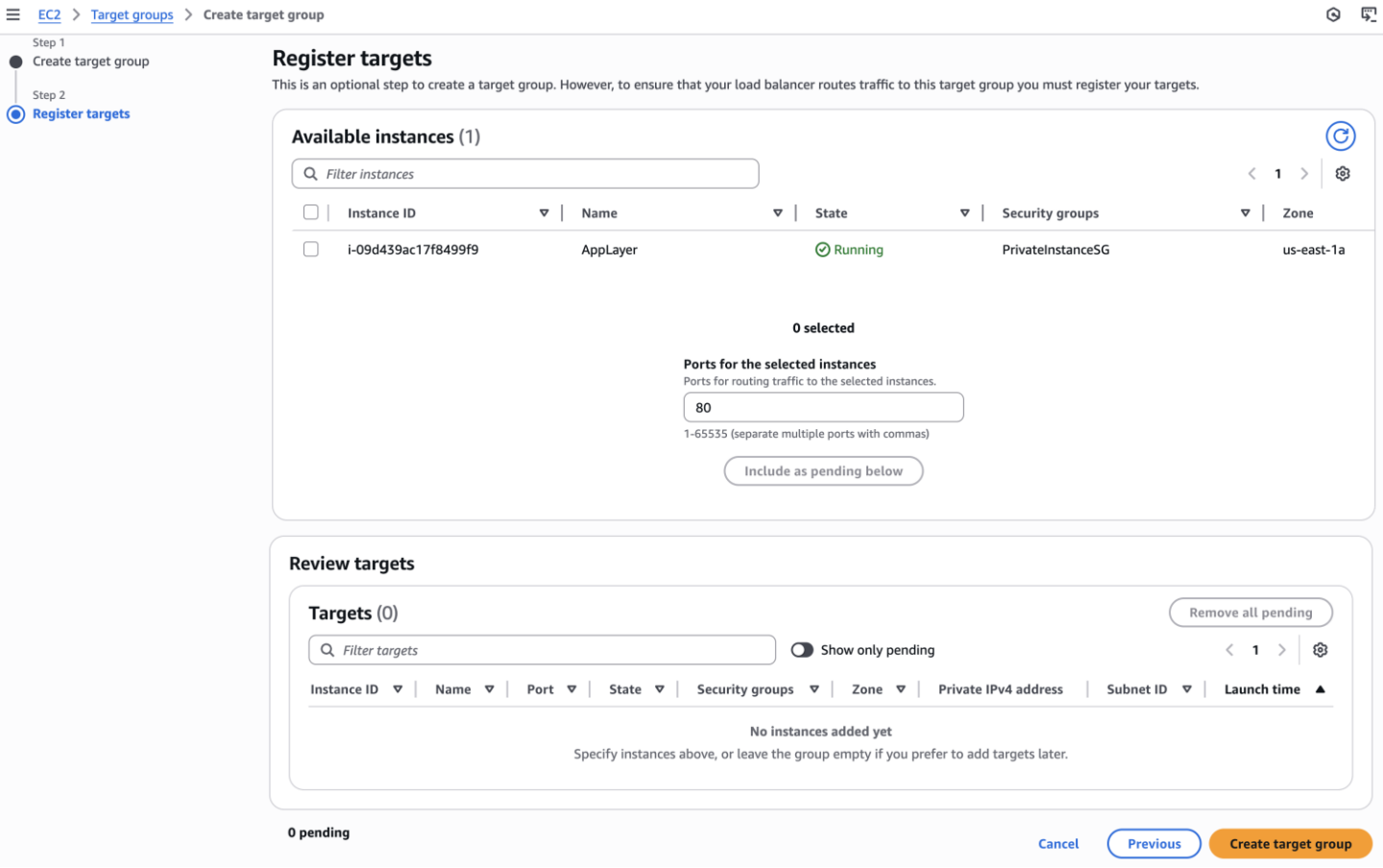
- Give the image a name and description and then click Create image. This will take a few minutes, but if you want to monitor the status of image creation you can see it by clicking AMIs under Images on the left hand navigation panel of the EC2 dashboard.



- 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.
  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.



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



- On the left hand side of the EC2 dashboard select Load Balancers under Load Balancing and click Create Load Balancer. We'll be using an Application Load Balancer for our HTTP traffic so click the create button for that option.
- 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.

Create Application Load Balancer Info

The Application Load Balancer distributes incoming HTTP and HTTPS traffic across multiple targets such as Amazon EC2 instances, microservices, and containers, based on request attributes. When the load balancer receives a connection request, it evaluates the listener rules in priority order to determine which rule to apply, and if applicable, it selects a target from the target group for the rule action.

▶ **How Application Load Balancers work**

Basic configuration

Load balancer name

Name must be unique within your AWS account and can't be changed after the load balancer is created.

app-tier-internal-lb

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Scheme | Info

Scheme can't be changed after the load balancer is created.

- ○ Internet-facing
  - Serves internet-facing traffic.
  - Has public IP addresses.
  - DNS name resolves to public IPs.
  - Requires a public subnet.
- ◉ Internal
  - Serves internal traffic.
  - Has private IP addresses.
  - DNS name resolves to private IPs.
  - Compatible with the **IPv4** and **Dualstack** IP address types.

Load balancer IP address type | Info

Select the front-end IP address type to assign to the load balancer. The VPC and subnets mapped to this load balancer must include the selected IP address types. Public IPv4 addresses have an additional cost.

◉ IPv4
Includes only IPv4 addresses.

Network mapping Info

The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.

VPC | Info

The load balancer will exist and scale within the selected VPC. The selected VPC is also where the load balancer targets must be hosted unless routing to Lambda or on-premises targets, or if using VPC peering. To confirm the VPC for your targets, view target groups.

vpc-030cb26268efc5870 (MydemoVPC)
10.0.0.0/16

Create VPC

IP pools | Info

You can optionally choose to configure an IPAM pool as the preferred source for your load balancers IP addresses. Create or view **Pools** in the Amazon VPC IP Address Manager console.

☐ Use IPAM pool for public IPv4 addresses
The IPAM pool you choose will be the preferred source of public IPv4 addresses. If the pool is depleted IPv4 addresses will be assigned by AWS.

Availability Zones and subnets | Info

Select at least two Availability Zones and a subnet for each zone. A load balancer node will be placed in each selected zone and will automatically scale in response to traffic. The load balancer routes traffic to targets in the selected Availability Zones only.

☑ **us-east-1a (use1-az1)**

Subnet

Only CIDR blocks corresponding to the load balancer IP address type are used. At least 8 available IP addresses are required for your load balancer to scale efficiently.

subnet-03220334593e6148e — Private-App-Subnet-AZ-1
IPv4 subnet CIDR: 10.0.3.0/24

⚠ The selected subnet does not have a route to an internet gateway. This means that your load balancer will not receive internet traffic.
You can proceed with this selection; however, for internet traffic to reach your load balancer, you must update the subnet's route table in the VPC console.

☑ **us-east-1b (use1-az2)**

Subnet

Only CIDR blocks corresponding to the load balancer IP address type are used. At least 8 available IP addresses are required for your load balancer to scale efficiently.

subnet-01604a8aab303c194 — Private-App-Subnet-AZ-2
IPv4 subnet CIDR: 10.0.4.0/24

⚠ The selected subnet does not have a route to an internet gateway. This means that your load balancer will not receive internet traffic.
You can proceed with this selection; however, for internet traffic to reach your load balancer, you must update the subnet's route table in the VPC console.

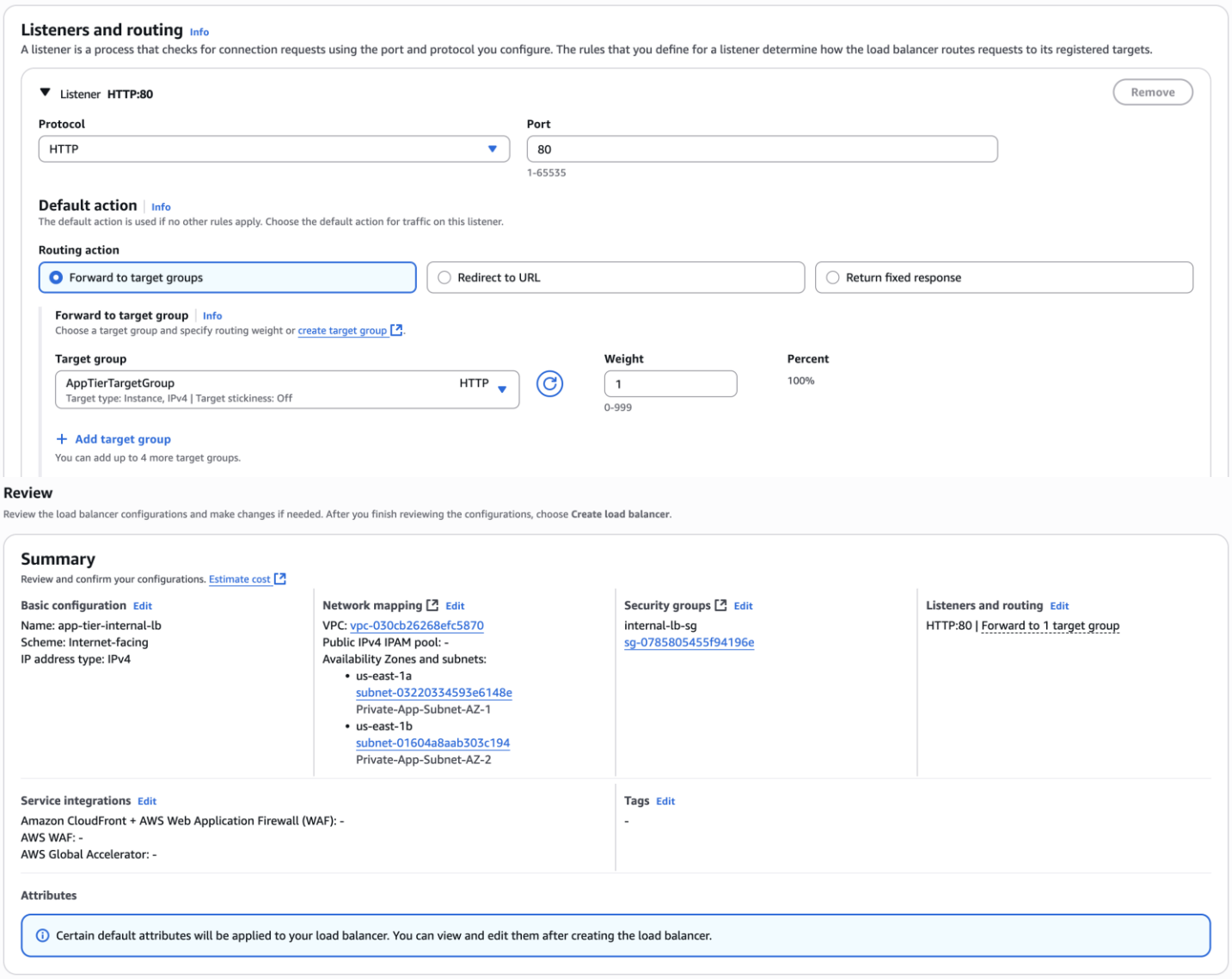
Security groups Info

A security group is a set of firewall rules that control the traffic to your load balancer. Select an existing security group, or you can create a new security group.

Security groups

Select up to 5 security groups

internal-lb-sg ×
sg-0785805455f94196e VPC: vpc-030cb26268efc5870



- 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.
- 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.

Create launch template

Creating a launch template allows you to create a saved instance configuration that can be reused, shared and launched at a later time. Templates can have multiple versions.

Launch template name and description

Launch template name - *required*

AppTierLaunchTemplate

Must be unique to this account. Max 128 chars. No spaces or special characters like '&', '*', '@'.

Template version description

Launch template for app tier

Max 255 chars

Auto Scaling guidance | Info

Select this if you intend to use this template with EC2 Auto Scaling

☐ Provide guidance to help me set up a template that I can use with EC2 Auto Scaling

▶ **Template tags**

▶ **Source template**

Summary

Software Image (AMI)
App Tier
ami-0b5b575c7c2ba2404

Virtual server type (instance type)
t2.micro

Firewall (security group)
PrivateInstanceSG

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet. Data transfer charges are not included

EC2 > Launch templates > Create launch template

Launch template contents

Specify the details of your launch template below. Leaving a field blank will result in the field not being included in the launch template.

Application and OS Images (Amazon Machine Image) Info

An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose **Browse more AMIs**.

Search our full catalog including 1000s of application and OS images

Recents | My AMIs | Quick Start

◯ Don't include in launch template
◉ Owned by me
◯ Shared with me

Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

AppTierImage
ami-0b5b575c7c2ba2404
2025-10-04T16:37:08.000Z Virtualization: hvm ENA enabled: true Root device type: ebs Boot mode: uefi-preferred

Description
App Tier

Architecture
x86_64

AMI ID
ami-0b5b575c7c2ba2404

Instance type Info | Get advice

Advanced

Instance type

t2.micro
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour On-Demand RHEL base pricing: 0.026 USD per Hour
On-Demand Linux base pricing: 0.0116 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name

Don't include in launch template

Create new key pair

Summary

Software Image (AMI)
App Tier
ami-0b5b575c7c2ba2404

Virtual server type (instance type)
t2.micro

Firewall (security group)
PrivateInstanceSG

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet. Data transfer charges are not included

Cancel | Create launch template

Summary

Software Image (AMI)
App Tier
ami-0b5b575c7c2ba2404

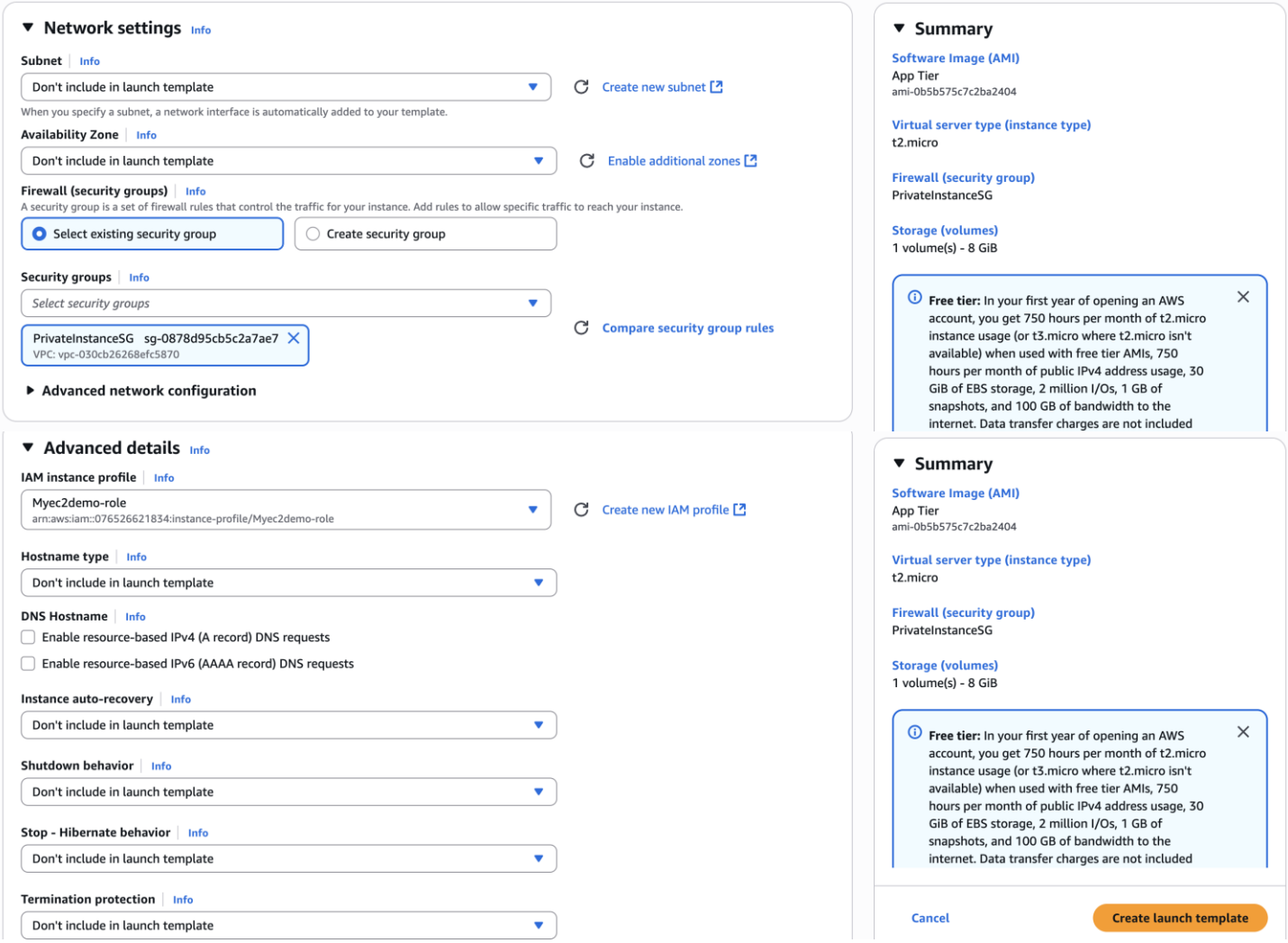
Virtual server type (instance type)
t2.micro

Firewall (security group)
PrivateInstanceSG

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet. Data transfer charges are not included

Cancel | Create launch template



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

Step 1
Choose launch template or configuration
Step 2
Choose instance launch options
Step 3 - optional
Integrate with other services
Step 4 - optional
Configure group size and scaling
Step 5 - optional
Add notifications
Step 6 - optional
Add tags
Step 7
Review

Choose launch template or configuration Info

Specify a launch template that contains settings common to all EC2 instances that are launched by this Auto Scaling group. If you currently use launch configurations, you might consider migrating to launch templates.

Name

Auto Scaling group name
Enter a name to identify the group.

AppTierASG

Must be unique to this account in the current Region and no more than 255 characters.

Launch template Info

Switch to launch configuration

Launch template
Choose a launch template that contains the instance-level settings, such as the Amazon Machine Image (AMI), instance type, key pair, and security groups.

AppTierLaunchTemplate

Create a launch template

Version

Default (1)

Create a launch template version

Description	Launch template	Instance type
Launch template for app tier	AppTierLaunchTemplate lt-07c36c540719a646f	t2.micro

Step 2
Choose instance launch options
Step 3 - optional
Integrate with other services
Step 4 - optional
Configure group size and scaling
Step 5 - optional
Add notifications
Step 6 - optional
Add tags
Step 7
Review

Instance type requirements Info

Override launch template

You can keep the same instance attributes or instance type from your launch template, or you can choose to override the launch template by specifying different instance attributes or manually adding instance types.

Launch template	Version	Description
AppTierLaunchTemplate lt-07c36c540719a646f	Default	Launch template for app tier

Instance type
t2.micro

Network Info

For most applications, you can use multiple Availability Zones and let EC2 Auto Scaling balance your instances across the zones. The default VPC and default subnets are suitable for getting started quickly.

VPC
Choose the VPC that defines the virtual network for your Auto Scaling group.

vpc-030cb26268efc5870 (MydemoVPC)
10.0.0.0/16

Create a VPC

Availability Zones and subnets
Define which Availability Zones and subnets your Auto Scaling group can use in the chosen VPC.

Select Availability Zones and subnets

use1-az1 (us-east-1a) | subnet-03220334593e6148e (Private-App-Subnet-AZ-1)
10.0.3.0/24

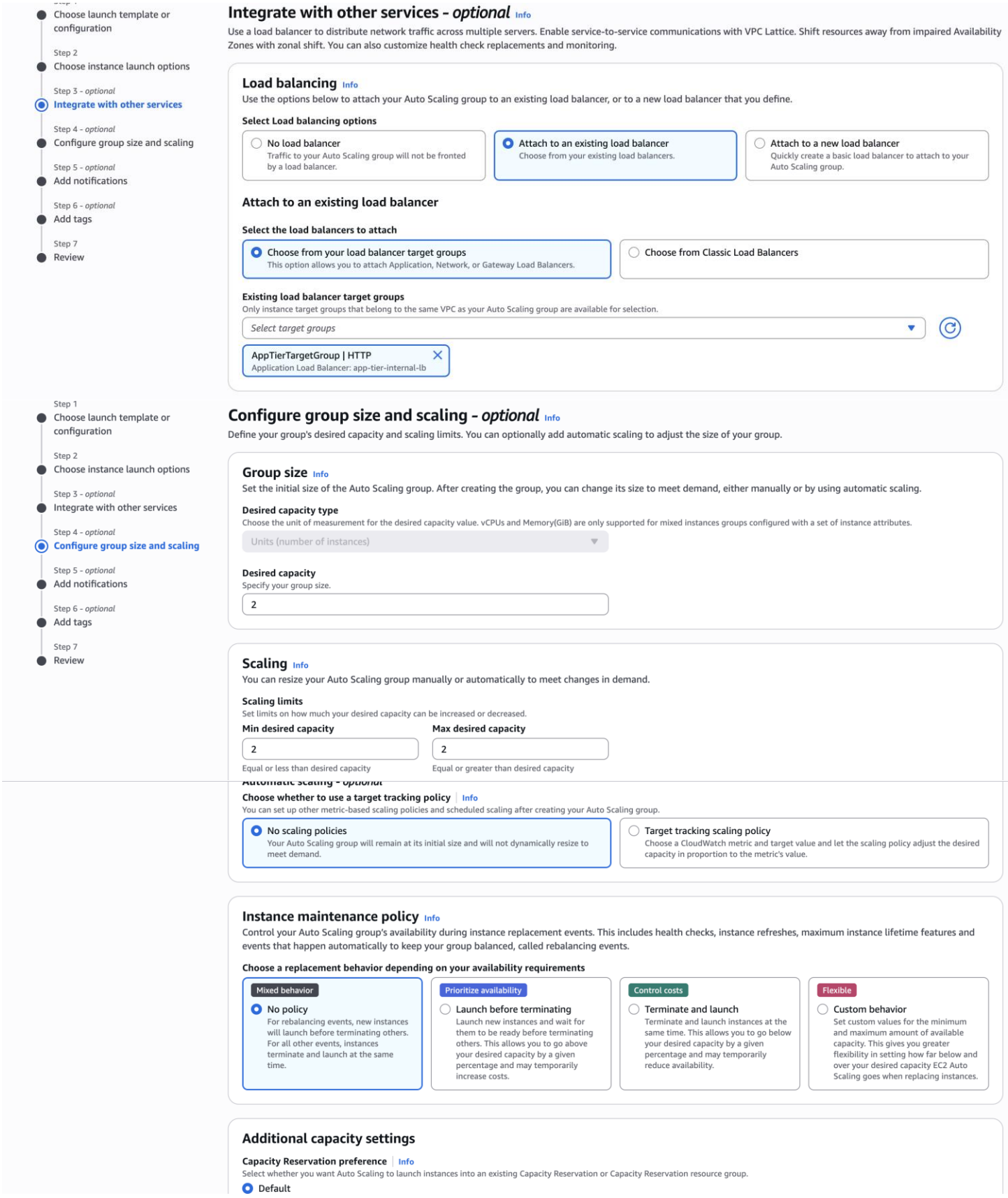
use1-az2 (us-east-1b) | subnet-01604a8aab303c194 (Private-App-Subnet-AZ-2)
10.0.4.0/24

Create a subnet

Availability Zone distribution - *new*
Auto Scaling automatically balances instances across Availability Zones. If launch failures occur in a zone, select a strategy.

- (●) Balanced best effort
  If launches fail in one Availability Zone, Auto Scaling will attempt to launch in another healthy Availability Zone.
- ( ) Balanced only
  If launches fail in one Availability Zone, Auto Scaling will continue to attempt to launch in the unhealthy Availability Zone to preserve balanced distribution.

Cancel | Skip to review | Previous | Next



Step-11: Web Tier Instance Deployment

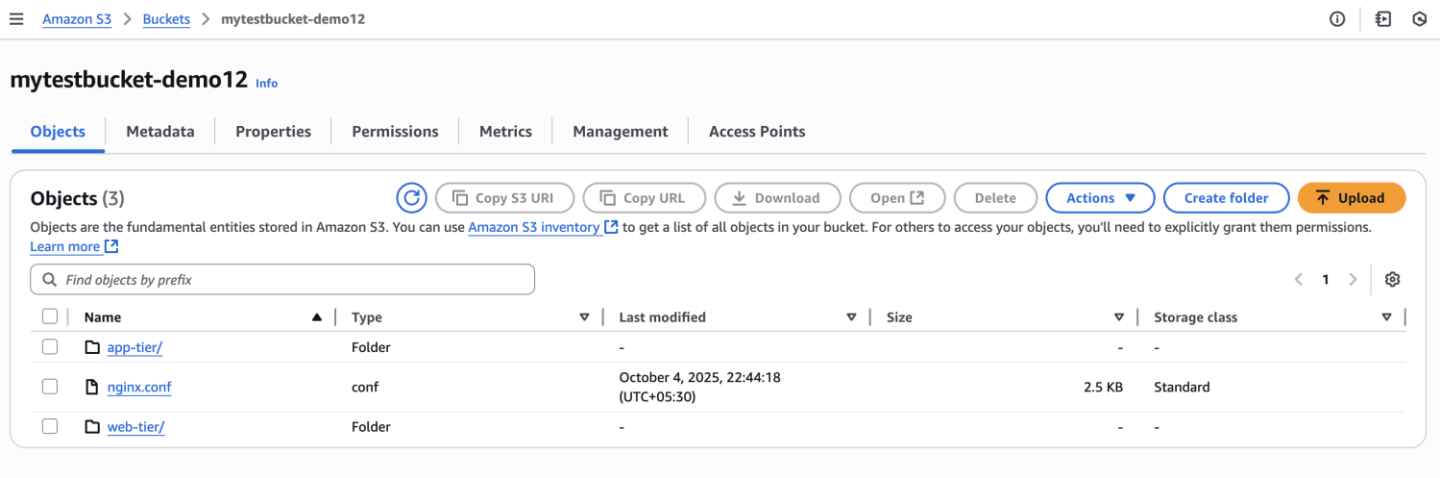
- Before we create and configure the web instances, open up the application-code/nginx.conf file from the repo we downloaded. Scroll down to line 58 and replace [INTERNAL-LOADBALANCER-DNS] with your

internal load balancer’s DNS entry. You can find this by navigating to your internal load balancer's details page.

nginx.conf — Edited

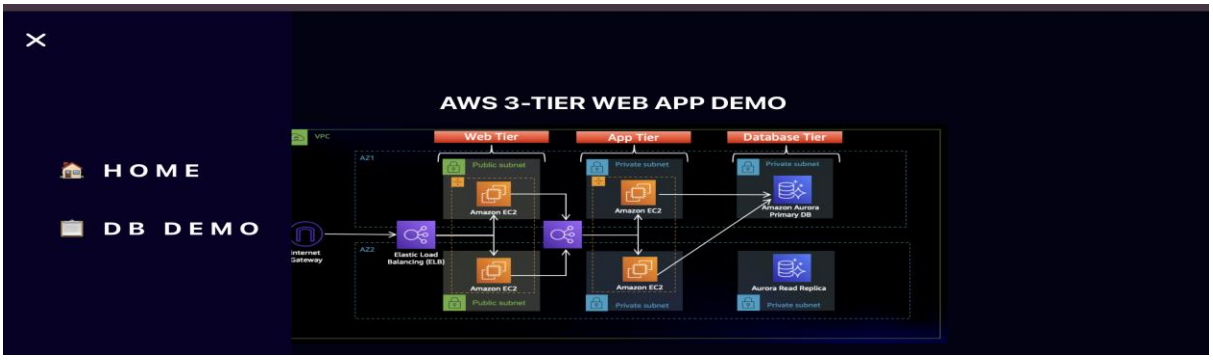
```
        listen       [::]:80;
        server_name  _;

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

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

        #proxy for internal lb
        location /api/{
                proxy_pass http://app-tier-internal-lb-52025663.us-east-1.elb.amazonaws.com:80/;
        }


    }

# Settings for a TLS enabled server.
#
#    server {
#        listen       443 ssl http2;
#        listen       [::]:443 ssl http2;
#        server_name  _;
#        root         /usr/share/nginx/html;
#
#        ssl_certificate "/etc/pki/nginx/server.crt";
#        ssl_certificate_key "/etc/pki/nginx/private/server.key";
#        ssl_session_cache shared:SSL:1m;
```

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



- Set the Name and Tag. Select Amazon Linux for the OS and Amazon Linux 2 AMI (HVM) - Kernel 5.10 for the AMI. For Instance Type, select t2.micro. (Select Proceed without a key pair.) For Network settings, click Edit.
- Select a VPC and a subnet. For Subnet, select Public-Web-Subnet. For Security Group, select WebTierSG, which you created in the previous step. For Auto-assign public IP, select Enable. click Advanced details.
- In the IAM instance profile field, select the IAM Role you created in the previous step. (In our example, we created the EC2andS3Role.) Finally, click Launch instance to create the instance.
- 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
- 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
- Now we need to download our web tier code from our s3 bucket:
    - Cd ~/
    - aws s3 cp s3://mytestbucket-demo12/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
- 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 dnf install nginx -y
- 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 /etc/nginx/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
- Now when you plug in the public IP of your web tier instance, you should see your website, which you can find on the Instance details page on the EC2 dashboard. If you have the database connected and working correctly, then you will also see the database working. You'll be able to add data. Careful with the delete button, that will clear all the entries in your database.



Step-12: External Load Balancer and Auto Scaling

- 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.
- 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.
- 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.
- The purpose of forming this target group is to use with our load blancer so it may balance traffic across our public web tier instances. Select Instances as the target type and give it a name.
- Then, set the protocol to HTTP and the port to 80. Remember this is the port NGINX is listening on. Select the VPC we've been using thus far, and then change the health check path to be /health. Click Next.
- We are NOT going to register any targets for now, so just skip that step and create the target group.
- On the left hand side of the EC2 dashboard select Load Balancers under Load Balancing and click Create Load Balancer.
- We'll be using an Application Load Balancer for our HTTP traffic so click the create button for that option.
- After giving the load balancer a name, be sure to select internet facing 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 public 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.
- 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.
- 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 web tier, and then under Advanced details use the same IAM instance profile we have been using for our EC2 instances.

Create launch template

Creating a launch template allows you to create a saved instance configuration that can be reused, shared and launched at a later time. Templates can have multiple versions.

Launch template name and description

Launch template name - *required*

WebTierLaunchTemplate

Must be unique to this account. Max 128 chars. No spaces or special characters like '&', '*', '@'.

Template version description

Launch template for web tier

Max 255 chars

Auto Scaling guidance | Info

Select this if you intend to use this template with EC2 Auto Scaling

☐ Provide guidance to help me set up a template that I can use with EC2 Auto Scaling

▶ **Template tags**

▶ **Source template**

Launch template contents

Specify the details of your launch template below. Leaving a field blank will result in the field not being included in the launch template.

Application and OS Images (Amazon Machine Image) Info

An AMI contains the operating system, application server, and applications for your instance. If you don't see a suitable AMI below, use the search field or choose **Browse more AMIs**.

Search our full catalog including 1000s of application and OS images

Recents | My AMIs | Quick Start

○ Don't include in launch template | ◉ Owned by me | ○ Shared with me

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

WebTierImage
ami-08abd5b75a525f90c
2025-10-04T18:24:38.000Z Virtualization: hvm ENA enabled: true Root device type: ebs Boot mode: uefi-preferred

Description

Image of our web tier instance

Instance type Info | Get advice

Advanced

Instance type

t2.micro
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand Ubuntu Pro base pricing: 0.0134 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour On-Demand RHEL base pricing: 0.026 USD per Hour
On-Demand Linux base pricing: 0.0116 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name

Don't include in launch template

Create new key pair

Network settings Info

Subnet | Info

Don't include in launch template

Create new subnet

When you specify a subnet, a network interface is automatically added to your template.

Summary

Software Image (AMI)
Image of our web tier instance...read more
ami-08abd5b75a525f90c

Virtual server type (instance type)
t2.micro

Firewall (security group)
Web-tier-sg

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet. Data transfer charges are not included

Summary

Software Image (AMI)
Image of our web tier instance...read more
ami-08abd5b75a525f90c

Virtual server type (instance type)
t2.micro

Firewall (security group)
Web-tier-sg

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet. Data transfer charges are not included

Cancel | Create launch template

Summary

Software Image (AMI)
Image of our web tier instance...read more
ami-08abd5b75a525f90c

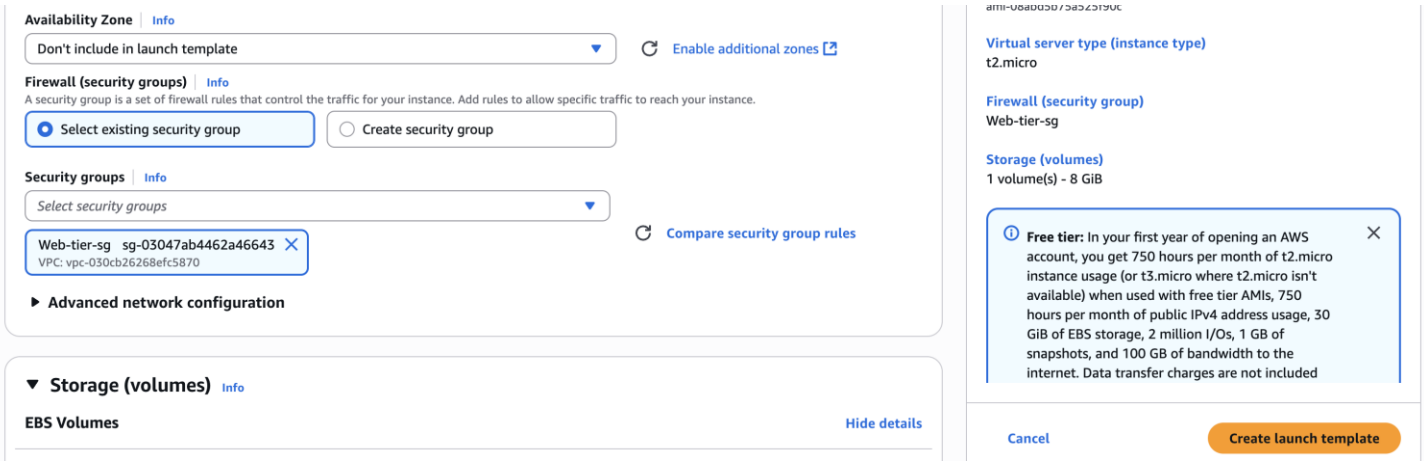
Virtual server type (instance type)
t2.micro

Firewall (security group)
Web-tier-sg

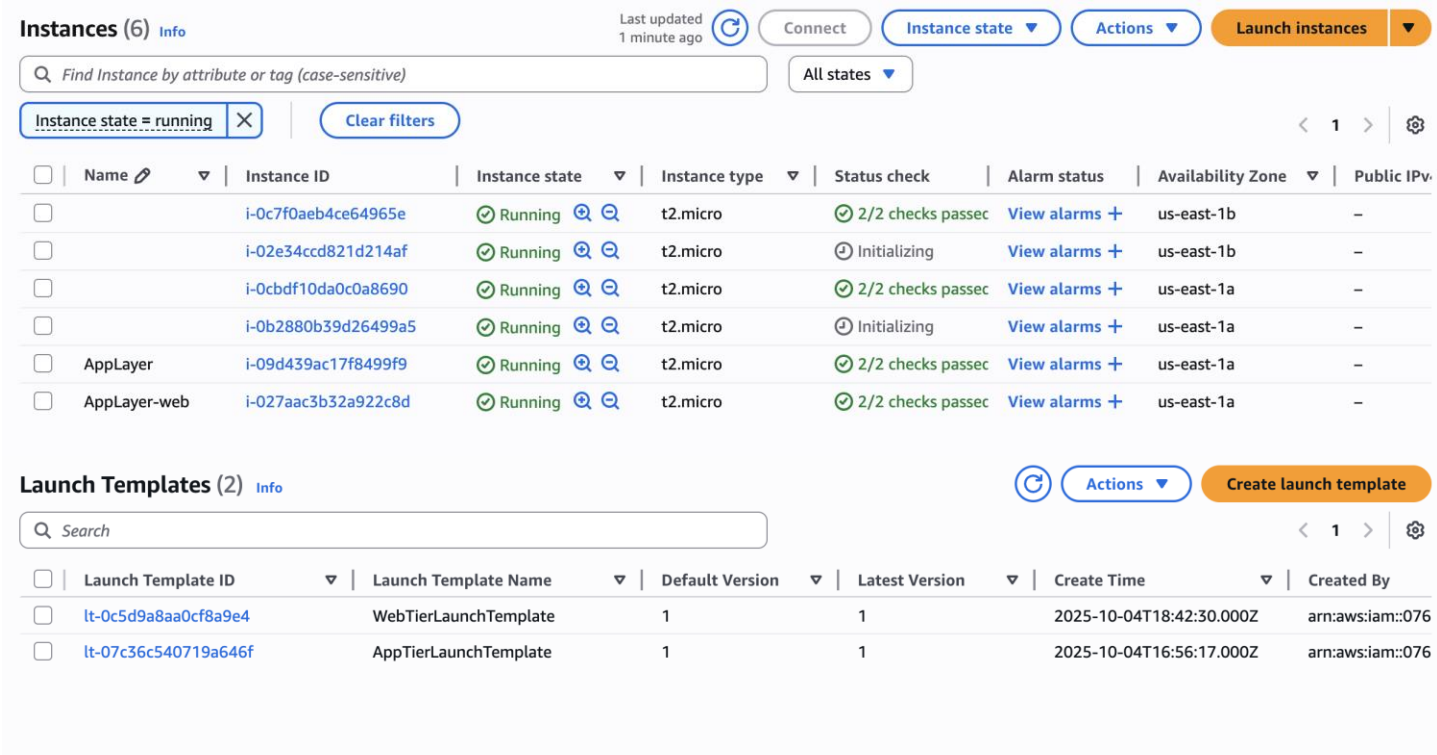
Storage (volumes)
1 volume(s) - 8 GiB

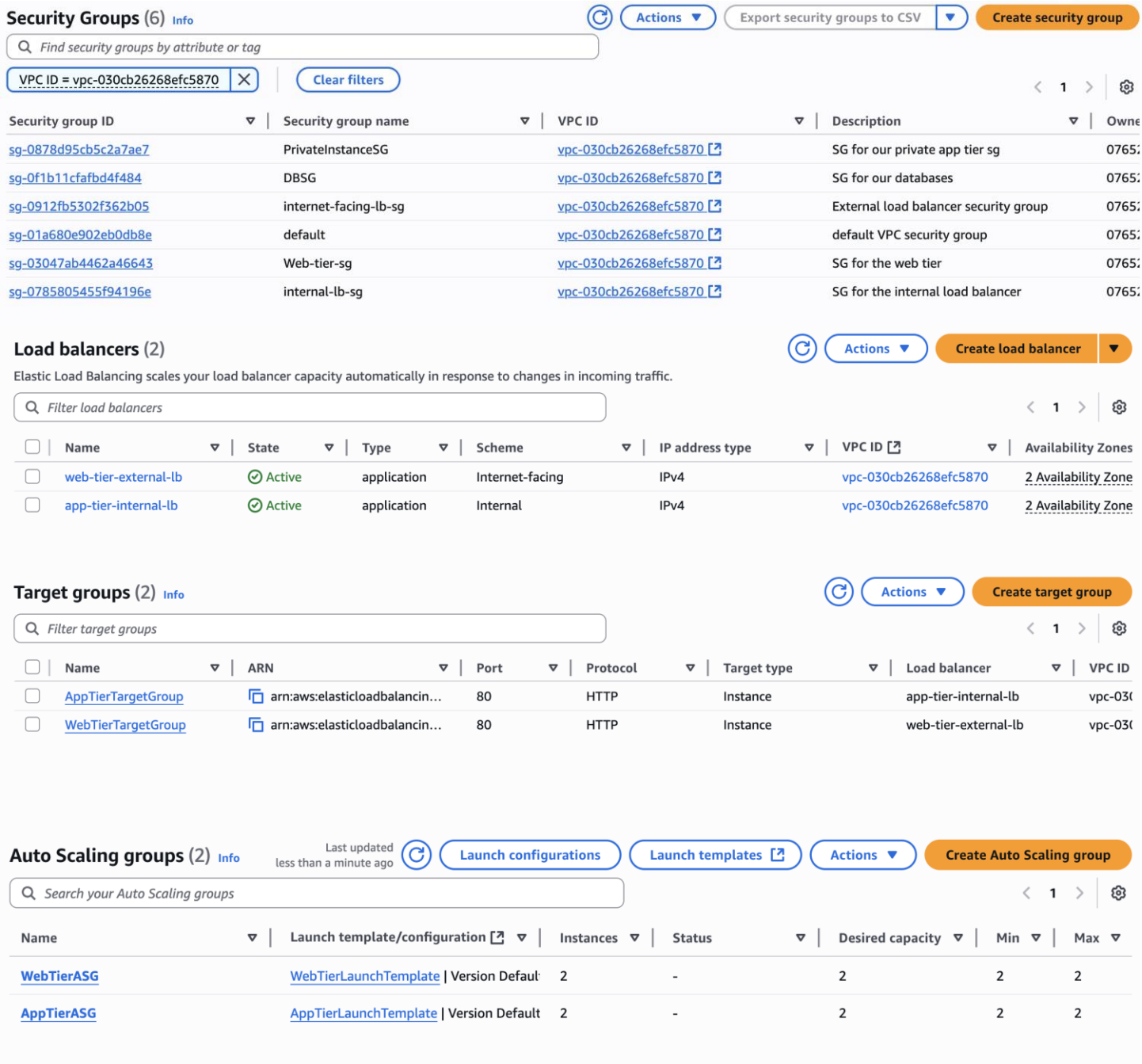
Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet. Data transfer charges are not included

Cancel | Create launch template



- We will now create the Auto Scaling Group for our web instances. On the left side of the EC2 dashboard navigate to Auto Scaling Groups under Auto Scaling and click Create Auto Scaling group.
- Give your Auto Scaling group a name, and then select the Launch Template we just created and click next.
- On the Choose instance launch options page set your VPC, and the public 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 web tier load balancer's target group from the dropdown. Then, click next.
- 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!