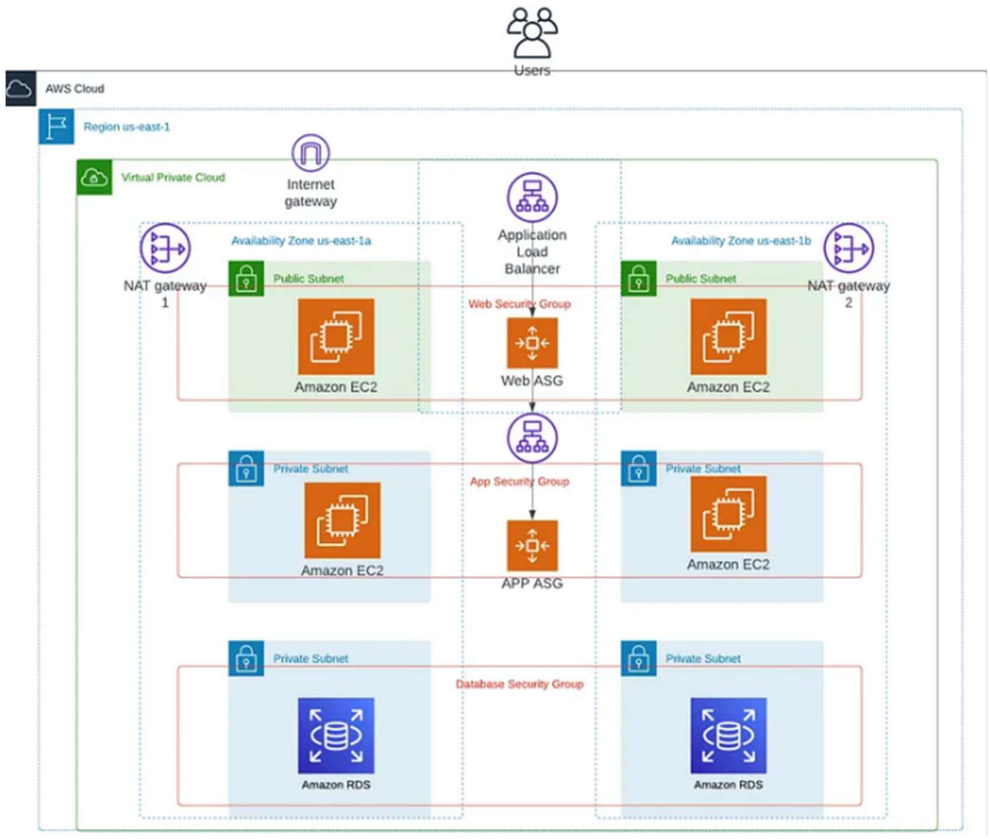
AWS 3-Tier Architecture: Scalable and Highly Available Web Application Deployment.

(by - Nikhil Mahesh Shebannavar)



Introduction

Deploying web applications with high availability, scalability, and security is a critical requirement in modern cloud infrastructure. This project demonstrates the implementation of a 3-Tier Architecture on Amazon Web Services (AWS) to host a web application in a production-ready environment.

The 3-Tier model is a widely adopted architectural pattern that separates application components into three distinct layers:

- Presentation Tier – the frontend, handling user interaction.
- Application Tier – the business logic or backend processing.
- Database Tier – the data layer that stores and manages application data.

This architecture was deployed across multiple Availability Zones (AZs) in AWS to ensure high availability, with components such as Elastic Load Balancers (ELBs), Auto Scaling Groups (ASGs), RDS (Relational Database Service), and secure networking using VPCs.

By implementing this solution, the project aims to simulate real-world cloud infrastructure practices suitable for hosting modern, scalable, and fault-tolerant web applications.

Why 3 tier architecture

The 3-tier architecture is a widely adopted design pattern in web application development and cloud deployments due to its modularity, scalability, and security benefits. It divides the application into three separate layers, each responsible for specific functions:

1. Presentation Tier (Frontend)

This is the user interface layer — typically a web browser or client app that interacts with users. Hosting this tier separately allows independent scaling based on user traffic.

2. Application Tier (Logic Layer)

This tier handles the core functionality and business logic. It processes user inputs, manages sessions, and communicates between the frontend and database. Isolating logic makes the application easier to maintain and scale.

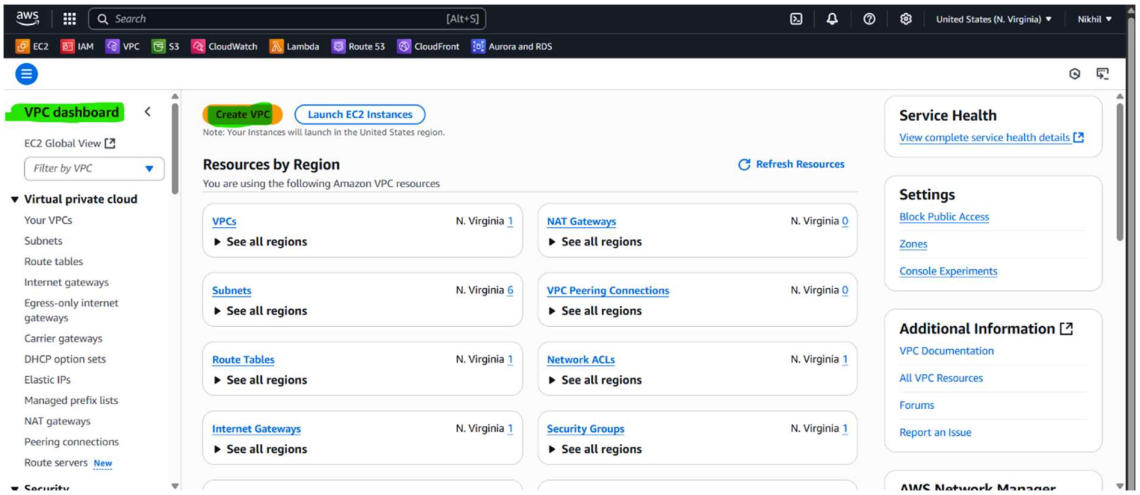
3. Database Tier (Data Layer)

Responsible for storing, retrieving, and managing data. It ensures data integrity, security, and availability. Keeping the database separate improves security and performance.

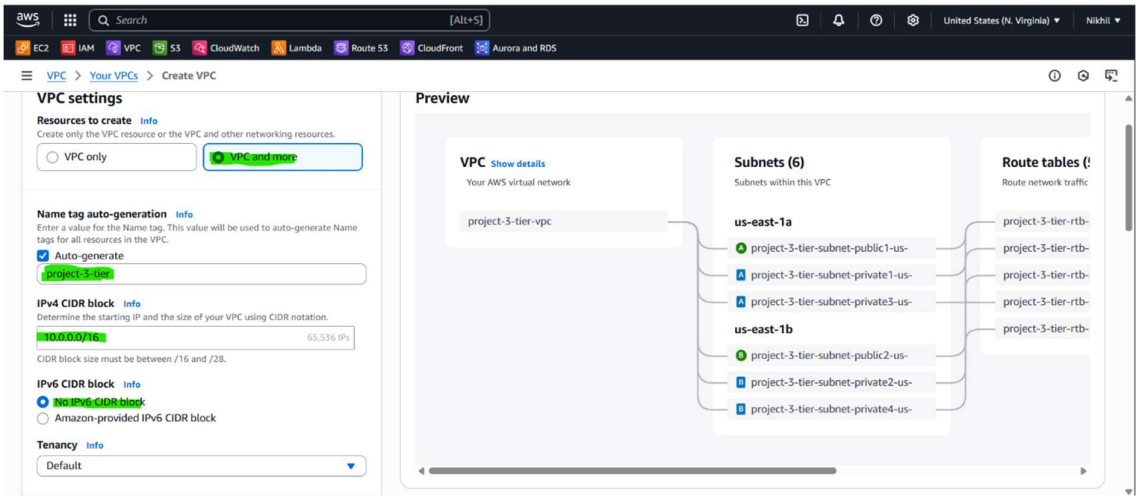
Task 1: Creating a VPC and Subnets

Using the architecture diagram as a reference, I have created new VPC with 2 public subnets and 4 private subnets.

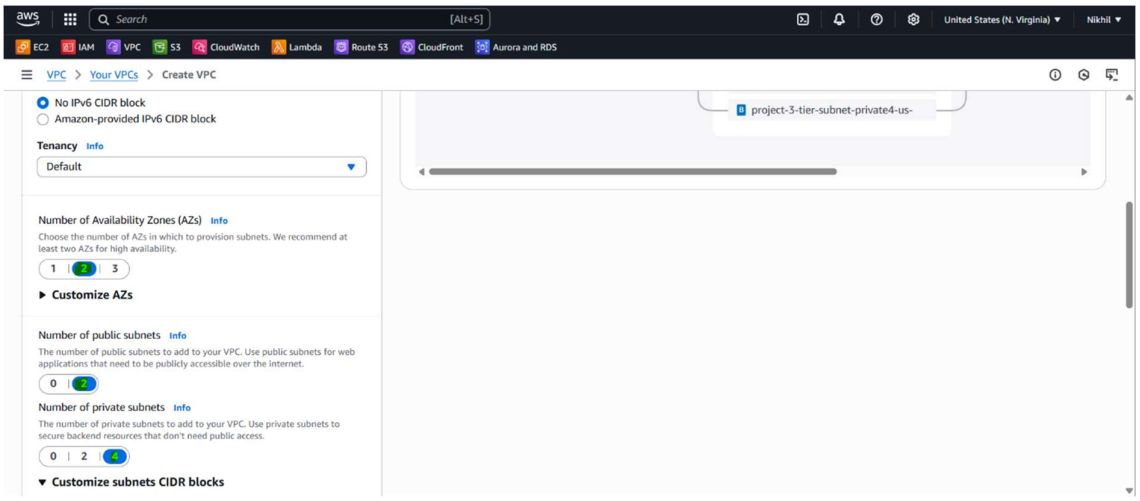
In VPC dashboard click on Create VPC.



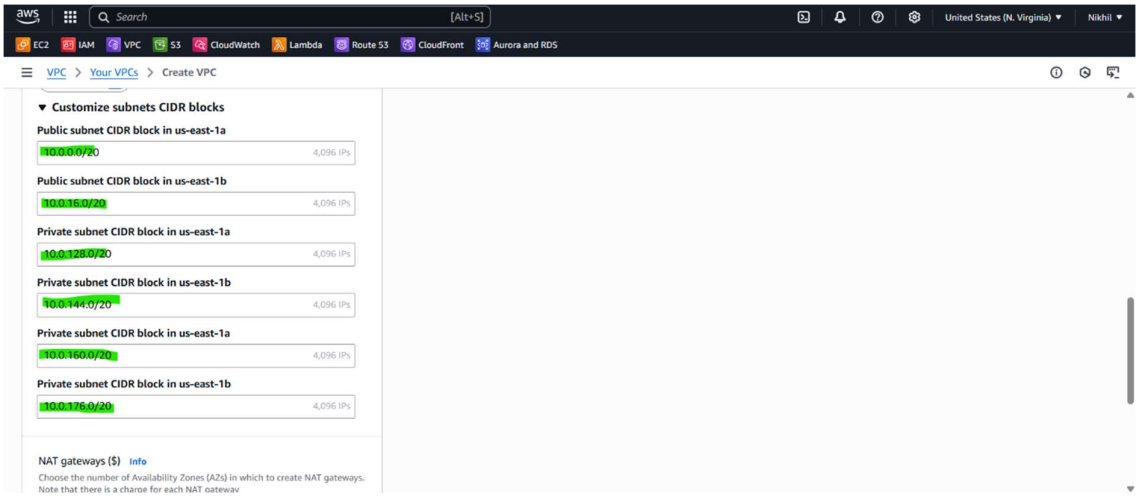
Select VPC and more, name the VPC, IPV4 CIDR block I have taken starting IP and size as 10.0.0.0/16 and IPV6 is not needed.



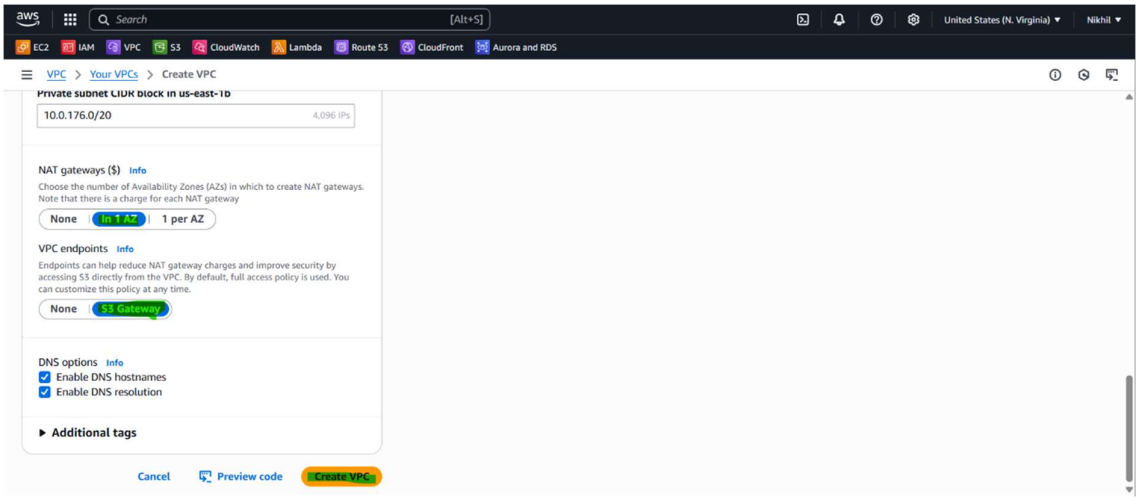
Select 2 AZs, Number of public subnets as 2 and Number of private subnets as 4



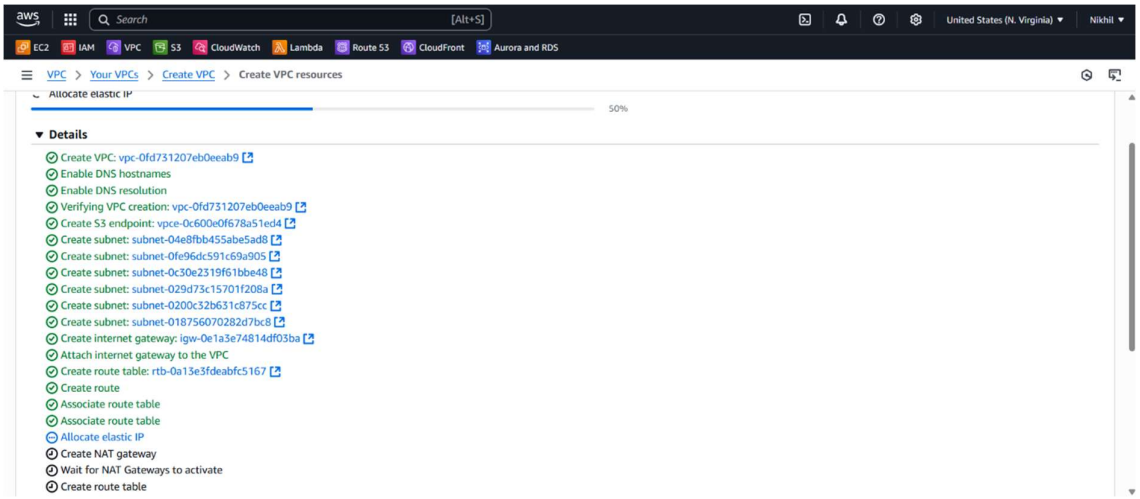
In customize subnet I have given 6 different IP for all subnets.



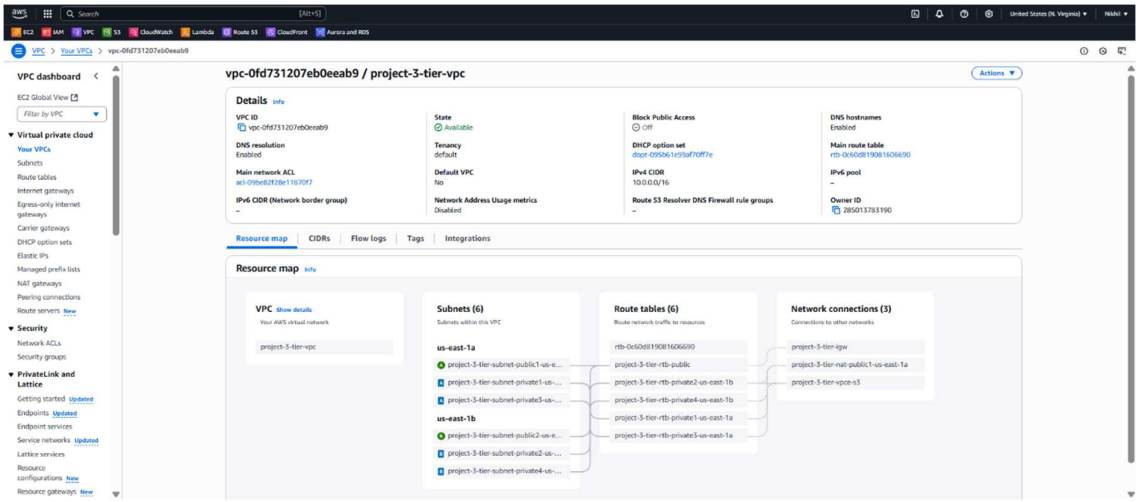
NAT gateway in 1 AZ. Click on create VPC



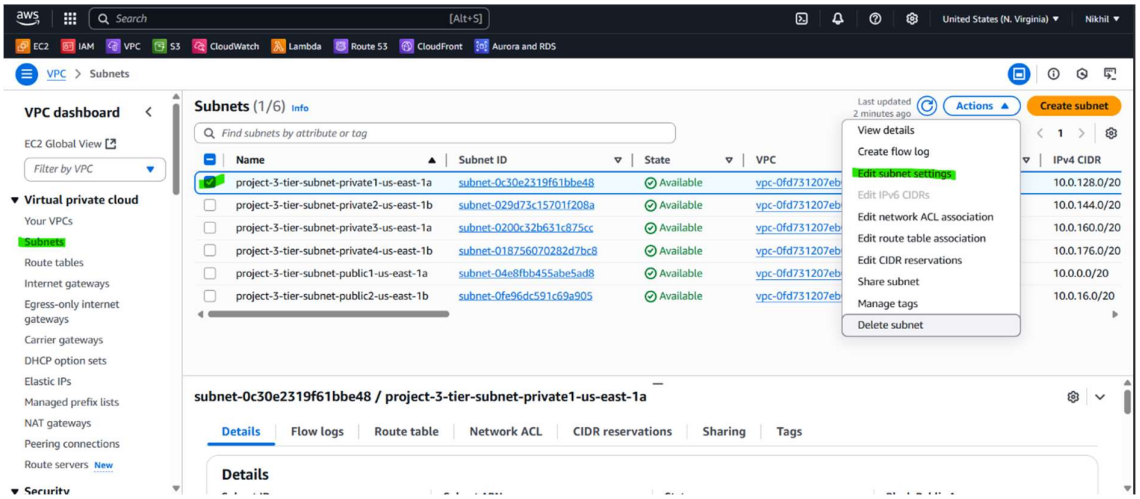
VPC is creating.



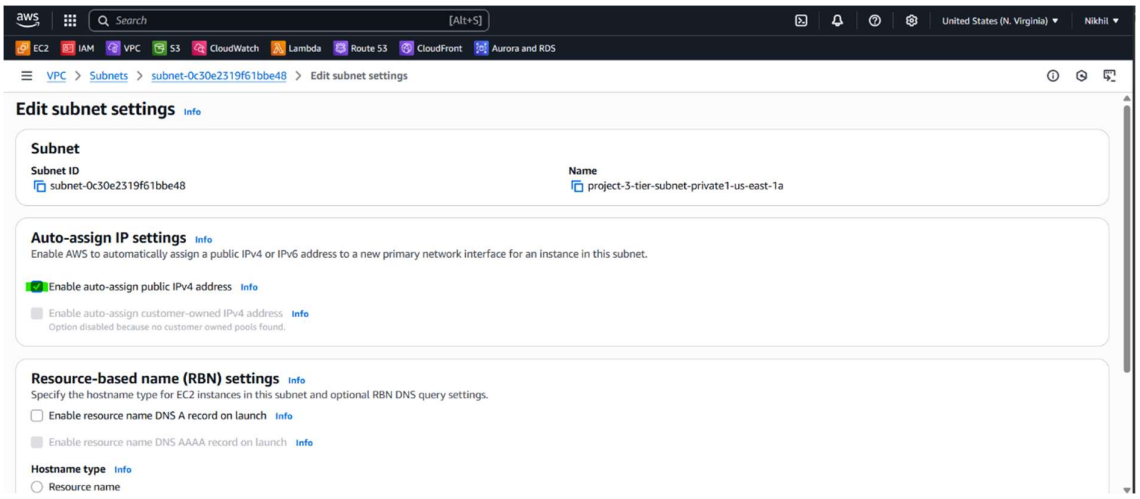
VPC with 2 public and 4 private subnets are created, also we can check resource map.



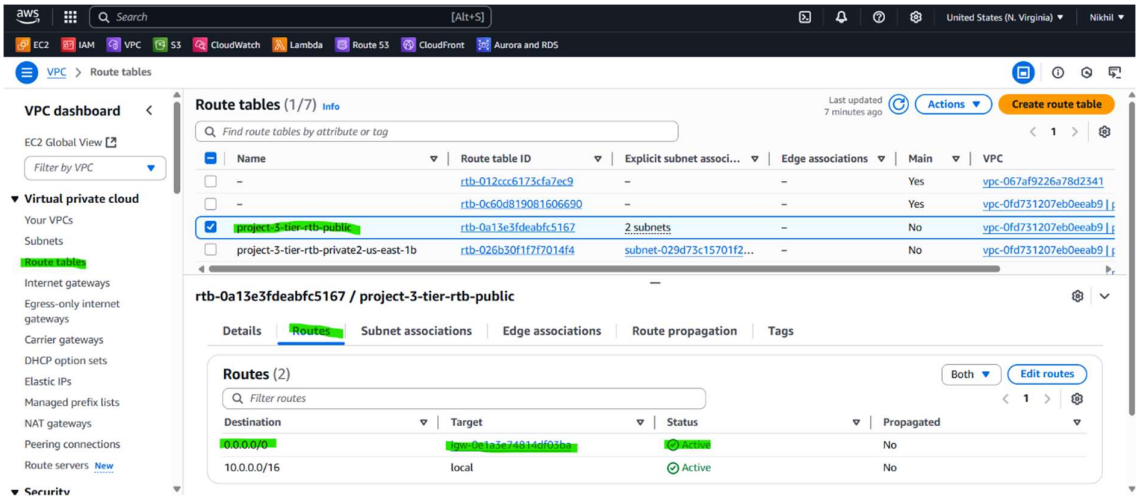
Now I changed the subnet setting to enable auto generate IPs, to do click on subnets in VPC dashboard, select one subnet, click on actions and click edit subnet settings



Now, Enable auto assign public IPv4 address and save. I have done this to all 6 subnets.



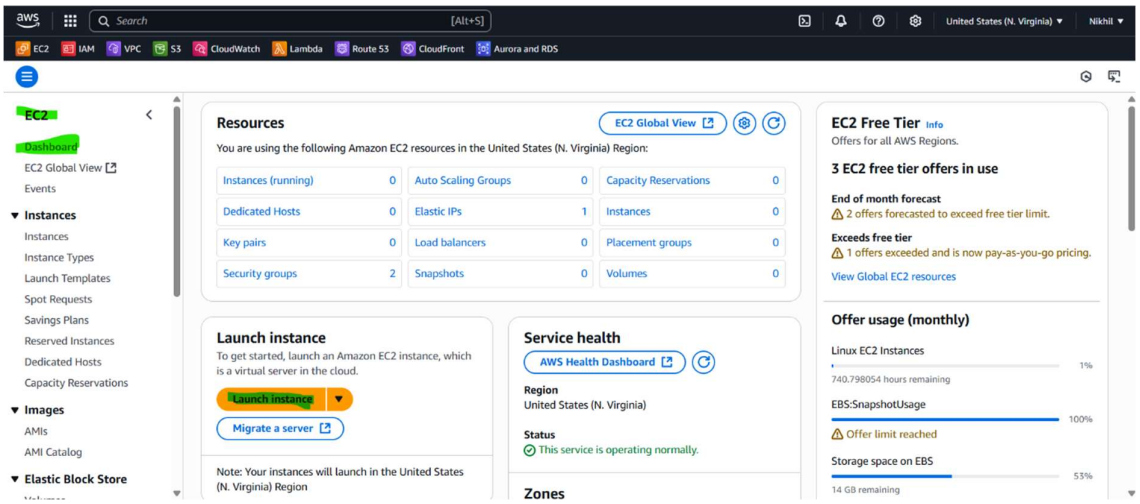
Check the route table if it is associated with correct subnets.



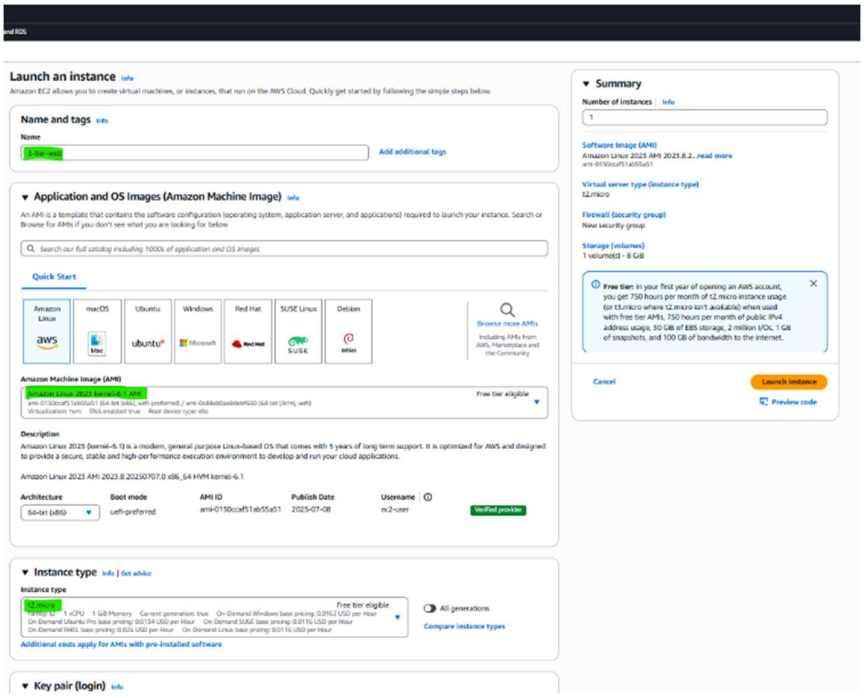
Task 2: Creating a Web Server Tier

Next, I have created EC2 Instance for first tire i.e web tier(front end), then I have created an auto scaling group of EC2 instance that will host webpage. Starting with creating EC2 instance.

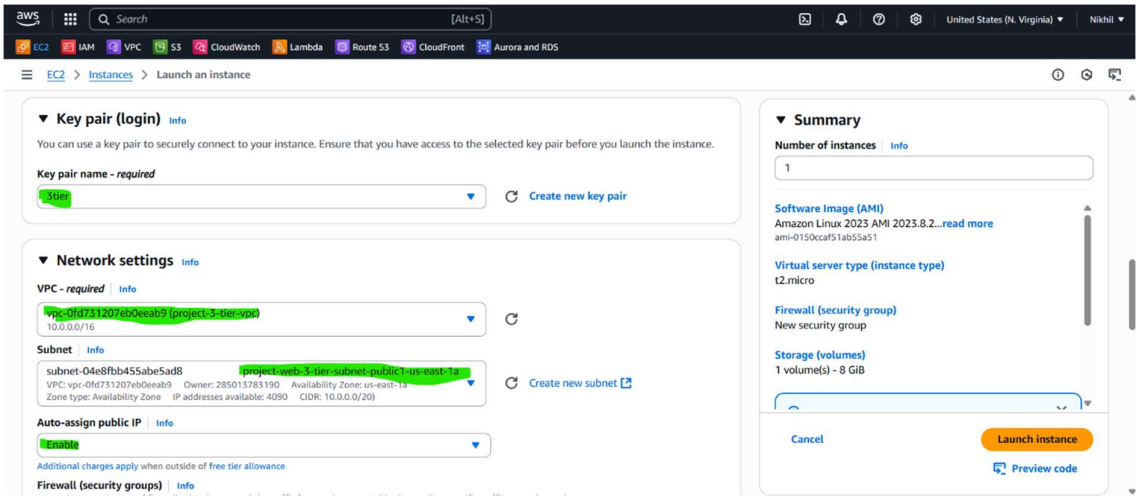
Go to EC2 dashboard, click on Launch Instance.



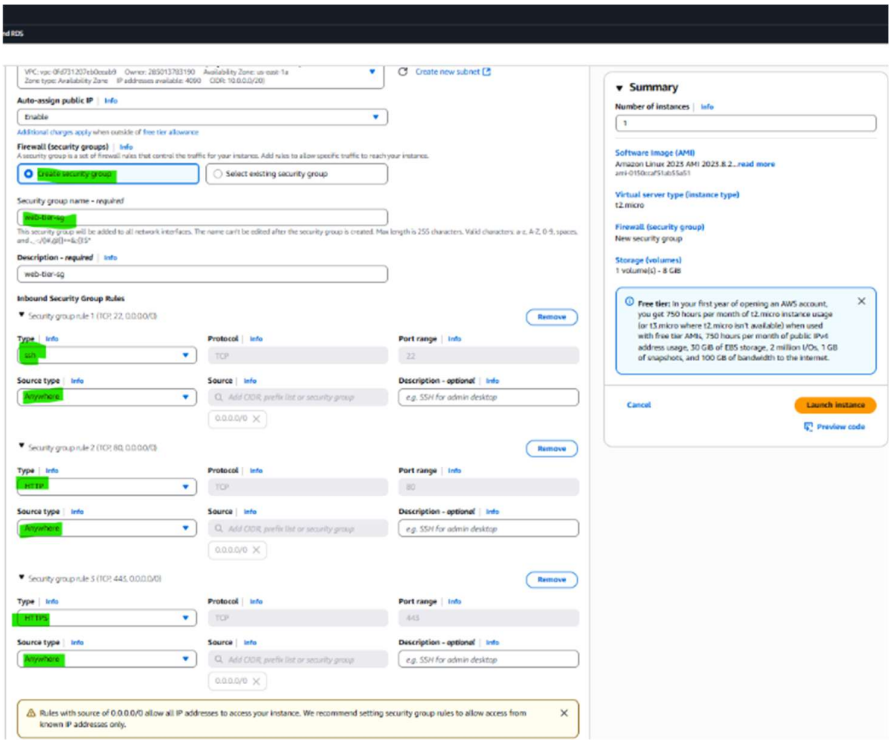
Named my instance, selected an AMI(I have selected Amazon Linux updated version) and instance type as T2.micro.



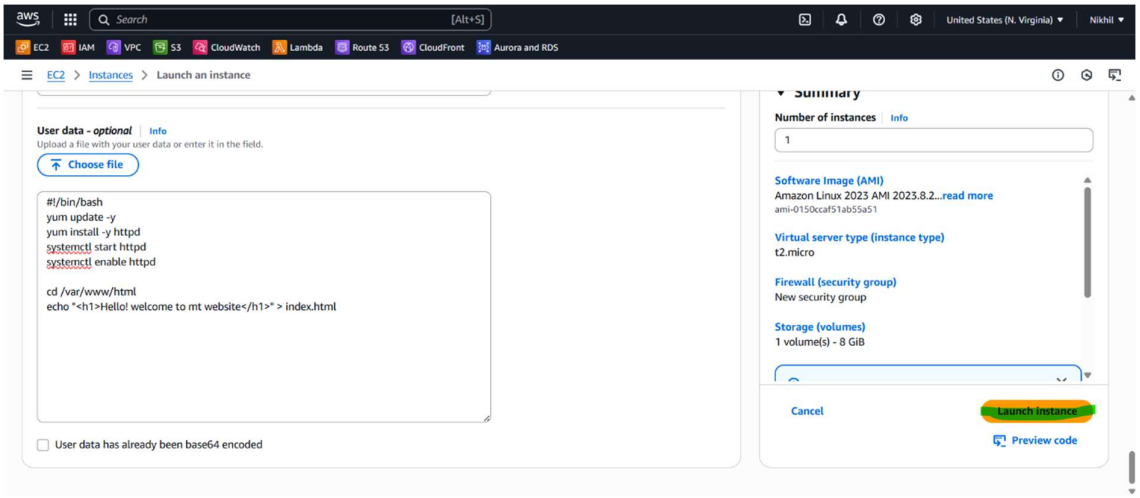
Select the key pair or create new one, select the new VPC and the correct subnet(public) and enable auto assign IP.



Create a new security group. For inbound security groups rules, add rules for ssh, HTTP, HTTPS from everywhere(not safe but for demo project it is fine).



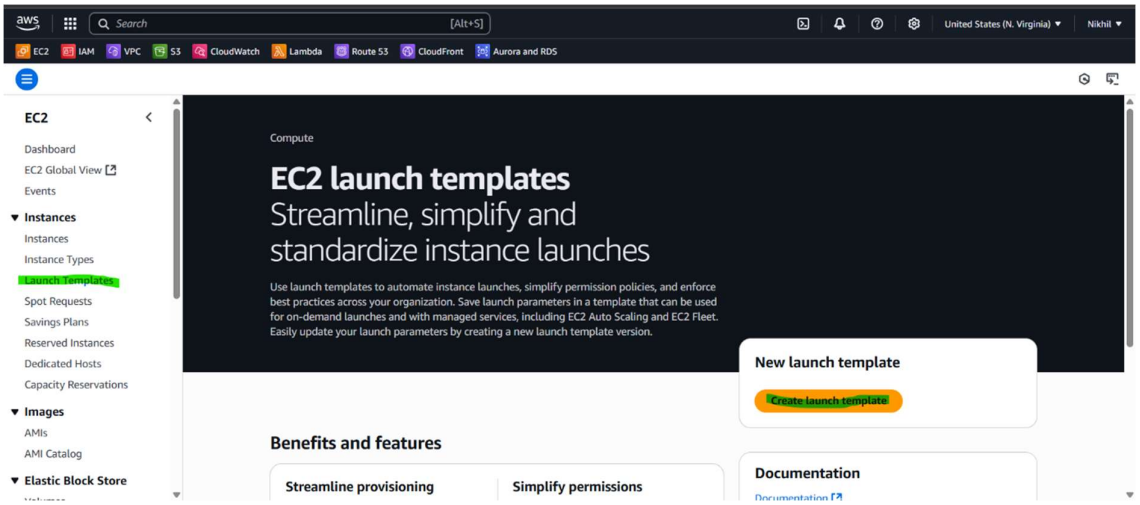
I have not changed anything for storage settings, Now, In the advanced detailed, at bottom, I ran the script to launch an Apache web server when instance starts. Click on launch instance



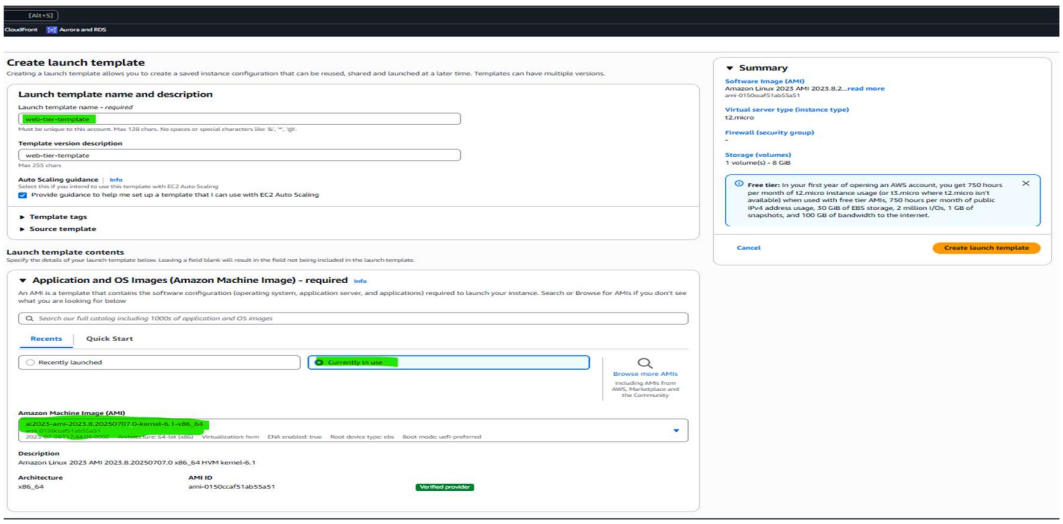
Once I launched the EC2 instance, copied the Public IP and Pasted on web server.

I can see that website is created in Apache server and It's working fine.

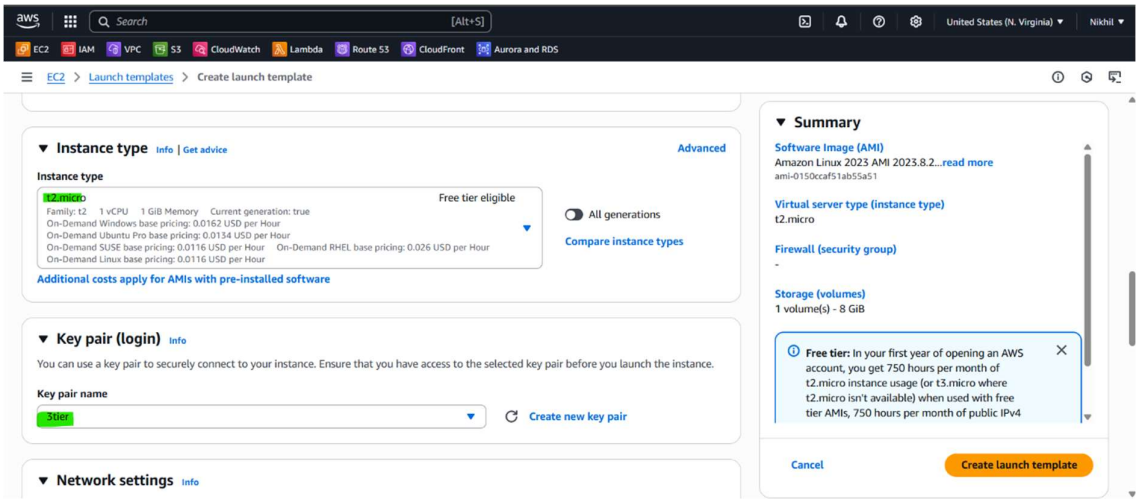
This project needs auto scaling group that I will attach to EC2 instance. This will increase reliability and availability. Next, I have created a launch template. Under EC2 dashboard, select launch templates, and click create launch template.



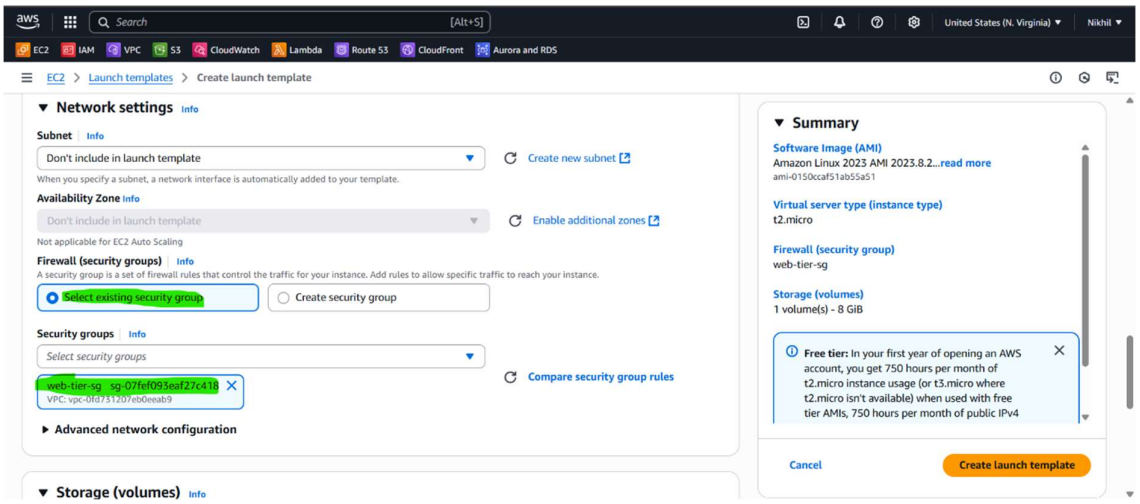
Name the template, and check the box to provide guidance.



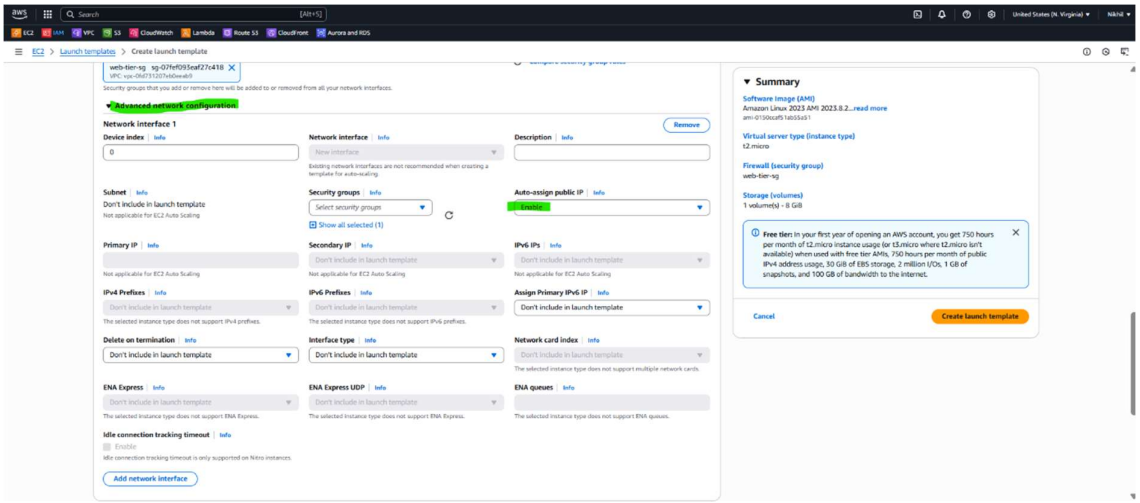
Used recently launched AMI T2.micro instance type and selected key pair.



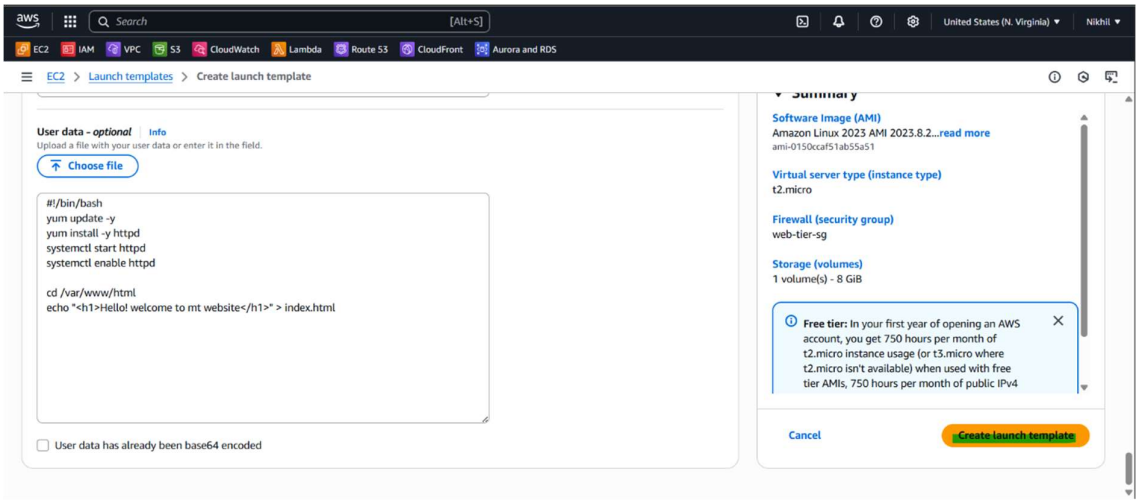
Selected existing security group (web tire sg).



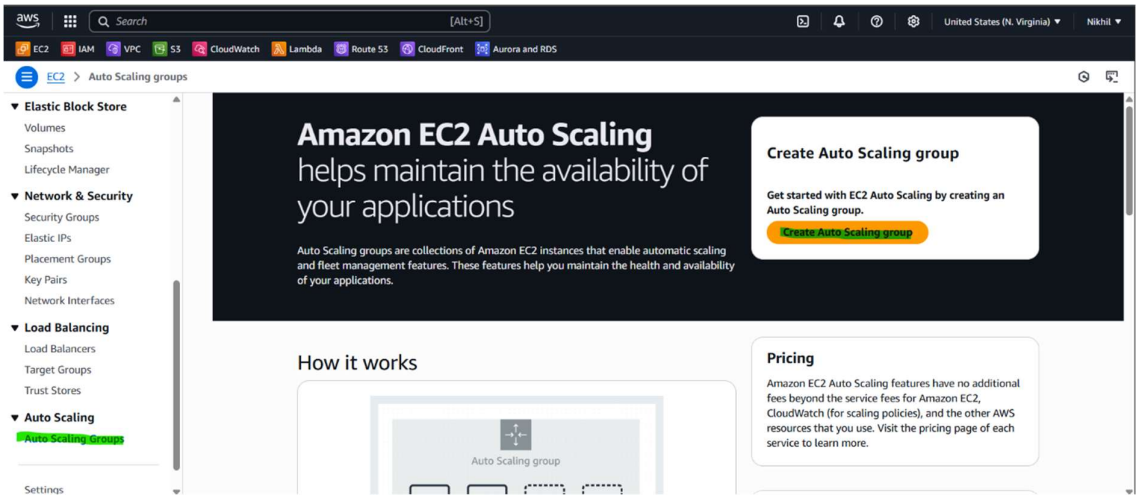
Under advanced network configuration, enable auto assign public IP.



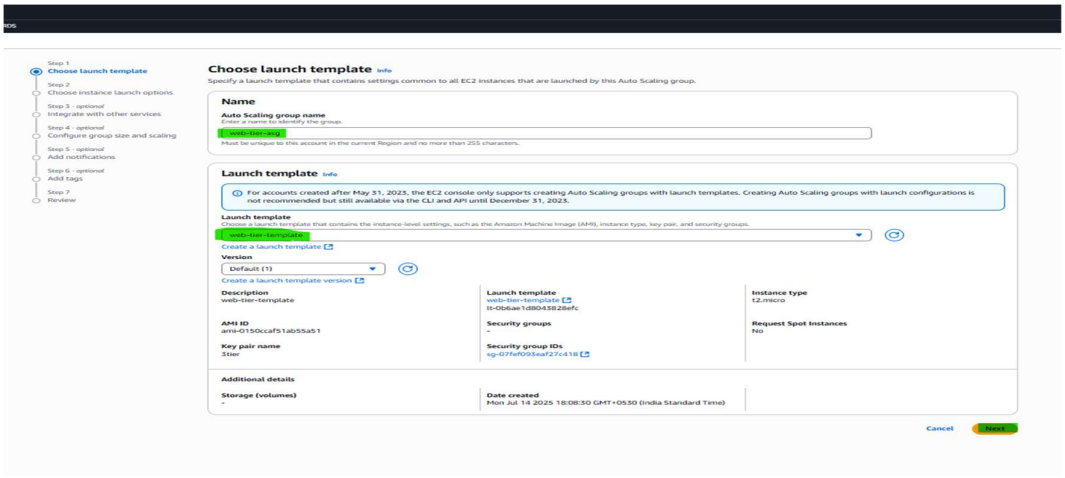
Click on the advanced details, at bottom enter the same script as I did earlier for EC2 instance, then click create launch template.



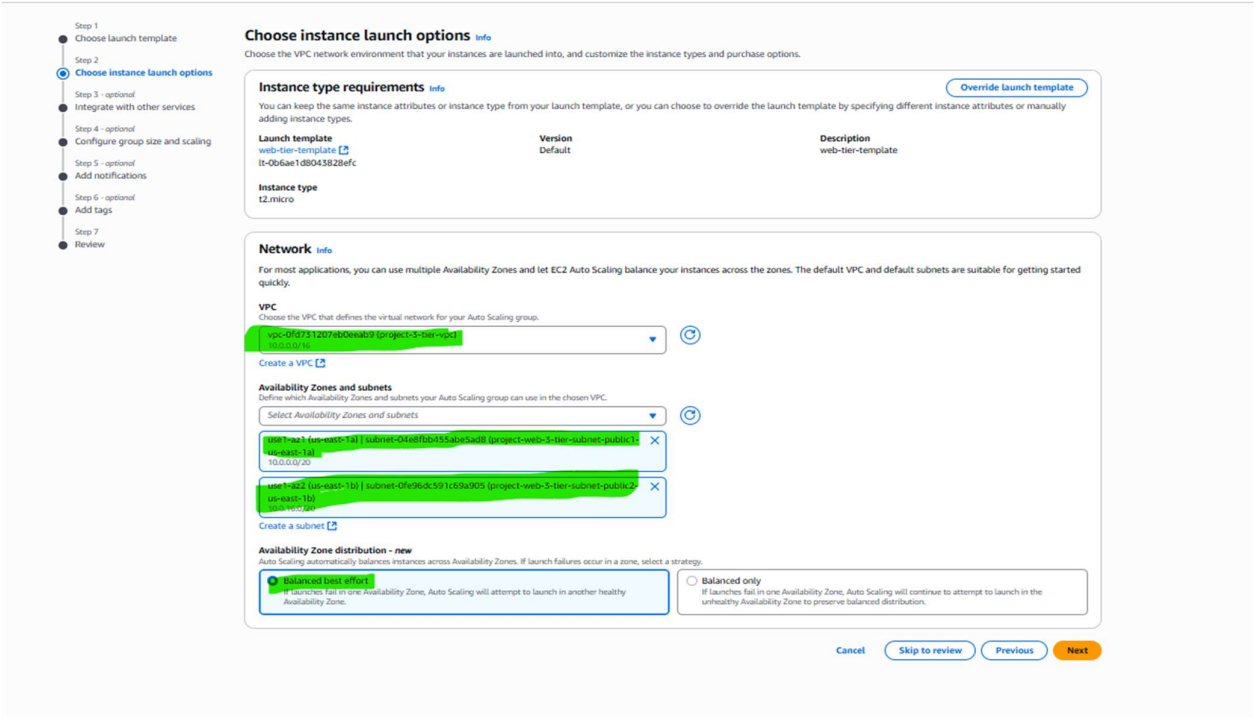
Now, I created auto scaling group. Click create auto scaling group,



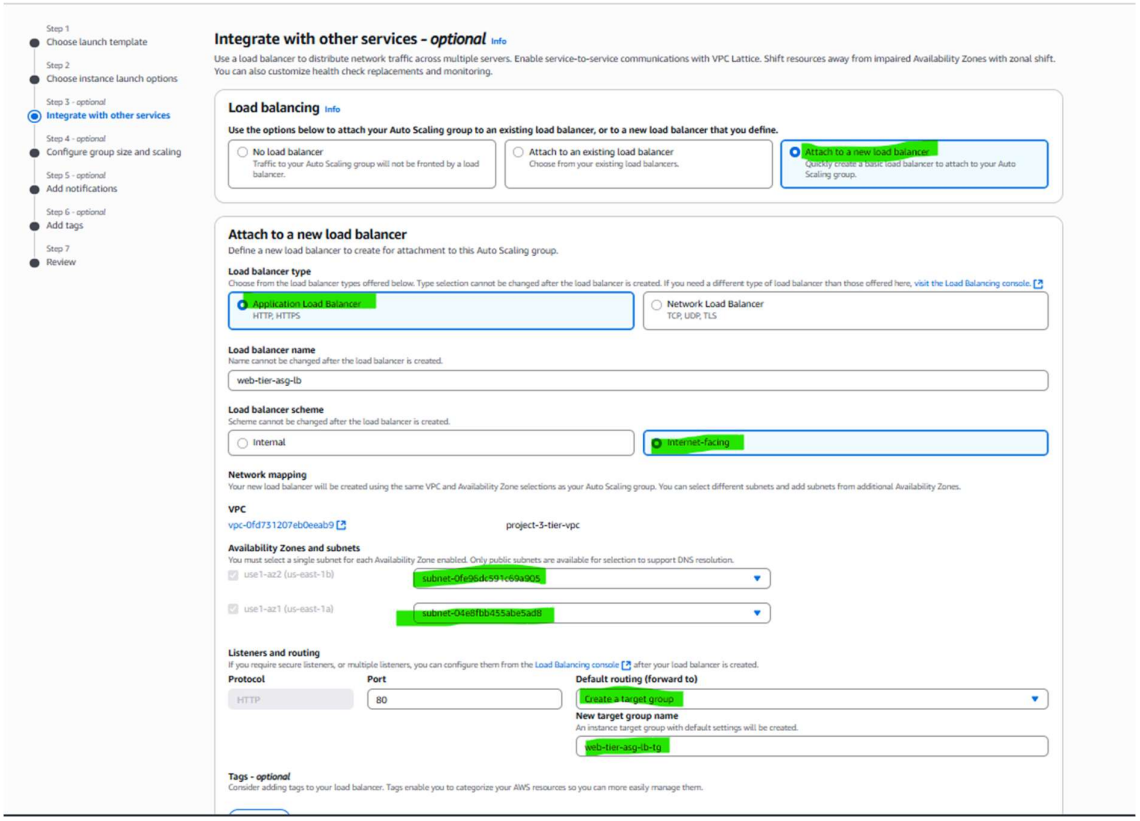
Name the ASG, choose the launch template that created just now, then click next.



Under Network, select VPC that created earlier, under availability zones and subnets select the public subnets that created , select Balanced best effort click next.



Now, click on attach a new load balancer, select Application load balancer, name the load balancer, selected internet facing, VPC and subnets are already be selected. Under the listeners and routing section select create a target group which should be on port 80 for HTTP traffic.

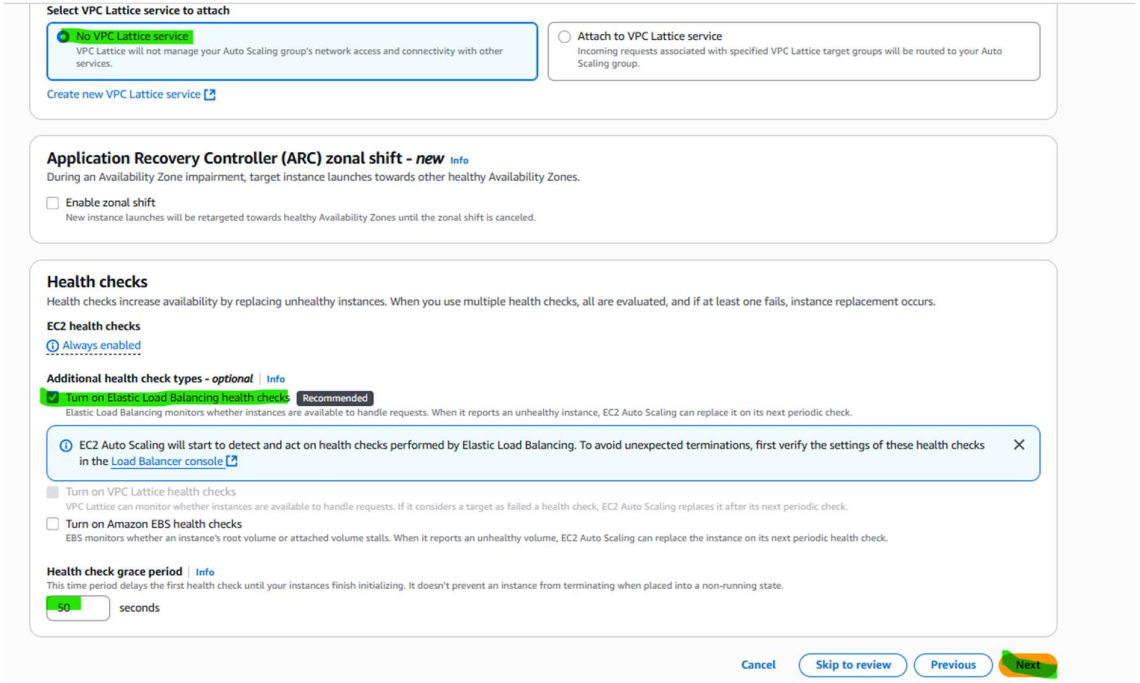


Leave no VPC lattice service checked.

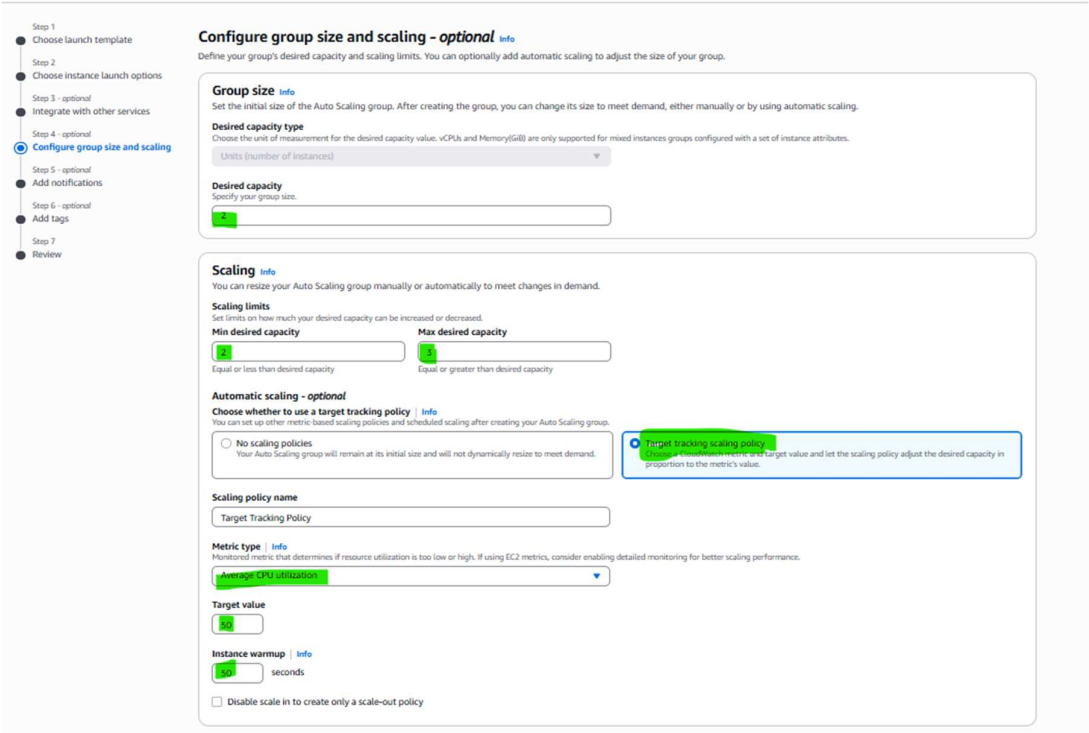
Click to turn on elastic load balancing health checks.

I have entered health check grace period as 50.

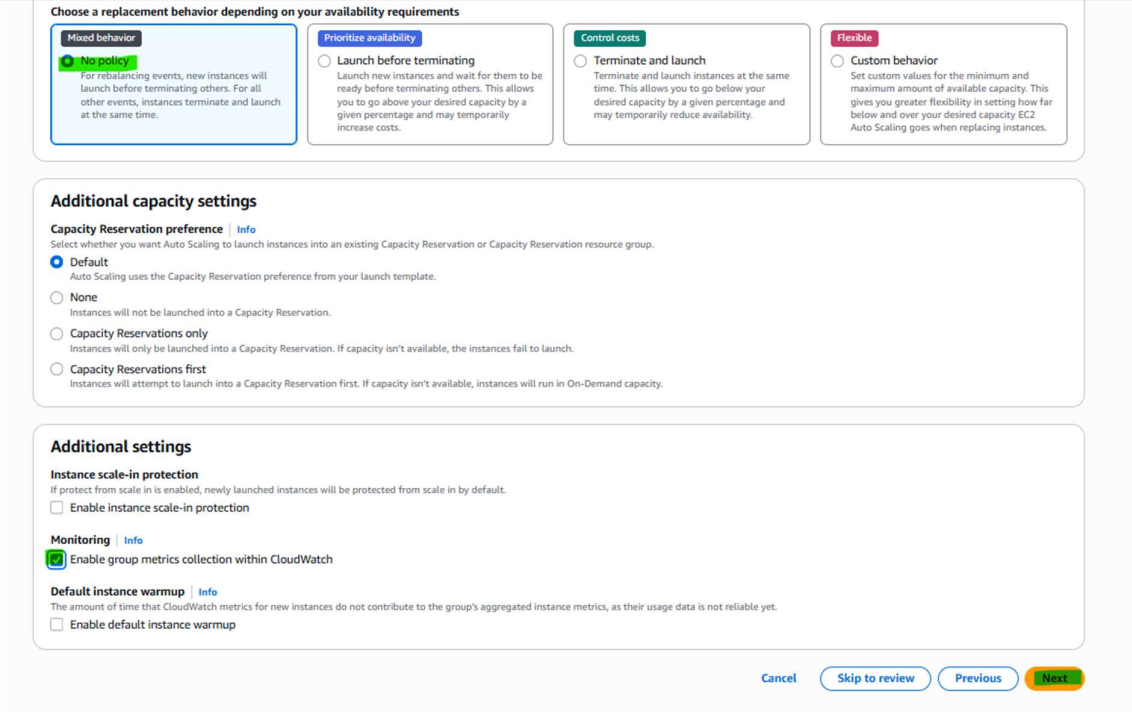
Click next.



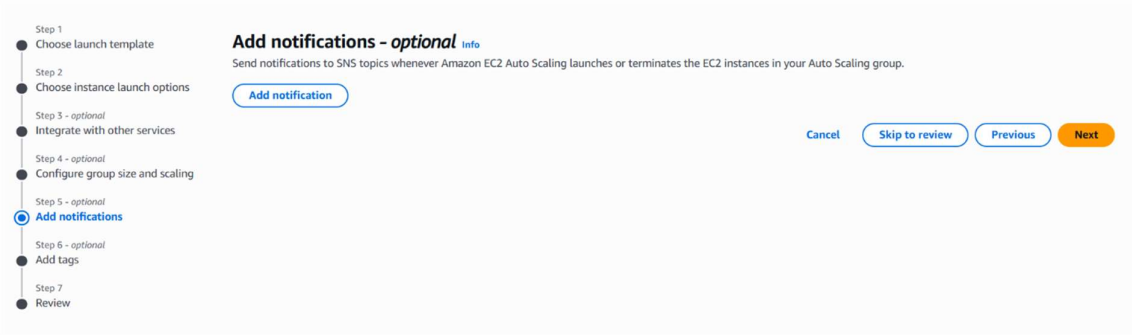
In configure group size and scaling, I have selected desired capacity as 2, Minimum capacity as 2, Maximum capacity as 3, In automatic scaling selected Target tracking scaling policy, Matric type as Average CPU utilization, Target value as 50, Instances need as 50.

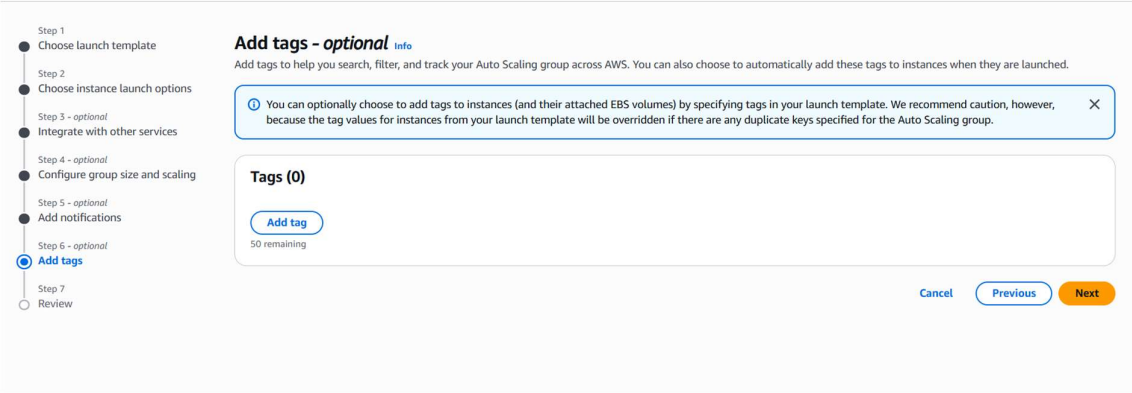


Selected no policy, and checked enable group metrics collection within cloudwatch. Click next.

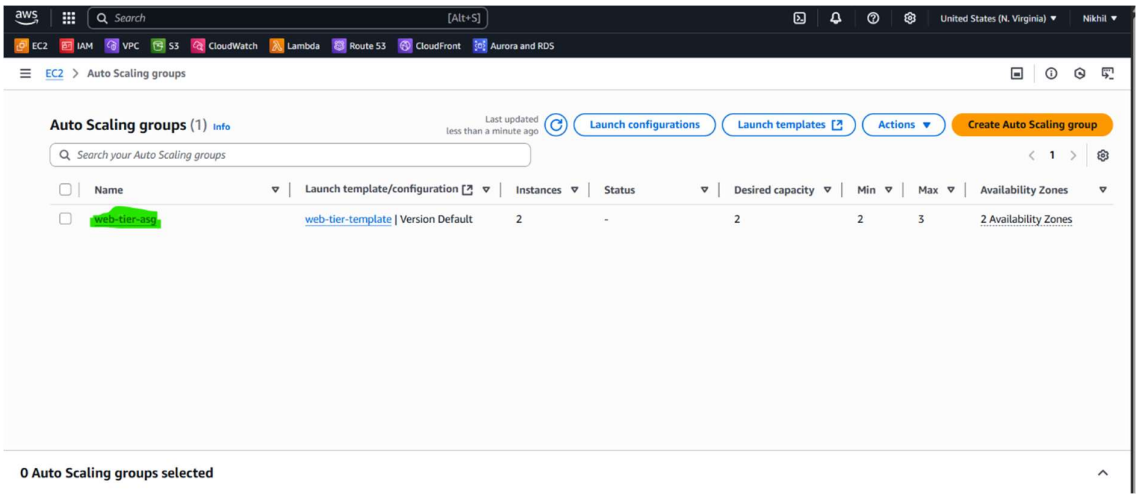


Click next, On next screen I have not added notifications and also tags, but I have skipped these for now. Click next.

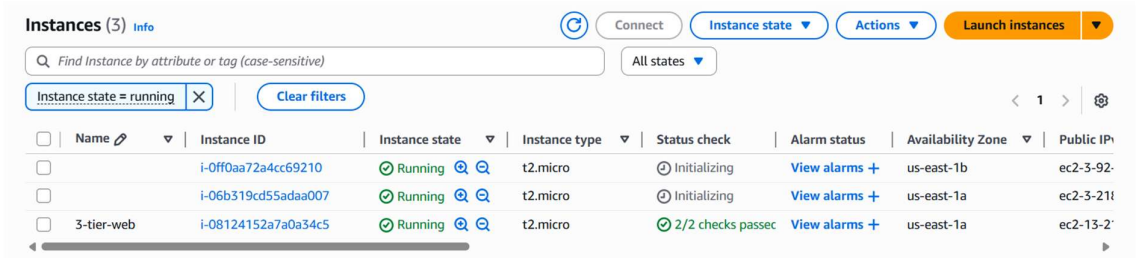




ASG is created.



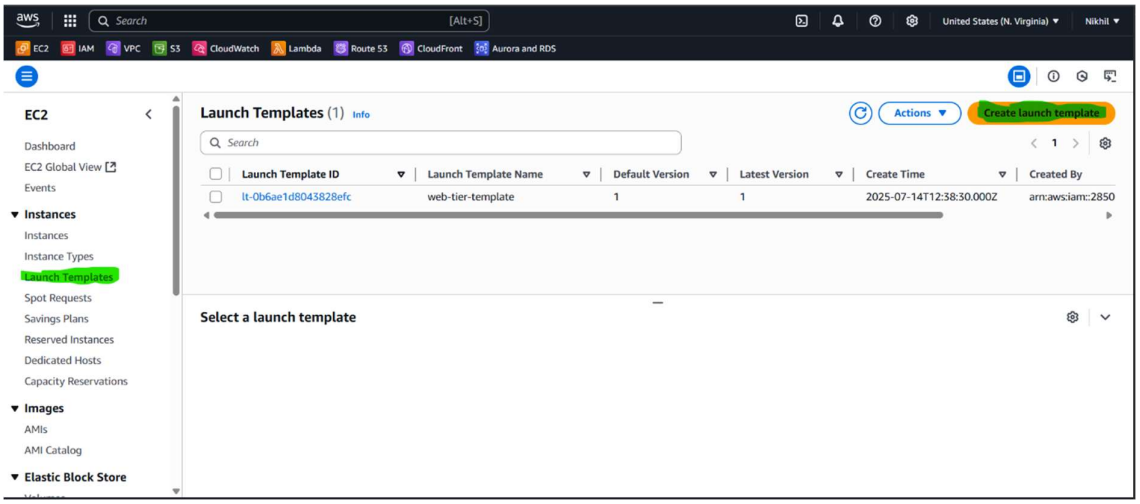
As I can see ASG is doing its job.



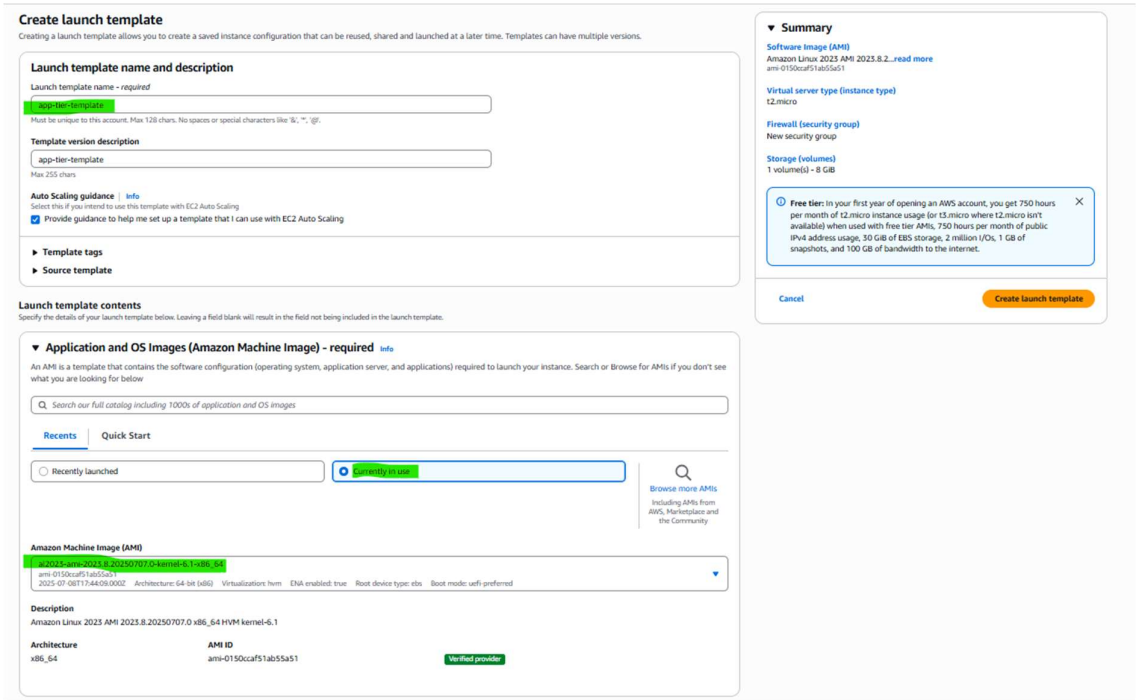
I have copied and pasted public IP of newly created Instance in web server, I can see the webpage, Its working.

Task 3: Creating Application Tier.

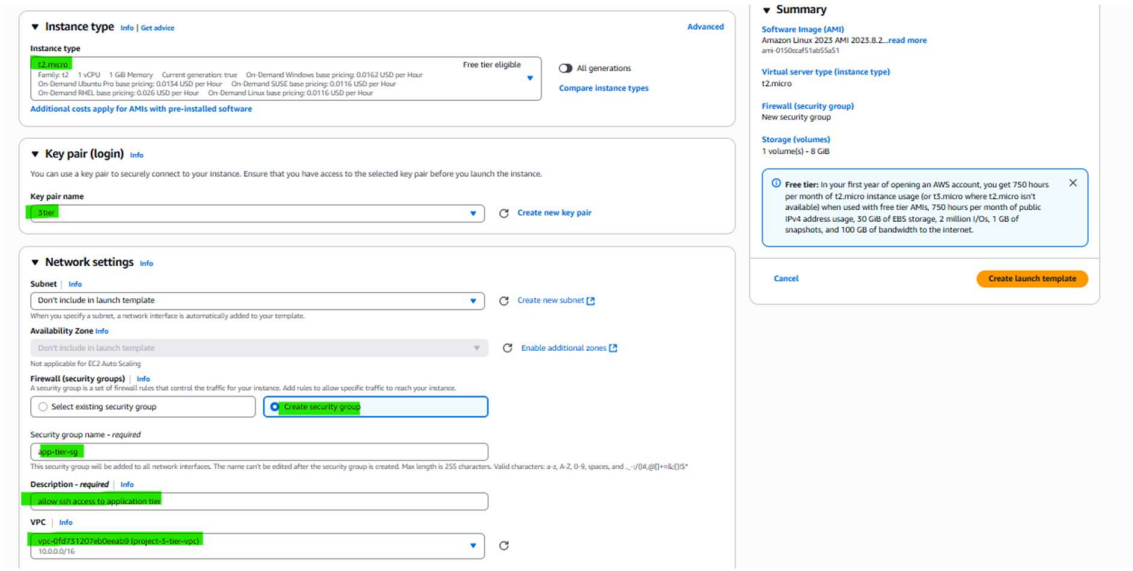
Go to EC2 dashboard, select launch template and click create launch template.



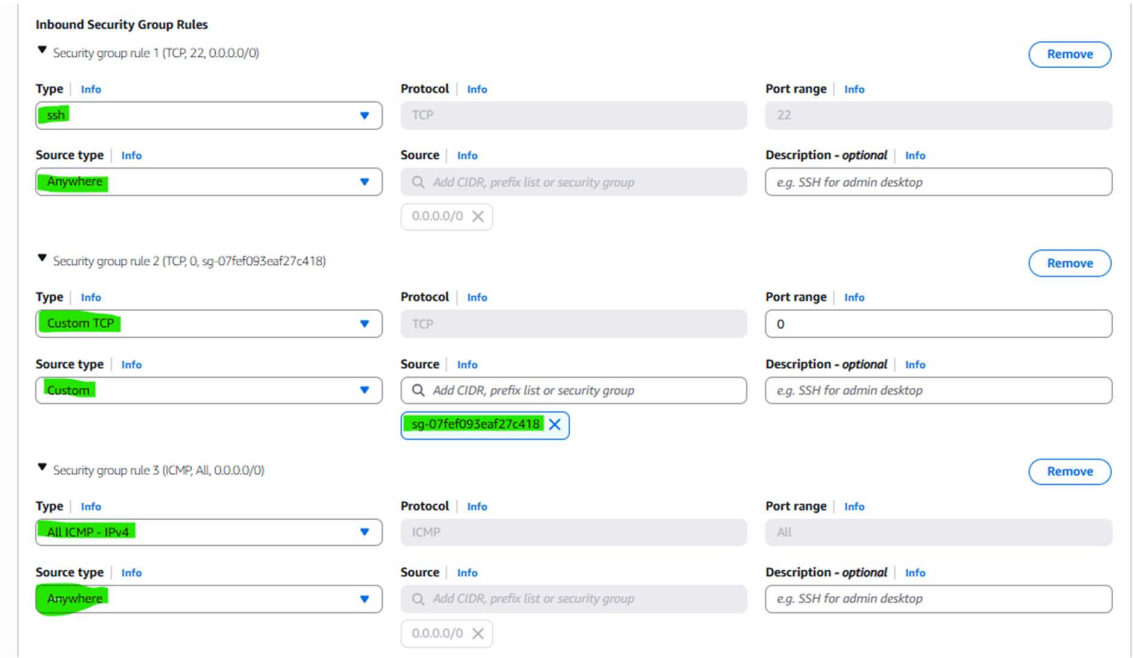
Name the template, enable guideance, Select currently in use AMI.



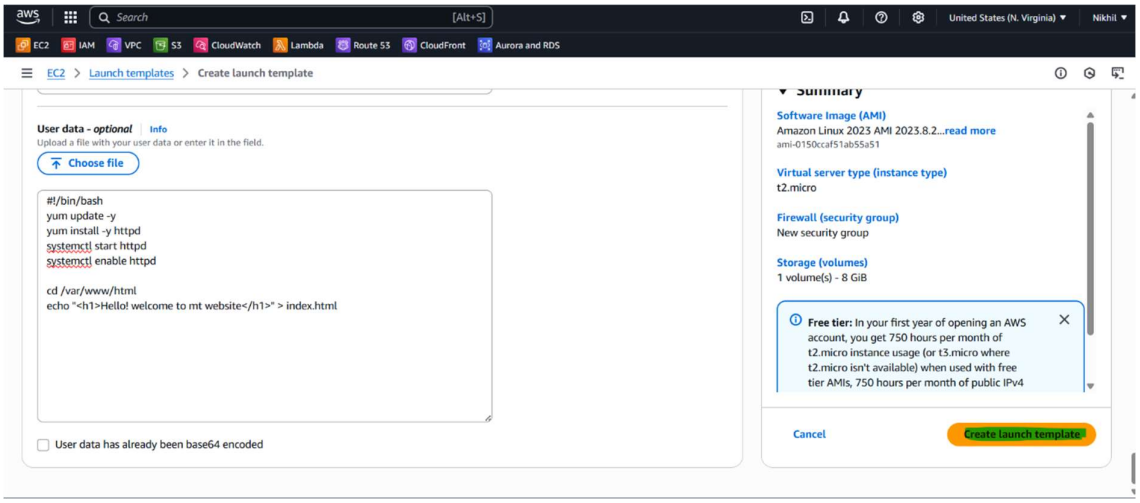
Selected t2.micro instance type, add key pair, create security group, name the sg, Select the VPC.



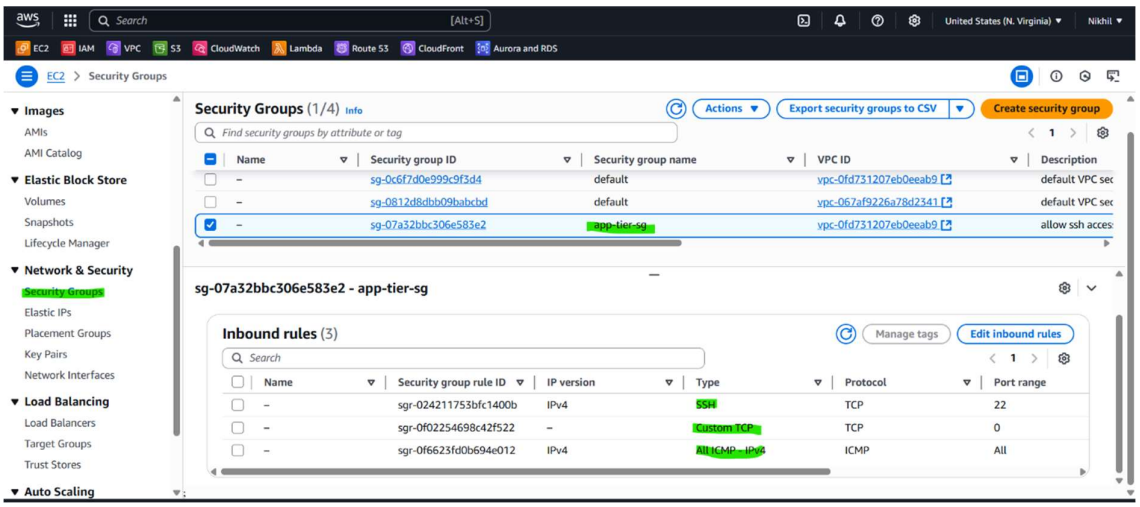
For inbound SG rules, first I added ssh from anywhere, second I added Custom TCP from custom id sg of web tire, Third I added ALL ICMP – Ipv4 from anywhere.



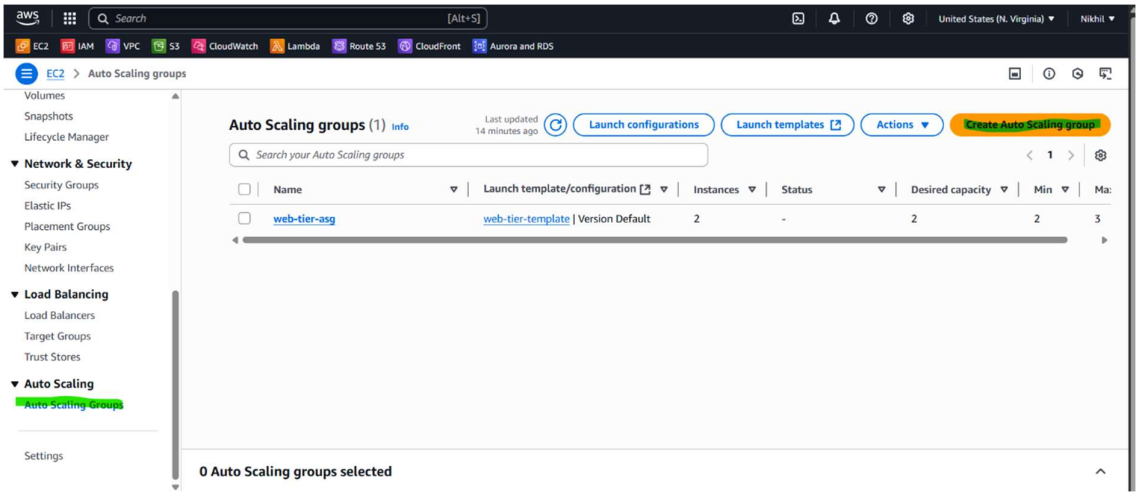
In advanced details, at bottom enter the same script as I did earlier for EC2 instance, then click create launch template.



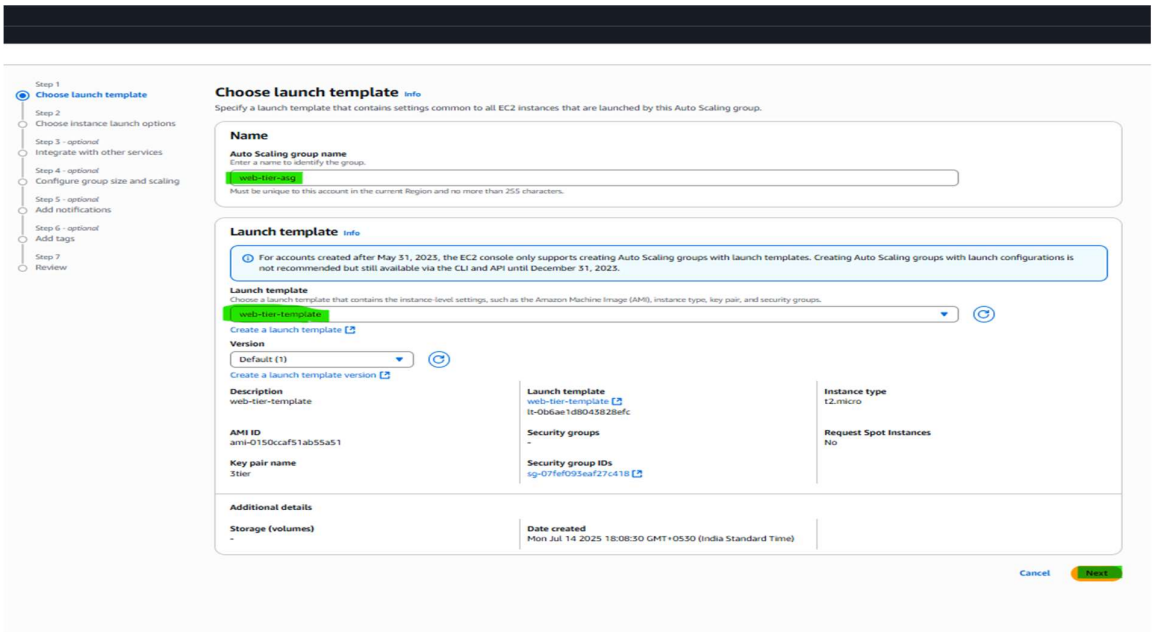
Goto security group and check the Inbound rules.



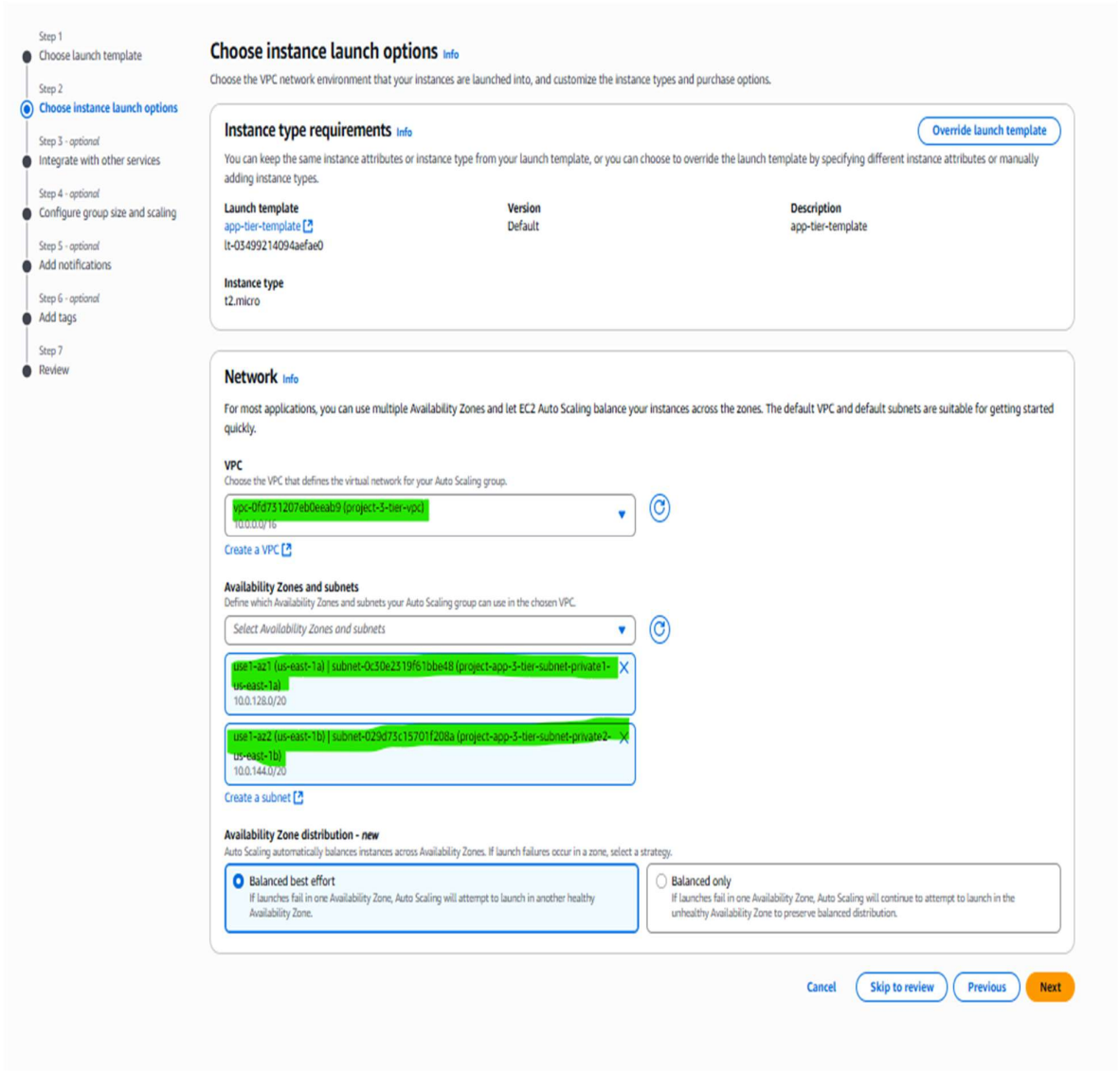
I have created ASG for app tier, Select Auto scaling groups in EC2 dashboard and click Create Auto scaling group.



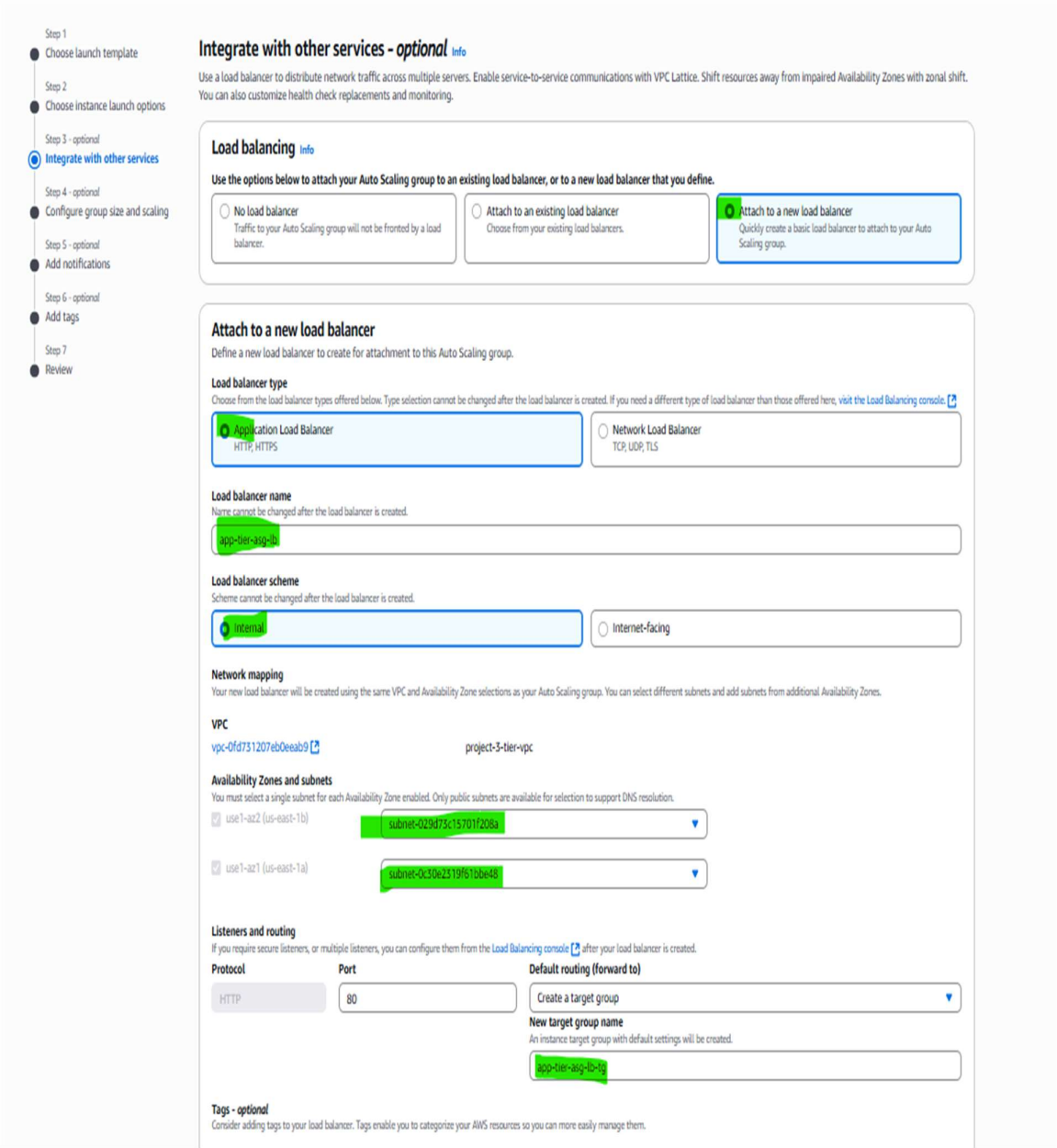
Name the ASG, Select the template that is created just now for app tier. Click Next.



In Network setting choose the VPC and subnets, select balanced best effort, Click next.

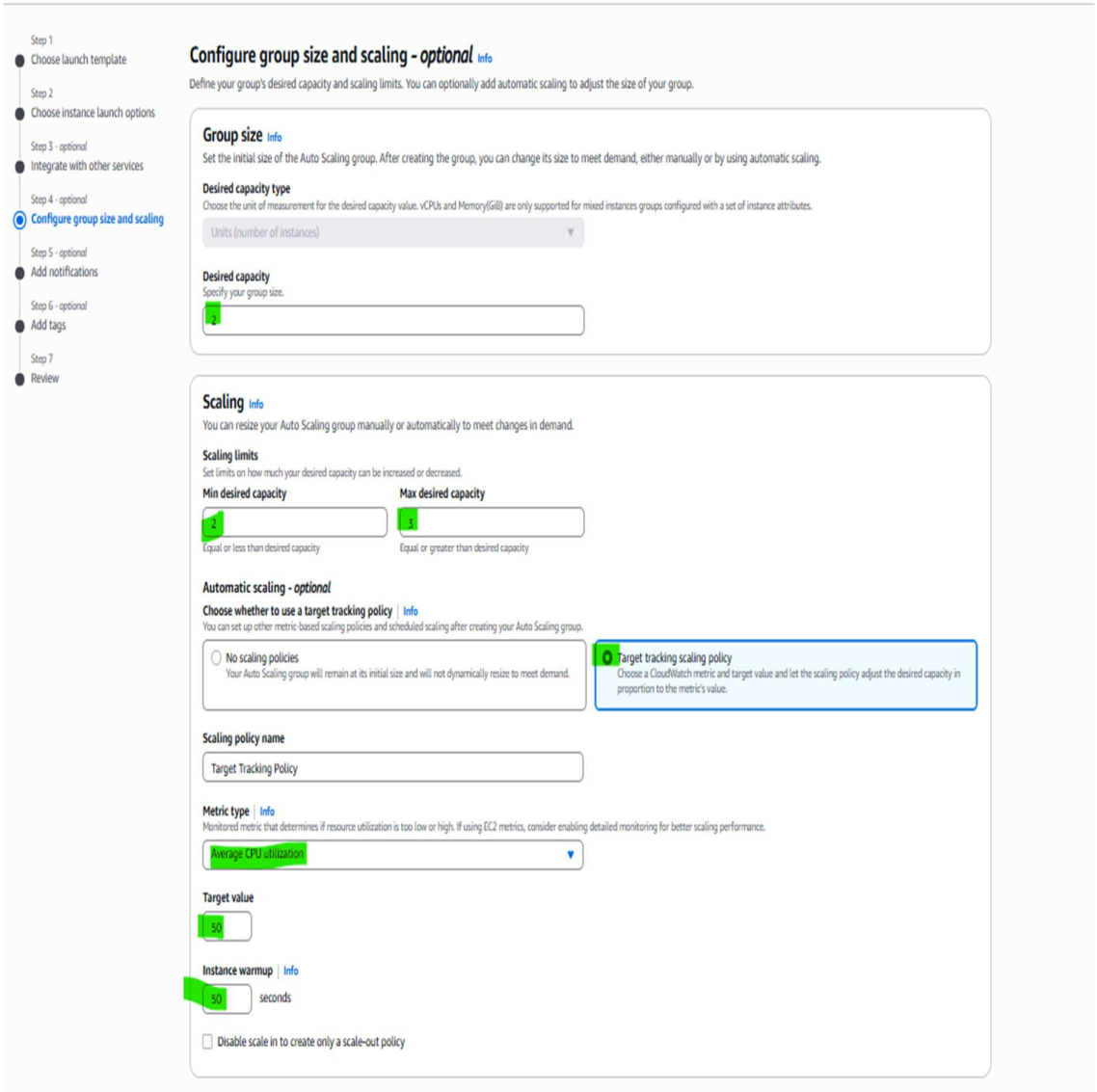


Select attach to a new load balancer, select application load balancer, name the load balancer, then select Internal, AZs and subnets are already selected. In listeners and routing select create a target group which should be on port 80 for HTTP traffic.

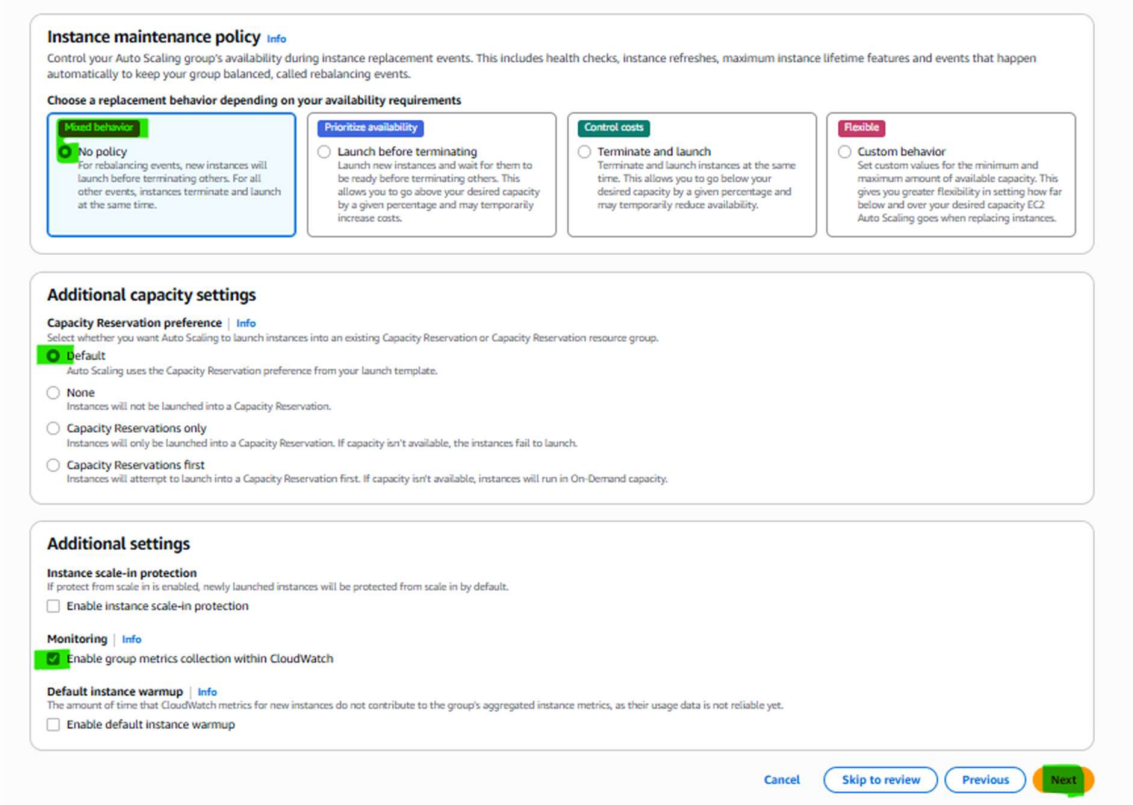


Select No VPC Lattice service, Turn on elastic load balancing health checks.

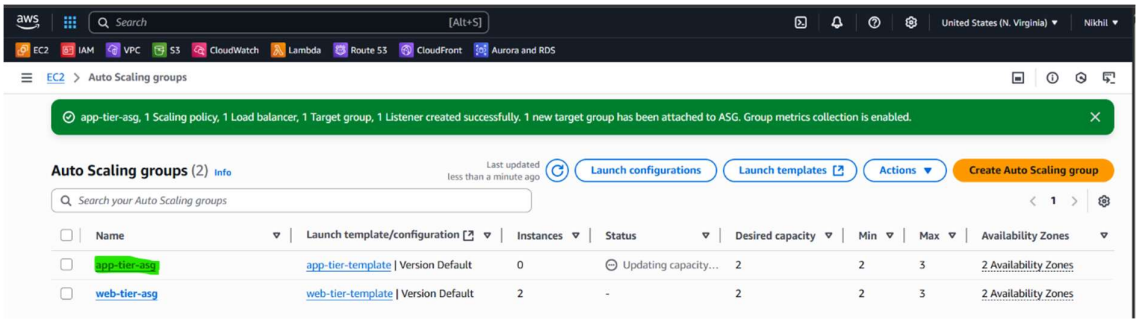
In configure group size and scaling, I have selected desired capacity as 2, Minimum capacity as 2, Maximum capacity as 3, In automatic scaling selected Target tracking scaling policy, Matric type as Average CPU utilization, Target value as 50, Instances need as 50.



Selected no policy, and checked enable group metrics collection within cloudwatch. Click next.



Click next, On next screen I have not added notifications and also tags, but I have skipped these for now. Click next.



ASG for app tier is created and its working.

I tried ssh to private ec2, showing timeout, It’s a good sign.

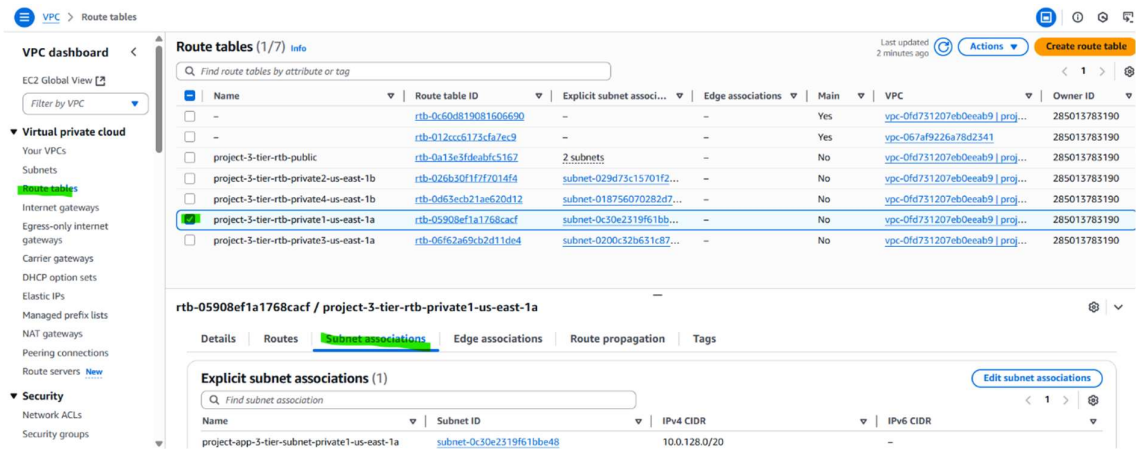
```
C:\Users\nikhi>cd downloads

C:\Users\nikhi\Downloads>ssh -i "3tier.pem" ec2-user@ec2-54-166-113-178.compute-1.amazonaws.com
ssh: connect to host ec2-54-166-113-178.compute-1.amazonaws.com port 22: Connection timed out
```

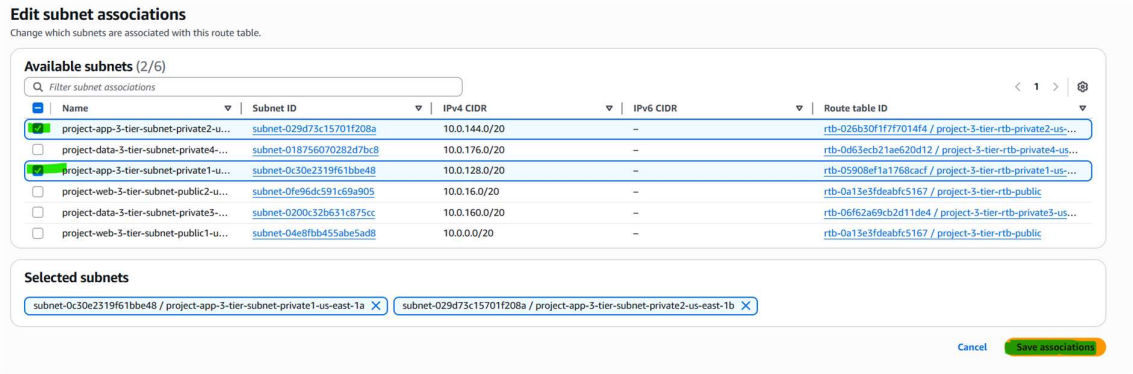
I tried ping private EC2 from public EC2. Its working.

```
C:\Users\nikhi\Downloads>ssh -i "3tier.pem" ec2-user@ec2-13-219-223-21.compute-1.amazonaws.com
The authenticity of host 'ec2-13-219-223-21.compute-1.amazonaws.com (13.219.223.21)' can't be established.
ED25519 key fingerprint is SHA256:thJKjmx9aYQnYi36HX16Pe1c/RLDhBA6d53qTehMsXc.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-13-219-223-21.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-10-0-12-158 ~]$ ping 10.0.148.177
PING 10.0.148.177 (10.0.148.177) 56(84) bytes of data.
64 bytes from 10.0.148.177: icmp_seq=1 ttl=127 time=1.94 ms
64 bytes from 10.0.148.177: icmp_seq=2 ttl=127 time=0.750 ms
64 bytes from 10.0.148.177: icmp_seq=3 ttl=127 time=0.912 ms
64 bytes from 10.0.148.177: icmp_seq=4 ttl=127 time=1.20 ms
64 bytes from 10.0.148.177: icmp_seq=5 ttl=127 time=0.869 ms
64 bytes from 10.0.148.177: icmp_seq=6 ttl=127 time=0.718 ms
```

Head back to the VPC dashboard, select Route tables, and select one of the route tables that was automatically created when we created our VPC. I only have 1 subnet associated with this table so click on Edit subnet associations.

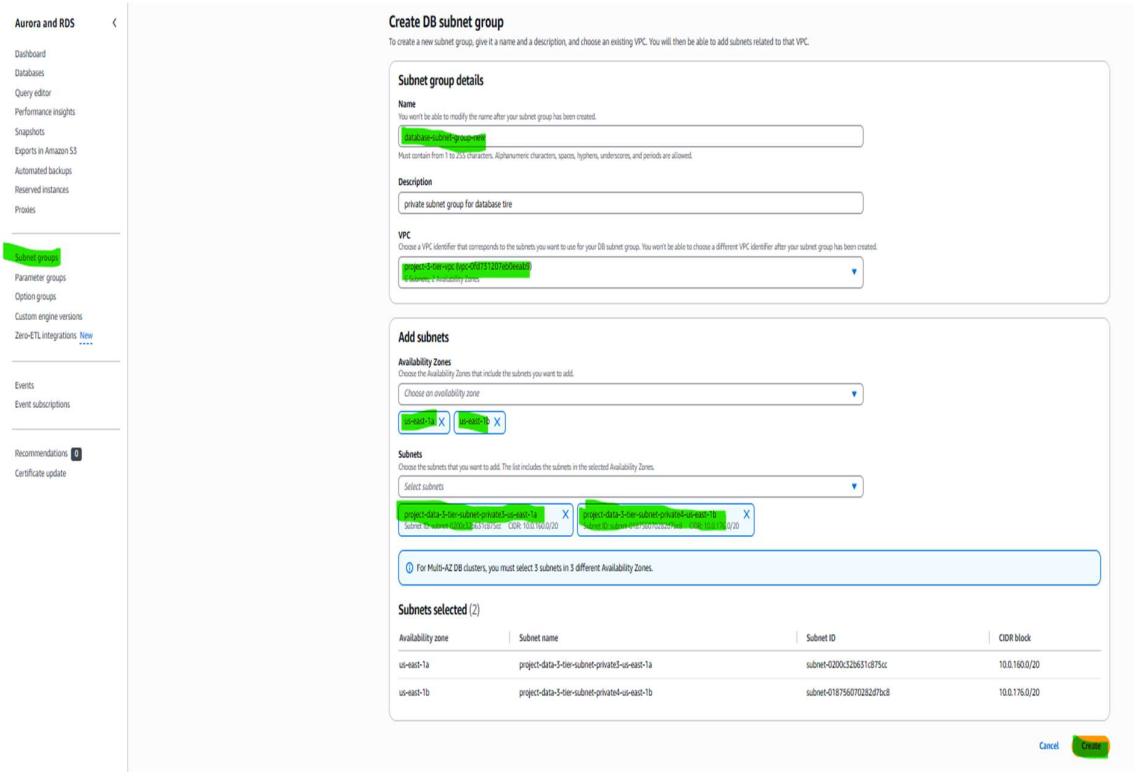


Add another subnet that is private.

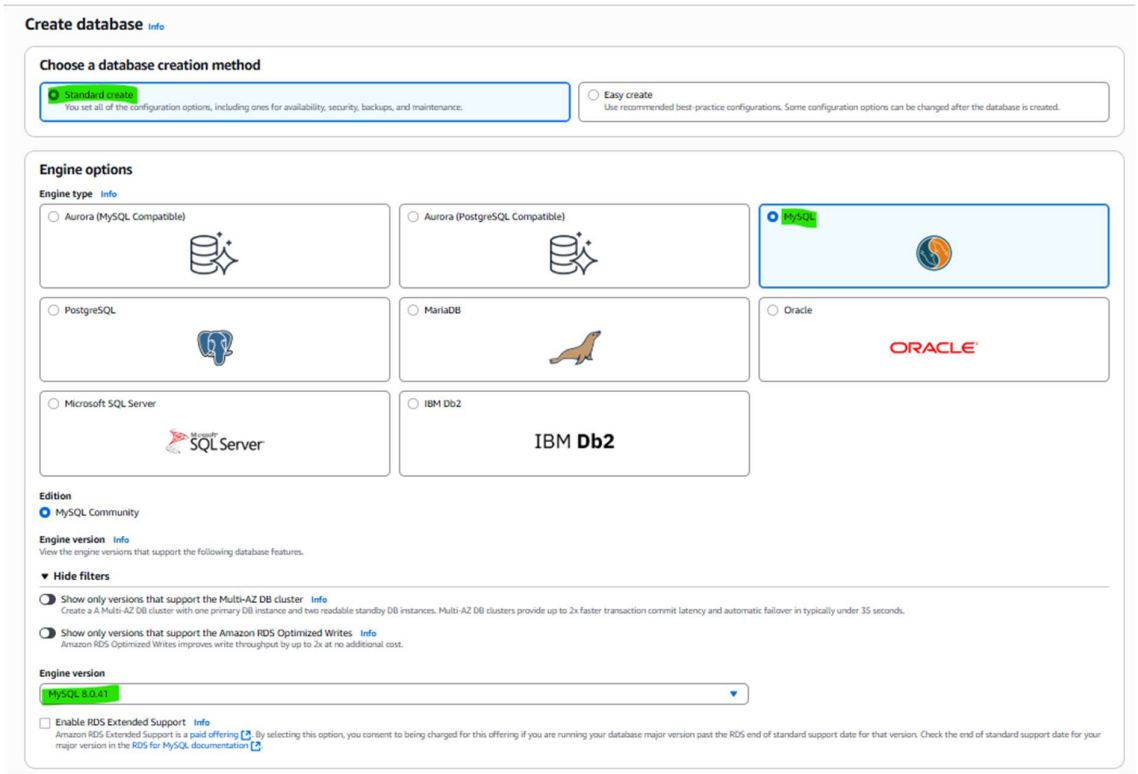


Task 4: Creating Database Tier

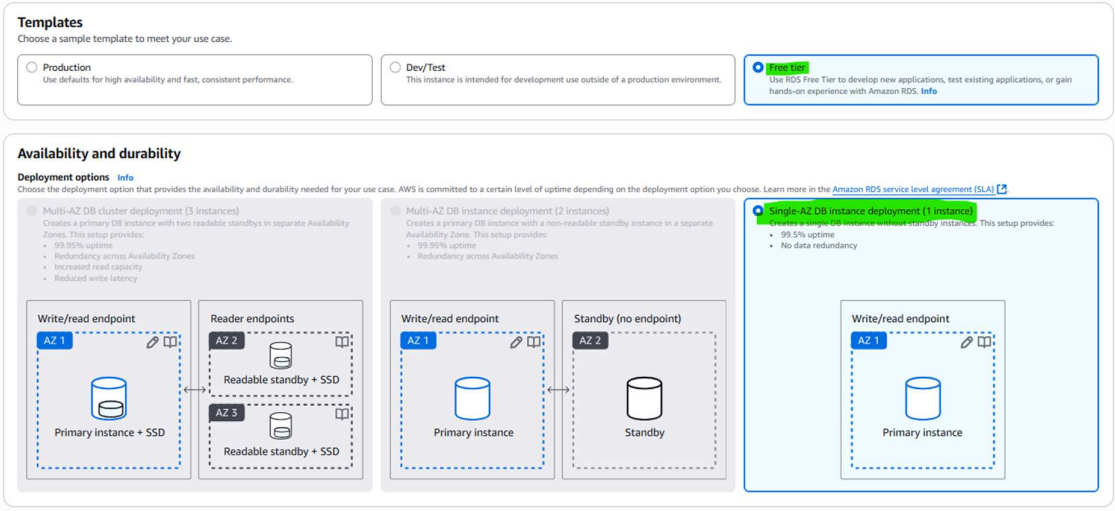
Go to RDS console and select subnet groups and click create subnet group. Name the subnet group, select VPC, add subnet in which database private ones AZs and subnet. Click Next.



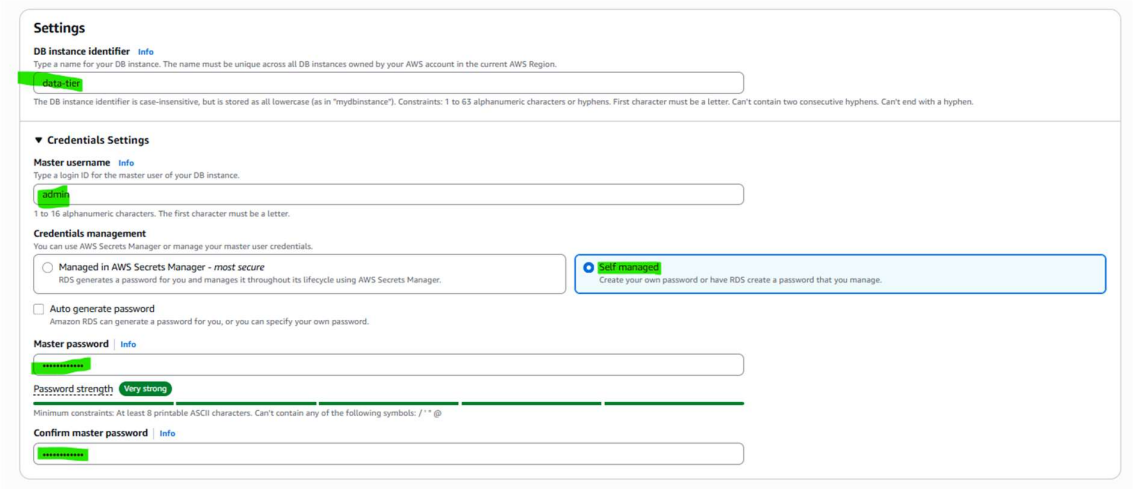
Back in the RDS console click Create database. Select the MySQL DB.



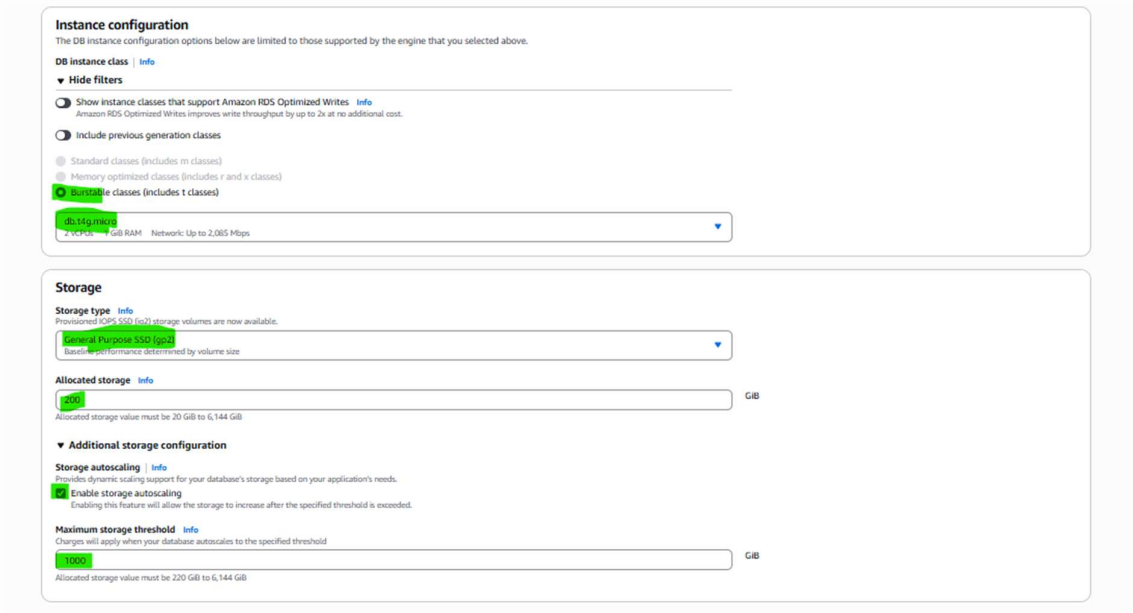
There are also availability and durability options; however, with the free-tier, none are available. We do not need them either.



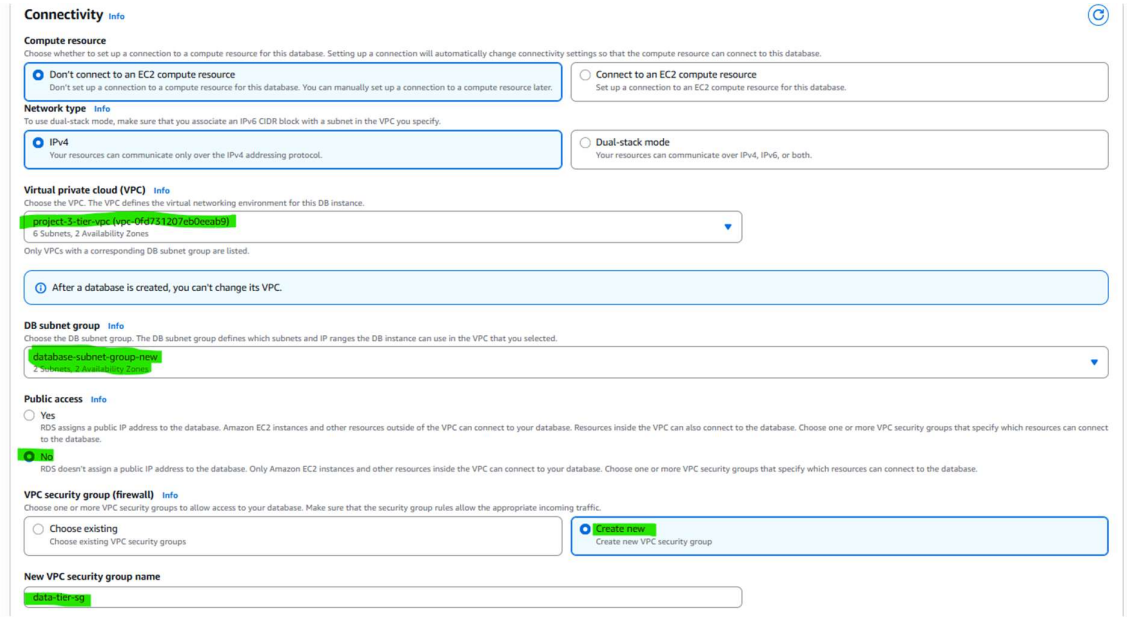
Under Settings, name your DB and create a master username and password. This username and password these should be different than your AWS account user login, as it is specific to the database you are creating.

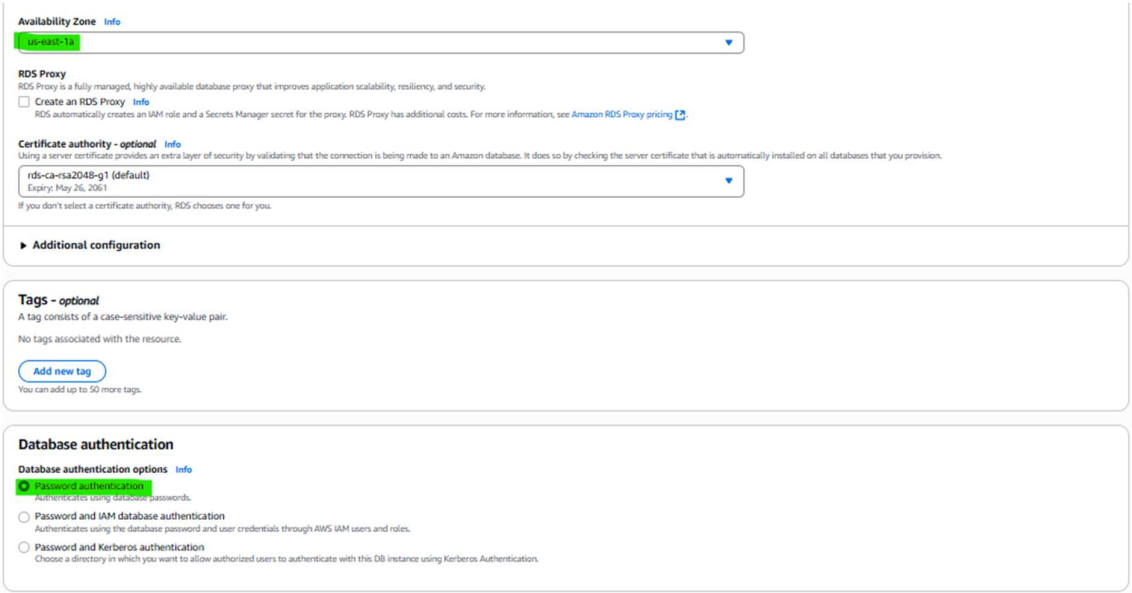


Under Instance configuration, the burstable classes option is pre-selected because it's the only one available for the free tier. I left my instance type as a db.t2.micro. You can add storage as needed; I left mine on default settings.



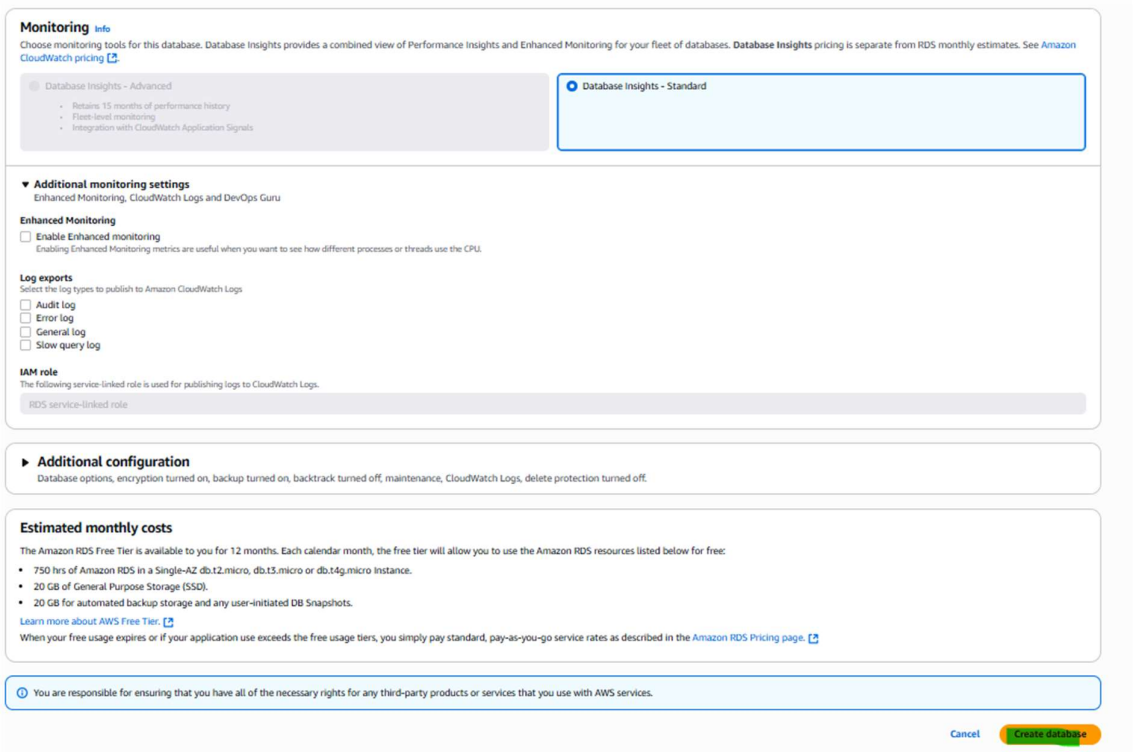
I am going to set up our network manually so choose not to connect to an EC2 resource. Select the proper VPC; the subnet group that you created earlier should be listed as default. Select Create new VPC security group (firewall).





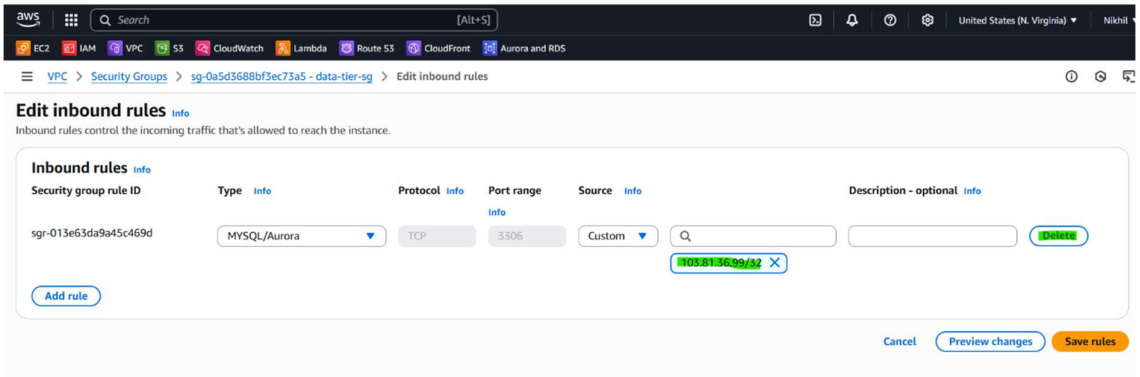
In Database authentication I left the default checked.

Click the “Create database” button.

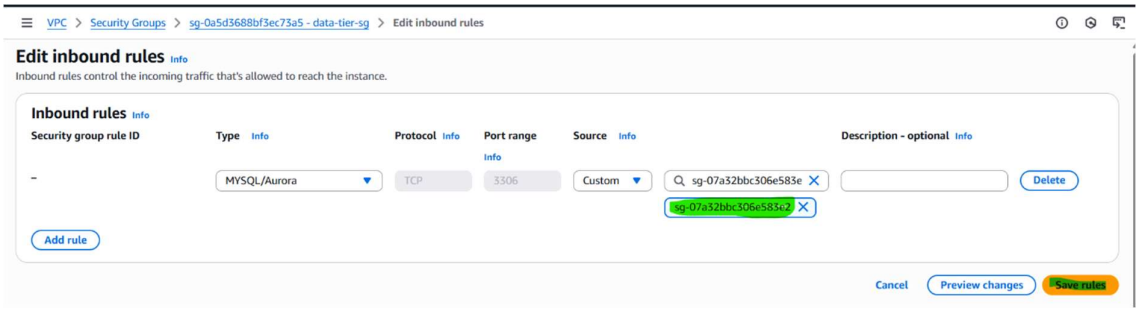


Go to the VPC console, select Security groups on the left side menu, and then find the database tier security group you just created.

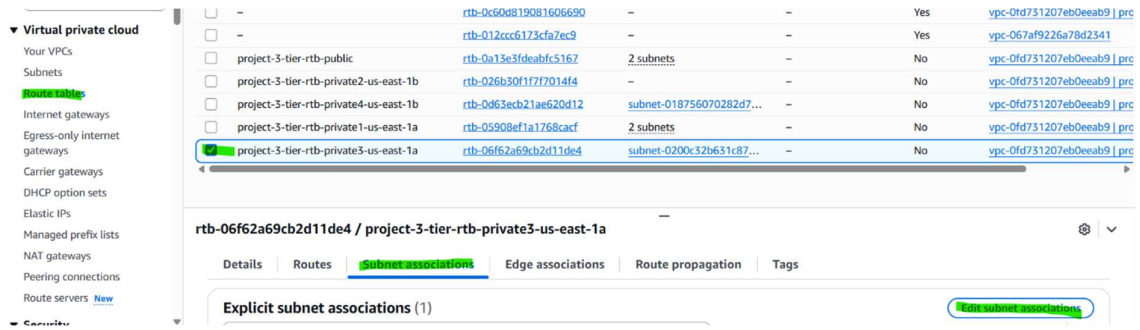
Select the security group that you just created. You need to edit inbound rules; by default the database SG has an inbound rule to allow MySQL/Aurora traffic on port 3306 from your IP address. Delete this rule.

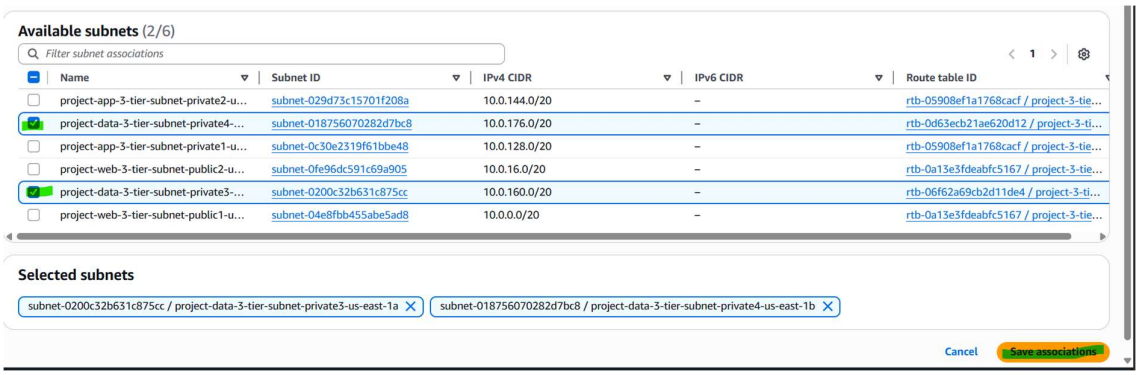


Create a new rule for MYSQL/Aurora on port 3306: for the Source, select Custom to add your security group for your application layer (tier-2 SG).



In the last step for our database tier, we need to make sure that the route table we associate with our databases private subnets has both subnets listed in subnet associations. If not, add the other subnet, and save.





Our three-tier architecture is done! We have already tested our web and application layers, but we are going to go a step further here.

Task5: Testing

We can't directly SSH to the database, but we can use an SSH forwarding agent to achieve this.

You need to add your access key pair file to your keychain. To do this, first make sure you are in your local host (use the command exit to get out of any EC2 instance you're connected to). Then use the following command:

```
PS C:\WINDOWS\system32> cd C:\Users\nikhi\Downloads
PS C:\Users\nikhi\Downloads> Start-Service ssh-agent
PS C:\Users\nikhi\Downloads> Get-Service ssh-agent | Set-Service -StartupType Automatic
PS C:\Users\nikhi\Downloads> ssh-add C:\path\to\your\3tier.pem
C:\path\to\your\3tier.pem: No such file or directory
PS C:\Users\nikhi\Downloads> ssh-add "C:\Users\nikhi\Downloads\3tier.pem"
Identity added: C:\Users\nikhi\Downloads\3tier.pem (C:\Users\nikhi\Downloads\3tier.pem)
PS C:\Users\nikhi\Downloads>
```

Now that your key pair file is added to your keychain, the SSH agent will scan through all of the keys associated with the keychain and find your matching key.

Now reconnect to the web tier EC2; however, this time use -A to specify you want to use the SSH agent.

```
PS C:\Users\nikhi\Downloads> ssh -A ec2-user@13.219.223.21
The authenticity of host '13.219.223.21 (13.219.223.21)' can't be established.
ED25519 key fingerprint is SHA256:thJKjmx9aYQnYi36HX16Pe1c/RLDhBA6d53qTehMsXc.
This host key is known by the following other names/addresses:
    C:\Users\nikhi/.ssh/known_hosts:88: ec2-13-219-223-21.compute-1.amazonaws.com
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '13.219.223.21' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Mon Jul 14 13:15:49 2025 from 103.81.36.99
[ec2-user@ip-10-0-12-158 ~]$
```

Once you are logged back into your tier-1 EC2, use the following command to check if the SSH agent forwarded the private key.

ssh-add -l

```
[ec2-user@ip-10-0-12-158 ~]$ ssh-add -l
2048 SHA256:ArCm+eWb1Olb7VwV1Ts286ciUKWkqEP0uR1wcdeQ1BE C:\Users\nikhi\Downloads\3tier.pem (RSA)
2048 SHA256:av3noIcqkYT1/a8uYEFlzjP76LjxiLrTa7lut9xHbdY .\3tire.pem (RSA)
[ec2-user@ip-10-0-12-158 ~]$
```

Our key pair has been forwarded to our public instance. Go copy your tier-2 application layer private IP address and copy it into the next command.

ssh -A ec2-user@<private-ip-address>

SSH'ed from your public tier 1 web instance into your private tier 2 application instance.

```
[ec2-user@ip-10-0-12-158 ~]$ ssh -A ec2-user@10.0.131.58
The authenticity of host '10.0.131.58 (10.0.131.58)' can't be established.
ED25519 key fingerprint is SHA256:8XZv1gI6R5Vt/LvW6b87hssfnIQg8G6wEBtUXJQ7kgA.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.131.58' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-10-0-131-58 ~]$
```

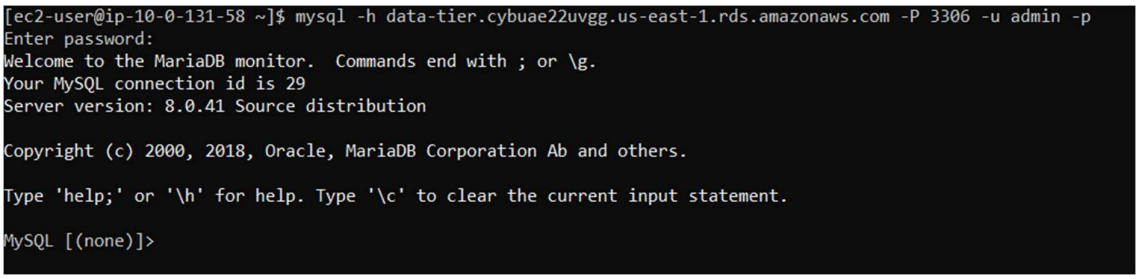
Installed database server (mariadb)

sudo dnf install mariadb105-server

```
[ec2-user@ip-10-0-131-58 ~]$ sudo dnf install mariadb105-server
Last metadata expiration check: 0:43:49 ago on Mon Jul 14 13:13:01 2025.
Dependencies resolved.
================================================================================
 Package                  Architecture  Version                  Repository     Size
================================================================================
Installing:
 mariadb105-server        x86_64        3:10.5.29-1.amzn2023.0.1 amazonlinux    10 M
Installing dependencies:
 mariadb-connector-c      x86_64        3.3.10-1.amzn2023.0.1    amazonlinux   211 k
 mariadb-connector-c-config noarch      3.3.10-1.amzn2023.0.1    amazonlinux   9.9 k
 mariadb105               x86_64        3:10.5.29-1.amzn2023.0.1 amazonlinux   1.5 M
 mariadb105-common        x86_64        3:10.5.29-1.amzn2023.0.1 amazonlinux    28 k
 mariadb105-errmsg        x86_64        3:10.5.29-1.amzn2023.0.1 amazonlinux   212 k
 mysql-selinux            noarch        1.0.4-2.amzn2023.0.3     amazonlinux    36 k
 perl-B                   x86_64        1.80-477.amzn2023.0.7    amazonlinux   177 k
 perl-DBD-MariaDB         x86_64        1.22-1.amzn2023.0.4      amazonlinux   153 k
```

Log into RDS MySQL database. Need RDS endpoint, user name, and password. To find RDS database endpoint, navigate to the database that created and find the endpoint under Connectivity & Security.

mysql -h <rds-database-endpoint> -P 3306 -u <username> -p



Successfully connected to MySQL database from the application tier. I have connectivity with all of our tiers.

By implementing a 3-tier architecture on AWS, organizations and developers can build applications that are not only robust and secure, but also easily scalable and maintainable over time. This architecture reflects best practices in cloud infrastructure design and serves as a strong foundation for deploying production-grade web applications. In this project, the 3-tier model is brought to life using various AWS services to demonstrate how cloud-native solutions can be used to meet modern application demands.