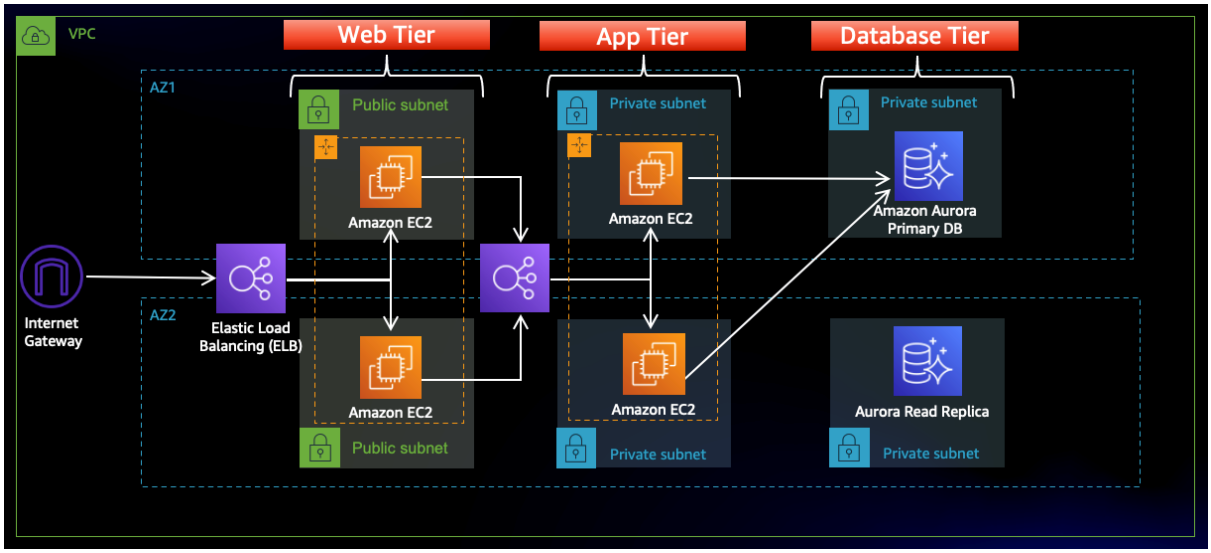
# DEPLOY A THREE-TIER ARCHITECTURE IN AWS

-BY

K.MANICHANDU

# AWS three-tier architecture :

This architecture shows the process of how a user will connect to the highly available and scalable app that is deployed on the internet in a three-tier model. This architecture clearly shows us the various features and services that are used from AWS service




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

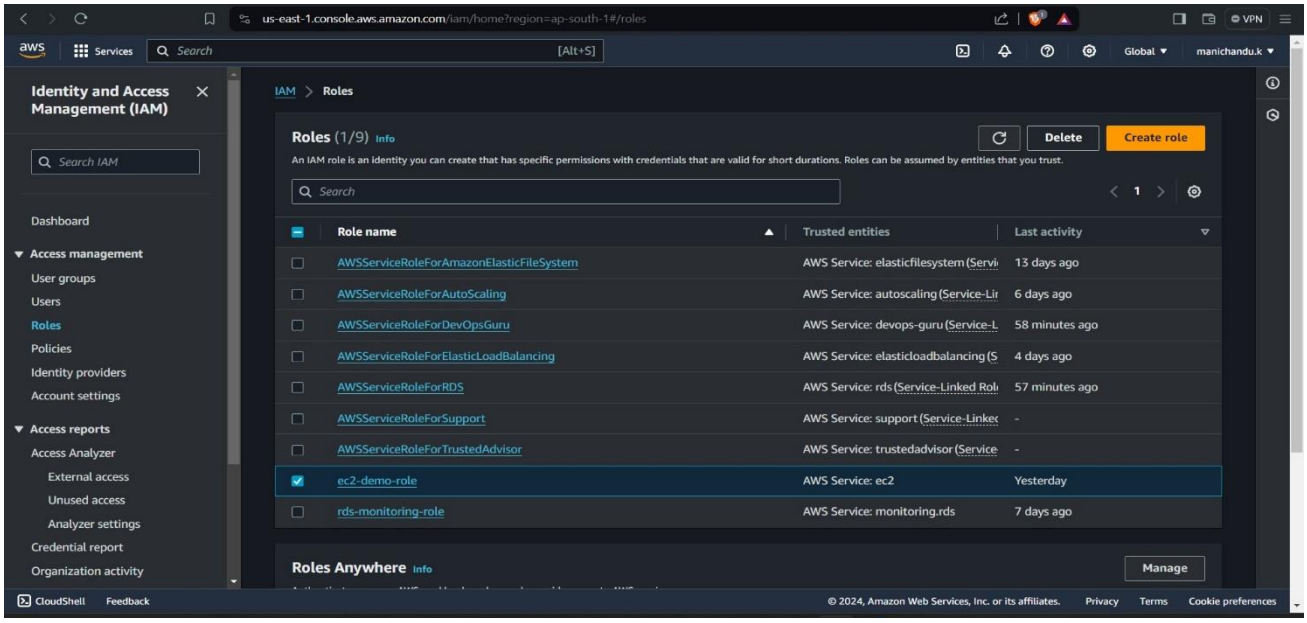
To deploy the above architecture one should know about AWS's basic features

1) VPC
2) EC2
3) IAM
4) Autoscaling
5) Load balancer

In this documentation, we will learn how to deploy a three-tier architecture from scratch.

## Set-Up :

- Sign in to your AWS console and go to IAM.
- Go to roles and click on Create a Role.
- Select the trusted entity as AWS service and select EC2.
- While adding permissions or policies add these two permissions to the role “AmazonSSMManagedInstanceCore” and “Amazon-S3ReadOnlyAcess.”
- Click on Create Role.

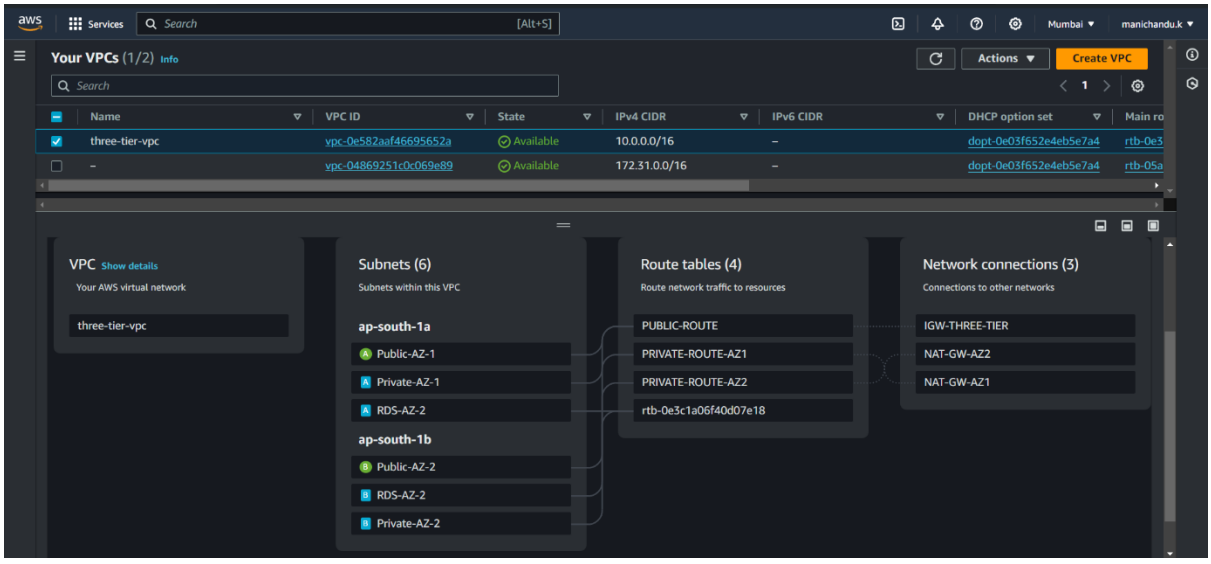



- If you have an application ready you can create an s3 bucket and upload your code into your bucket.

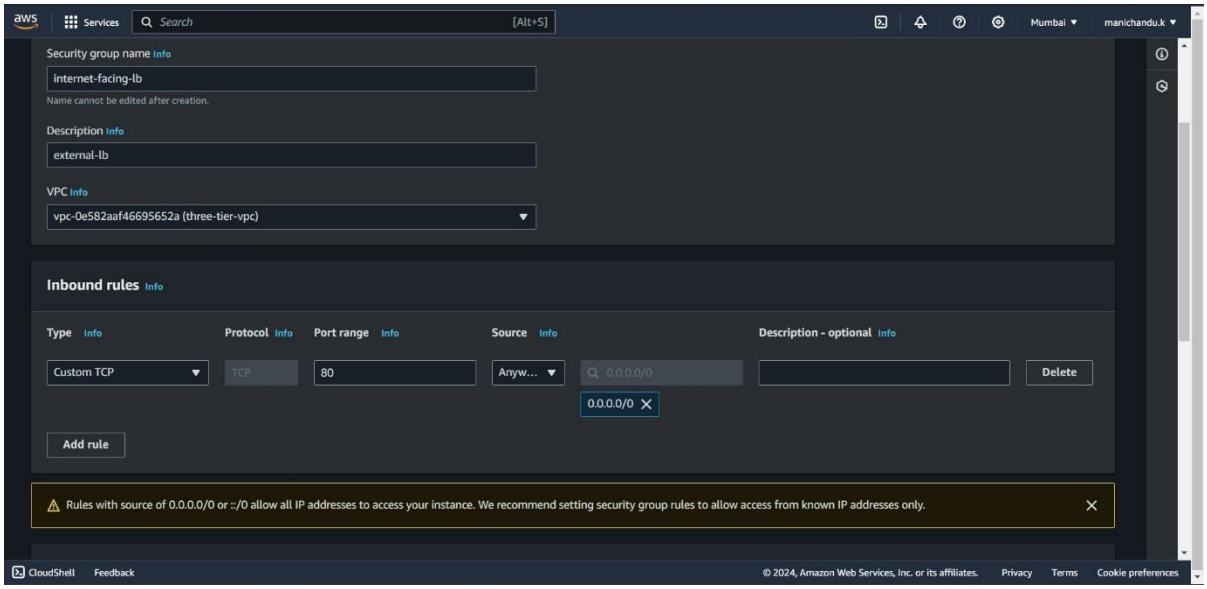
## Networking and security :

- In this section we will be building a VPC and its components and security groups which will add a protection layer to our EC2 instances, load balancers, and aurora databases.
- Go to the VPC section in your AWS console click on Create a VPC and select only VPC.
- Here I created a VPC with the CIDR block 10.0.0.0/16 with the name tag three-tier-vpc.

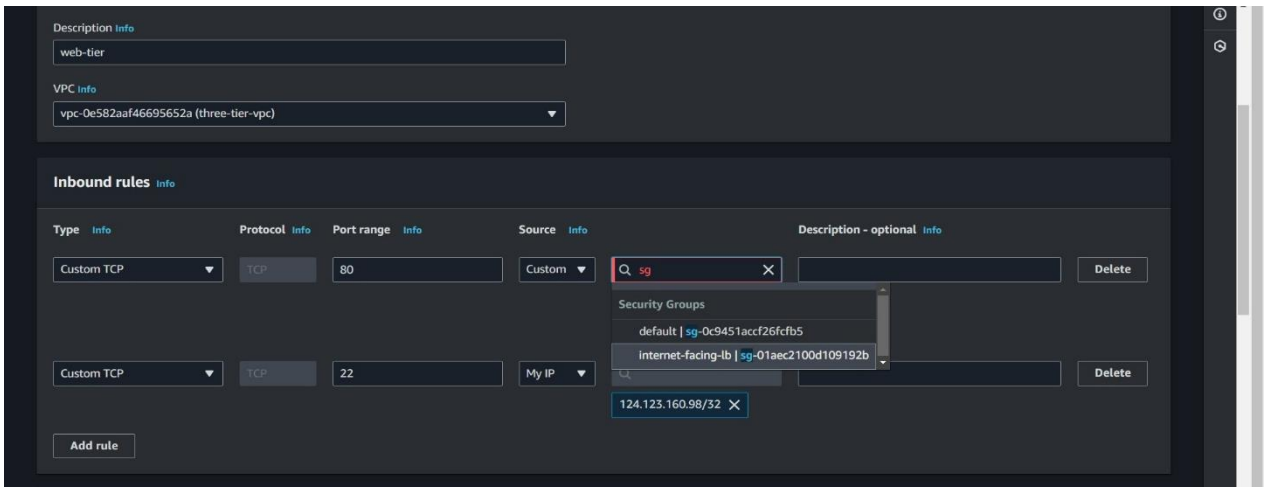
- You can create your VPC with your own CIDR block range and name tag.
- Now create subnets for this VPC. According to the architecture we need 2 public subnets for our web page deployment and 4 private subnets for our app and database deployment.
- Click on Create Subnet and remove the default VPC, select our custom-created VPC, and start creating subnets with different CIDR ranges.
- Now create an internet gateway and attach it to the VPC.
- Create NAT gateways in our VPC for accessing the internet from the private subnets.
- Go to the nat-gateway section, click on create, and select the subnets of the public subnet so that the nat-gateways will have internet access.
- Now go to the route-table section and create route tables for the subnets.
- Create a route table for the public subnets, edit the route table, and give internet access by giving an internet gateway route. And associate it with both public subnets.
- Create two separate route tables for private subnets and edit routes and give internet connection by giving nat-gateway respectively. And associate with their private subnets respectively.



- The final VPC should look like this.
- Now it's time to create the security groups for the architecture.
- Go to security groups in the VPC section and start creating route tables.
- First create a security group for our internet-facing elastic load balancer. It has the web-page code of our app and it has to get requests from the internet so we give port number 80.



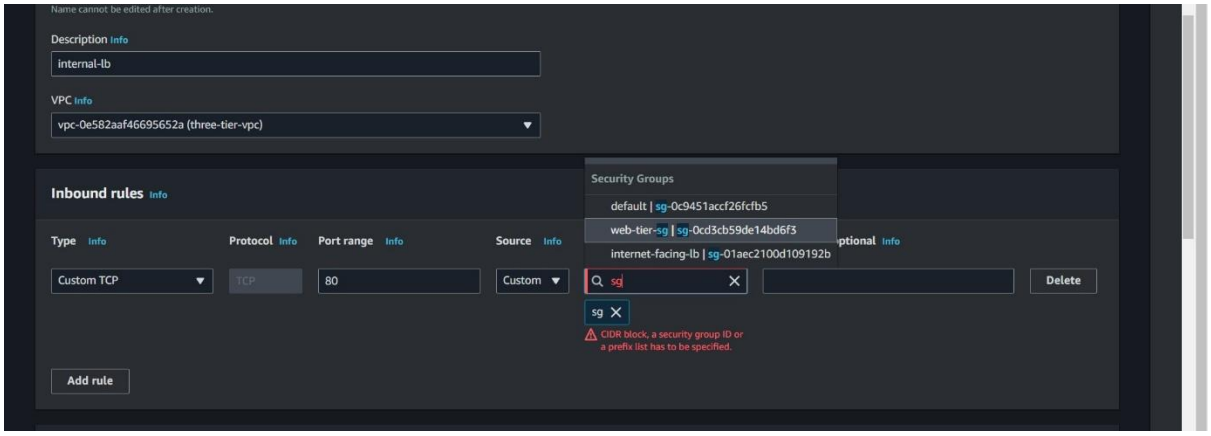

- Make sure to select the manually created VPC every time you create a security group.
- Now create a security group for our web-tier EC2 instances but



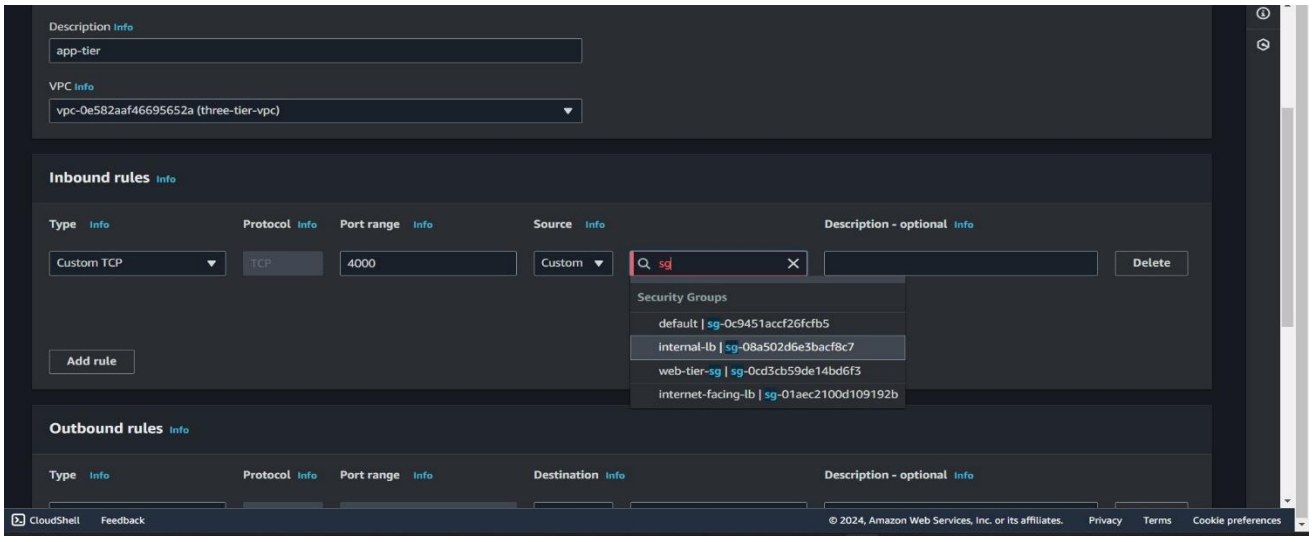

the incoming traffic should only be from the internet-load balancer. So when you create the security group for web-tier

instances give the source the security group id of the internet-facing-lb.

- Same with the internal load balancer for the app-tier instances the traffic is to be allowed only from the web-tier security group.

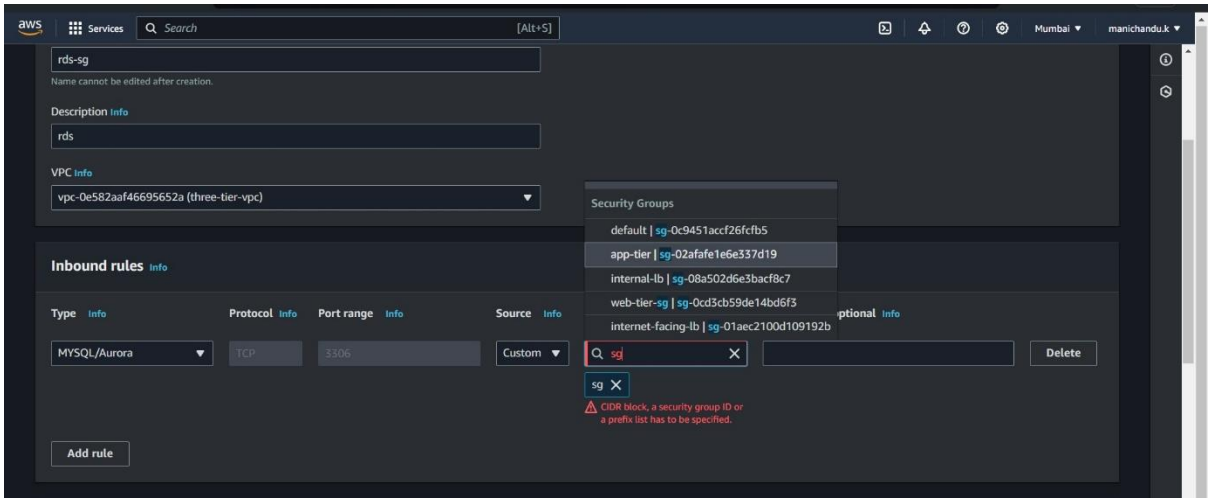



- Now create a security group for app-tier instances where our app is deployed here I gave port number 4000 as most of the apps are written in node.js and it is allowed through port 4000. But the app you deploy may be written in any language so it may be from 3000-4000. So choose the correct port number where your app works.

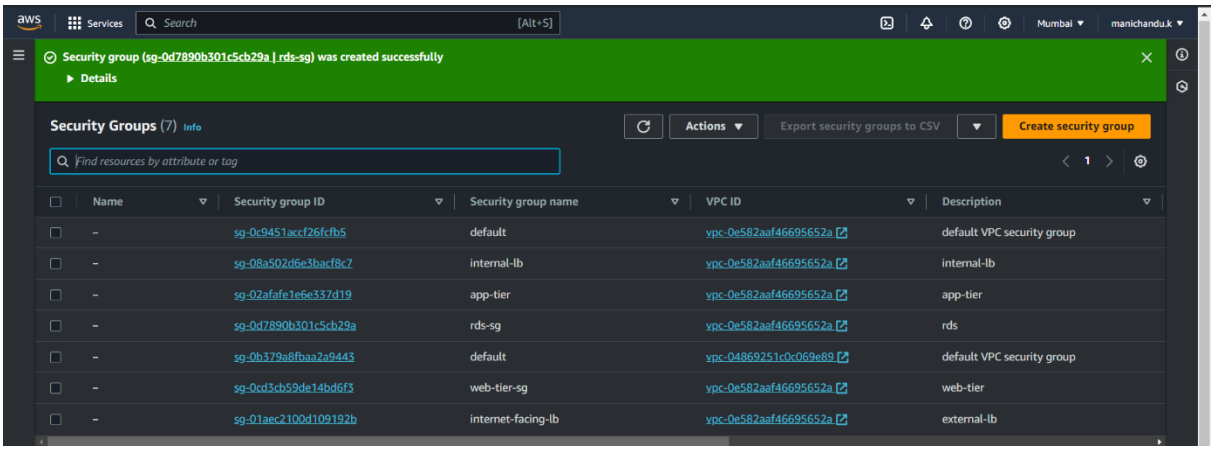



- Now lastly create a security group for our database in which I took aurora mysql which has a default port number of 3306.

- Remember to follow the same steps to give secured protection for your application and select the previous sg-id as your source for your current security group.




- Lastly you should have 5 security for your three-tier architecture and these are the security groups that are used in your app deployment.

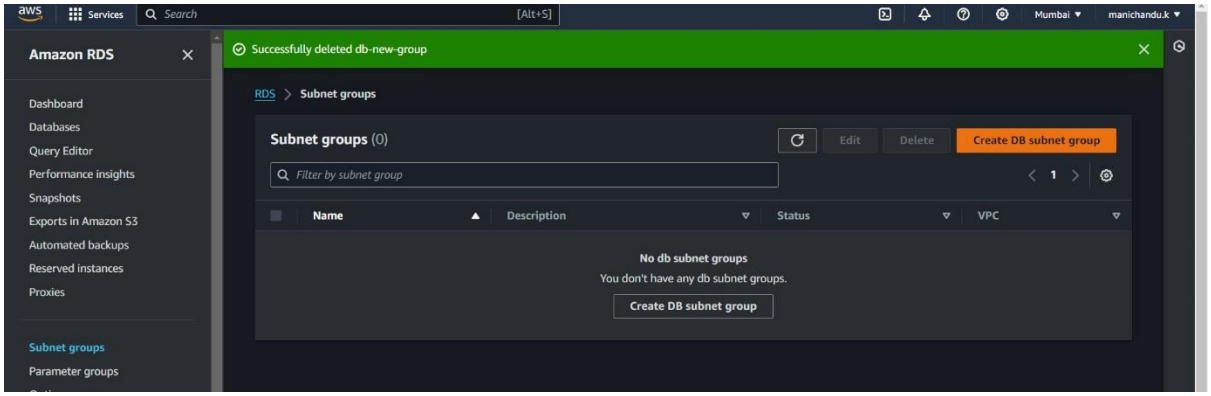


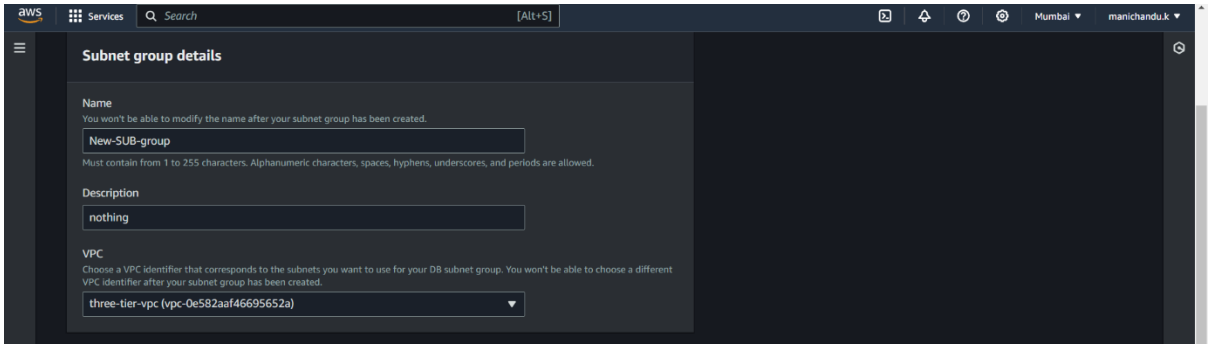

- The networking and security configuration is now completed and we will move on to the next step.

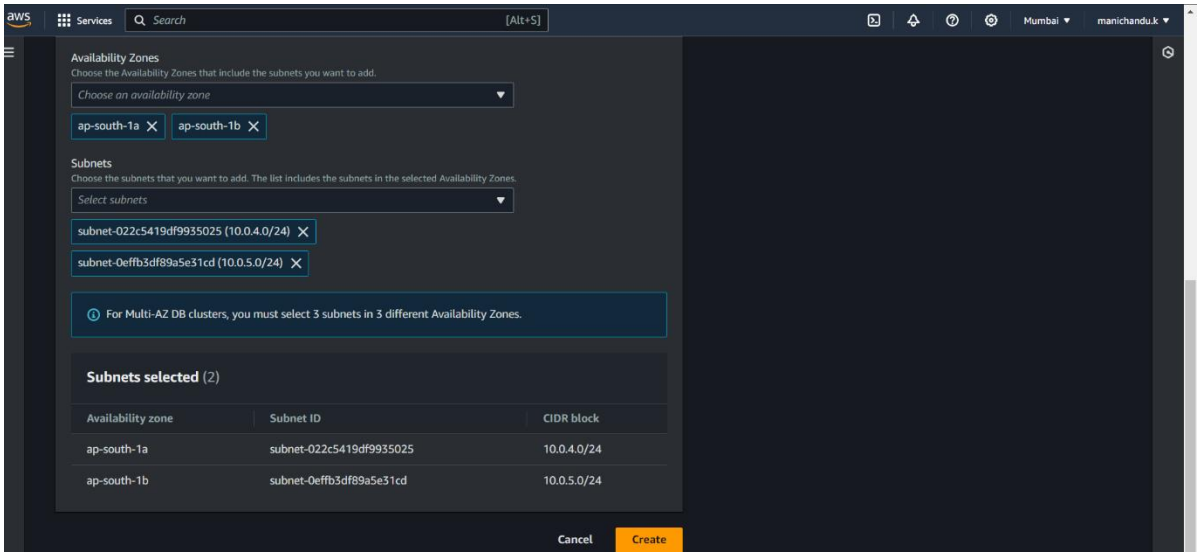
# Database Configuration :

- In this section we will configure database subnet groups and multi-AZ databases.
- Navigate to the RDS dashboard, select the subnet groups section, click on Create DB subnet groups, and select the VPC you will create the subnet group and select the available zone, in our case

they are zones 1a, 1b, and select the subnets we created in that zones exclusively for the databases.

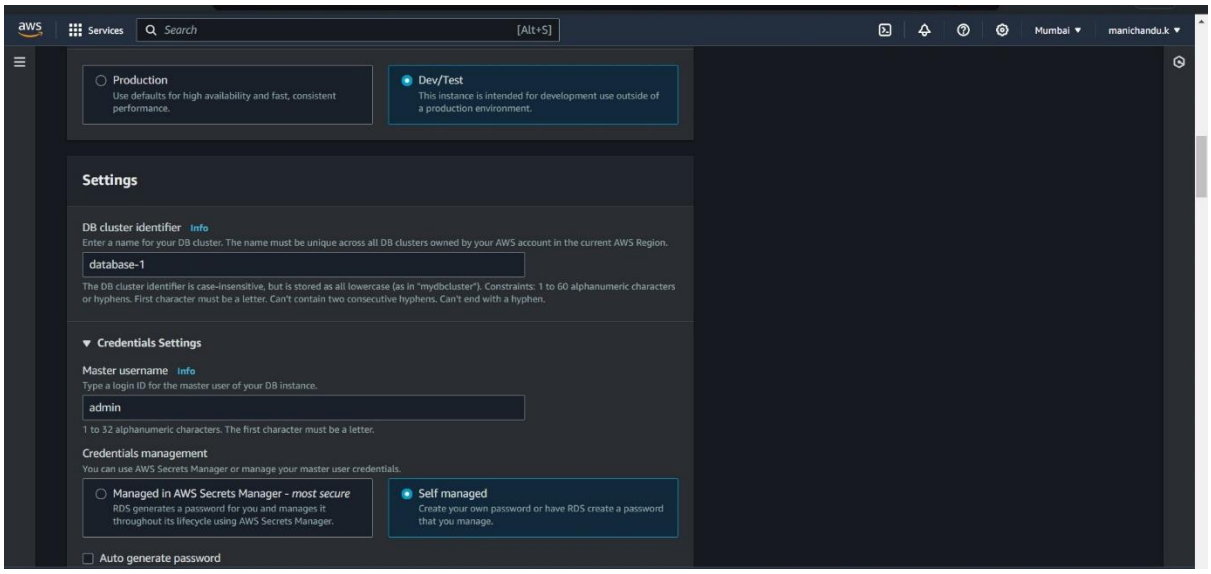


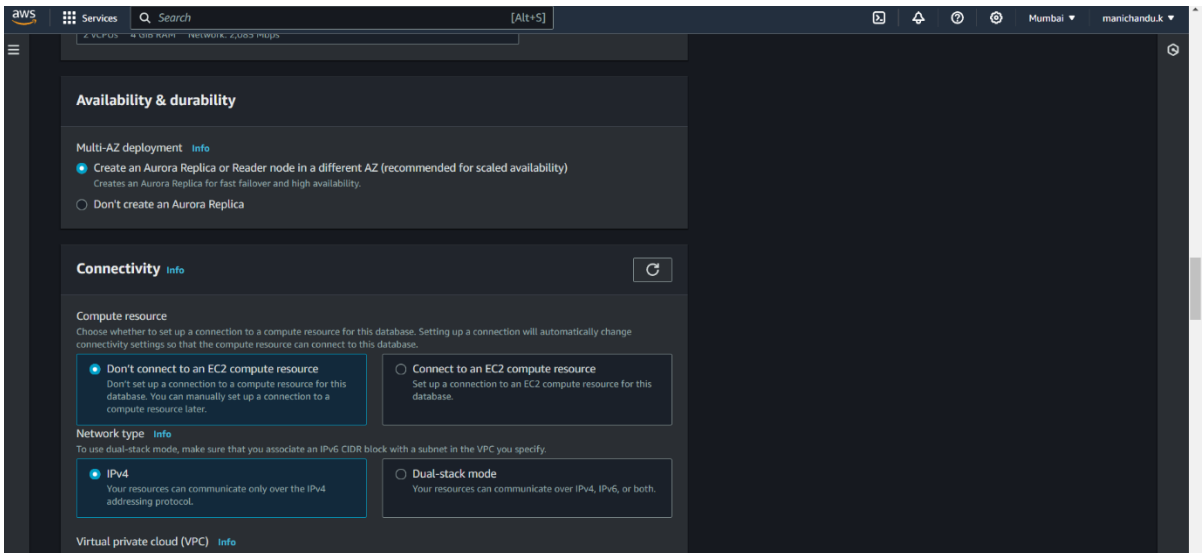



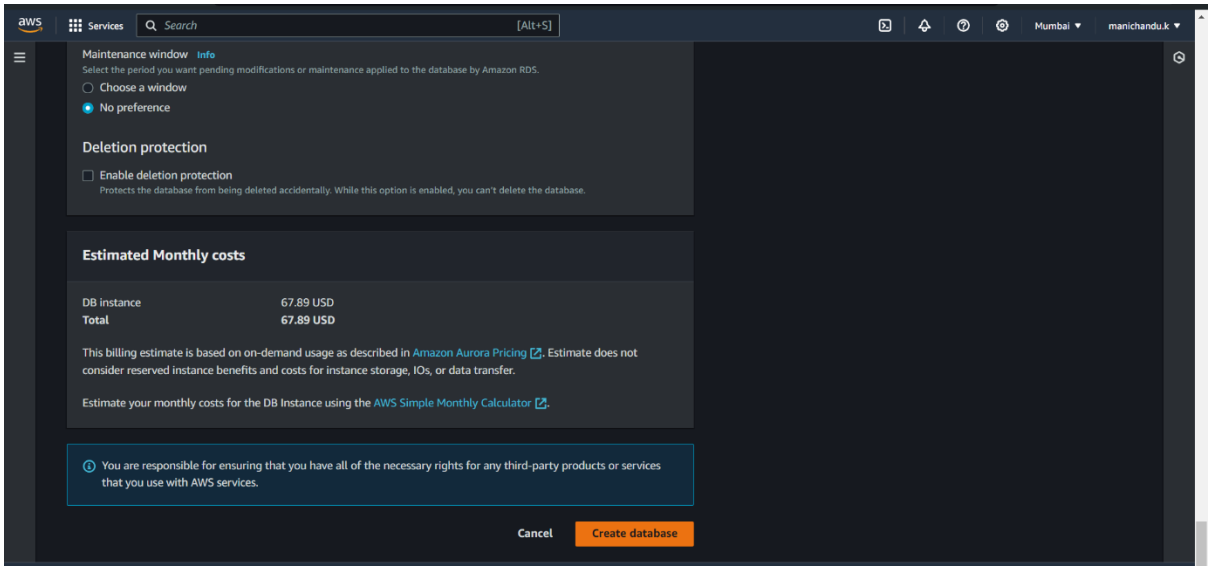

- Click on Create and your DB subnet will be created.
- Now go to the database section and create a database with any name of your choice and select the Standard create option, select the aurora(MySQL) database as we have given its port number in our security group.

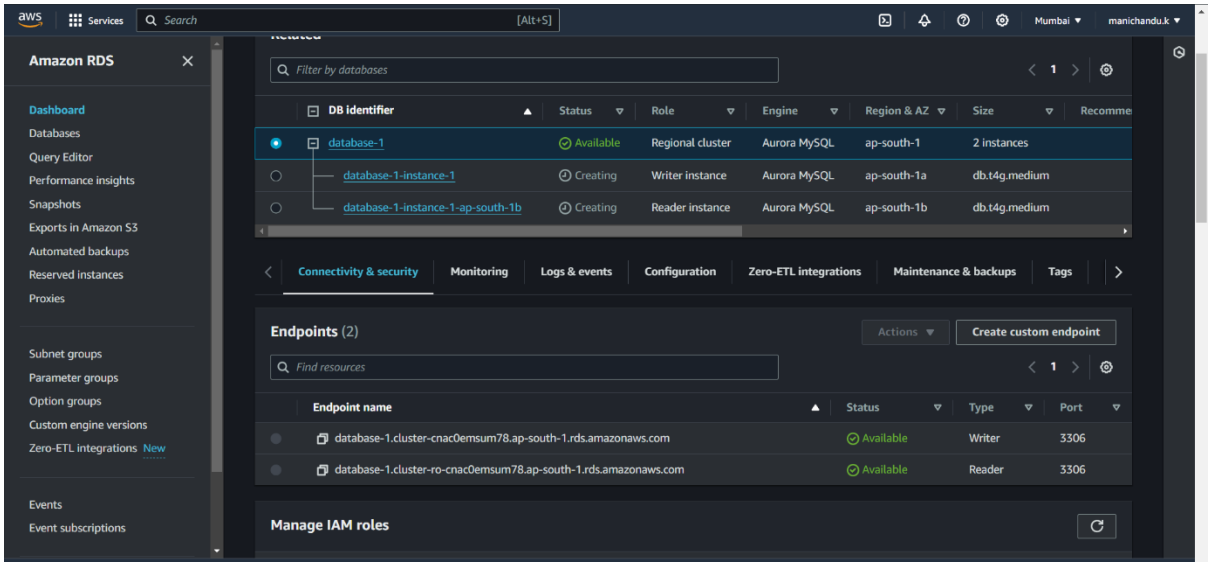
- Now select your preferred type class, here I selected dev/test class. In the process, you can mostly keep it default. You can self-manage your password but remember the credentials.
- when you have come to the section of connection to an EC2 instance select don’t connect to an instance and change the VPC to our custom VPC and the subnet group will default select our DB subnet group.




- Here you should make sure to select multi-AZ deployment so that our application will be highly available as it will create a read replica of our DB instance in another available zone.
- Now click on create and our DB-instance cluster will be created.



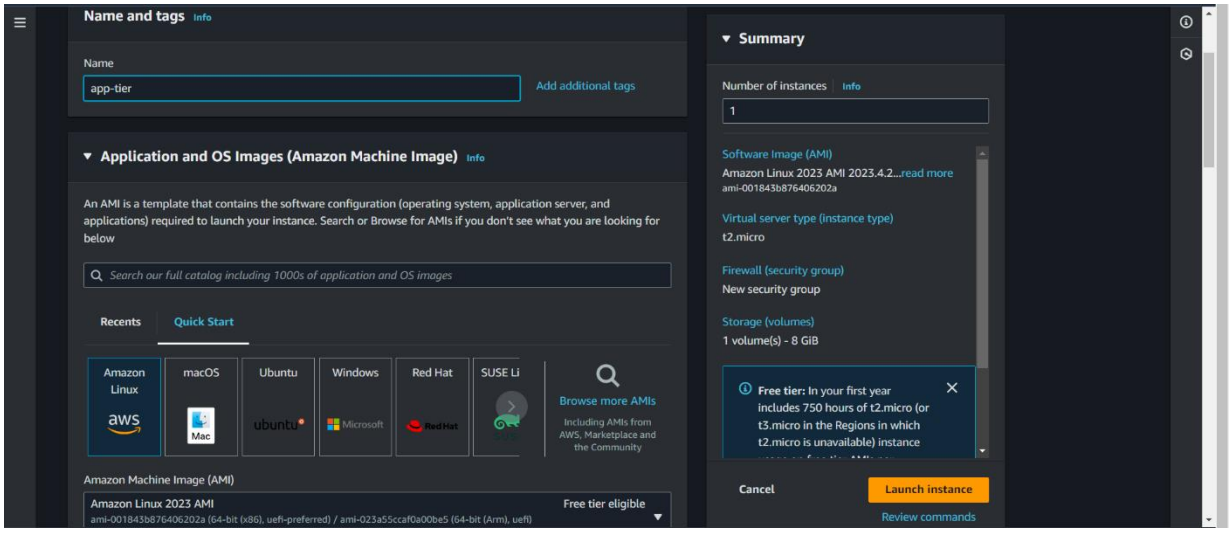



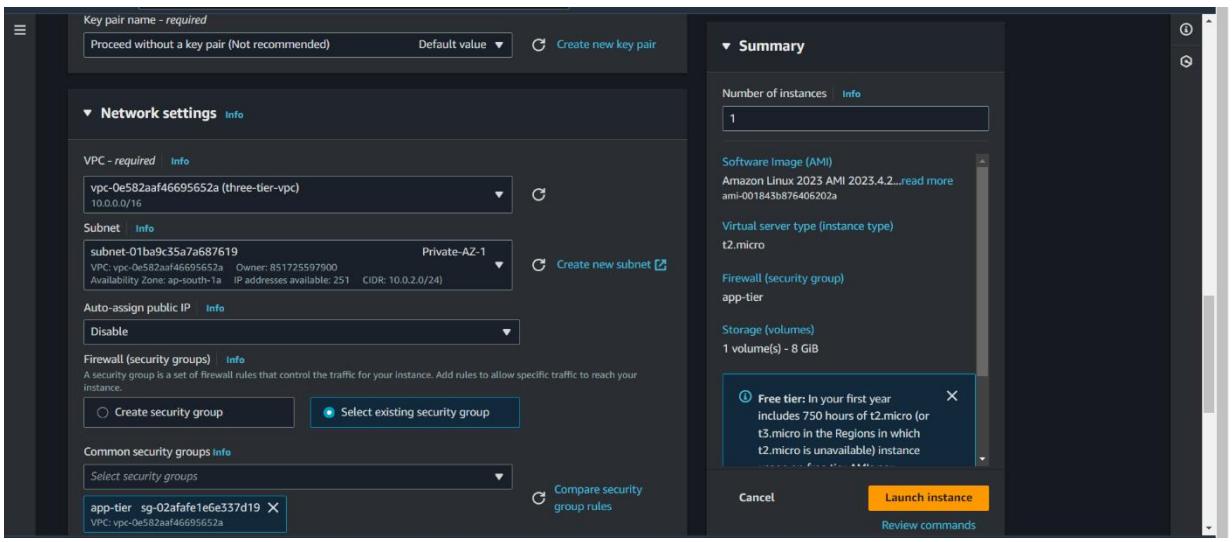

- The database cluster is successfully created and we should see our writer instance end-point.
- Now we should move on to the next step which is deploying our application in the ec2 instance.

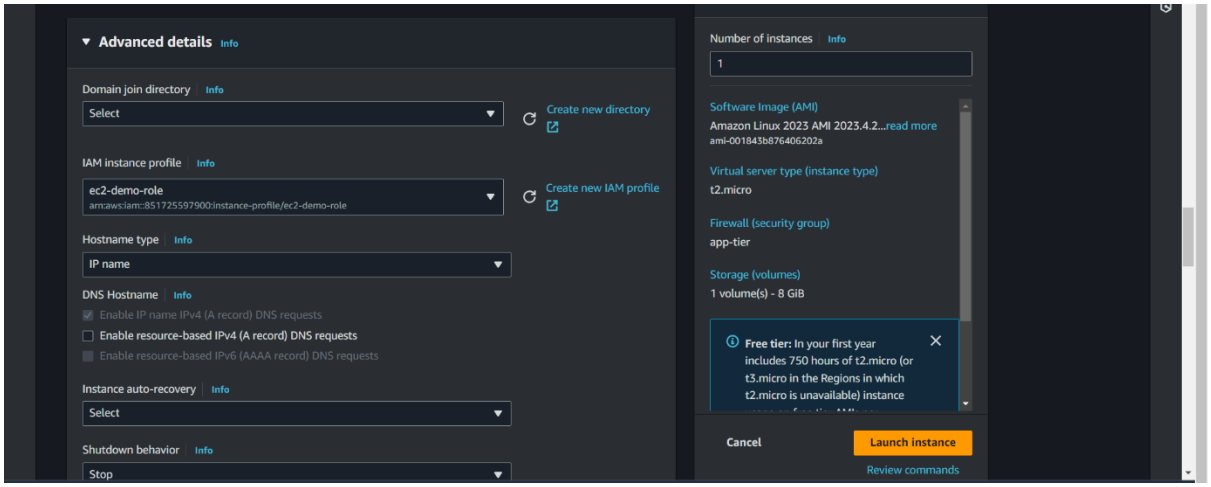
# App-Tier Instance Deployment :

- In this section we will deploy our application in a private instance and create a template out of it using AMI.
- We will also configure our database in this private instance so it will be more secure as shown in the architecture.
- Go to EC2 service and click on launch instance.

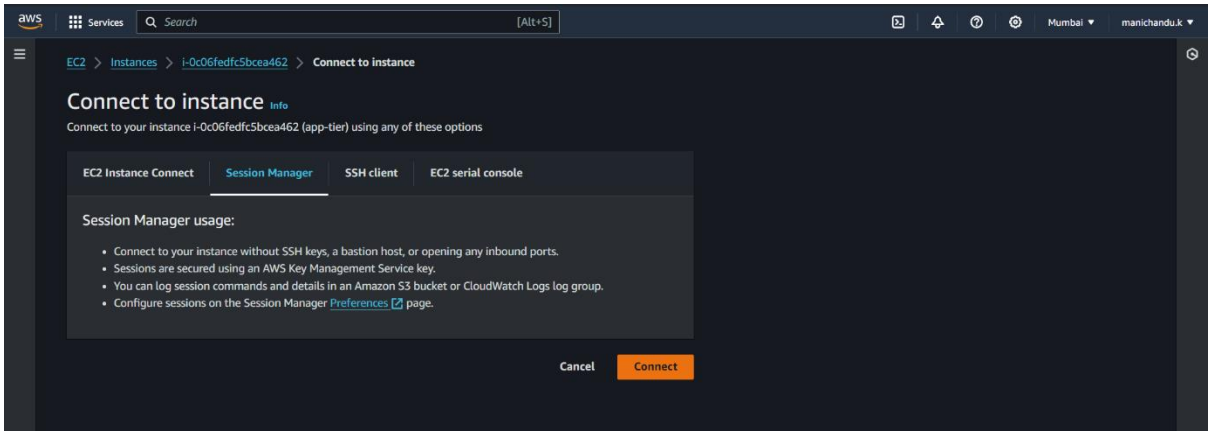
- Now give the name as app-tier, select AMI as Amazon Linux, and select the instance type of your choice. Here I selected t2.micro as it is free-tier eligible.
- Now proceed without key pair configure network details, select our custom VPC, and use any private subnets we created as we deploy our app in private subnets. Do not assign a public address for this instance
- Now select our app-tier security group for this instance, before launching the instance go to the advanced details section, click on the IAM instance profile, and select the previously created role.



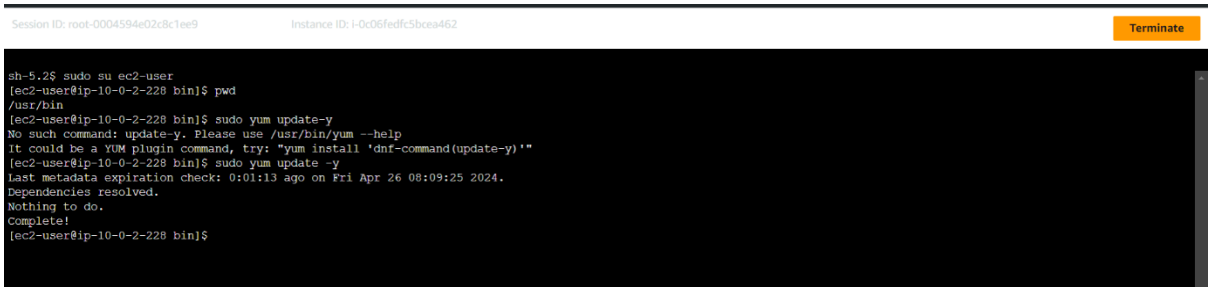






- Now launch the instance and when it is available click on connect, you will go to the session manager connect page and click on connect.




- Now you will go to the terminal page of the instance and switch to the ec2 user.



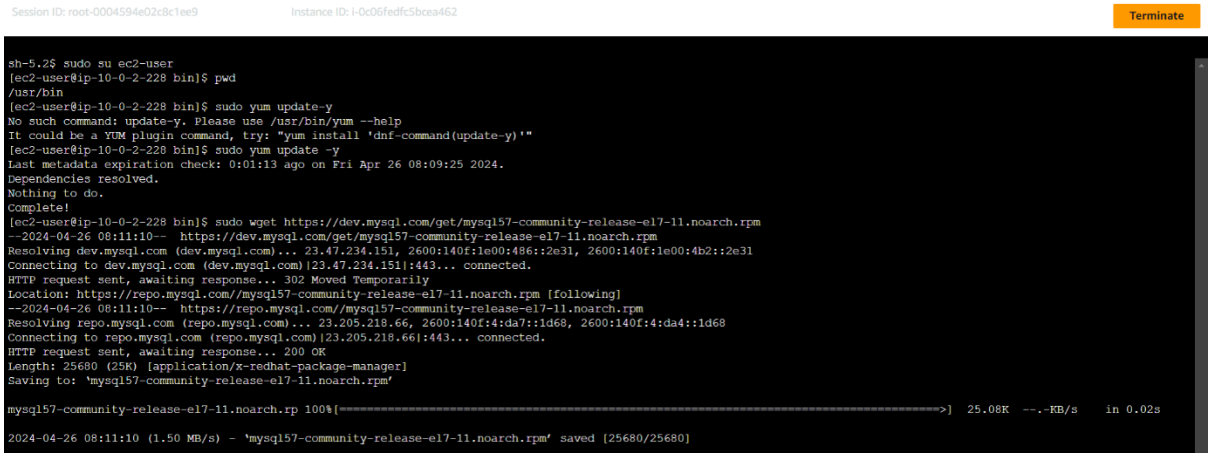

- After you connect to your private instance we have to install MySQL in our server to access the database we created.
- The following commands should be given in order.

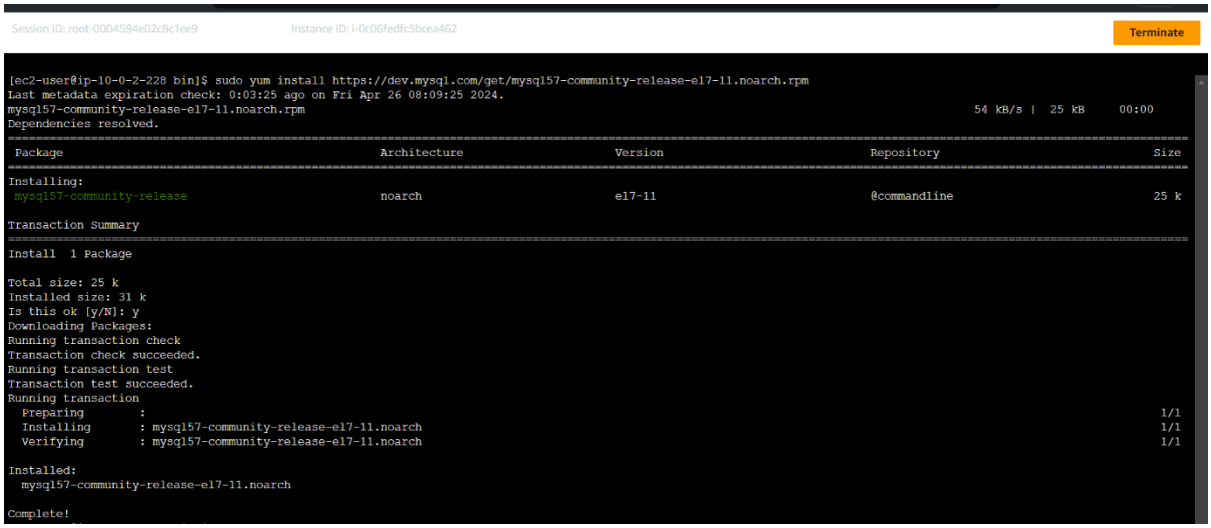
sudo wget https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm

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

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

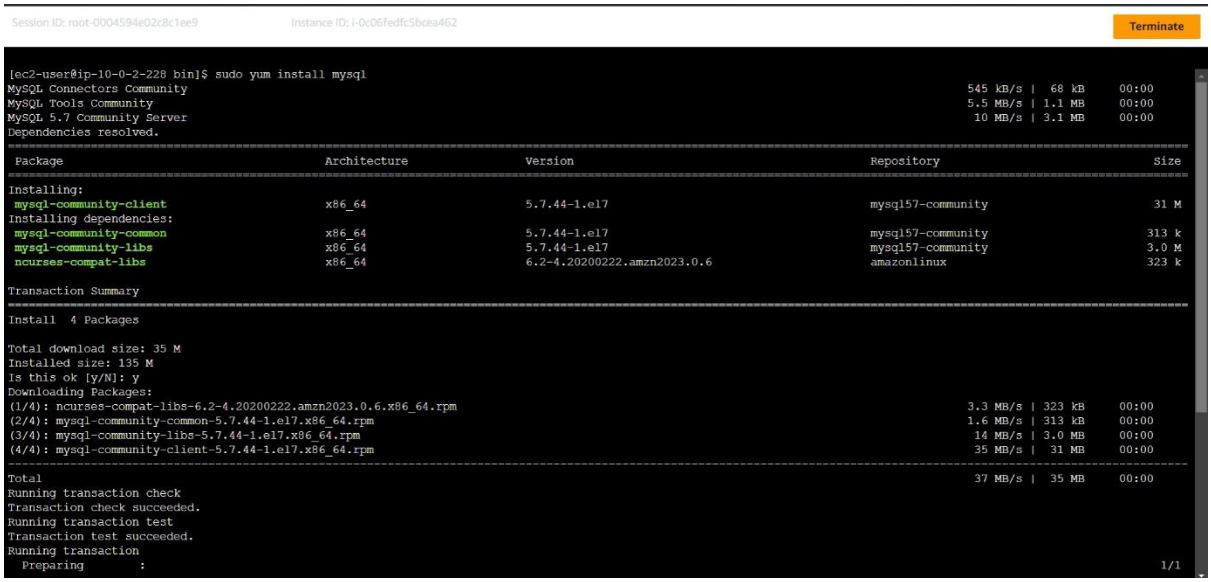
sudo yum install mysql

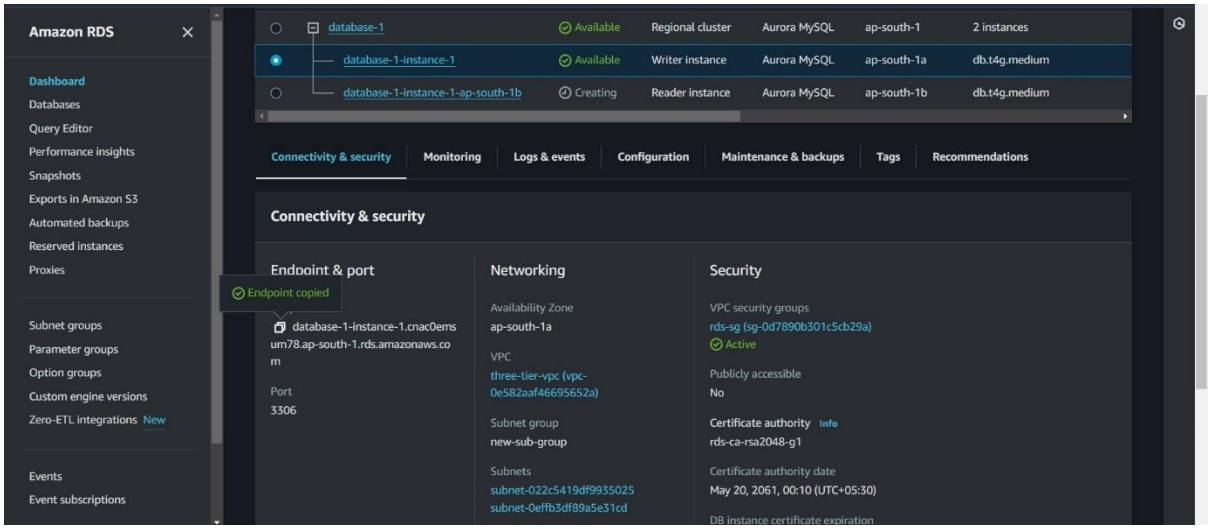


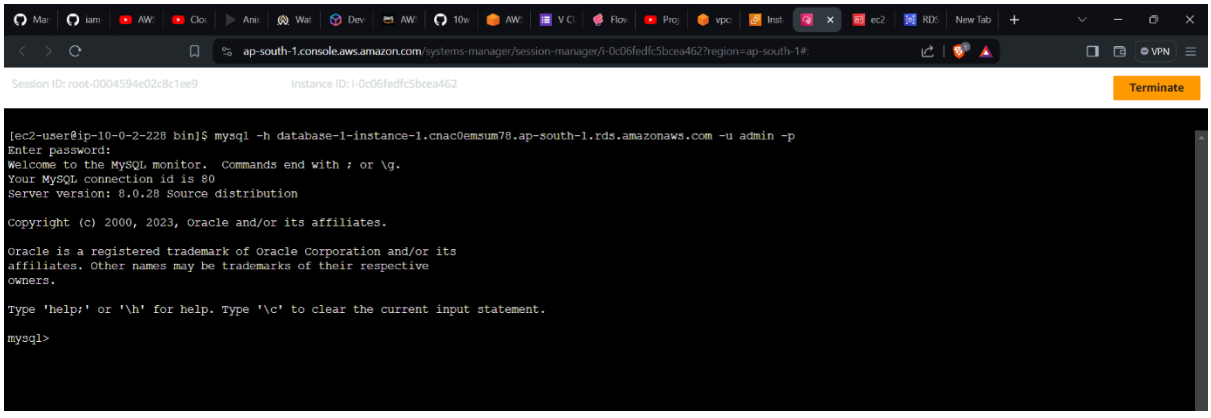



- Now give the following command to connect to the database we created.

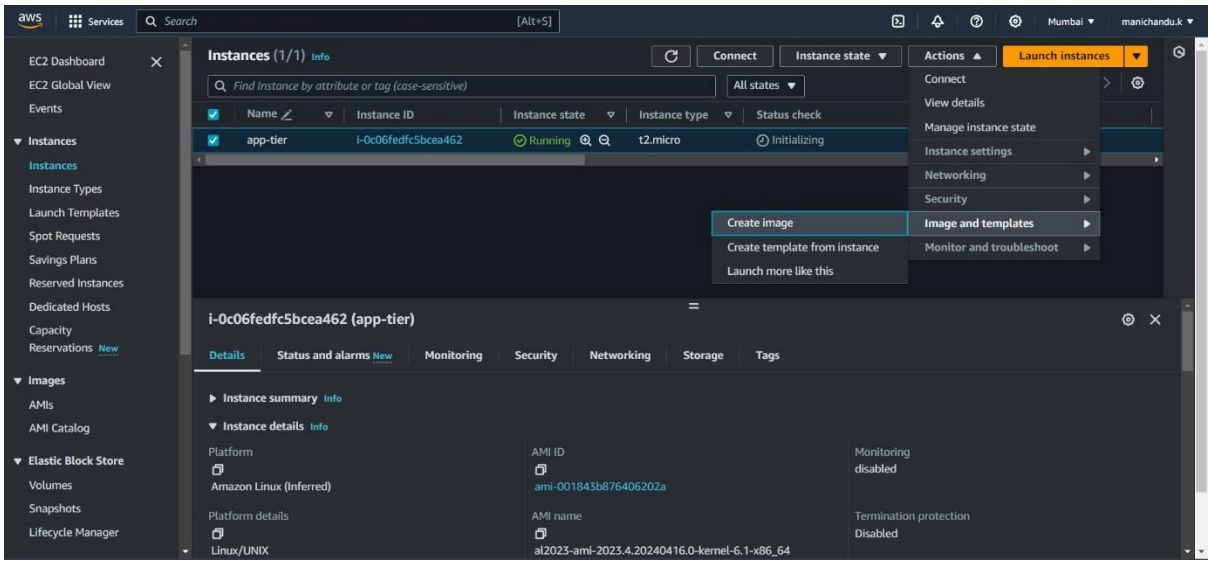
  mysql -h database_writer_instance_endpoint -u username -p

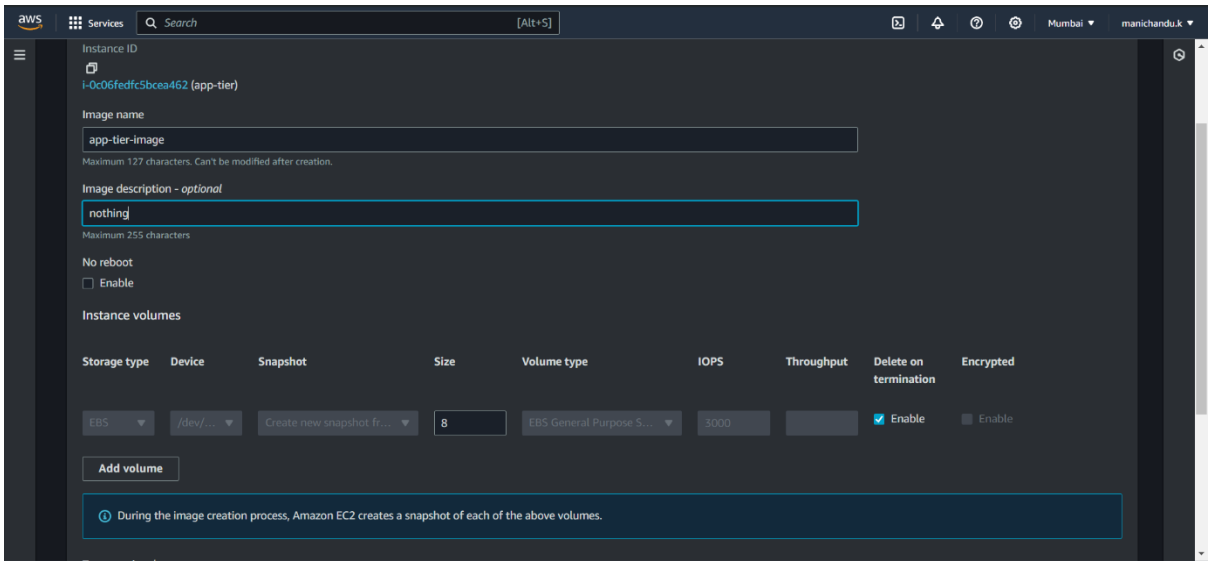


- You will be connected to the database and you can create databases, and tables through CLI in this database.
- After creating databases, if you have any application code, upload app tier configuration files or node.js files to your server using s3 bucket through simple CLI commands.
- Now log out of this session and create an image of the app-tier instance.

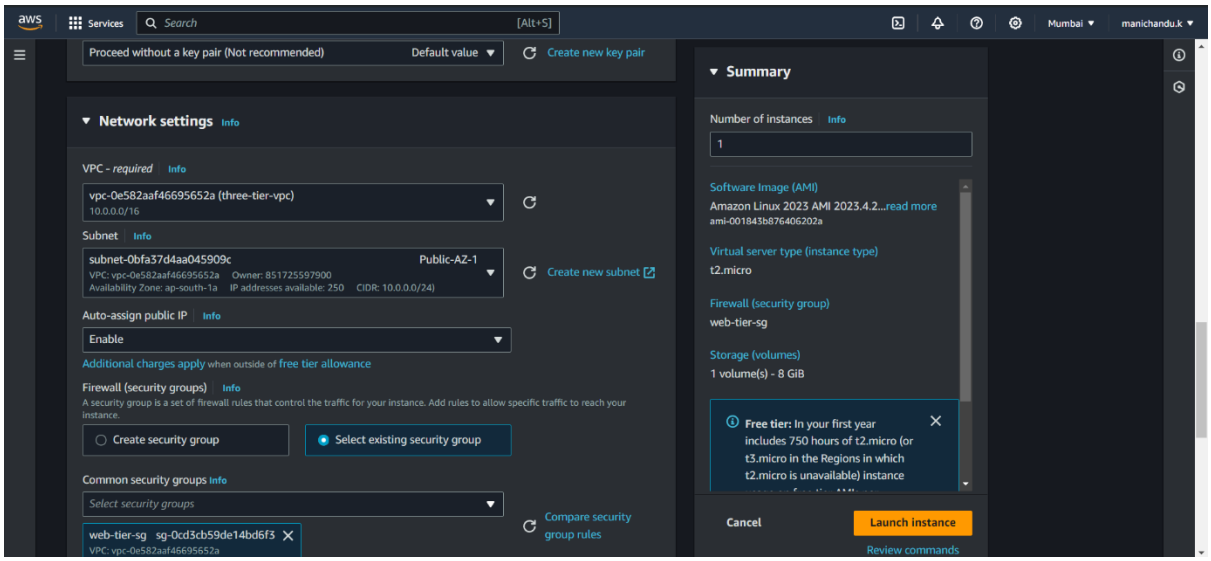


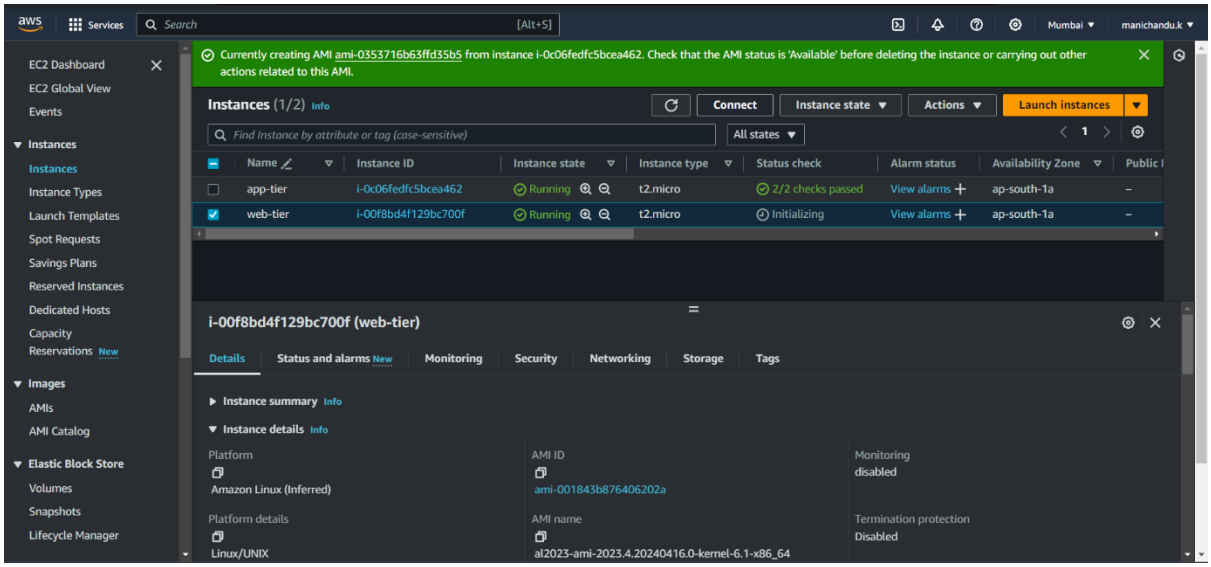

- Leave the image aside and launch another instance for our webpage. The webpage instance should have the web tier configuration files which is a user interface.

# Web-tier instance deployment:

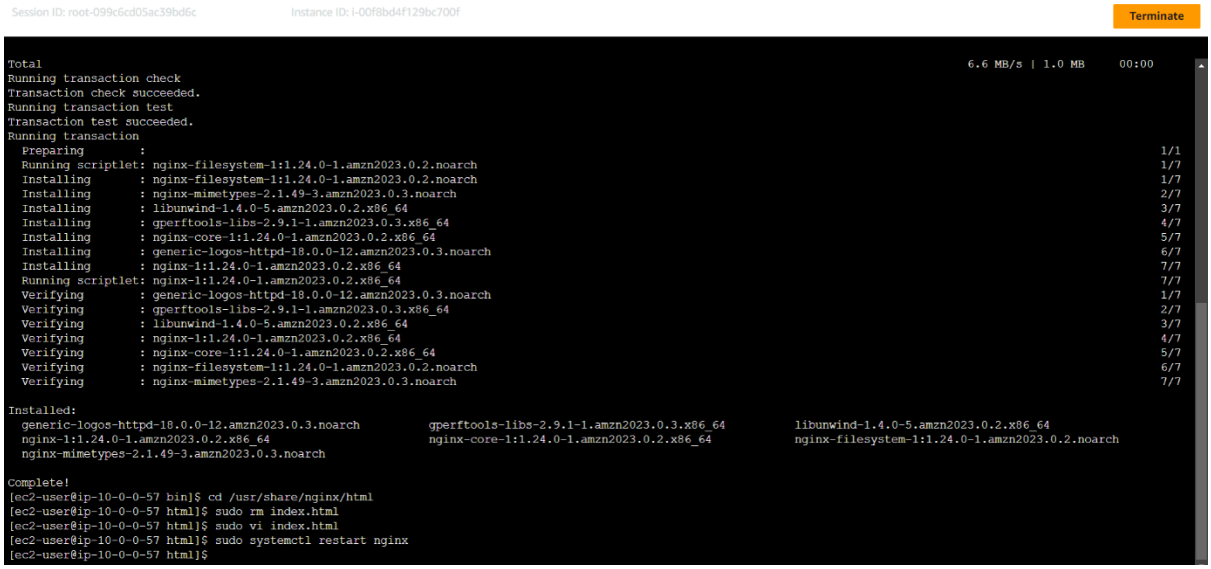
- It has the same process of app-tier instance deployment.
- When launching the instance make sure that you configure your network settings, and switch your VPC to custom as it should have internet access and the instance should be accessed by the user, it should be launched in the public subnet.



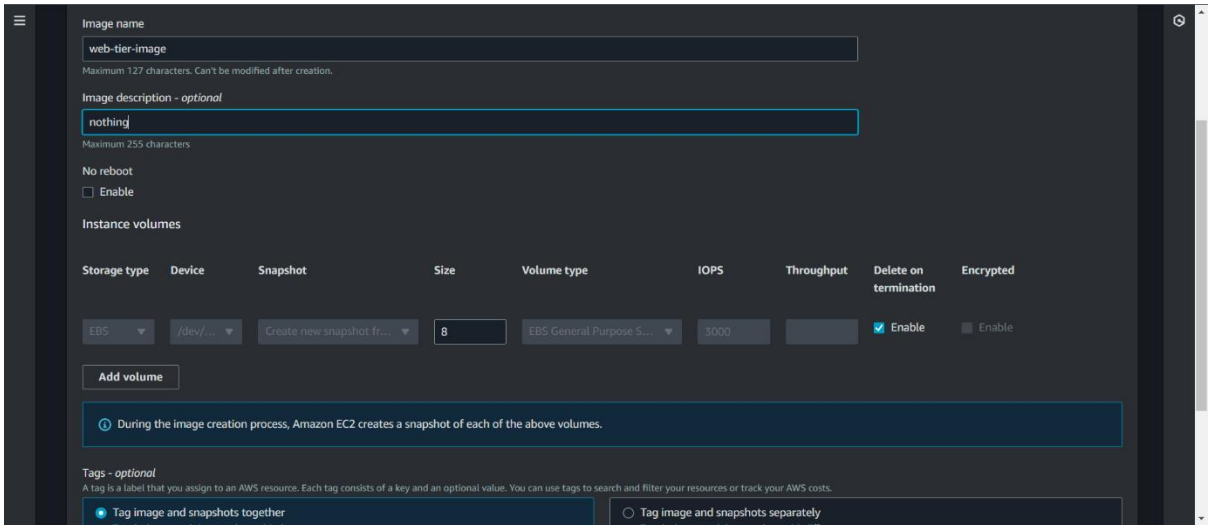

- Select the security group as web-tier-sg, as we have created previously.
- As we have proceeded without keypair, we should give the IAM role to this instance to connect to it.
- After launching the instance, connect to it through the session manager as we did before.




- Once you connect to the instance, you can upload webpage configuration files to your instance through the s3 bucket. Your web page can be based on nginx or Apache as we have only given port number 80 in the security groups.



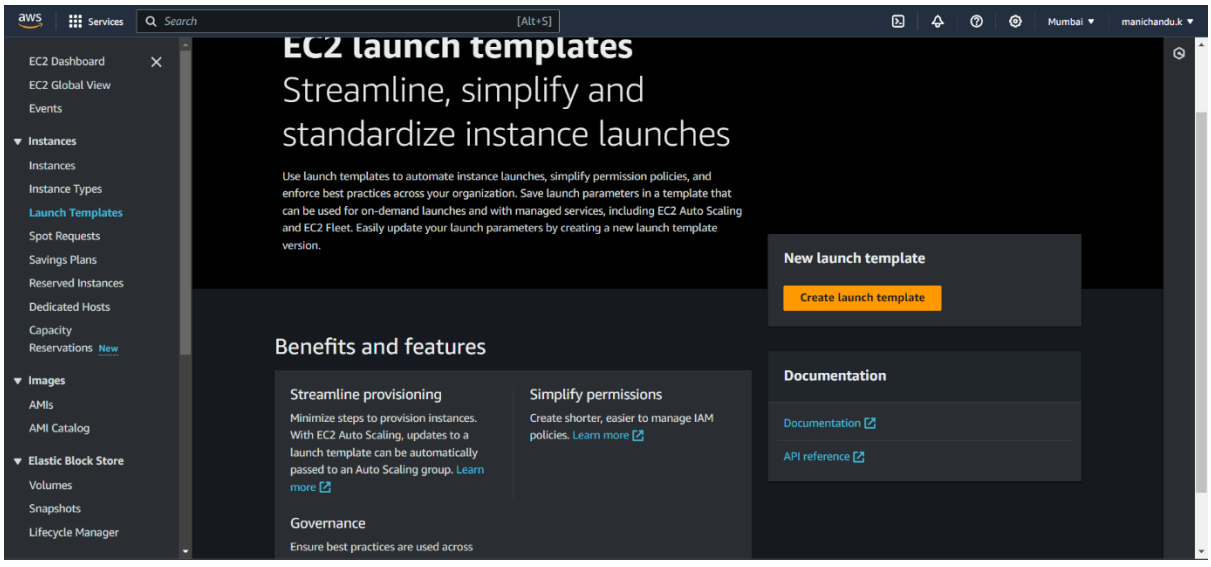

- Now log out of the session and create an image of the web-tier instance.

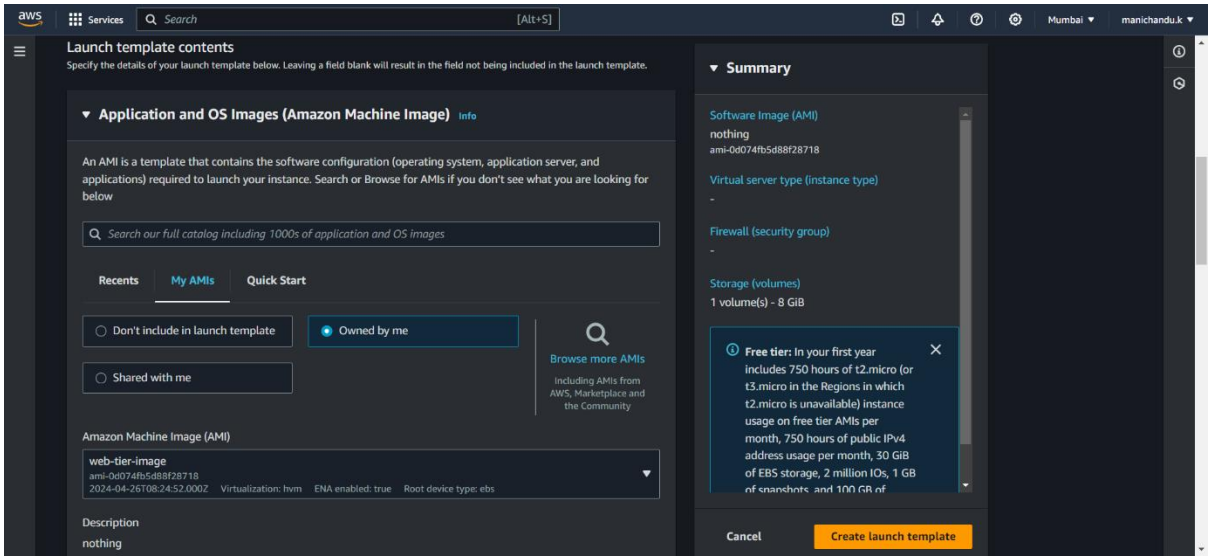


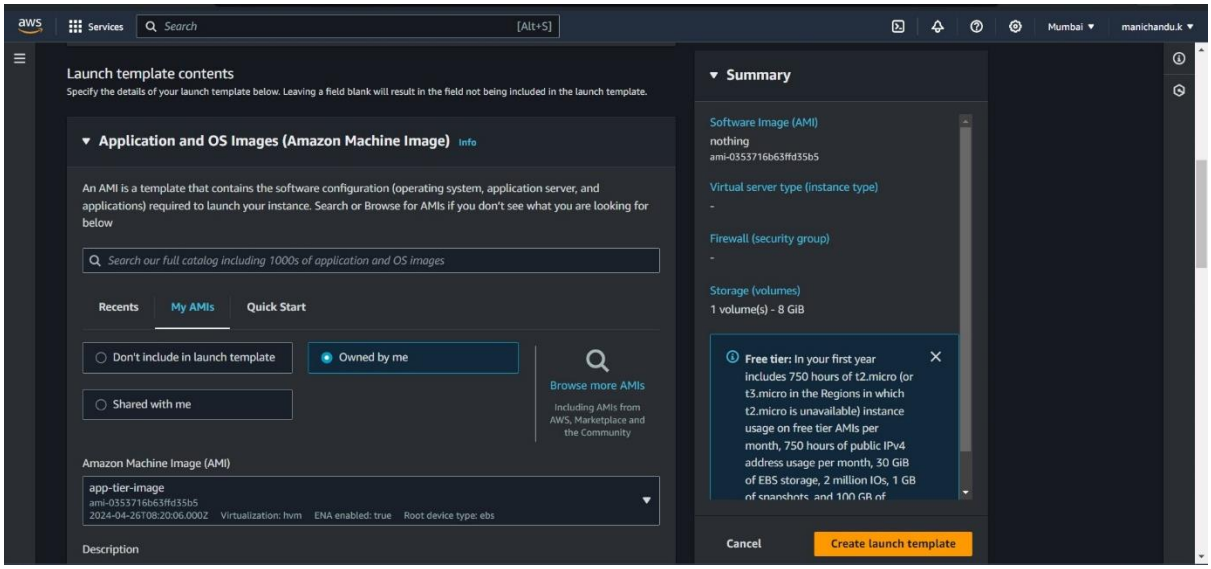

- We have two AMIs, one is a web-tier image and another is an app-tier image, now launch two templates using these two AMIs.

# Launch Templates:

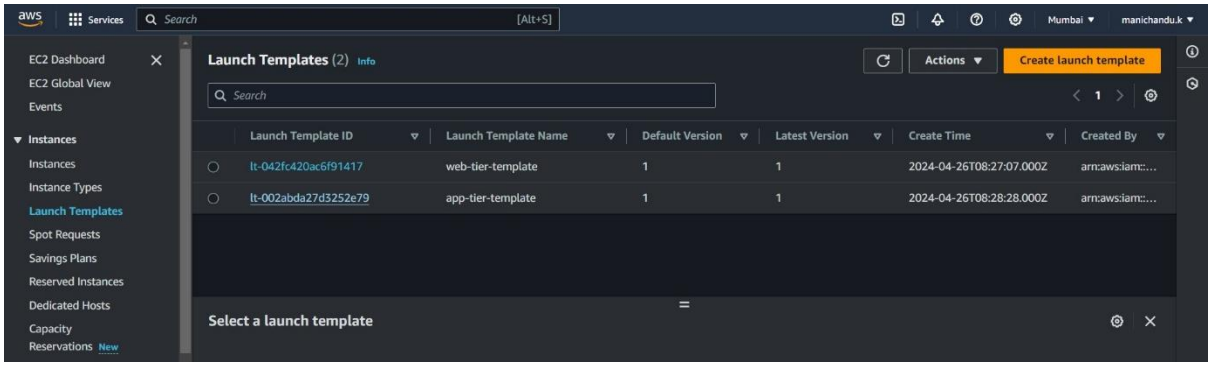
- While creating templates use their respective AMI's in the AMI section.
- now launch those two templates and they are helpful while creating autoscaling groups.





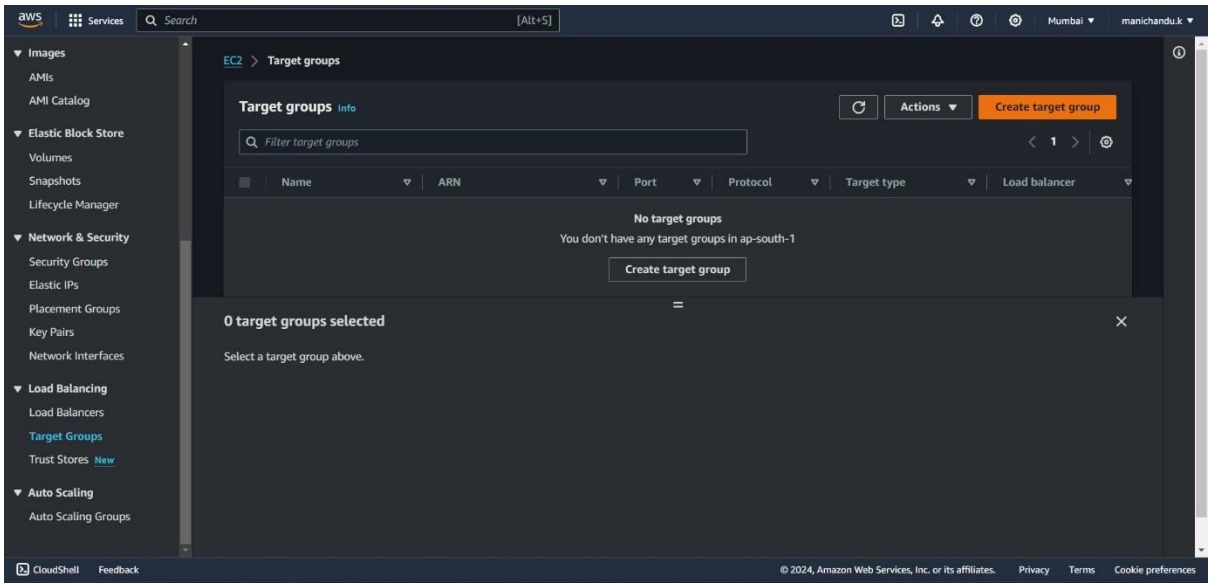




- Pay attention while creating these two templates, you should give their respective security groups to function properly.

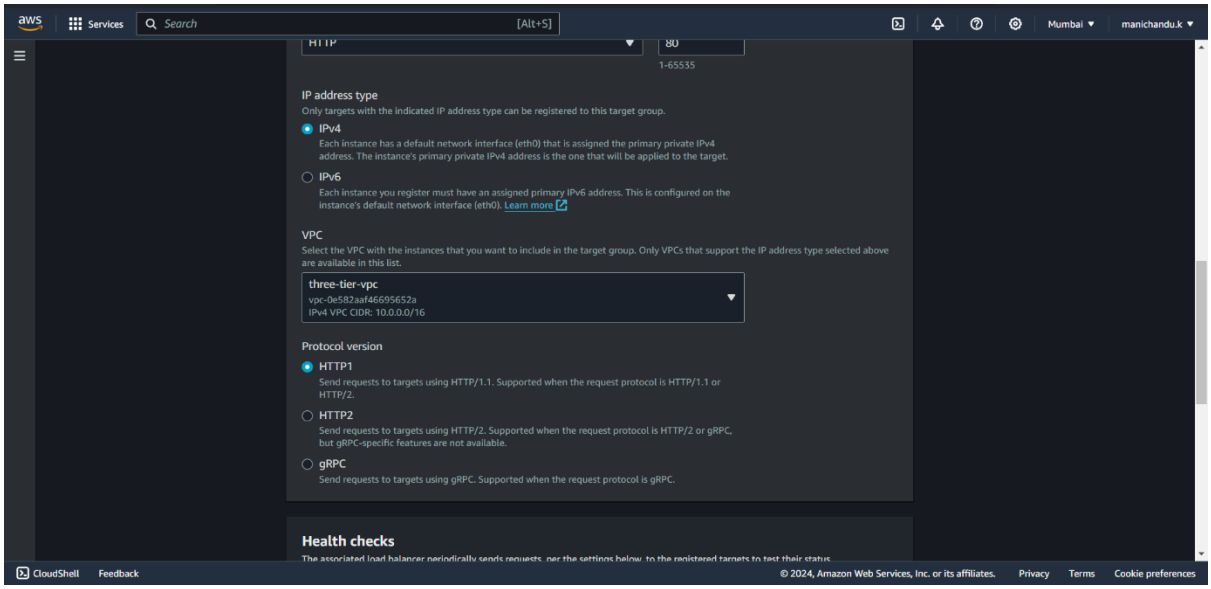


# Load balancing and Auto scaling :

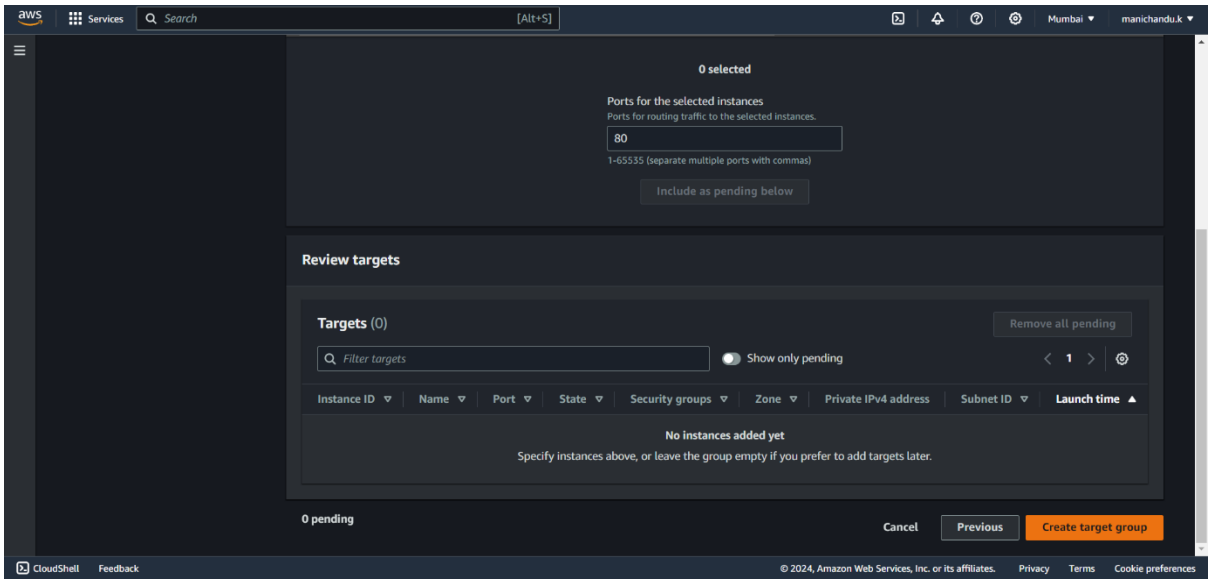
- After creating the launch templates, go to the Load balancing feature of EC2 and click on target groups to create a target group for our load balancer yet to be created.



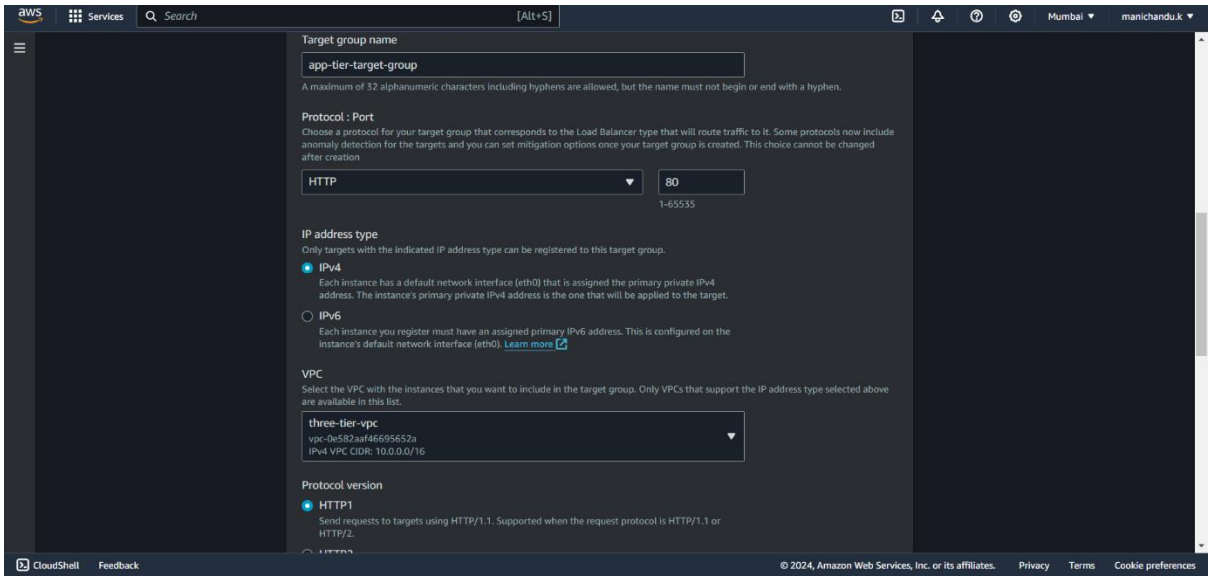

- Click on create target group and give a name to our target group, here I created a target group for my web page instances, so give a name according to your preferences.
- Select our manually created VPC when choosing VPC and give the port number as 80.

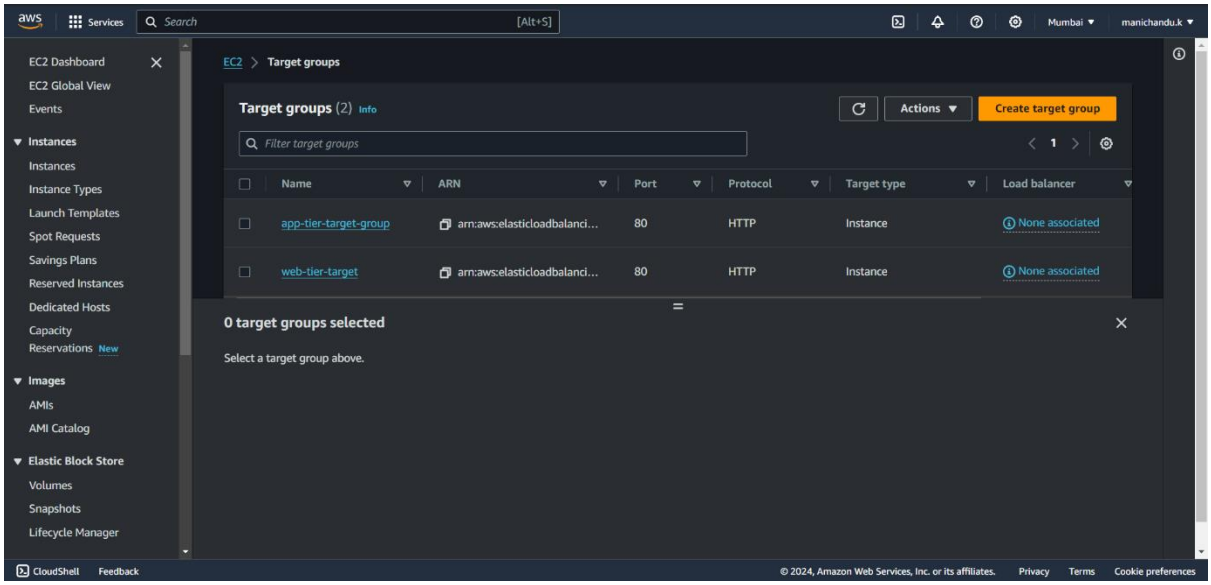


- Click on next and don’t select any instances as pending below. Create the target group without any instances as the target.

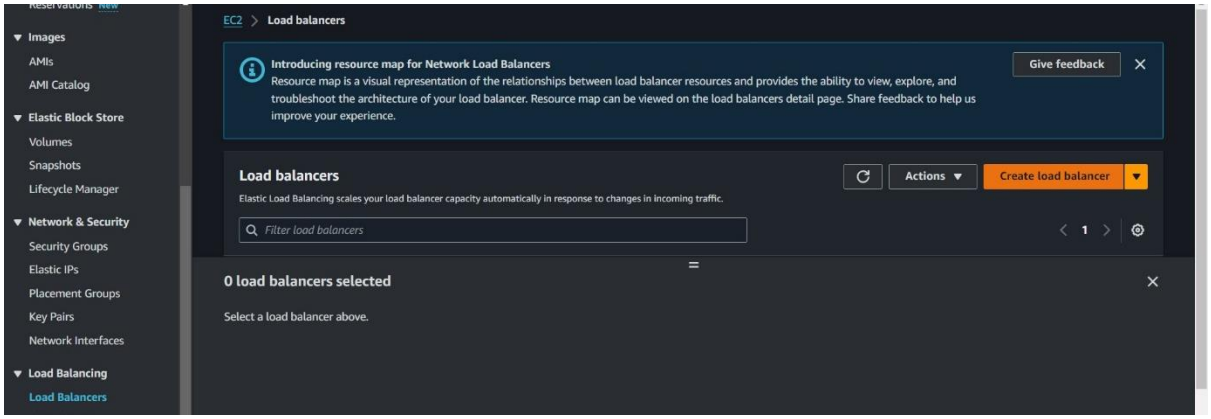



- Now create a target group for our app tier instances following the same steps.

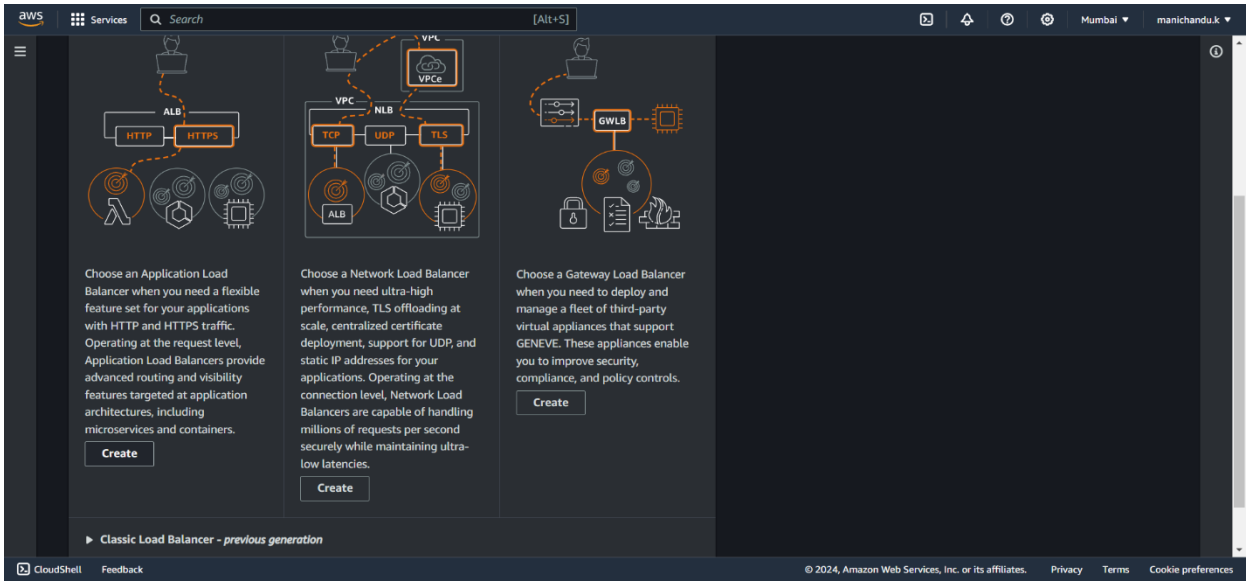




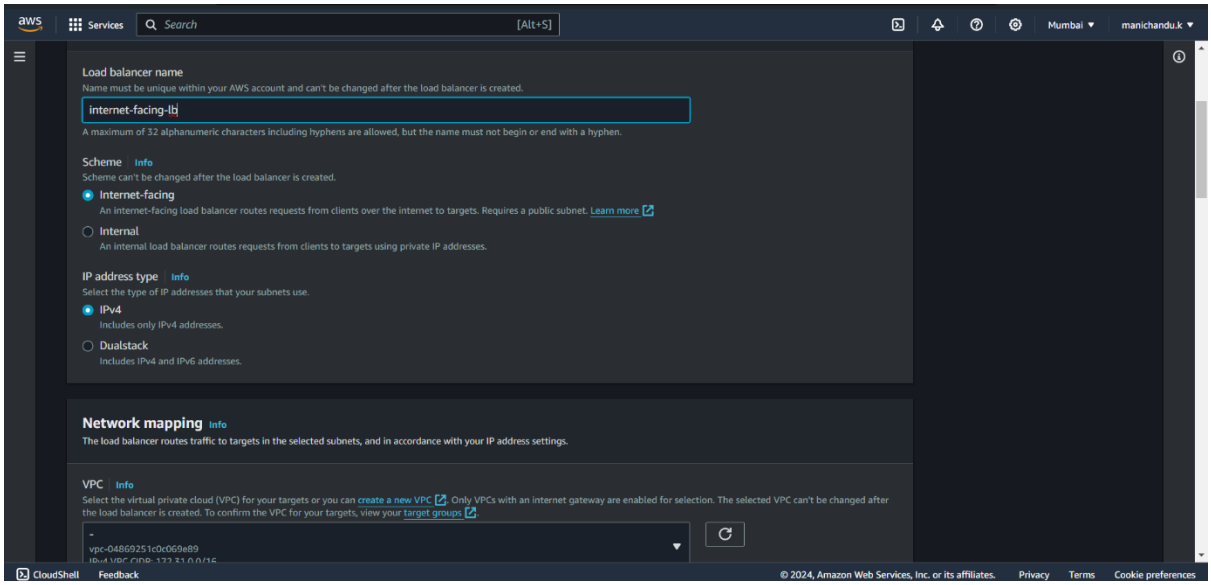

- Now go to the load balancers option and create two load balancers for both instances using target groups.



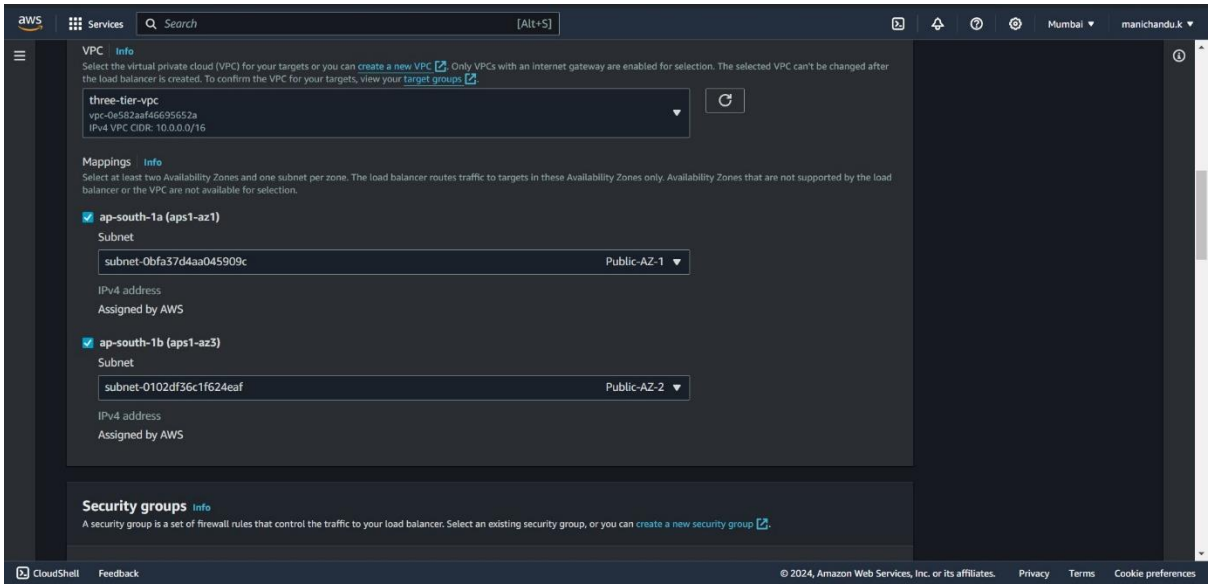

- Click on create load balancer and select application load balancer.



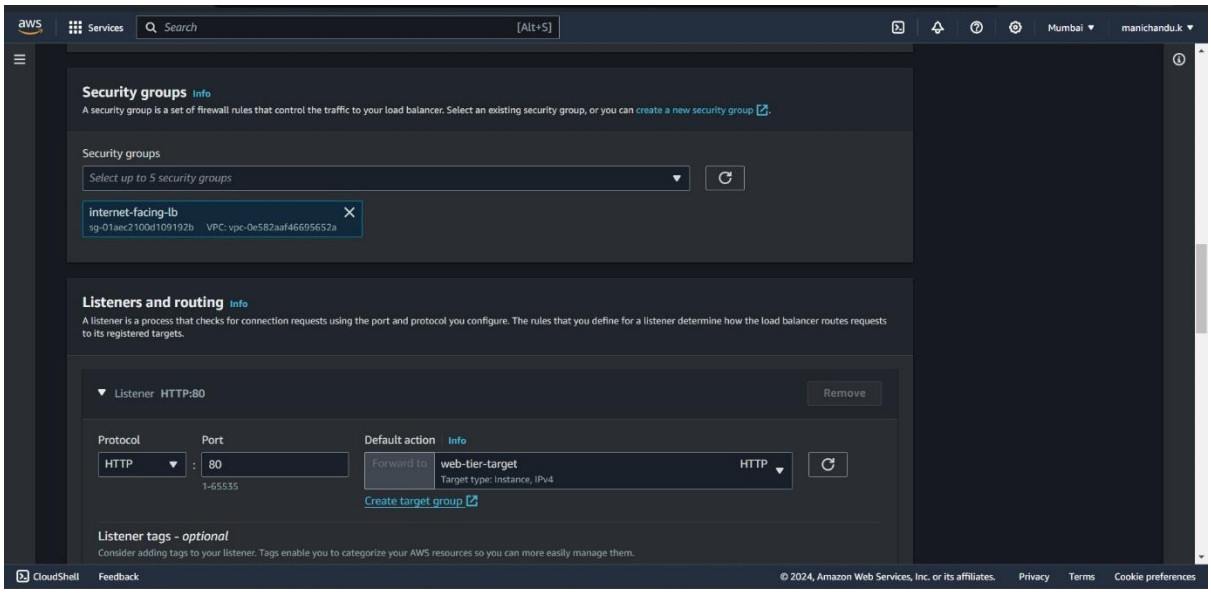

- Click on Create application load balancer and give a name to it.
- Select internet facing option as it is our web tier instances load balancer and it should have access from the internet.



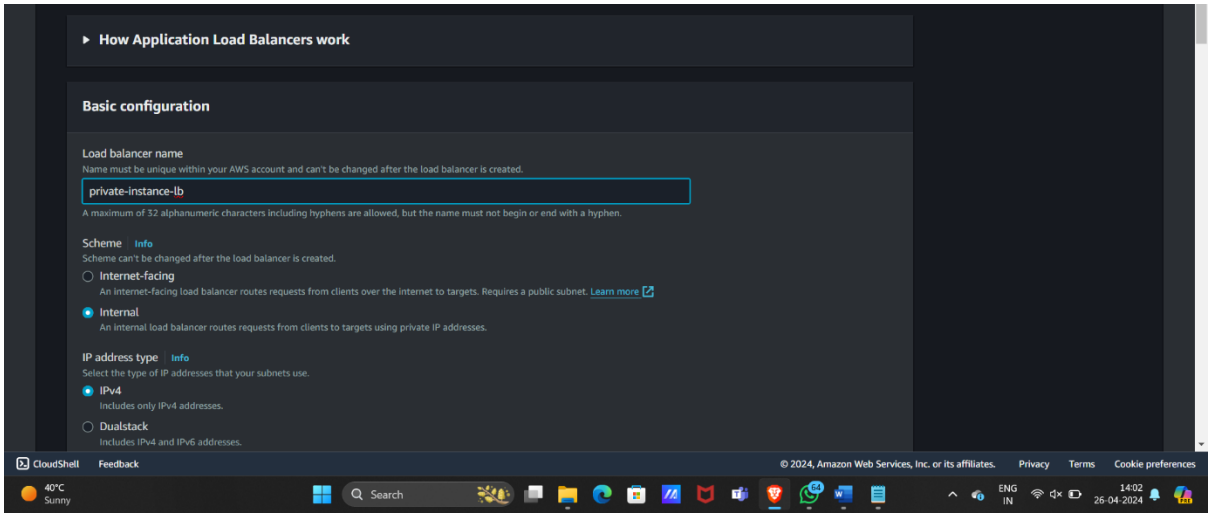

- Now select our custom VPC and select regions where we created our public subnets and select only public subnets.

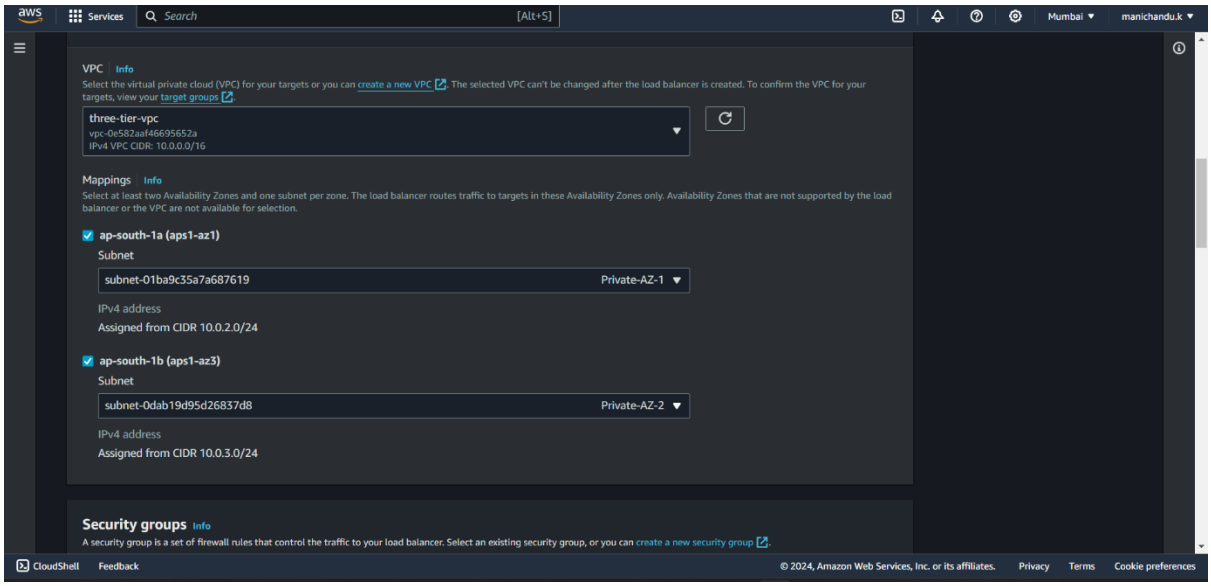



- Now move to security groups and select the internet-facing-lb security group that we created earlier, select the target group that we created for our web pages here.

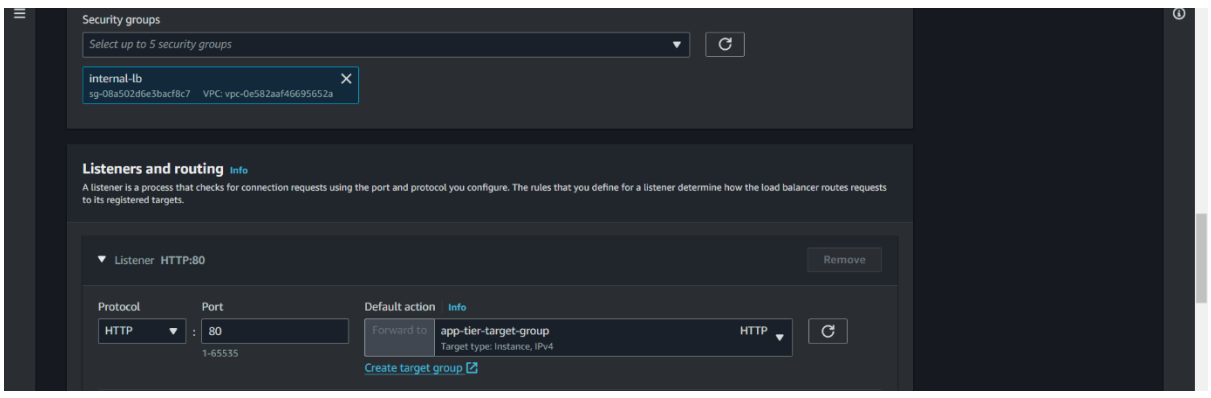


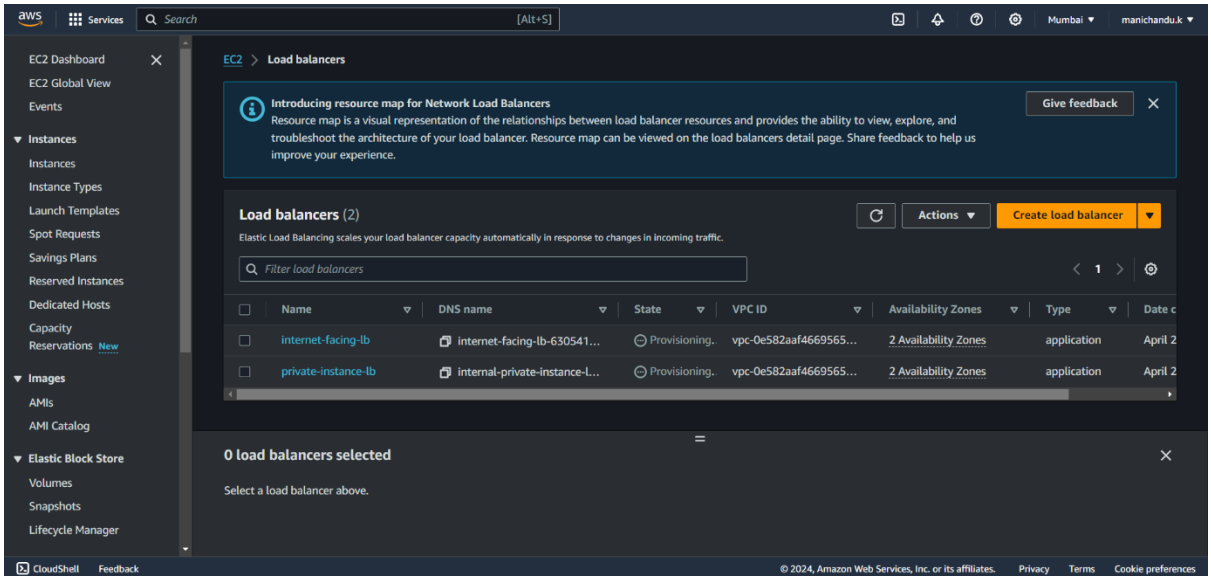

- Create the load balancer for web tier instances and now create the load balancer for app tier instances.
- When creating the load balancer for app-tier instances make sure that you select internal facing as the app-tier instances do not have to be accessed from the internet.
- Select the manual VPC and select the private subnets in both regions.



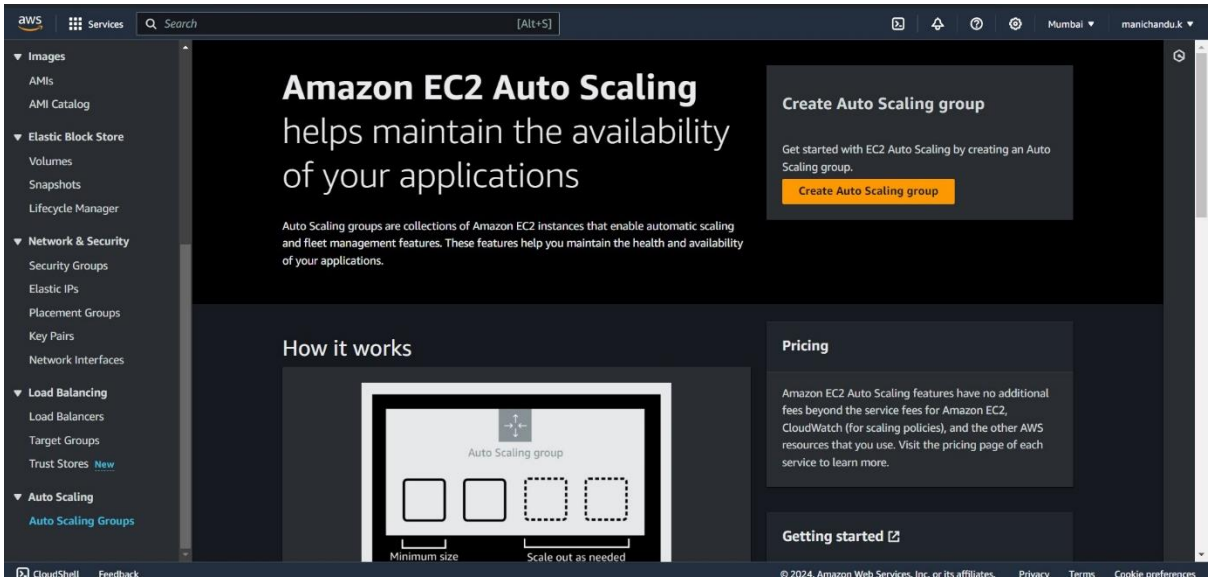




- Now select the internal-lb security group and app-tier target group and launch the load balancer.
- Now we have two load balancers, one is for web tier instances and another is for app tier instances.

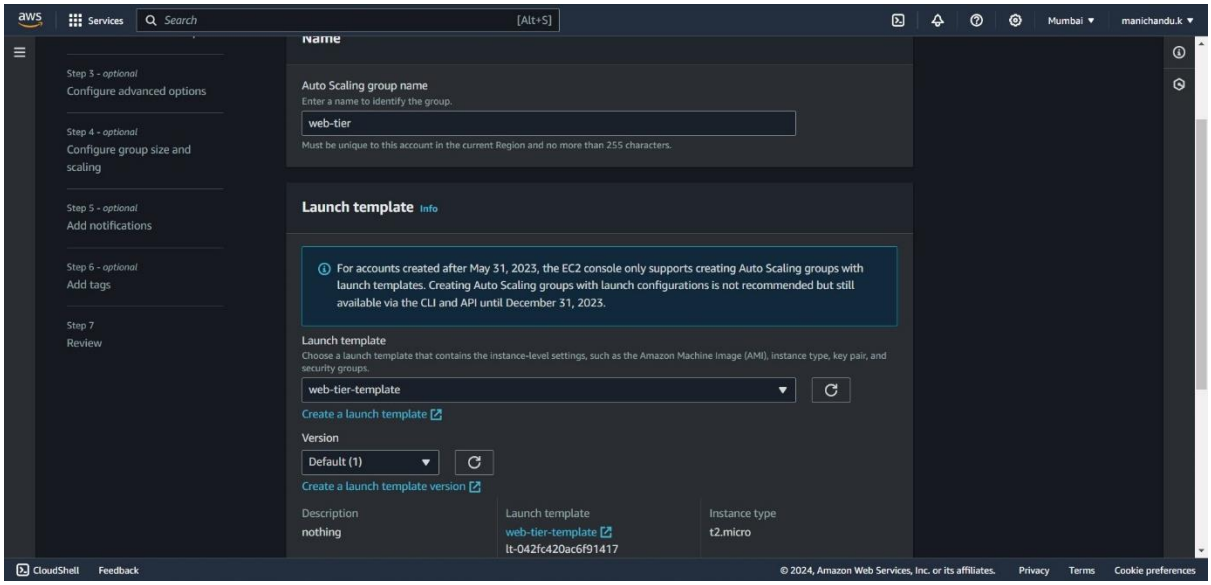


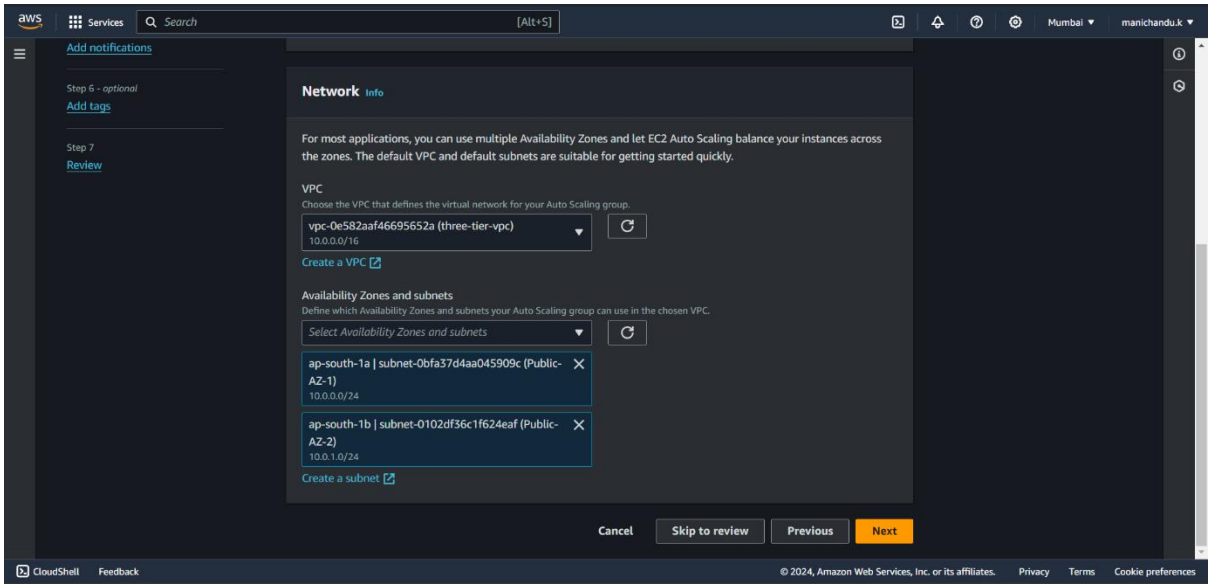



- Now it’s time to create an auto-scaling group for both app-tier and web-tier instances.
- Go to the auto-scaling option to create auto-scaling groups for the app tier and web tier instances.

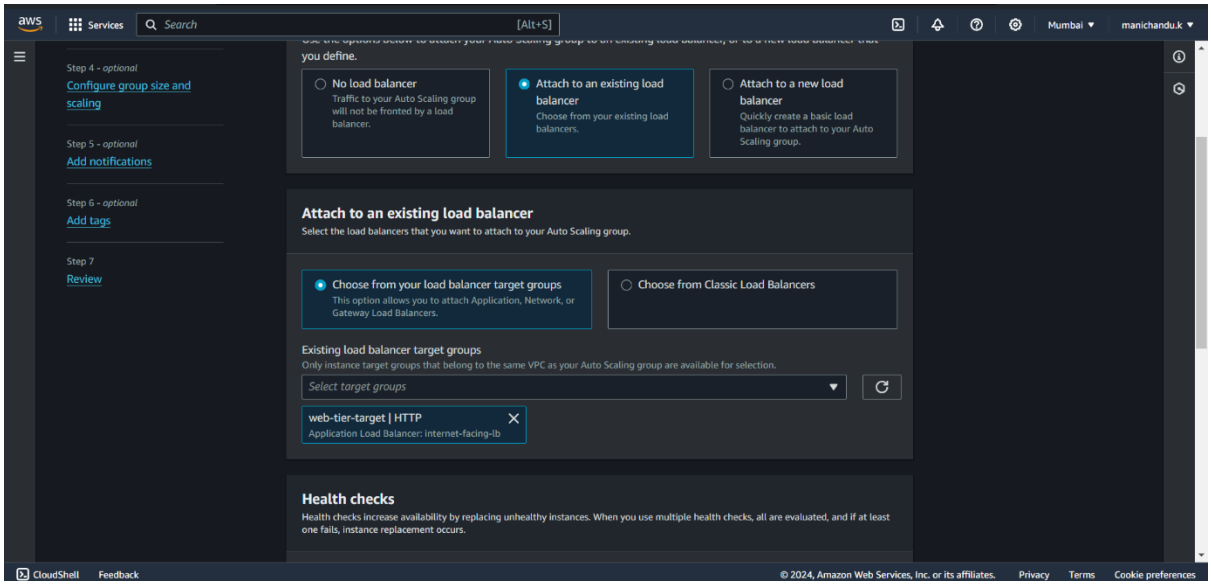


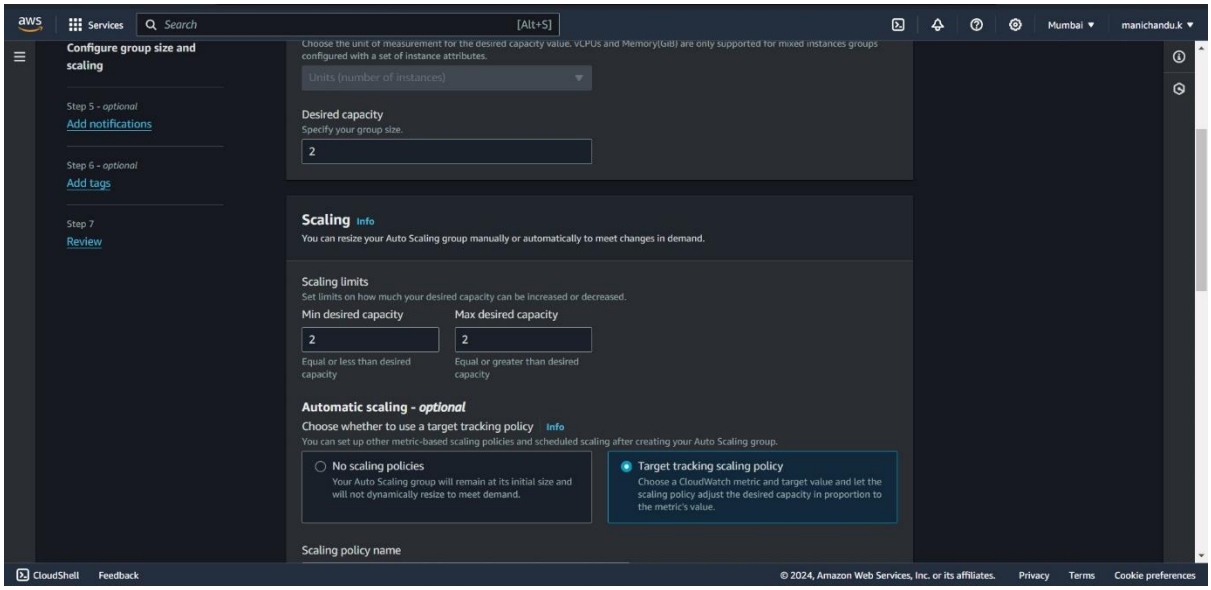

- Click on Create Auto Scaling group, give the name as web-tier, and select the web-tier template that we created earlier, configure the network settings by selecting the custom VPC and select the public subnets in both regions.



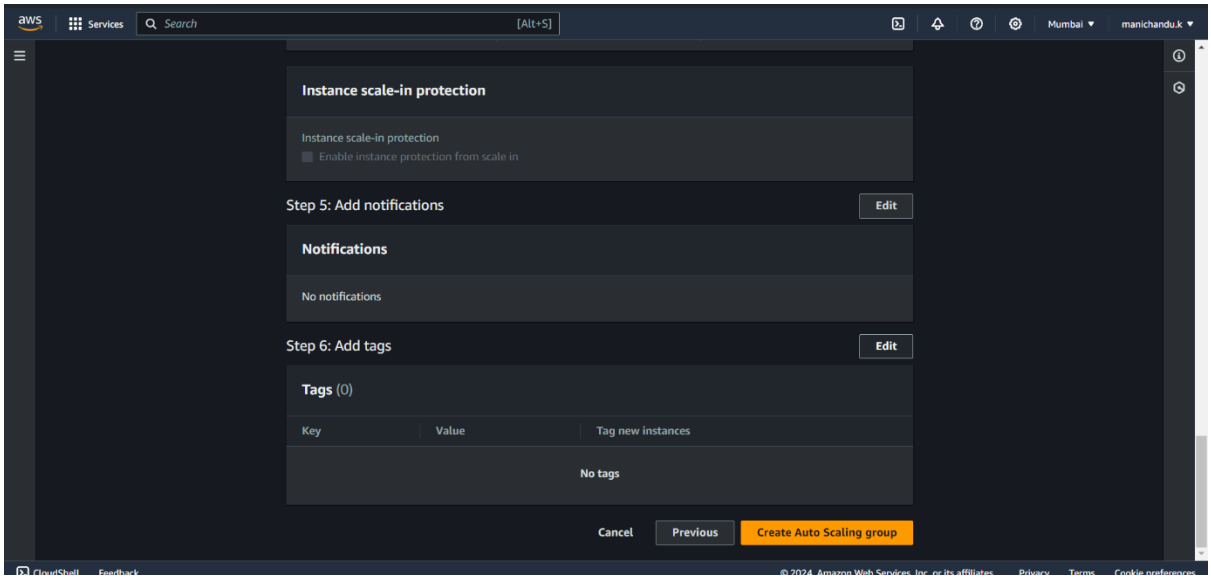




- Click on Next and select Attach it to an existing load balance, now select the target group that we created earlier for the web-tier instances.
- The health check time can be put default or custom. Here I left it as default. Click on Next and select the number of instances you want to scale, here I want two instances to be always running.
- So I selected two minimum and maximum desired capacities of instances.

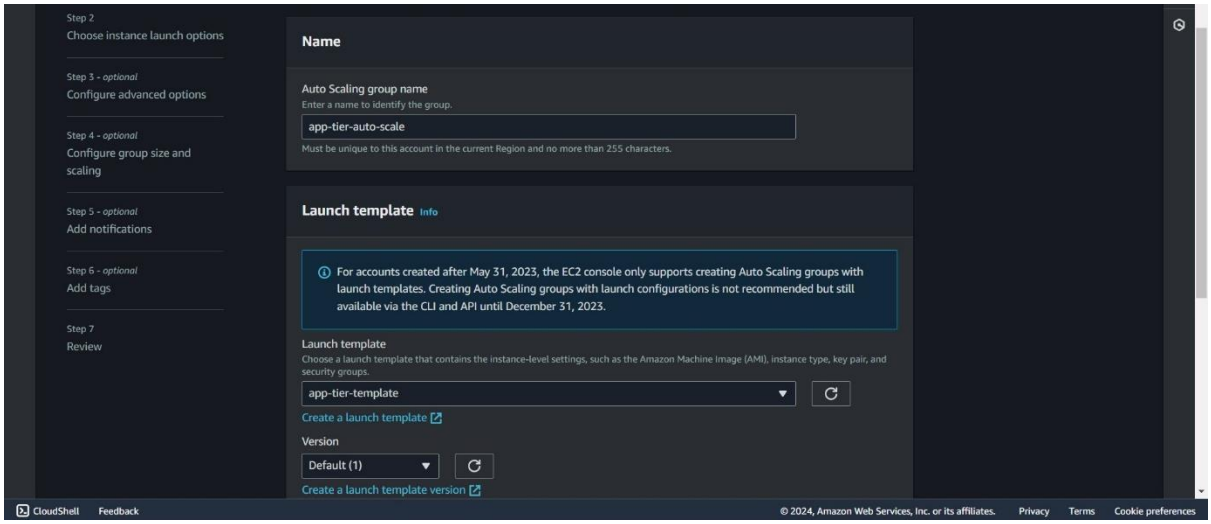


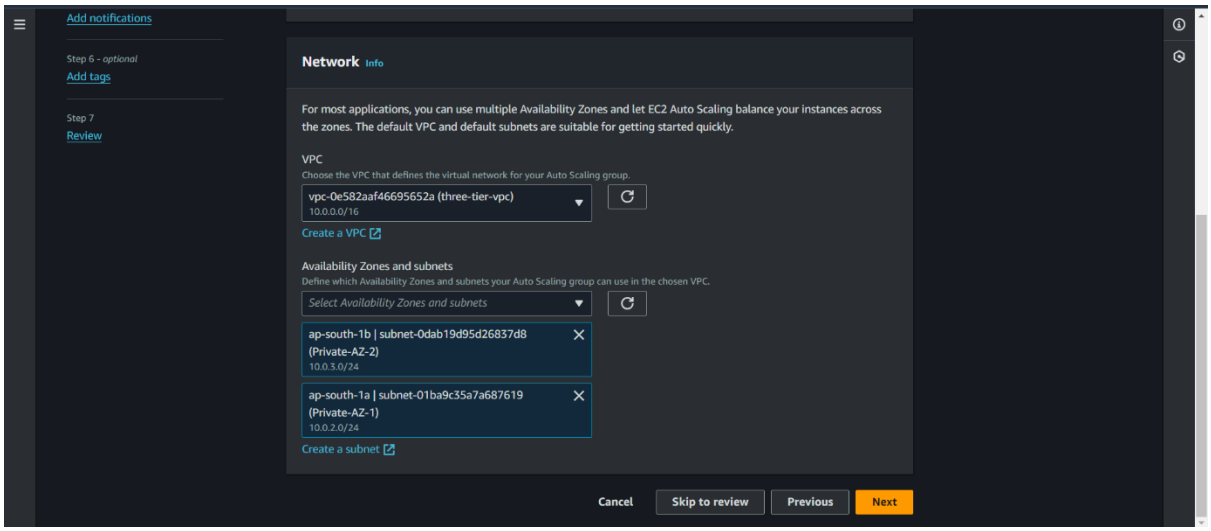



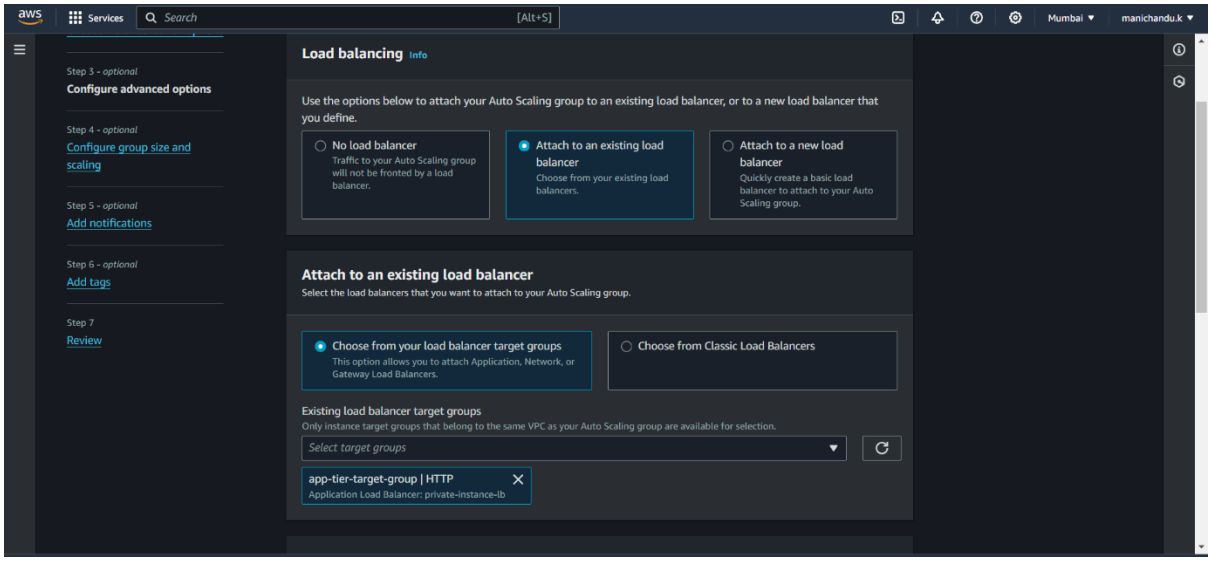

- Select the Target tracking policy, and configure CPU utilization to your preference.
- Skip step 5,6 and go to step 7. Here review your entire process and click on Create Auto scaling group.



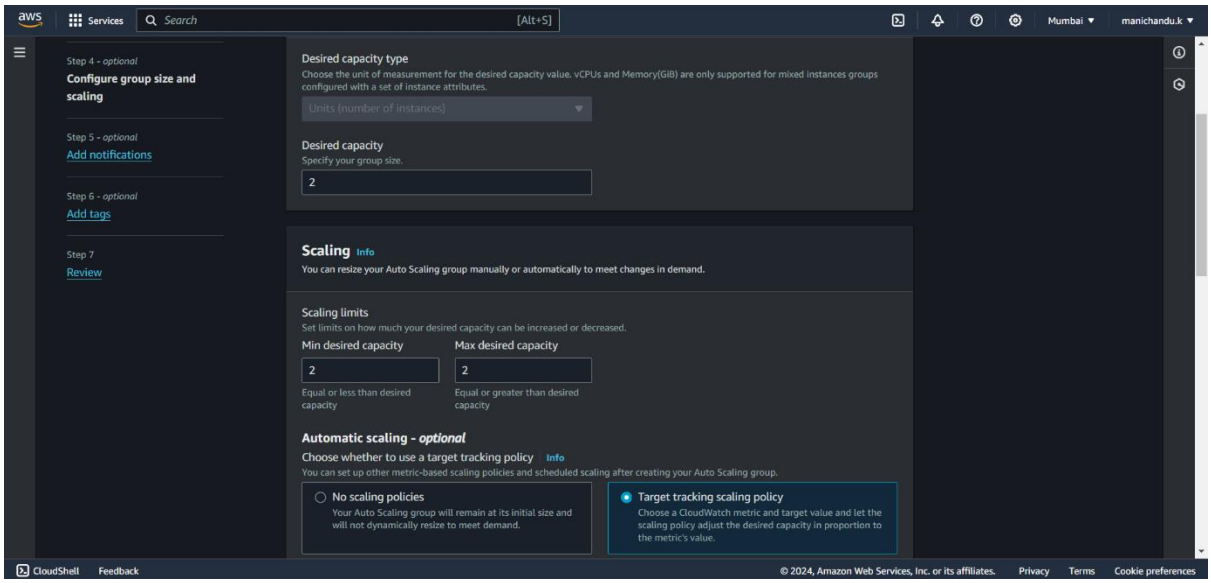

- Now repeat the same process for the app-tier instances.

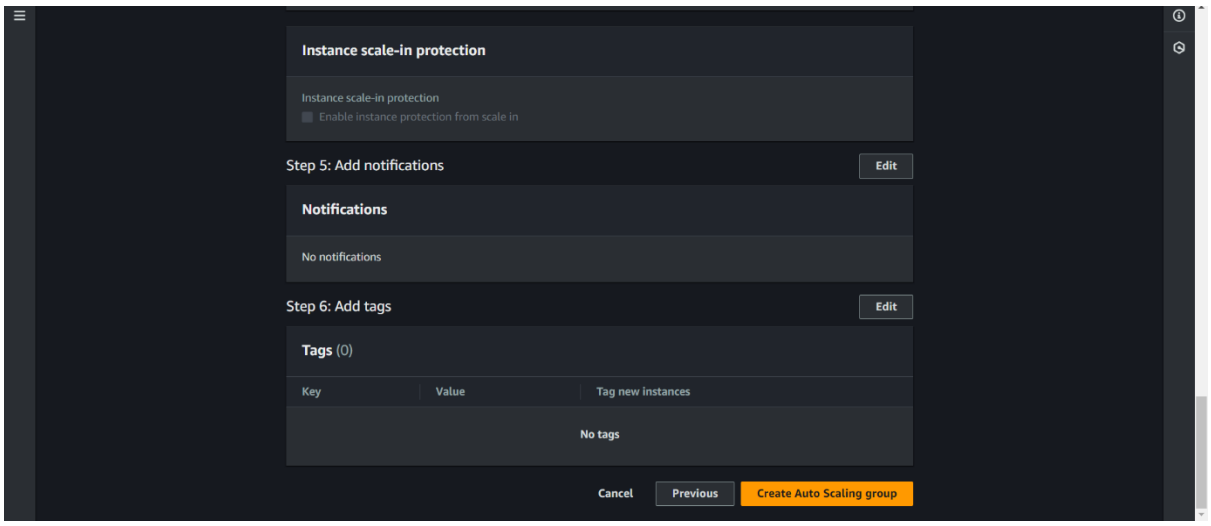


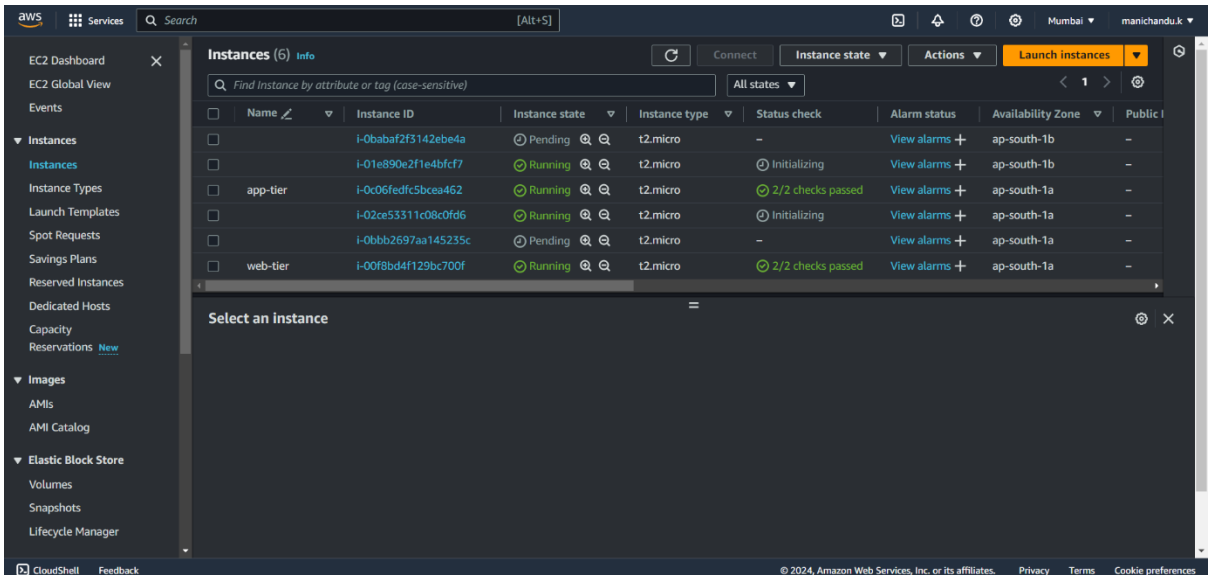



- Make sure that you select the correct subnets, target group and load balancer.



- Click on Create Auto Scaling group, now you should have two auto-scaling groups.
- The instances should look like this.




- We can access our application through the DNS link of our web-tier application load balancer.
- We successfully constructed a three-tier-architecture and successfully deployed the databases.