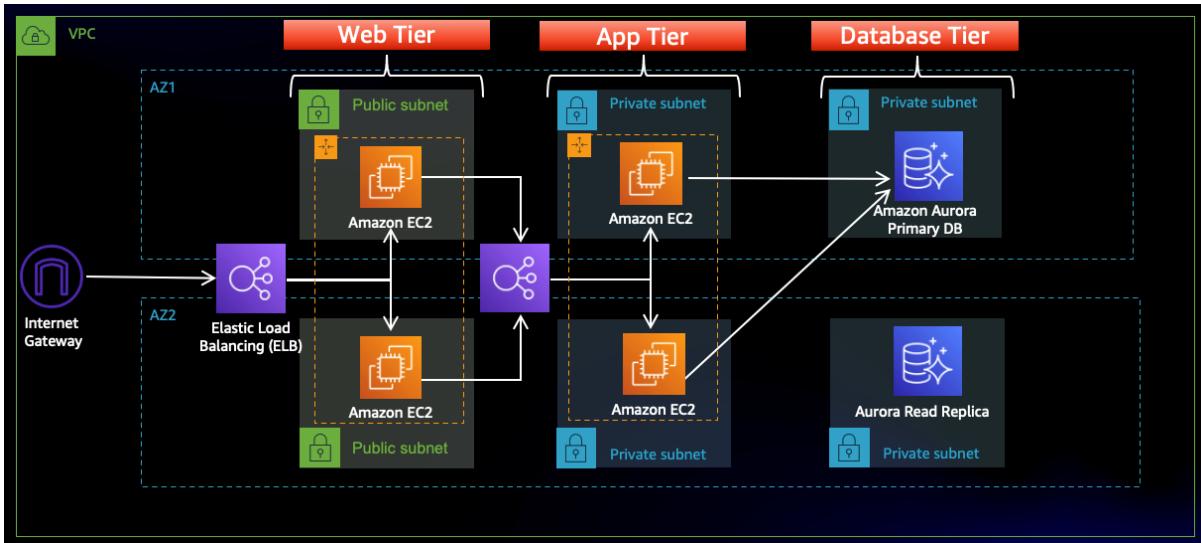


# **DEPLOY A THREE-TIER ARCHITECTURE IN AWS**

**-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.

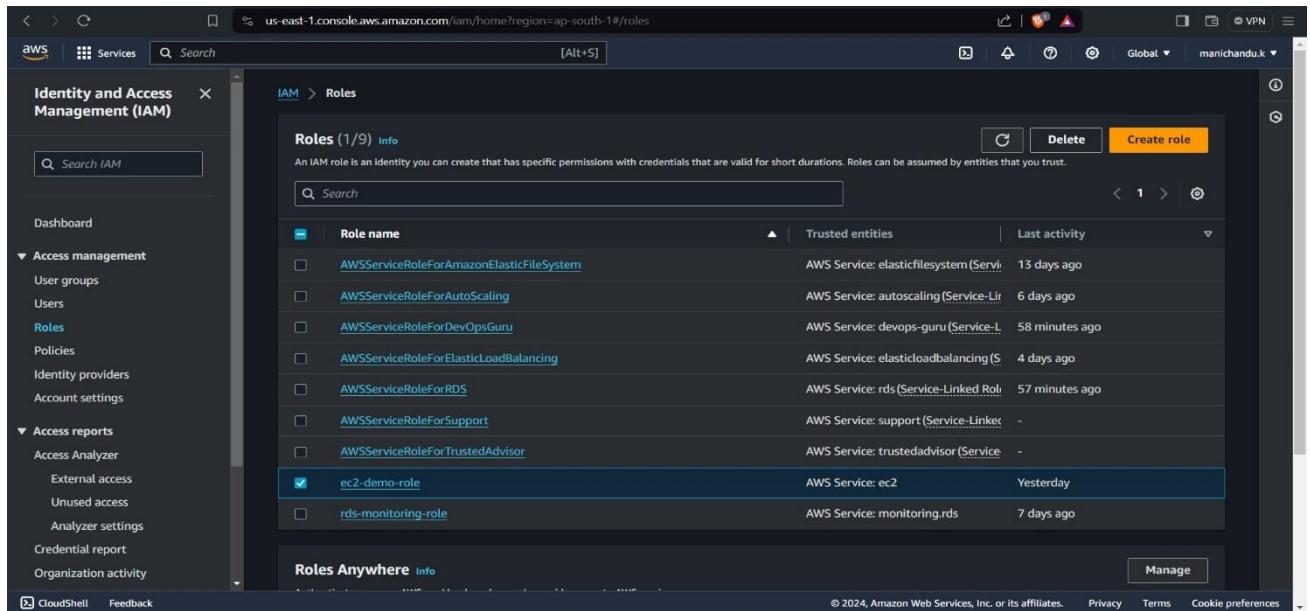
To deploy the above architecture one should know about AWS's basic features

- 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 “AmazonS3ReadOnlyAccess.”
- Click on Create Role.



The screenshot shows the AWS IAM Roles page. The left sidebar is titled "Identity and Access Management (IAM)" and includes sections for Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), and Access reports (Access Analyzer, External access, Unused access, Analyzer settings). The main content area is titled "Roles (1/9) Info" and contains a table of existing roles:

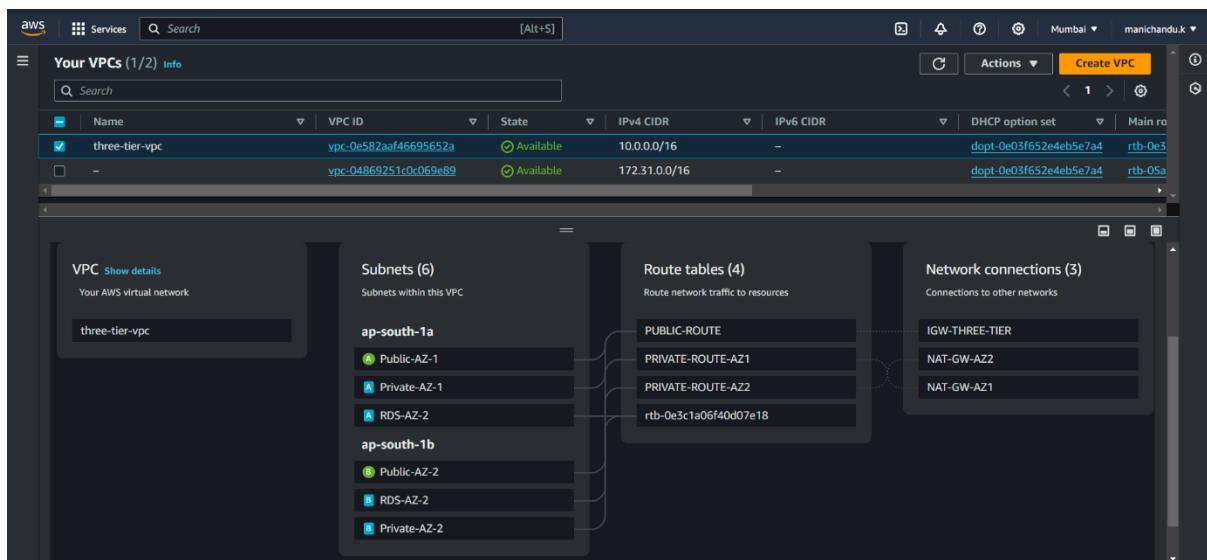
Role name	Trusted entities	Last activity
<a href="#">AWSServiceRoleForAmazonElasticFileSystem</a>	AWS Service: elasticfilesystem (Service-Linked Role)	13 days ago
<a href="#">AWSServiceRoleForAutoScaling</a>	AWS Service: autoscaling (Service-Linked Role)	6 days ago
<a href="#">AWSServiceRoleForDevOpsGuru</a>	AWS Service: devops-guru (Service-Linked Role)	58 minutes ago
<a href="#">AWSServiceRoleForElasticLoadBalancing</a>	AWS Service: elasticloadbalancing (Service-Linked Role)	4 days ago
<a href="#">AWSServiceRoleForRDS</a>	AWS Service: rds (Service-Linked Role)	57 minutes ago
<a href="#">AWSServiceRoleForSupport</a>	AWS Service: support (Service-Linked Role)	-
<a href="#">AWSServiceRoleForTrustedAdvisor</a>	AWS Service: trustedadvisor (Service-Linked Role)	-
<input checked="" type="checkbox"/> <a href="#">ec2-demo-role</a>	AWS Service: ec2	Yesterday
<a href="#">rds-monitoring-role</a>	AWS Service: monitoring.rds	7 days ago

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

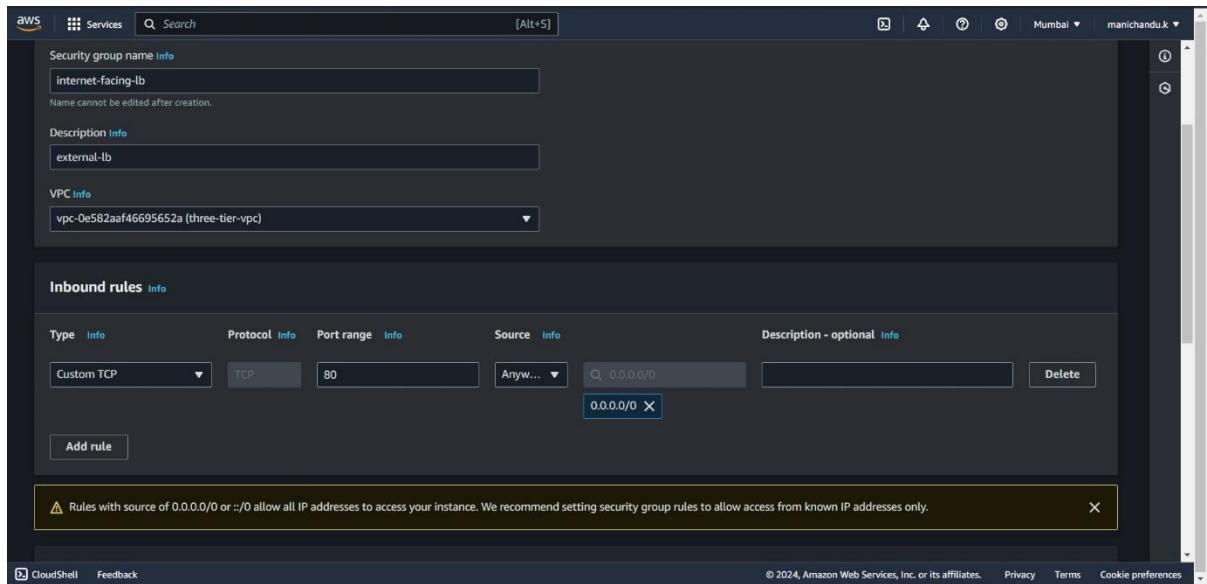
## Networking and security :

- 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.

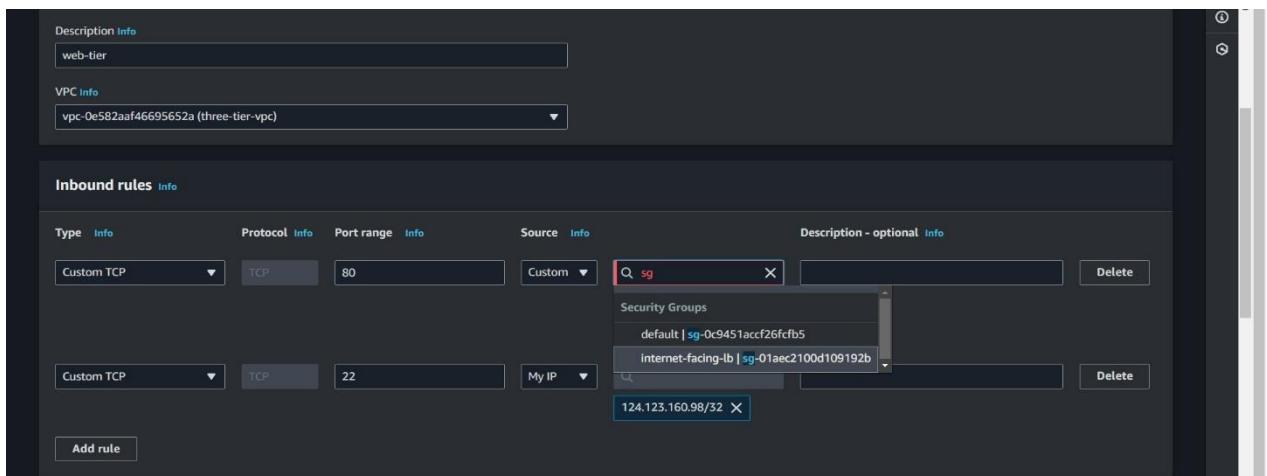
- You can create your VPC with your own CIDR block range and name tag.
- Now create subnets for this VPC. According to the architecture we need 2 public subnets for our web page deployment and 4 private subnets for our app and database deployment.
- Click on Create Subnet and remove the default VPC, 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.



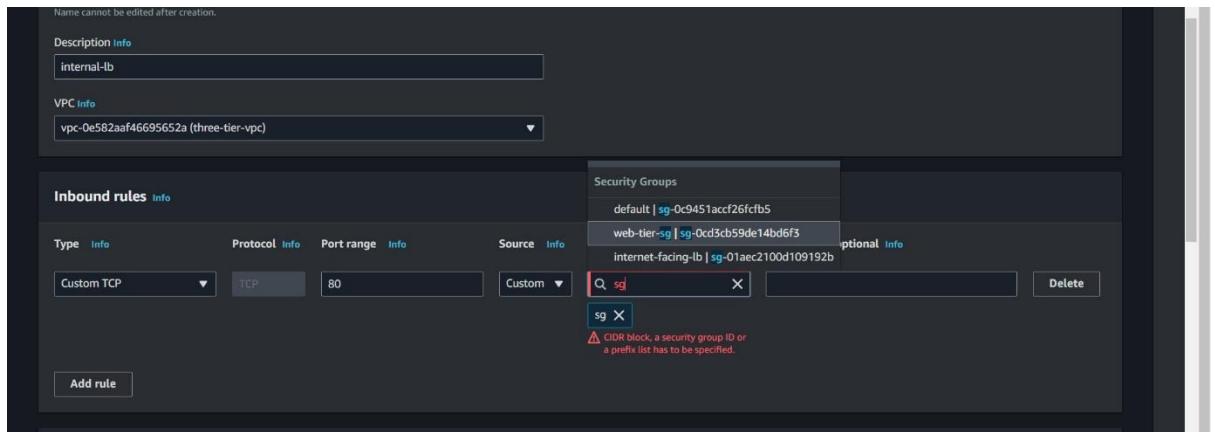
- Make sure to select the manually created VPC every time you create a security group.
- Now create a security group for our web-tier EC2 instances but



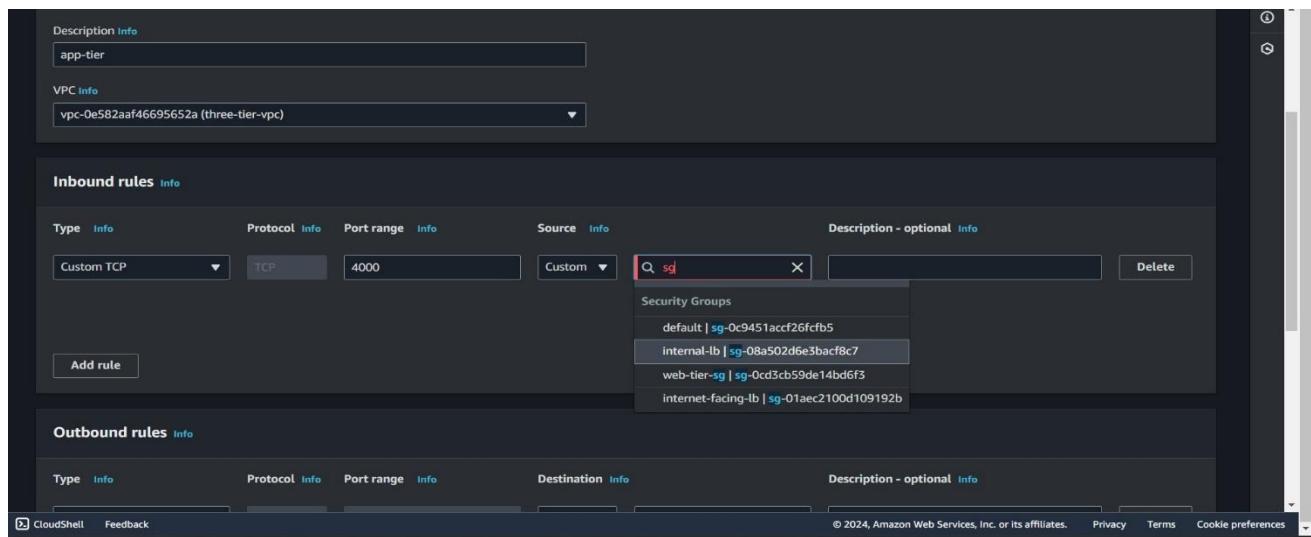
the incoming traffic should only be from the internet-load balancer. So when you create the security group for web-tier

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.

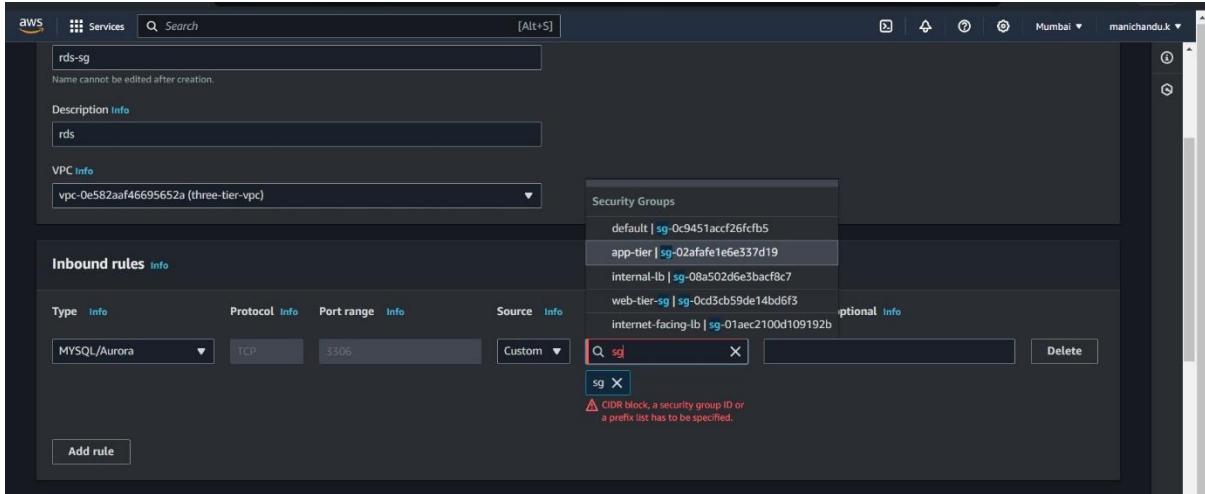


- Now create a security group for app-tier instances where our app is deployed here I gave port number 4000 as most of the apps are written in node.js and it is allowed through port 4000. But the app you deploy may be written in any language so it may be from 3000-4000. So choose the correct port number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

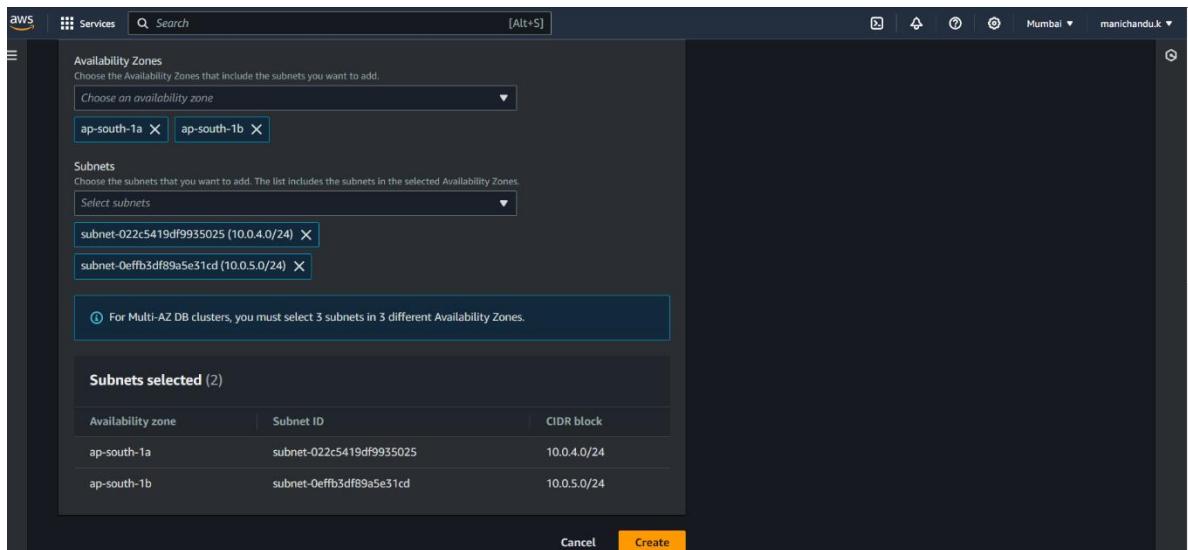
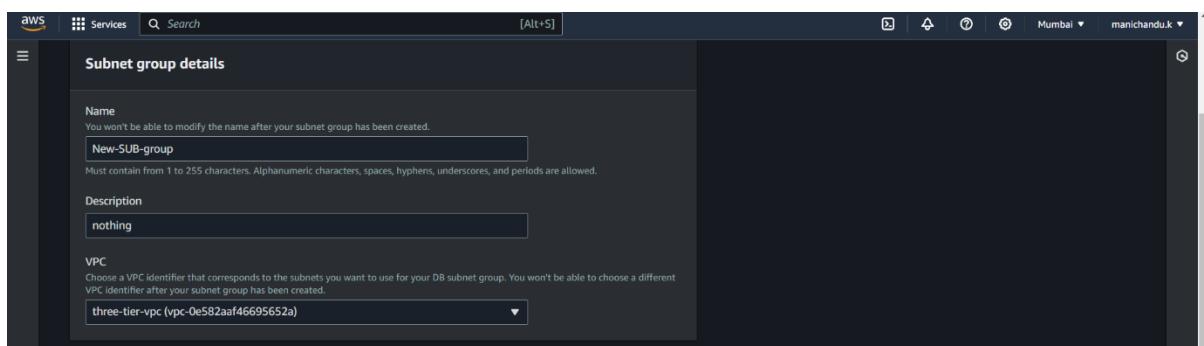
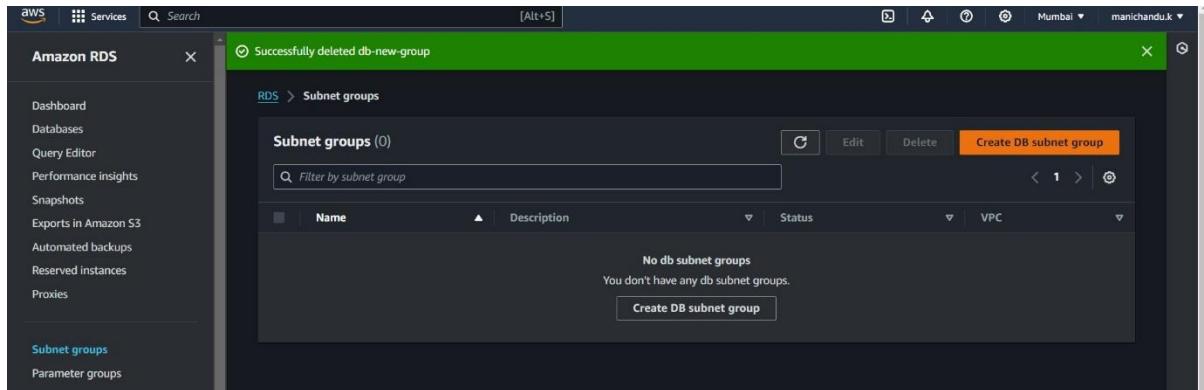
Name	Security group ID	Security group name	VPC ID	Description
-	sg-0c9451accf26fcfb5	default	vpc-0e582aaaf46695652a	default VPC security group
-	sg-08a502d6e3b3ac8c7	internal-lb	vpc-0e582aaaf46695652a	internal-lb
-	sg-02afafe1e6e337d19	app-tier	vpc-0e582aaaf46695652a	app-tier
-	sg-0d7890b301c5cb29a	rds-sg	vpc-0e582aaaf46695652a	rds
-	sg-0b379a0fbba2a9443	default	vpc-04869251c0c069e89	default VPC security group
-	sg-0cd3cb59de14bd6f3	web-tier-sg	vpc-0e582aaaf46695652a	web-tier
-	sg-01aec2100d109192b	internet-facing-lb	vpc-0e582aaaf46695652a	external-lb

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

## Database Configuration :

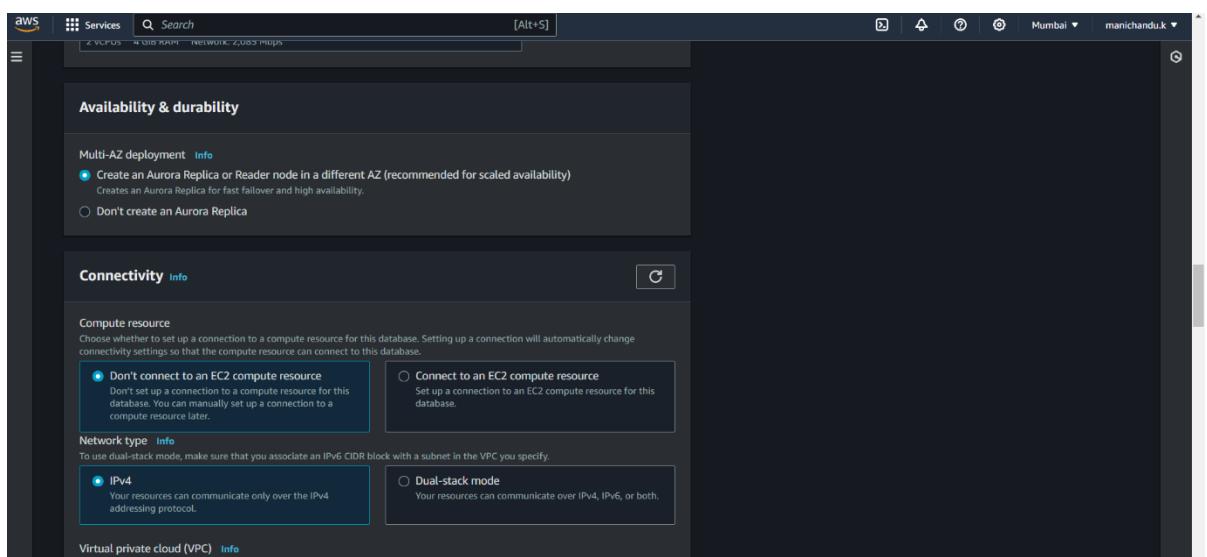
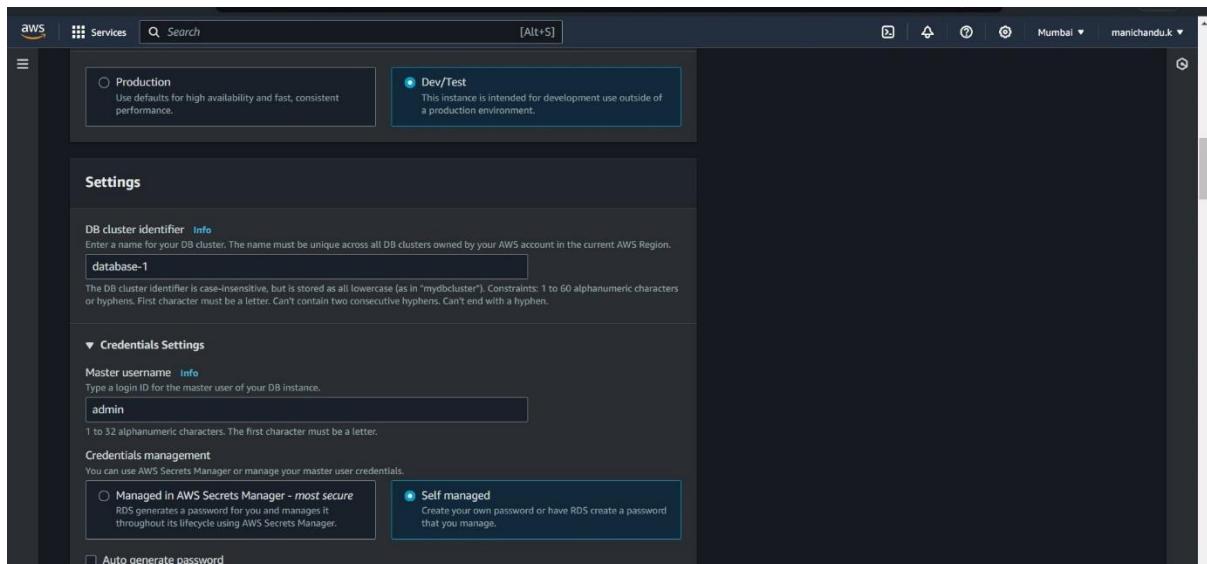
- In this section we will configure database subnet groups and multi-AZ databases.
- Navigate to the RDS dashboard, select the subnet groups section, click on Create DB subnet groups, and select the VPC you will create the subnet group and select the available zone, in our case

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

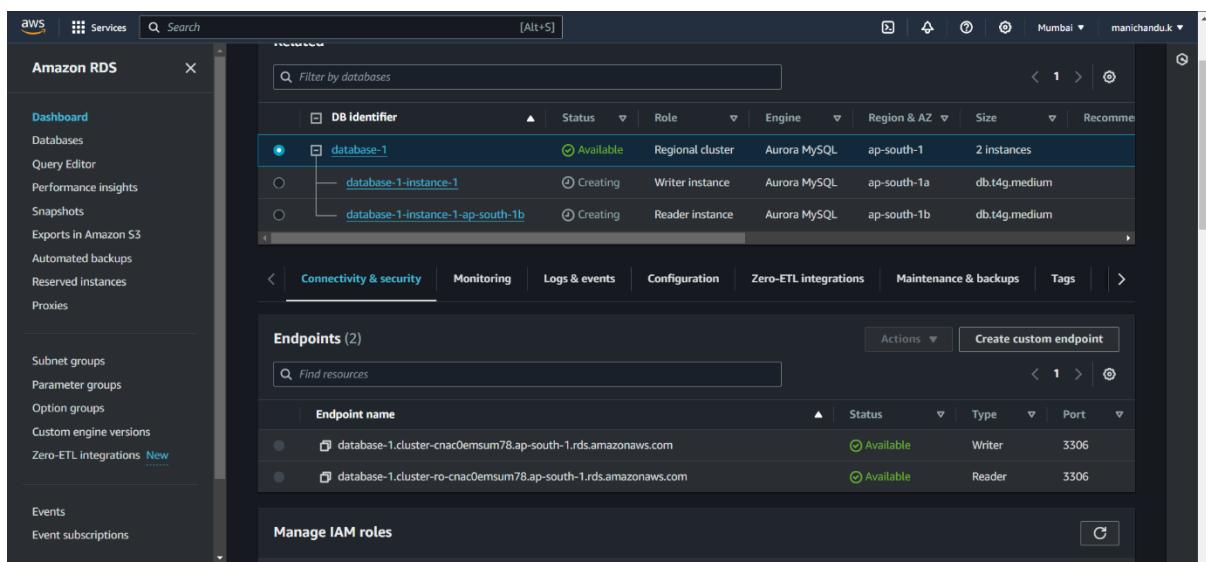
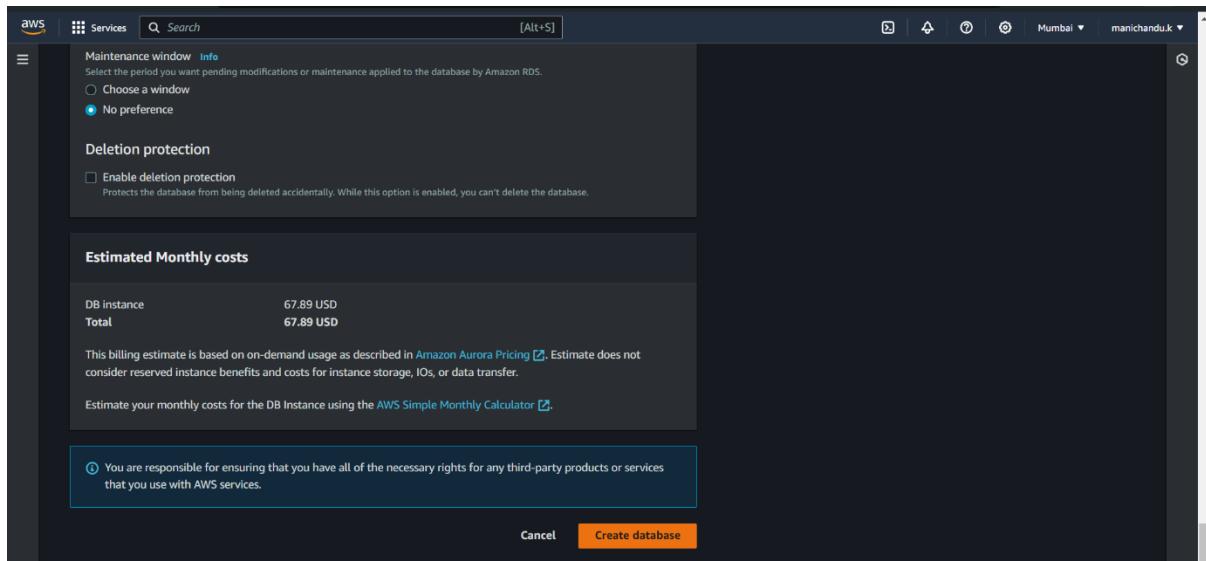


- Click on Create and your DB subnet will be created.
- Now go to the database section and create a database with any name of your choice and select the Standard create option, select the aurora(MySQL) database as we have given its port number in our security group.

- 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.

The screenshots show the AWS Lambda console interface during the creation of a new function:

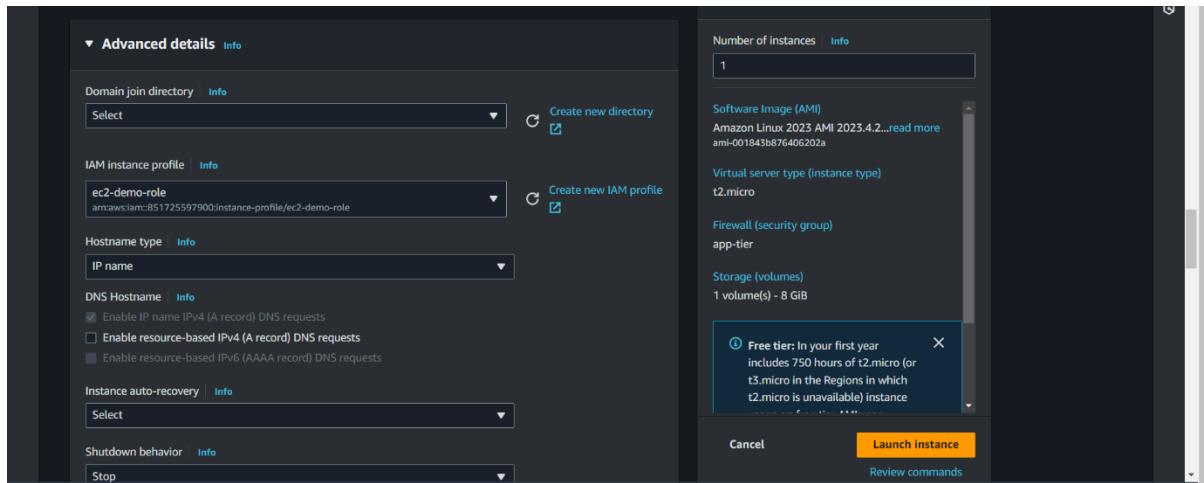
**Top Screenshot (Function code tab):**

- Code Editor: Contains the following code:

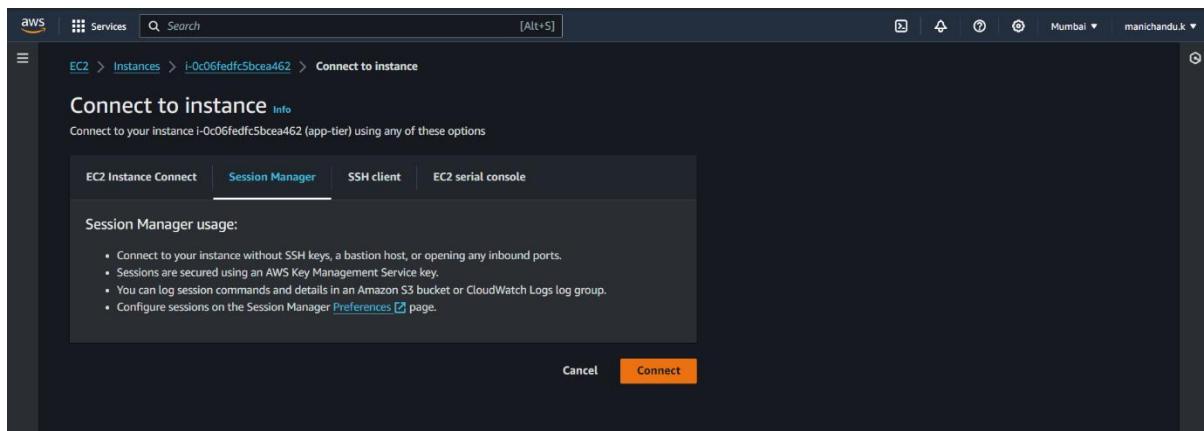
```
function handler(event, context) {
  context.succeed("Hello, world!");
}
```

**Bottom Screenshot (Configuration tab):**

- Handler: Set to `index.handler`.



- 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.

```
Session ID: root-0004594e02c8c1ee9
Instance ID: i-0c06fedfc5bcea462
Terminate
```

```
sh-5.2$ sudo su ec2-user
[ec2-user@ip-10-0-2-228 bin]$ pwd
/usr/bin
[ec2-user@ip-10-0-2-228 bin]$ sudo yum update-y
No such command: update-y. Please use /usr/bin/yum --help
It could be a YUM plugin command, try: "yum install 'dnf-command(update-y)'"
[ec2-user@ip-10-0-2-228 bin]$ sudo yum update -y
Last metadata expiration check: 0:01:13 ago on Fri Apr 26 08:09:25 2024.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-10-0-2-228 bin]$
```

- 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
```

```
Session ID: root-0004594e02c8c1ee9 Instance ID: i-0c06fedfc5bcea462 Terminate
[ec2-user@ip-10-0-2-228 bin]$ sudo su ec2-user
[ec2-user@ip-10-0-2-228 bin]$ pwd
/usr/bin
[ec2-user@ip-10-0-2-228 bin]$ sudo yum update -y
No such command: update-y. Please use /usr/bin/yum --help
It could be a YUM plugin command, try: "yum install 'dnf-command(update-y)'"
[ec2-user@ip-10-0-2-228 bin]$ sudo yum update -y
Last metadata expiration check: 0:01:13 ago on Fri Apr 26 08:09:25 2024.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-10-0-2-228 bin]$ sudo wget https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm
--2024-04-26 08:11:10-- https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm
Resolving dev.mysql.com (dev.mysql.com)... 23.47.234.151, 2600:140f:1e00:4b2::2e31, 2600:140f:1e00:4b2::2e31
Connecting to dev.mysql.com (dev.mysql.com)|23.47.234.151|:443... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://repo.mysql.com/mysql57-community-release-el7-11.noarch.rpm (following)
--2024-04-26 08:11:10-- https://repo.mysql.com/mysql57-community-release-el7-11.noarch.rpm
Resolving repo.mysql.com (repo.mysql.com)... 23.205.218.66, 2600:140f:4:da7::1d68, 2600:140f:4:da4::1d68
Connecting to repo.mysql.com (repo.mysql.com)|23.205.218.66|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 25680 (25K) [application/x-redhat-package-manager]
Saving to: 'mysql57-community-release-el7-11.noarch.rpm'

mysql57-community-release-el7-11.noarch.rpm 100%[=====] 25.08K --.-KB/s in 0.02s

2024-04-26 08:11:10 (1.50 MB/s) - 'mysql57-community-release-el7-11.noarch.rpm' saved [25680/25680]
```

```
Session ID: root-0004594e02c8c1ee9 Instance ID: i-0c06fedfc5bcea462 Terminate
[ec2-user@ip-10-0-2-228 bin]$ sudo yum install https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm
Last metadata expiration check: 0:03:25 ago on Fri Apr 26 08:09:25 2024.
mysql57-community-release-el7-11.noarch.rpm
Dependencies resolved.
=====
      Package           Architecture   Version       Repository      Size
=====
Installing: mysql57-community-release               noarch        el7-11        @commandline    25 k
=====
Transaction Summary
=====
Install 1 Package

Total size: 25 k
Installed size: 31 k
Is this ok [y/N]: y
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing          : 1/1
  Installing        : mysql57-community-release-el7-11.noarch 1/1
  Verifying         : mysql57-community-release-el7-11.noarch 1/1
Installed:
  mysql57-community-release-el7-11.noarch
Complete!
```

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

```
mysql -h database_writer_instance_endpoint -u username -p
```

Session ID: root-0004594e02c8c1ee9      Instance ID: i-0c06fedfc5bcea462      Terminate

```
[ec2-user@ip-10-0-2-228 bin]$ sudo yum install mysql
MySQL Connectors Community
MySQL Tools Community
MySQL 5.7 Community Server
Dependencies resolved.
=====
Package           Architecture      Version       Repository      Size
=====
Installing:
mysql-community-client      x86_64          5.7.44-1.el7   mysql57-community    31 M
Installing dependencies:
mysql-community-common       x86_64          5.7.44-1.el7   mysql57-community    313 k
mysql-community-libs         x86_64          5.7.44-1.el7   mysql57-community   3.0 M
ncurses-compat-libs          x86_64          6.2-4.20200222.amzn2023.0.6   amazonlinux           323 k
=====
Transaction Summary
Install 4 Packages

Total download size: 35 M
Installed size: 135 M
Is this ok [y/N]: y
Downloading Packages:
(1/4) : ncurses-compat-libs-6.2-4.20200222.amzn2023.0.6.x86_64.rpm      3.3 MB/s | 323 kB  00:00
(2/4) : mysql-community-common-5.7.44-1.el7.x86_64.rpm                  1.6 MB/s | 313 kB  00:00
(3/4) : mysql-community-libs-5.7.44-1.el7.x86_64.rpm                  14 MB/s | 3.0 MB  00:00
(4/4) : mysql-community-client-5.7.44-1.el7.x86_64.rpm                 35 MB/s | 31 MB   00:00
=====
Total                                         37 MB/s | 35 MB  00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  preparing : 
```

1/1

Amazon RDS X

**Databases**

- database-1
  - database-1-instance-1 (Available) Writer instance Aurora MySQL ap-south-1 2 instances
  - database-1-instance-1-ap-south-1b (Creating) Reader instance Aurora MySQL ap-south-1b db.t4g.medium

**Connectivity & security**

Endpoint & port	Networking	Security
Endpoint copied database-1-instance-1.cnac0emsu78.ap-south-1.rds.amazonaws.com Port 3306	Availability Zone ap-south-1a VPC three-tier-vpc (vpc-0e582aaaf46695652a) Subnet group new-sub-group Subnets subnet-022c5419df9935025 subnet-0effb3df89a5e31cd	VPC security groups rds-sg (sg-0d7890b301c5cb29a) Active Publicly accessible No Certificate authority Info rds-ca-rsa2048-g1 Certificate authority date May 20, 2061, 00:10 (UTC+05:30) DB instance certificate expiration

Session ID: root-0004594e02c8c1ee9      Instance ID: i-0c06fedfc5bcea462      Terminate

```
[ec2-user@ip-10-0-2-228 bin]$ mysql -h database-1-instance-1.cnac0emsu78.ap-south-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 80
Server version: 8.0.28 Source distribution

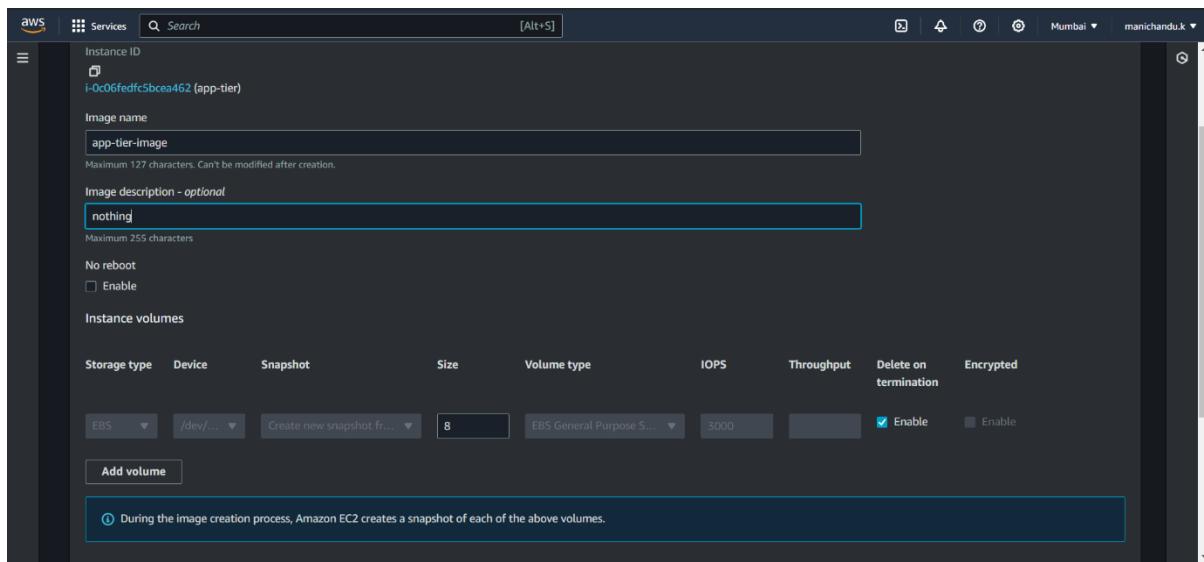
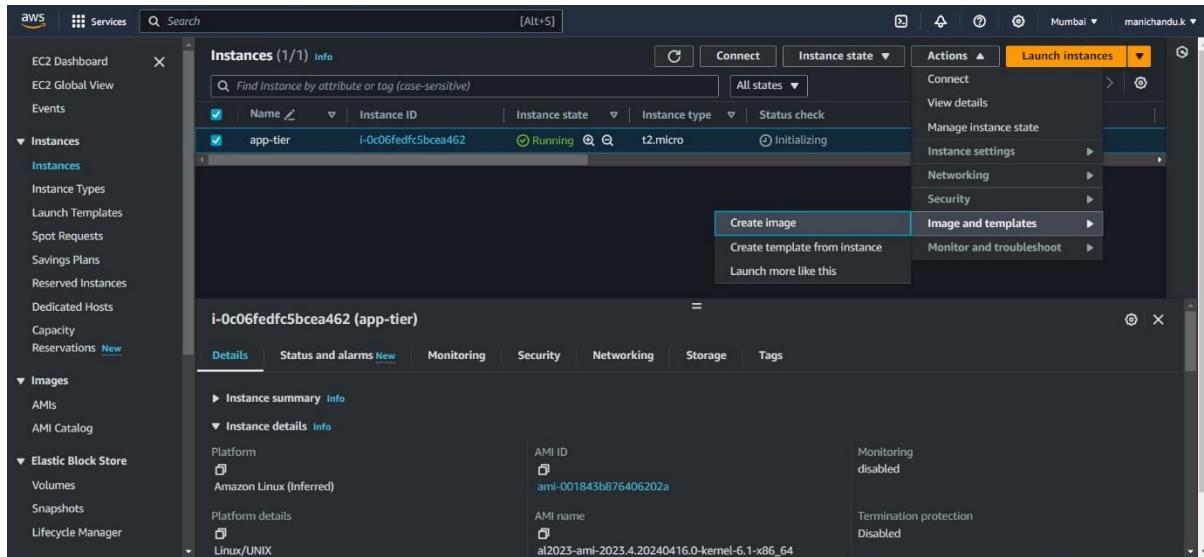
Copyright (c) 2000, 2023, Oracle and/or its affiliates.

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>
```

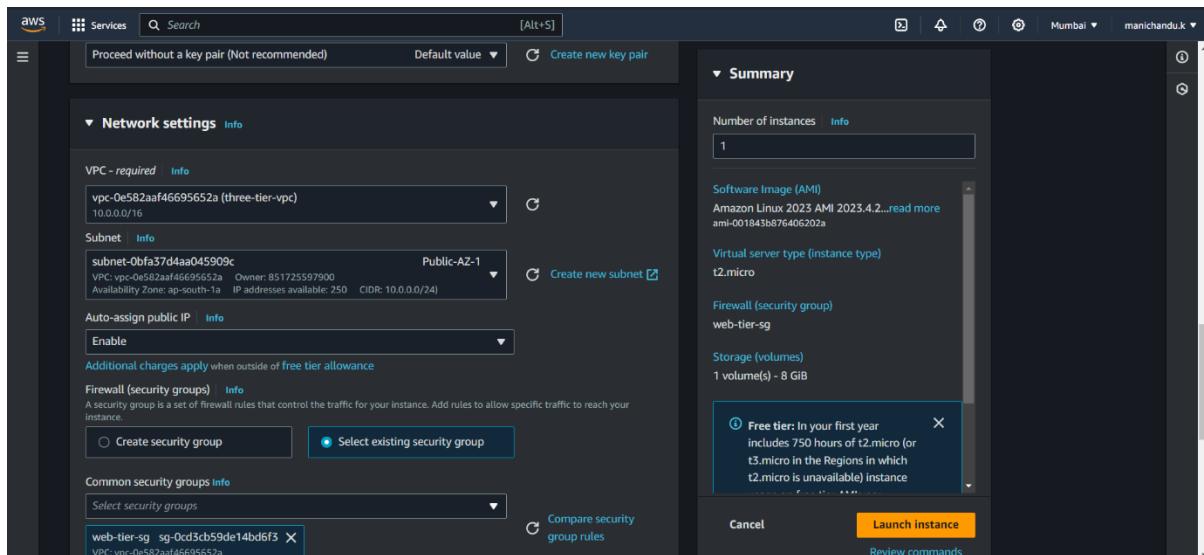
- You will be connected to the database and you can create databases, and tables through CLI in this database.
- After creating databases, if you have any application code, upload app tier configuration files or node.js files to your server using s3 bucket through simple CLI commands.
- Now log out of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



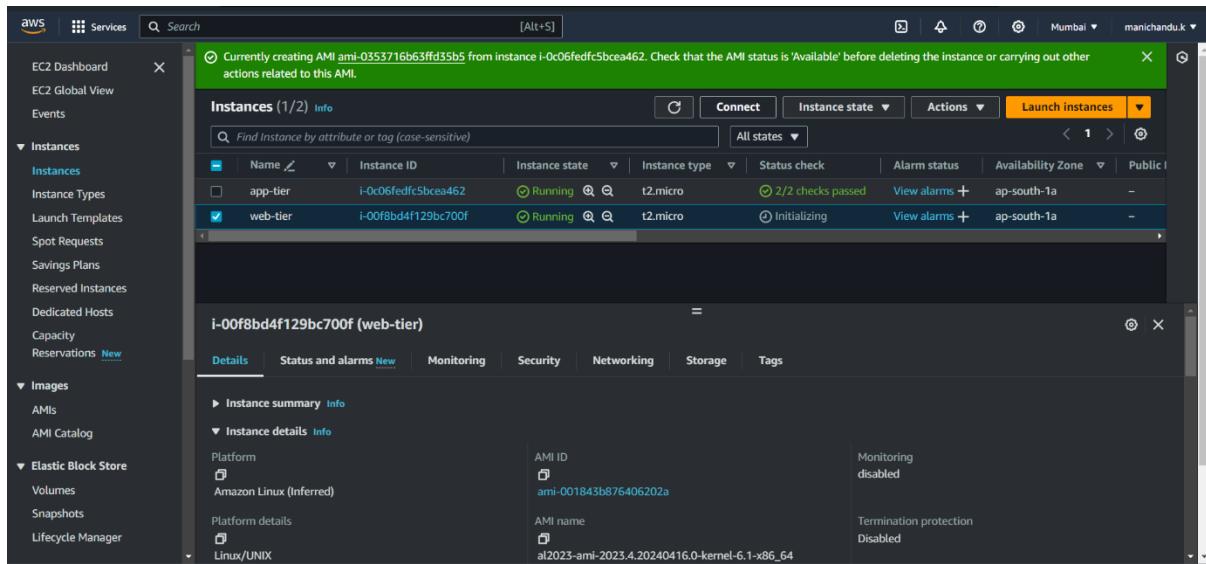
- Leave the image aside and launch another instance for our webpage. The webpage instance should have the web tier configuration files which is a user interface.

## 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.

Session ID: root-099c6cd05ac39bd6c      Instance ID: i-00f8bd4f129bc700f      Terminate

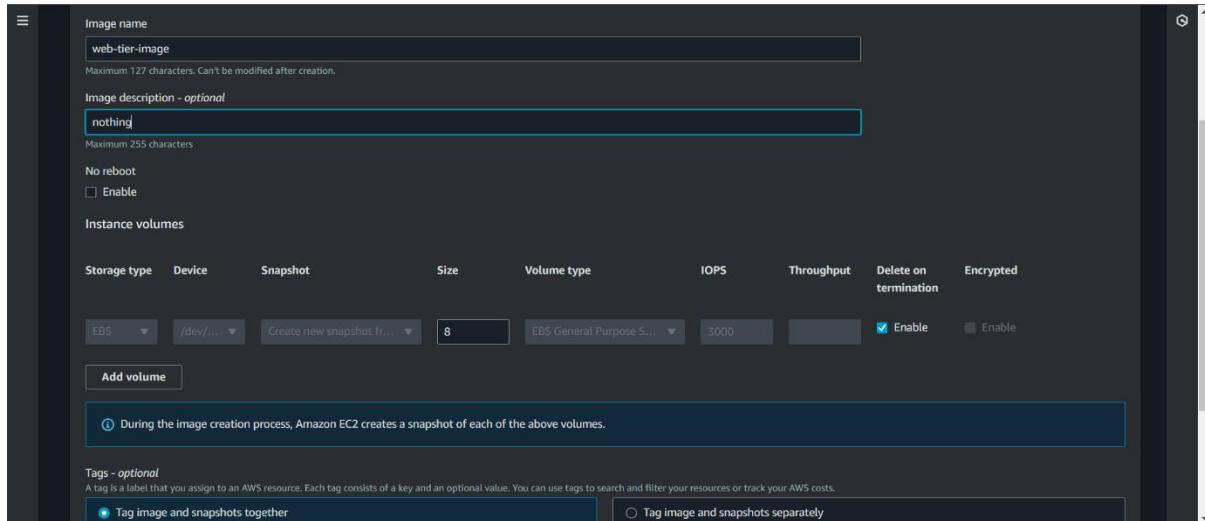
```

Total
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing :
Running scriptlet:
Installing : nginx-fs-1:1.24.0-1.amzn2023.0.2.noarch
Installing : nginx-fs-1:1.24.0-1.amzn2023.0.2.noarch
Installing : nginx-mime-2:1.49-3.amzn2023.0.3.noarch
Installing : libunwind-1:4.0-5.amzn2023.0.2.x86_64
Installing : gperftools-libs-2.9.1-1.amzn2023.0.3.x86_64
Installing : nginx-core-1:1.24.0-1.amzn2023.0.2.x86_64
Installing : generic-logos-https-18.0.0-12.amzn2023.0.3.noarch
Installing : nginx-1:1.24.0-1.amzn2023.0.2.x86_64
Running scriptlet: nginx-1:1.24.0-1.amzn2023.0.2.x86_64
Verifying : generic-logos-https-18.0.0-12.amzn2023.0.3.noarch
Verifying : gperftools-libs-2.9.1-1.amzn2023.0.3.x86_64
Verifying : libunwind-1:4.0-5.amzn2023.0.2.x86_64
Verifying : nginx-1:1.24.0-1.amzn2023.0.2.x86_64
Verifying : nginx-core-1:1.24.0-1.amzn2023.0.2.x86_64
Verifying : nginx-fs-1:1.24.0-1.amzn2023.0.2.noarch
Verifying : nginx-mime-2:1.49-3.amzn2023.0.3.noarch

Installed:
generic-logos-https-18.0.0-12.amzn2023.0.3.noarch      gperftools-libs-2.9.1-1.amzn2023.0.3.x86_64      libunwind-1:4.0-5.amzn2023.0.2.x86_64
nginx-1:1.24.0-1.amzn2023.0.2.x86_64                 nginx-core-1:1.24.0-1.amzn2023.0.2.x86_64      nginx-fs-1:1.24.0-1.amzn2023.0.2.noarch
nginx-mime-2:1.49-3.amzn2023.0.3.noarch

Complete!
[ec2-user@ip-10-0-0-57 bin]$ cd /usr/share/nginx/html
[ec2-user@ip-10-0-0-57 html]$ sudo rm index.html
[ec2-user@ip-10-0-0-57 html]$ sudo vi index.html
[ec2-user@ip-10-0-0-57 html]$ sudo systemctl restart nginx
[ec2-user@ip-10-0-0-57 html]$
```

- 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.

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)**

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below.

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

Recents My AMIs Quick Start

Owned by me

Shared with me

Browse more AMIs

Amazon Machine Image (AMI)

web-tier-image  
ami-0d074fb5d8f28718  
2024-04-26T08:24:52.000Z Virtualization: hvm ENA enabled: true Root device type: ebs

Description nothing

Software Image (AMI)  
nothing  
ami-0d074fb5d8f28718

Virtual server type (instance type)  
-

Firewall (security group)  
-

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

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of

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)**

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below.

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

Recents My AMIs Quick Start

Owned by me

Shared with me

Browse more AMIs

Amazon Machine Image (AMI)

app-tier-image  
ami-0553716b63ff635b5  
2024-04-26T08:20:06.000Z Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Software Image (AMI)  
nothing  
ami-0553716b63ff635b5

Virtual server type (instance type)  
-

Firewall (security group)  
-

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

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of

Create launch template

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

EC2 Dashboard    X

Launch Templates (2) Info

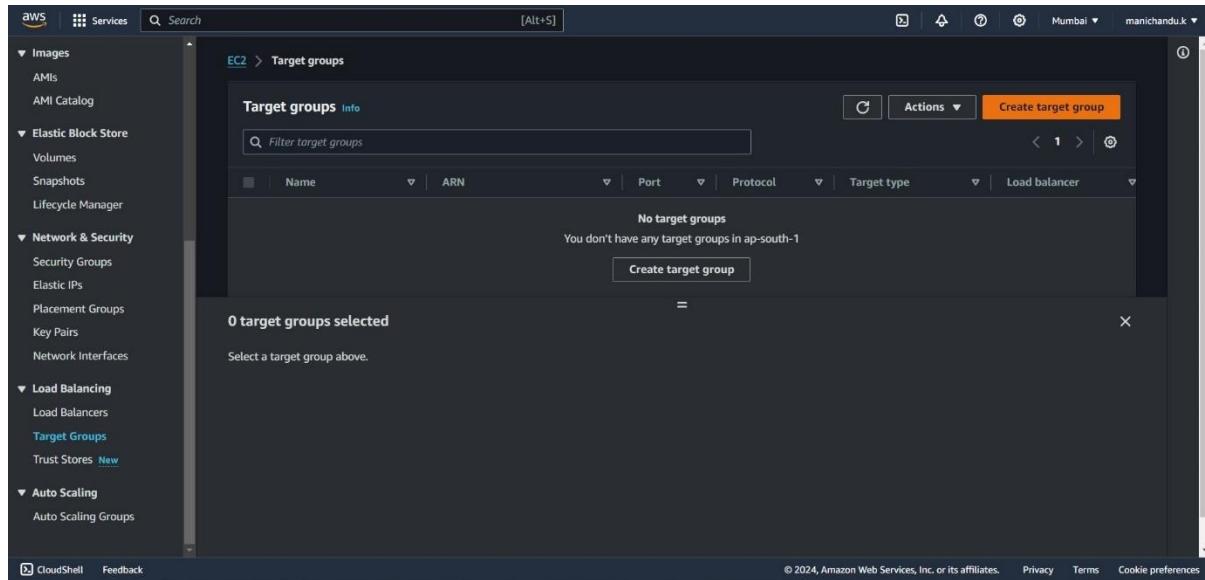
Actions Create launch template

Launch Template ID	Launch Template Name	Default Version	Latest Version	Create Time	Created By
lt-042fc420ac6f91417	web-tier-template	1	1	2024-04-26T08:27:07.000Z	arn:aws:iam::...
lt-002abda27d3252e79	app-tier-template	1	1	2024-04-26T08:28:28.000Z	arn:aws:iam::...

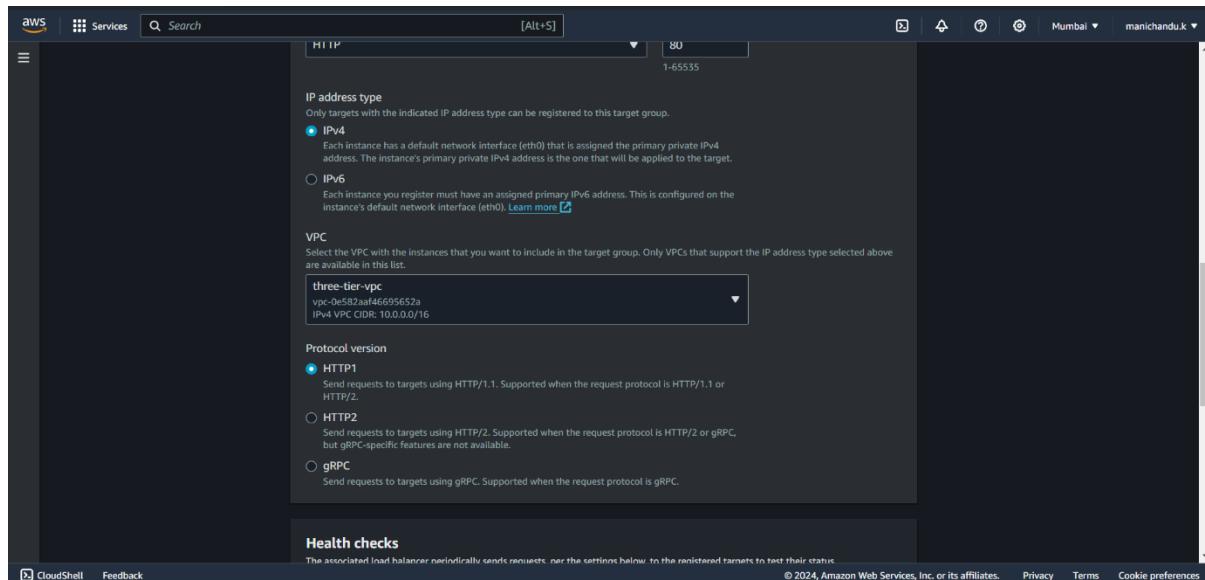
Select a launch template

# Load balancing and Auto scaling :

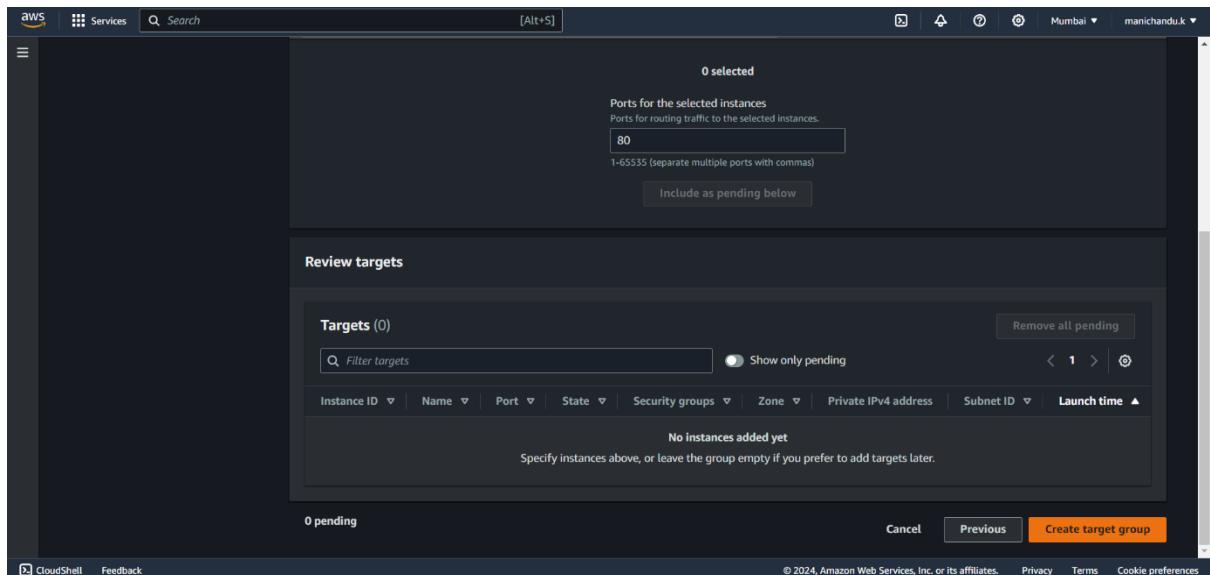
- After creating the launch templates, go to the Load balancing feature of EC2 and click on target groups to create a target group for our load balancer yet to be created.



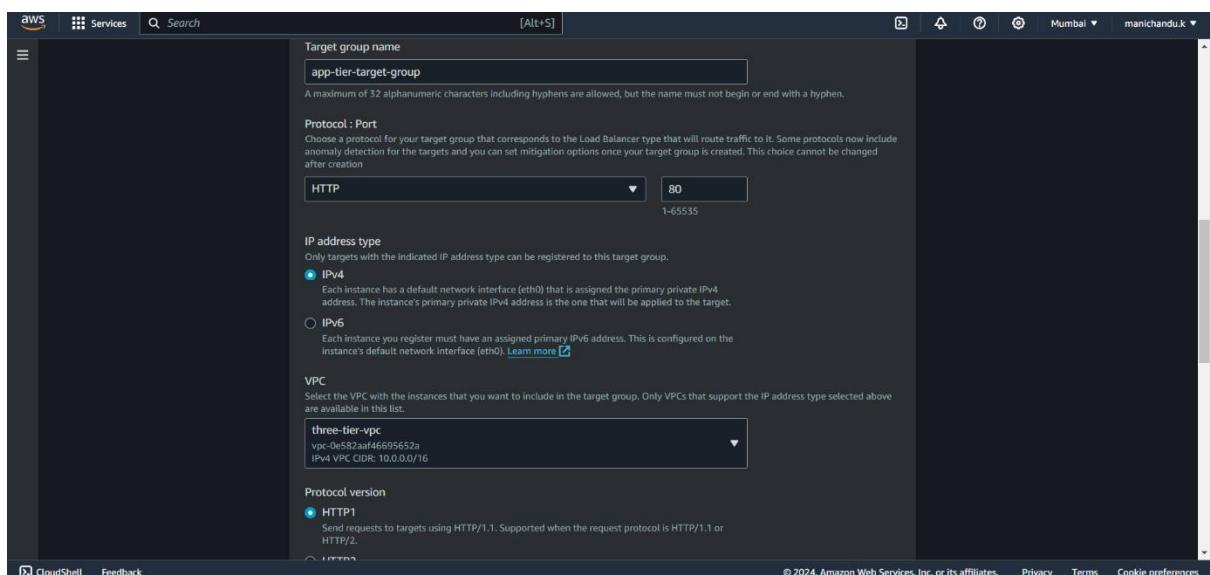
- Click on create target group and give a name to our target group, here I created a target group for my web page instances, so give a name according to your preferences.
- Select our manually created VPC when choosing VPC and give the port number as 80.



- 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.



The screenshot shows the AWS EC2 Target groups page. The left sidebar includes options like EC2 Dashboard, EC2 Global View, Events, Instances, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Reservations, CloudShell, and Feedback. The main content area displays 'Target groups (2) Info' with a table:

Name	ARN	Port	Protocol	Target type	Load balancer
app-tier-target-group	arn:aws:elasticloadbalancing:...:80	80	HTTP	Instance	None associated
web-tier-target	arn:aws:elasticloadbalancing:...:80	80	HTTP	Instance	None associated

Below the table, it says '0 target groups selected' and 'Select a target group above.'

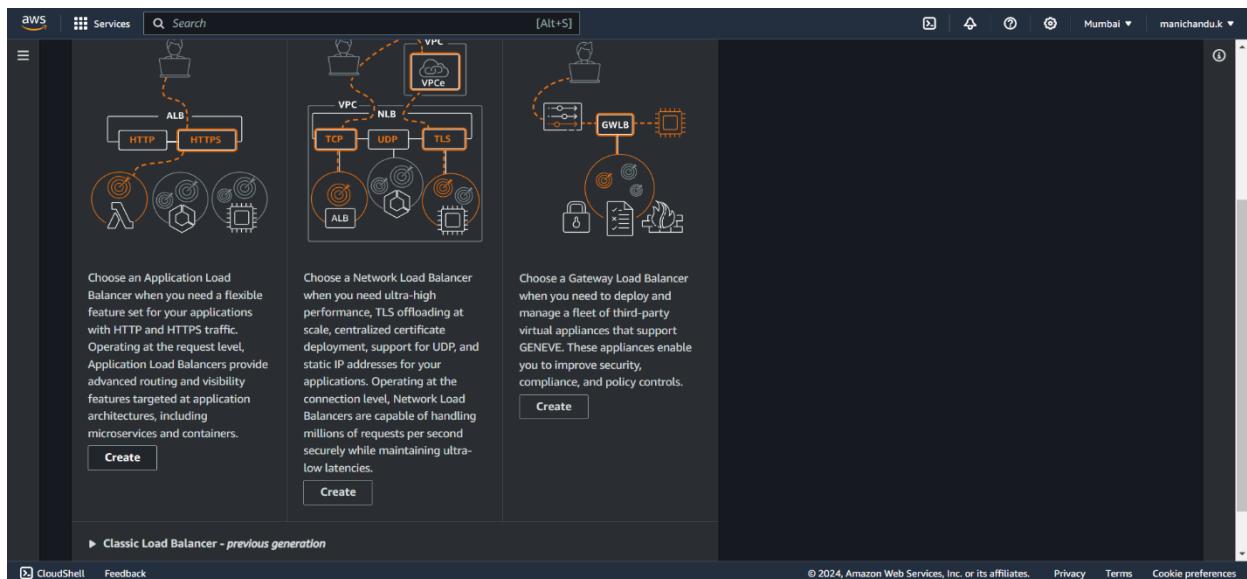
- Now go to the load balancers option and create two load balancers for both instances using target groups.

The screenshot shows the AWS EC2 Load balancers page. The left sidebar includes options like Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network interfaces, and Load Balancing. The main content area displays a message about the resource map for Network Load Balancers, followed by a table:

Load balancers
0 load balancers selected

Below the table, it says 'Select a load balancer above.'

- 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.

This screenshot shows the first step of the AWS Create Application Load Balancer wizard. The form includes the following fields:

- Load balancer name:** A text input field containing "internet-facing-lb".
- Scheme:** A radio button group where "Internet-facing" is selected. A tooltip explains that it routes requests from clients over the internet to targets.
- IP address type:** A radio button group where "IPv4" is selected. A tooltip indicates it includes only IPv4 addresses.
- Network mapping:** A section describing how the load balancer routes traffic to targets in selected subnets.
- VPC:** A dropdown menu showing a single VPC entry: "vpc-04860251c0c069e89".

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

The screenshot shows the AWS VPC configuration interface. Under the 'Mappings' section, two subnets are selected: 'ap-south-1a (aps1-az1)' and 'ap-south-1b (aps1-az3)'. Both subnets are assigned by AWS and are located in Public-AZ-1 and Public-AZ-2 respectively.

- 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.

The screenshot shows the AWS Security Groups configuration interface. A single security group, 'internet-facing-lb', is selected from a dropdown menu. Below this, under the 'Listeners and routing' section, a listener for port 80 is configured to forward traffic to a target group named 'web-tier-target'.

- 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.

**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.  
**private-instance-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**  
An internet-facing load balancer routes requests from clients over the internet to targets. Requires a public subnet. [Learn more](#)  
 **Internal**  
An internal load balancer routes requests from clients to targets using private IP addresses.

**IP address type** [Info](#)  
Select the type of IP addresses that your subnets use.  
 **IPv4**  
Includes only IPv4 addresses.  
 **Dualstack**  
Includes IPv4 and IPv6 addresses.

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences  
40°C Sunny Search ENG IN 14:02 26-04-2024 manichandu.k

**VPC** [Info](#)  
Select the virtual private cloud (VPC) for your targets or you can [create a new VPC](#). The selected VPC can't be changed after the load balancer is created. To confirm the VPC for your targets, view your [target groups](#).  
**three-tier-vpc**  
vpc-0e582aa4695652a  
IPv4 VPC CIDR: 10.0.0.0/16

**Mappings** [Info](#)  
Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.  
 **ap-south-1a (aps1-az1)**  
Subnet: **subnet-01ba9c35a7a687619** Private-AZ-1  
IPv4 address: Assigned from CIDR 10.0.2.0/24  
 **ap-south-1b (aps1-az3)**  
Subnet: **subnet-0dab19d95d26837db** Private-AZ-2  
IPv4 address: Assigned from CIDR 10.0.3.0/24

**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](#).  
**internal-lb**  
sg-08a502d6e3bacf8c7 VPC: vpc-0e582aa4695652a

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- 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.

**Listeners and routing** [Info](#)  
A listener is a process that checks for connection requests using the port and protocol you configure. The rules that you define for a listener determine how the load balancer routes requests to its registered targets.

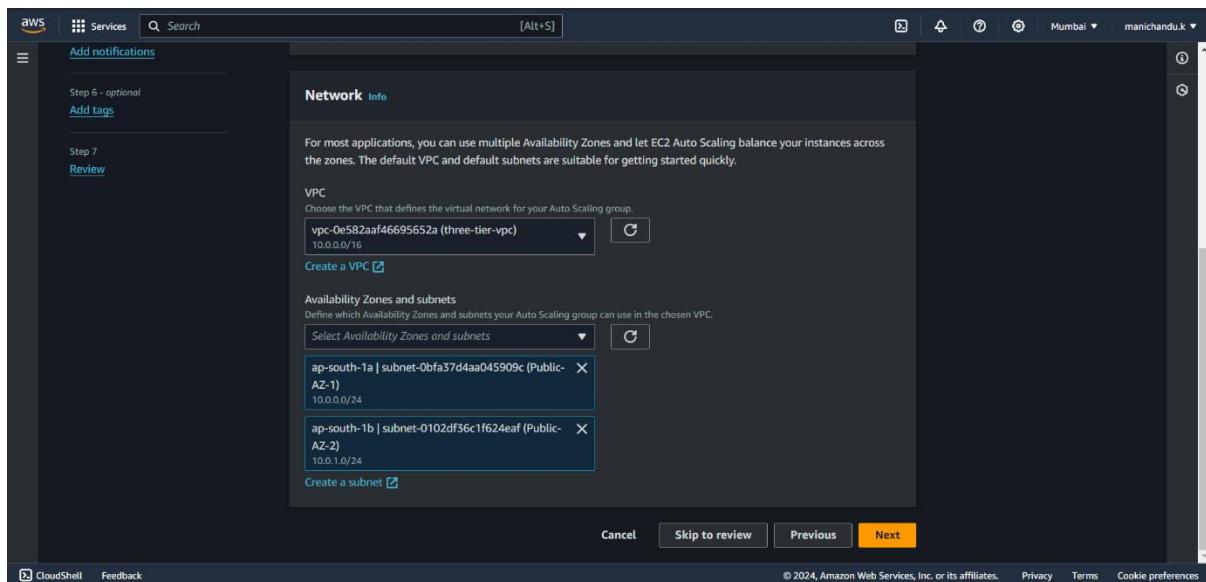
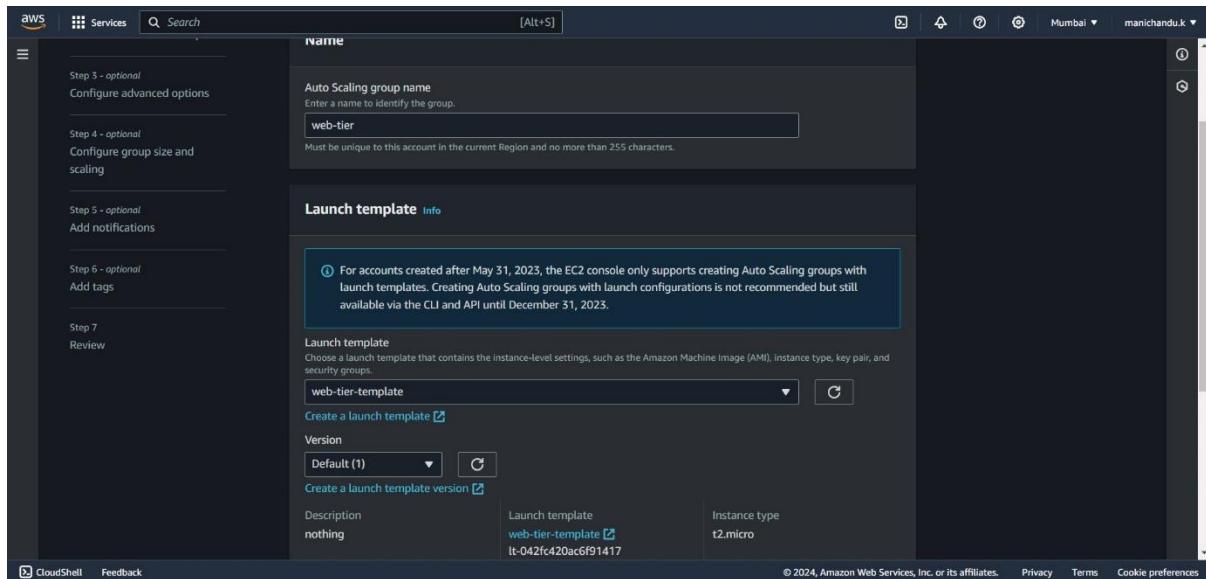
**Listener HTTP:80**  
Protocol: **HTTP** Port: **80** Default action: [Info](#) Forward to: **app-tier-target-group** Target type: Instance, IPv4  
Create target group

The screenshot shows the AWS EC2 Load Balancers page. On the left, there's a navigation sidebar with options like EC2 Dashboard, EC2 Global View, Events, Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity, Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, and CloudShell. The main content area has a header 'EC2 > Load balancers' with a 'Give feedback' button. A modal window titled 'Introducing resource map for Network Load Balancers' explains the feature. Below it, a table lists 'Load balancers (2)'. The table columns include Name, DNS name, State, VPC ID, Availability Zones, Type, and Date created. The first row is for 'internet-facing-lb' and the second for 'private-instance-lb'. At the bottom, a message says '0 load balancers selected' and 'Select a load balancer above.'

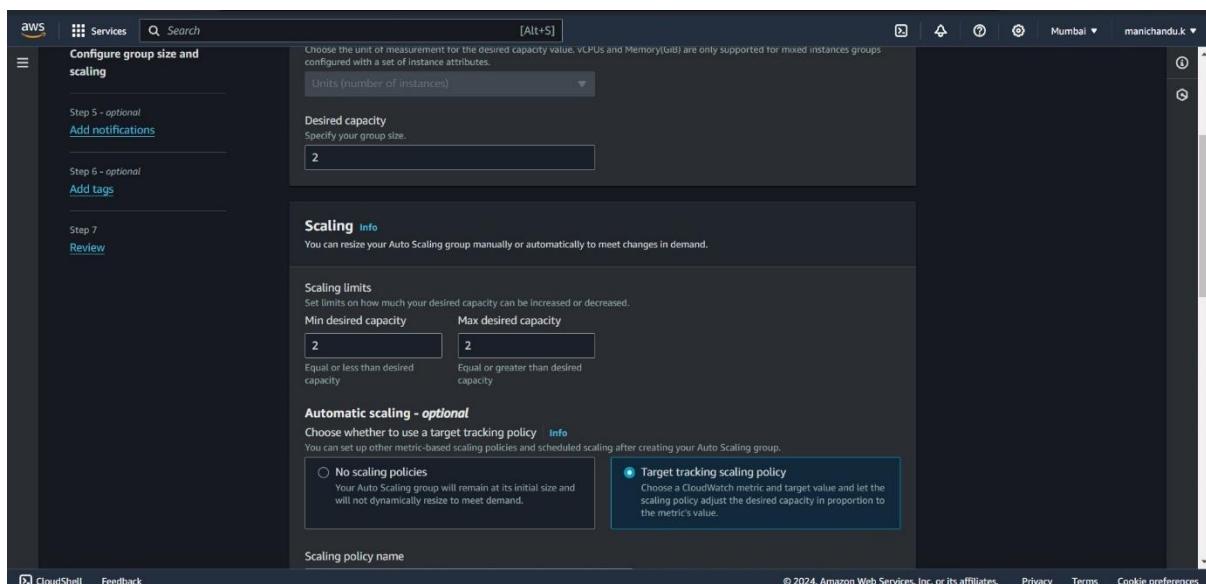
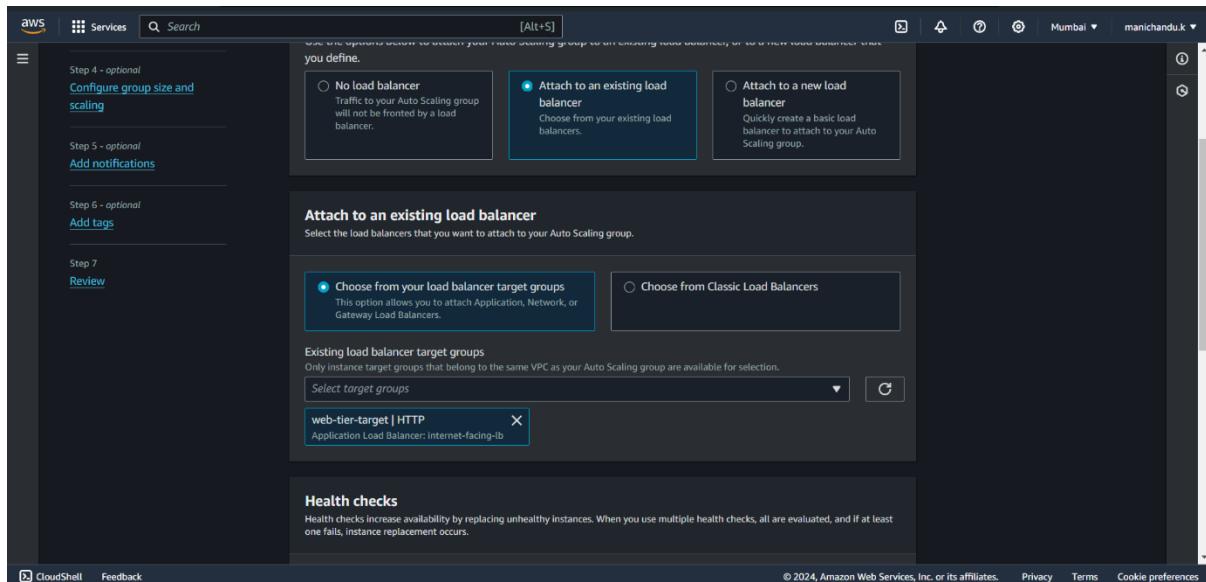
- Now it's time to create an auto-scaling group for both app-tier and web-tier instances.
- Go to the auto-scaling option to create auto-scaling groups for the app tier and web tier instances.

The screenshot shows the AWS Auto Scaling Groups page. The left sidebar includes options for Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces, Load Balancing, Load Balancers, Target Groups, Trust Stores, and Auto Scaling. The main content area has a large banner with the text 'Amazon EC2 Auto Scaling helps maintain the availability of your applications'. Below the banner, a section titled 'How it works' shows a diagram of an 'Auto Scaling group' containing four boxes: two solid and two dashed. Labels 'Minimum size' and 'Scale out as needed' are shown below the boxes. To the right, there are sections for 'Pricing' and 'Getting started'.

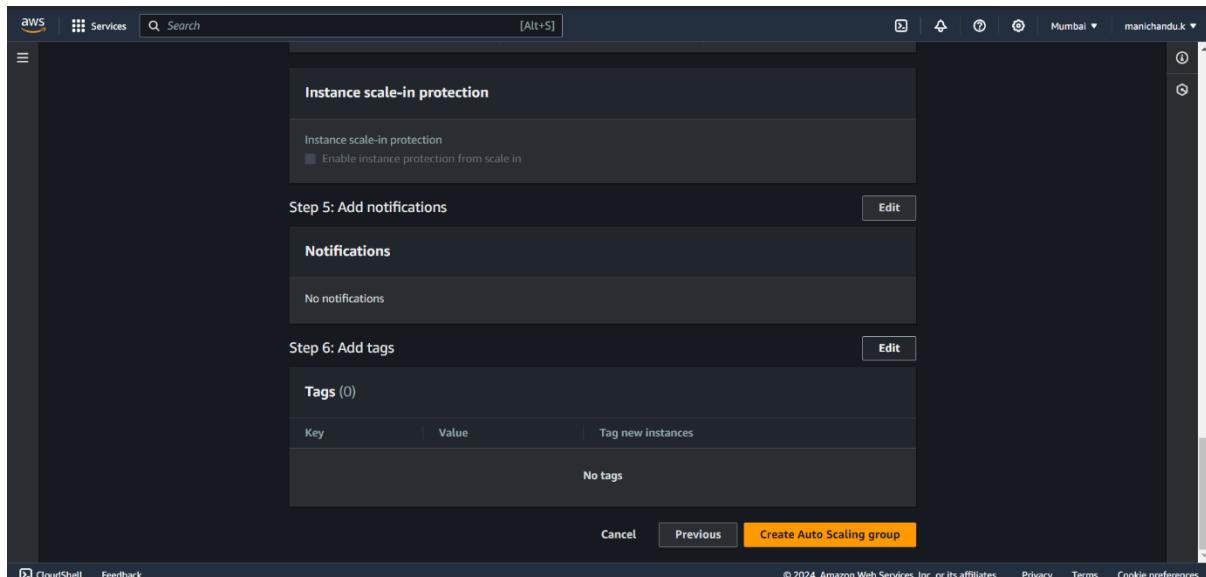
- Click on Create Auto Scaling group, give the name as web-tier, and select the web-tier template that we created earlier, configure the network settings by selecting the custom VPC and select the public subnets in both regions.



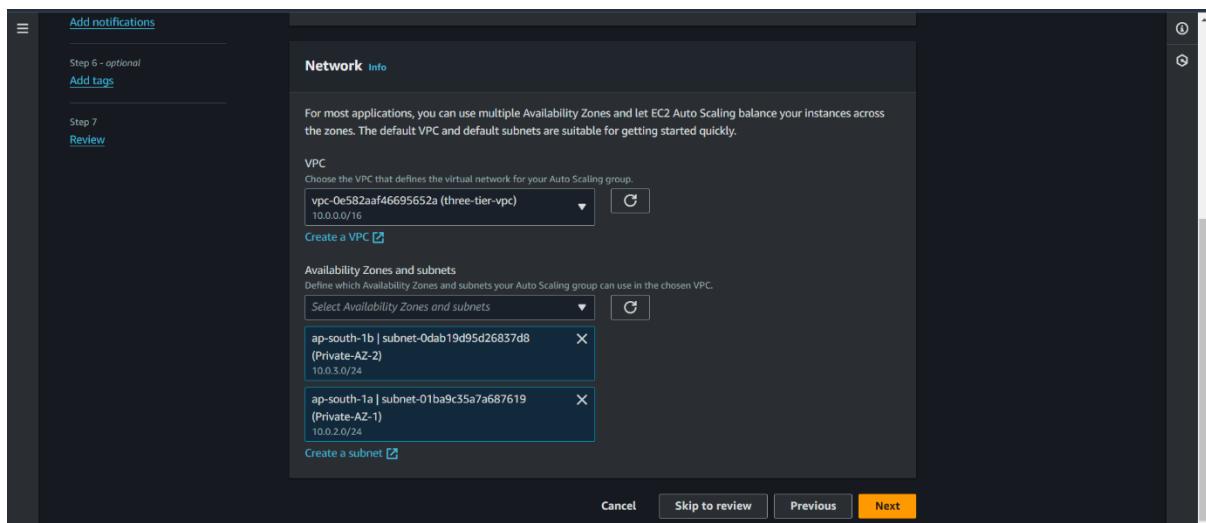
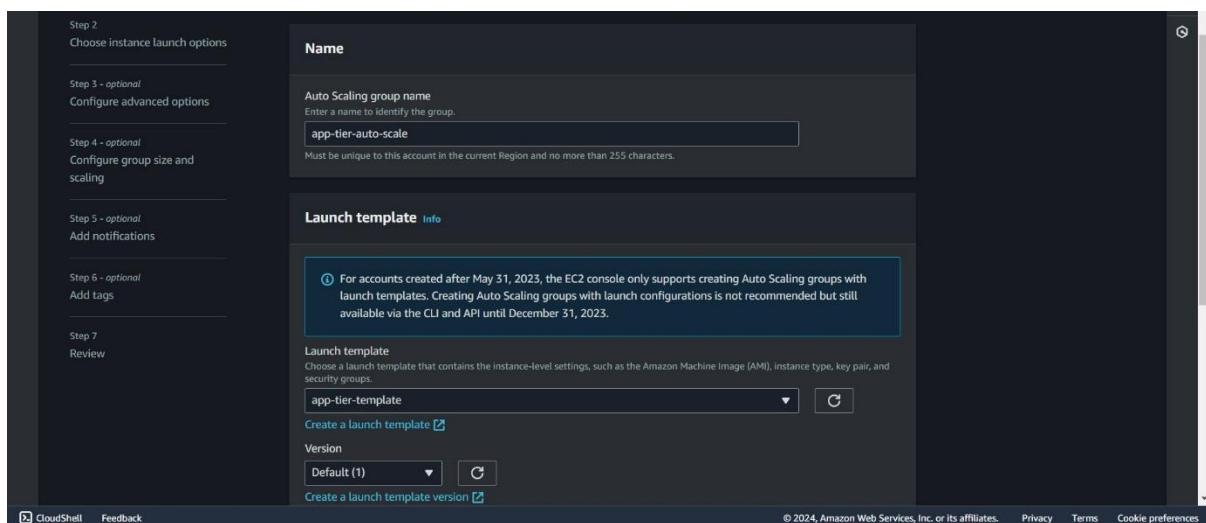
- Click on Next and select Attach it to an existing load balance, now select the target group that we created earlier for the web-tier instances.
- The health check time can be put default or custom. Here I left it as default. Click on Next and select the number of instances you want to scale, here I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



**Load balancing** Info

Use the options below to attach your Auto Scaling group to an existing load balancer, or to a new load balancer that you define.

No load balancer  
Traffic to your Auto Scaling group will not be fronted by a load balancer.

Attach to an existing load balancer  
Choose from your existing load balancers.

Attach to a new load balancer  
Quickly create a basic load balancer to attach to your Auto Scaling group.

**Attach to an existing load balancer**

Select the load balancers that you want to attach to your Auto Scaling group.

Choose from your load balancer target groups  
This option allows you to attach Application, Network, or Gateway Load Balancers.

Choose from Classic Load Balancers

**Existing load balancer target groups**

Only instance target groups that belong to the same VPC as your Auto Scaling group are available for selection.

Select target groups

app-tier-target-group | HTTP  
Application Load Balancer: private-instance-lb

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

**Desired capacity type**

Choose the unit of measurement for the desired capacity value. vCPUs and Memory(GiB) are only supported for mixed instances groups configured with a set of instance attributes.

Units (number of instances)

**Desired capacity**

Specify your group size.

2

**Scaling** Info

You can resize your Auto Scaling group manually or automatically to meet changes in demand.

**Scaling limits**

Set limits on how much your desired capacity can be increased or decreased.

Min desired capacity	Max desired capacity
2	2

Equal or less than desired capacity      Equal or greater than desired capacity

**Automatic scaling - optional**

Choose whether to use a target tracking policy Info

You can set up other metric-based scaling policies and scheduled scaling after creating your Auto Scaling group.

No scaling policies  
Your Auto Scaling group will remain at its initial size and will not dynamically resize to meet demand.

Target tracking scaling policy  
Choose a CloudWatch metric and target value and let the scaling policy adjust the desired capacity in proportion to the metric's value.

**Step 5: Add notifications**

**Notifications**

No notifications

**Step 6: Add tags**

**Tags (0)**

Key	Value	Tag new instances
No tags		

Cancel Previous Create Auto Scaling group

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

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with various EC2-related options like Instances, Images, and Elastic Block Store. The main area is titled 'Instances (6) Info' and contains a table with columns: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IP. There are six rows, each representing an instance. The instances are grouped into two Auto Scaling groups: 'app-tier' and 'web-tier'. The 'app-tier' group has three instances, and the 'web-tier' group has three instances. All instances are in the 'Running' state, indicated by green dots. The 'Status check' column shows '2/2 checks passed' for most instances. The 'Alarm status' column shows 'View alarms +'. The 'Availability Zone' column shows 'ap-south-1a' or 'ap-south-1b'. The 'Public IP' column shows '-'.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IP
	i-0babaf2f3142eb4a	Pending	t2.micro	-	<a href="#">View alarms +</a>	ap-south-1b	-
	i-01e890e2f1e4fcf7	Running	t2.micro	Initializing	<a href="#">View alarms +</a>	ap-south-1b	-
app-tier	i-0c6fedfc5bce462	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>	ap-south-1a	-
	i-02ce53311c08c0fd6	Running	t2.micro	Initializing	<a href="#">View alarms +</a>	ap-south-1a	-
	i-0bbb2697aa145235c	Pending	t2.micro	-	<a href="#">View alarms +</a>	ap-south-1a	-
web-tier	i-0fb8bd4f129b700f	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>	ap-south-1a	-

- 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.