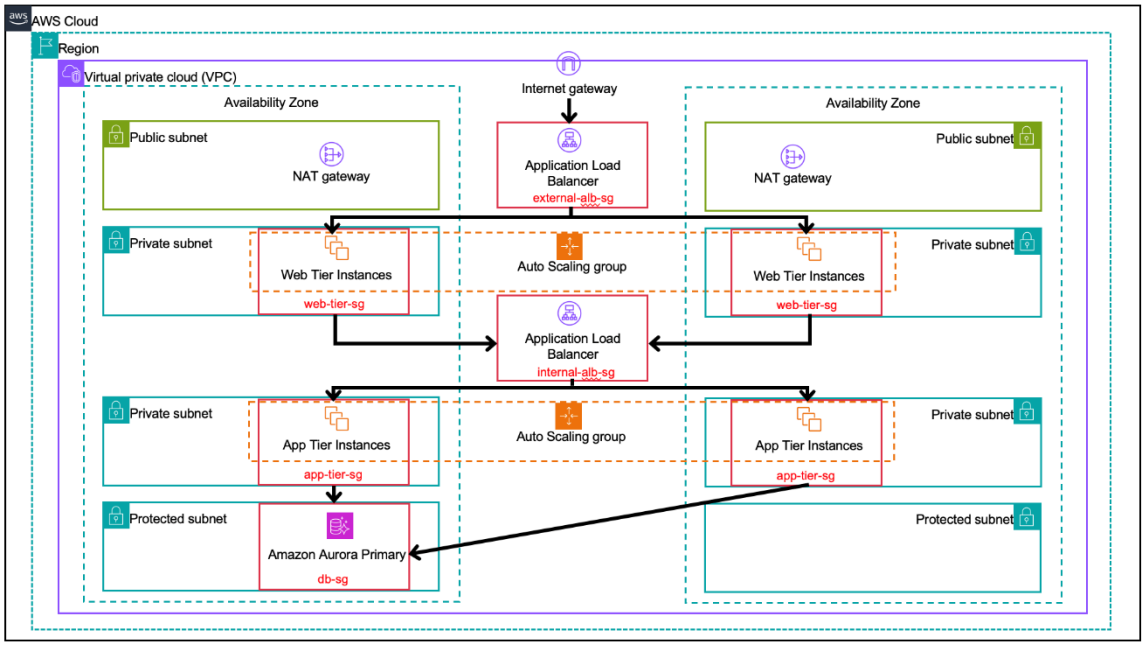
AWS Workshop: Highly available & scalable three-tier application deployment on AWS

Table of Contents

AWS Architecture 3
Step 1: Setting up networking & IAM roles as pre-requisite 4
Step 2: Creating a Web Server using EC2 Instance 6
Step 3: Creating App Server using EC2 Instance 9
Step 4: Connect Web Server with App Server 11
Step 5: Create Database 13
Step 6: Connect Database with App Tier 16
Step 7: Create Load Balancer and Autoscaling for Backend 18
Step 8: Create Load Balancer and Autoscaling for Frontend 24

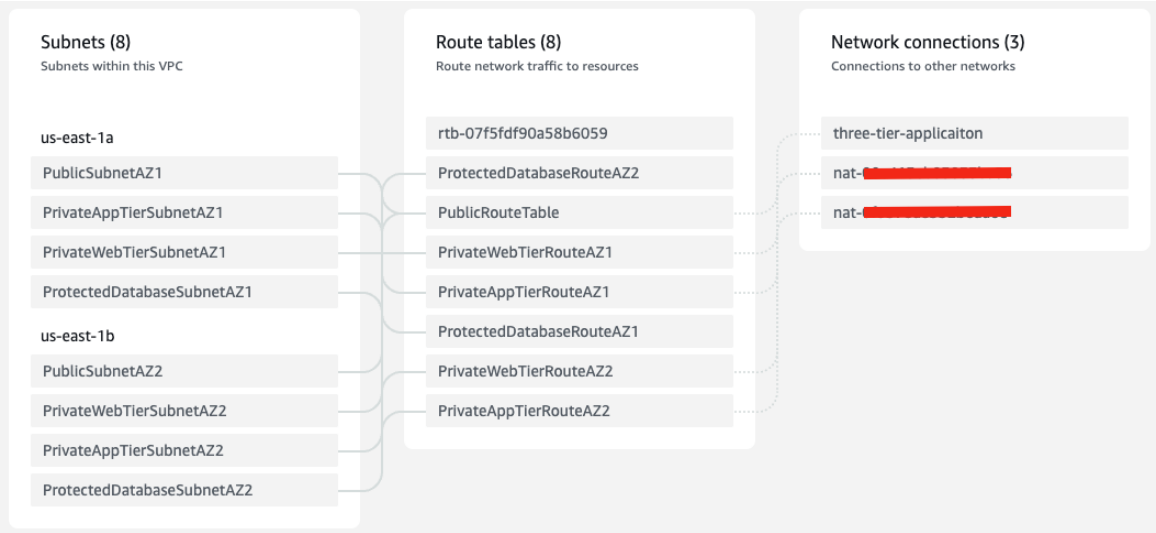
AWS Architecture



Step 1: Setting up networking & IAM roles as pre-requisite

1. Go to CloudFormation service.
2. Click on **Create a stack With new resources.**
3. Download the 'CFT link' to deploy the pre-requisite for the LAB
4. Choose **Upload a template** file and upload the file downloaded.
5. You can leave all parameters with default values.
6. Click on **Next** on **Step 2 (Specify stack details)** and **Step 3 (Configure stack options)**.
7. Finally, under review section click on **Submit**.
8. This stack creates the following resources:

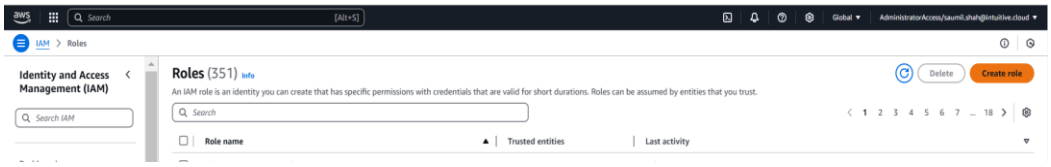
    - **VPC** with **2 public**, **4 private**, and **2 protected subnets**. Two public subnets would be connected to a common route table, having network connections to the **internet gateway**. Four private subnet will have 4 separate route tables, each route table will have network connects to to the **NAT gateway**. Protected subnets will have no path to the NAT gateways. Two private subnets will be used for frontend (web tier) logic and the other two private subnets for backend (app tier) logic. Protected subnets will have our RDS database.



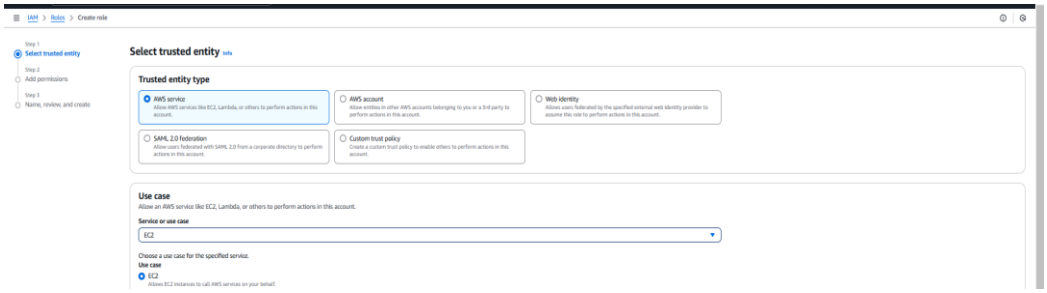
    - Security groups namely
        1. WebTierSecurityGroup – To be used for all WebTier resources [EC2, ELB]
        2. AppTierSecurityGroup – To be used for AppTier resources- [EC2]
        3. DatabaseSecurityGroup- To be used for DatabaseTier [RDS]

9. Create an **IAM instance profile** for EC2

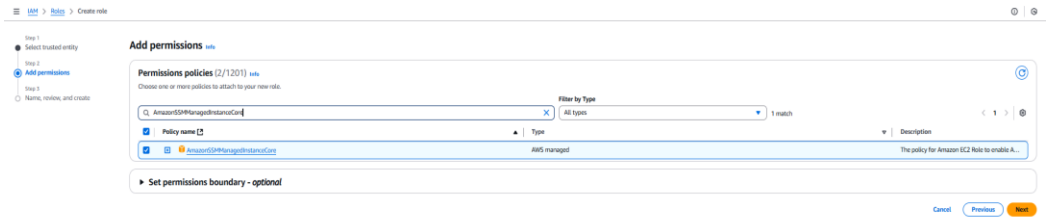
    1. Open AWS Console and go to the **IAM** service.
    2. Click on **Roles**, then click **Create role.**



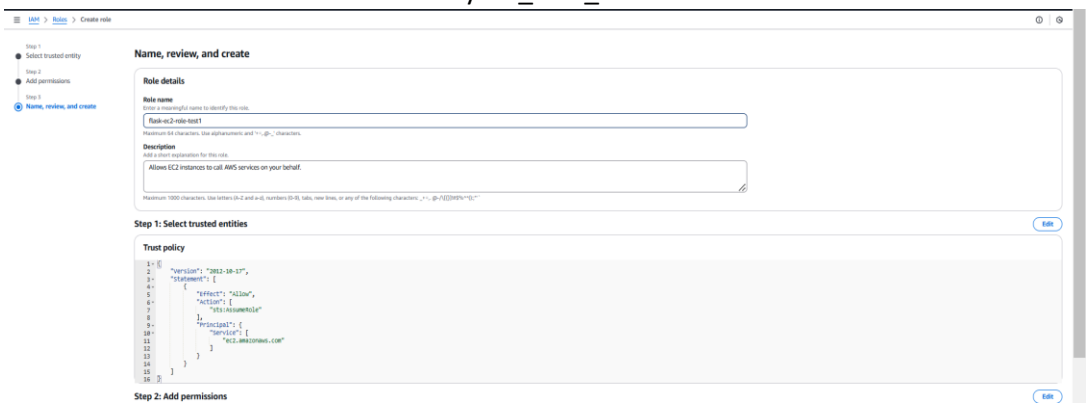
3. Select AWS service as the **trusted entity** and choose **EC2**. **Click Next.**



4. Search for and attach the '**AmazonS3FullAccess'** and '**AmazonSSMManagedInstanceCore'** policies. **Click Next.**



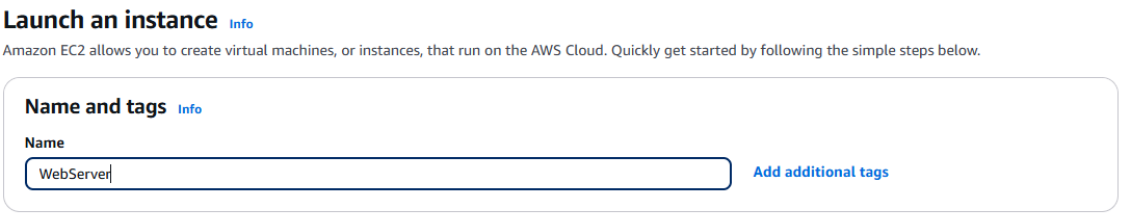
5. Enter the role name as flask-ec2-role-<your_user_id> and click **Create role.**



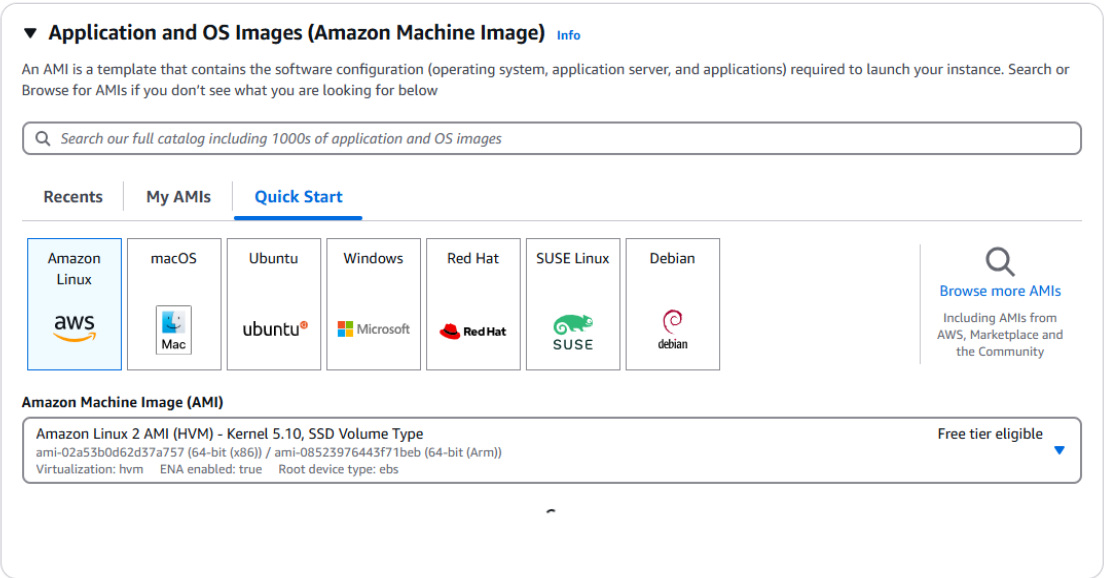
Step 2: Creating a Web Server using EC2 Instance

1. Open the Amazon EC2. From the EC2 console dashboard, in the Launch instance pane, choose **Launch instance**.

2. Under Name and tags, for name enter Websever.

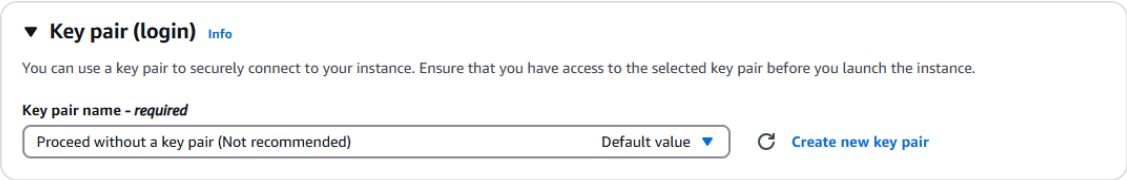


3. Under Application and OS Images (Amazon Machine Image). Choose **Quick Start** and then choose the operating system (OS) for your instance. From Amazon Machine Image (AMI), select **Amazon Linux 2AMI**.

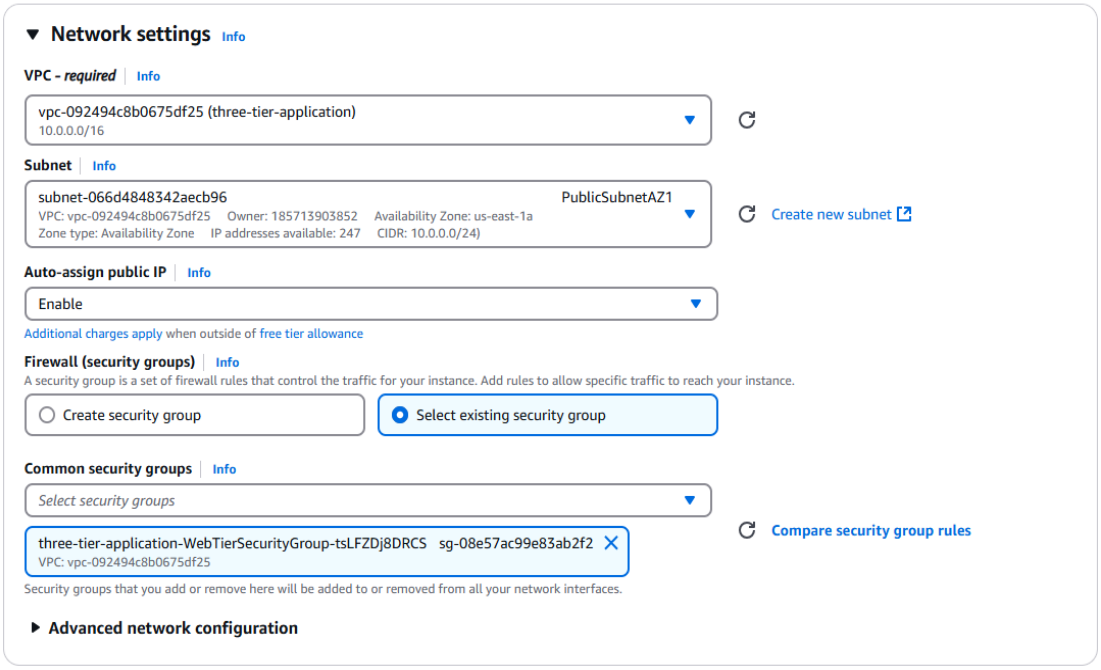


4. Under Instance type, for Instance type, choose **t2.micro**.

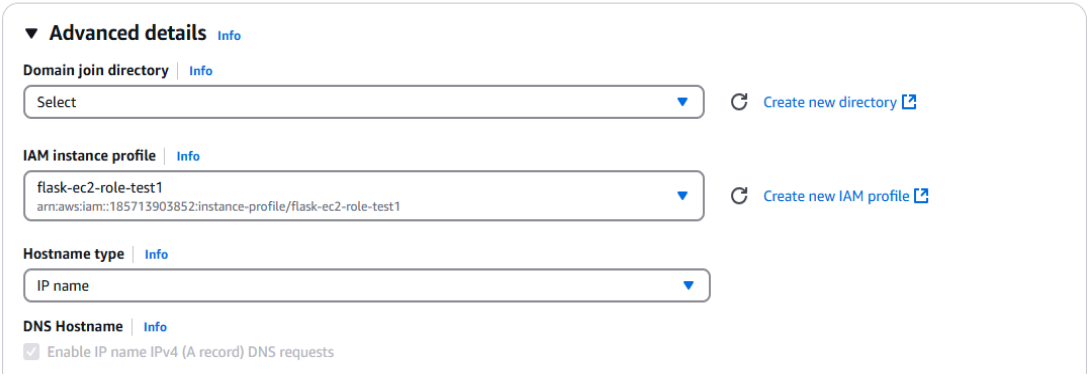
5. Under Key pair (login), **Proceed without a key pair**.



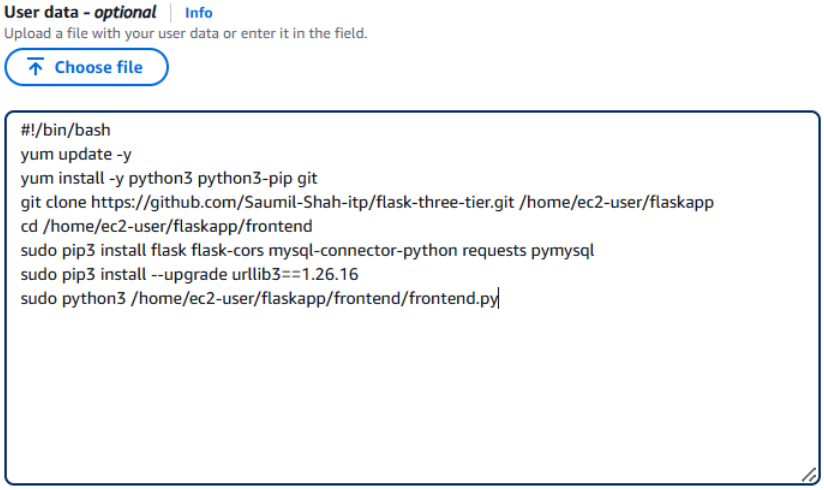
6. Under Network settings, select **Edit**, under VPC choose **three-tier-application** VPC created in step 1. Under subnet select **PublicSubnetAZ1**. For Auto-assign public IP select **Enable**. Choose Select **existing security group** and choose **WebTierSecurityGroup.**



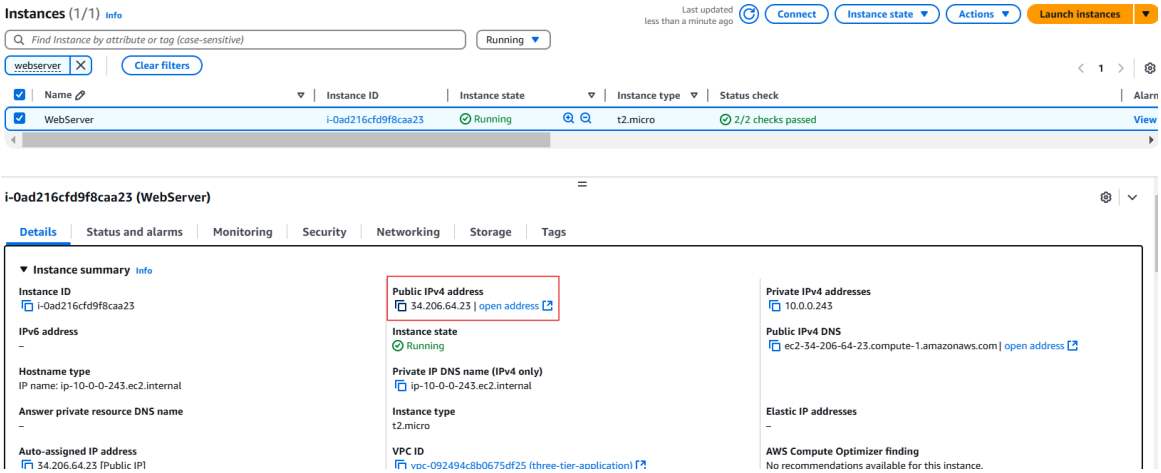
7. Under Advanced Details section, for **IAM instance profile** select **flask-ec2-role-<your_user_id>** created in step-1.



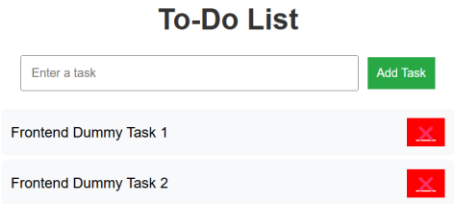
8. Scroll down to **User Data** and copy contents from this Link.



9. Click on **Launch Instance**.
10. Once the instance is up and running in **Healthy** state with **2/2 checks passed**. Select the instance and copy the public IP.



11. Open your browser and search for http://<public_id>:80. The screen below should appear.

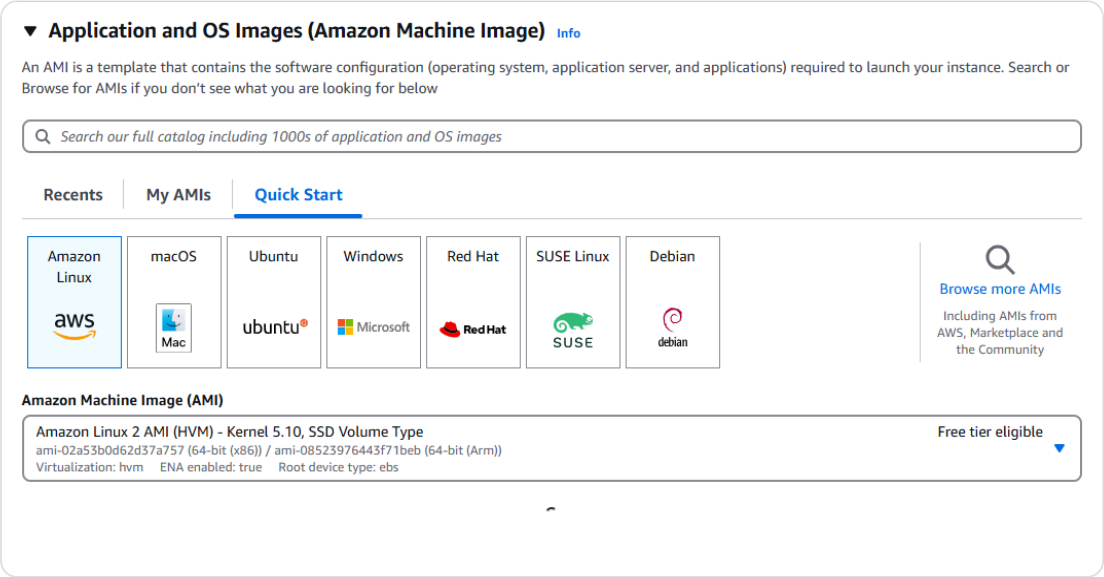


Step 3: Creating App Server using EC2 Instance

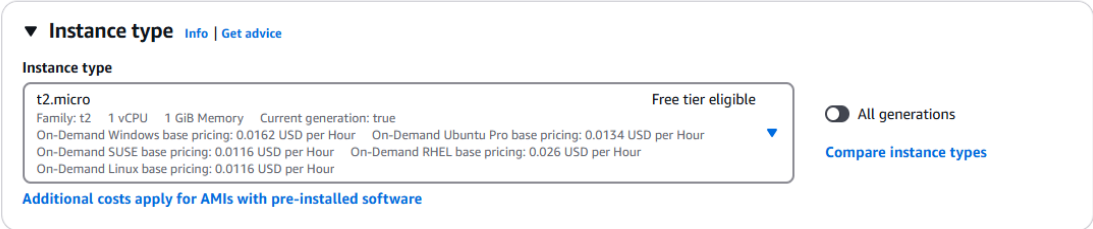
1. Open the Amazon EC2. From the EC2 console dashboard, in the Launch instance pane, choose **Launch instance**.

2. Under Name and tags, for name enter **Appserver**.

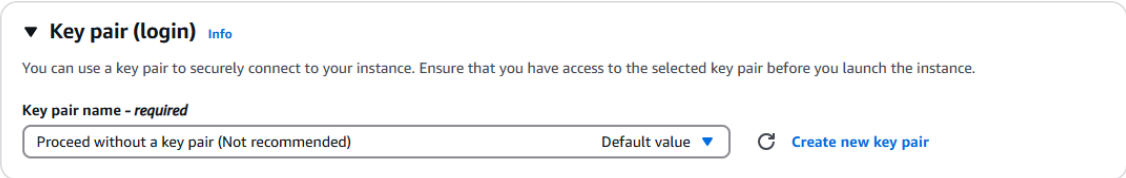
3. Under Application and OS Images (Amazon Machine Image). Choose **Quick Start** and then choose the operating system (OS) for your instance. From Amazon Machine Image (AMI), select **Amazon Linux 2AMI**.



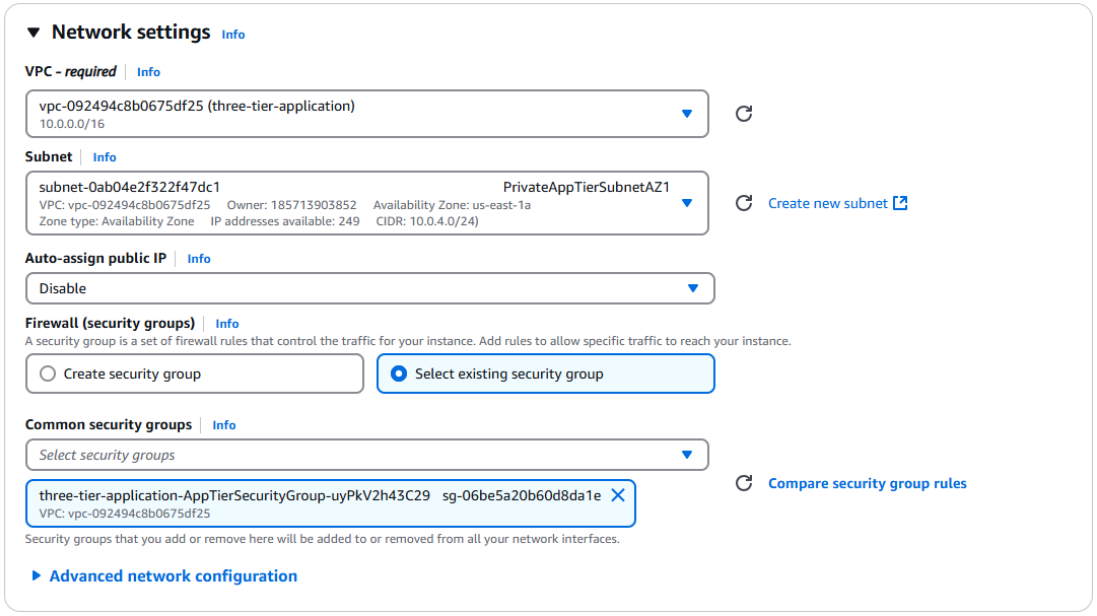
4. Under Instance type, for Instance type, choose **t2.micro**.



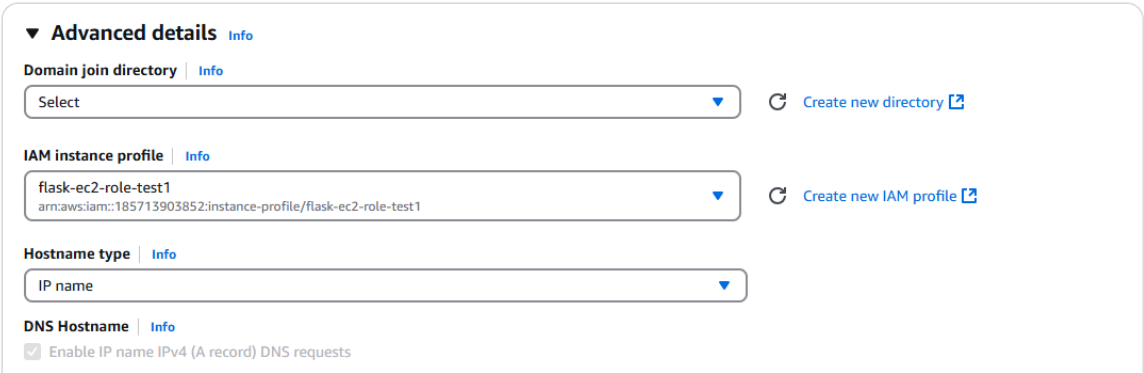
5. Under Key pair (login), **Proceed without a key pair**.



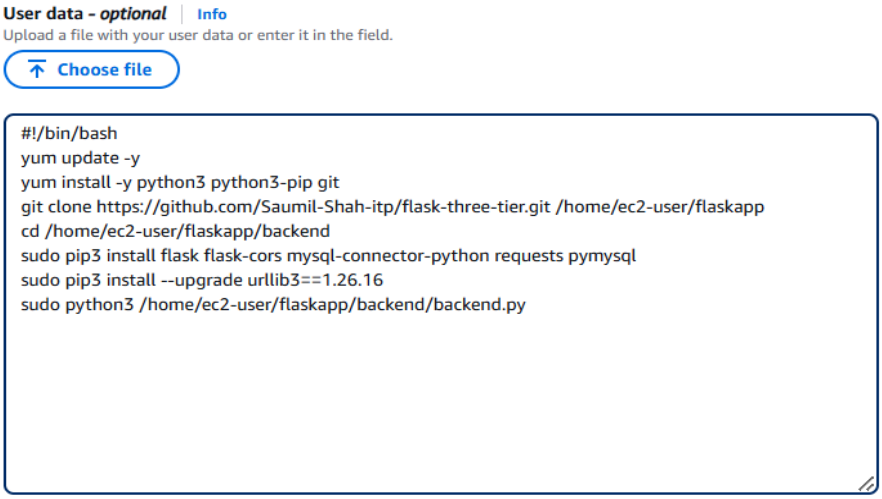
6. Under Network settings, select **Edit**, under VPC choose **three-tier-application** VPC created in step 1. Under subnet select **PrivateAppTierSubnetAZ1**. Choose Select **existing security group** and choose **AppTierSecurityGroup.**



7. Under Advanced Details section, for **IAM instance profile** select **flask-ec2-role-<your_user_id>** created in step-1.



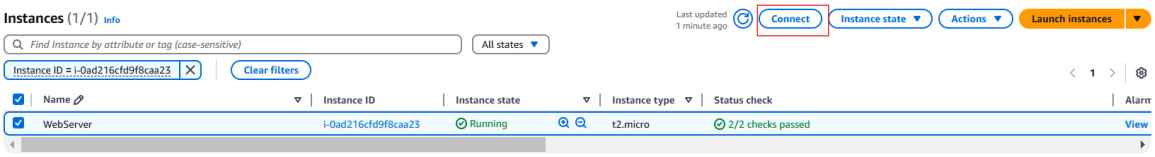
8. Scroll down to **User Data** and copy contents from this Link.



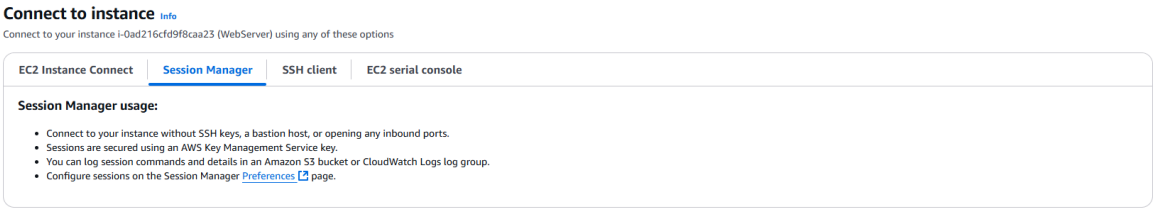
9. Click on **Launch Instance**.
10. Once the instance is up and running in **Healthy** state with **2/2 checks passed**.

Step 4: Connect Web Server with App Server

1. Navigate to EC2 Instance and choose **Webserver.** Select **Connect.**



2. Select **Session Manager** tab and click on **Connect**.



3. Execute following steps:
   a. sudo su ec2-user
   b. cd ~
   c. cd flaskapp/frontend/

```
sh-4.2$ sudo su ec2-user
[ec2-user@ip-10-0-0-243 bin]$ cd ~
[ec2-user@ip-10-0-0-243 ~]$ cd flaskapp/frontend/
[ec2-user@ip-10-0-0-243 frontend]$ sudo vim frontend.py
[ec2-user@ip-10-0-0-243 frontend]$
```

   d. sudo vim frontend.py
   e. Press the **i** key to enter insert mode and replace the **BACKEND_API** variable with the **Private IP** address of the **Appserver**.

```
from flask import Flask, render_template, request, redirect, url_for
import requests

app = Flask(__name__)

# ✅ Backend API URL - Replace with your actual backend DNS or IP
BACKEND_API = "http://10.0.4.40:5000"  # Update this with Load Balancer later

# ✅ Dummy data for fallback
tasks = [{"id": 1, "name": "Frontend Dummy Task 1"}, {"id": 2, "name": "Frontend Dummy Task 2"}]


@app.route("/", methods=["GET", "POST"])
def index():
    global tasks

    if request.method == "POST":
        task_name = request.form.get("task_name")
        try:
            requests.post(f"{BACKEND_API}/add", json={"name": task_name})
        except requests.exceptions.RequestException:
            new_id = max([task["id"] for task in tasks]) + 1 if tasks else 1
            tasks.append({"id": new_id, "name": task_name})

        return redirect(url_for("index"))

    try:
        response = requests.get(f"{BACKEND_API}/tasks")
        response.raise_for_status()
        tasks = response.json()
    except requests.exceptions.RequestException as e:
        print(f"⚠️ Backend API failed: {e}")

    return render_template("index.html", tasks=tasks)


@app.route("/delete/<int:task_id>")
def delete_task(task_id):
    global tasks
    try:
        requests.delete(f"{BACKEND_API}/delete/{task_id}")
    except requests.exceptions.RequestException:
:wq!
```

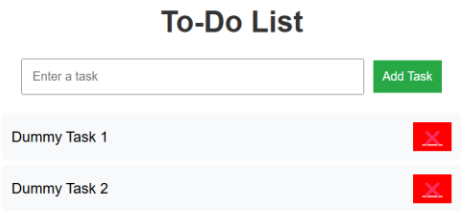
   f. Press the **Esc** key, type :**wq!**, and hit **Enter** to save and exit.
   g. sudo fuser -k 80/tcp

h. sudo nohup python3 frontend.py --port 80 &

```
[ec2-user@ip-10-0-4-40 backend]$ sudo nohup python3 backend.py --port 5000 &
[1] 1169
[ec2-user@ip-10-0-4-40 backend]$ nohup: ignoring input and appending output to 'nohup.out'

[ec2-user@ip-10-0-4-40 backend]$
```

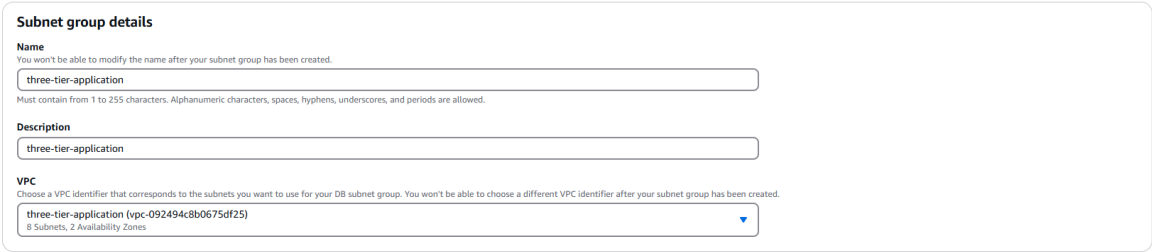
i. Hit **Enter** twice.

10. Open your browser and search for http://<**public_id_of_webserver**>:80. The screen below should appear.

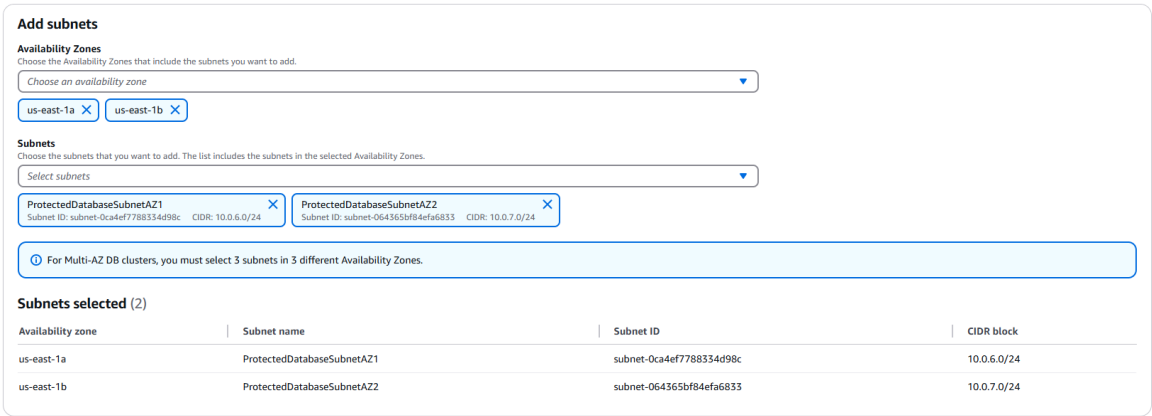


Step 5: Create Database

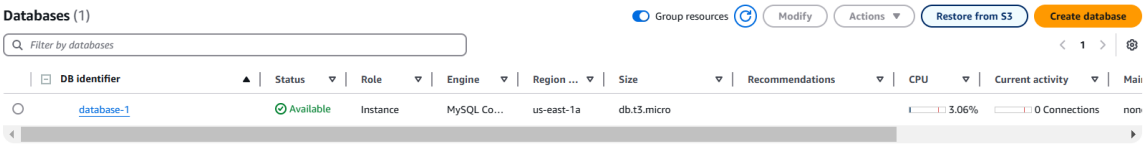
1. Navigate to the RDS dashboard in the AWS console and click on **Subnet groups** on the left-hand side. Click **Create DB subnet group**.
2. Name the subnet group as **three-tier-subnet-group** and choose VPC named **three-tier-application**



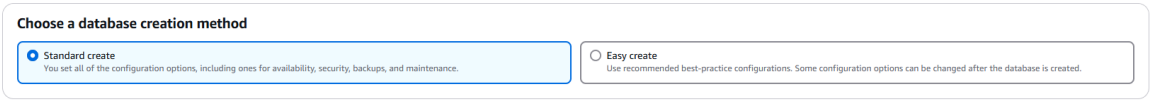
3. Under **Availability Zones** select **us-east-1a** and **us-east1b.** For subnet choose **ProtectedDatabaseSubnetAZ1** and **ProtectedDatabaseSubnetAZ2.** Click **Create.**



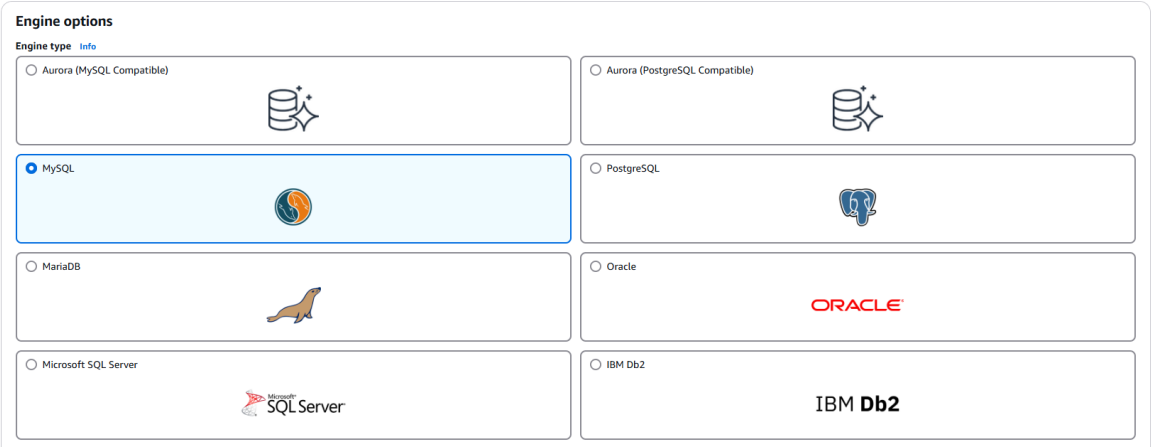
4. Sign in to the AWS Management Console and open the Amazon RDS console.
5. In the navigation pane, choose **Databases**.
6. Choose **Create database.**



7. Choose **Standard create**.

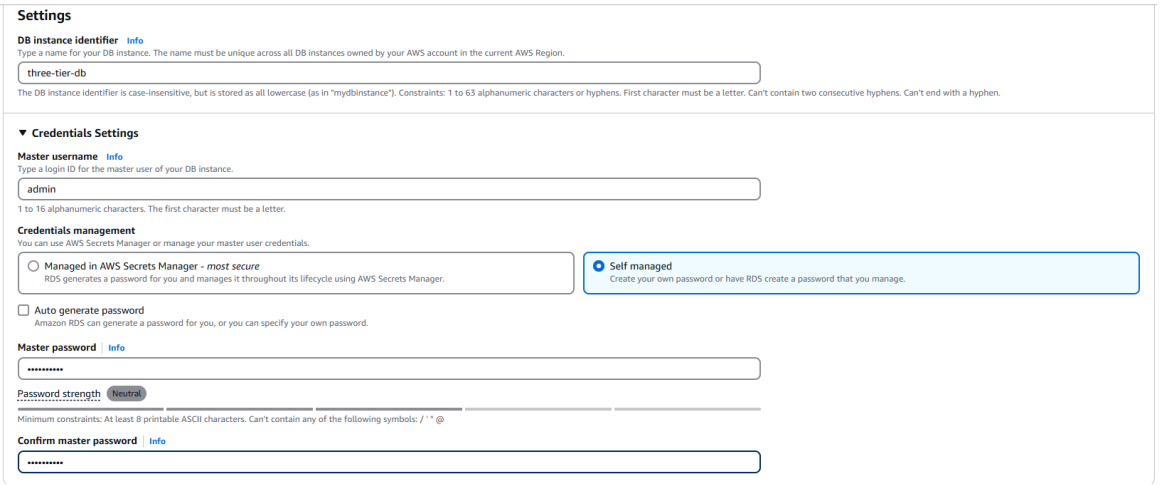


8. For Engine type, choose **MySQL**.

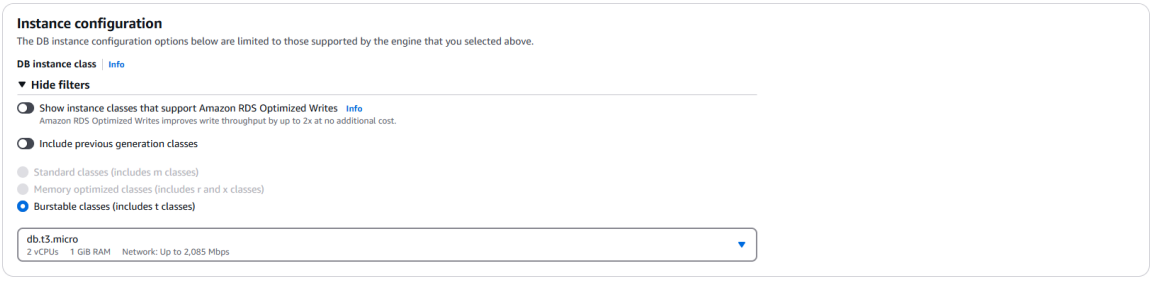


9. In Templates, choose the template that matches your use case. Choose **Free Tier.**

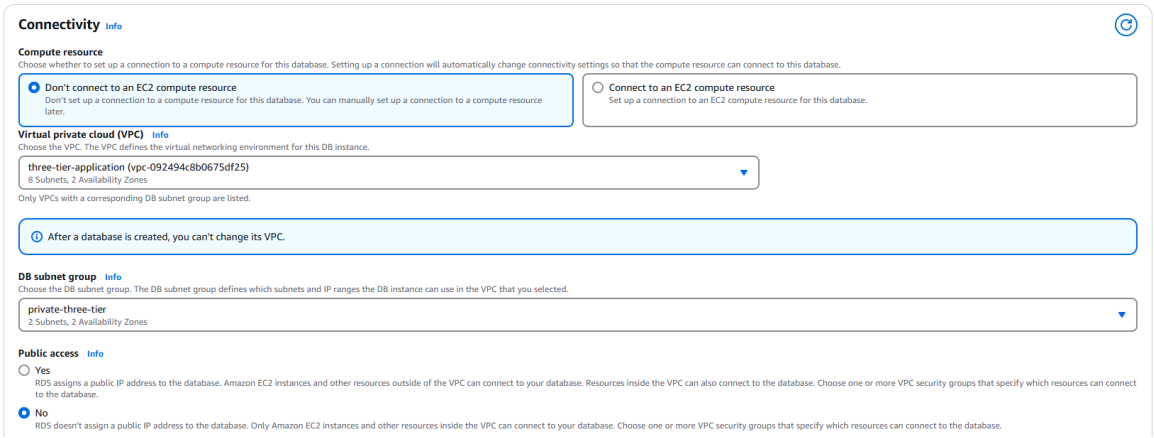
10. In the Settings section **name** the database as **three-tier-db** and open Credential management and select **Self managed**. Enter the same password in Master password and Confirm password.



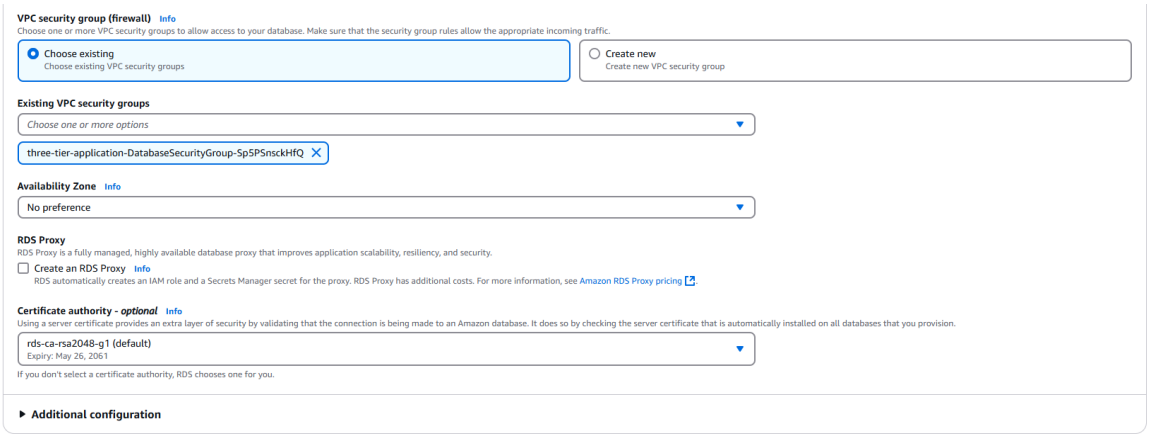
8. Under **Instance configuration** section choose **Burstable classes** and select **db.t3.micro** instance type.



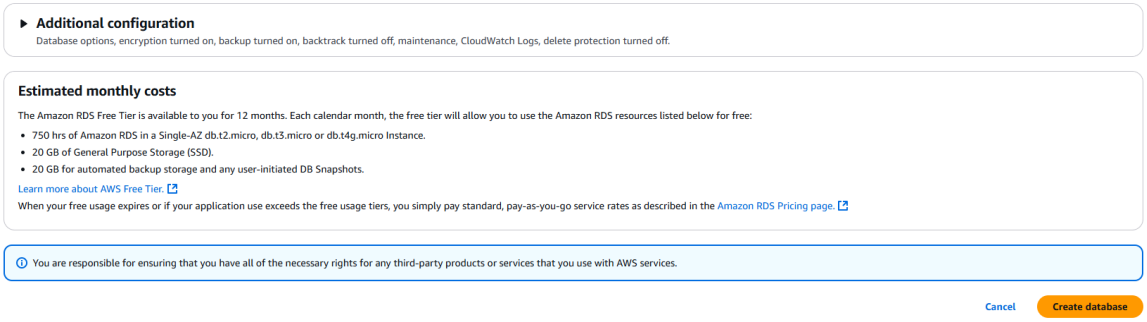
9. Under **Connectivity** section, under VPC select **three-tier-application** and for subnet group select **three-tier-subnet group.**



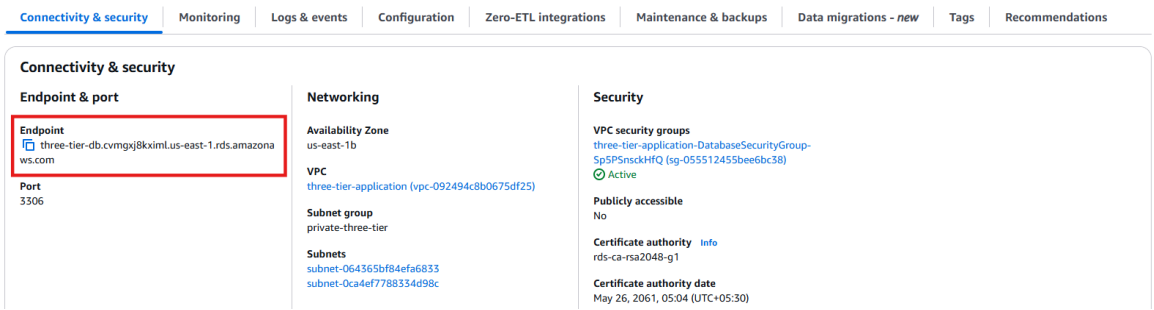
10. For the **VPC Security Group**, select **"Existing"** and choose **three-tier-application-DatabaseSecurityGroup.**



11. Scroll and create database.

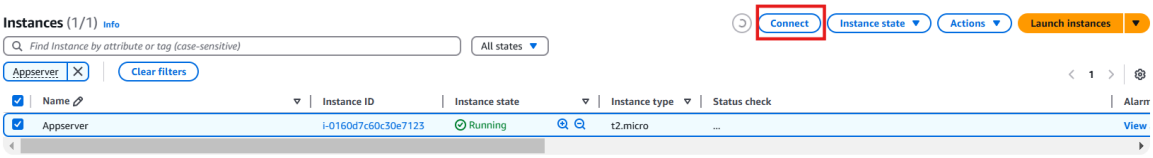


12. Once the database is in **running** state. Select the database and navigate to **Endpoint & port** copy the **endpoint.**

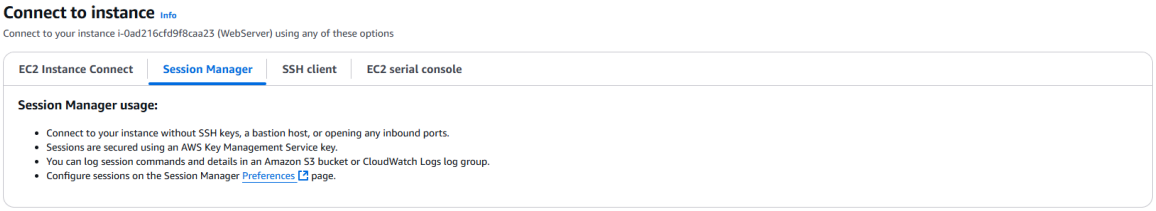


Step 6: Connect Database with App Tier

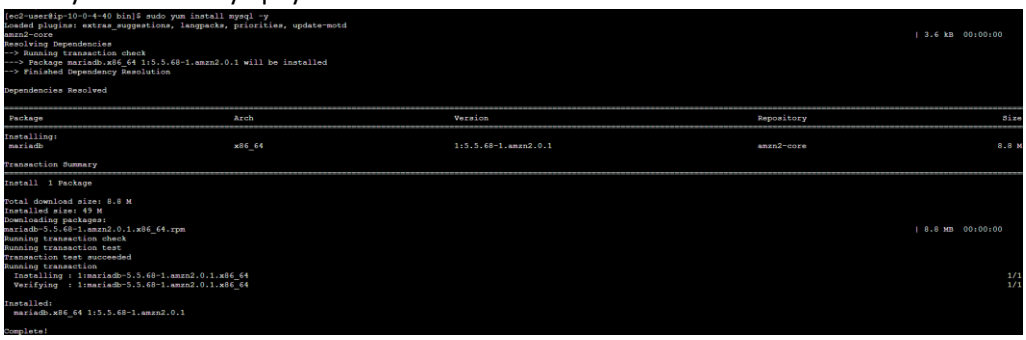
1. Navigate to EC2 Instance and choose **Appserver.** Select **Connect.**



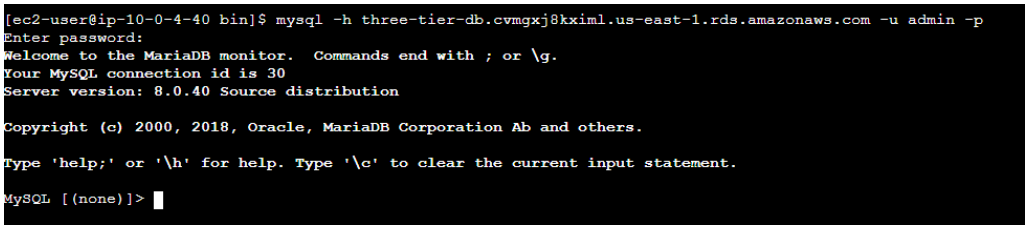
2. Select **Session Manager** tab and click on **Connect**.



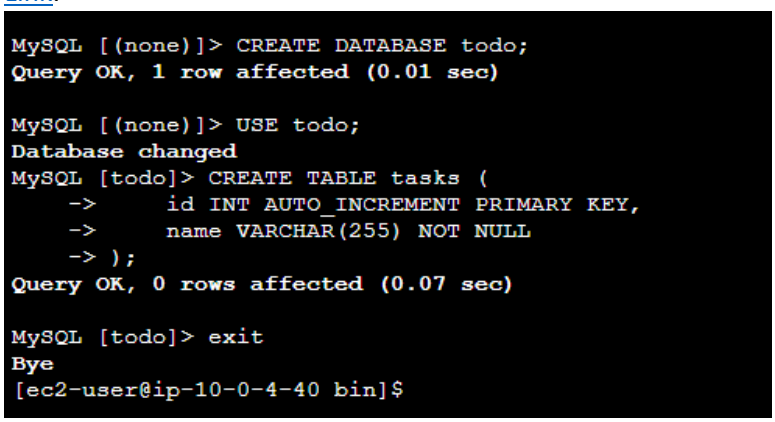
3. Perform the following steps to create a table.
   a. For connecting to the database execute following commands:
      i. sudo su ec2-user
      ii. sudo yum install mysql -y



      iii. mysql -h <database-endpoint> -u admin -p



      iv. Once the database is connected, execute the SQL queries from the provided [Link](#).



4. Perform the following steps to connect to the database to app tier.
   a. cd /home/ec2-user/flaskapp/backend/
   b. sudo vim backend.py
   c. Press the **i** key to enter insert mode and replace the database config as:
      i. Host -> **database endpoint**
      ii. User -> **admin**
      iii. Password -> **<your_database_password>**
      iv. Database -> **todo**

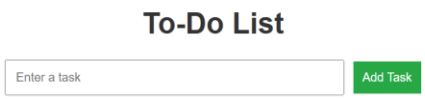
```
from flask import Flask, jsonify, request
import pymysql

app = Flask(__name__)

# ✅ Replace with your RDS MySQL details
DB_CONFIG = {
    "host": "three-tier-db.cvmgxj8kximl.us-east-1.rds.amazonaws.com",
    "user": "admin",
    "password": "<type-your-password-here>",
    "database": "todo"
}

# ✅ Try to connect to DB
```

   d. Press the **Esc** key, type :**wq!**, and hit **Enter** to save and exit.
   e. sudo fuser -k 5000/tcp
   f. sudo nohup python3 backend.py --port 5000 &

```
[ec2-user@ip-10-0-4-40 backend]$ sudo nohup python3 backend.py --port 5000 &
[1] 1169
[ec2-user@ip-10-0-4-40 backend]$ nohup: ignoring input and appending output to 'nohup.out'

[ec2-user@ip-10-0-4-40 backend]$
```

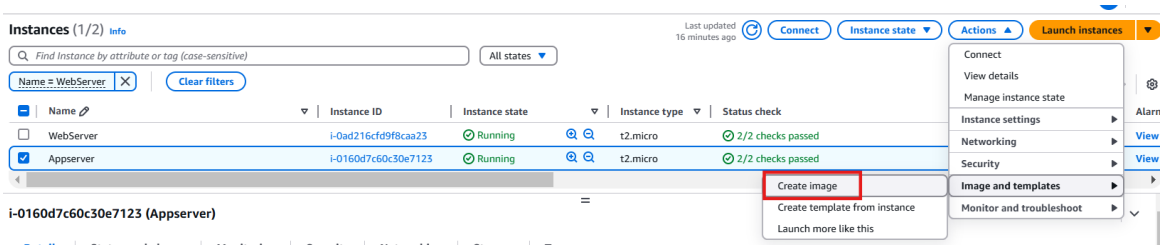
   g. Hit **Enter** twice.

5. Open your browser and search for http://<**public_id_of_webserver**>:80. The screen below should appear.

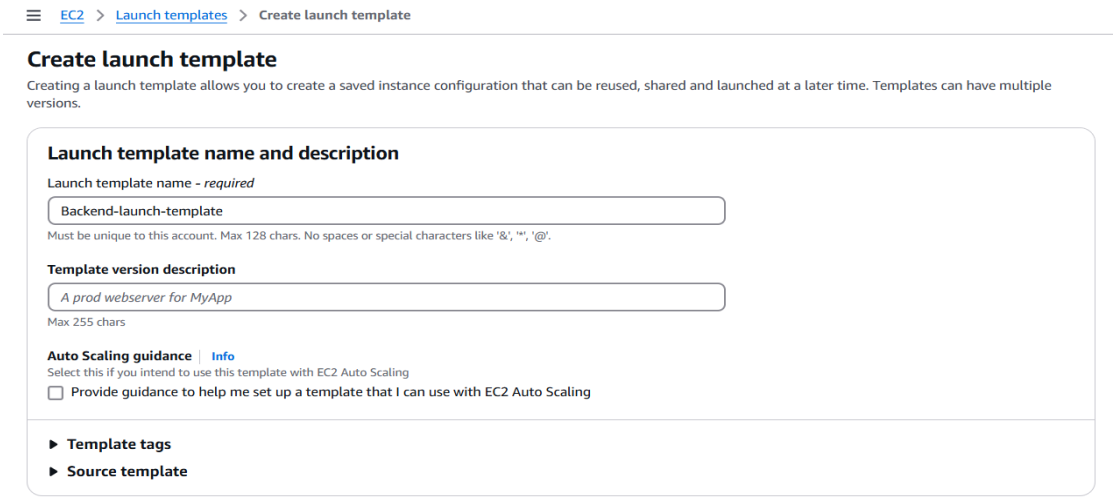


Step 7: Create Load Balancer and Autoscaling for Backend

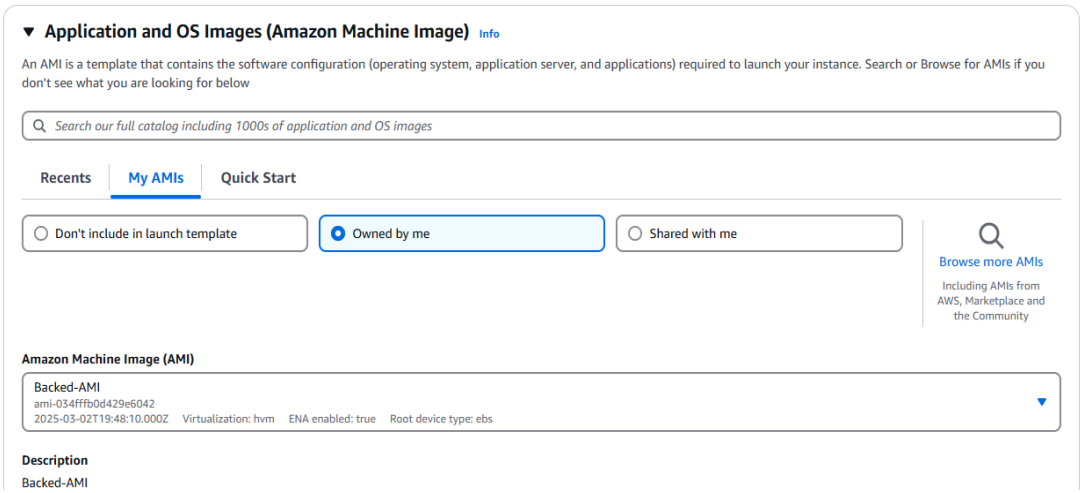
1. Create Image for Autoscaling group, Click on actions & select **'Image & template'** create an image using the instance. Provide the **Image name** as **Backend-AMI** and **Create Image**. Make sure its status appears **Available.**



2. Navigate to EC2 Console. Click Launch Templates in the left sidebar and click on create a **New Launch Template.** Enter Launch template name **backend-launch-template.**

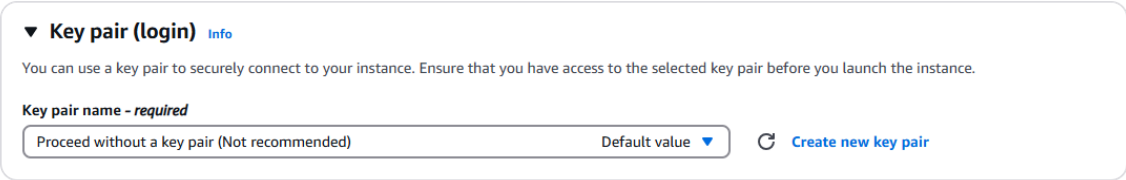


3. Under **Amazon Machine Image (AMI)**, select **'Backend-AMI'** created above

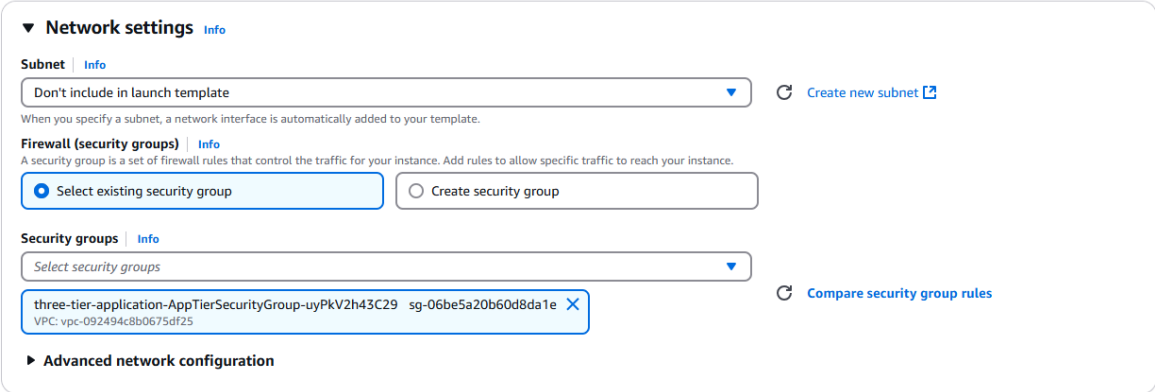


4. Choose an **Instance Type** (e.g., ‘t2.micro’)

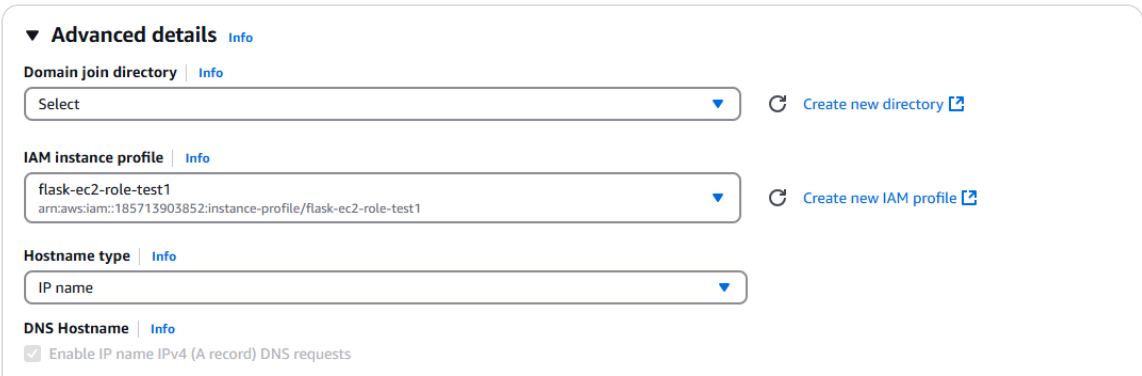
5. Under Key pair (login), **Proceed without a key pair**.



6. Keep default network settings as **‘don’t include in launch template’.** For security group select **three-tier-application-AppTierSecurityGroup.**



7. Under Advanced Details section, for **IAM instance profile** select **flask-ec2-role-<your_user_id>** created in step-1.



11. Scroll down to **User Data** and copy contents from this Link.

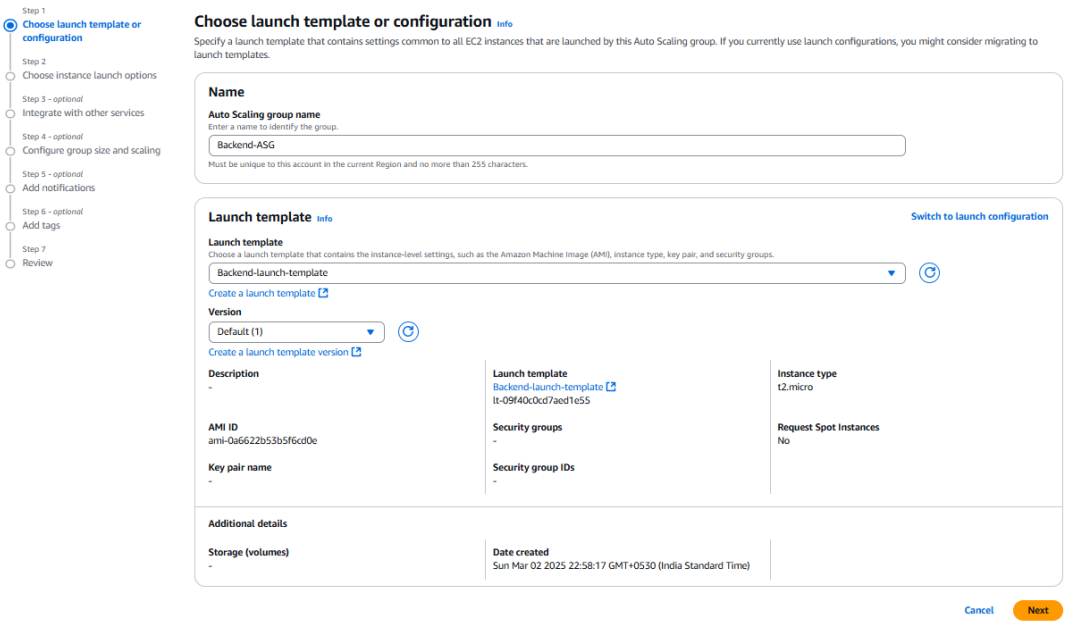
User data - *optional* | Info

Upload a file with your user data or enter it in the field.

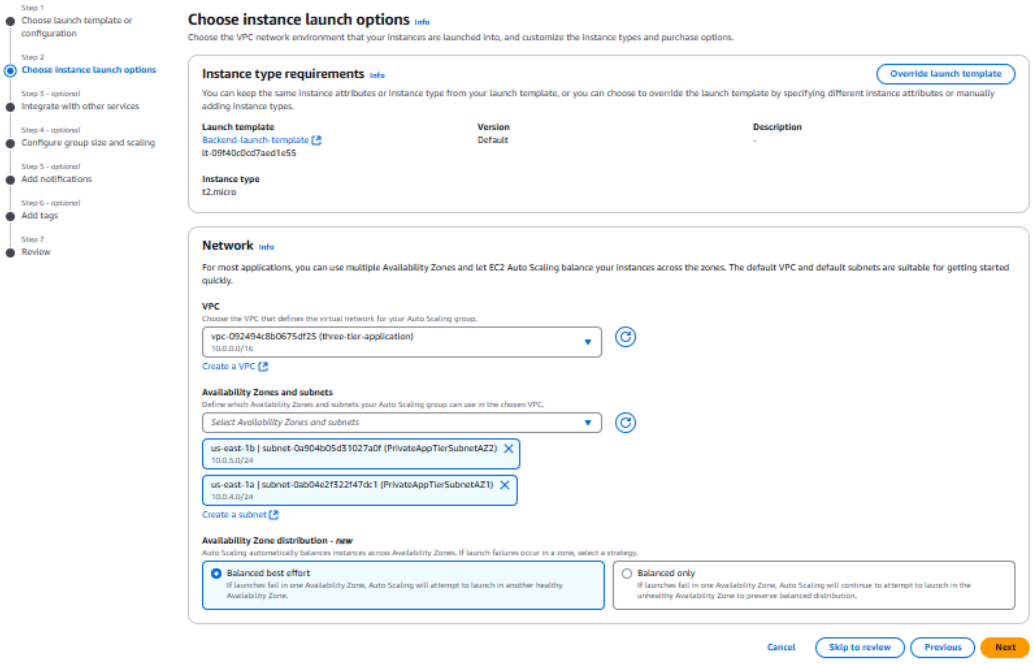
Choose file

```
#!/bin/bash
yum update -y
yum install -y python3 python3-pip git
git clone https://github.com/Saumil-Shah-itp/flask-three-tier.git /home/ec2-user/flaskapp
cd /home/ec2-user/flaskapp/backend
sudo pip3 install flask flask-cors mysql-connector-python requests pymysql
sudo pip3 install --upgrade urllib3==1.26.16
sudo python3 /home/ec2-user/flaskapp/backend/backend.py
```

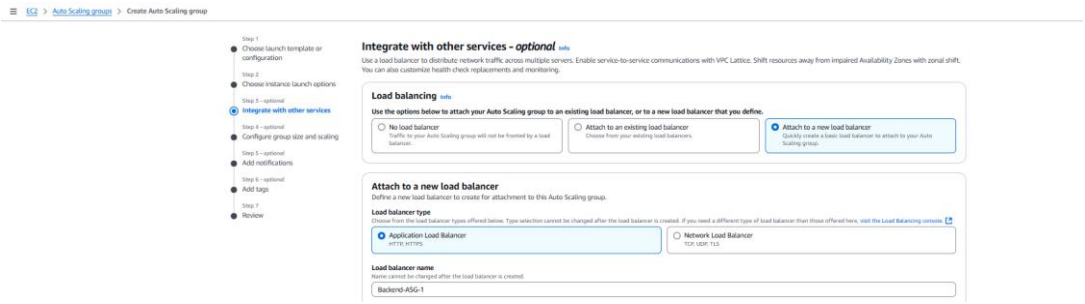
12. Click on **Create Launch Template.**
13. Open the **AWS Auto Scaling Console.** Click **Create an Auto Scaling group**
14. Enter ASG Name **backend-asg.** Under **Launch Template**, select the one you created **backend-launch-template**



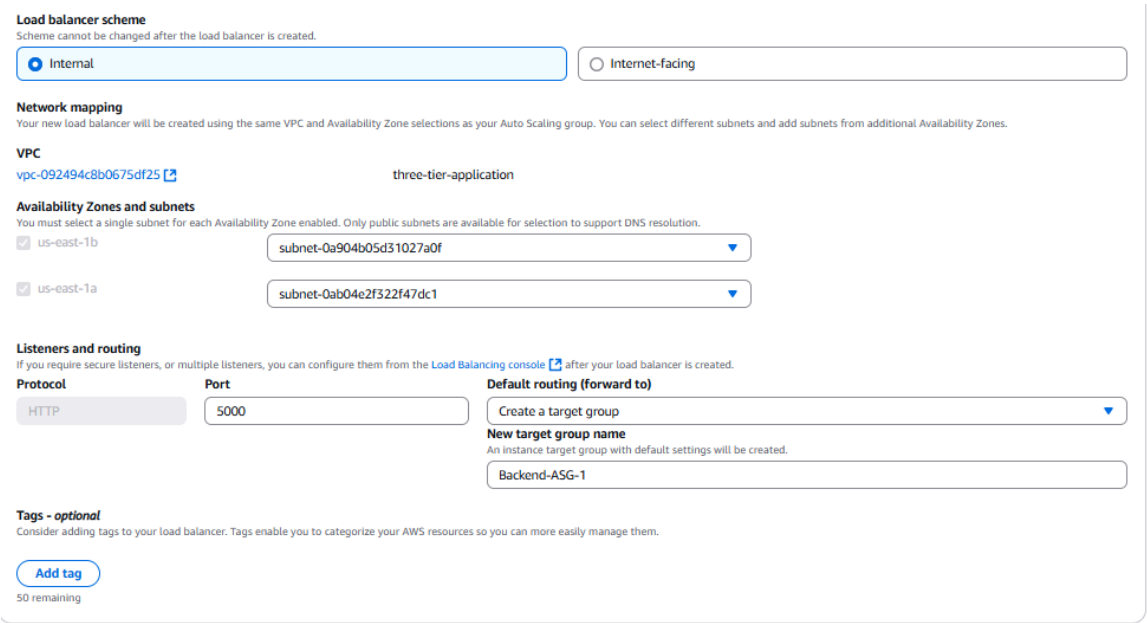
15. Choose the Network. Select the **'three-tier-application'** VPC**.** Choose two private **PrivateAppTierSubnetAZ1** and **PrivateAppTierSubnetAZ2.**



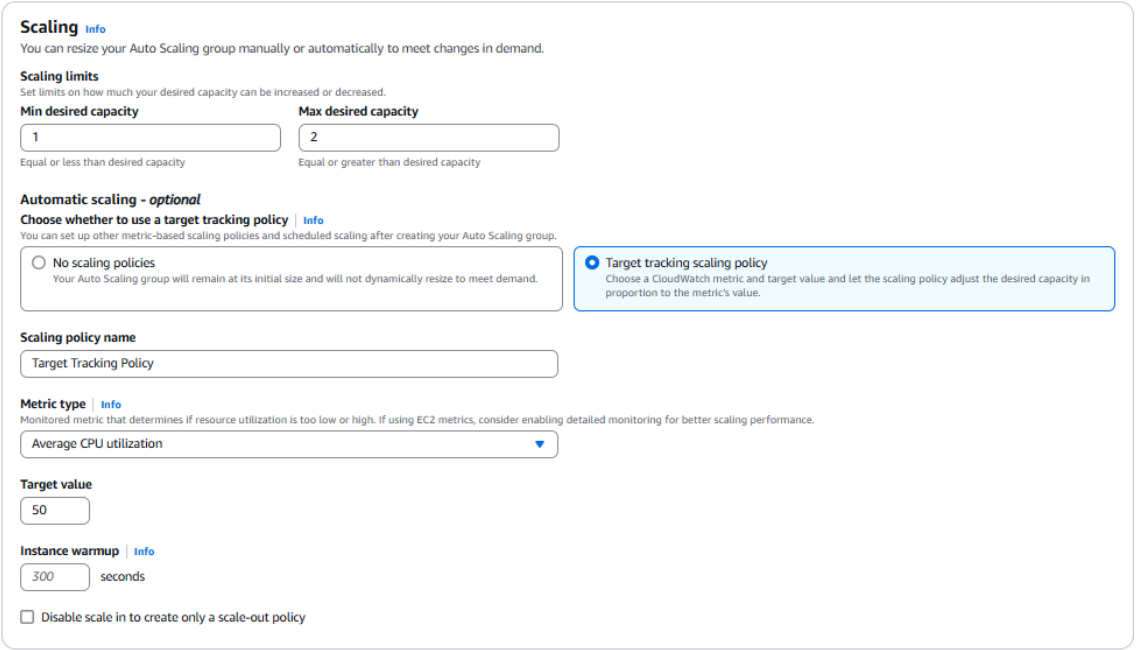
16. Add Health Check & Load Balancer. Create and attach a **new load-balancer**



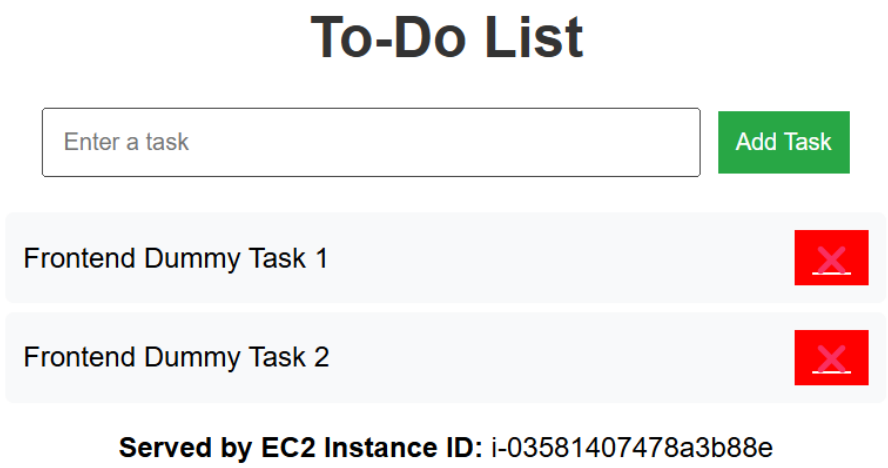
17. Select **'Internal load-balancer'** & **'private'** subnets, under port type **'5000'.** Under default routing select **'Create a target group'**



18. Update the scaling configuration by setting the **'max desired capacity' to '2'** and selecting the **'target tracking scaling policy'** under 'Automatic scaling' options:



19. **Review** & **create** autoscaling group.
20. Stress test the EC2 instance via SSM login to webserver, use the below command on primary webserver created by autoscaling group this will trigger the second instance in ASG to launch:
    1. python3 -c "while True: x = [i**2 for i in range(1000000)]"
21. Wait for the **second instance** to spin up by ASG, Observe the different instance ID on the webpage by accessing the webpage via Loadbalancer, refresh the page to see different instance id's



To-Do List

Enter a task

Add Task

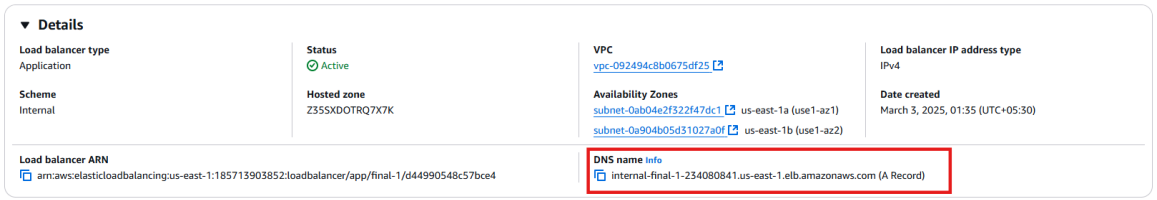
Frontend Dummy Task 1

Frontend Dummy Task 2

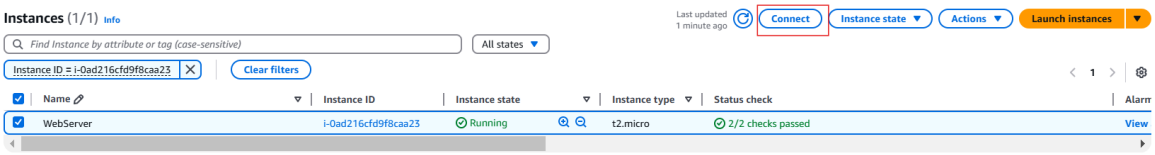
Served by EC2 Instance ID: i-0dbee27b5a9a4c635

Step 8: Create Load Balancer and Autoscaling for Frontend

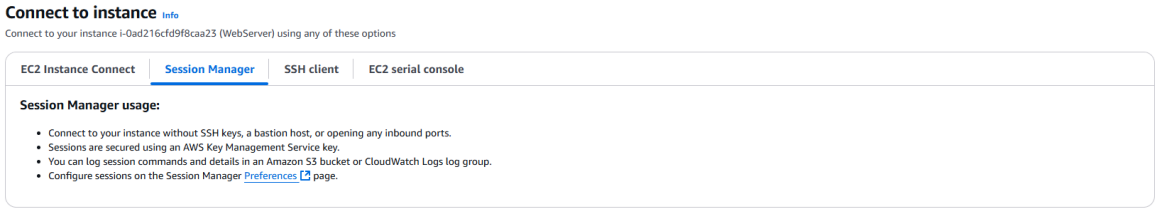
1. Navigate to **load balancer** in console and make sure internal load balancer is in Available state. Copy its **DNS name**.



2. Navigate to EC2 Instance and choose **Webserver.** Select **Connect.**



3. Select **Session Manager** tab and click on **Connect**.



4. Execute following steps:
   b. sudo su ec2-user
   c. cd ~
   d. cd flaskapp/frontend/

```
sh-4.2$ sudo su ec2-user
[ec2-user@ip-10-0-0-243 bin]$ cd ~
[ec2-user@ip-10-0-0-243 ~]$ cd flaskapp/frontend/
[ec2-user@ip-10-0-0-243 frontend]$ sudo vim frontend.py
[ec2-user@ip-10-0-0-243 frontend]$
```

   e. sudo vim frontend.py
   f. Press the **i** key to enter insert mode and replace the **BACKEND_API** variable with the endpoint of the **LoadBalancer** in the format of http://<loadbalancer_endpoint> 5000/

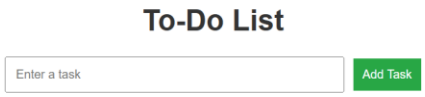
```
from flask import Flask, render_template, request, redirect, url_for
import requests

app = Flask(__name__)

# ✅ Backend API URL - Replace with your actual backend DNS or IP
BACKEND_API = "http://internal-final-1-234080841.us-east-1.elb.amazonaws.com:5000"  # Update this with Load Balancer later
```

   g. Press the **Esc** key, type :**wq!**, and hit **Enter** to save and exit.
   h. sudo fuser -k 80/tcp
   i. sudo nohup python3 frontend.py --port 80 &

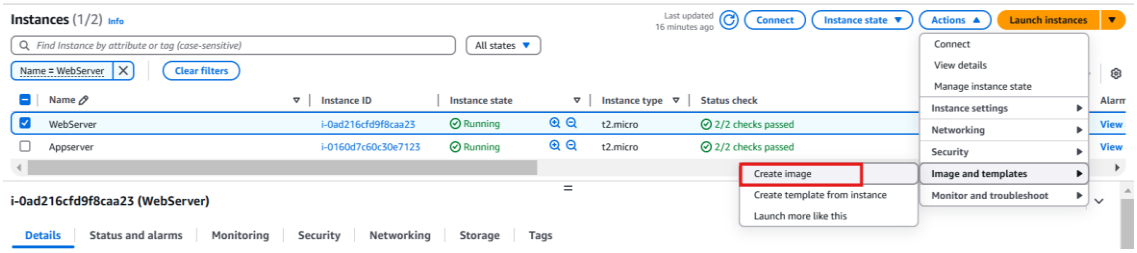
```
[ec2-user@ip-10-0-4-40 backend]$ sudo nohup python3 backend.py --port 5000 &
[1] 1169
[ec2-user@ip-10-0-4-40 backend]$ nohup: ignoring input and appending output to 'nohup.out'

[ec2-user@ip-10-0-4-40 backend]$
```

   j. Hit **Enter** twice.

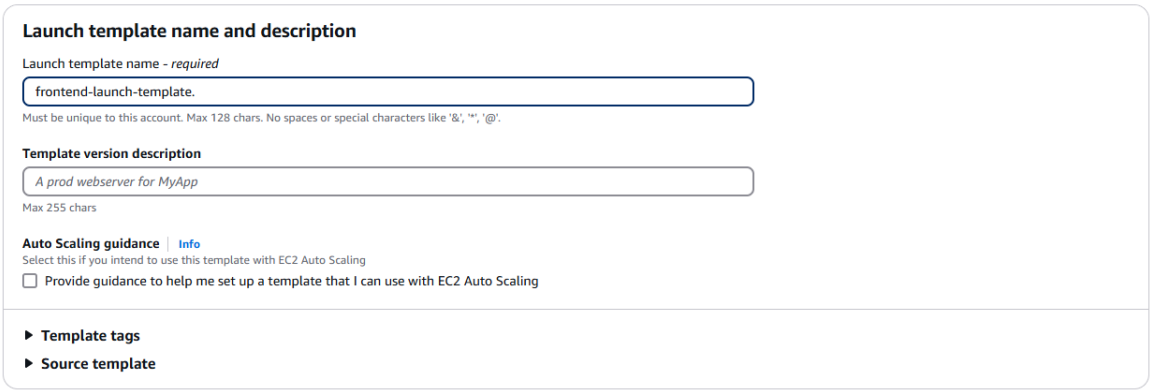
13. Open your browser and search for http://<**public_id_of_webserver**>:80. The screen below should appear.



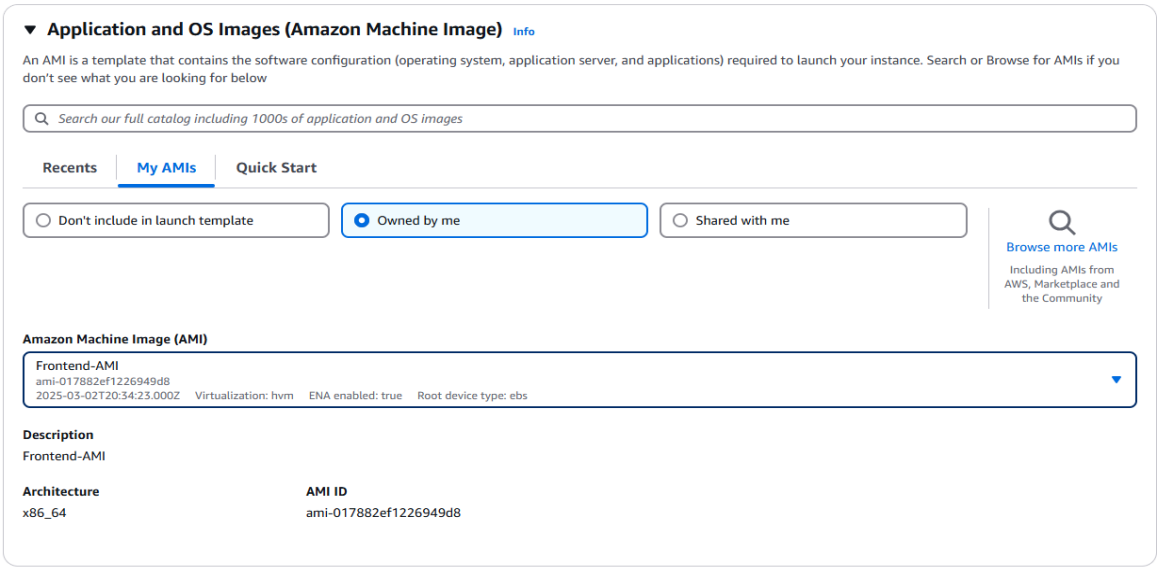
14. Create Image for Autoscaling group, Click on actions & select **'Image & template'** create an image using the instance. Provide the **Image name** as **Frontend-AMI** and **Create Image**. Make sure its status appears **Available.**



15. Navigate to EC2 Console. Click Launch Templates in the left sidebar and click on create a **New Launch Template.** Enter Launch template name **frontend-launch-template.**

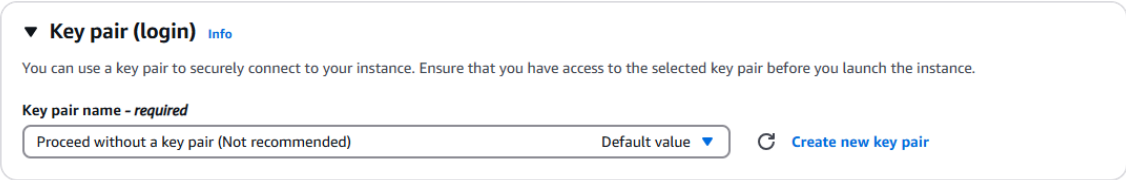


16. Under **Amazon Machine Image (AMI)**, select **'Frontend-AMI'** created above

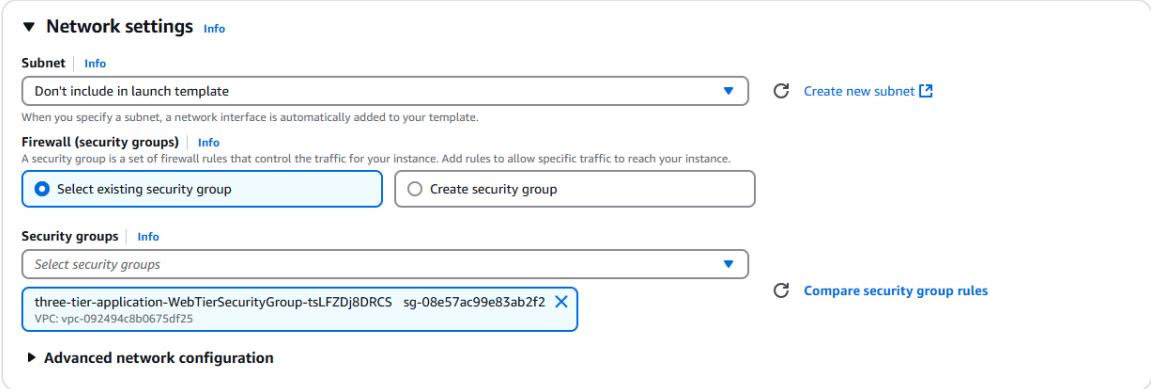


17. Choose an **Instance Type** (e.g., ‘t2.micro’)

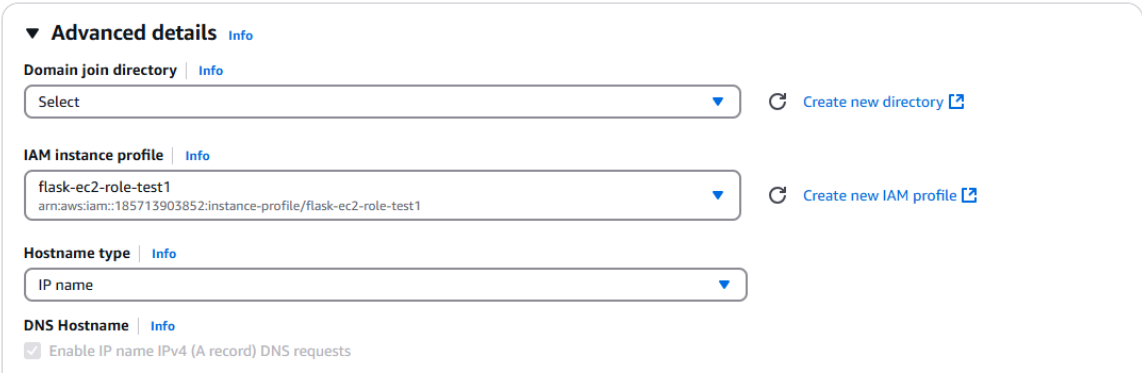
18. Under Key pair (login), **Proceed without a key pair**.



19. Keep default network settings as **‘don’t include in launch template’.** For security group select **three-tier-application-WebTierSecurityGroup.**



20. Under Advanced Details section, for **IAM instance profile** select **flask-ec2-role-<your_user_id>** created in step-1.



21. Scroll down to **User Data** and copy contents from this Link.

User data - *optional* | Info

Upload a file with your user data or enter it in the field.

Choose file

```
#!/bin/bash
yum update -y
yum install -y python3 python3-pip git
pip3 install flask flask-cors mysql-connector-python requests pymsql
cd /home/ec2-user/flaskapp/frontend/
python3 /home/ec2-user/flaskapp/frontend/frontend.py
```

22. Click on **Create Launch Template.**
23. Open the **AWS Auto Scaling Console.** Click **Create an Auto Scaling group**
24. Enter ASG Name **frontend-asg.** Under **Launch Template**, select the one you created **frontend-launch-template**

Name

Auto Scaling group name

Enter a name to identify the group.

frontend-asg

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

Launch template Info

Switch to launch configuration

Launch template

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

frontend-launch-template

Create a launch template

Version

Default (1)

Create a launch template version

Description -	**Launch template** frontend-launch-template lt-0575f5ca7e16222dc	**Instance type** t2.micro
AMI ID ami-017882ef1226949d8	**Security groups** -	**Request Spot Instances** No
Key pair name -	**Security group IDs** -	

25. Choose the Network. Select the **'three-tier-application'** VPC**.** Choose two private **PublicSubnetAZ1** and **PublicTierSubnetAZ2.**

Network Info

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

VPC

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

vpc-092494c8b0675df25 (three-tier-application)
10.0.0.0/16

Create a VPC

Availability Zones and subnets

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

Select Availability Zones and subnets

us-east-1a | subnet-066d4848342aecb96 (PublicSubnetAZ1)
10.0.0.0/24

us-east-1b | subnet-0fa0ab3ca9af653ee (PublicSubnetAZ2)
10.0.1.0/24

Create a subnet

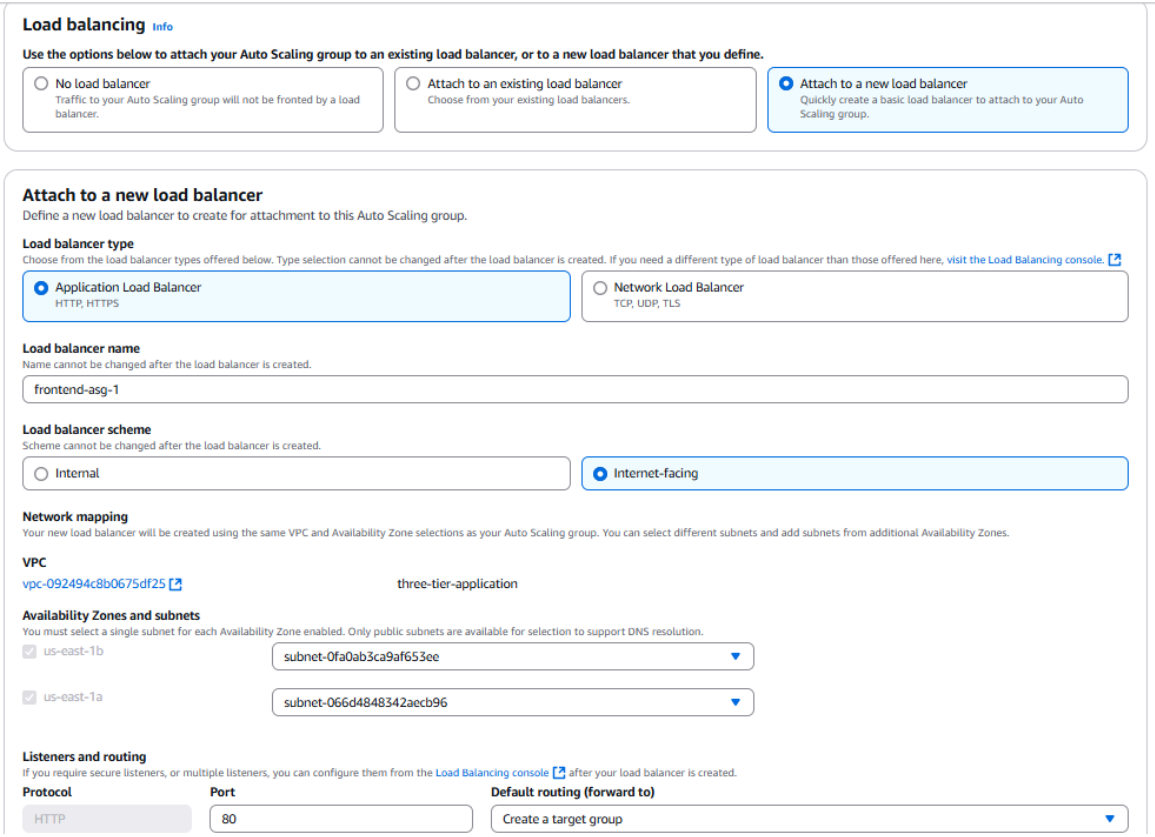
Availability Zone distribution - *new*

Auto Scaling automatically balances instances across Availability Zones. If launch failures occur in a zone, select a strategy.

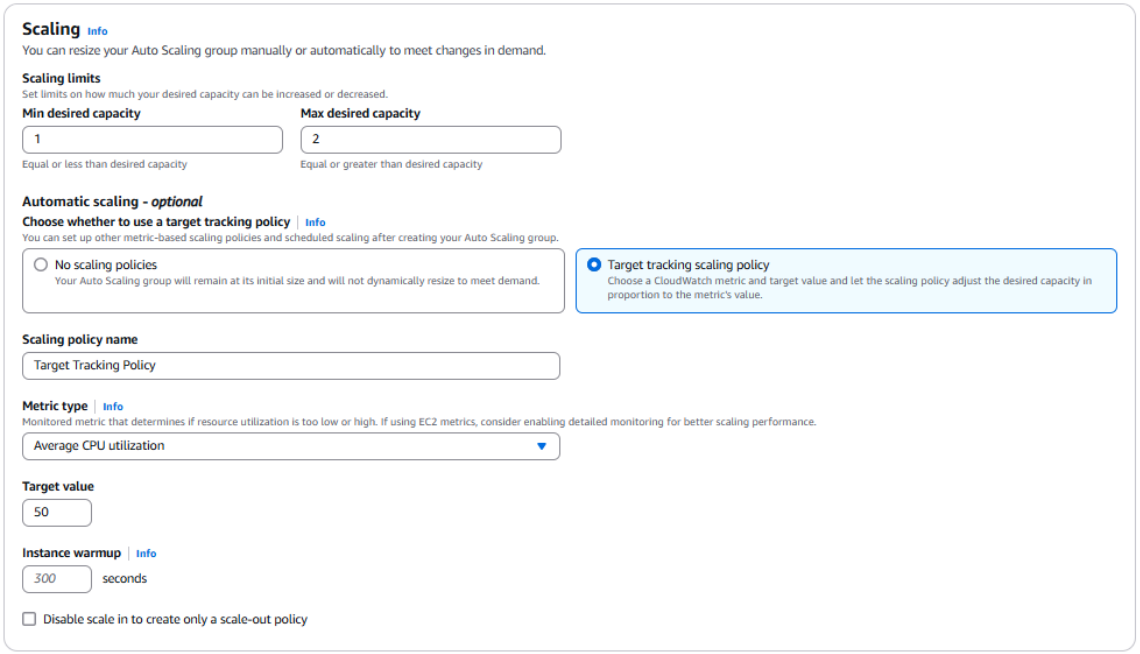
- (●) Balanced best effort — If launches fail in one Availability Zone, Auto Scaling will attempt to launch in another healthy Availability Zone.
- ( ) Balanced only — If launches fail in one Availability Zone, Auto Scaling will continue to attempt to launch in the unhealthy Availability Zone to preserve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

Cancel | Skip to review | Previous | Next

26. Add Health Check & Load Balancer. Create and attach a **new load-balancer**
27. Select **'Internet facing load-balancer'** & **'public'** subnets, under port type **'80'.** Under default routing select **'Create a target group'**



28. Update the scaling configuration by setting the **'max desired capacity' to '2'** and selecting the **'target tracking scaling policy'** under 'Automatic scaling' options:



29. **Review** & **create** autoscaling group.
30. The application now can be accessed using DNS name of the new **frontend load balancer.**