🧪 Part 3: Simulate Load (Crash Servers)

For Server 1 (Using Bitvise SSH Client)

1. Copy the **Public IPv4 Address** of one EC2 instance.
2. Open **Bitvise SSH Client**:
   - **Host**: Paste IP address
   - **Username**: `ubuntu`
   - **Authentication method**: Public key
   - Load your `.pem` key file in **Client Key Manager**
   - Click **Login**
3. In Terminal:
   - Create an infinite loop script:
   - `sudo nano infi1.sh`

Add:

```
#!/bin/bash
while true
do
echo "Looping forever"
# Additional commands can be added here
done
```

   - Save the file, then run:
   - `sudo chmod +x infi1.sh`
   -
   `./infi1.sh`

For Server 2 (AWS Connect Terminal)

1. Go to EC2 → Select second instance → Click **Connect**
2. In terminal, repeat same steps:

```
3. sudo nano infi1.sh
4. sudo chmod +x infi1.sh
5. ./infi1.sh
```

📊 Monitor CPU Utilization

- Go to **CloudWatch** or EC2 Monitoring tab.
- As the CPU load increases due to infinite loops, new instances will automatically be launched by Auto Scaling group to handle the load.

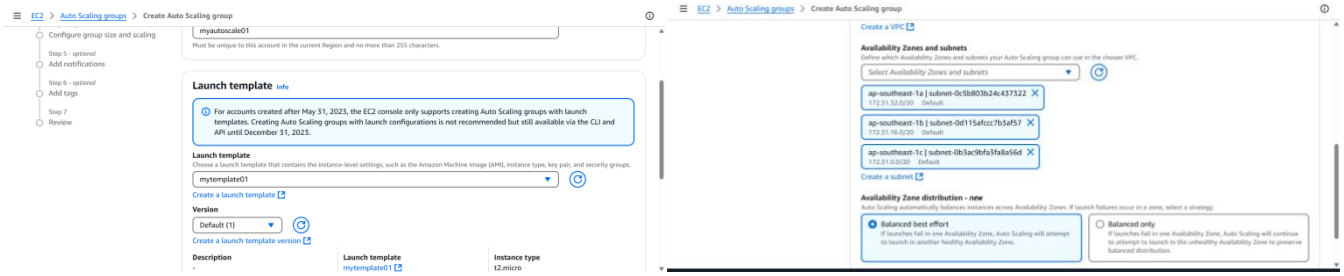
🗑️ Part 4: Clean Up Resources

Delete in This Order:

1. **Auto Scaling Group** → Actions → Delete
2. **Load Balancer** → Actions → Delete
3. **Target Group** → Actions → Delete
4. **Launch Template** → Actions → Delete
5. **Running EC2 Instances** → Select → Instance state → Terminate

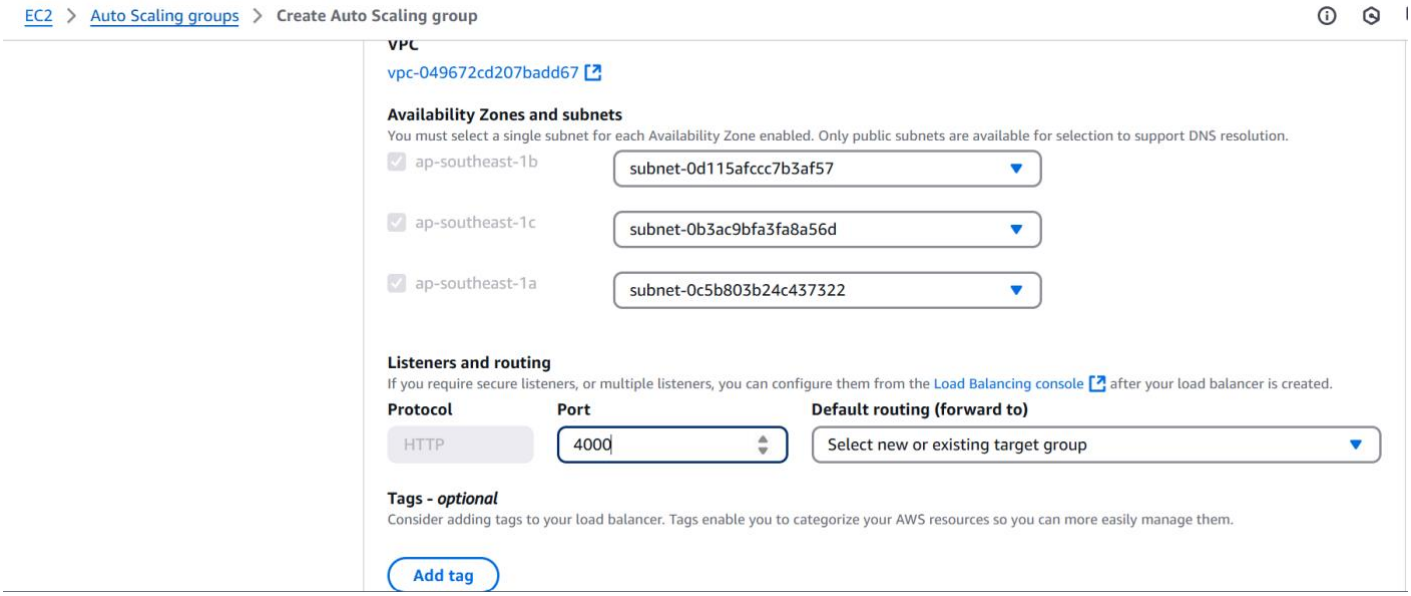
4. Select Network Options

- **Availability Zones and Subnets**: Select **all available subnets** to ensure high availability.
- Click **Next.**



5. Attach Load Balancer

- In **Load Balancing** section:
  - Choose **"Attach to a new load balancer"**
  - **Load balancer scheme**: `Internet-facing`
  - **Listener Port**: `4000`
  - **Default routing**: Select the created Auto Scaling Group.



6. Configure Group Size

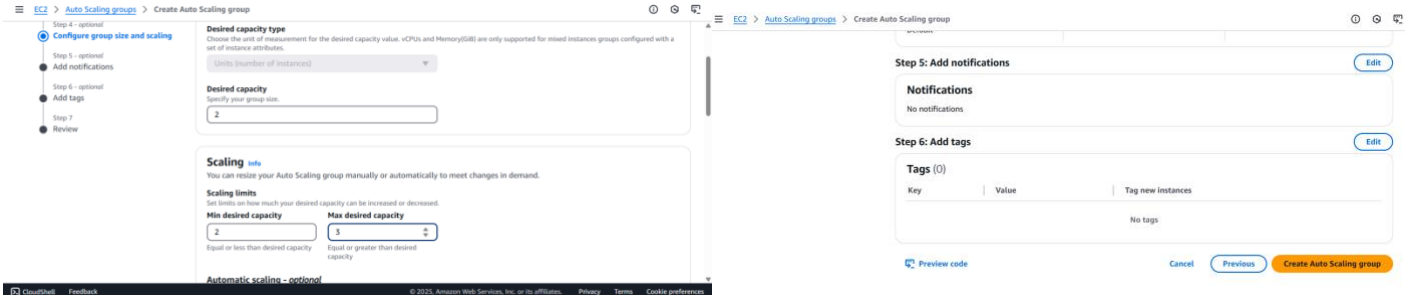
- **Desired Capacity**: 2
- **Minimum Capacity**: 2
- **Maximum Capacity**: 3

7. Set Scaling Policy

- Choose **"Target tracking scaling policy"**
- Set **Target value**: `300` (this refers to CPU utilization target)

8. Review and Create

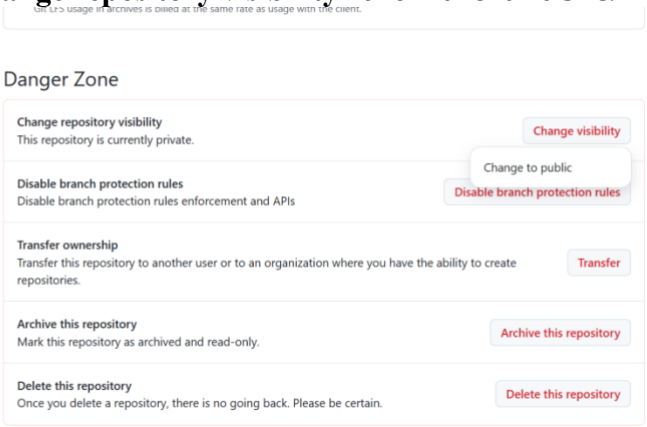
- Click **Next** and then **Create Auto Scaling Group**



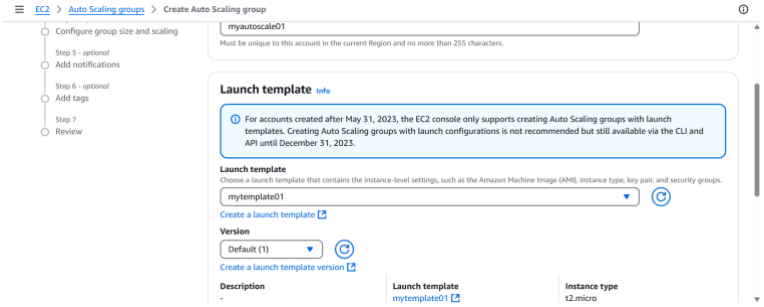
🎁 Make Sure GitHub Repository is Public

If your Node.js GitHub repo is private:

- Go to **GitHub → Repo Settings → Scroll down to "Danger Zone".Assignment 11**
- Click **"Change repository visibility"** and make it **Public**.



⚙️ Part 2: Create an Auto Scaling Group



1. Go to Auto Scaling Groups

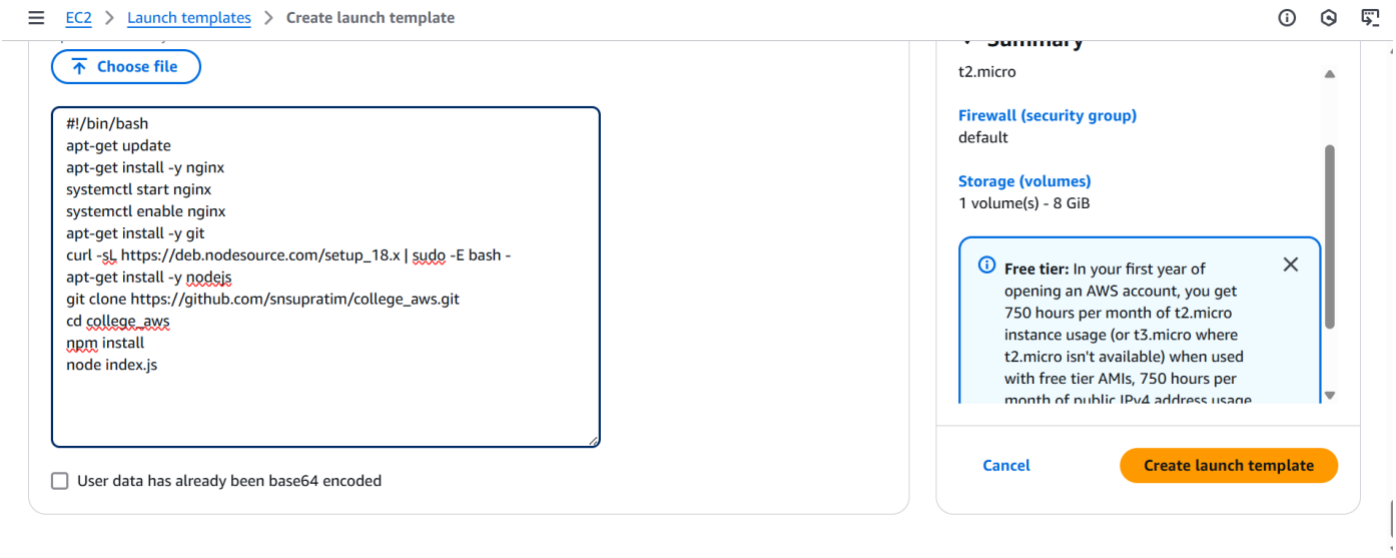
- In EC2 Dashboard → Click **"Auto Scaling Groups"**
- Click **"Create Auto Scaling Group"**

2. Configure Group Settings

- **Name**: e.g., `myautoscale1`
- **Launch Template**:
  - Choose the launch template created earlier (e.g., `mytemplate1`)
  - Select **Latest version (1)**

3. Add User Data (Startup Script)

Paste the following script into **"User data"** section:



ASSIGNMENT 11 :
SUPRATIM NAG (CSE-AIML/22/57)

Build Scaling Plans in AWS that Balance Load on Different EC2 Instances

Objective:

To create an Auto Scaling environment with EC2 instances and Elastic Load Balancer (ELB) that automatically handles traffic and manages high availability using a GitHub repo with a Node.js app

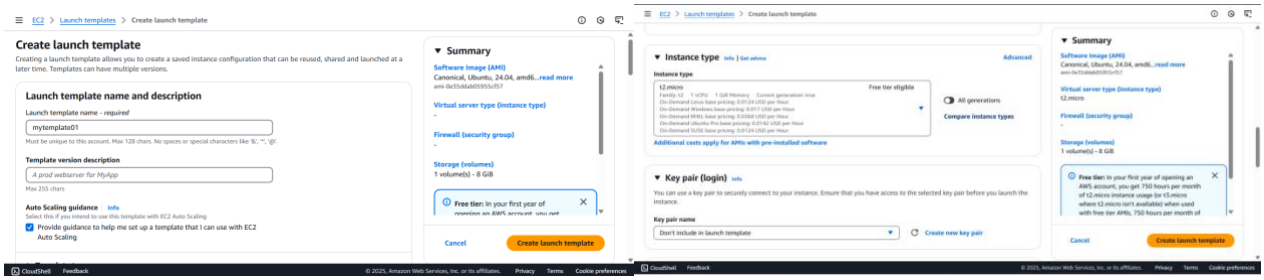
🔧 Part 1: Create a Launch Template

1. Go to EC2 Dashboard

- Log in to your AWS Management Console.
- Navigate to **EC2** from the Services.

2. Create a Launch Template

- On the left sidebar, click **"Launch Templates"**.
- Click **"Create Launch Template"**.
  - **Template name**: e.g., `mytemplate1`
  - **Template version description**: Optional
  - **Check the box** for **"Provide guidance to help me set up a template that can be used with EC2 Auto Scaling"**



3. Configure Launch Template

- **Amazon Machine Image (AMI)**: Select **Ubuntu**.
- **Instance Type**: Choose **t2.micro**.
- **Key Pair**:
  - If you already have one, select it.
  - If not, click **"Create a new key pair"**, download the `.pem` file.

4. Network Settings

- Under **Network settings**, select the **Security Group** you previously created (e.g., `Mytemp1`)