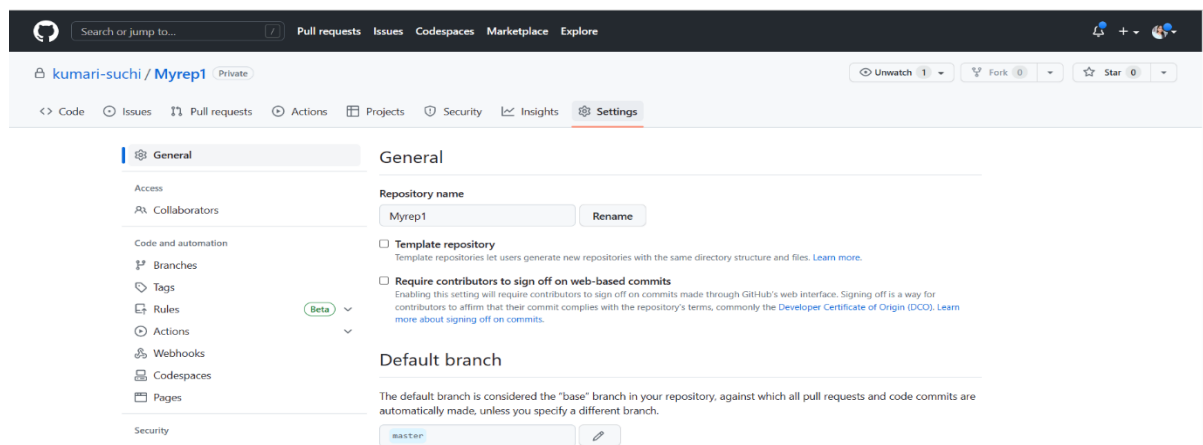
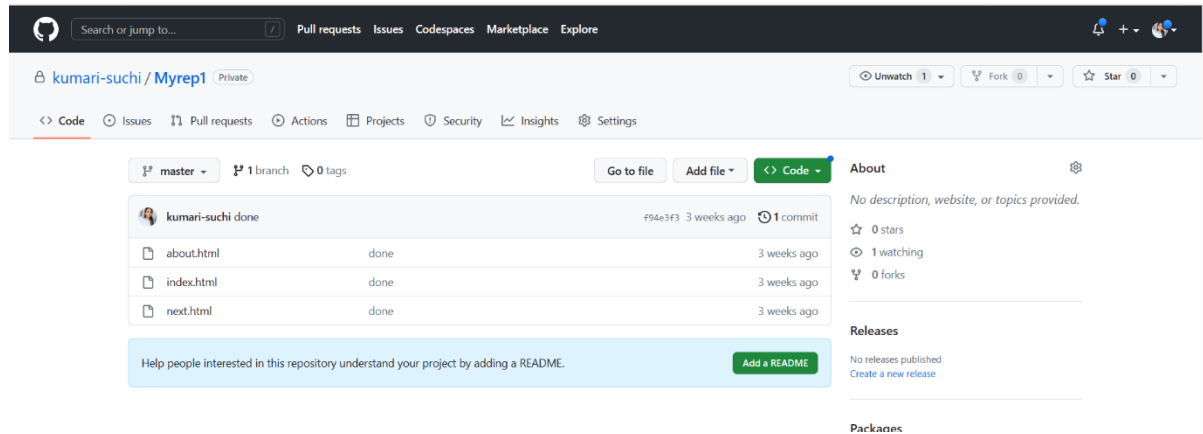


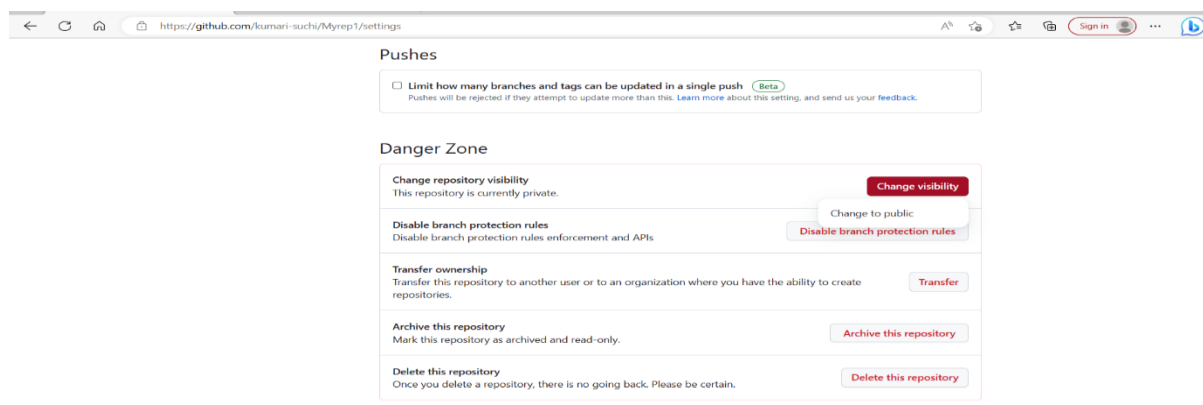
ASSIGNMENT-11

Problem Statement :Build scaling plans in AWS that balance load on different EC2 instances.

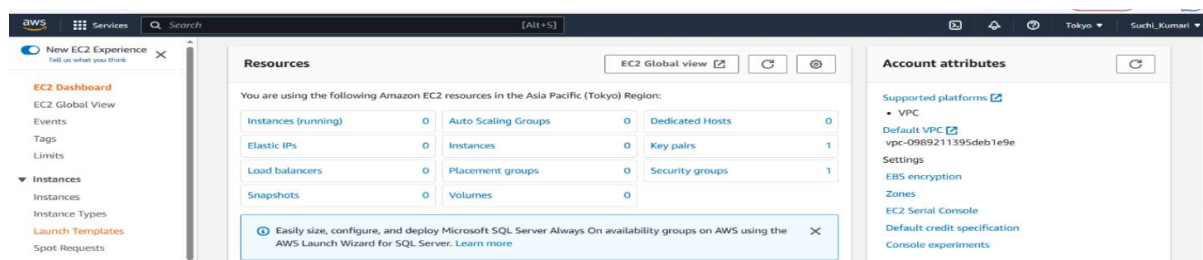
- 1.Sign in to your GitHub account.
- 2.Then go to your repository . Make sure the Repository which will be cloned is made public or not.
- 3.For public repository first go to settings.



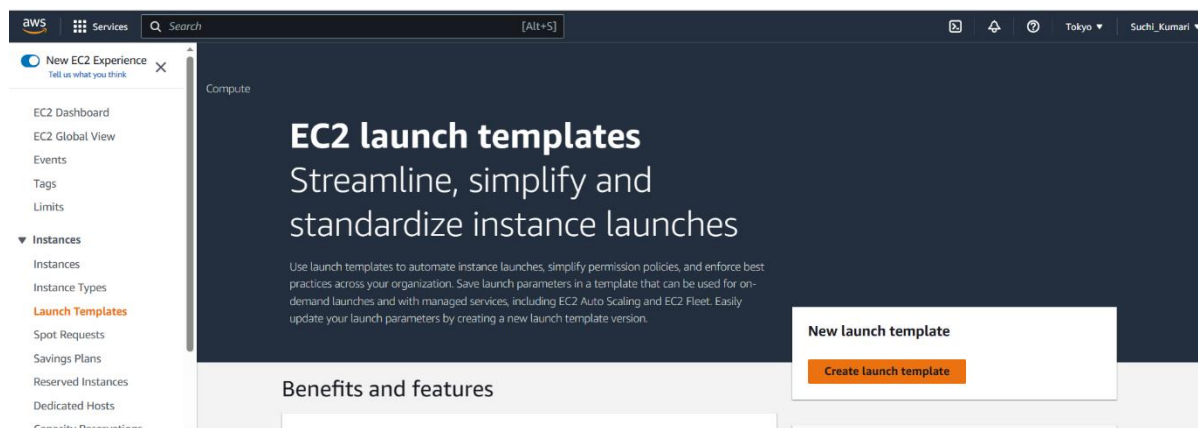
- 4.Then scroll down and go to Danger Zone then click on change visibility and change to public.



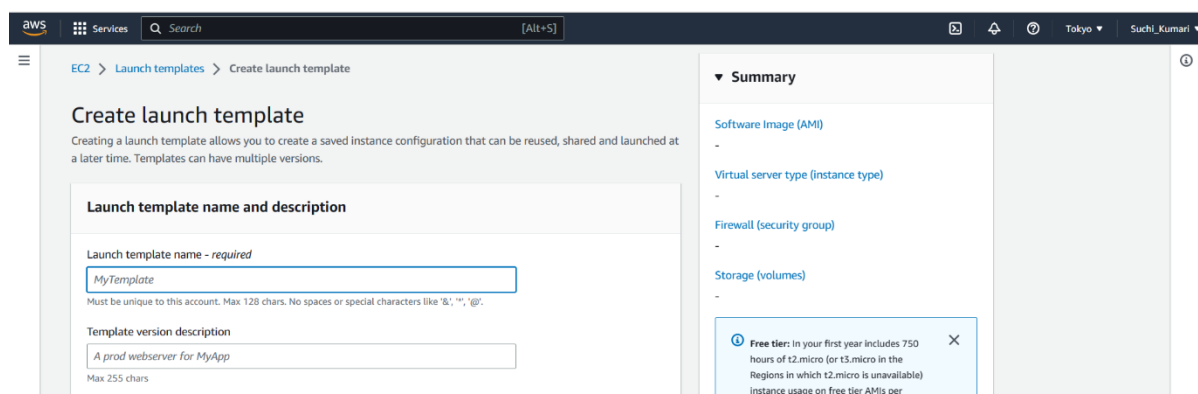
5. Then sign in to aws account then click EC2.Then go to EC2 Dashboard and then click on Launch Templates.



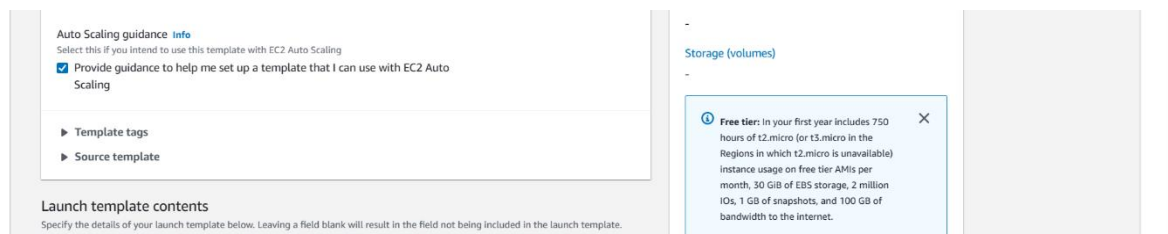
6. Then click on Create Launch template.



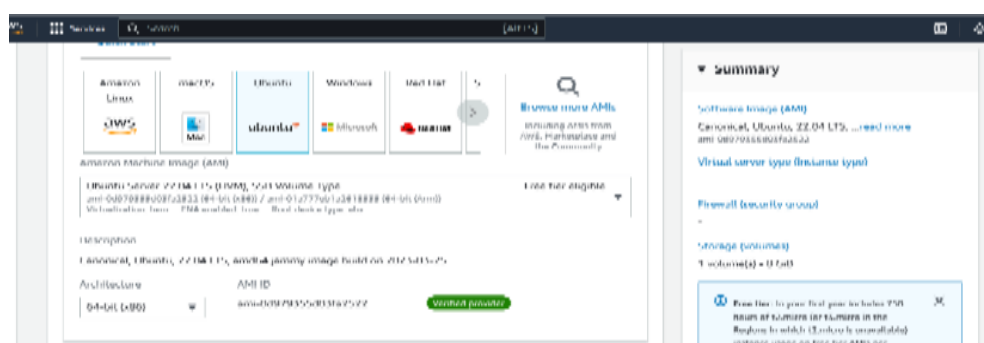
7. Then enter Launch template name and Template version description.



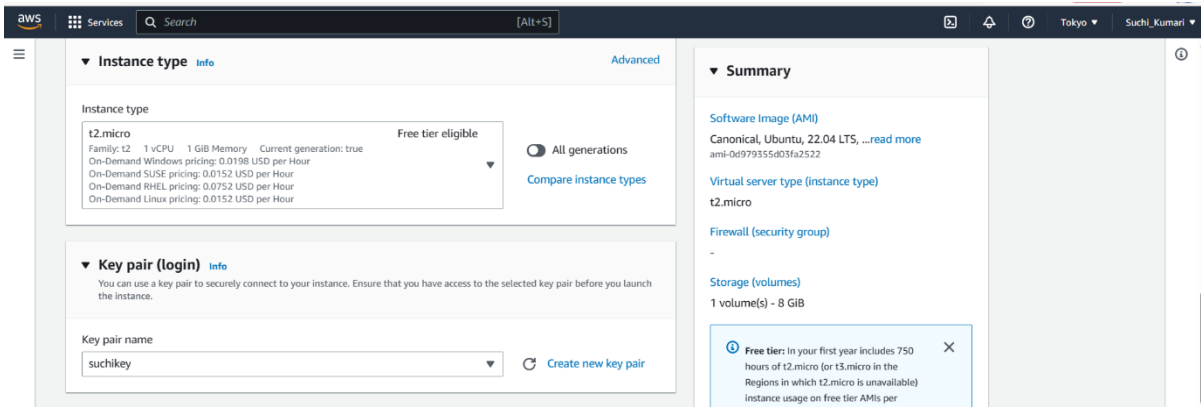
8. Then Check the "Provide Guidance" box.



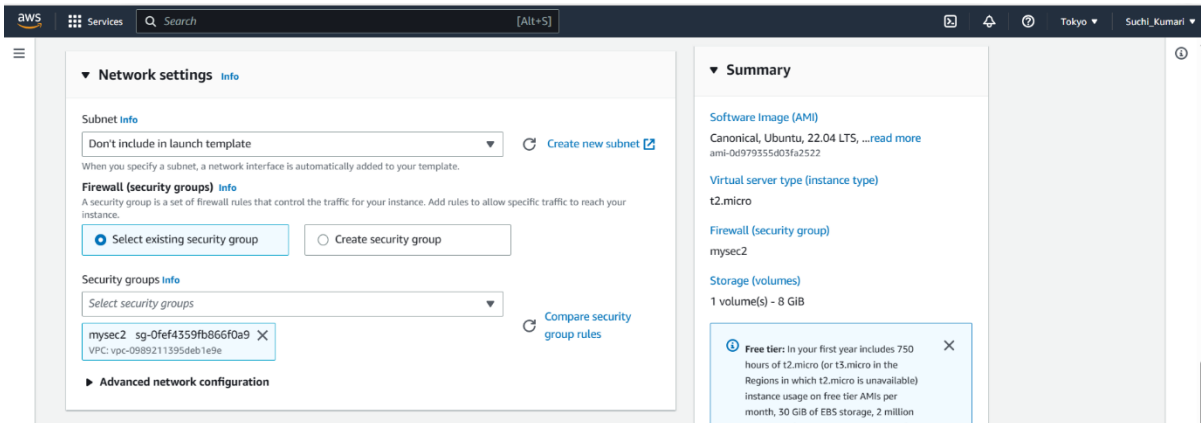
9. Then click on Ubuntu.



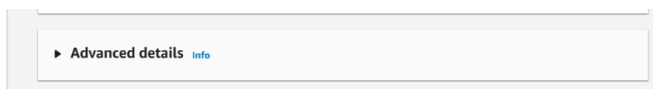
10. Under Instance type select t2.micro type of configuration. Then enter key pair name.



11. Then in Network settings select existing security group then select Security groups.



12. Then click on Advanced details.



13. scroll down and then in User data write some command

```
#!/bin/bash

apt-get update

apt-get install -y nginx

systemctl start nginx

systemctl enable nginx

apt-get install -y git

curl -sL https://deb.nodesource.com/setup_18.x | sudo -E bash -

apt-get install -y nodejs

git clone YourRepositoryURLhere

cd YourRepositoryNamehere/

npm install

node index.js

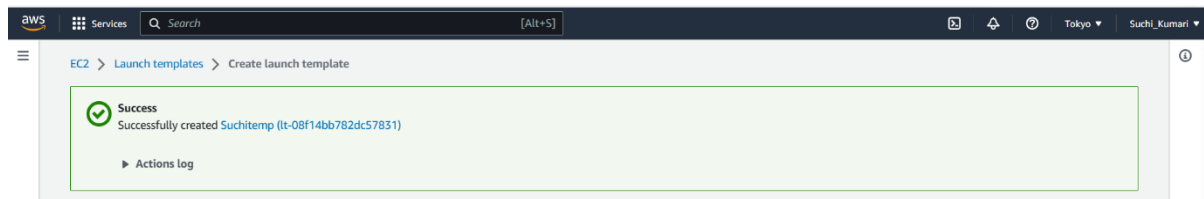
Then click on Launch Instance.
```

User data - optional [info](#)

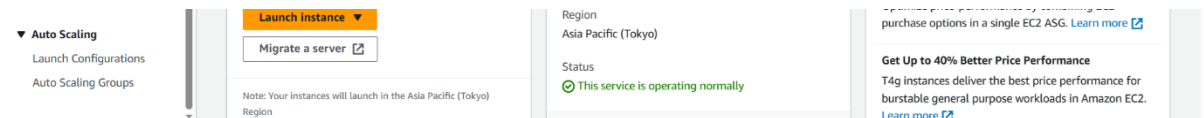
Enter user data in the field.

```
#!/bin/bash
apt-get update
apt-get install -y nginx
systemctl start nginx
systemctl enable nginx
apt-get install -y
curl -sL https://deb.nodesource.com/setup_18.x | sudo -E bash -
apt-get install -y nodejs
git clone githttps://github.com/kumari-suchi/Myrep1.git
cd Myrep1
npm install
node index.js
```

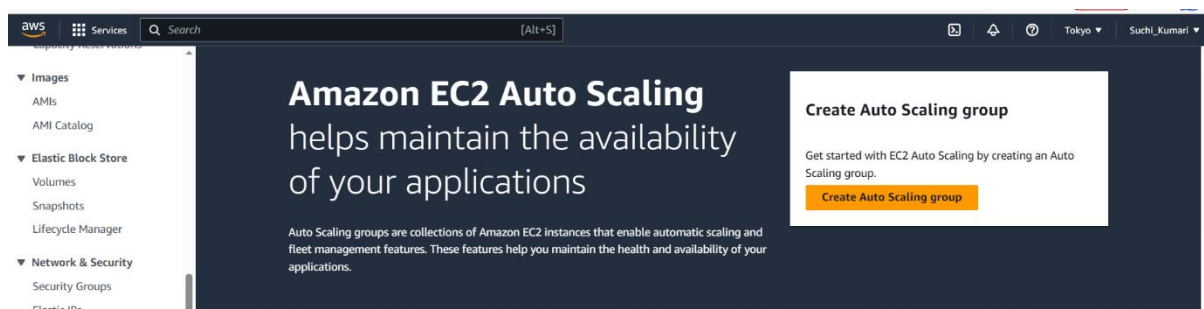
14. Then successfully created Launch templates.



15. Then again go to EC2 Dashboard the click on Auto Scaling and then Auto Scaling Groups.



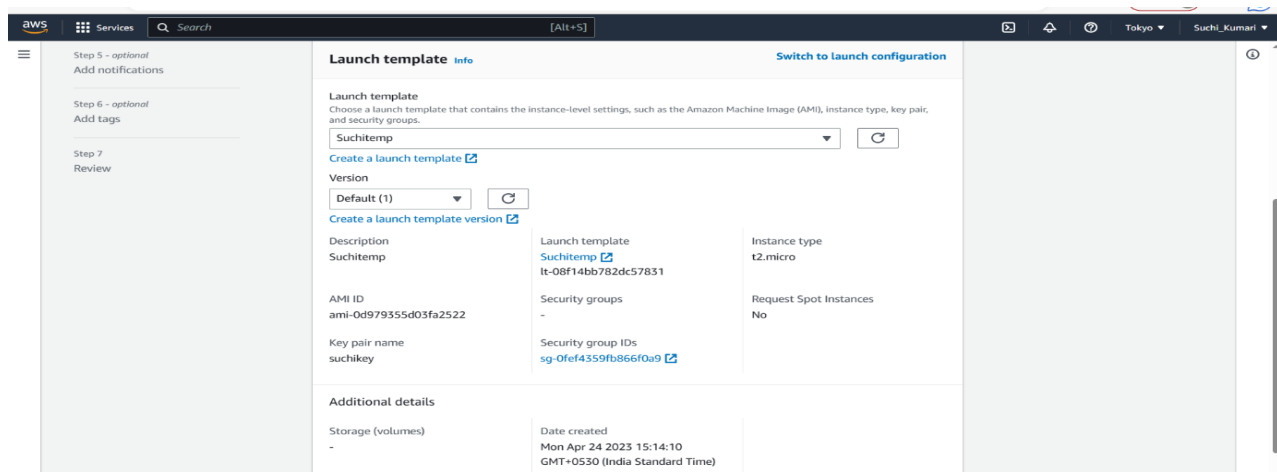
16. Then click on Create Auto Scaling group.



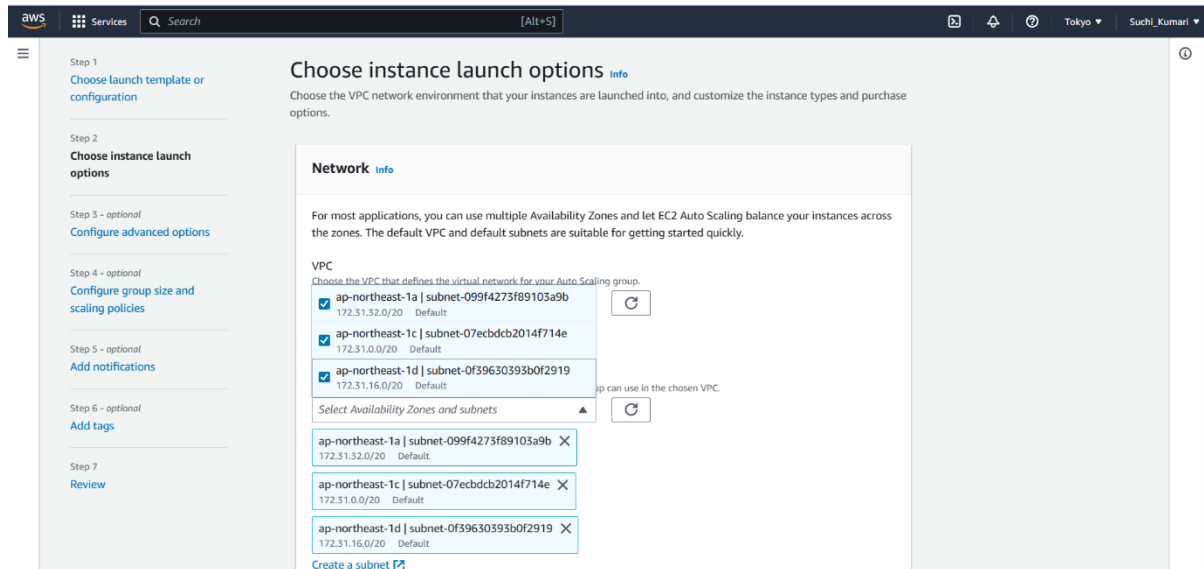
17. Then enter Auto Scaling group name.



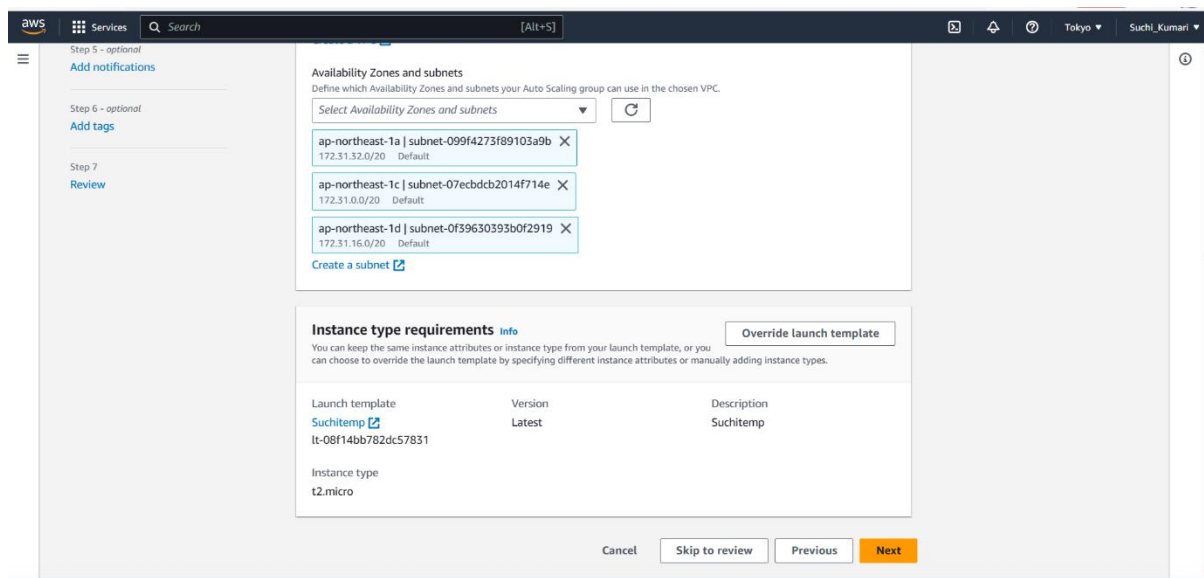
18. Then select Launch template which is you have created then in version select Default(1). Then click on next.



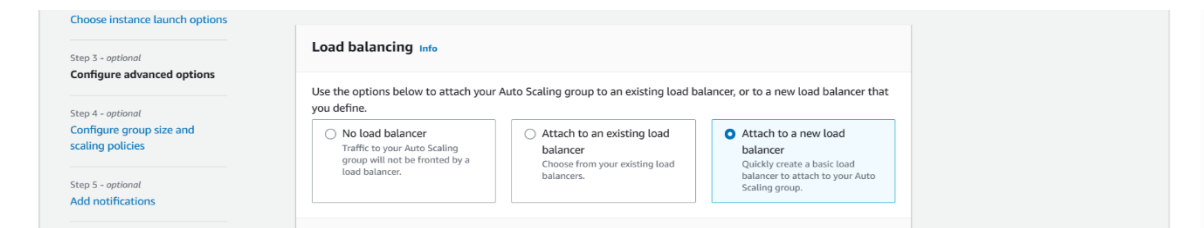
19. After that, Under Availability Zones and Subnets select all the zones that appear.



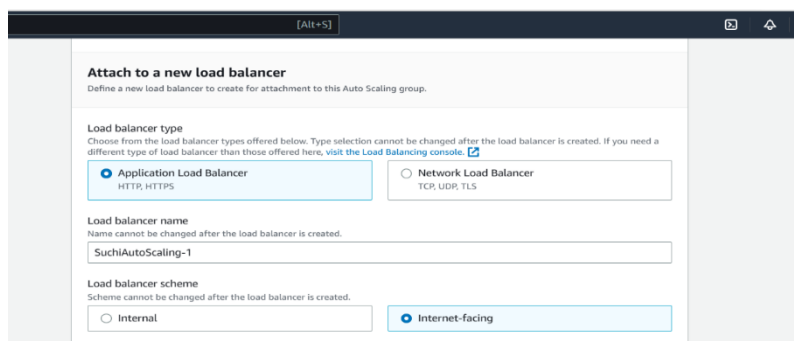
20. Then click on next.



21. Now Under Load Balancers select the Attach to a New Load balancer option.



22. Now select Internet-Facing under Load balancer scheme.



23. Under Listeners and Routing enter the port no. of the project and select Create target group followed by giving the target group a name

Listeners and routing

If you require secure listeners, or multiple listeners, you can configure them from the [Load Balancing console](#) after your load balancer is created.

Protocol: HTTP Port: 4000

Default routing (forward to): Create a target group

New target group name: SuchiAutoScaling-1

Tags - optional

Consider adding tags to your load balancer. Tags enable you to categorize your AWS resources so you can more easily manage them.

Add tag

50 remaining

24. Now click on the next button. After clicking on the Next button, a new page will open. Under Group Size mention:

- Desired Capacity = 2
- Minimum Capacity = 2
- Maximum Capacity = 3

Configure group size and scaling policies - optional

Set the desired, minimum, and maximum capacity of your Auto Scaling group. You can optionally add a scaling policy to dynamically scale the number of instances in the group.

Group size - optional

Specify the size of the Auto Scaling group by changing the desired capacity. You can also specify minimum and maximum capacity limits. Your desired capacity must be within the limit range.

Desired capacity: 2

Minimum capacity: 2

Maximum capacity: 3

25. Now under Scaling policies Choose the Target Tracking Scaling policy option.

- Select the metric type as Average CPU utilization.
- Set Target Value to 50.
- Set Warm-Up time to 300 seconds under Instances Need then click on next.

Scaling policies - optional

Choose whether to use a scaling policy to dynamically resize your Auto Scaling group to meet changes in demand.

Target tracking scaling policy

Scaling policy name: Target Tracking Policy

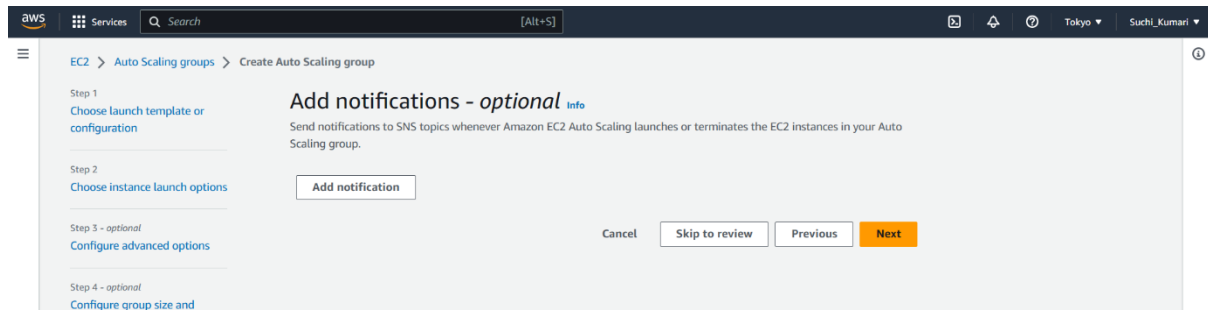
Metric type: Average CPU utilization

Target value: 50

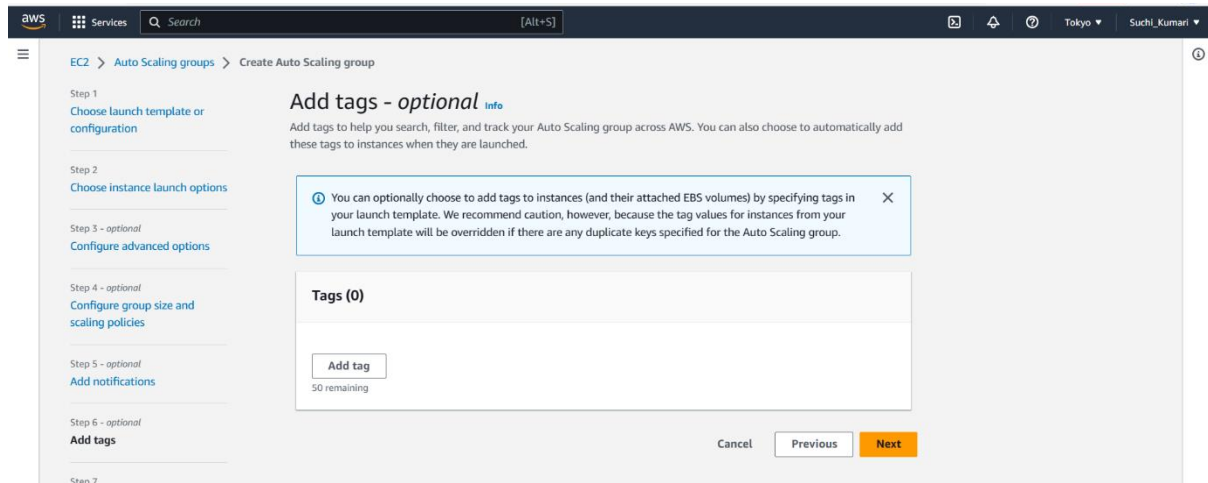
Instances need: 300 seconds warm up before including in metric

Disable scale in to create only a scale-out policy

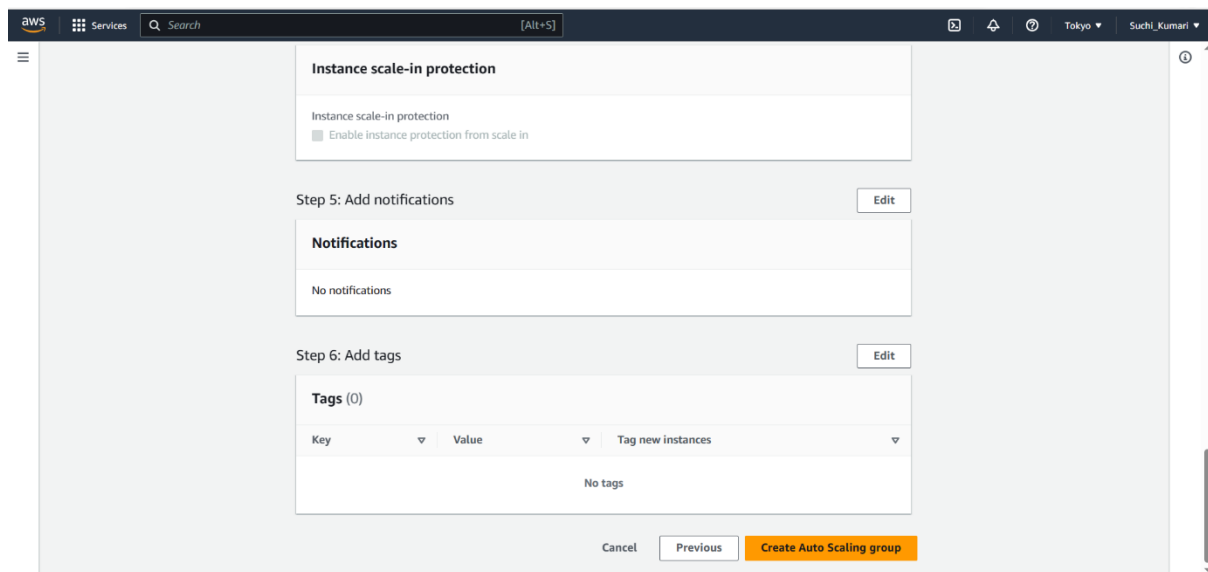
26. Then click on next.



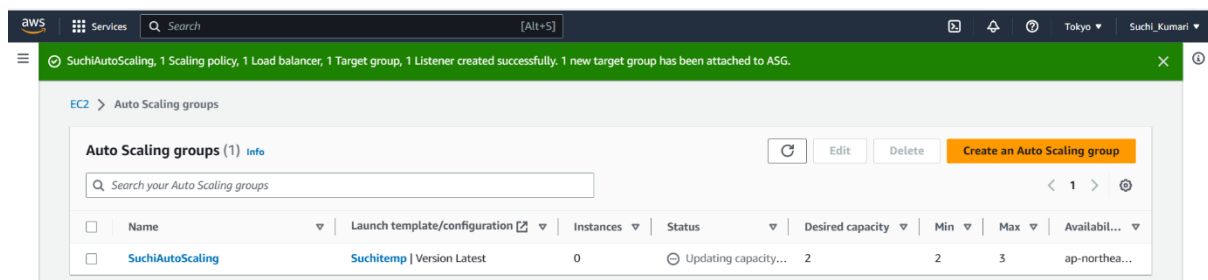
27. Then again click on next.



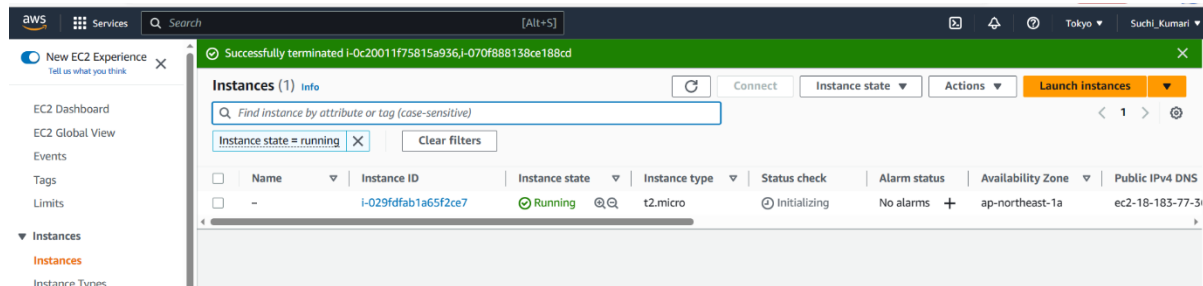
28. Then click on Create Auto Scaling group.



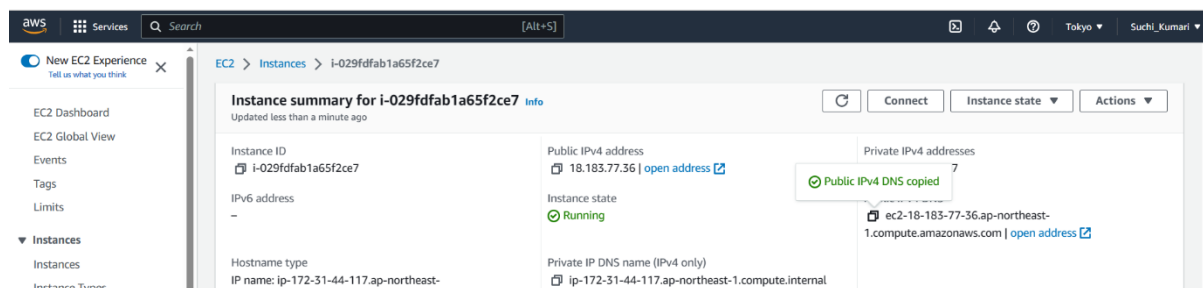
29. Then successfully Auto Scaling group created.



30. Return to the Instances Page using the Left side Nav bar. After some you will notice a new instance server will appear automatically! To help finding it more easily we need to activate the instance running filter. Click on the search box below the Instances Section Heading. Start typing running. Select the option "Instance state = running" option in the suggestion dropdown. The filter will be activated. After some few seconds of refreshing we will be able to see two new servers are running. Then click on Instance ID.



31. Then Copy its public IPV4 DNS.



32. Paste it in another browser.



33. Now to access our project webpage we need to append the port no. (4000) of our project with a ":"

Hello, I am Suchi Kumari

34. We will now OVERLOAD THE SERVER INSTANCES by running scripts and increasing CPU utilization value above the threshold that we specified during Configuration of the Auto scaling group.

35. For it we will use:

- Use Bitwise SSH client for instance 1.
- Use direct connect terminal in AWS for instance 2

36. For Instance-1:

- Copy the public IPV4 address

- Open Bitwise SSH client.
- Paste the IP and select/specify the necessary options.
- Now Log-In to your server.
- Open the new Terminal.
- Now enter the command:
 - nano su1.sh
- After the command a new nano Editor window will open. Type the following in it.

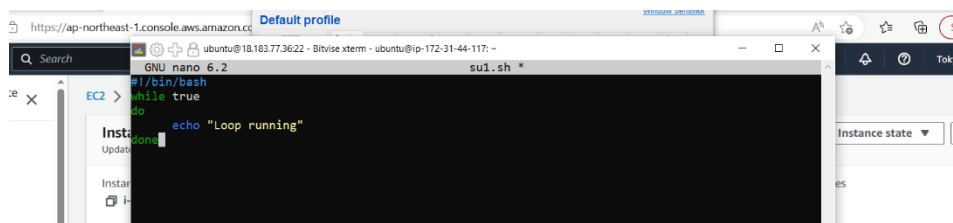
```
#!/bin/bash
```

```
while true
```

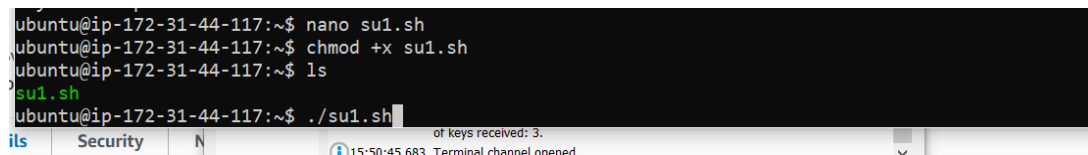
```
do
```

```
    echo "Loop running"
```

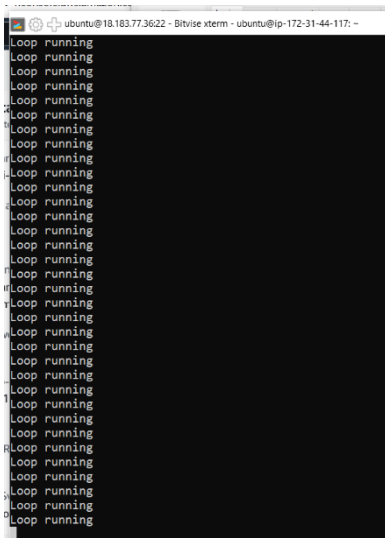
```
done
```



- Now, to save and close the shell script we need to press the following shortcuts and keys sequentially:
 - Ctrl+X
 - Y
 - Enter
- Now you will be returned back to the terminal.
- Now type the following commands:
 - chmod +x su1.sh
 - ./su1.sh (Used to execute the su1.sh script).



- Now the script will start running infinitely!
- Do not close the terminal. Keep it minimized.



37. For Instance-2:

Click on the instance 2. Now click on the connect button

- nano su1.sh

- After the command a new nano Editor window will open.

Type the following in it.

```
#!/bin/bash
```

```
while true
```

do

```
echo "Loop running"
```

done

➤ Now, to save and close the shell script we need to press the following shortcuts and keys sequentially:

Ctrl+X

Y

Enter

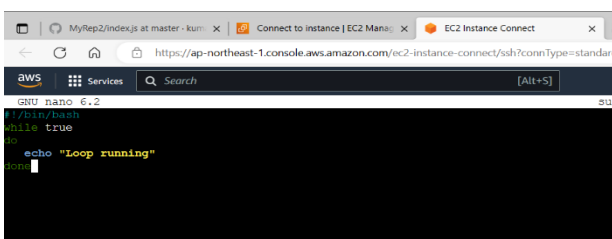
- Now you will be returned back to the terminal.

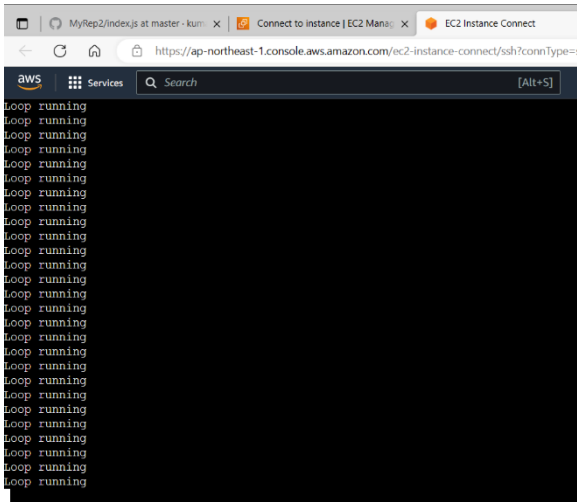
➤ Now type the following commands:

- `chmod +x su1.sh`
- `./su1.sh` (Used to execute the su1.sh script)

➤ Now the script will start running infinitely!

- Do not close the terminal. Go back to the previous tab to keep working in AWS.

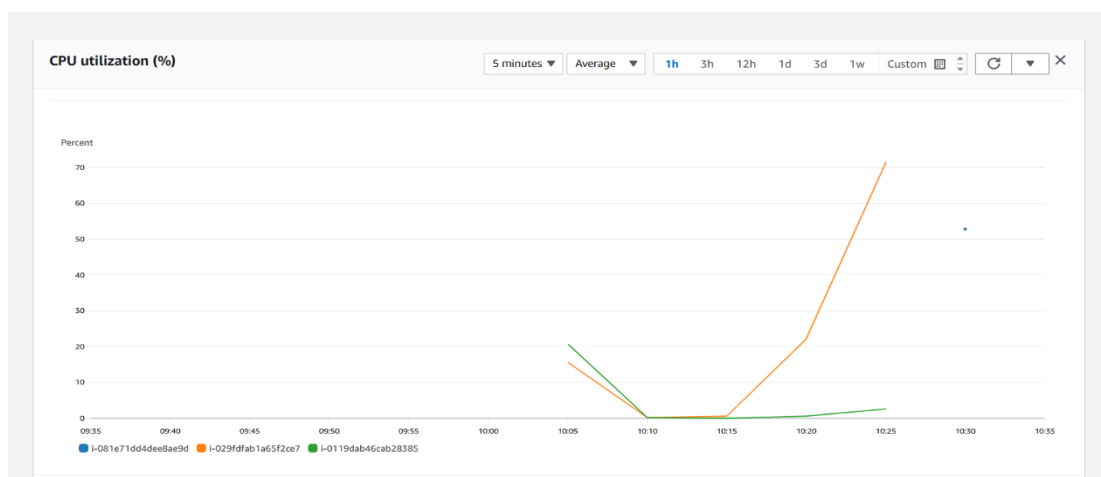
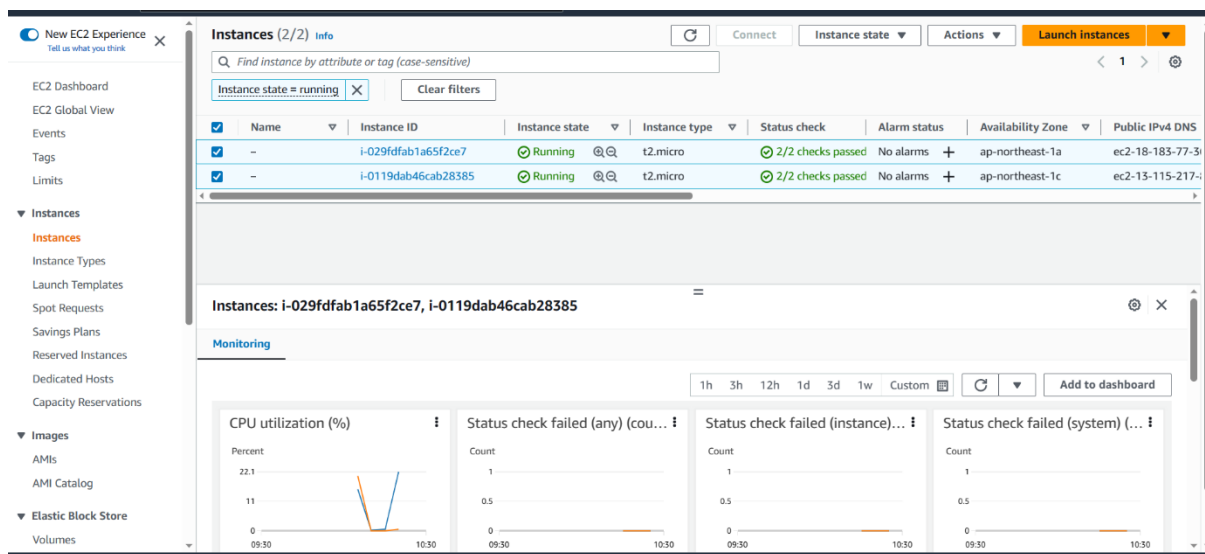




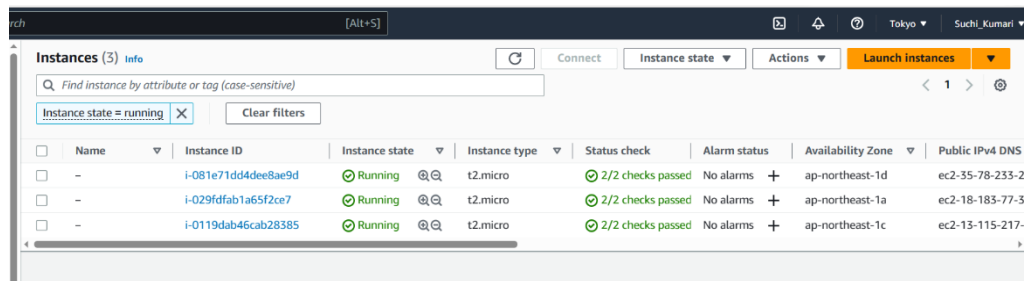
38. Now go to the instances page. Select both the instances.

Click on the instances white bar at the bottom of the page. Now drag the two bars to expand the view. We are interested only in the CPU utilization graph. Click on the maximize icon by hovering over the graph as shown in the fig to maximize this graph.

Our 1st instance has already crossed over 50% utilization. That’s why we can see already a new third instance has been initiated by our auto-scaling group to compensate for the overload.

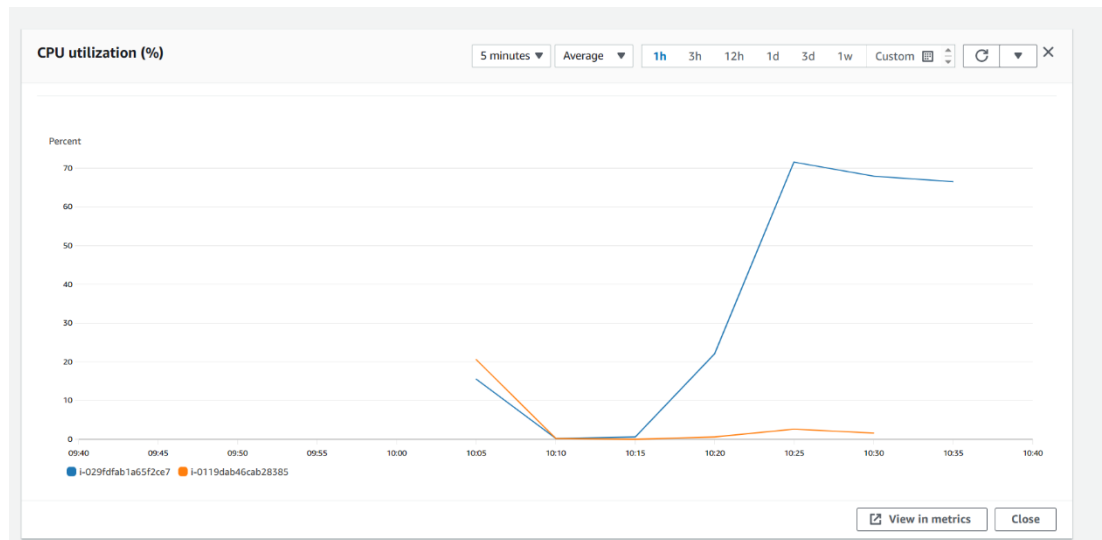


39. There can only be 3 servers running at a time for us as specified in our Auto Scaling group when we were creating it. Hence, we have reached our maximum limit of instances running concurrently.



The screenshot shows the AWS Management Console 'Instances' page. It displays a table with 3 instances, all in a 'Running' state. The table columns include Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4 DNS. The instances are t2.micro type and are located in the ap-northeast-1 region.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
-	i-081e71dd4dee8ae9d	Running	t2.micro	2/2 checks passed	No alarms	ap-northeast-1d	ec2-35-78-233-2
-	i-029fdfab1a65f2ce7	Running	t2.micro	2/2 checks passed	No alarms	ap-northeast-1a	ec2-18-183-77-3
-	i-0119dab46cab28385	Running	t2.micro	2/2 checks passed	No alarms	ap-northeast-1c	ec2-13-115-217-



Hence, our Auto-Scaling Group can handle instance overloading by providing new instances to handle the overloading.

Now observe that whenever we close or terminate any instance then a new instance gets created. Hence, we cannot delete them if we want to delete them finally.

Then first delete Auto Scaling groups

Then delete Load balancer

Then delete Target group

Then You will find that all the instances created by the Auto-Scaling group will automatically be terminated.