A Project Report On

TOPIC

Creating a Highly Available Architecture

Submitted By

Name: Kiran K V

Batch: Certified AWS Cloud Practitioner

Batch Code: 2024-11761 Bangalore-CG AWS CP

Enrollment ID: EBEON0424902026

Email: kvkirankumar1201@gmail.com

Mob: +91 7899283024

ACKNOWLEDGEMENT

I would also like to extend my heartfelt thanks to my mentor, **Mr. Prasiddh Saxena**. His invaluable guidance, insightful advice, and constant encouragement have been pivotal throughout this journey. His expertise and mentorship have not only enhanced my understanding of cloud technologies but also inspired me to strive for excellence.

I would also like to extend my thanks to my peers and instructors in the **Certified AWS Cloud Practitioner** course, **Batch 2024-11761 in Bangalore**. The collaborative learning environment and the shared knowledge among batch mates have greatly enhanced my understanding and application of cloud computing concepts.

Special thanks to the administrative staff and technical support team for their assistance in providing the necessary resources and infrastructure to carry out this project.

Kiran K V

Batch Code: 2024-11761 Bangalore

Enrollment ID: EBEON0424902026

Email: kvkirankumar1201@gmail.com

Mob: +91 7899283024

Table of Contents.

Chapter 1

INTRODUCTION

Amazon Web Services (AWS) is a comprehensive and widely adopted cloud platform offering over 200 fully-featured services from data centers globally. It provides on-demand cloud computing resources and APIs to individuals, companies, and governments on a metered pay-as-you-go basis. AWS services range from computing, storage, and databases to machine learning, analytics, and IoT. Its flexible infrastructure enables businesses to scale efficiently and deploy applications worldwide with high availability and low latency. AWS also emphasizes robust security measures, compliance certifications, and data protection. With its extensive ecosystem, AWS supports various industries, fostering innovation and transformation.

The primary objective of this project is to create an automated, highly available web application infrastructure that can dynamically scale based on traffic demands. The project encompasses the following key components:

1. **AWS VPC (Virtual Private Cloud):** Amazon Virtual Private Cloud (VPC) is a service that allows you to launch AWS resources in a logically isolated virtual network that you define. It gives you complete control over your virtual networking environment, including resource placement, connectivity, and security.
2. **AWS Simple Notification Service (SNS):** Used for setting up notifications to monitor and manage our infrastructure. SNS provides a reliable way to send alerts for different events within our application, ensuring we stay informed about the system's health and performance.
3. **Custom Amazon Machine Image (AMI**): AMIs are essential for creating pre-configured environments that can be replicated across multiple instances. This ensures consistency and speeds up the deployment process.
4. **Amazon EC2 Instances:** These virtual servers provide the computing power needed for our application. We will launch and configure EC2 instances to host our web application.
5. **Elastic Load Balancer (ELB):** A load balancer is set up to distribute incoming traffic across multiple EC2 instances, ensuring no single instance is overwhelmed and improving the application's availability and fault tolerance.
6. **Security Groups:** Configuring security groups to manage inbound and outbound traffic to ensure our instances are secure and accessible only through predefined rules.

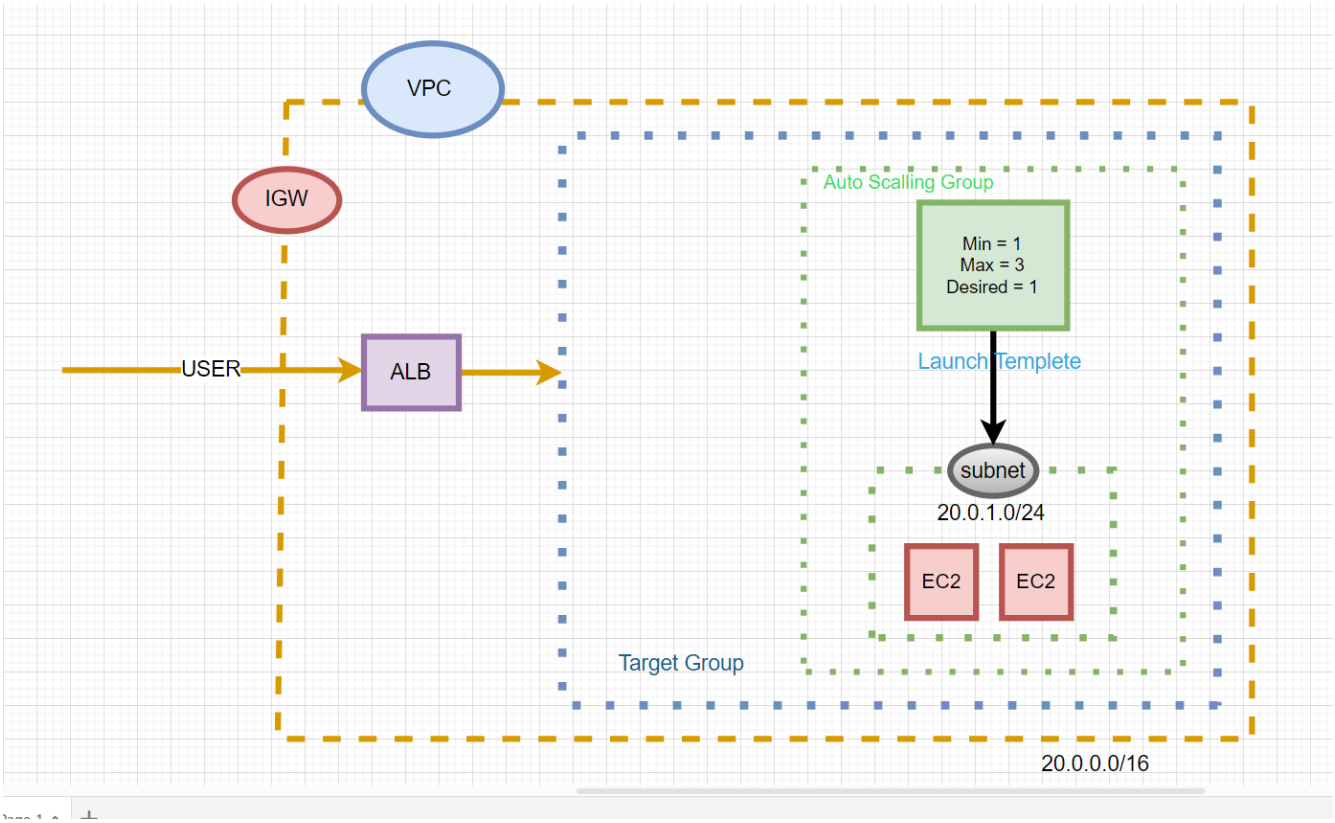
7. **Auto Scaling Groups:** These are configured to automatically adjust the number of EC2 instances based on the current demand. This ensures that the application can handle varying loads efficiently, scaling up during peak times and scaling down when demand is low.
8. **AWS Cloud Watch:** Used for monitoring the infrastructure, setting up alarms, and triggering scaling actions. Cloud Watch provides detailed insights into the system's performance and operational health, enabling proactive management.

The following sections of the report provide a step-by-step guide to implementing each component, starting with setting up virtual private cloud, notifications using SNS, creating a custom AMI, launching and configuring EC2 instances, and setting up security groups, load balancers, auto-scaling groups, and Cloud Watch alarms. By the end of this project, we will have a resilient, scalable, and automated web hosting solution capable of handling fluctuating workloads while maintaining high availability and performance.

Chapter 2

ARCHITECTURE

N.Virginia



Words used in naming conventions:-

VPC – Mproject-Test-VPC (Virtual private cloud)

IGW – Mproject-Test-IGW (Internet gateway)

Subnet - Mproject-test-public-subnet-1a

Simple notification service - Mproject-Test-SNS

AMI- Mproject-Test-AMI

ALb – Mproject-Test-ALB

EC2 – Mproject-Test-Ec2-Instance

ASG - Mproject-Test-AutoScallingGroup

Target Group – Homepage

Alarms = Scal-In, Scal-Out

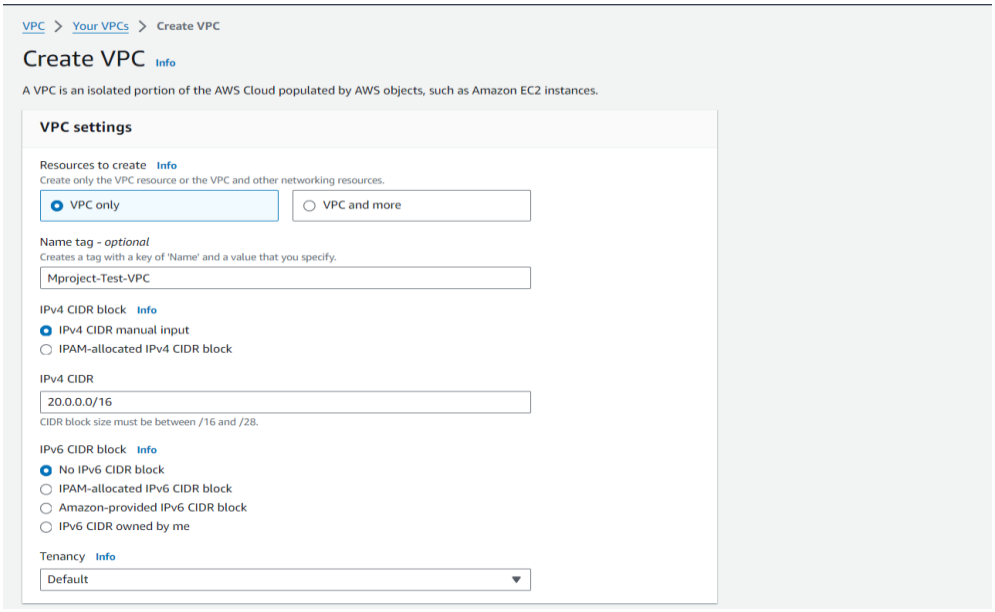
Chapter 3

EXECUTION

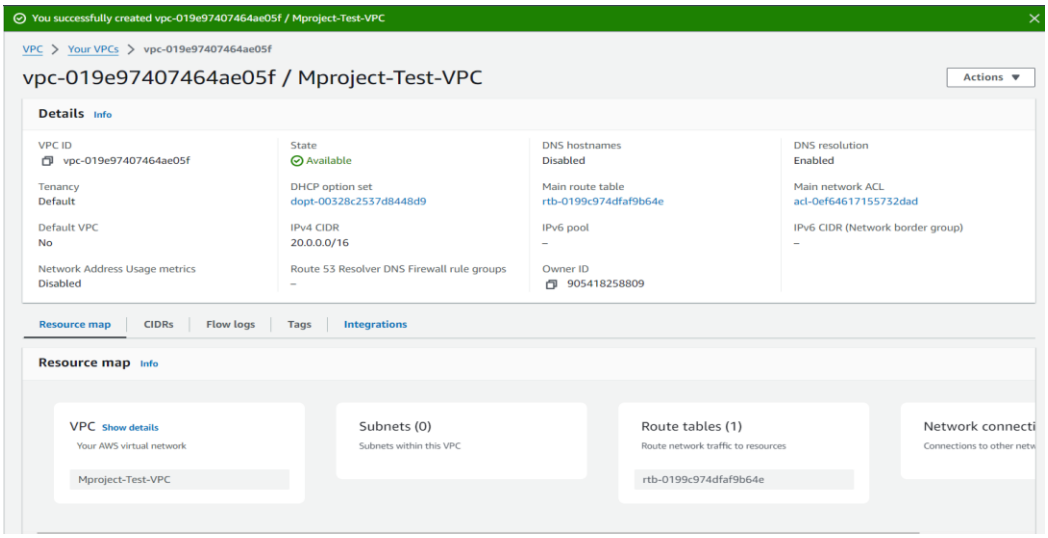
Step 1: VPC configuration.

a. Once log-in to the AWS Console, click Services -> Network & Content Delivery -> VPC.

- Select resource - VPC only
- Name- Mproject-Test-VPC
- Enable Ipv4 manual input,
- Ipv4 CIDR – 20.0.0.0/16

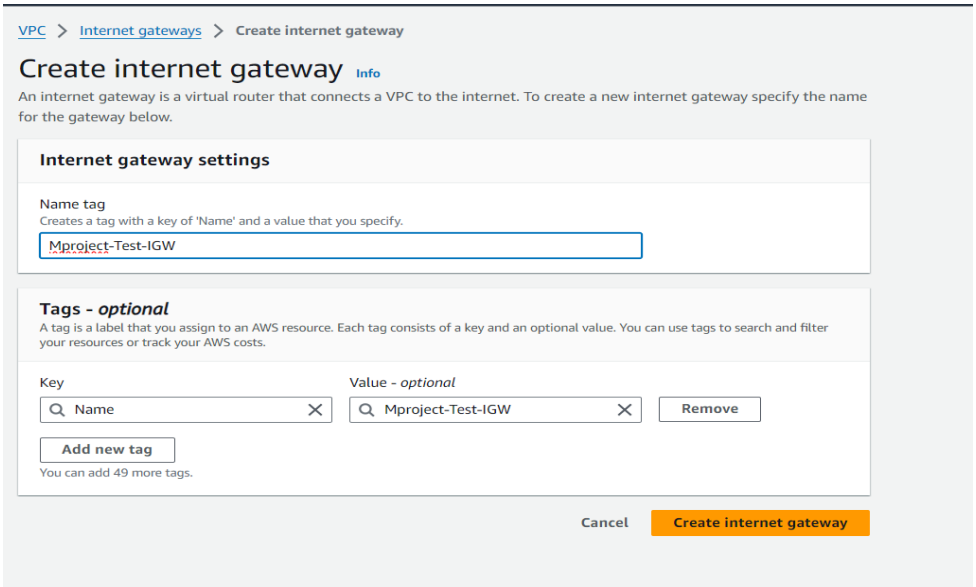


- Next Create VPC.

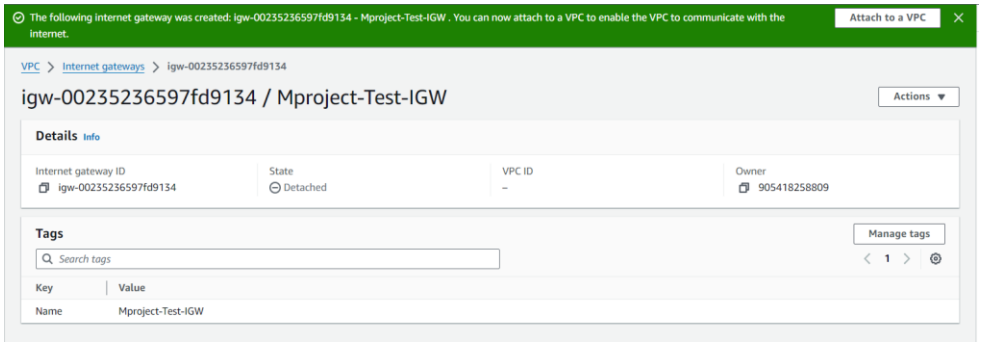


b. Create Internet gateway and attaching it into the VPC, Services -> Network & Content Delivery -> Internet Gateways.

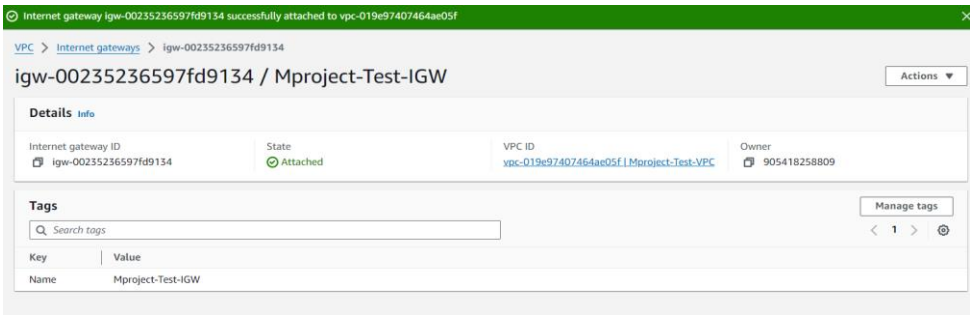
- Name - Mproject-Test-IGW



- Created Internet gateway.

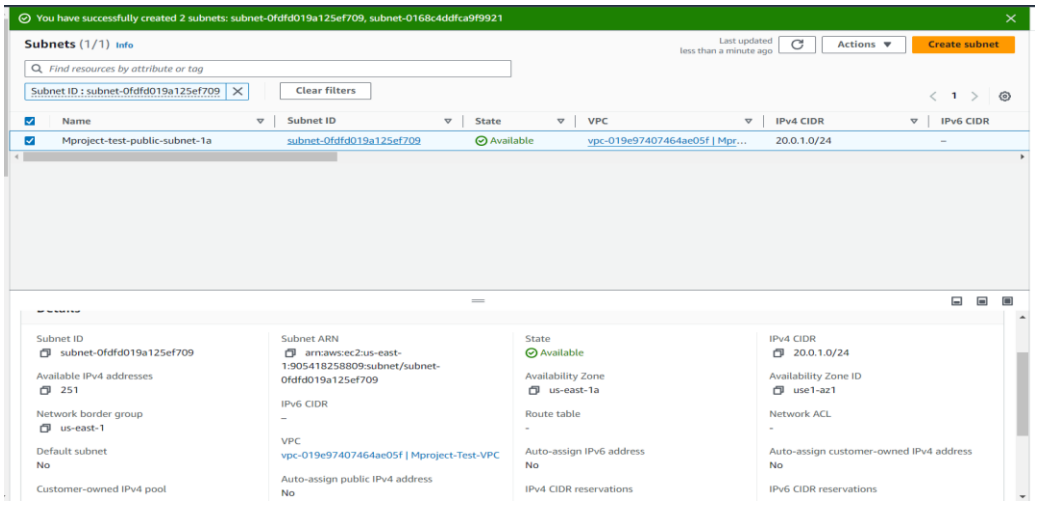


- Attached to the VPC. Click on Actions-> attach VPC -> select VPC -> attach

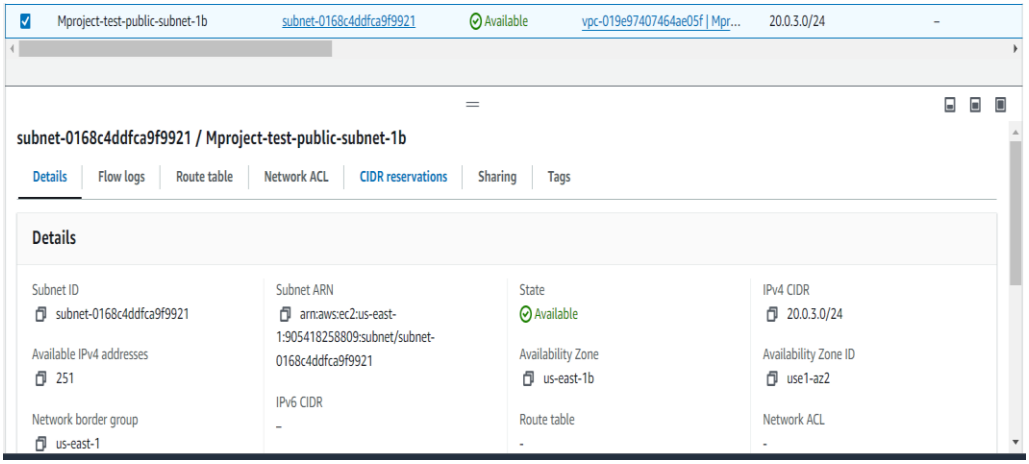


c. Creating subnets, Click Services -> Network & Content Delivery -> Subnets->create subnet.

- Select VPC – Mproject-Test-VPC
- Subnet name - Mproject-test-public-subnet-1a
- Ipv4 subnet CIDR block – 20.0.1.0/24

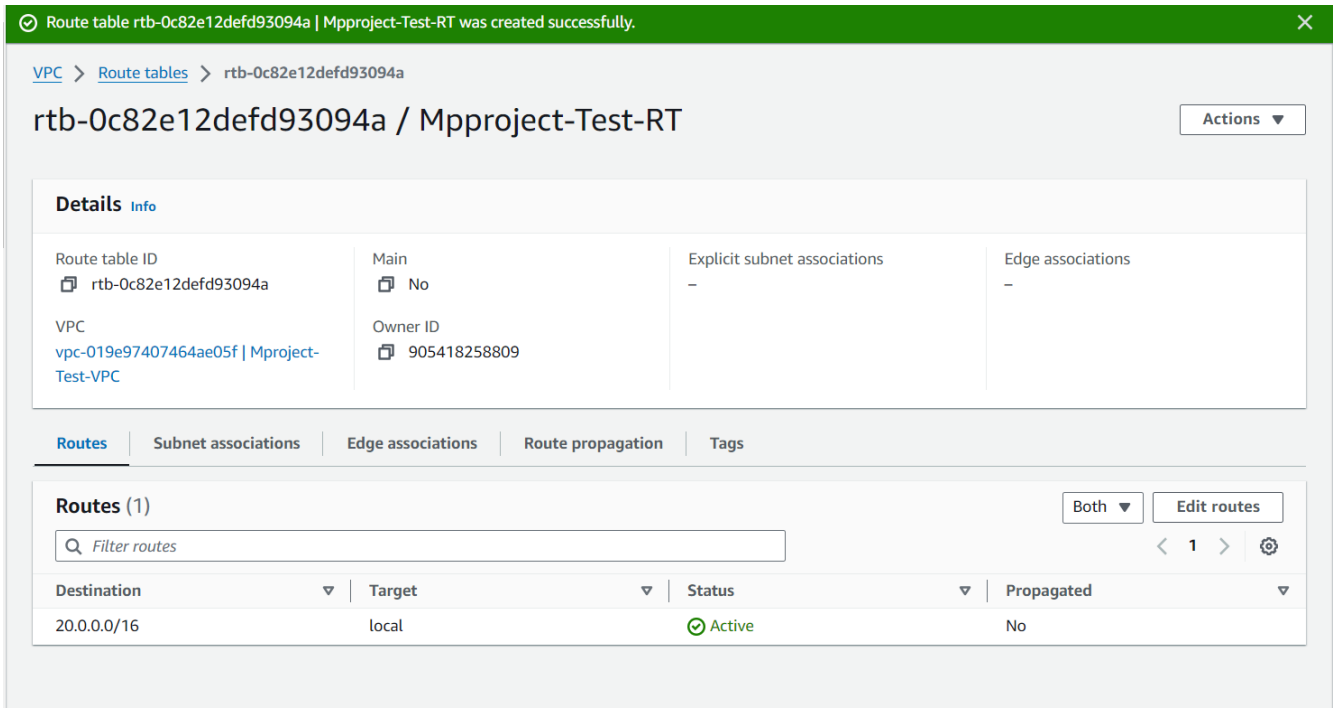


- Subnet name – Mproject-test-public-subnet-1b
- Ipv4 subnet CIDR block – 20.0.3.0/24

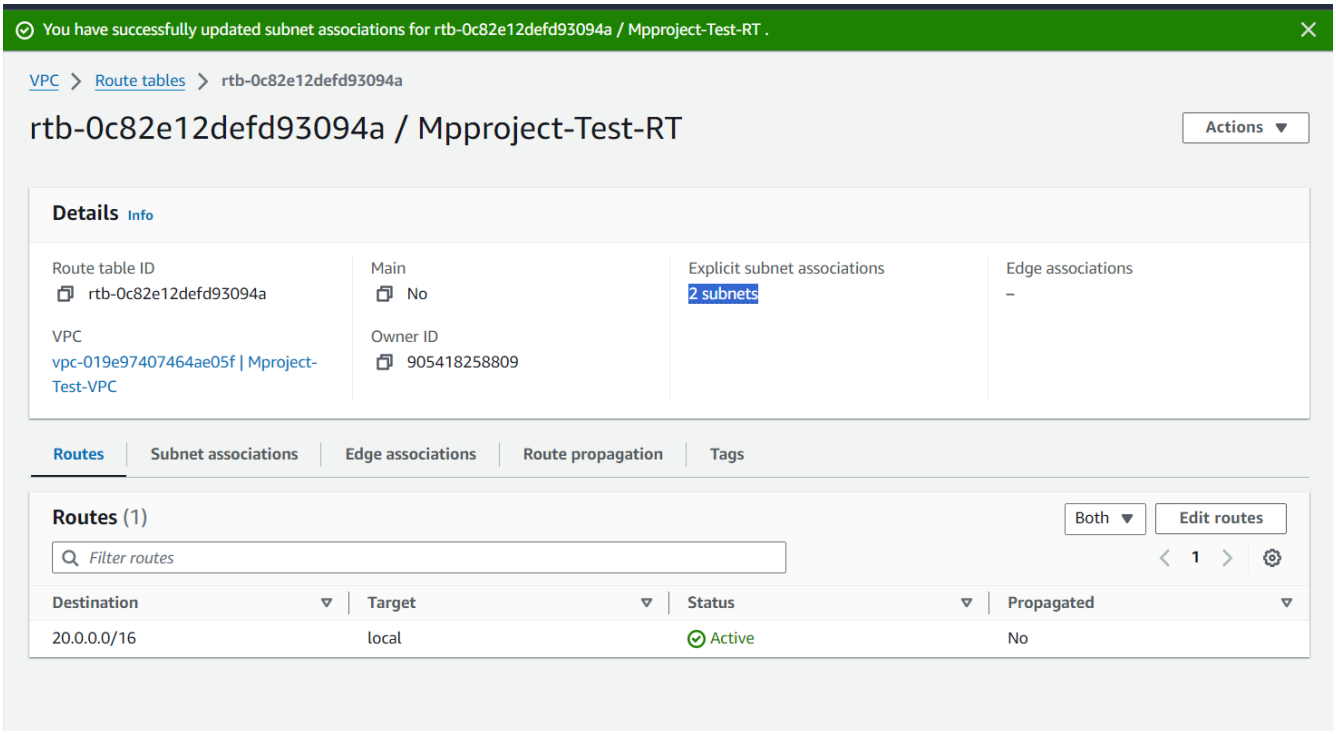


d. Create Rout table, , Click Services -> Network & Content Delivery -> Route Tables ->create Route table.

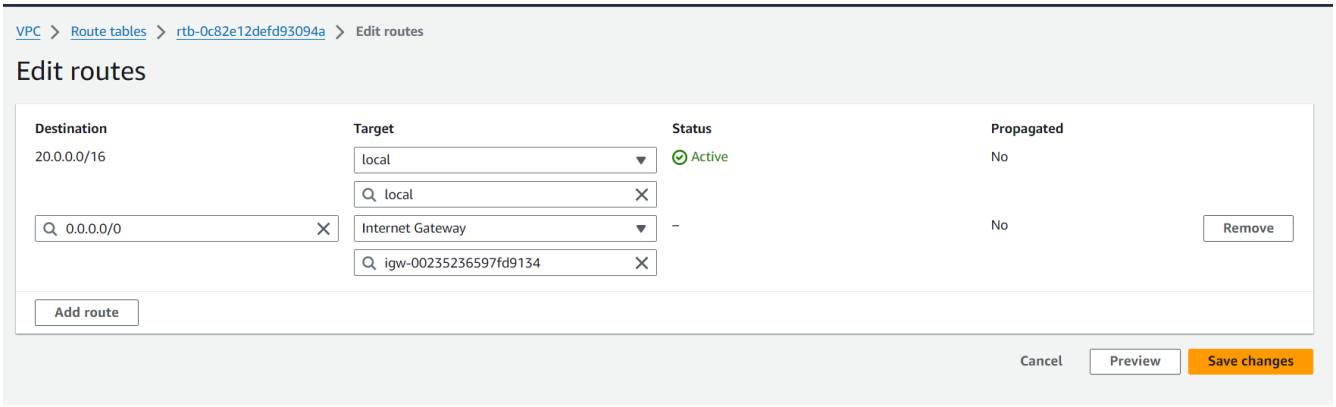
- Name – Mpproject-Test-RT
- Select VPC – Mproject-Test-VPC

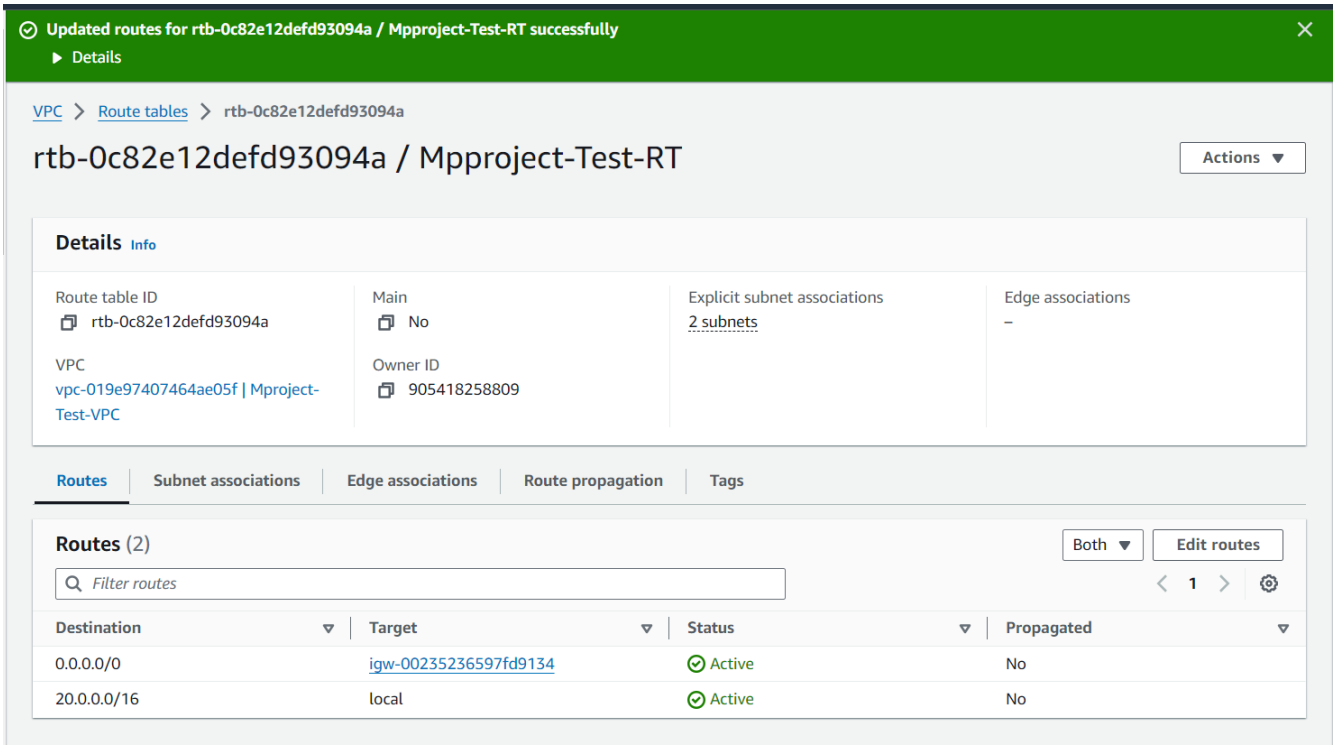


- Now associate with public subnets to routable. Click Subnet association -> edit subnet association -> select both the subnets -> save the association.



- This route table also provide the internet access, add the internet route Click route -> edit route -> add route -> select 0.0.0.0/0 for internet access and target type is internet gateway select -> Mproject-Test-IGW -> save the changes.

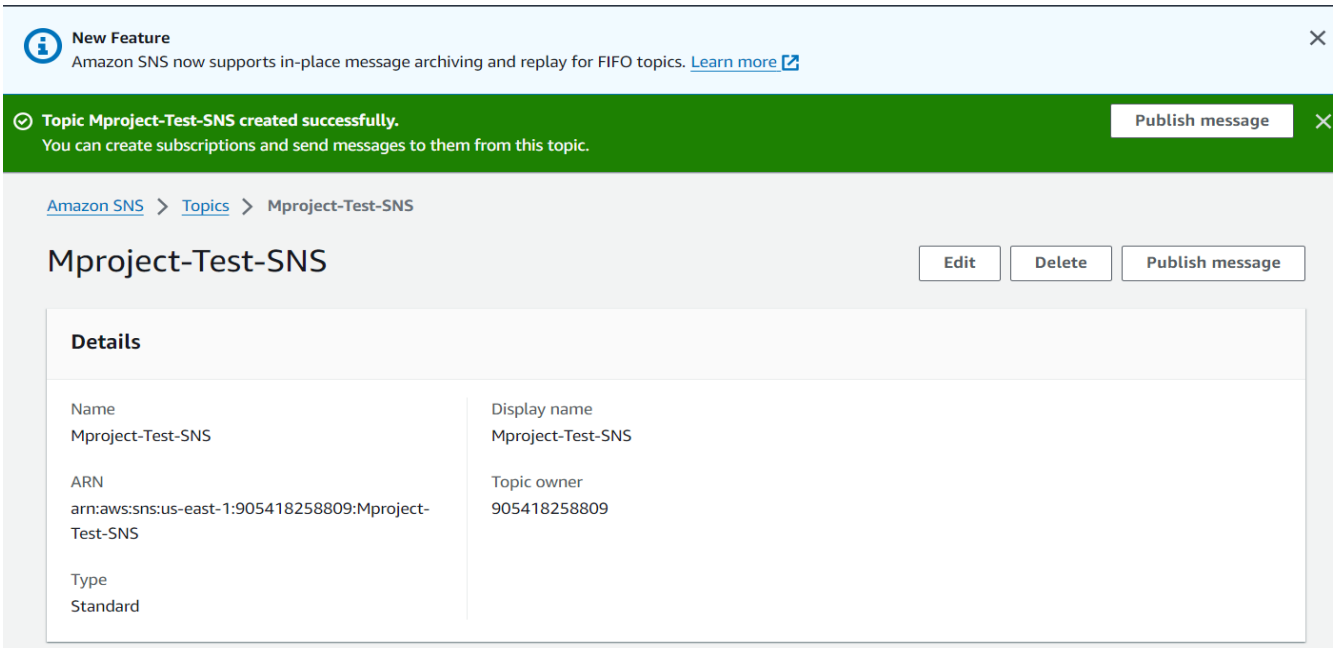




Step 2: AWS Simple Notification Service configuration.

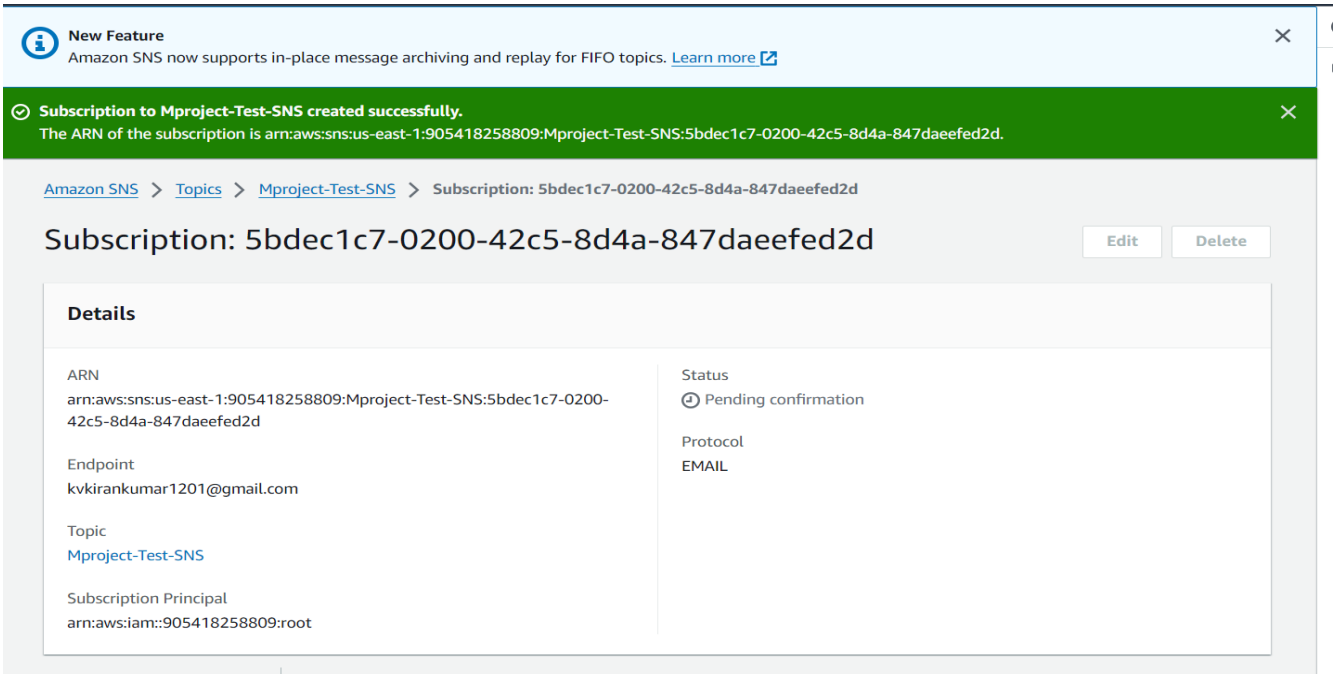
a. Click Services -> Application Integration -> Simple Notification Service.

- Create Topic name – Mproject-Test-SNS
- Type – Standard
- Display Name - Mproject-Test-SNS

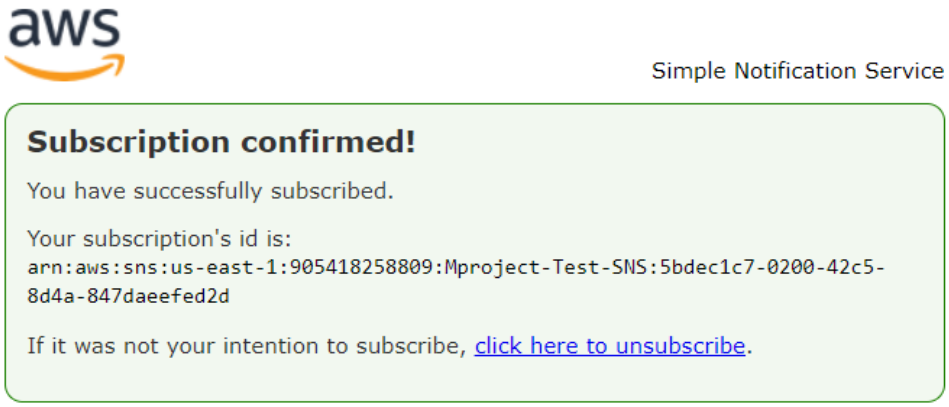


b. Create Subscription, Click create subscription

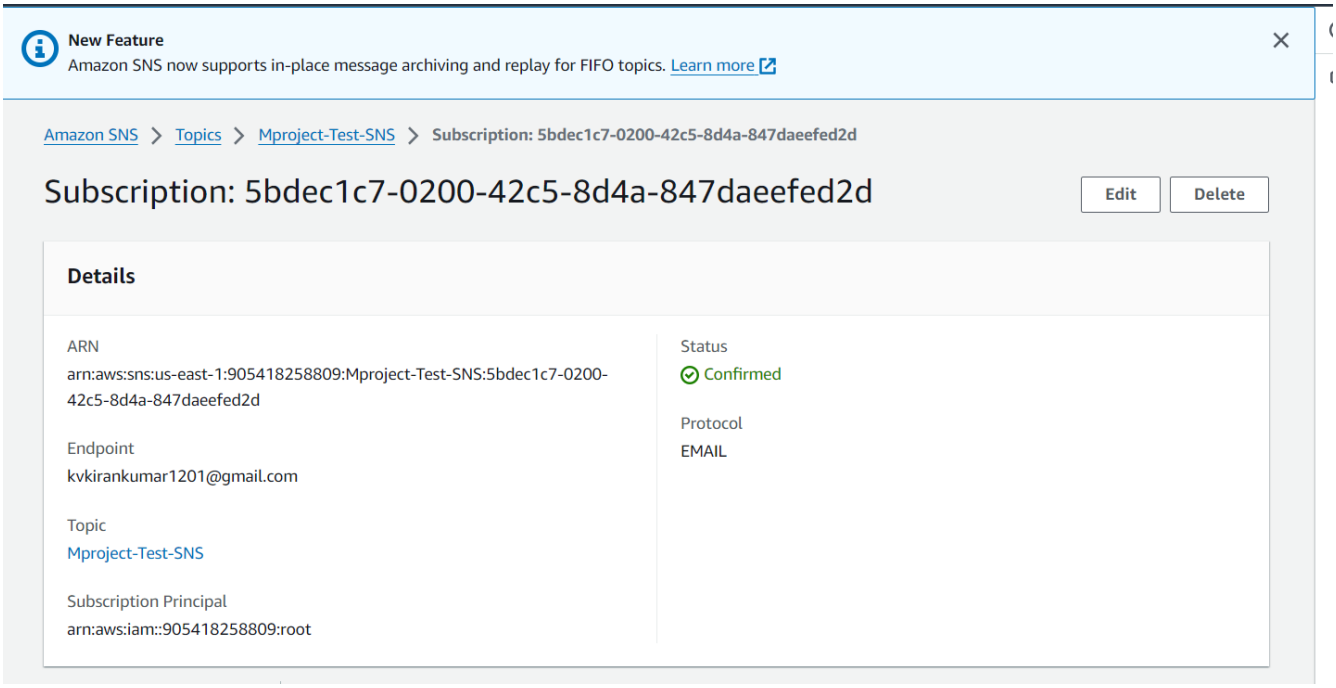
- Select Protocal – Email
- EndPoint – kvkirankumar1201@gmail.com



c. Confirm the subscription in the Email endpoint.



d. Refresh the page. Status has been turned to " Confirmed "

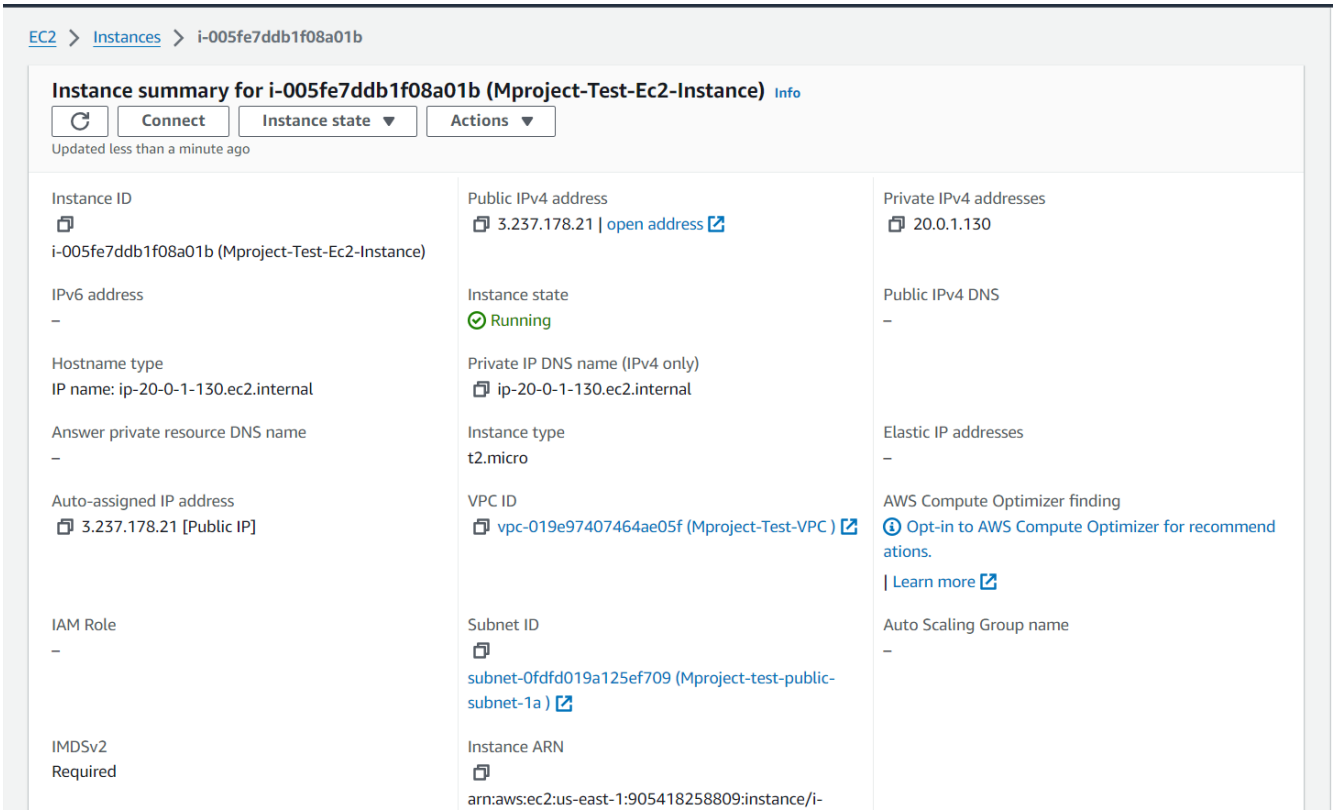


Simple Notification configuration has done.

Step 3: Custom AMI configuration.

a. Click Services -> Compute -> EC2 -> Launch Instance

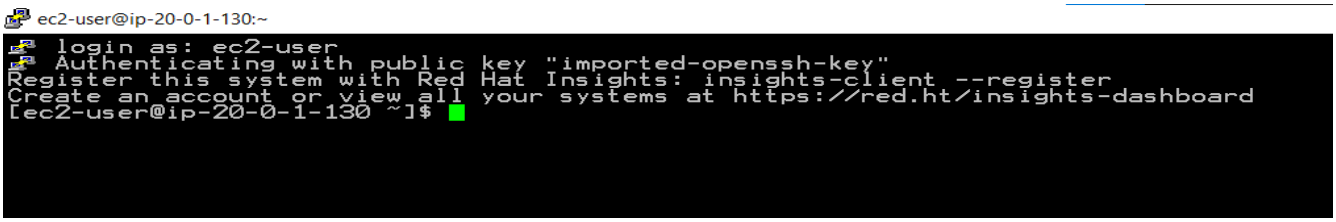
- Name - Mproject-Test-Ec2-Instance
- Select Amazon Machine Image as quick start Red Hat (Free tire eligible) enterprises Linux 9.
- Instance Type – t2.micro free tire eligible.
- Click Create key-pair -> name (Linux_key) -> key pair type (RSA) -> file formate(.ppk) -> Create key pair.
- Select Key pair – linux_key
- Network settings VPC – Mproject-Test-VPC.
- Subnet – Mproject-Test-public-subnet-1a
- Auto-asign public ip – enable
- Create security group - > Security group name(Mproject-Test-SG)
- Inbond security group rules - > Allow SSH port:22 protocal(TCP) for instance connection and click add security group rule -> select type HTTP port:80 source type Anywhere for server hosting.
- Configure storage -> 10 GIB and gp2 root valume (Up to 30 GB included in free tire).
- Click Launch Instance
- Wait Instance state turning into the Running state.



Instance has been successfully created. Instance state become a "Running"

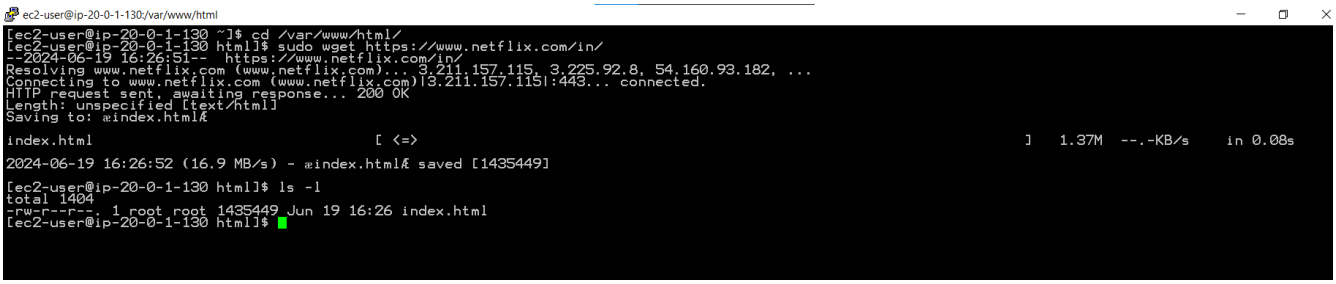
b. Let's connect the putty.

- Open Putty
- In session -> Host name or IP address -> paste Instance Public IPV4 address (3.237.178.21) -> port : 80, connection type SSH,
- Next window -> appetence -> Change – Courier name -> Terminal.
- Connection -> Enable TCP_Keepalives -> set TCP_Keepalives as (100).
- Connection -> SSH -> Auth -> Credentials -> Browse private key while insance creation we have created private key pair -> upload.
- Click open
- Accept putty alert.
- Login as ec2-user.



c. No let's deploy the any website (Netflix).

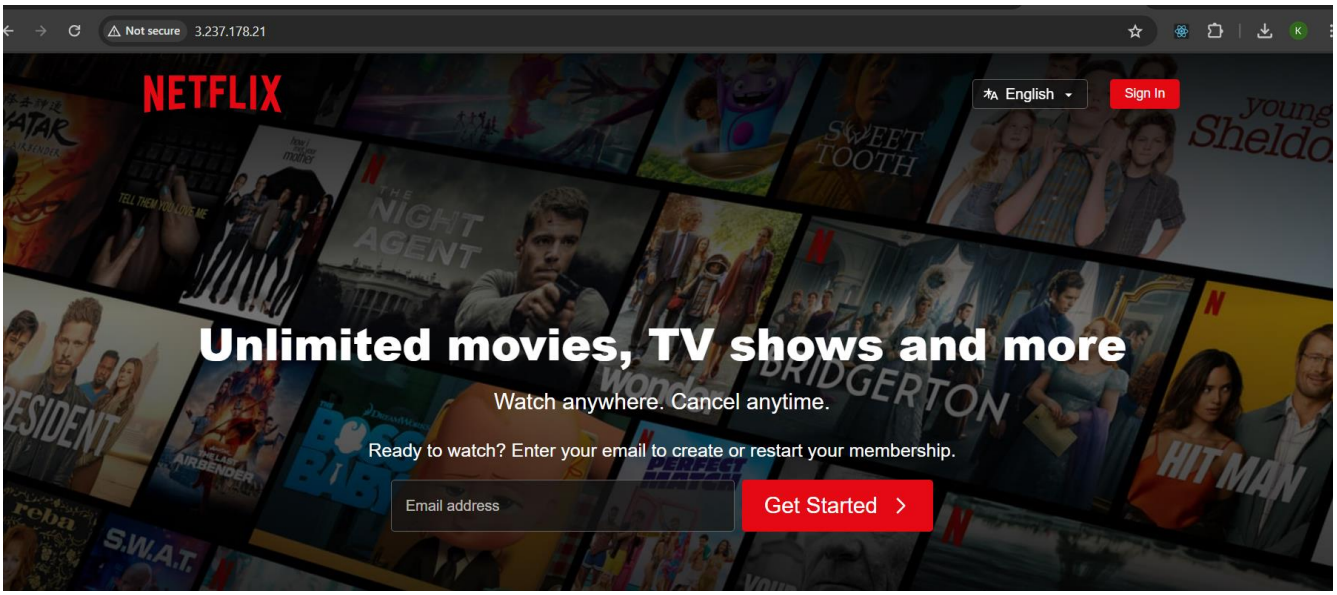
- Install httpd and wget servers – sudo yum install httpd wget –y
- Put developed index.html code into - /var/www/html/ directory



- Start deploying the Netflix website
- Run – sudo systemctl start httpd
- Enable server – sudo systemctl enable httpd
- Check status – sudo systemctl status httpd (Active status)

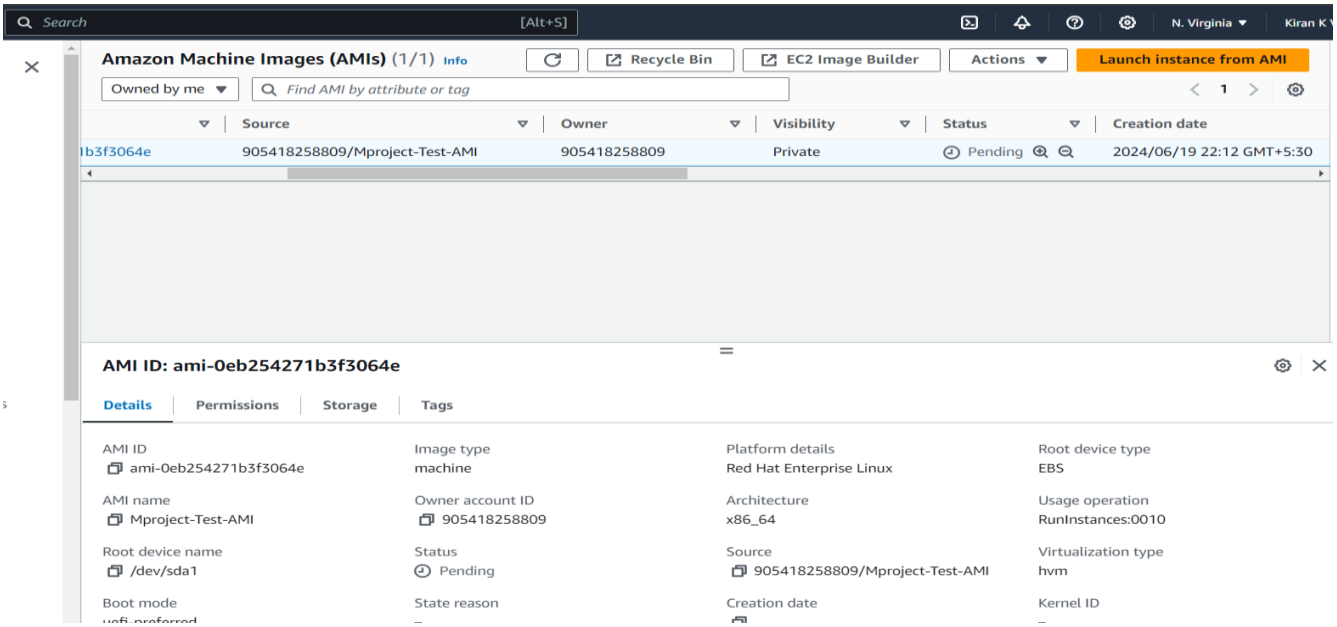
d. Verify the website successfully deployed into the server

- Go to the instance copy the public ipv4 address (3.237.178.21), paste in to the any web browser check server up and running.

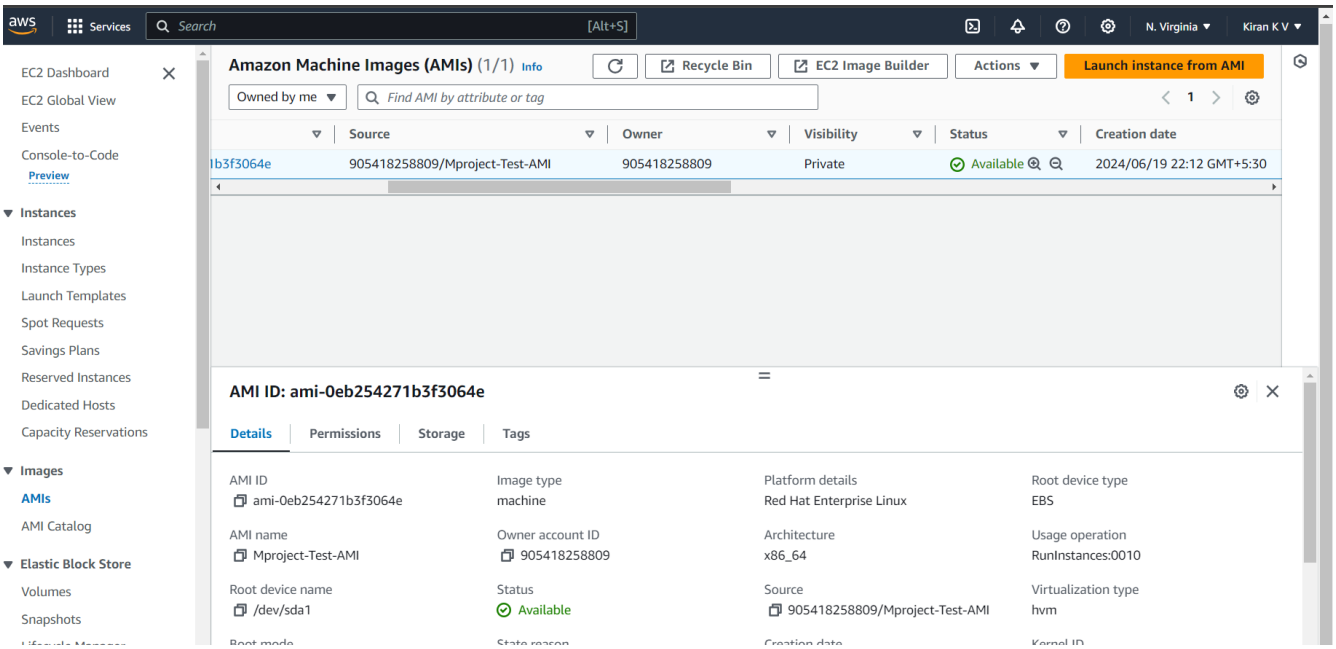


e. Creation of custom AMI:

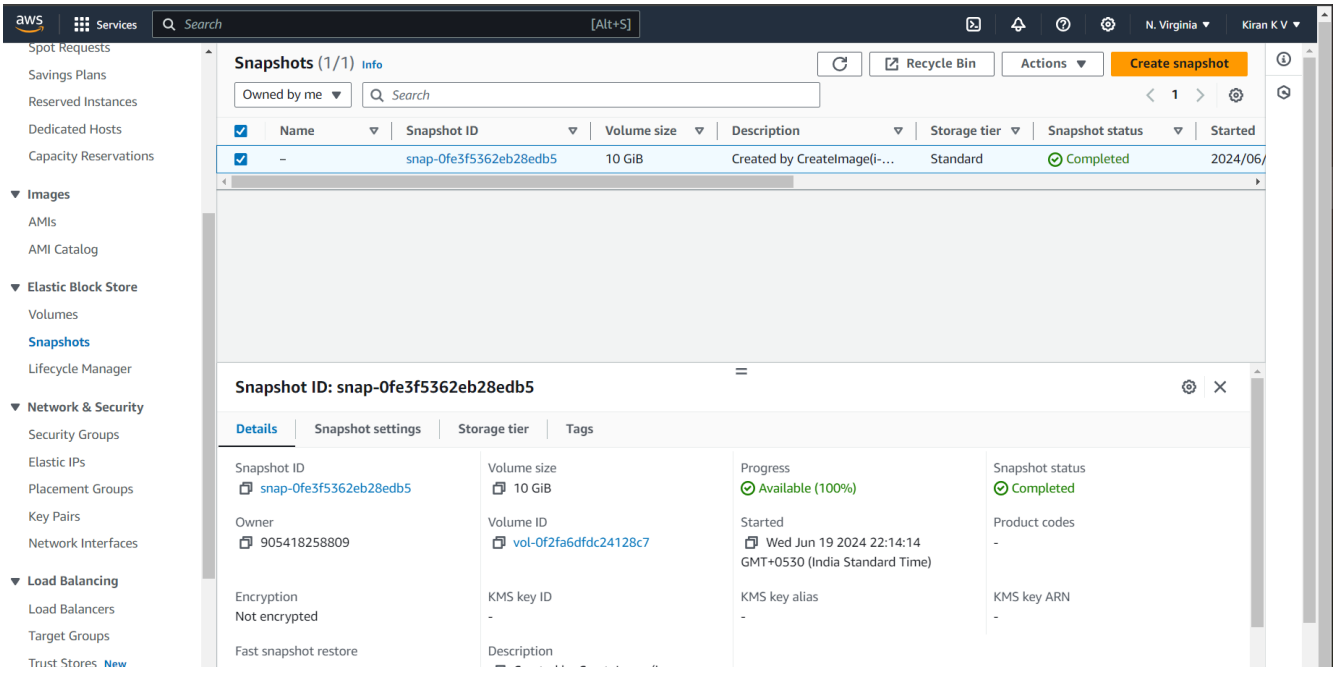
- Select the instance -> click action -> select image and templates -> Create image.
- Name - Mproject-Test-AMI,
- Enable Tag image and snapshot together
- Create Image.
- Click service -> compute -> EC2-> AMI.



- Select the AMI. You can see that the status is pending. Wait till it becomes Available (take 5-10 mins)



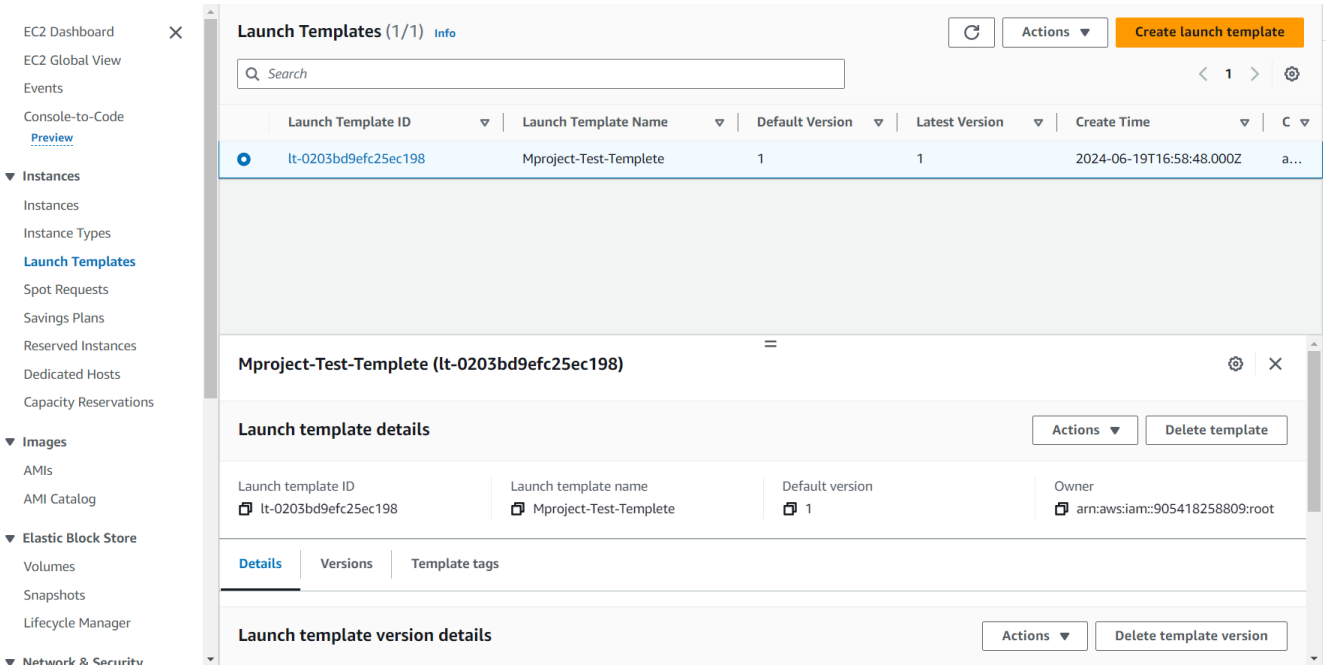
- In the same panel check snapshot created and it's turned into the available state.



Step 4: Configuration of Launch template.

a. Click Service -> Ec2 -> Launch Templates ->Create Launch Templates

- Name – Mproject-Test-Templete
- Version Template description - Mproject-Test-Templete
- AMI -> My AMI -> browse Mproject-Test-AMI -> select.
- Instance type – t2.micro (Free tire eligible).
- Security group – Mproject – Test-SG
- Click create launch templete.

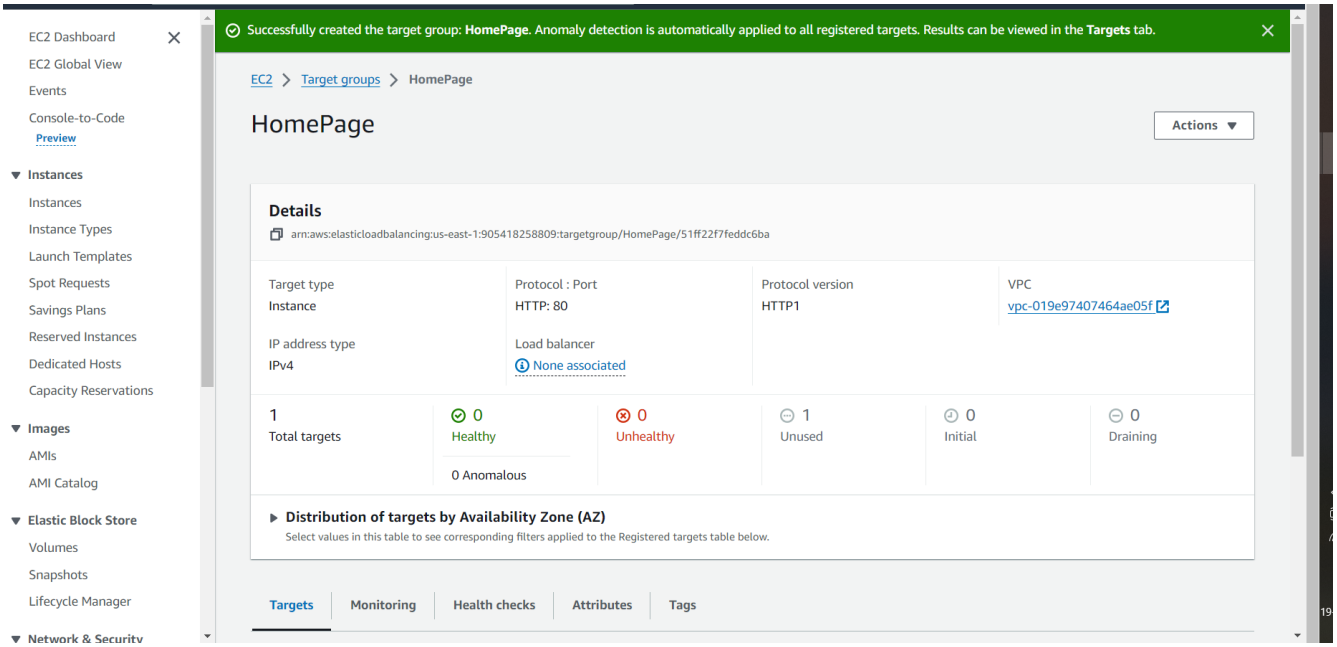


Launch template configuration has done.

Step 5: Application Load Balancer Configuration.

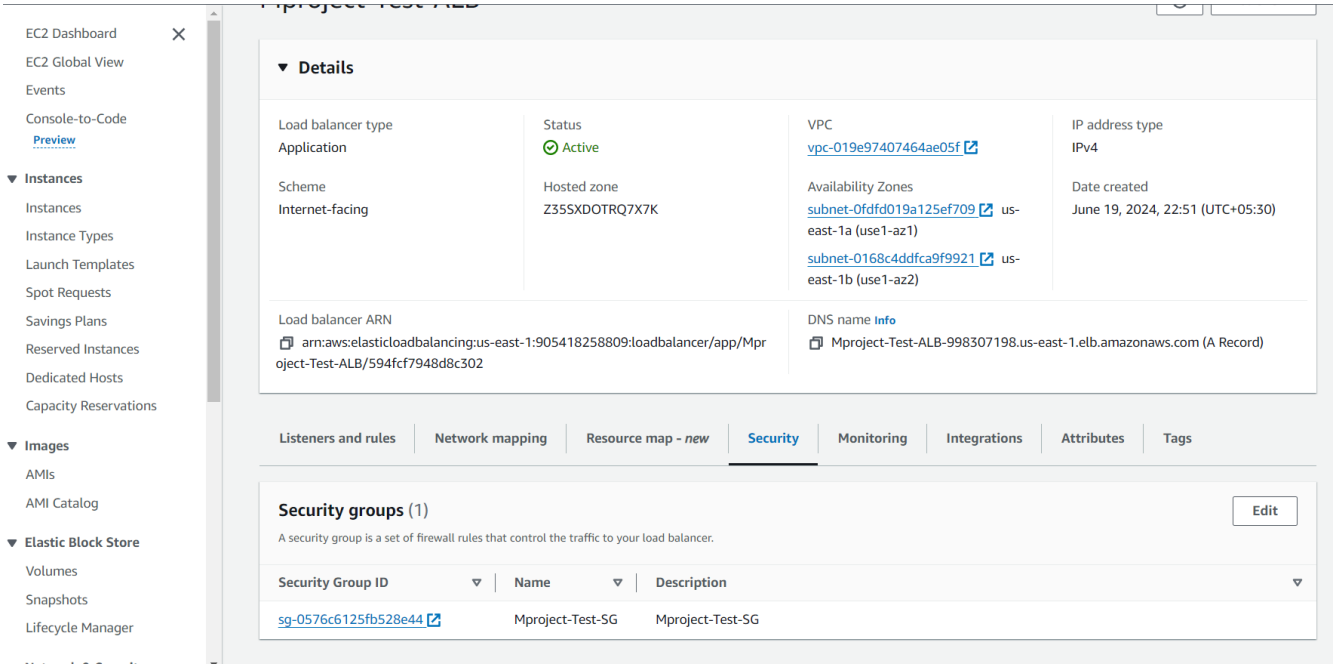
a. Click Service -> compute -> Ec2-> Load balancer -> Target Group ->Create Target group.

- Target type – Instance
- Target group name – HomePage.
- Protocol – HTTP, port:80.
- Ip address type – Ipv4.
- VPC – Mproject – Test-VPC
- Protocol version – HTTP1.
- Health check path - /index.html.
- Advance health check setting- health threshold(2), Unhealth threshold(2), Time out(2), Interval (5) and success codes as 200.
- Click Next -> select available instance (Mproject-Test-Ec2-instance) -> click include as pending bellow.
- Create target group.

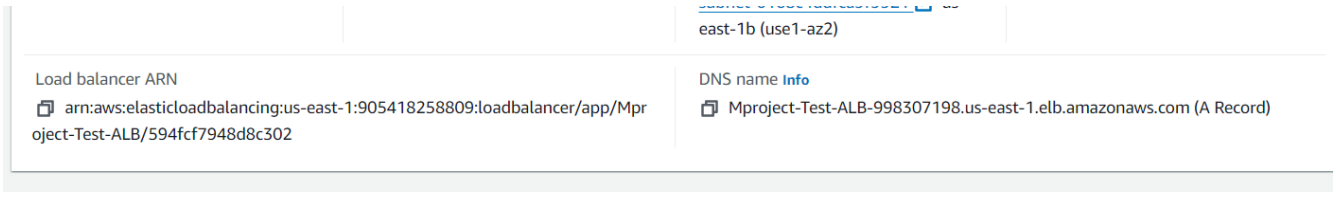


b. Creating Application load balancer

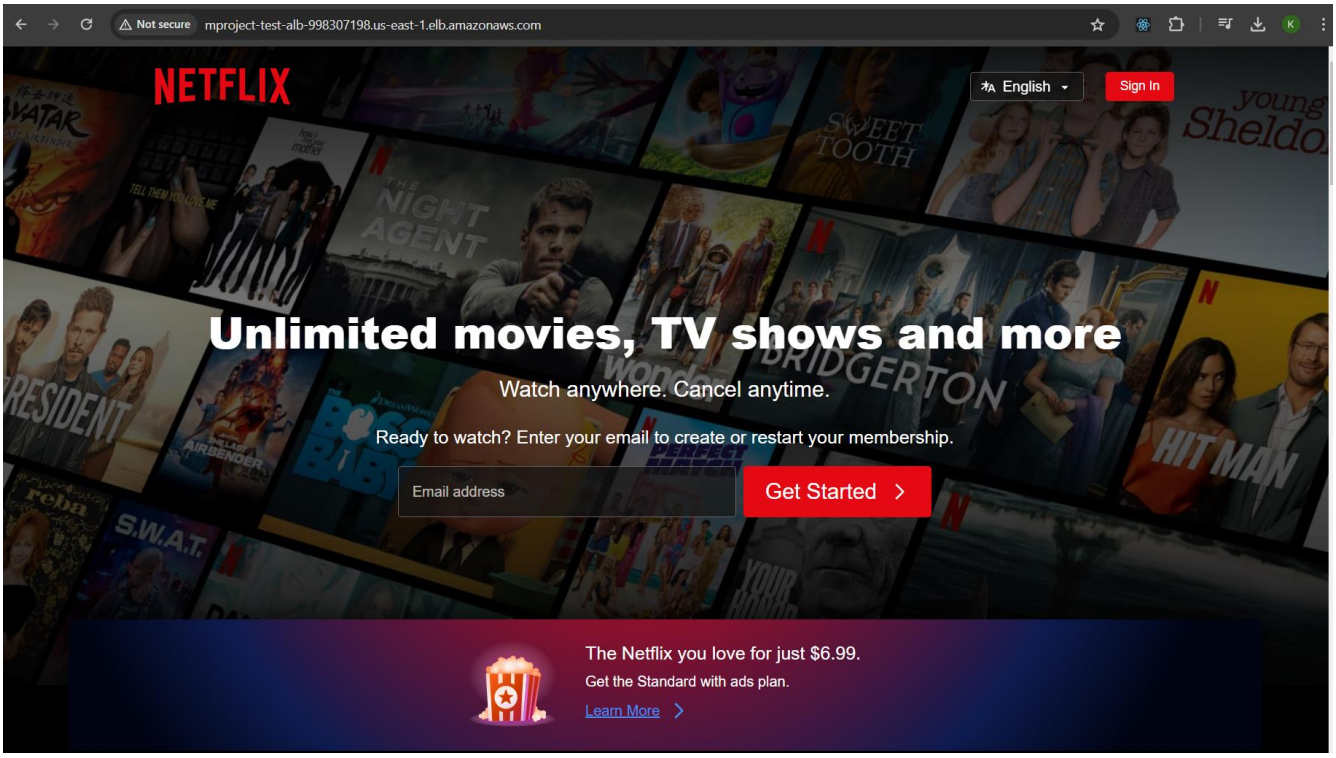
- Click in same panel Load balancer -> create load balancer.
- Select application load balancer -> create.
- Load balancer name - Mproject-Test-ALB
- Scheme – internet – facing
- Ip address type – ipv4.
- Network Mapping – VPC(Mproject-Test-VPC) and Maping select Mproject-Test-public-subnet-1a and Mproject-Test-public-subnet-1b (N.Virginia availability zone).
- Select security group as – Mproject-Test-SG
- Listner routing – protocol(HTTP), port:80, Default action target group (HomePage).
- Create load balancer.
- Wait for status to turn Active.



- Copy the DNS name and Check status on web browser
- DNS name - Mproject-Test-ALB-998307198.us-east-1.elb.amazonaws.com



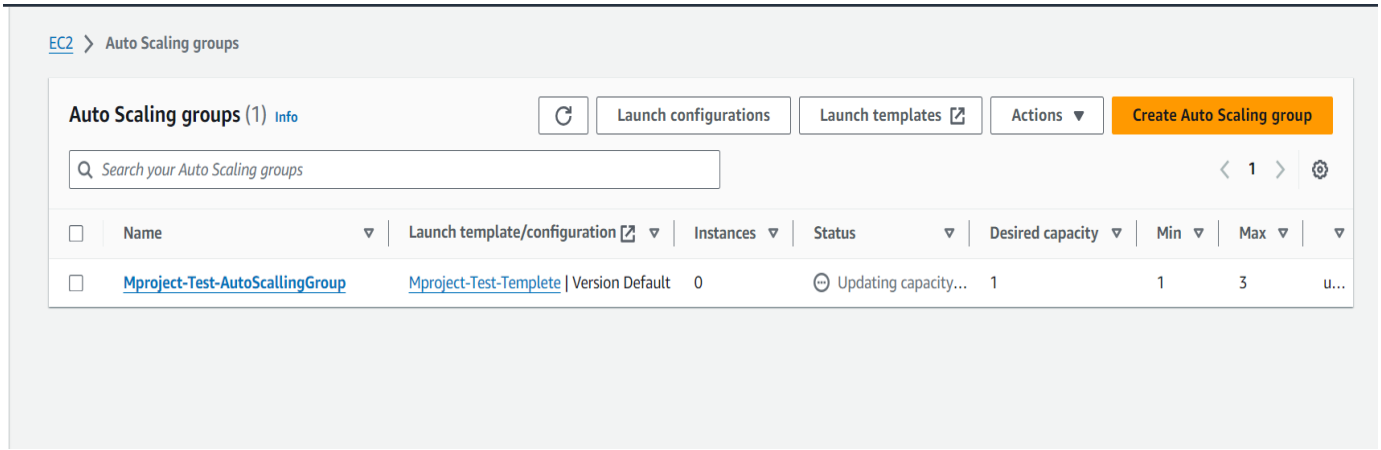
Application load balancer has successfully configured.



Step 6: Auto Scaling Group Configuration.

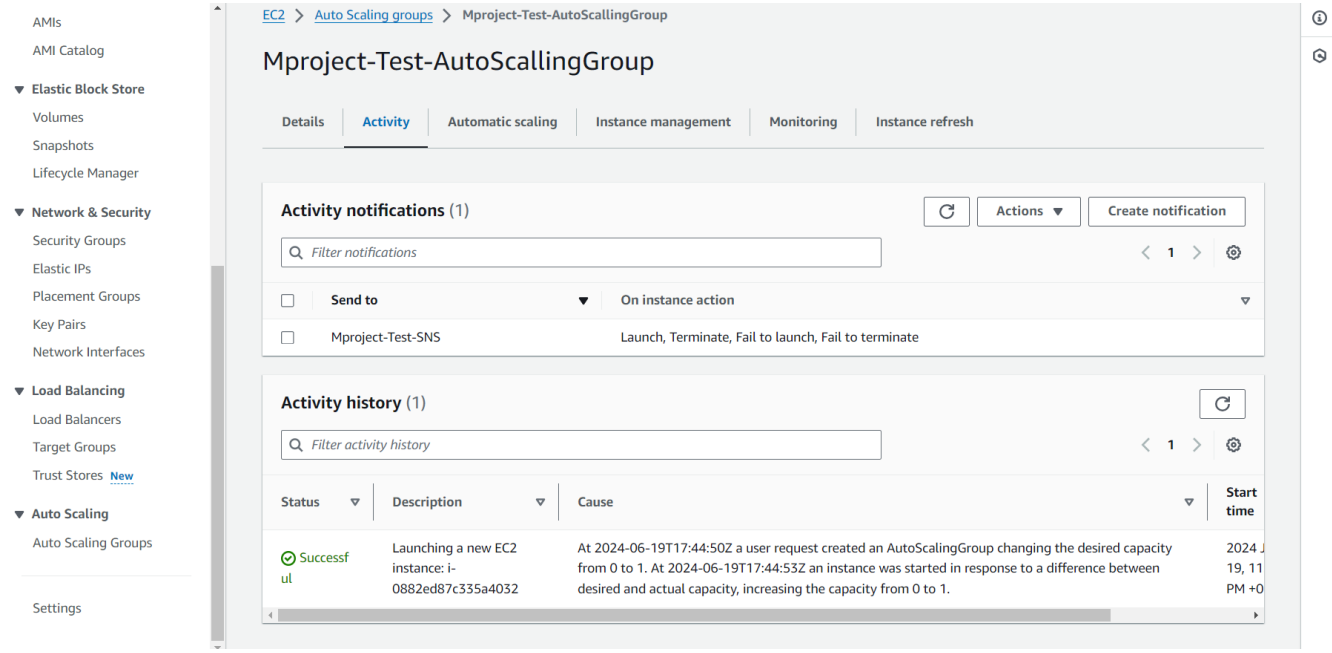
a. Click service -> compute -> ec2 -> Auto scaling -> create auto scaling groups.

- Name – Mproject-Test-AutoScallingGroup.
- Select the Launch template – Mproject-Test-Template.
- Click – next
- Select VPC – Mproject-Test-VPC.
- Select availability zones and subnets – Mproject-Test-public-subnet-1a and Mproject-Test-public-subnet-1b.
- Next.
- Load balancing – > enable Attach on existing load balancer –> enable chose from your load balancer -> select the existing load balancer target group (HomePage).
- Enable No VPC lattice service.
- .In additional settings, Checkmark the Enable group metrics collection within Cloud Watch option.
- Now click Next.
- Set desired capacity to 1 (because only 1 we can create in free-tier).
- Set Minimum desired capacity to 1, and max to 3. Select No scaling policies.
- Select instance maintenance policy as No policy.
- Checkmark the enable instance scale-in protection and click next.
- Click Add Notification -> Set SNS topic to Mproject-Test-SNS, allow all event types and click next.
- Click next.
- Review the configuration and create auto scaling group.

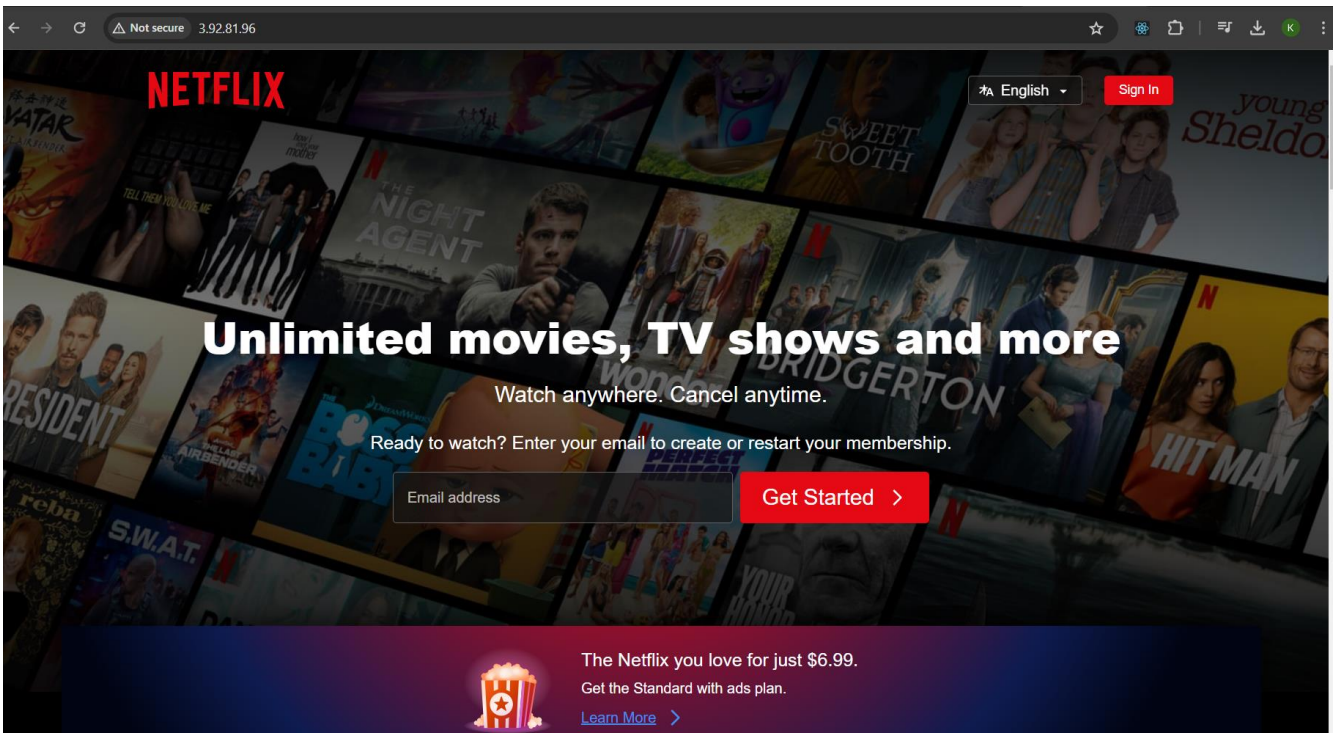


Auto Scaling Group has been created successfully and running server.

We can observe the auto scaling group activity.



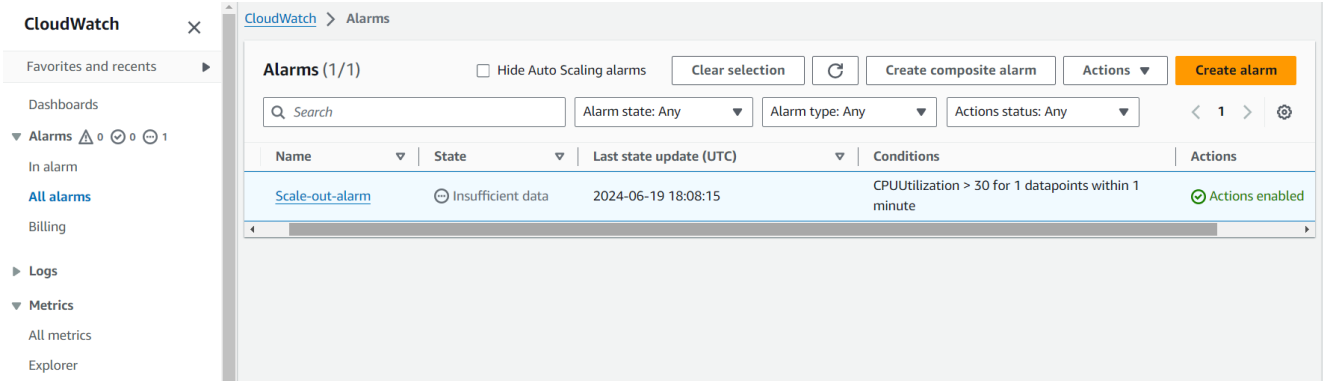
We can see the newly created server with new Ipv4 address (3.92.81.96)



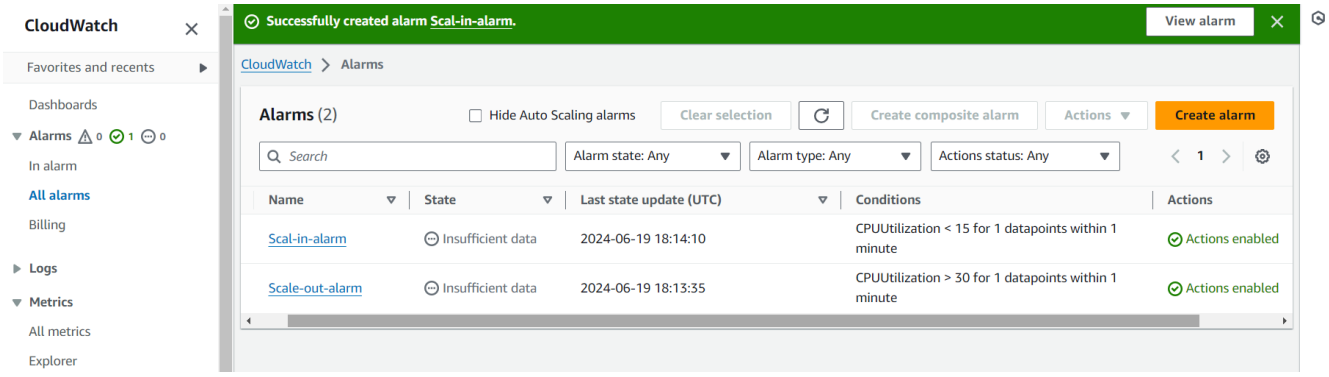
Step 7: AWS Cloud Watch Configuration.

a. Click Services -> Management and Governance -> Cloud Watch.

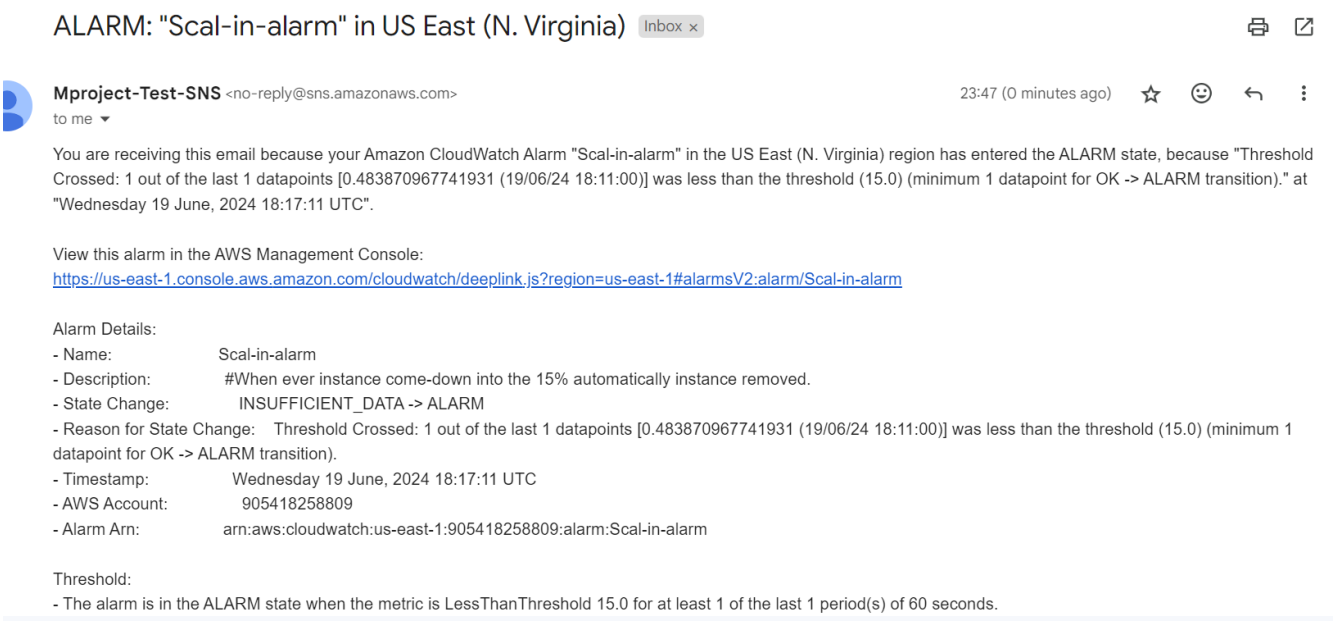
- Click All alarms -> create alarm.
- Click select matrix.
- Matrix –> EC2 –> per Instance Matrix.
- Filter the based on instance id
- Select CPU-Utilization
- Click select Matrix.
- Set statistic to maximum and period to 1 minute.
- Select static, let's create for scale-out operation, select greater and value to 30 and click next.
- Alarm state trigger – In-alarm.
- Select existing sns topic – Mproject-Test-SNS, then click next.
- Alarm name – Scale-out-alarm
- Add alarm description - # When ever CPU-Utilization threshold value crosses 30%, Automatically Auto scaling group create new instance.
- Click next review configuration and create alarm.



- Create a Scal-in- alarm following above steps and Select static, let's create for scale-in operation, select lower and value to 15 and click next.



All the details regarding the scale-in alarm is given in the email.



ALARM: "Scal-in-alarm" in US East (N. Virginia) Inbox ×

Mproject-Test-SNS <no-reply@sns.amazonaws.com> 23:47 (0 minutes ago)
to me

You are receiving this email because your Amazon CloudWatch Alarm "Scal-in-alarm" in the US East (N. Virginia) region has entered the ALARM state, because "Threshold Crossed: 1 out of the last 1 datapoints [0.483870967741931 (19/06/24 18:11:00)] was less than the threshold (15.0) (minimum 1 datapoint for OK -> ALARM transition)." at "Wednesday 19 June, 2024 18:17:11 UTC".

View this alarm in the AWS Management Console:
https://us-east-1.console.aws.amazon.com/cloudwatch/deeplink.js?region=us-east-1#alarmsV2:alarm/Scal-in-alarm

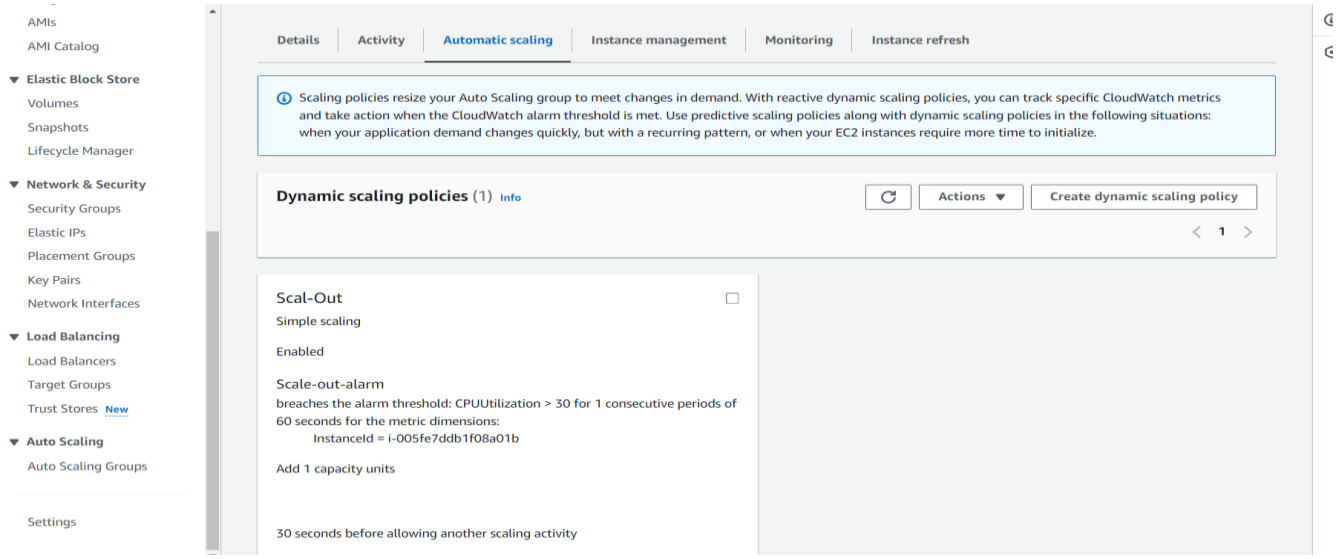
Alarm Details:
- Name: Scal-in-alarm
- Description: #When ever instance come-down into the 15% automatically instance removed.
- State Change: INSUFFICIENT_DATA -> ALARM
- Reason for State Change: Threshold Crossed: 1 out of the last 1 datapoints [0.483870967741931 (19/06/24 18:11:00)] was less than the threshold (15.0) (minimum 1 datapoint for OK -> ALARM transition).
- Timestamp: Wednesday 19 June, 2024 18:17:11 UTC
- AWS Account: 905418258809
- Alarm Arn: arn:aws:cloudwatch:us-east-1:905418258809:alarm:Scal-in-alarm

Threshold:
- The alarm is in the ALARM state when the metric is LessThanThreshold 15.0 for at least 1 of the last 1 period(s) of 60 seconds.

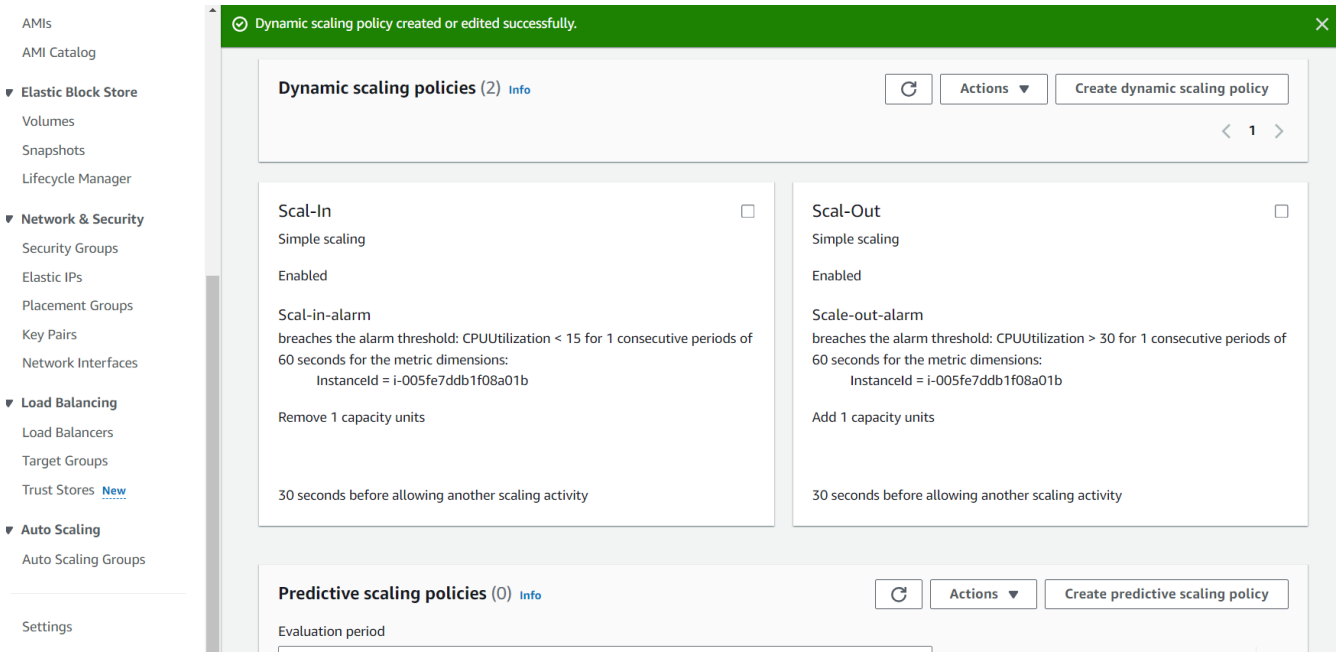
Step 8: Attach the Scal-in-alarm and Scal-out-alarm into Auto scaling group.

a. Click Auto scaling groups -> slect Mproject-Test-AutoScalingGroup -> automatic scaling -> create dynamic scaling policy.

- Click automatic scaling -> create dynamic scaling policy.
- Select policy type – simple scaling.
- Name – Scal-Out
- Select the Cloud Watch alarm: Scale-out-alarm
- Action: Add 1
- Time: 30 seconds and click create.



- Follow the above steps
- Name – Scal-in
- Select cloud watch alarm as : Scale-in-alarm
- Set Action: Remove 1
- create



Now successfully configured scal-in and scal-out policies.

Now let us increase the load on CPU.

Select instance, copy IPv4 (3.237.178.21) public address and set up putty.

Login as ec2-user

```
ec2-user@ip-20-0-1-130:~
login as: ec2-user
Authenticating with public key "imported-openssh-key"
Register this system with Red Hat Insights: insights-client --register
Create an account or view all your systems at https://red.ht/insights-dashboard
Last login: Wed Jun 19 16:19:48 2024 from 103.5.132.170
[ec2-user@ip-20-0-1-130 ~]$ top
```

Now let us run top command to see CPU Usage.

We can see %Cpu is 0.0. Let us increase the load now. Press q to quit.

```
ec2-user@ip-20-0-1-130:~
top - 18:48:46 up  2:04,  1 user,  load average: 0.00, 0.00, 0.00
Tasks: 110 total,   1 running, 109 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    765.7 total,    254.2 free,    295.3 used,    333.0 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    470.4 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1799 apache    20   0  981516  12628   6272 S   0.3   1.6   0:01.21 httpd
      1 root      20   0  172780  15984  10652 S   0.0   2.0   0:02.21 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     11 root      20   0       0      0      0 I   0.0   0.0   0:00.33 kworker/u30:1-events_unbound
     12 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_kthre
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_rude_
     14 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tasks_trace
     15 root      20   0       0      0      0 S   0.0   0.0   0:00.02 ksoftirqd/0
     16 root      20   0       0      0      0 S   0.0   0.0   0:01.55 pr/ttyS0
     17 root      20   0       0      0      0 S   0.0   0.0   0:00.68 pr/tty0
     18 root      20   0       0      0      0 I   0.0   0.0   0:00.07 rcu_preempt
     19 root      rt   0       0      0      0 S   0.0   0.0   0:00.01 migration/0
     20 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     22 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     24 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kdevtmpfs
     25 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 inet_frag_wq
     26 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kauditd
     27 root      20   0       0      0      0 S   0.0   0.0   0:00.00 khungtaskd
     28 root      20   0       0      0      0 I   0.0   0.0   0:00.23 kworker/u30:2-events_unbound
```

Execute the below command 5 times to increase the CPU Utilization.

yes > /dev/null &

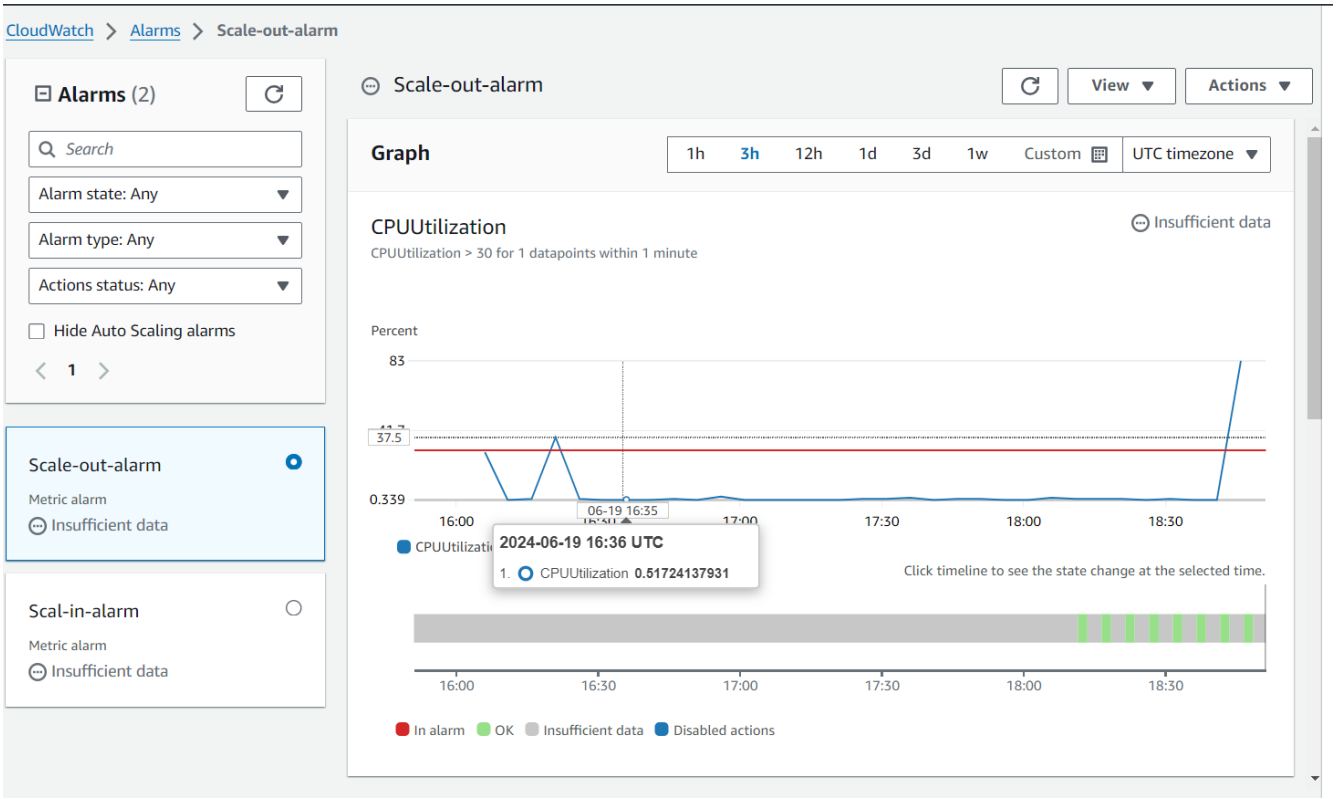
```
[ec2-user@ip-20-0-1-130 ~]$ yes > /dev/null &
[1] 2164
[ec2-user@ip-20-0-1-130 ~]$
[ec2-user@ip-20-0-1-130 ~]$
[ec2-user@ip-20-0-1-130 ~]$ yes > /dev/null &
[2] 2165
[ec2-user@ip-20-0-1-130 ~]$ yes > /dev/null &
[3] 2166
[ec2-user@ip-20-0-1-130 ~]$ yes > /dev/null &
[4] 2167
[ec2-user@ip-20-0-1-130 ~]$ yes > /dev/null &
[5] 2168
[ec2-user@ip-20-0-1-130 ~]$ yes > /dev/null &
[6] 2169
[ec2-user@ip-20-0-1-130 ~]$
```

Run the top command check the CPU Utilization.

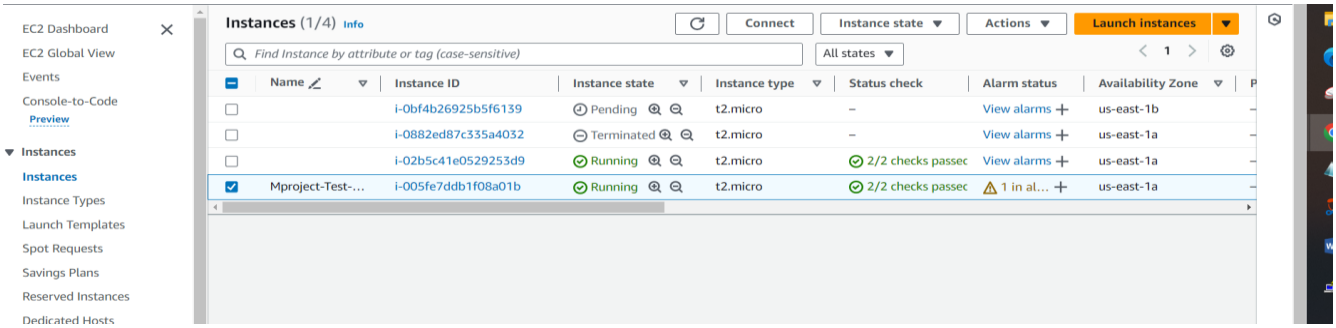
```
ec2-user@ip-20-0-1-130:~
top - 18:50:53 up  2:06,  1 user,  load average: 2.77, 0.71, 0.24
Tasks: 115 total,   7 running, 108 sleeping,   0 stopped,   0 zombie
%Cpu(s): 60.1 us, 39.2 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.7 hi,  0.0 si,  0.0 st
MiB Mem :    765.7 total,    245.5 free,    303.9 used,    333.1 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.    461.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2164 ec2-user  20   0    5580   1664   1664 R  16.6   0.2   0:10.89 yes
   2165 ec2-user  20   0    5580   1664   1664 R  16.6   0.2   0:06.50 yes
   2167 ec2-user  20   0    5580   1792   1664 R  16.6   0.2   0:05.83 yes
   2168 ec2-user  20   0    5580   1664   1664 R  16.6   0.2   0:05.53 yes
   2169 ec2-user  20   0    5580   1664   1664 R  16.6   0.2   0:05.29 yes
   2166 ec2-user  20   0    5580   1792   1664 R  16.3   0.2   0:06.12 yes
   2076 ec2-user  20   0   19596   6952   5120 S   0.3   0.9   0:00.03 sshd
      1 root      20   0  172780  15984  10652 S   0.0   2.0   0:02.21 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_flushwq
```

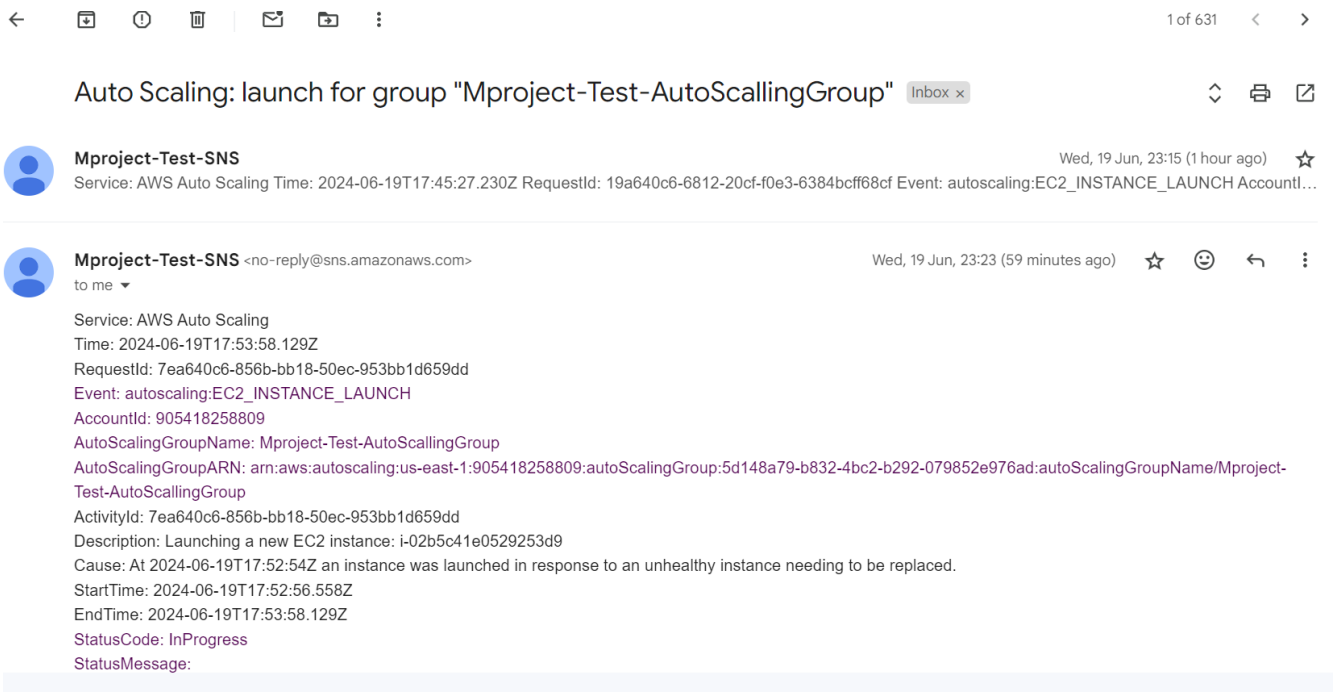
Now from graph we can see Scale-out has crossed the threshold value of 30%.



Which means a new instance will be created as demand is crossed. See the instance has changed from 1 to 2.



We have received 2 emails. One for the scale-out alarm, and other for new instance creation.



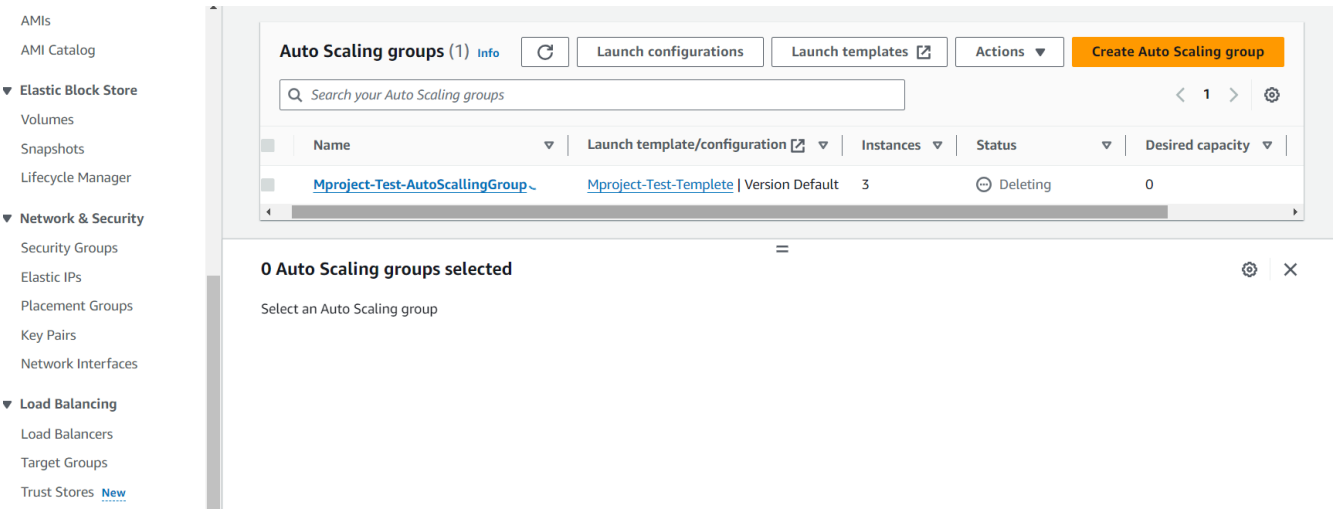
Completed has been completed and tested successfully.

Chapter-4

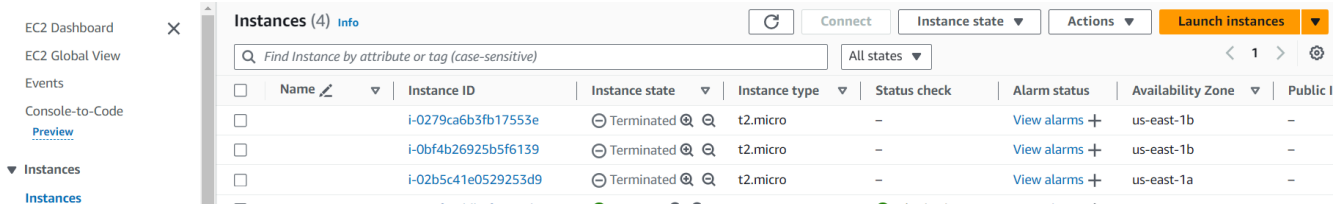
CLEAN-UP

1. Delete Auto Scaling Groups.

Select Group (Mproject-Test-AutoScalingGroup) -> Actions -> Delete

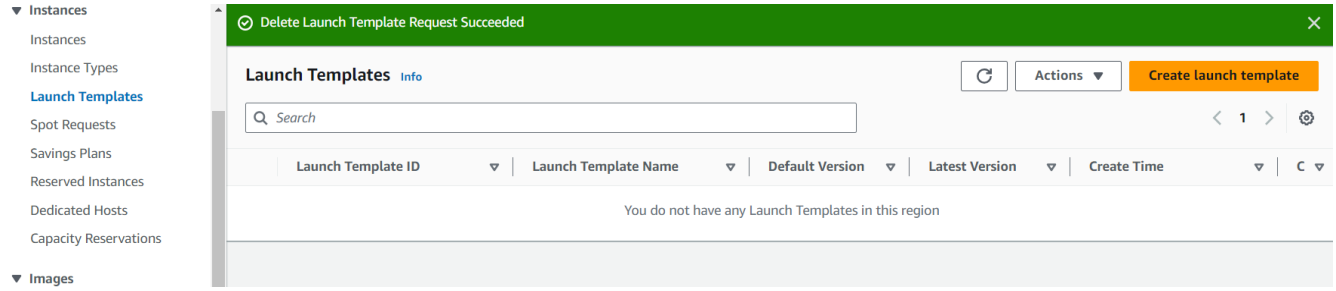


we can see the instance has also terminated.



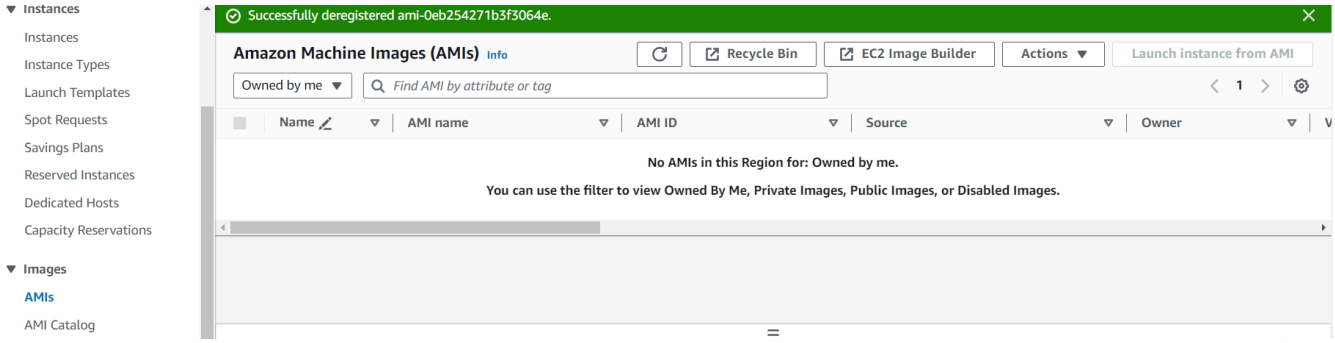
2. Delete Launch Templates.

Select Launch Template -> Actions -> Delete template

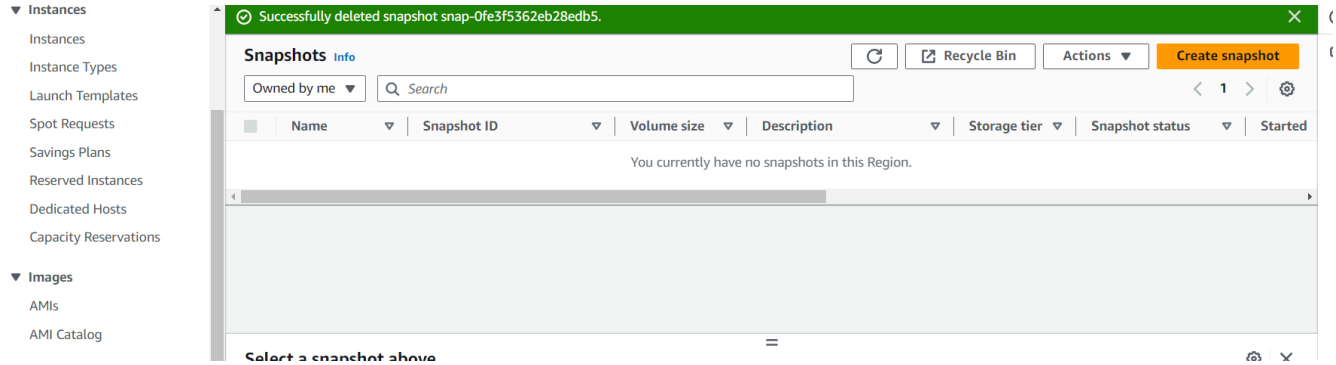


3. Delete Images

AMI -> select Image -> actions -> de-register AMI

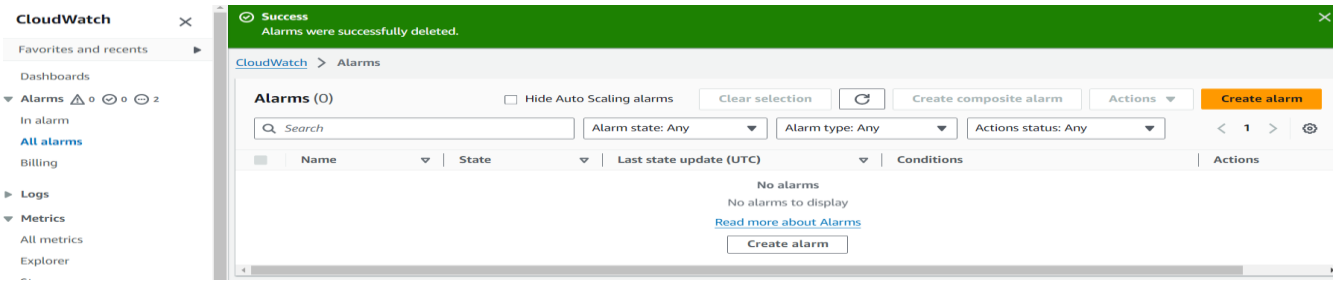


4. Delete Snapshots.



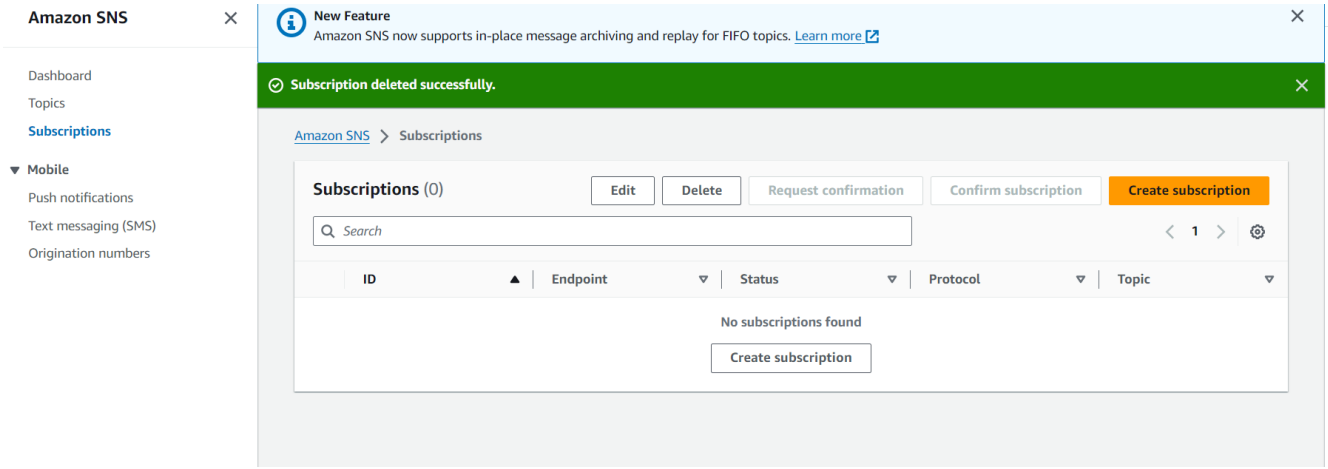
5. Delete Alarms.

Alarms -> all alarms -> select both alarms -> actions -> delete

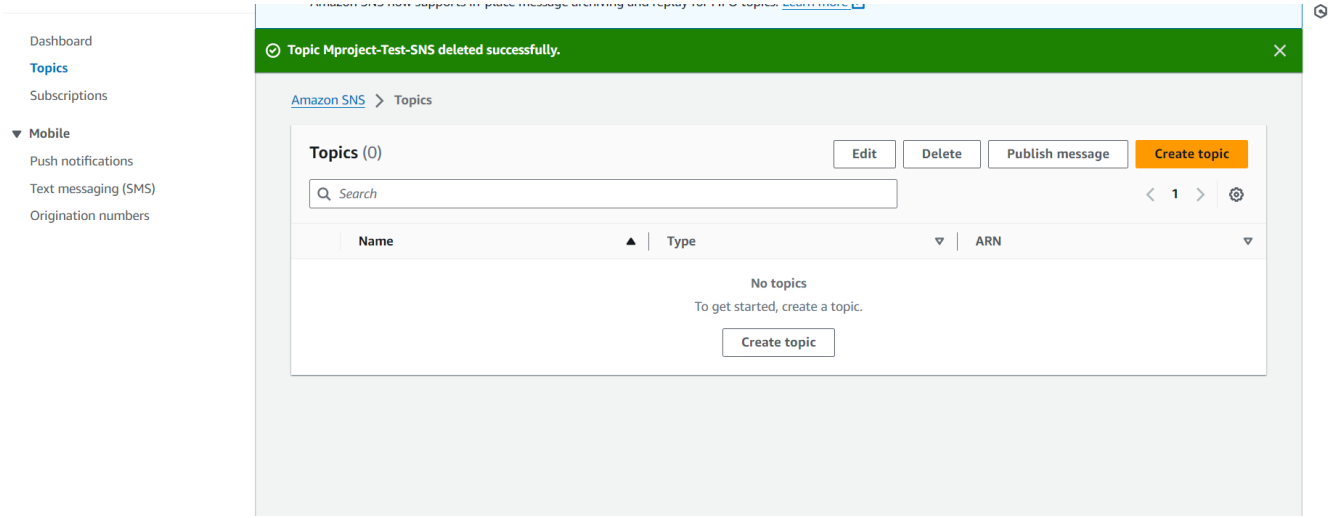


6. Delete SNS.

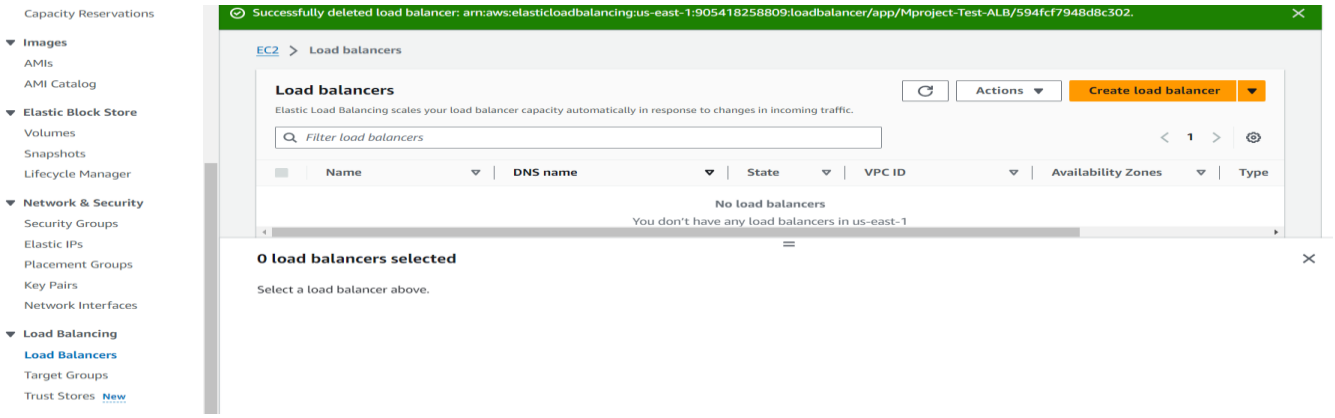
Delete Subscription



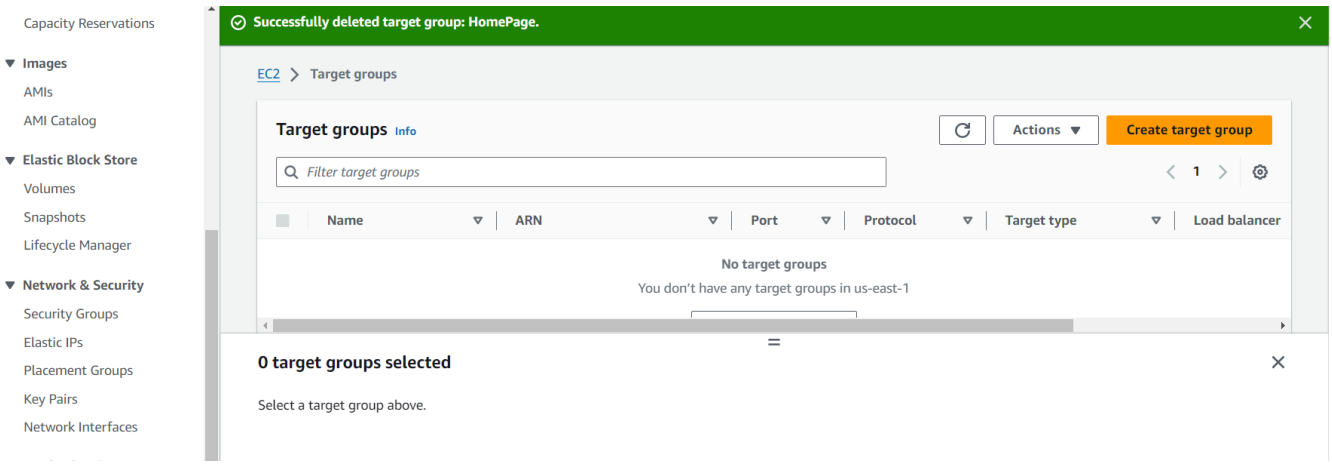
7. Delete Topic



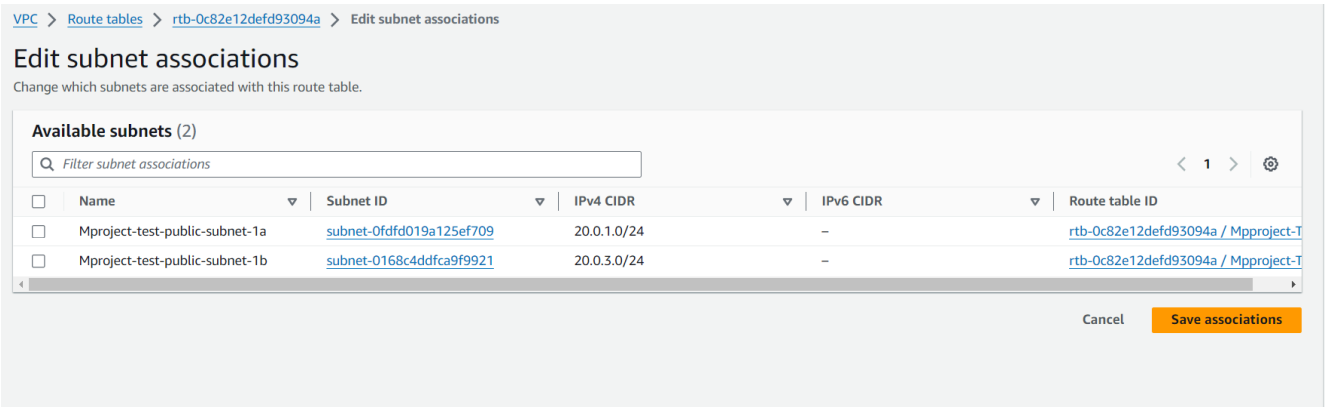
8. .Delete Load Balancer



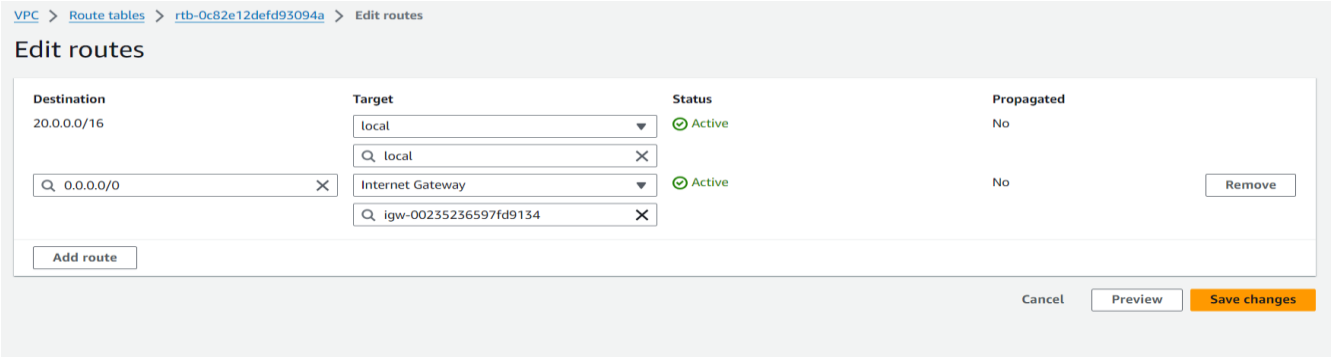
9. Delete Target group



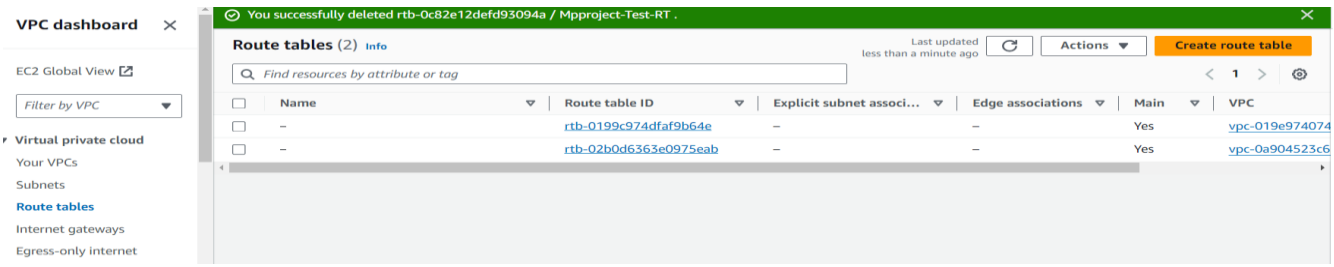
10. Remove association subnets in route table.



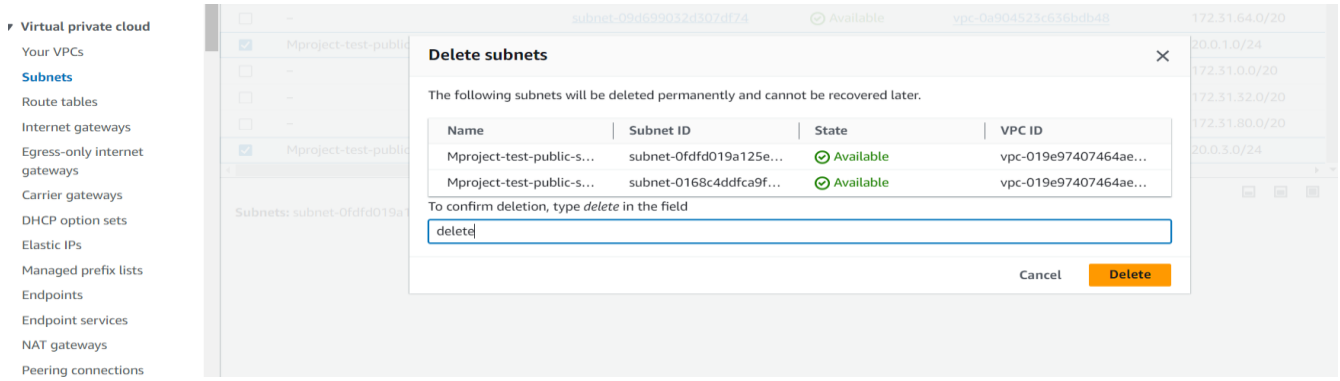
11. Remove the internet route in route table



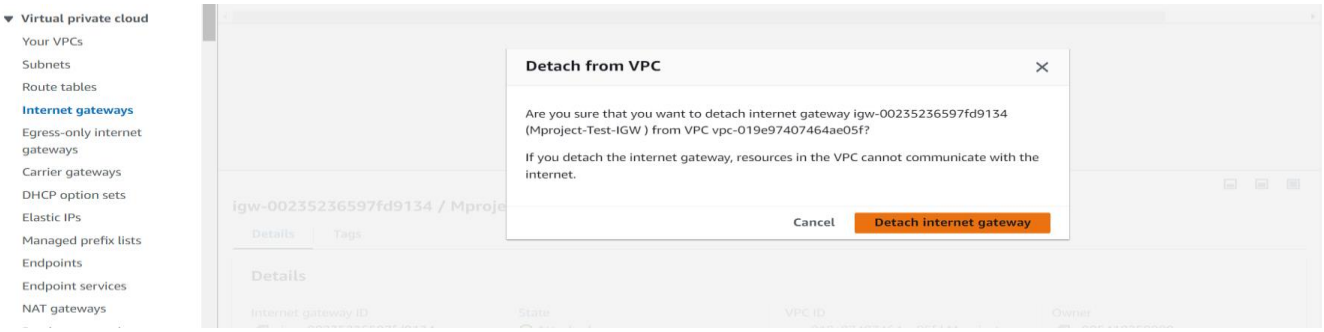
12. Delete the route table.



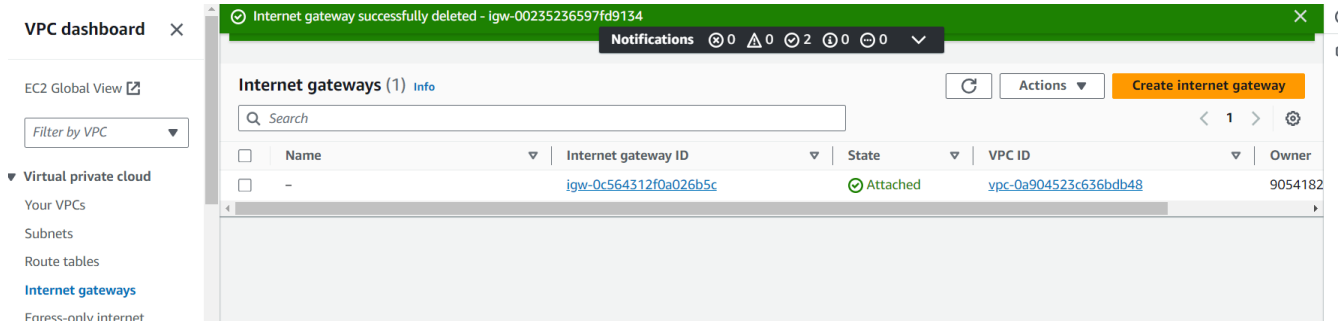
13. Delete the subnets.



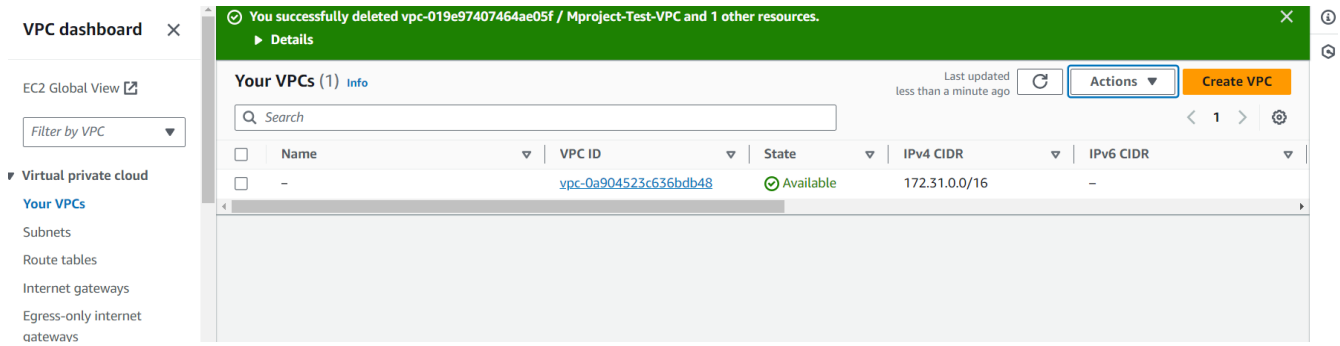
14. Detech Internet gateway to VPC.



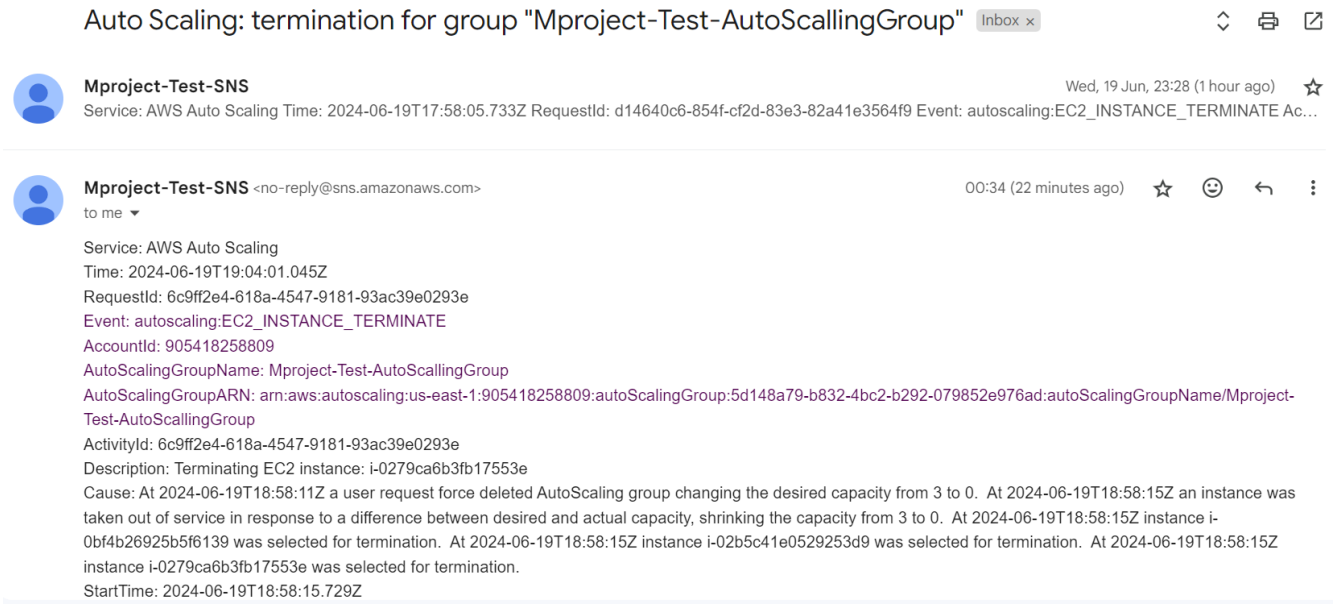
15. Delete internet gateway.



16. Delete VPC



17. We have received the email of the termination process.



Auto Scaling: termination for group "Mproject-Test-AutoScallingGroup" Inbox ×

Mproject-Test-SNS Wed, 19 Jun, 23:28 (1 hour ago)
Service: AWS Auto Scaling Time: 2024-06-19T17:58:05.733Z RequestId: d14640c6-854f-cf2d-83e3-82a41e3564f9 Event: autoscaling:EC2_INSTANCE_TERMINATE Ac...

Mproject-Test-SNS <no-reply@sns.amazonaws.com> 00:34 (22 minutes ago)
to me

Service: AWS Auto Scaling
Time: 2024-06-19T19:04:01.045Z
RequestId: 6c9ff2e4-618a-4547-9181-93ac39e0293e
Event: autoscaling:EC2_INSTANCE_TERMINATE
AccountId: 905418258809
AutoScalingGroupName: Mproject-Test-AutoScallingGroup
AutoScalingGroupARN: arn:aws:autoscaling:us-east-1:905418258809:autoScalingGroup:5d148a79-b832-4bc2-b292-079852e976ad:autoScalingGroupName/Mproject-Test-AutoScallingGroup
ActivityId: 6c9ff2e4-618a-4547-9181-93ac39e0293e
Description: Terminating EC2 instance: i-0279ca6b3fb17553e
Cause: At 2024-06-19T18:58:11Z a user request force deleted AutoScaling group changing the desired capacity from 3 to 0. At 2024-06-19T18:58:15Z an instance was taken out of service in response to a difference between desired and actual capacity, shrinking the capacity from 3 to 0. At 2024-06-19T18:58:15Z instance i-0bf4b26925b5f6139 was selected for termination. At 2024-06-19T18:58:15Z instance i-02b5c41e0529253d9 was selected for termination. At 2024-06-19T18:58:15Z instance i-0279ca6b3fb17553e was selected for termination.
StartTime: 2024-06-19T18:58:15.729Z

Chapter-5

CONCLUSION

This project successfully demonstrates the practical application of several AWS services to build a scalable and efficient web hosting solution. By following the detailed steps outlined, we achieved the creation and configuration of AWS VPC SNS for notifications, custom AMIs for tailored server environments, EC2 instances for compute power, and a load balancer to distribute traffic effectively. Additionally, we implemented Auto Scaling Groups and Cloud Watch alarms to ensure the system can dynamically adjust to changing demand, maintaining performance while optimizing costs.

The completion of this project not only highlights the power and flexibility of AWS services but also underscores the importance of automation and monitoring in modern cloud infrastructure. Through careful planning and execution, we have created a resilient web hosting environment capable of adapting to real-world usage patterns, providing a robust foundation for future cloud-based projects