CLOUD ARCHITECTURE ASSGINMENT -1

DEEPANSHU

CLOUD ARCHITECTURE

DATE: 30 -10 -2024

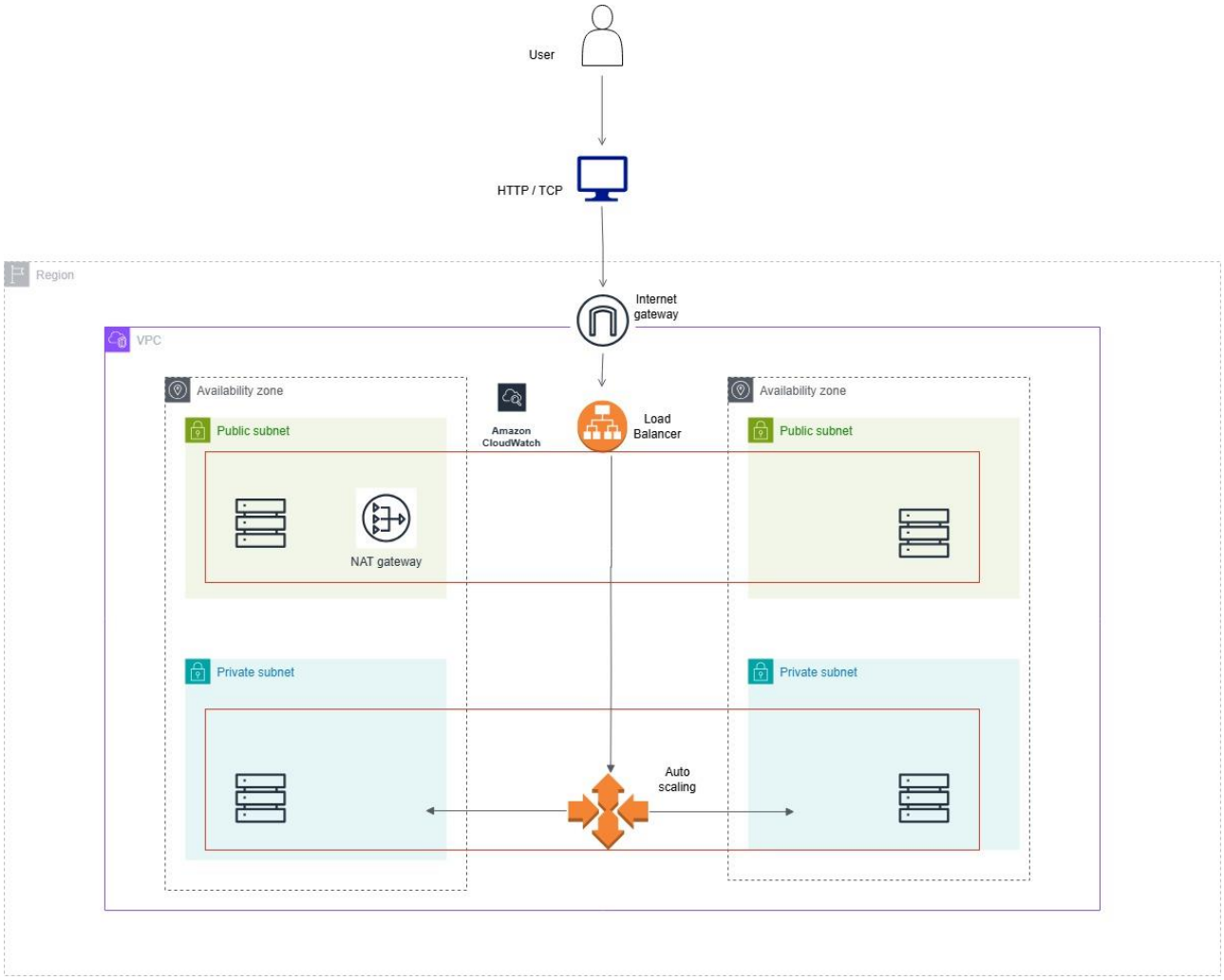
Academic Integrity Declaration

I certify that this assignment is all my own work and contains no plagiarism. By submitting this assignment, I agree to the following terms: Any text, diagrams or other material copied or generated from other sources (including, but not limited to, books, journals and the internet) have been clearly acknowledged and referenced as such in the text by the use of ‘quotation marks’ (or indented italics for longer quotations) followed by a reference to the original source either in the text or in a footnote/endnote. These details are then confirmed by a fuller reference in the bibliography. I have read the section on plagiarism in the SETU Academic Misconduct Policy (Appendix 1) and I understand that only assignments which are free of plagiarism will be awarded marks. I further understand that breaches of this policy can lead to sanctions up to the suspension or permanent expulsion of students in serious cases.

Signed: Deepanshu

TABLE OF CONTENTS

Architecture diagram:



This is a fundamental illustration of my application running on the AWS console.

The user transmits a request to the application via the HTTP/TCP port using the web browser.

Internet gateway: Then the request goes into the internet gateway, which plays an important role in communication between your VPC and the internet.

VPC: The Virtual Private Cloud is the entire network where all your application infrastructure components happened.

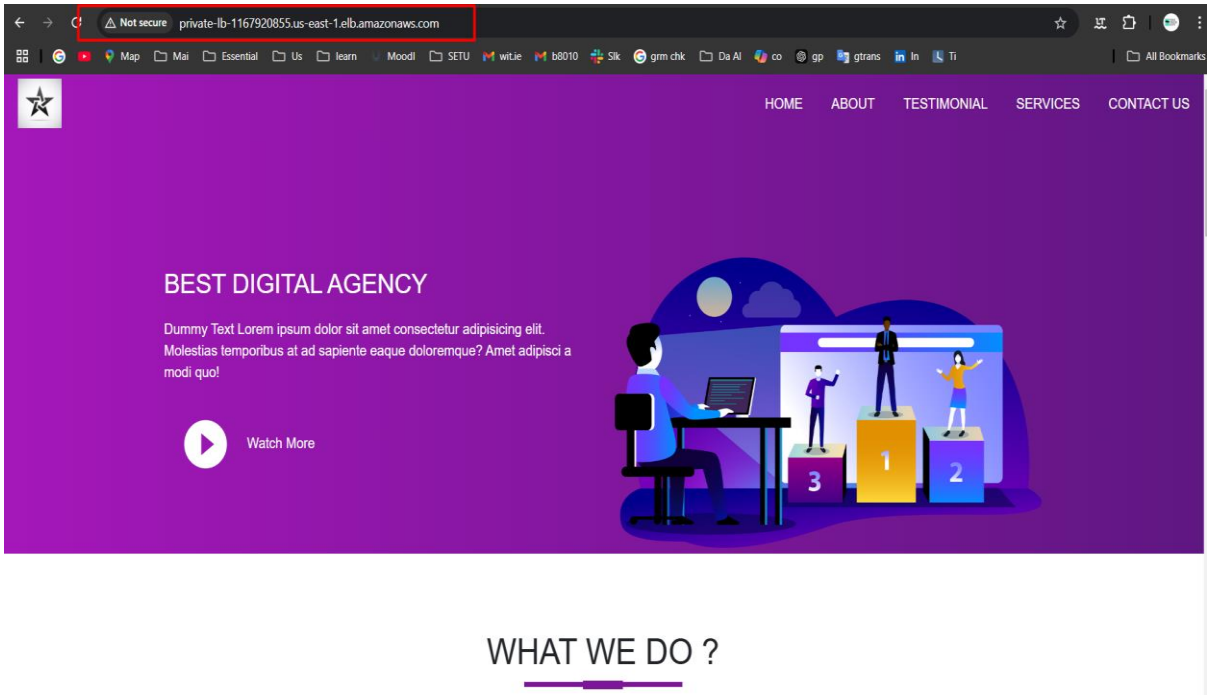
Load Balancer: After that, traffic comes into the load balancer and is distributed across multiple servers within the application according to your configuration. This ensures high availability with auto scaling group policy and manages application failovers.

Auto scaling: Inside that, we assign launch template AMI, this AMI of the EC2 instance where application have when traffic load come on. This AMI defines auto scaling policy rules to optimize and different availability zone and subnets so application handle traffic with the help of load balancer.

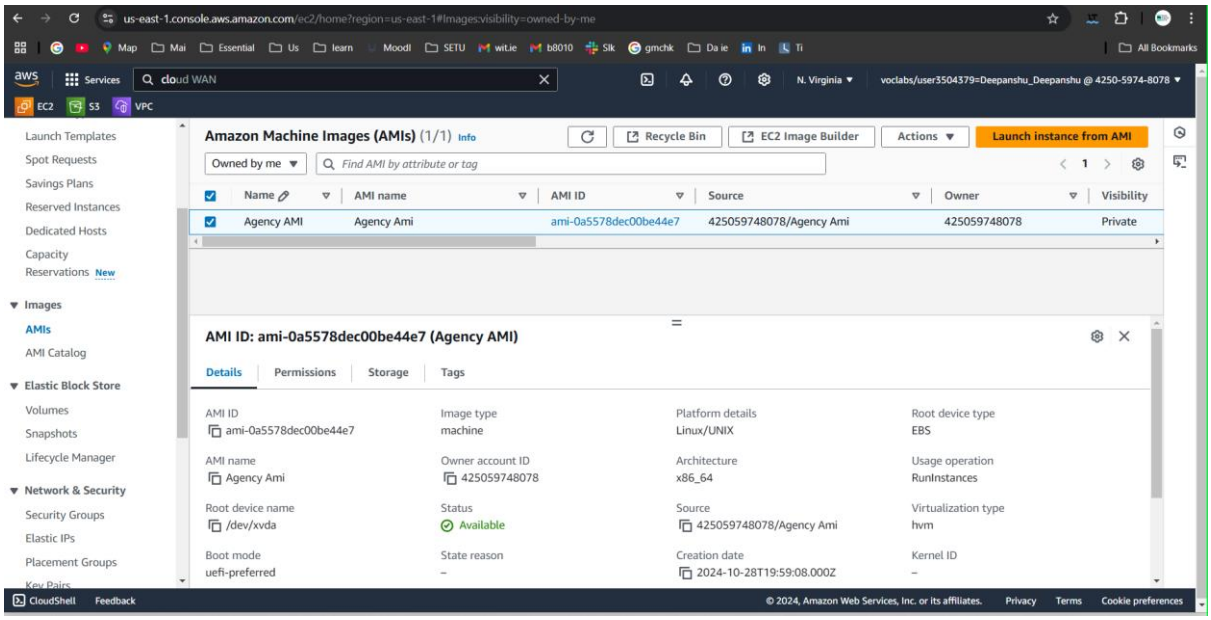
Nat Gateway: It provides access to your application resources from your public subnet to your private subnet.

Amazon Cloud Watch: It helps us to monitor our application and provide operational insights by tracking metrics, logging, and triggering alerts for specific conditions.

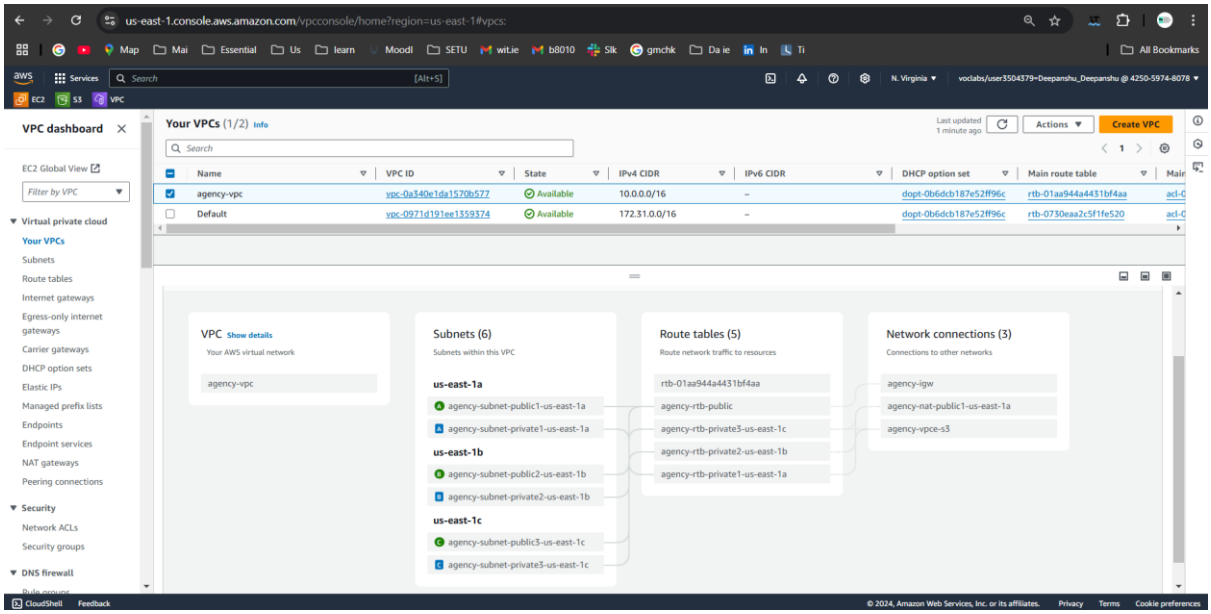
Application home page



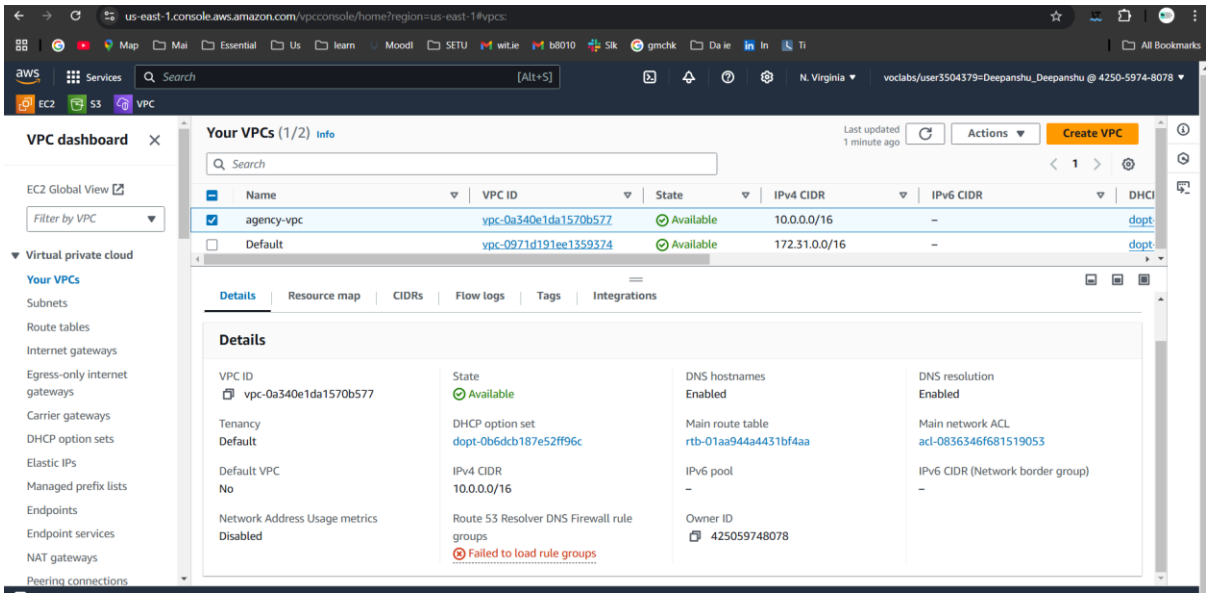
AMI



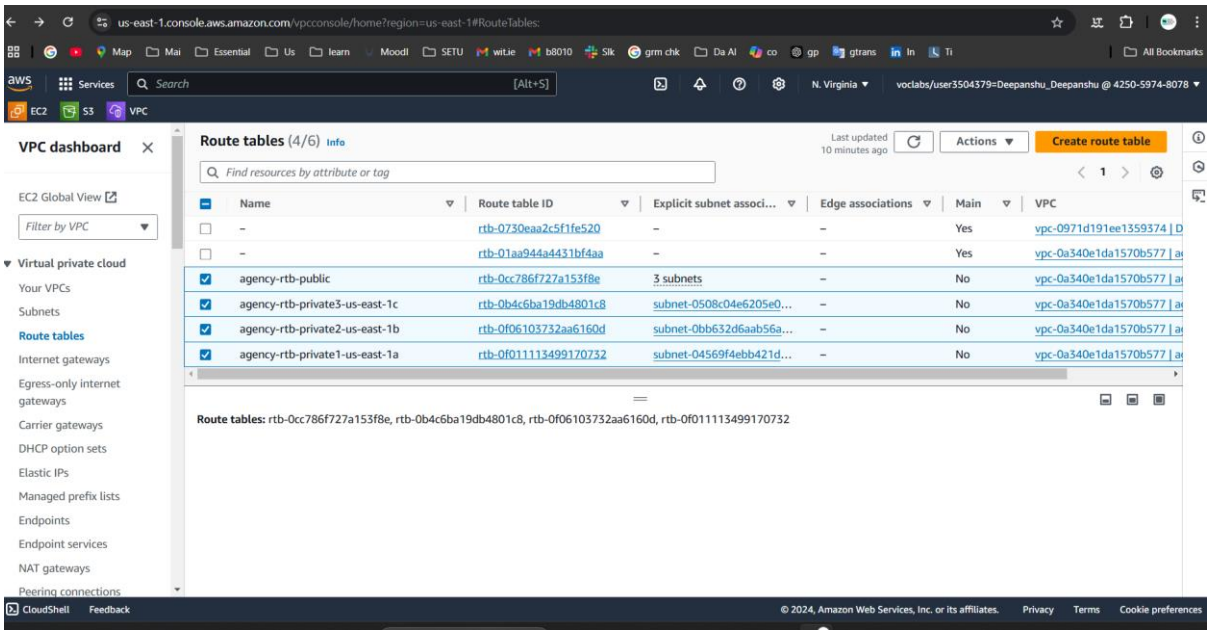
VPC : Flow chat



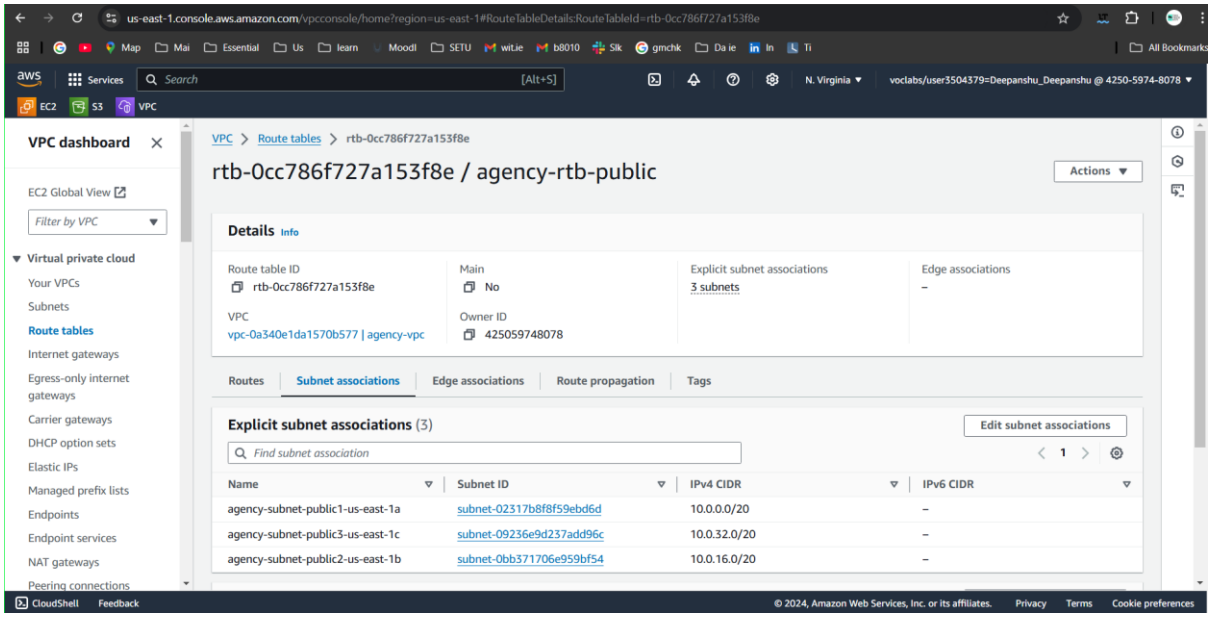
VPC details:



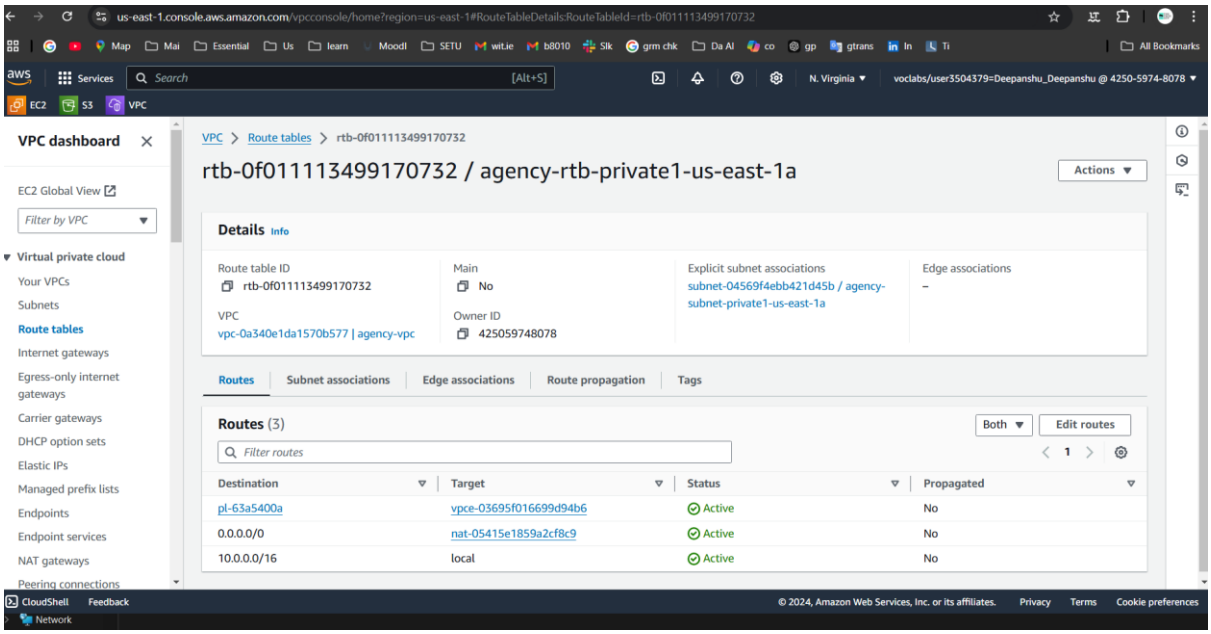
Route table



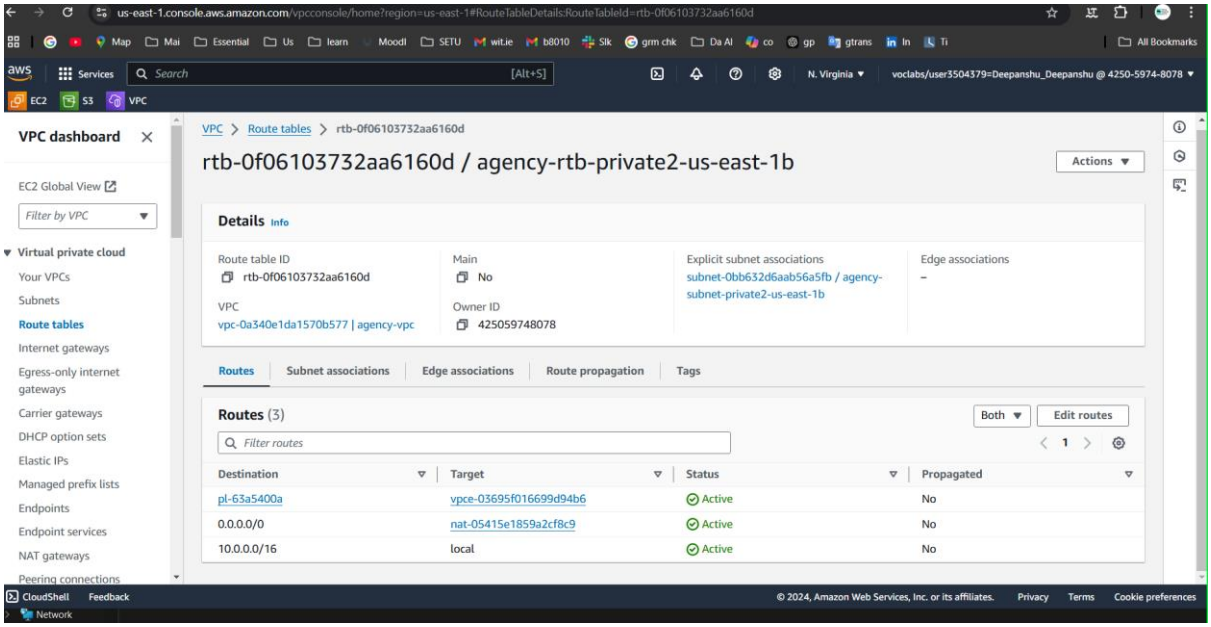
Public route table



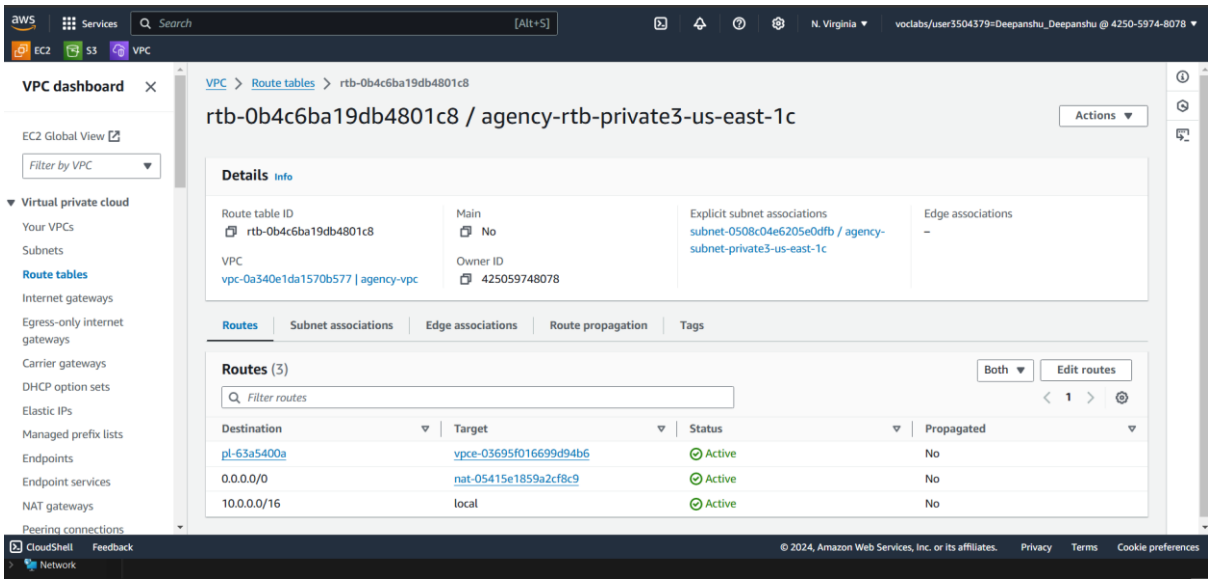
Private us-east- 1a



Private us-east-1b

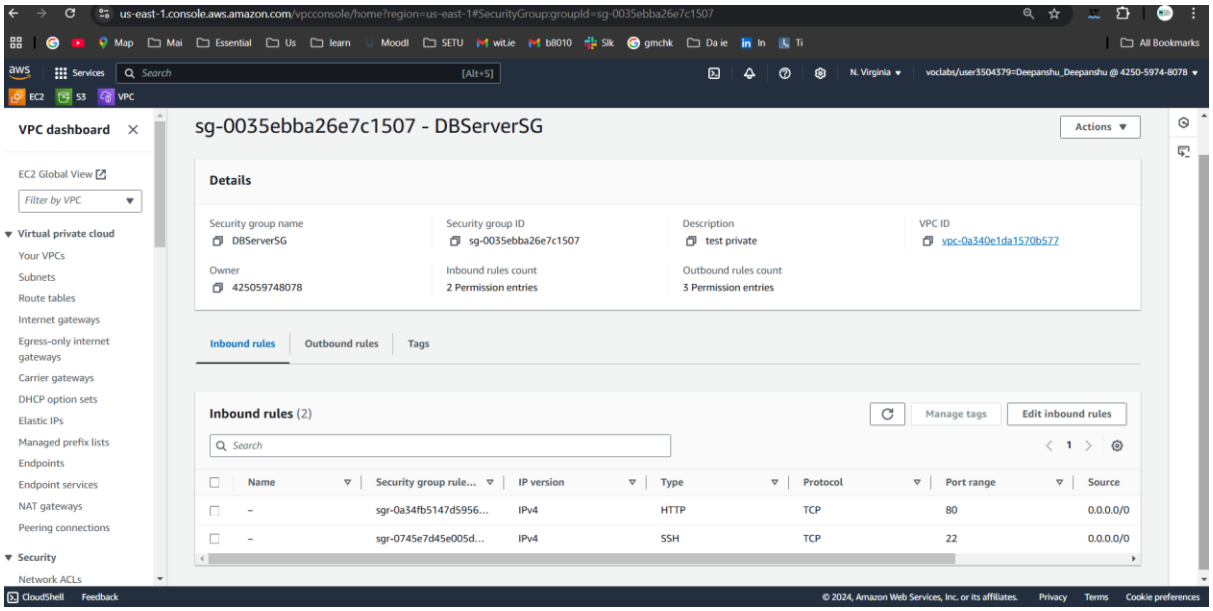


Private us-east-1c

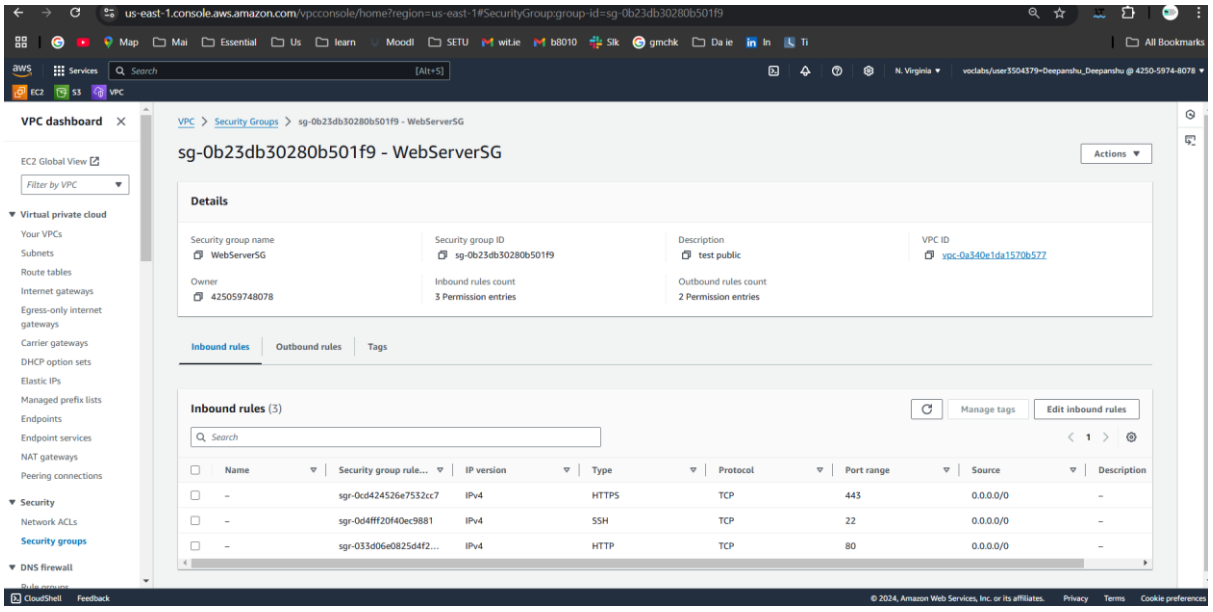


Security group

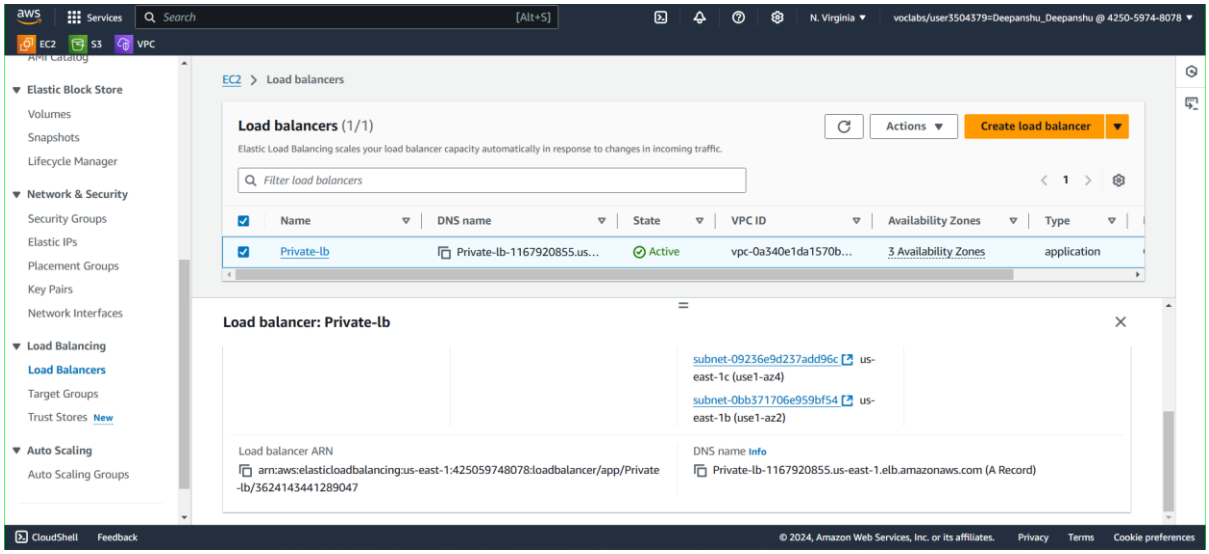
Private



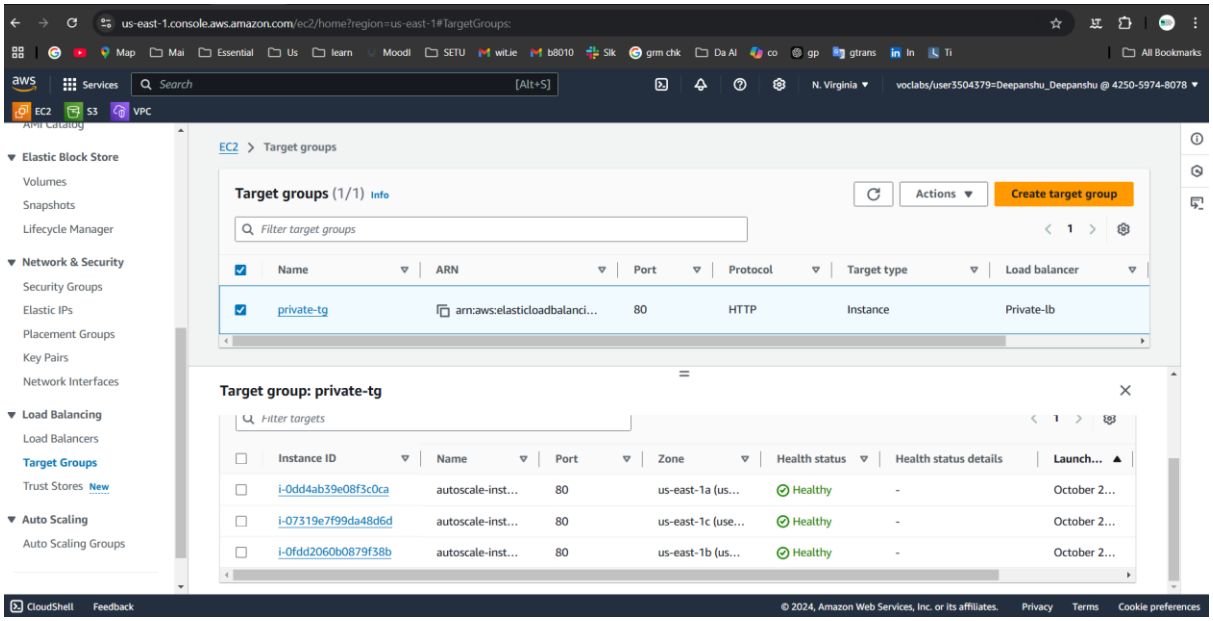
Public



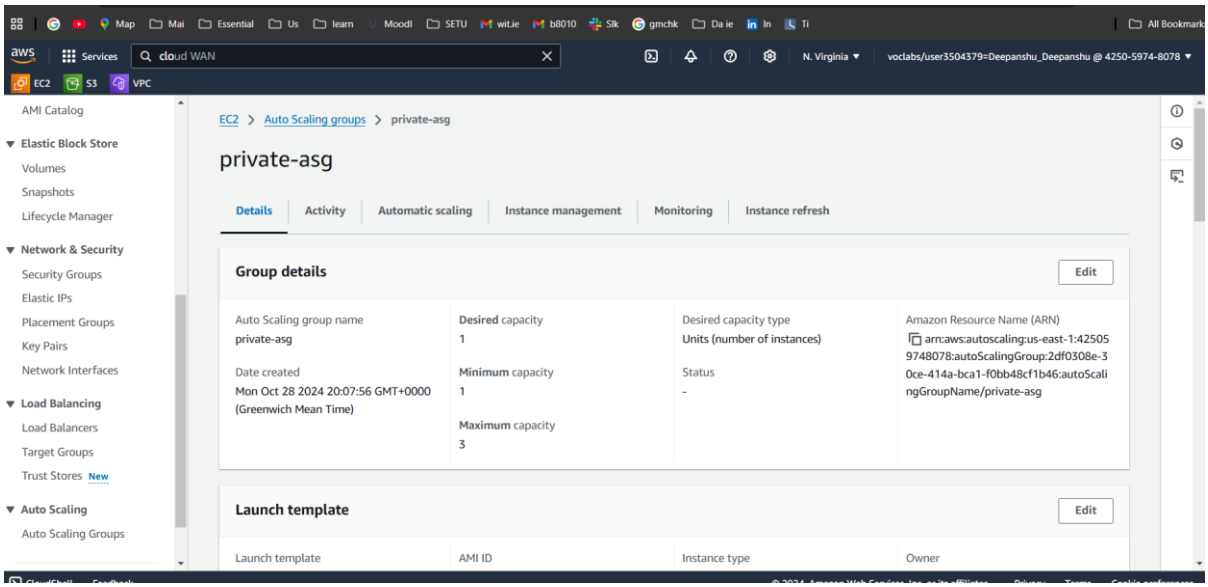
Load Balancer



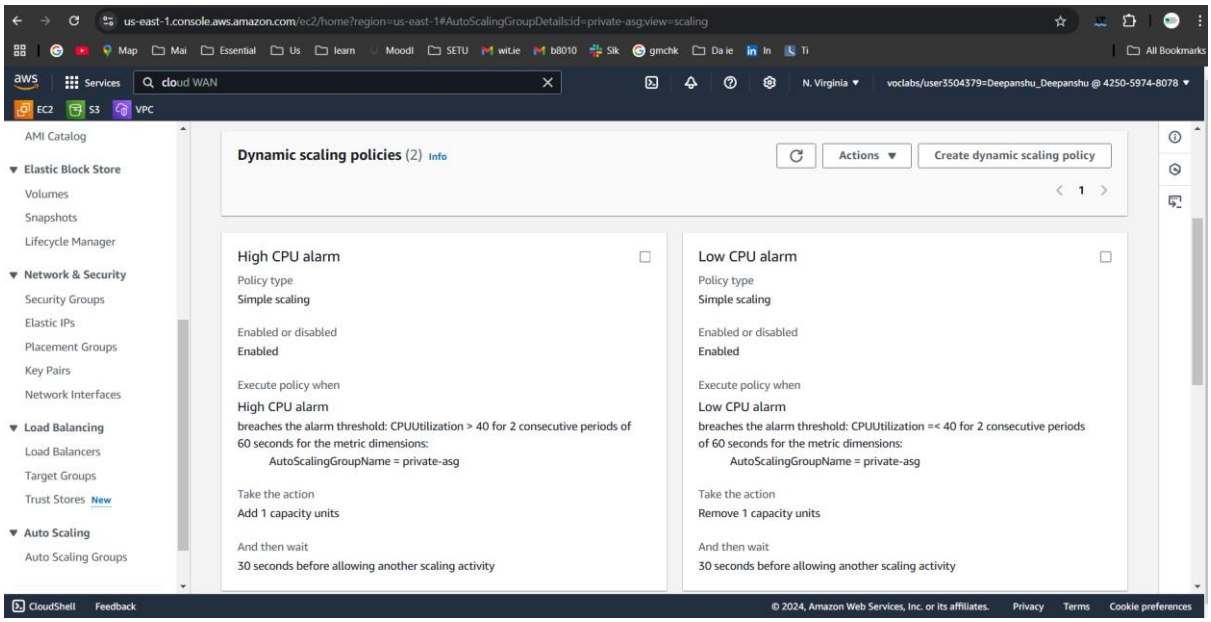
Target group showing registered targets



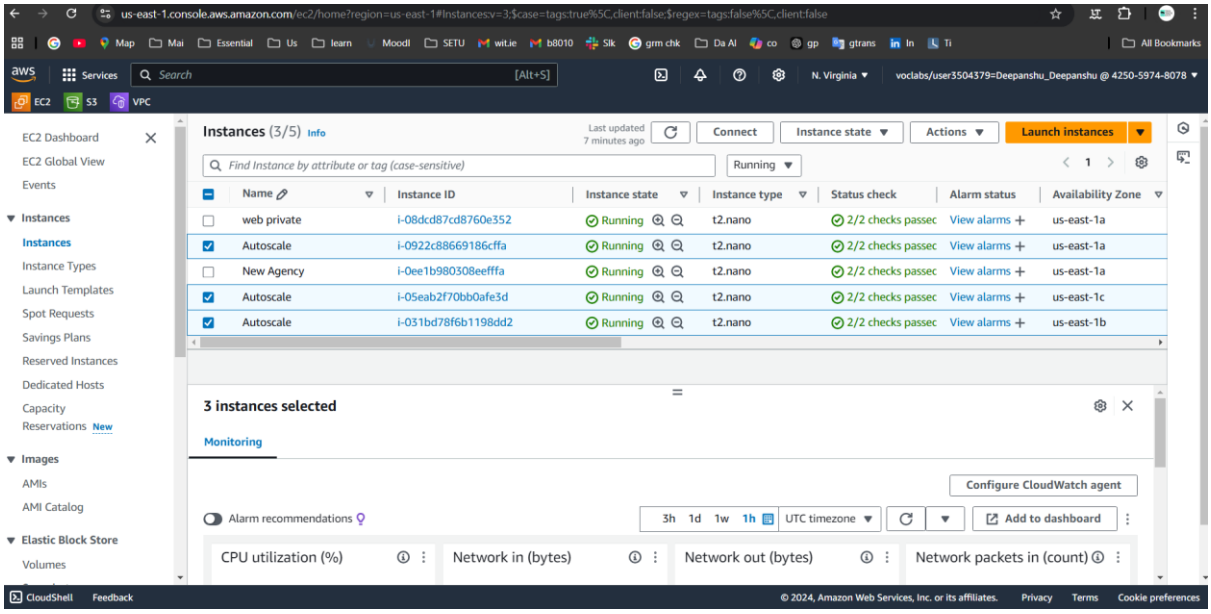
Auto Scaling Group



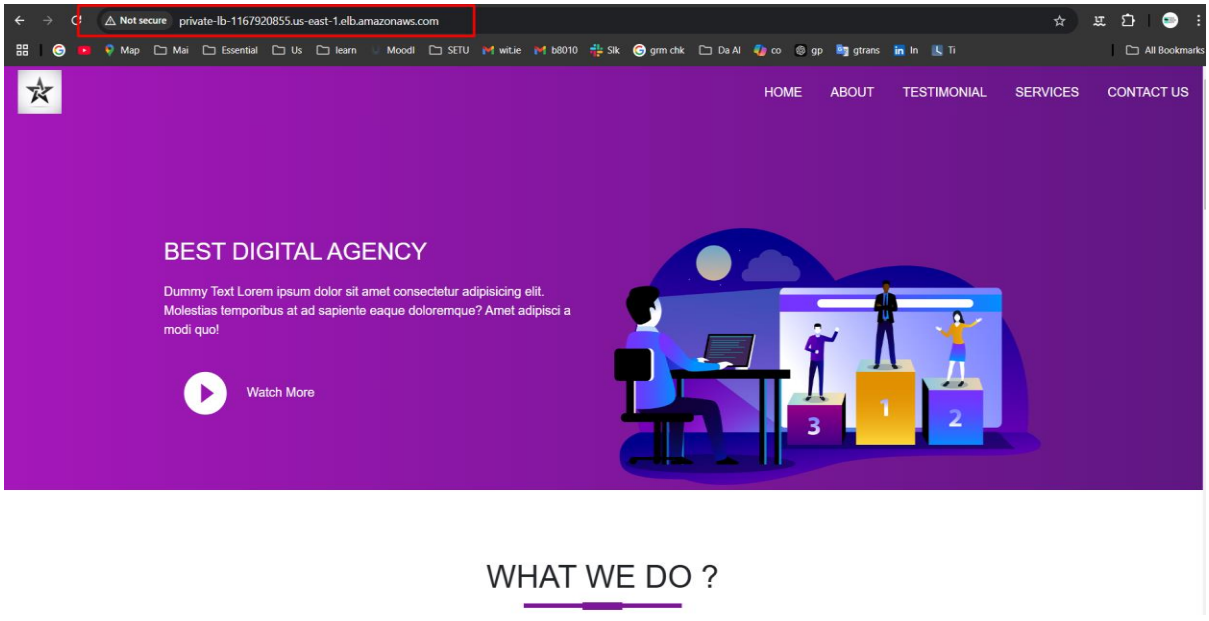
Scaling policies



After running **while** loop and **top** command auto scale instance trigger

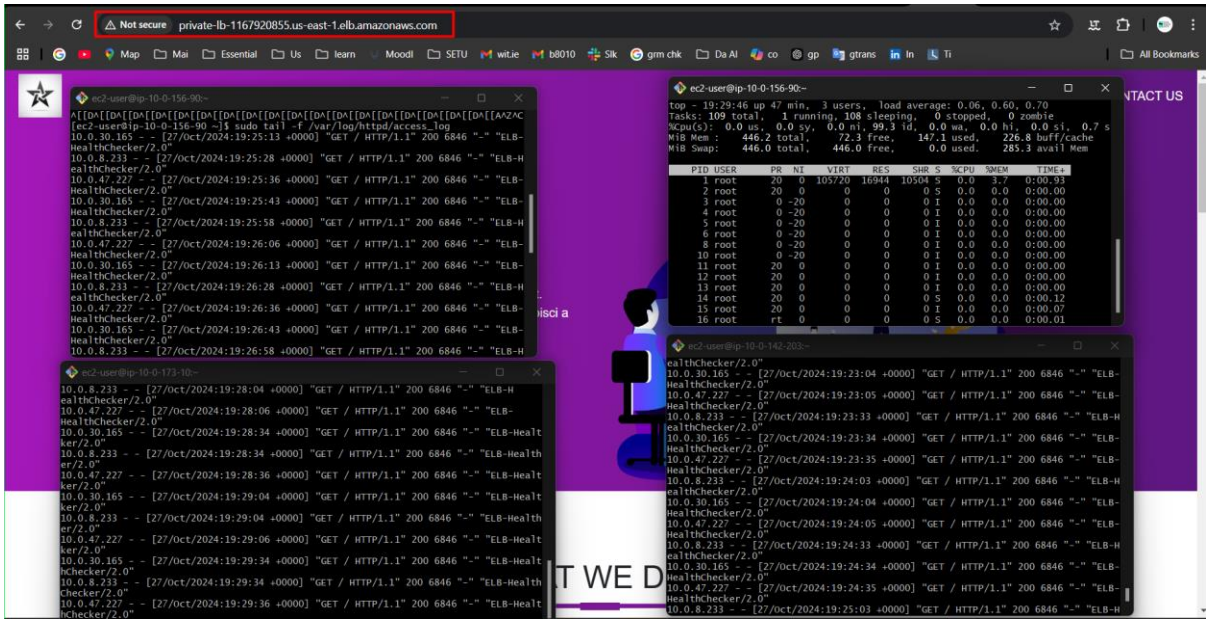


Load balancer working

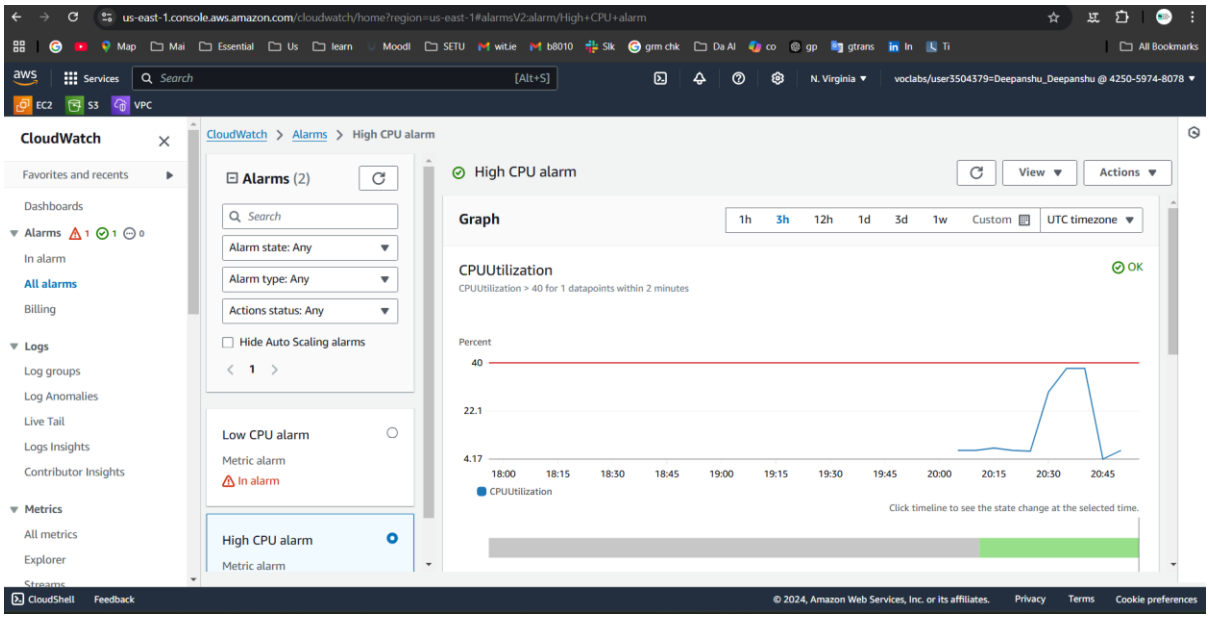


WHAT WE DO ?

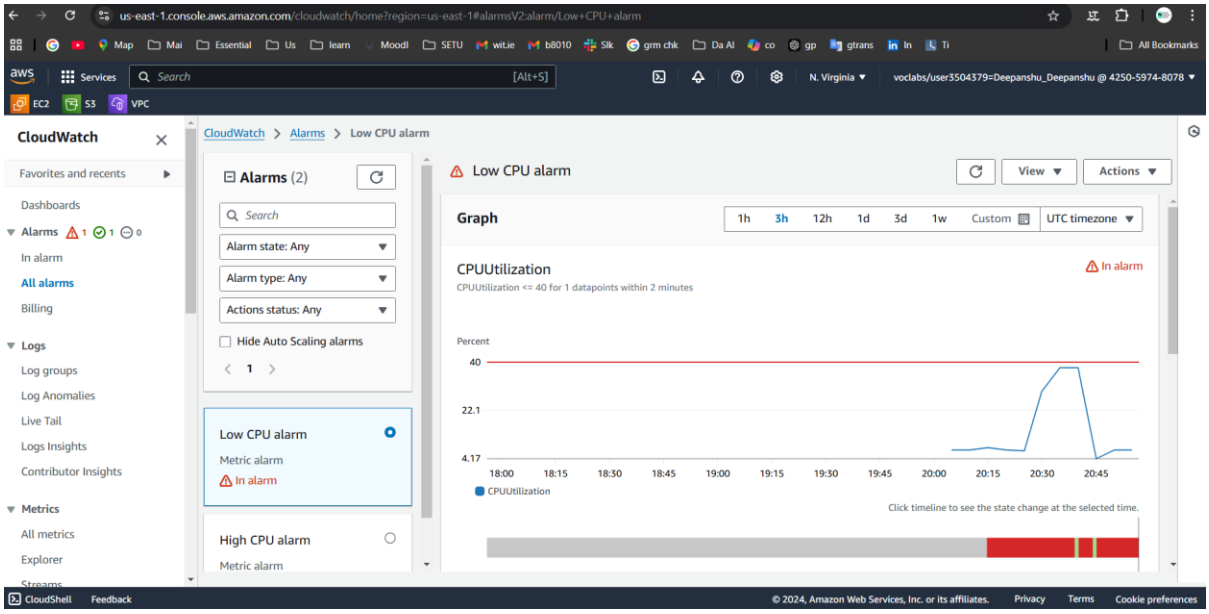
Load balancer working action using test traffic on each auto scale instance trigger.



CloudWatch High CPU alarm

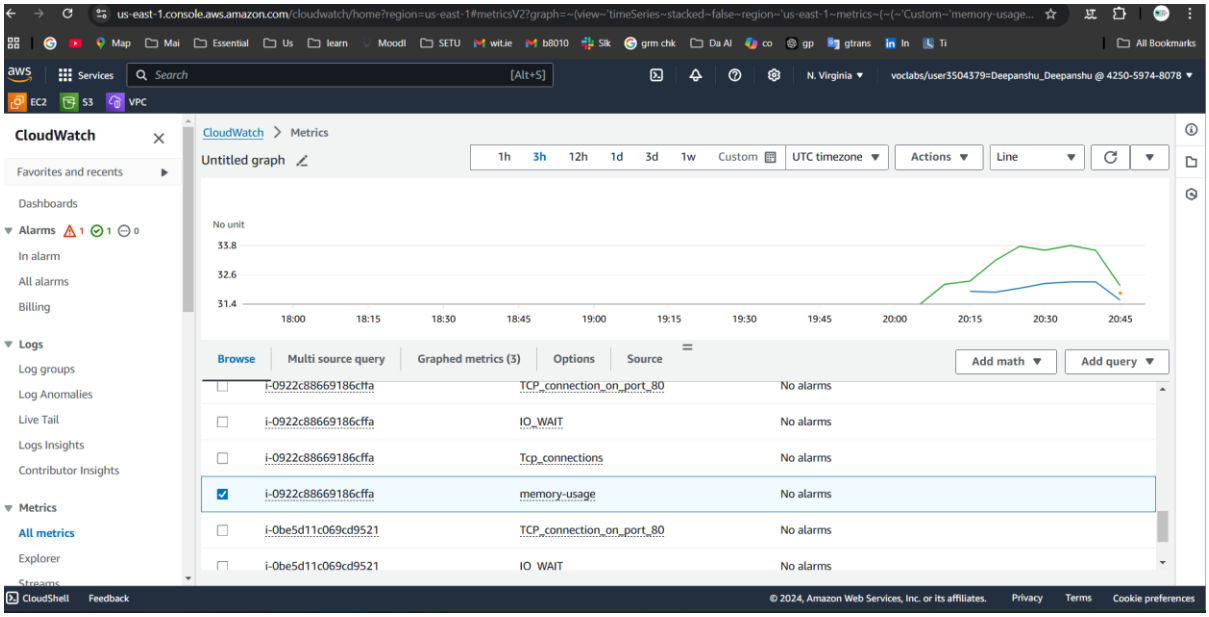


CloudWatch Low CPU alarm

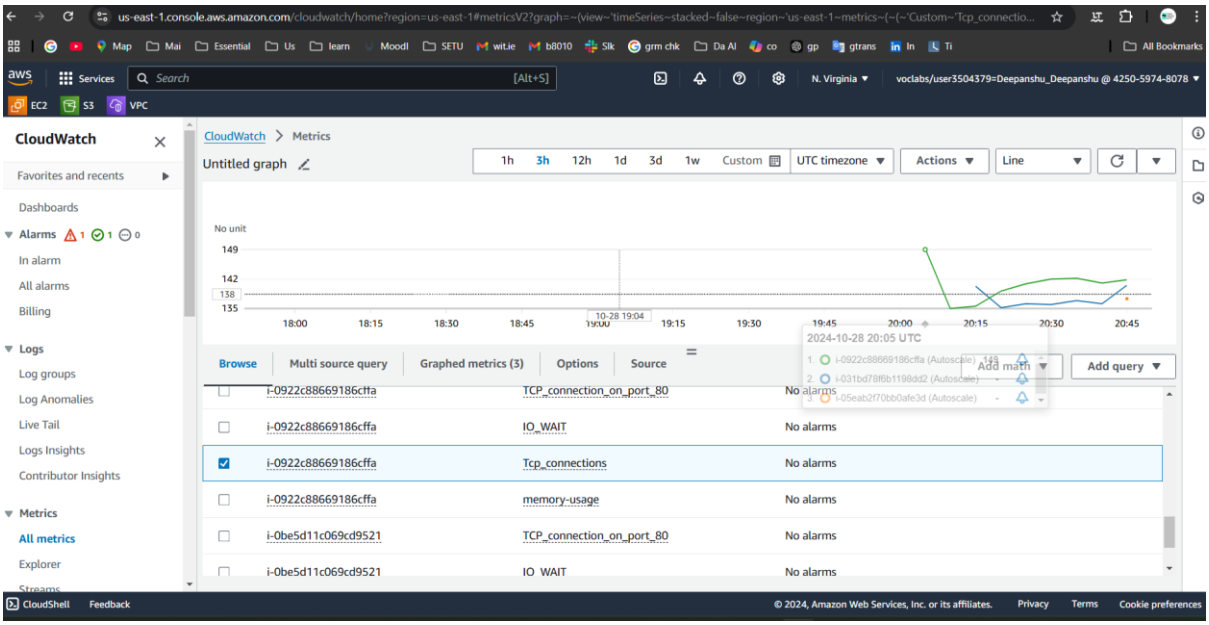


CloudWatch custom metrics data of auto scale instance:

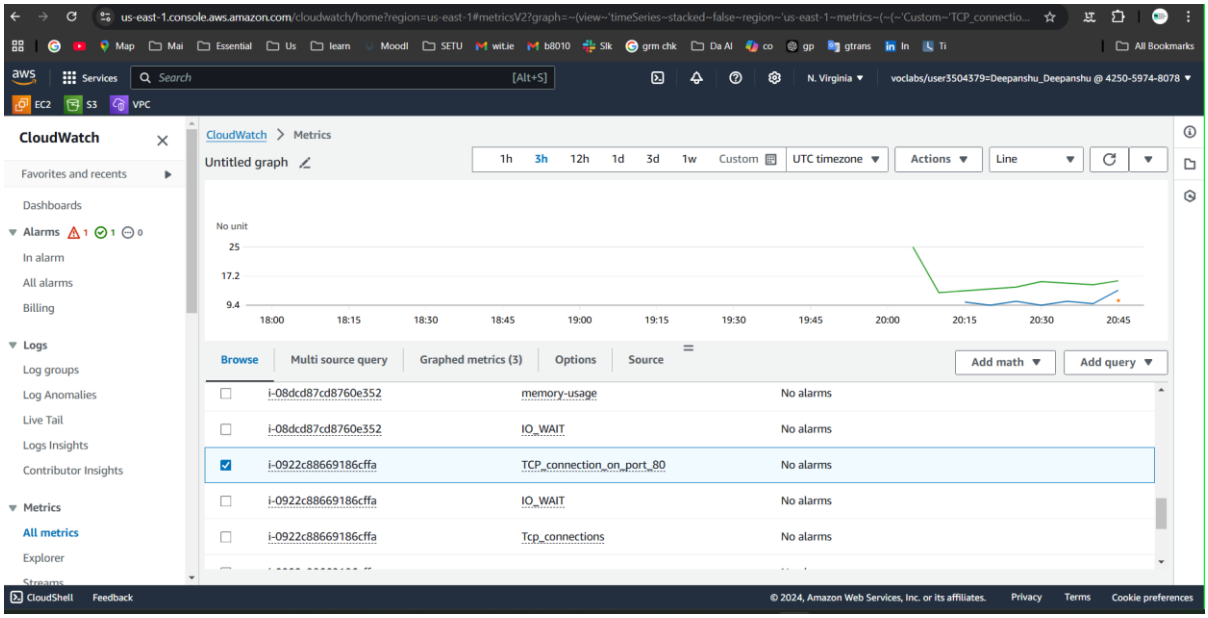
```
--metric-name memory-usage
```



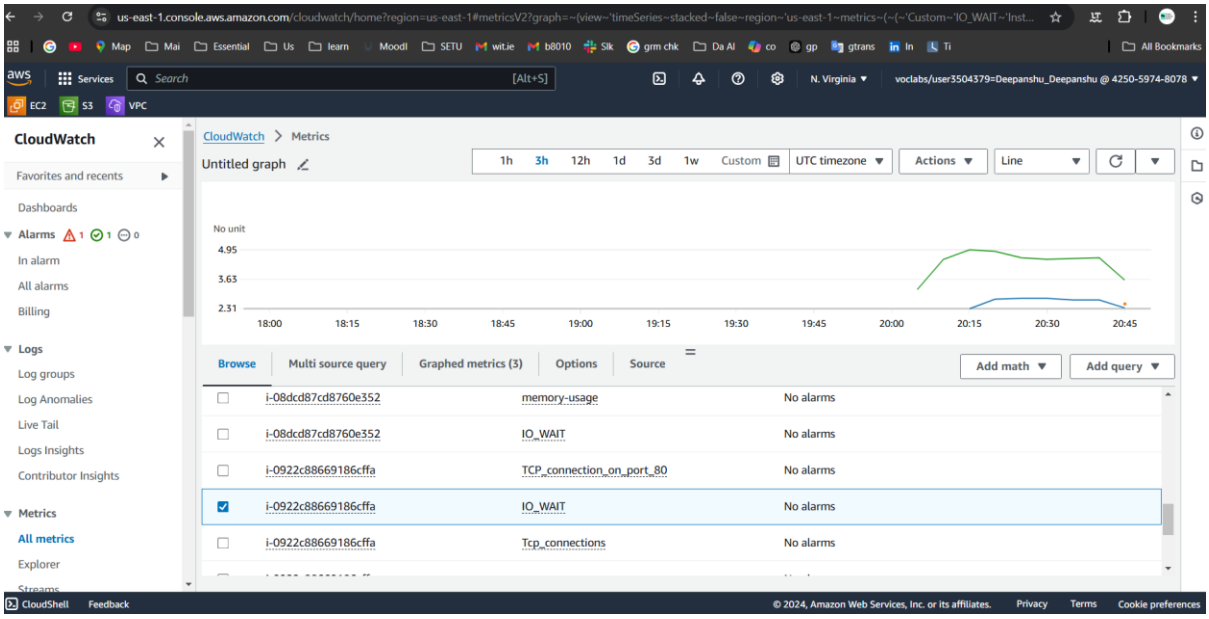
```
--metric-name Tcp_connections
```




```
--metric-name TCP_connection_on_port_80
```

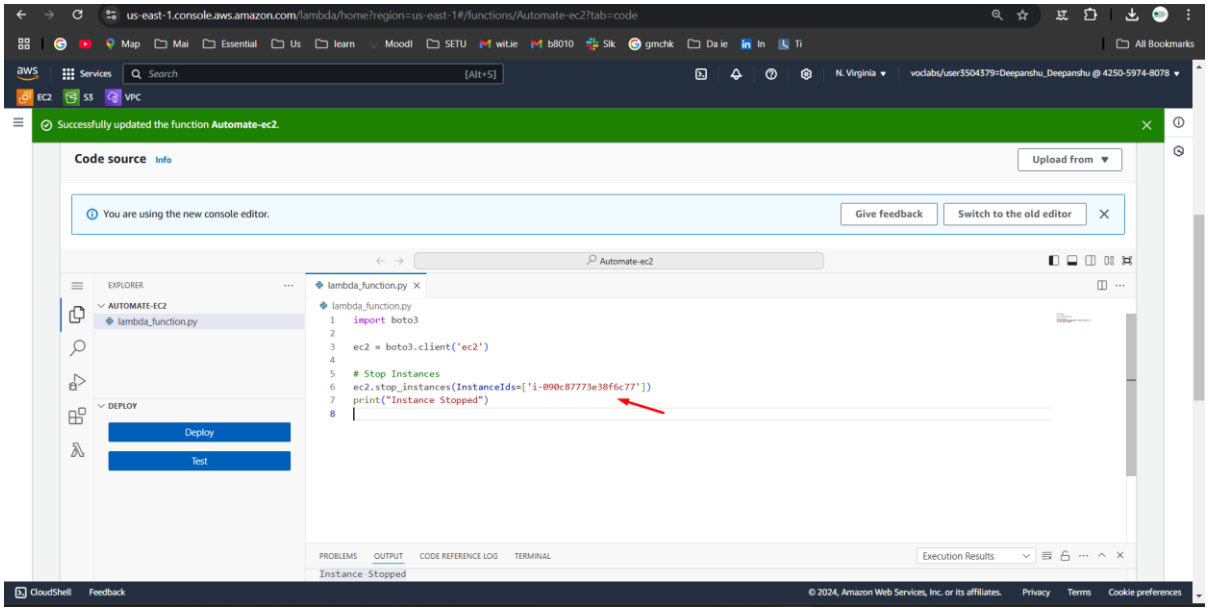


```
--metric-name IO_WAIT
```

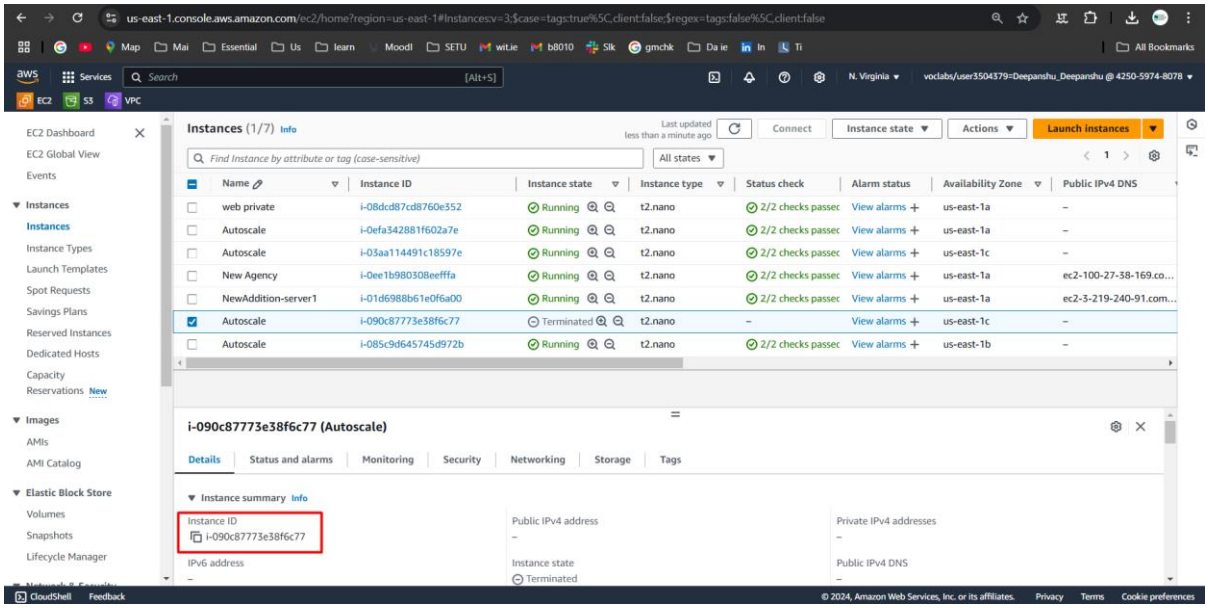


Additional functionality 1

AWS lambda: I integrated the AWS lambda function into my architecture because it provides a powerful way to run code in response to an event or request without having to manage servers. It works for event driven architectures and serverless applications.



After running test code on the lambda function, it's stopped instance

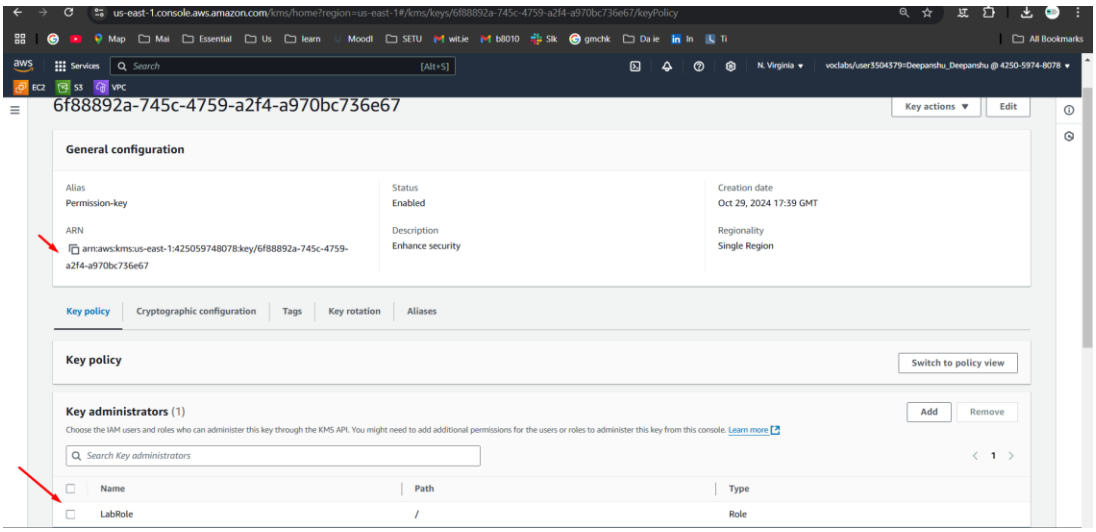


I used the aws lambda function to manage the EC2 instances because we interact daily with the EC2 instance in our console. If we have a large number of EC2 instances, stopping manually can be challenging. I found python boto3 documentation on the boto3.amazonaws.com website to stop any ec2 instance.

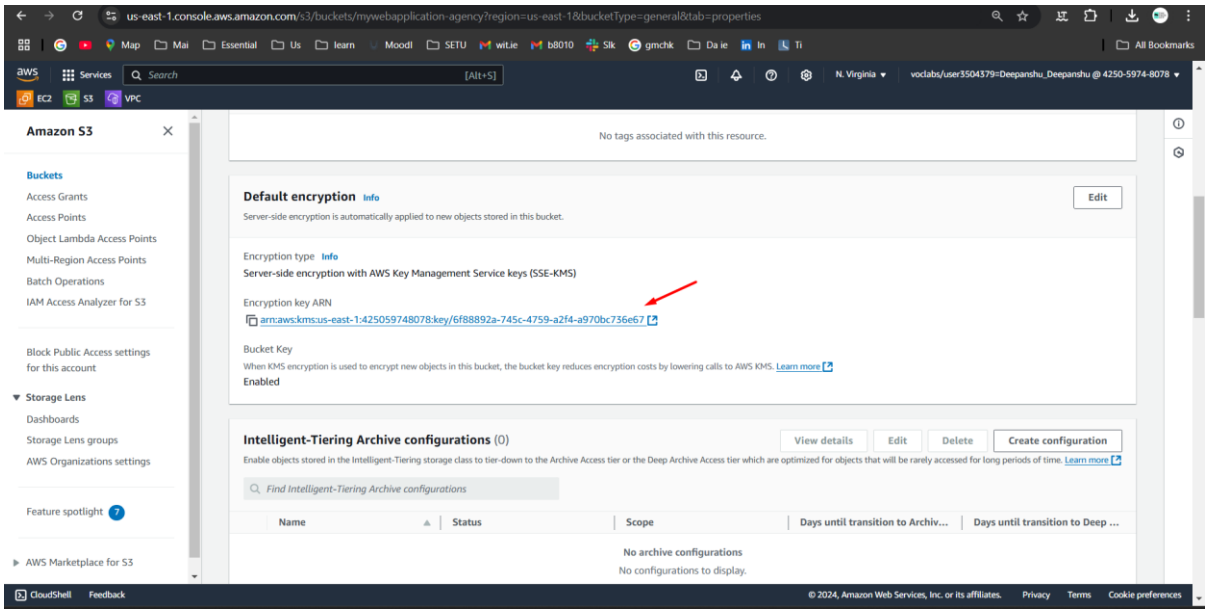
Additional functionality 2:

The **AWS Key Management Service** (KMS) is a security service that facilitates the creation and management of encryption keys to safeguard your data. We can provide a KMS service to encrypt data in S3 buckets, databases, or while transferring between Amazon Web Services.

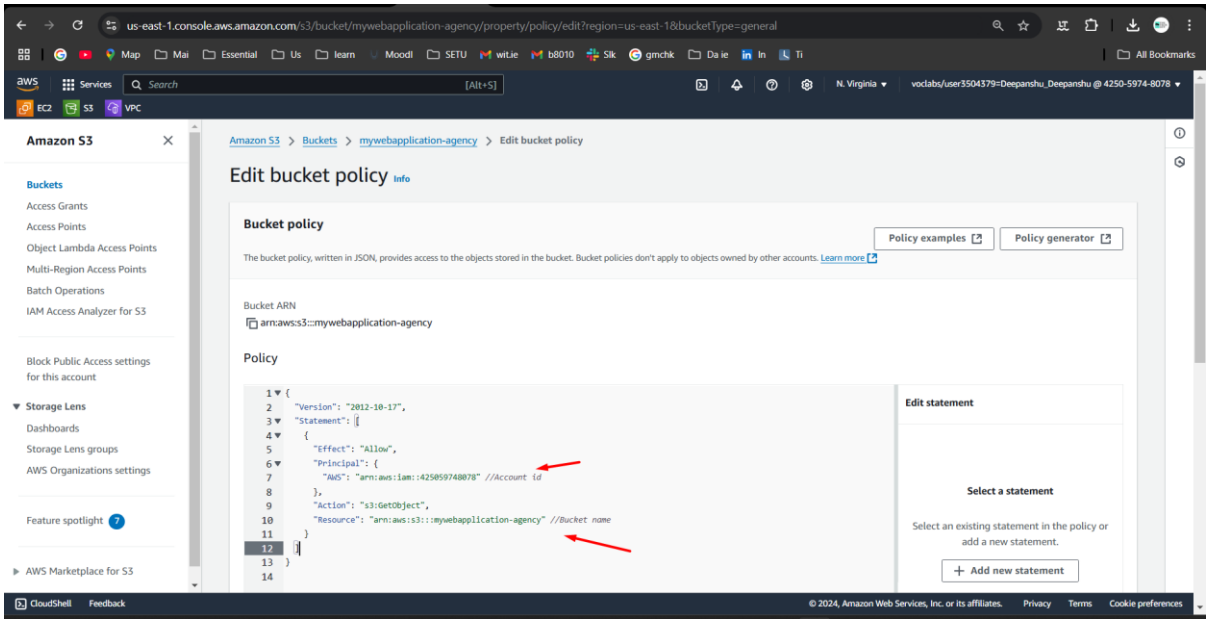
Create AWS KMS encryption key:



By default s3 bucket basic key comes so we have to edit and update KMS key inside the bucket permission tab.



Define bucket policy who can access this file which account .



I use the AWS Key Management service to secure my S3 bucket when I want to share a file with an account. I want to know who got this file and how they access it. The owner can manage key policies, including who can use the encryption keys and for what purpose. When we compare S3 keys, they do not provide this functionality.

Discussion/Conclusion:

I have created a static HTML website and deployed it on a private server. Therefore, migrating to a private server proved to be challenging. Initially, I uploaded the website to the public instance, then transferred the application to the private EC2 instance on /var/www/html, and finally, I created a custom AMI to utilize an auto scaling group. In this deployment, I created a VPC architecture with public and private subnets, since private application server resource can be accessible from publicly.

I set up load balancer and scaling policies triggered by CloudWatch alarms, because they help us scale our application and safeguard against failover. It ensures that the architecture is capable of automatically scaling up or down based on demand. I also tested load balancing to confirm that traffic was evenly distributed across multiple auto-scale instances that were created in the EC2 instance console.

Security Consideration:

In this Assignment we deployed this application into private subnet to limit direct internet access.

We have first deployed the application on public, and then we have to move the application into private.

Security group: We have created two security groups, WebServerSG for public traffic, and DBServerSG for private traffic. In private security group inbound permissions are less than public security groups.

Custom Watch : The Mem.sh file which contains customized metric parameters, was uploaded on a private subnet and an AMI was created to monitor memory usage, TCP port 80 utilization, CPU utilization, and other related metrics within the cloud metric column within the cloud watch.

Encryption: I use AWS Key Management service to secure sensitive data when we share it, so we can control which account they can use that file from and we have log's of that file in cloud trails.

AWS lambda: We do manually stop or terminate any instance, so I use aws lambda and I created a function we have just run this function and our instance ID is stopped. That instance ID is put in the function.

References:

AWS lambda: Stop instance
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ec2/client/stop_instances.html

AWS Key Management services
https://docs.aws.amazon.com/kms/latest/developerguide/key-policy-overview.html

AWS S3 bucket policies

https://docs.aws.amazon.com/AmazonS3/latest/userguide/example-bucket-policies.html