Scalable 3-tier AWS infrastructure with terraform

Terraform project

Made by:

Eslam Farag Mohamed El Sharkawy

Project idea

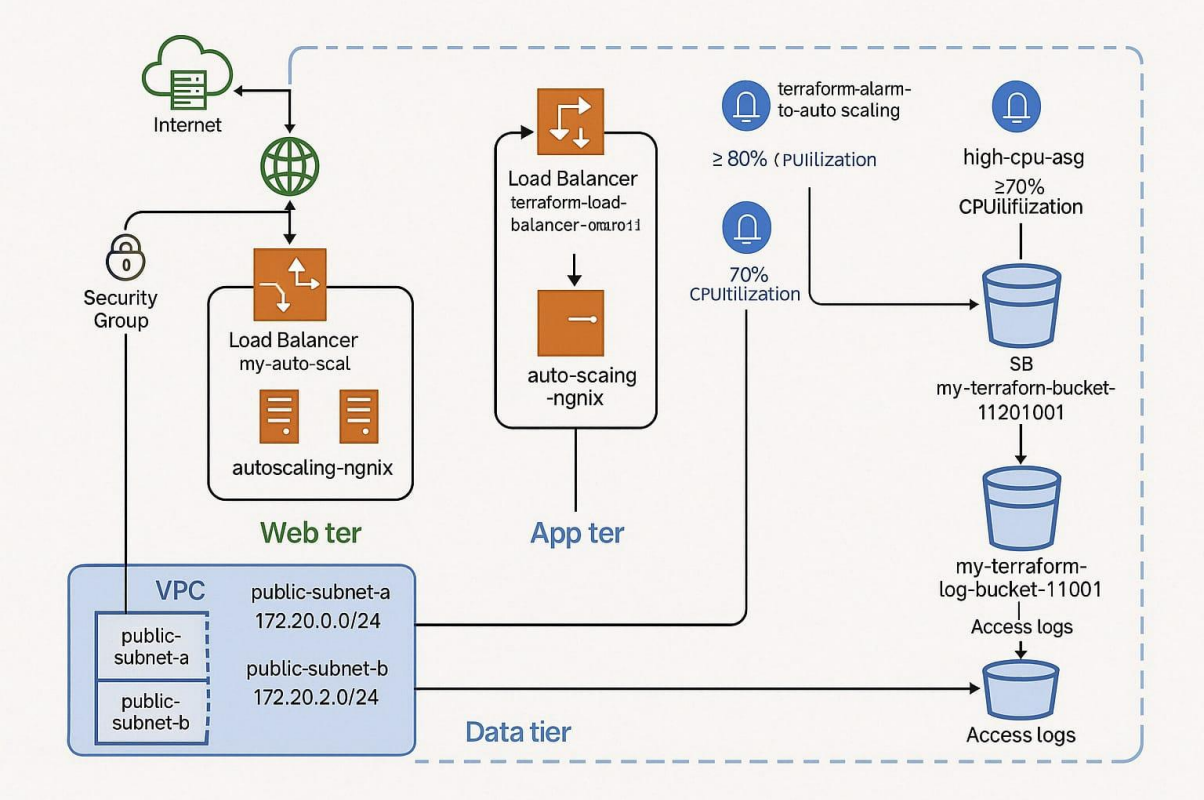
This project aims to build a flexible and scalable infrastructure on the AWS platform using Terraform. The architecture is based on the Three-Tier Architecture model, which is divided into three main tiers:

Web Tier: Contains an Application Load Balancer and Auto Scaling Group to serve requests from the internet via EC2 instances acting as web servers.

App Tier: Hosts the application logic and uses a separate Auto Scaling Group to ensure performance and efficiency in processing requests.

Data Tier: Contains data storage and logging using Amazon S3 to collect and analyze performance and load logs.

The project uses Terraform to automate all components of the architecture, such as VPCs, subnets, load balancers, Auto Scaling, Security Groups, CloudWatch Alarms, and SNS Notifications, making the process of creating, updating, and managing them easier and more efficient.



component

1. VPC Networking

VPC

Public Subnets

Internet Gateway

Route Table

2. Compute

EC2 Instances

Instance in Public subnet

Launch Template

User Data script (Nginx website)

Auto Scaling Group

Security Group

3. S3 Bucket

Bucket to store Logs

4. ALB (Application Load Balancer)

Target group

Listener

5. CloudWatch

Alarms (CPU, Memory)

Logging from EC2

Project structure

project/

│

├── auto_scaling.tf

├── variables.tf

├── cloudwatch.tf

├── dynamoDB.tf

├── load_balancer.tf

├── provider.tf

├── s3.tf

├── security_group.tf

├── SNS.tf

├── vpc.tf

Project stages

1. VPC (Virtual Private Cloud)

The Internet Gateway is connected to the VPC.

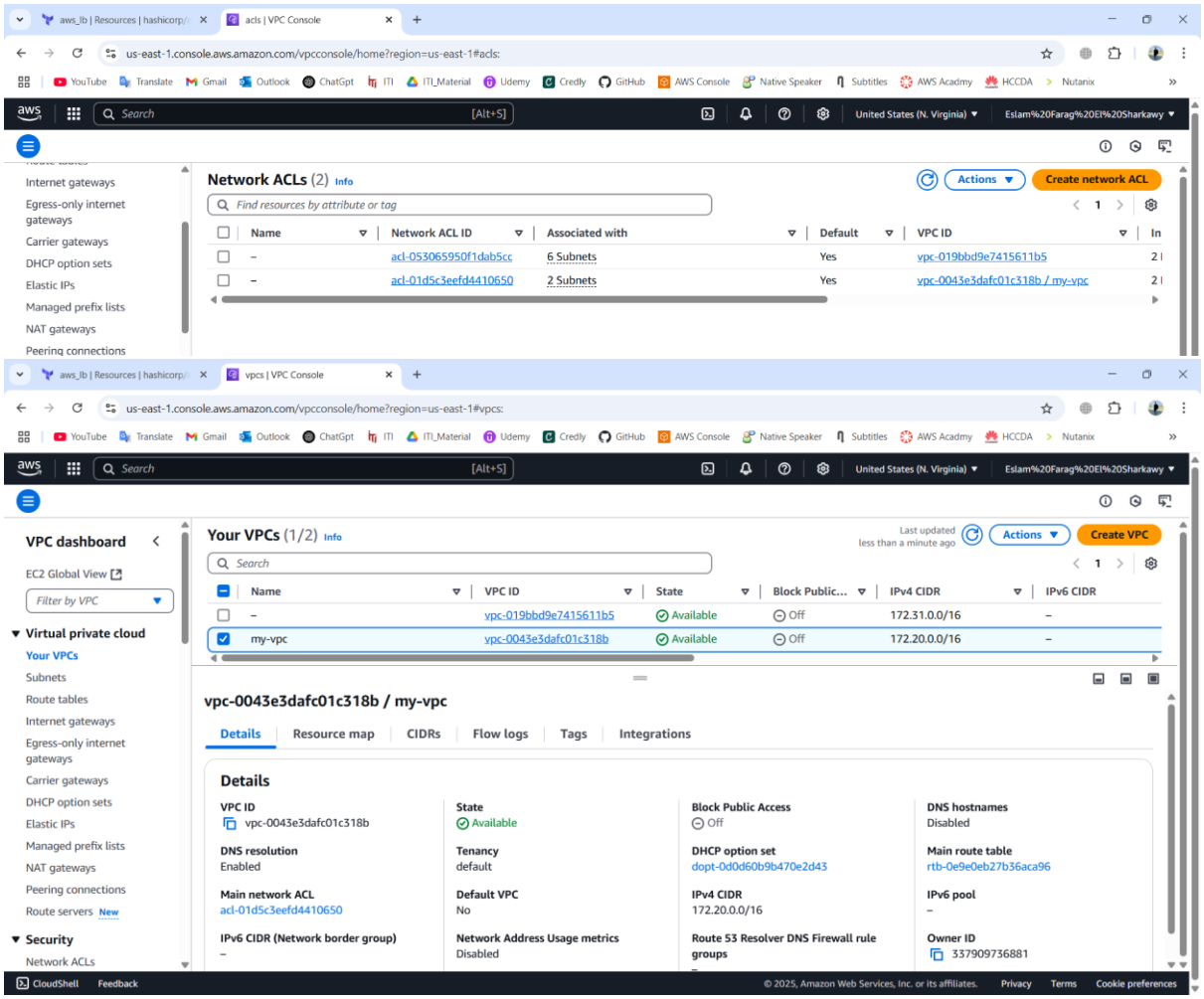
The Route Table contains a route that directs all traffic (0.0.0.0/0) to the Internet Gateway. Associate this Route Table with subnets so they can reach the internet.

Confirm that the Route Table is named terraform-route-table and is indeed associated with two subnets, as shown in the code. It displays the VPC named my-vpc, with its status "Available." This is the same VPC to which all resources in the project are connected.

It shows that the subnets (public-subnet-a and public-subnet-b) are present in the VPC, and each one is associated with a Route Table so it can reach the internet.

It shows that there is an Internet Gateway named terraform-internet-gateway connected to your project's VPC. This allows instances within the VPC to access and enter the internet.

There are two Network ACLs, each connected to different subnets within the VPCs. This demonstrates that you are structuring your network to control incoming and outgoing traffic, which helps with security and segmentation.



2. provider

The file identifies the provider we'll be working on.

We specify the AWS provider and specify that the version be greater than or equal to 5.0.

We specify the region we'll be working in, which is "us-east-1" (meaning Northern Virginia).

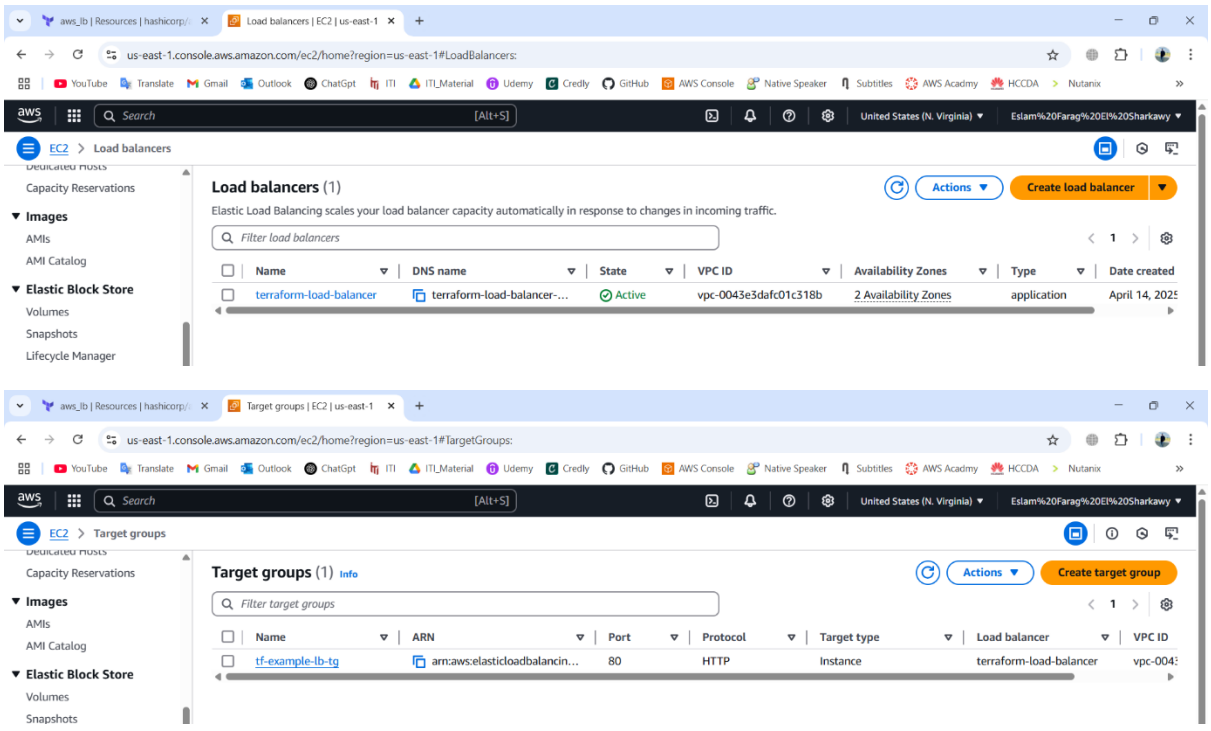
3. Application Load Balancer (ALB)

There is a load balancer for distributing traffic between EC2 instances generated by autoscaling. It is connected to a Target Group running on port 80, using the HTTP protocol, and is connected to a VPC. It also sends S3 logs containing two ACLs (network control rules), one of which is connected to our VPC. This controls who can enter and exit the network at the subnet level.

1. The Target Group connects the Load Balancer to the instances.

2. The Load Balancer is an Application type and is connected to subnets.

3. The Listener on port 80 forwards traffic to the Target Group.



4. S3 Bucket

The Terraform code shows how to create an IAM policy on an S3 bucket specifically for logging:

The first policy allows the Load Balancer to write logs (s3:PutObject) to the bucket.

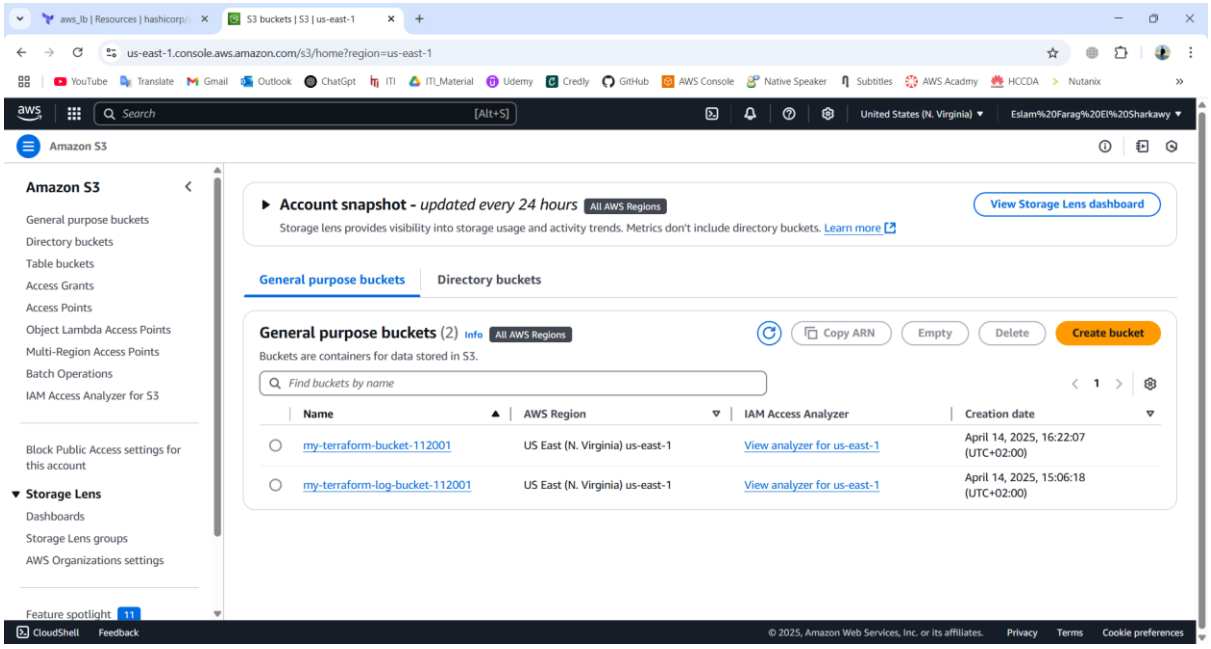
The second allows it to know the bucket location (s3:GetBucketLocation).

This policy is linked to the log bucket using aws_s3_bucket_policy.

Two buckets are created: one for primary (my_s3) and one for logs (my_log_bucket).

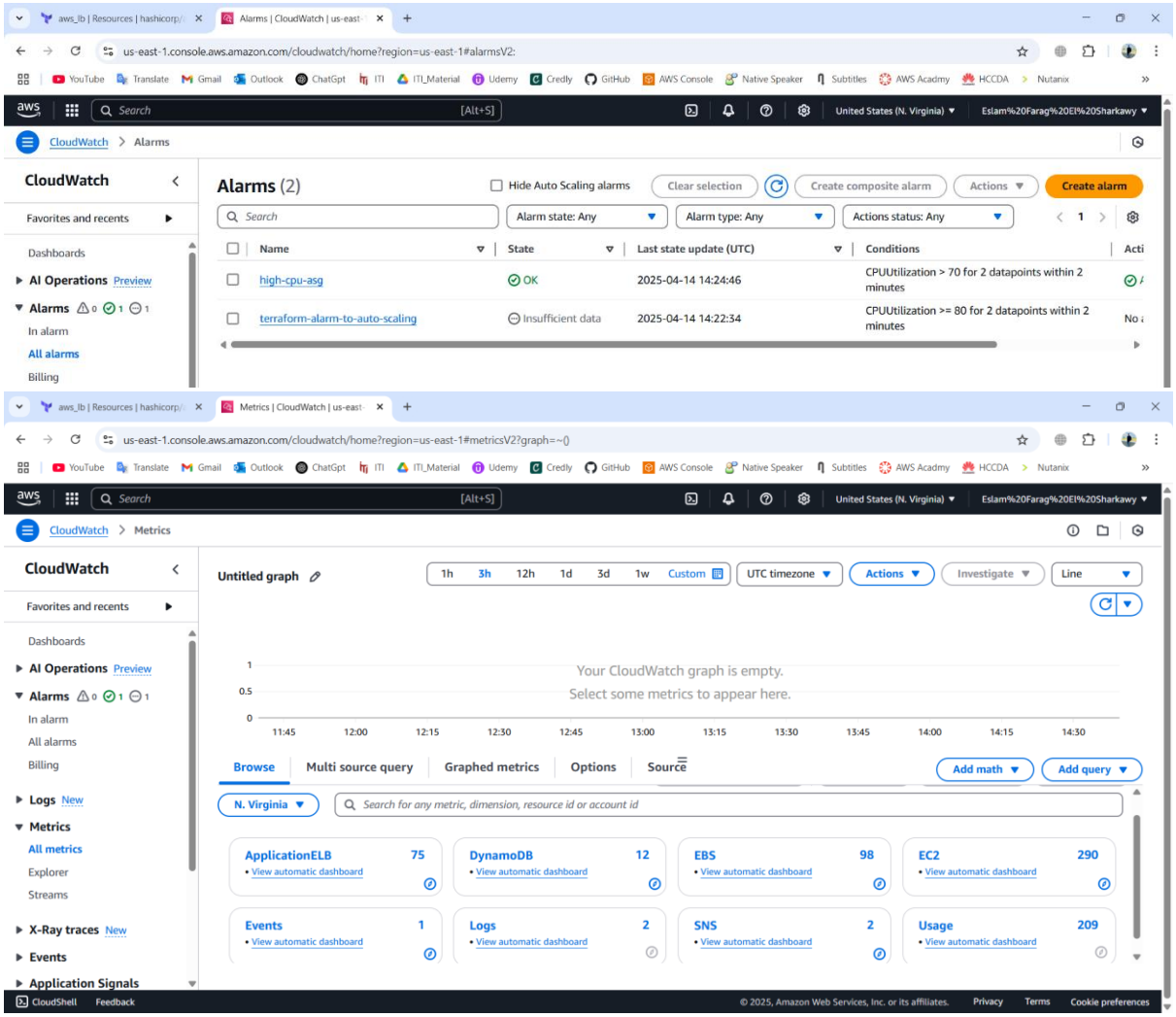
aws_s3_bucket_logging links the primary bucket to the log bucket to log to it.

A second policy is then created that allows an AWS canonical user to log access logs to the log bucket.



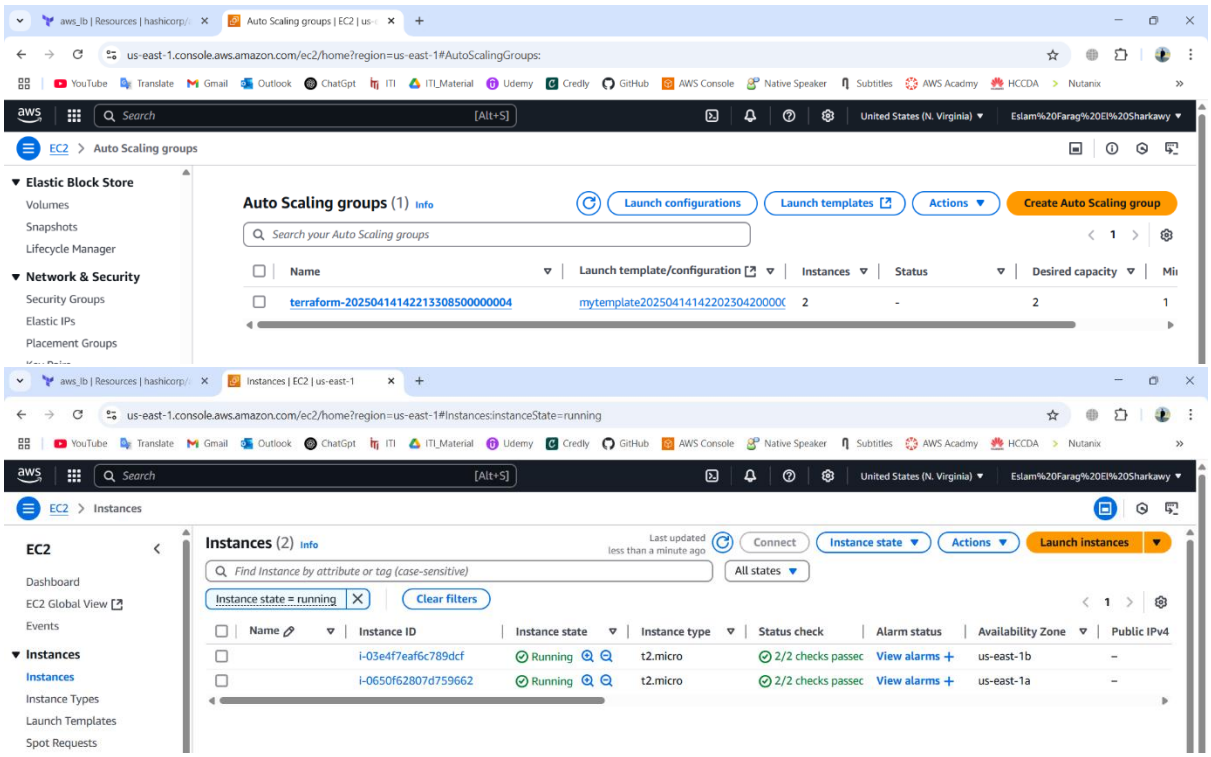
5. CloudWatch Monitoring

CloudWatch code monitors EC2 CPU usage and if it exceeds 80%, it sends an Auto Scaling warning to create a new EC2, and sends an alert via SNS.



6. Auto Scaling

This is the Terraform code, it creates a Launch Template that runs an EC2 instance of t2.micro with an Amazon Linux image and an Nginx website, writes a welcome message, and then creates an Auto Scaling Group that runs 2 servers on different subnets, and is ready to grow to 6 servers.

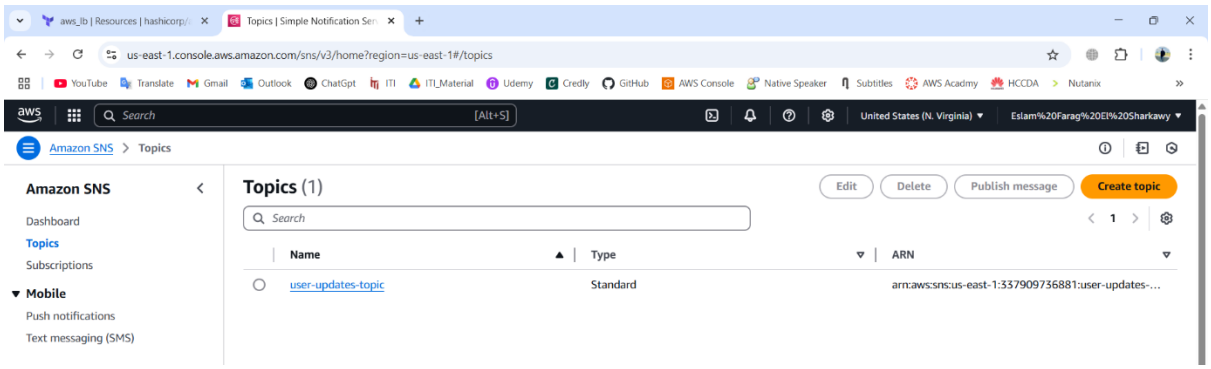


7. SNS

The code creates a new topic called user-updates-topic using aws_sns_topic.

It subscribes to this topic as "Standard," allowing notifications to be tracked quickly and flexibly. It uses the email protocol to send notifications to eslamfarag577@gmail.com.

Whenever something related to the SNS Alarm occurs (such as high CPU usage), a message will be sent to the email address you specified.



8. Security Group

I created a firewall that allows incoming traffic from the internet on the primary ports (80 and 443) and outgoing traffic anywhere. This is suitable for servers that host traffic like Nginx or Load Balancers.

It shows that there are three Security Groups in the account, one of which is named allow_tls. This is the one you created manually using Terraform, and it is linked to the VPC your project is running on.

The code defines a Security Group named allow_tls.

It allows incoming traffic on:

Port 80 (HTTP) for IPv4.

Port 443 (HTTPS) for IPv6.

It allows outgoing traffic to any destination (0.0.0.0/0).