3-tier Infrastructure with Terraform and Ansible

Overview

This project involves creating a scalable and highly available infrastructure on AWS using Terraform for infrastructure provisioning and Ansible for configuration management. The setup includes a VPC with subnets, NAT gateway, internet gateway, EC2 instances, security groups, a load balancer, RDS instance, CloudWatch for monitoring, SNS for notifications, and Route 53 for DNS management. The EC2 instances are configured using Ansible to host a web application with a backend and frontend server setup.

Prerequisites

Before proceeding with the infrastructure setup and configuration, ensure the following prerequisites are met on your local Ubuntu machine:

- **Create SSH Key Pair:**
    - Generate an SSH key pair and import the public key into your AWS account.

      ```
      #ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
      ```

- **AWS CLI Installation:**
    - Install AWS CLI.
    - Configure AWS CLI with your access key and secret key.
    - Install Terraform.
    - Install Ansible.

Tools and Technologies

- **Infrastructure Provisioning**: Terraform
- **Configuration Management**: Ansible, shell scripting
- **Secrets Management**: Terraform Vault, Ansible Vault
- **Cloud Provider**: AWS
- **Services Used**: VPC, EC2, RDS, S3, CloudWatch, SNS, Route 53, Load Balancer

Three-Tier Architecture

- **Presentation Tier**: Managed by the frontend server (NGINX) which handles user interaction.
- **Application Tier**: Managed by the backend server (Apache) which handles business logic.
- **Data Tier**: Managed by the RDS instance which handles data storage and management.

Infrastructure Setup with Terraform

1. **VPC Creation**
    - Create a new VPC with CIDR 192.168.0.0/20.
    - Create public and private subnets in different availability zones.
    - Enable auto-assign public IP on the public subnet.
    - Create and attach an internet gateway to the VPC.
    - Create a route table, add a route to the internet gateway, and associate it with the public subnet

- Create a NAT gateway in the public subnet and update the route table for the private subnet to route traffic through the NAT gateway.

2. **EC2**
    - Launch an EC2 instances in the public and private subnet.
    - Create security groups for the instances with appropriate inbound rules (SSH, HTTP, MYSQL/Aurora, ALL ICMP, Custom TCP on port 8080).
    - Create an Application Load Balancer and configure it to balance traffic to the private instance.
    - Create a launch template and auto-scaling group to manage the EC2 instances in the private subnet.

1. **RDS:**
    - Create parameter groups and subnet groups for RDS.
    - Launch an RDS instance using the created groups
2. **CloudWatch and SNS:**
    - Set up CloudWatch for monitoring and SNS for notifications.
3. **Route 53:**
    - Create a Route 53 hosted zone.
    - Add a new A record with a simple routing policy.
    - Create a health check for the DNS.

Configuration Management

Shell script :

- I wrote a shell script to create an inventory file with the IP addresses of the running instances.
- I saved the terraform output in Json format and then I used that output from variable in Ansible playbook.

Ansible :

1. **Private and Public Instances Configuration:**
   - Use Ansible to install Apache on the backend server (private instance).
   - Store the WAR file of the application in the webapps folder and the sql-connector.jar file in the lib folder.
   - Configure context.xml
2. **Frontend Configuration:**
   - Install Nginx on the frontend server (public instance).
   - Configure Nginx to proxy pass requests to the backend server load balancer DNS.

3. **Database Configuration:**
   - Ansible Playbook to Create Database
   - Use the RDS endpoint stored in the JSON file to create a database

4. **Securing Credentials:**
   - Use Terraform Vault and Ansible Vault to secure database passwords.

Terraform and ansible Script:

https://github.com/Sameerpatlekar/project/tree/main/terraform_project/3-tier

Setup terraform vault on ec2 instance

Install Vault on the EC2 instance

To install Vault on the EC2 instance, you can use the following steps:

Install gpg

sudo apt update && sudo apt install gpg

Download the signing key to a new keyring

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

Verify the key's fingerprint

gpg --no-default-keyring --keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg --fingerprint

Add the HashiCorp repo

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update

Finally, Install Vault

sudo apt install vault

Configure Terraform to read the secret from Vault.

Detailed steps to enable and configure AppRole authentication in HashiCorp Vault:

1. **Enable AppRole Authentication**:

To enable the AppRole authentication method in Vault, you need to use the Vault CLI or the Vault HTTP API.

Using Vault CLI:

Run the following command to enable the AppRole authentication method:

vault auth enable approle

This command tells Vault to enable the AppRole authentication method.

2. **Create an AppRole**:

We need to create policy first,

```
vault policy write terraform - <<EOF
path "*" {
  capabilities = ["list", "read"]
}

path "secrets/data/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}

path "kv/data/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
```

EOF

Now you'll need to create an AppRole with appropriate policies and configure its authentication settings. Here are the steps to create an AppRole:

a. Create the AppRole:

```
vault write auth/approle/role/terraform \
   token_policies=terraform


 /*Optional
 secret_id_ttl=10m \
   token_num_uses=10 \
   token_ttl=20m \
   token_max_ttl=30m \
   secret_id_num_uses=40 \
*/
```

3. **Generate Role ID and Secret ID**:

After creating the AppRole, you need to generate a Role ID and Secret ID pair. The Role ID is a static identifier, while the Secret ID is a dynamic credential.

a. Generate Role ID:

You can retrieve the Role ID using the Vault CLI:

```
vault read auth/approle/role/my-approle/role-id
```

Save the Role ID for use in your Terraform configuration.

b. Generate Secret ID:

To generate a Secret ID, you can use the following command:

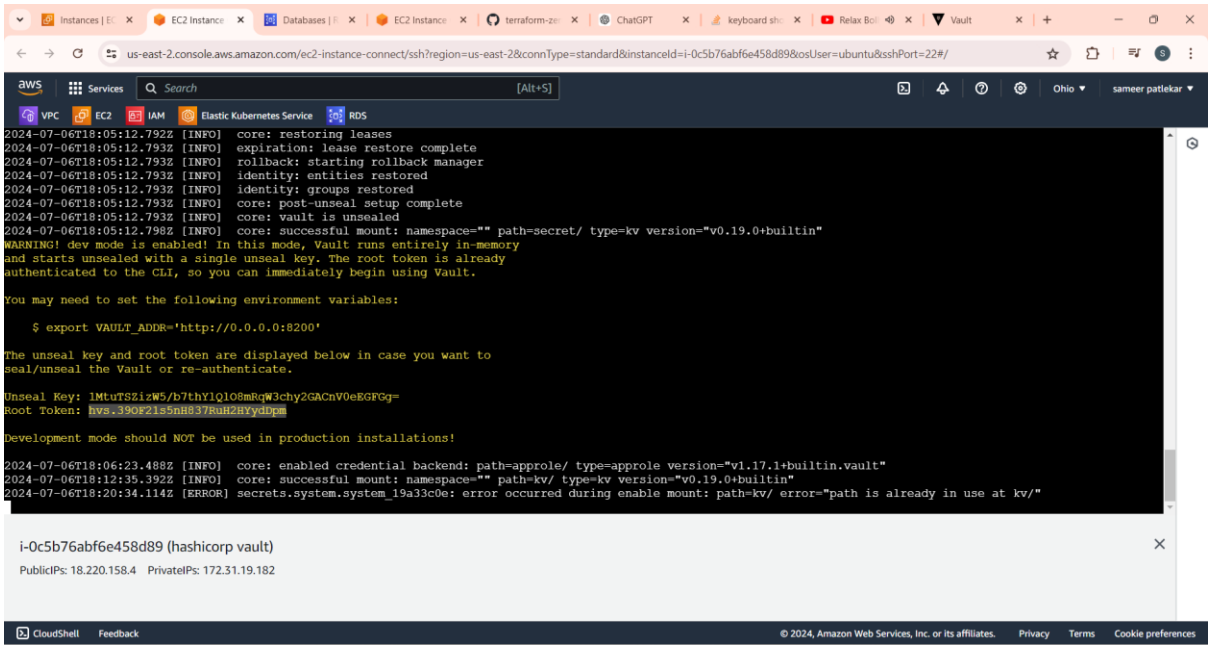
vault write -f auth/approle/role/my-approle/secret-id

Start Vault.

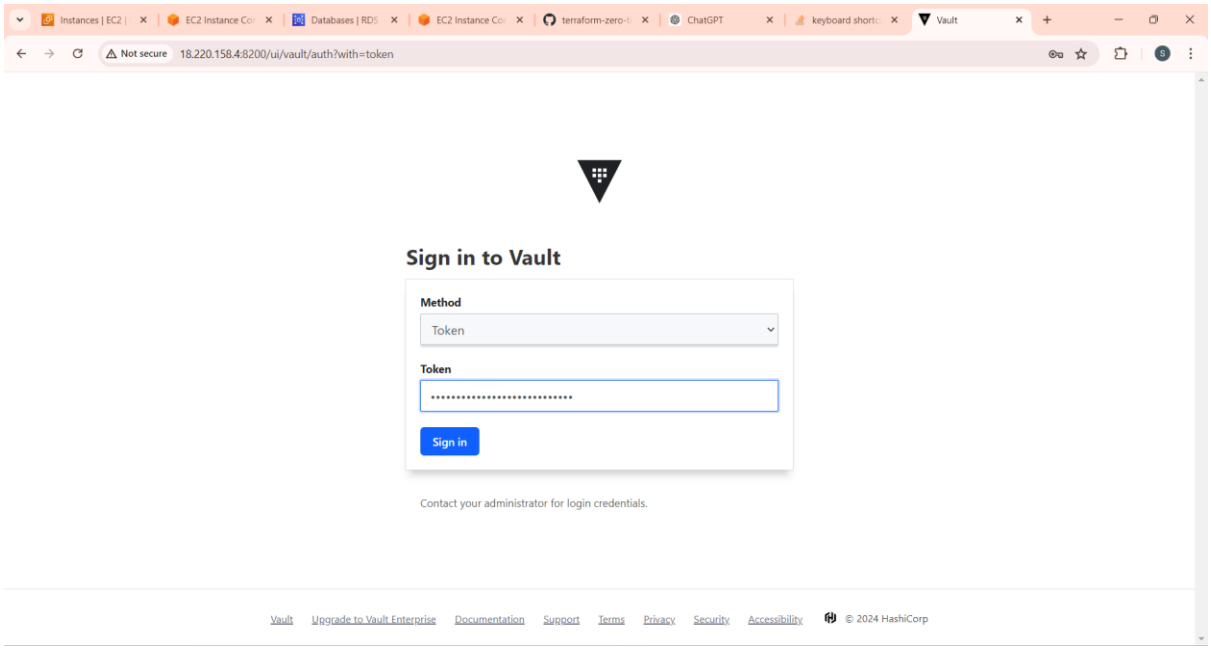
To start Vault, you can use the following command:

vault server -dev -dev-listen-address="0.0.0.0:8200"

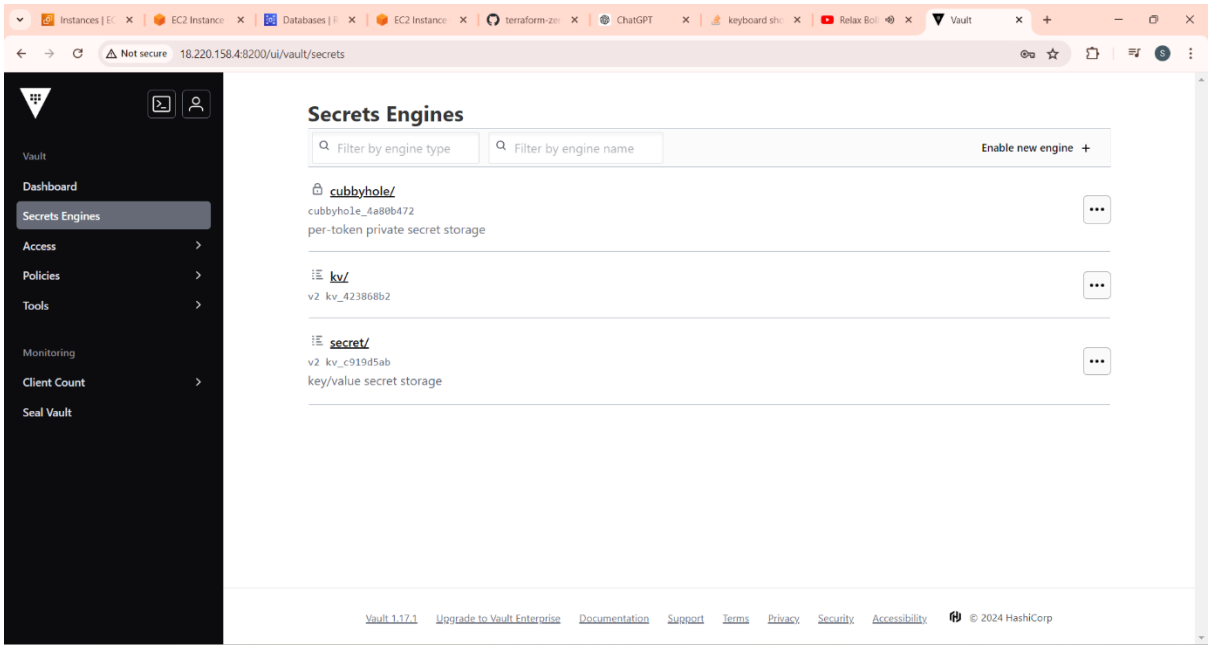
Copy the root token



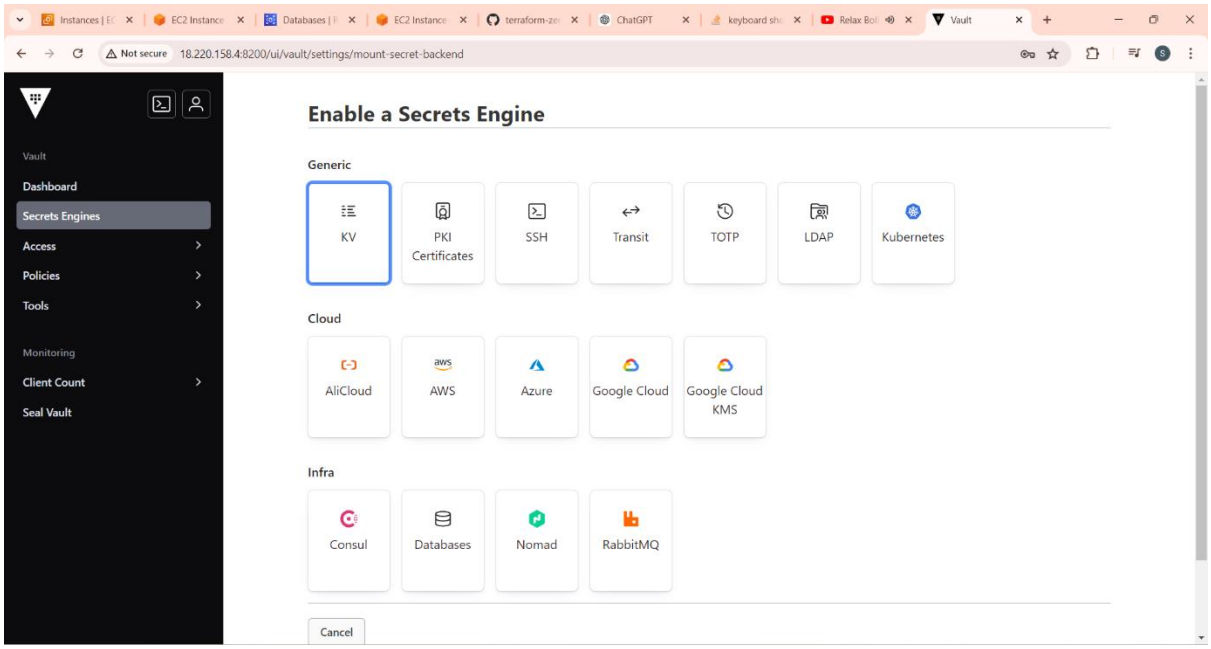
Open chrome enter ip address with port 8200 then paste token and then hit

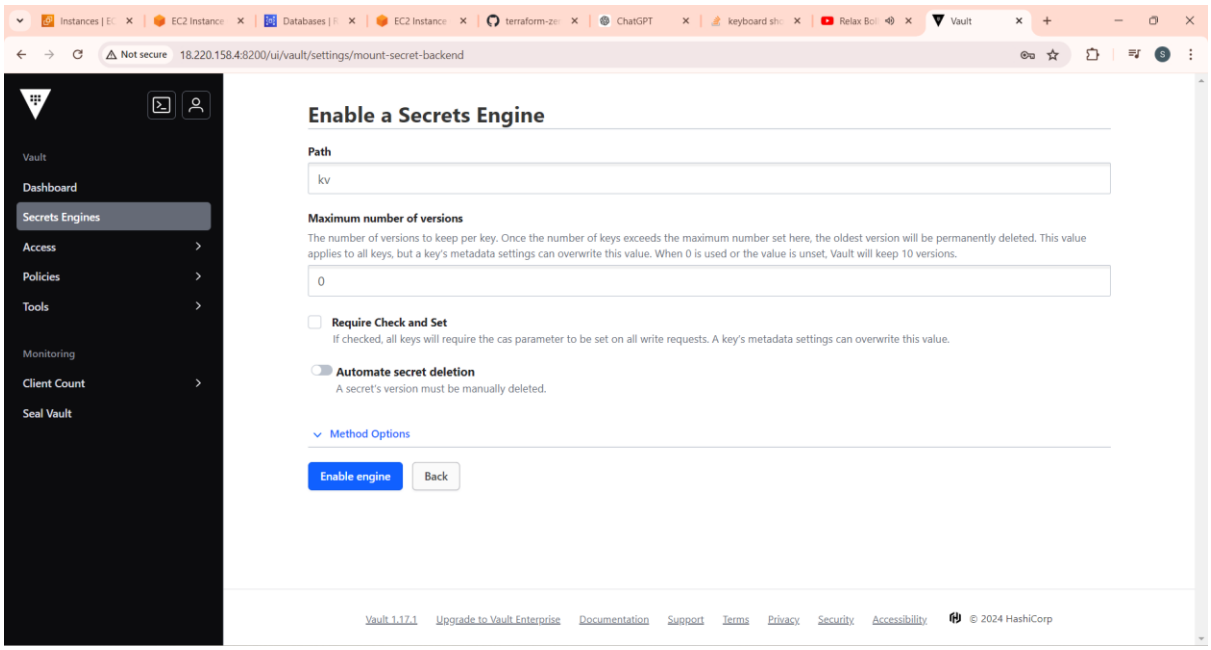


In the Dashboard of the terraform vault go to secrets engines and click on enable new engine



Select engine





Setup Ansible Vault

1. Create a password for vault

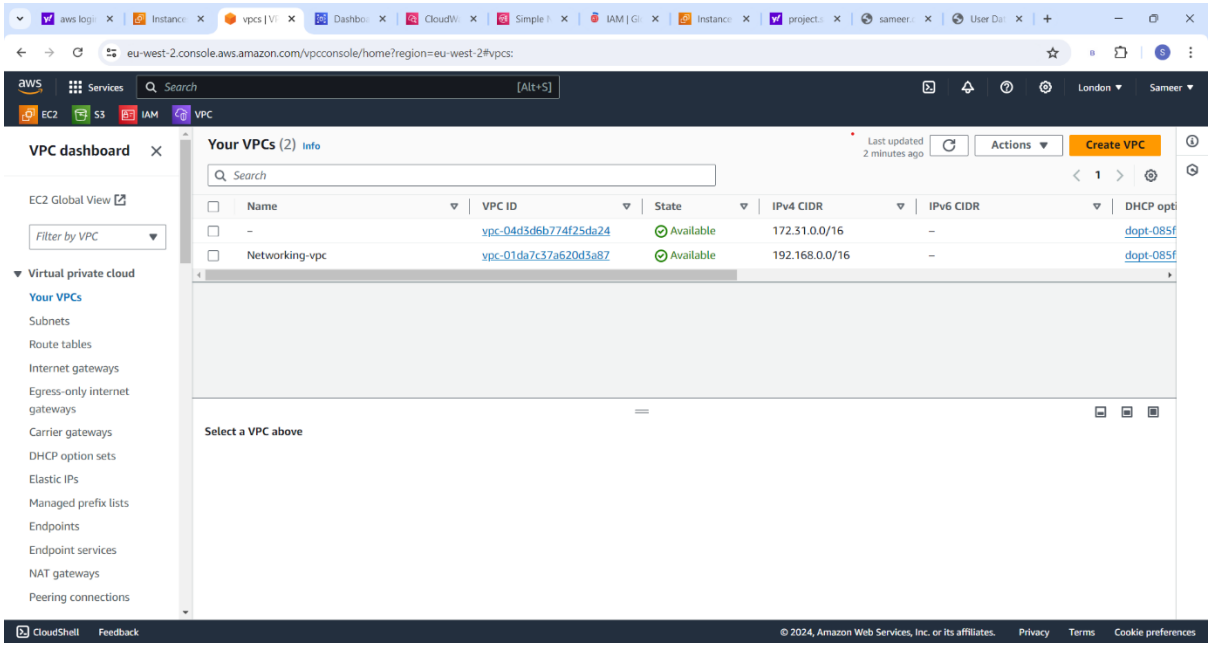
openssl rand -base64 2048 > vault.pass

2. Add your AWS credentials using the below vault command

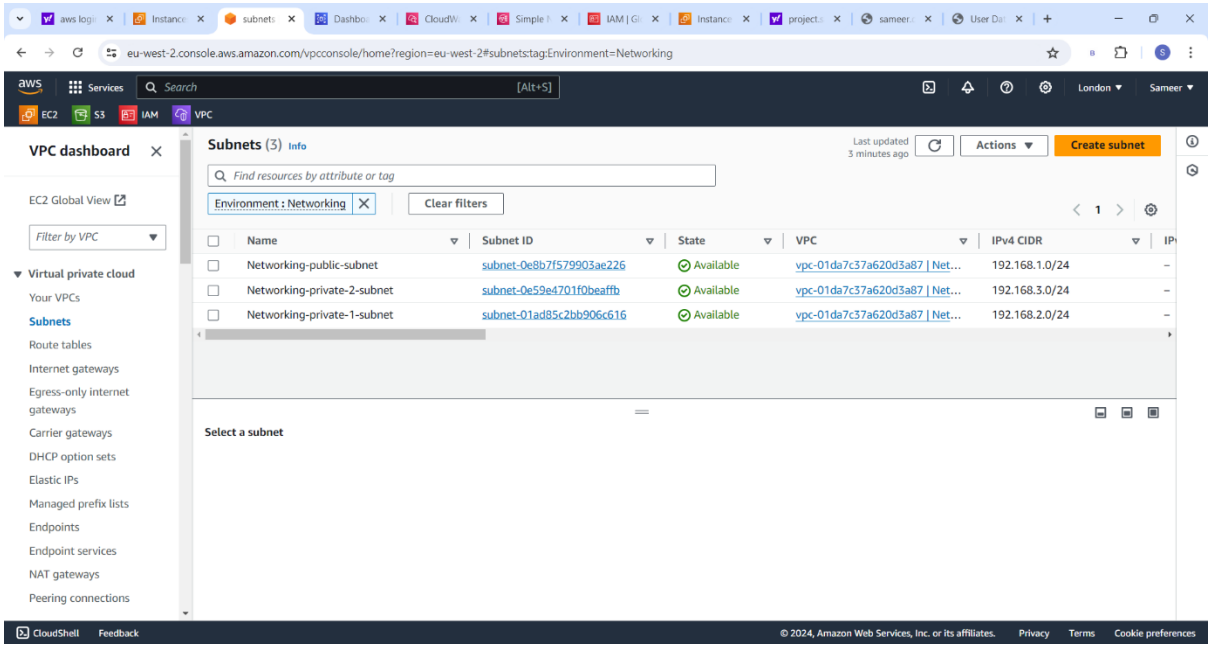
ansible-vault create group_vars/all/pass.yml --vault-password-file vault.pass

Infrastructure Setup with terraform

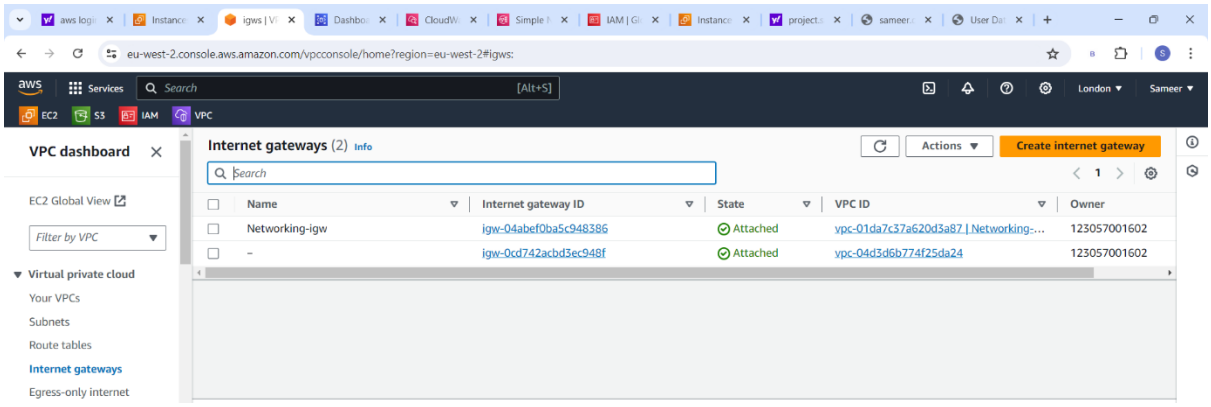
Create a new VPC with CIDR 192.168.0.0/20.



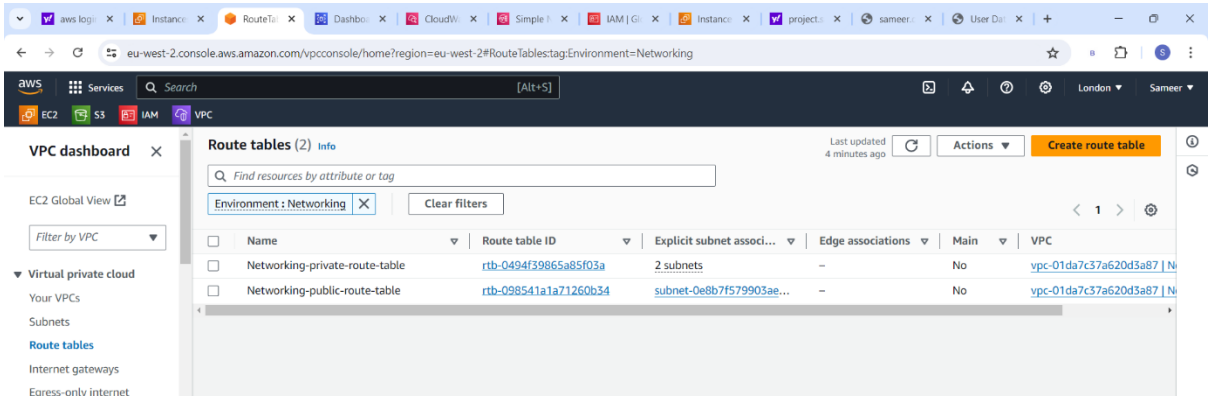
Create public and private subnets in different availability zones. Enable auto-assign public IP on the public subnet.



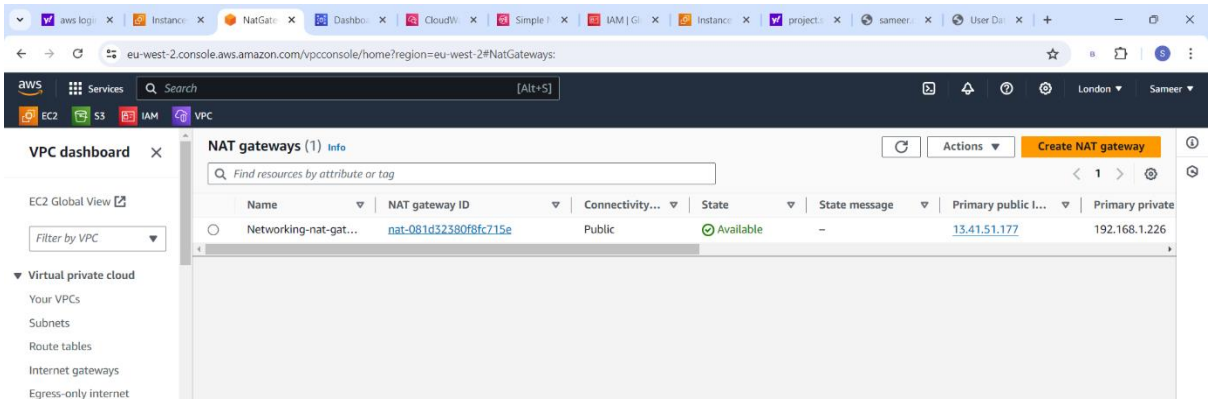
Create and attach an internet gateway to the VPC.



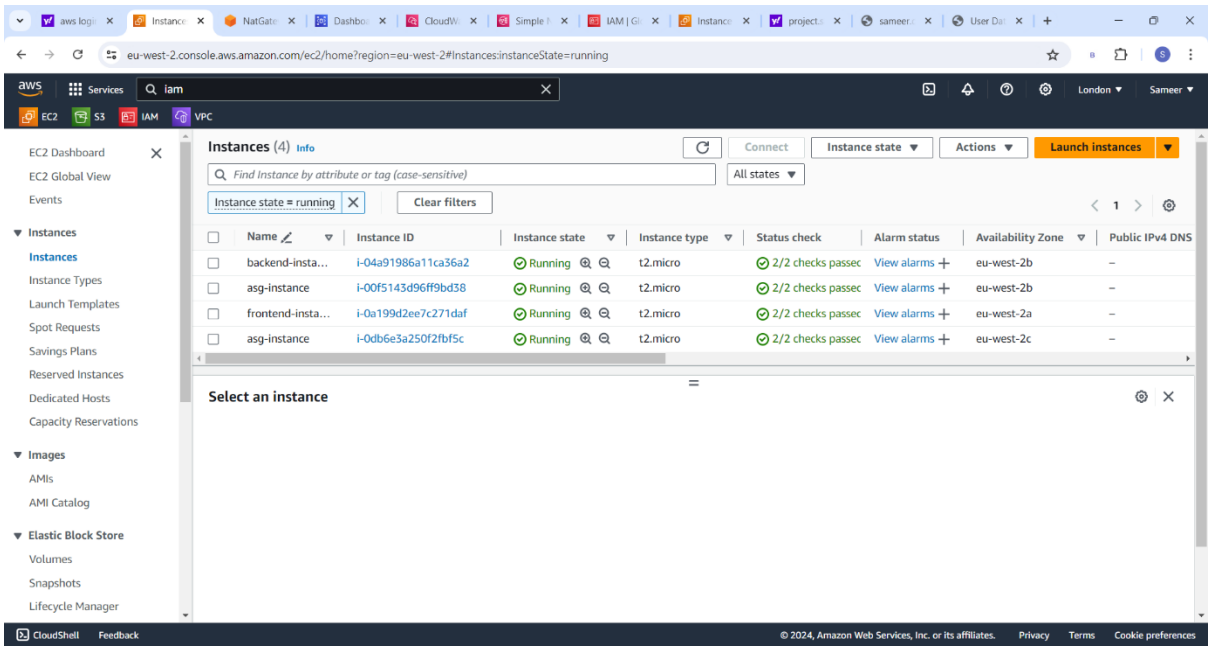
Create a route table, add a route to the internet gateway, and associate it with the public and private subnet



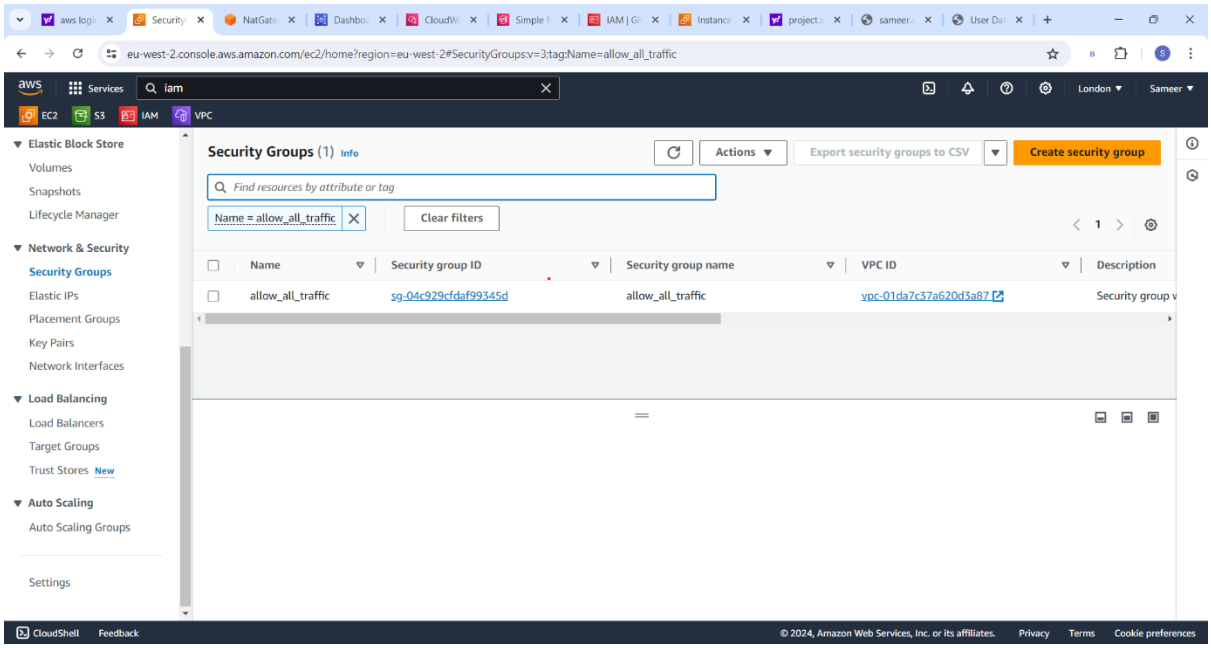
Create a NAT gateway in the public subnet and update the route table for the private subnet to route traffic through the NAT gateway.



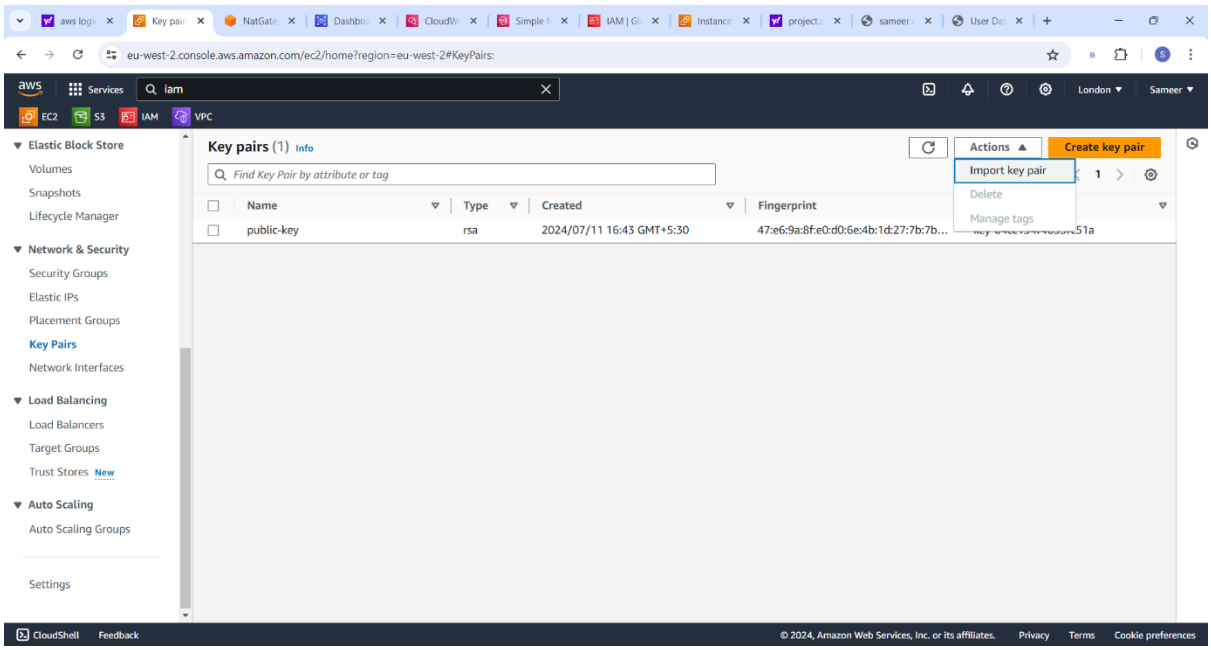
Launch an EC2 instances in the public and private subnet.



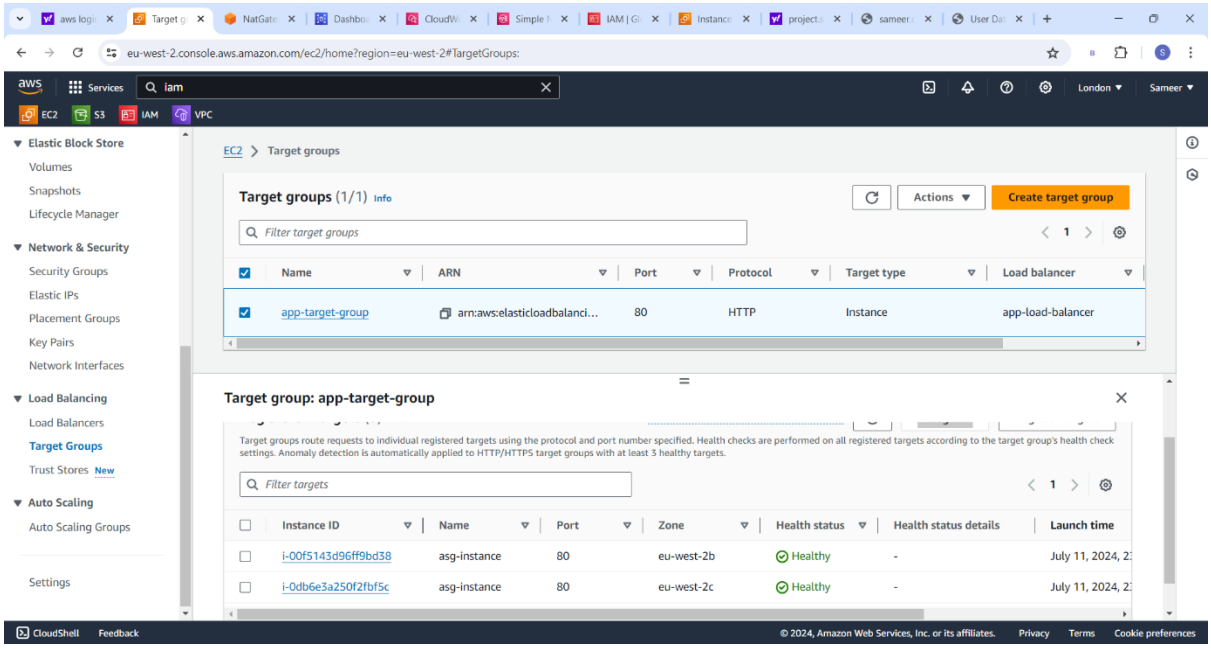
Create security groups for the instances with appropriate inbound rules (SSH, HTTP, MYSQL/Aurora, ALL ICMP, Custom TCP on port 8080).



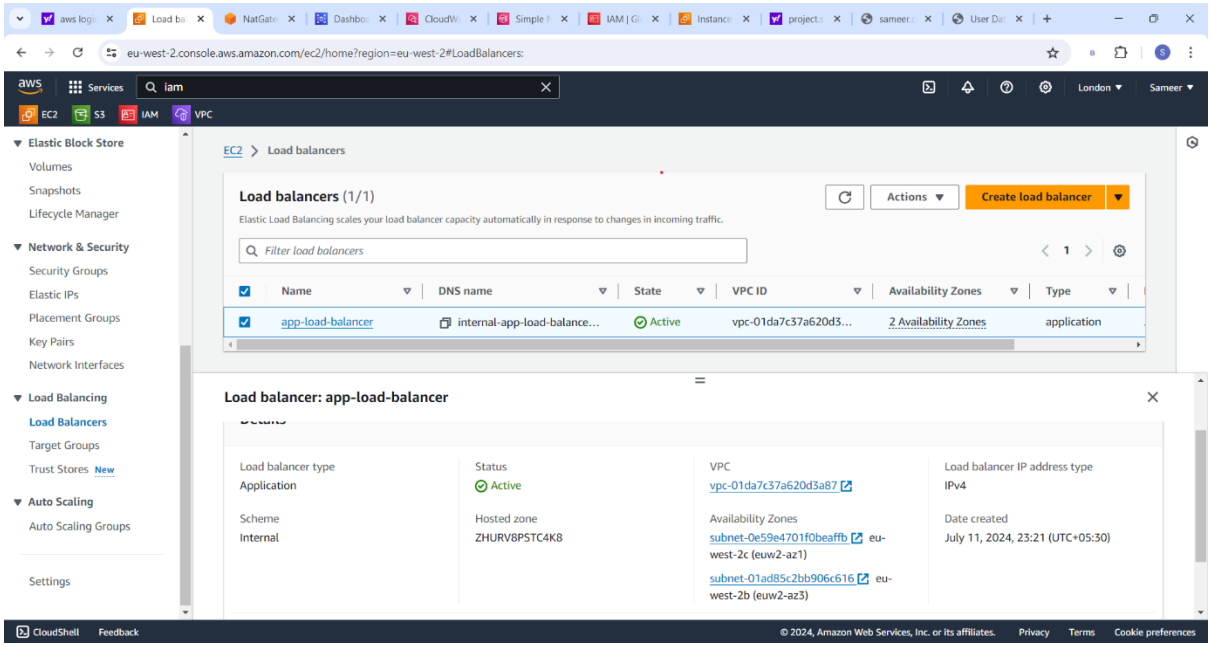
Import public key of my local machine



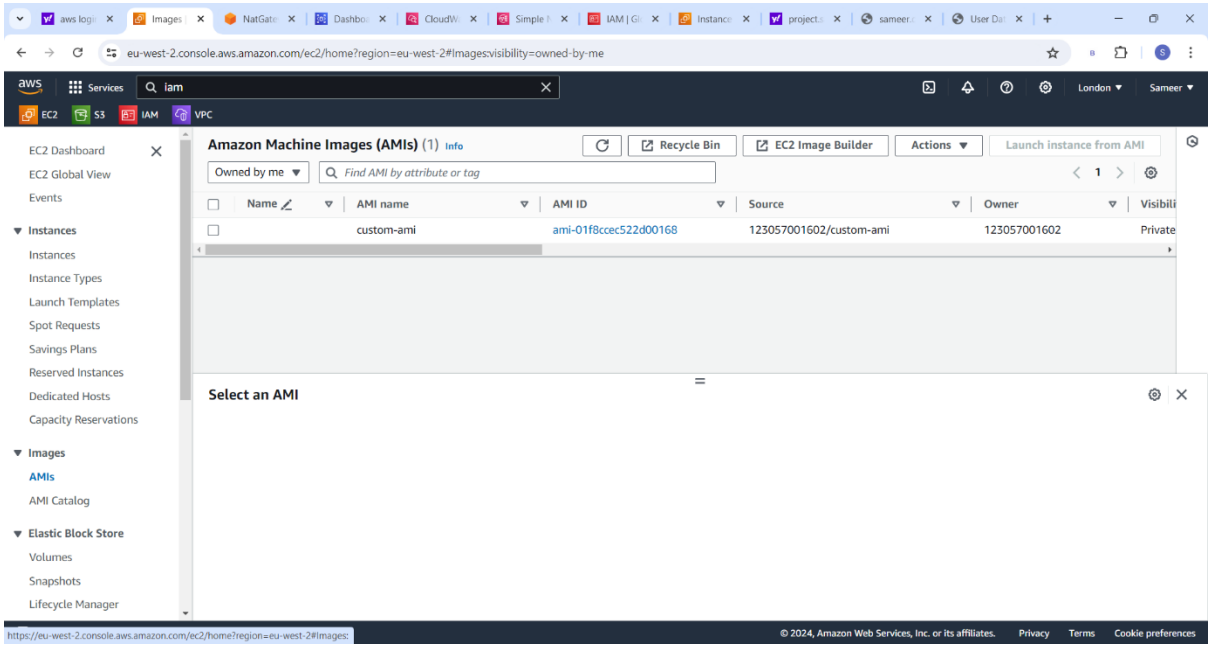
Create target group with private instance



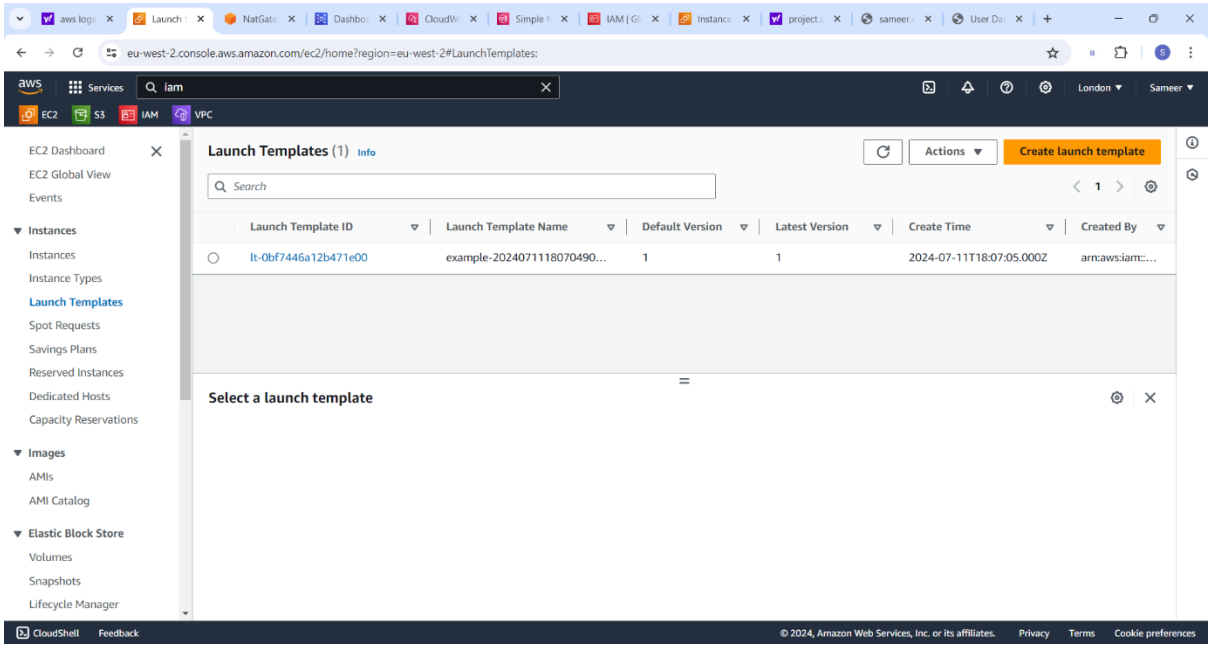
Create an Application Load Balancer and configure it to balance traffic to the private instance.



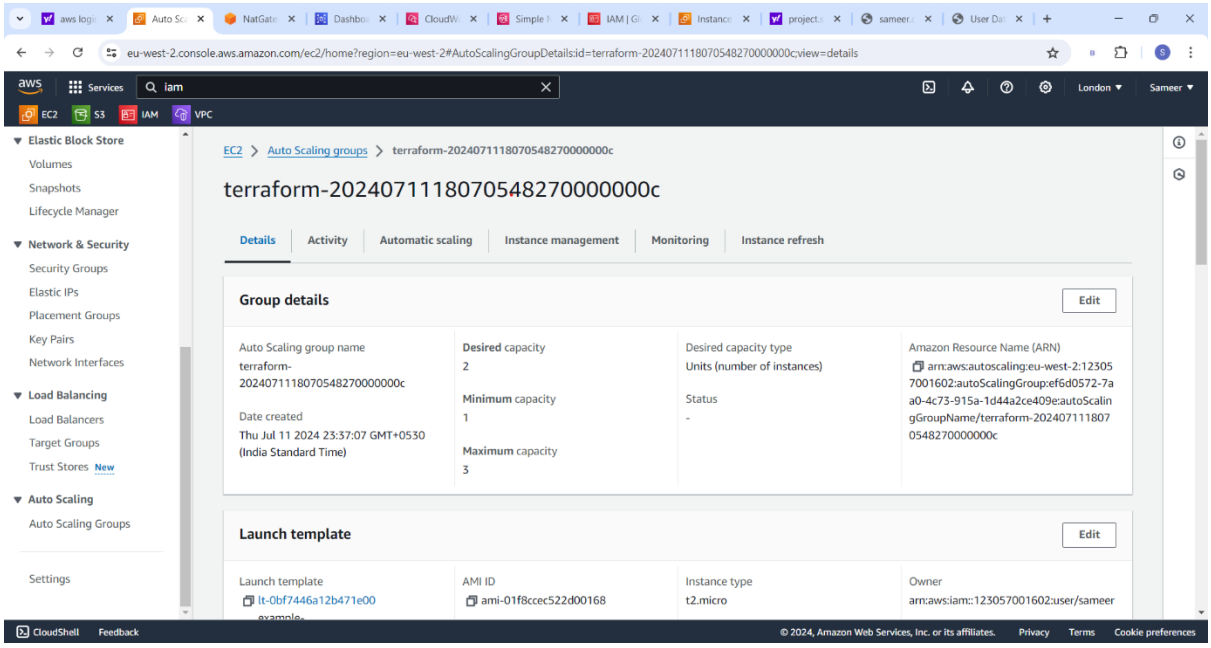
Create AMI of private instance



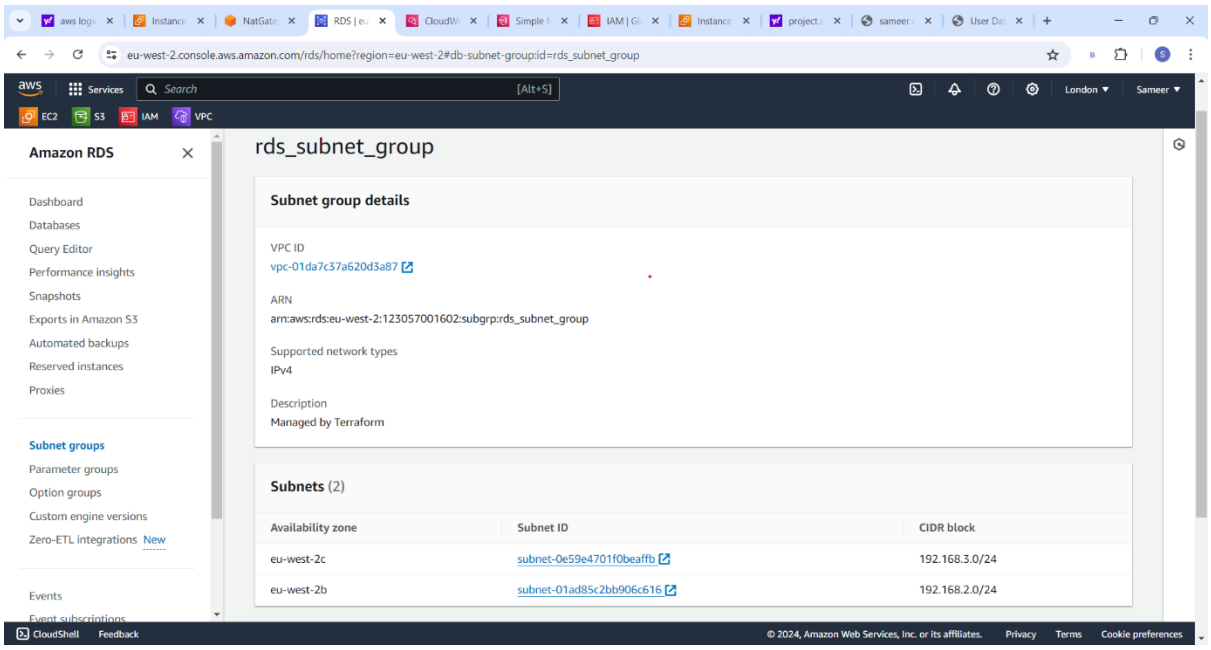
Create launch template with exisiting AMI



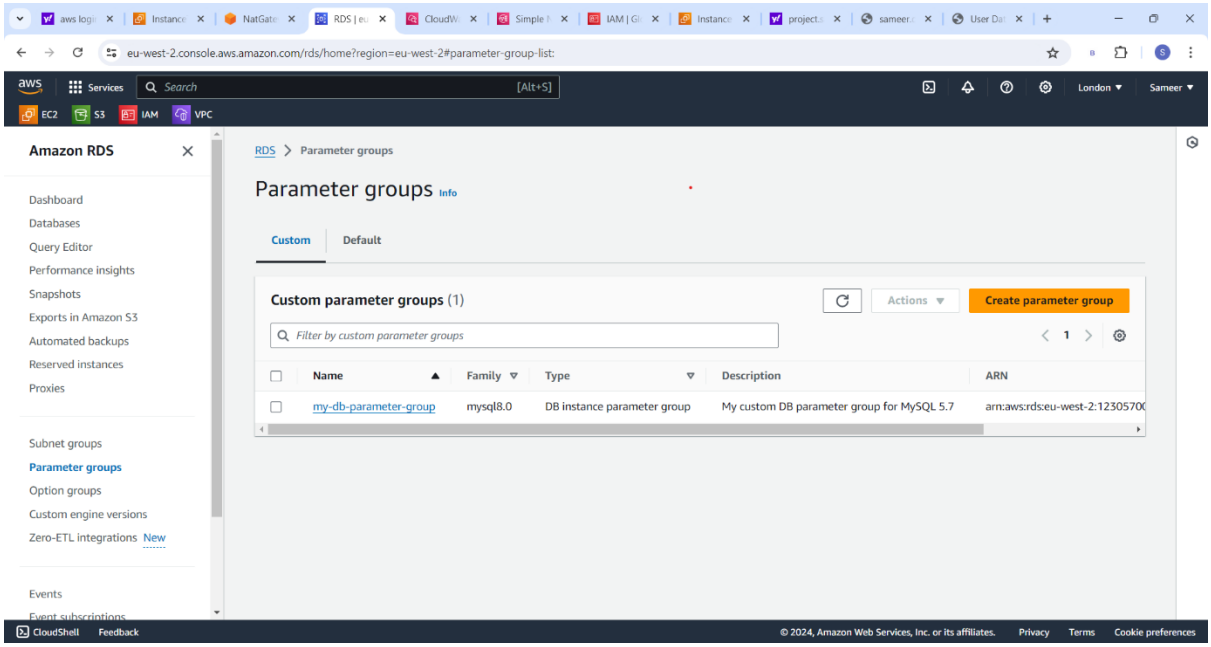
Create a launch template and auto-scaling group to manage the EC2 instances in the private subnet.



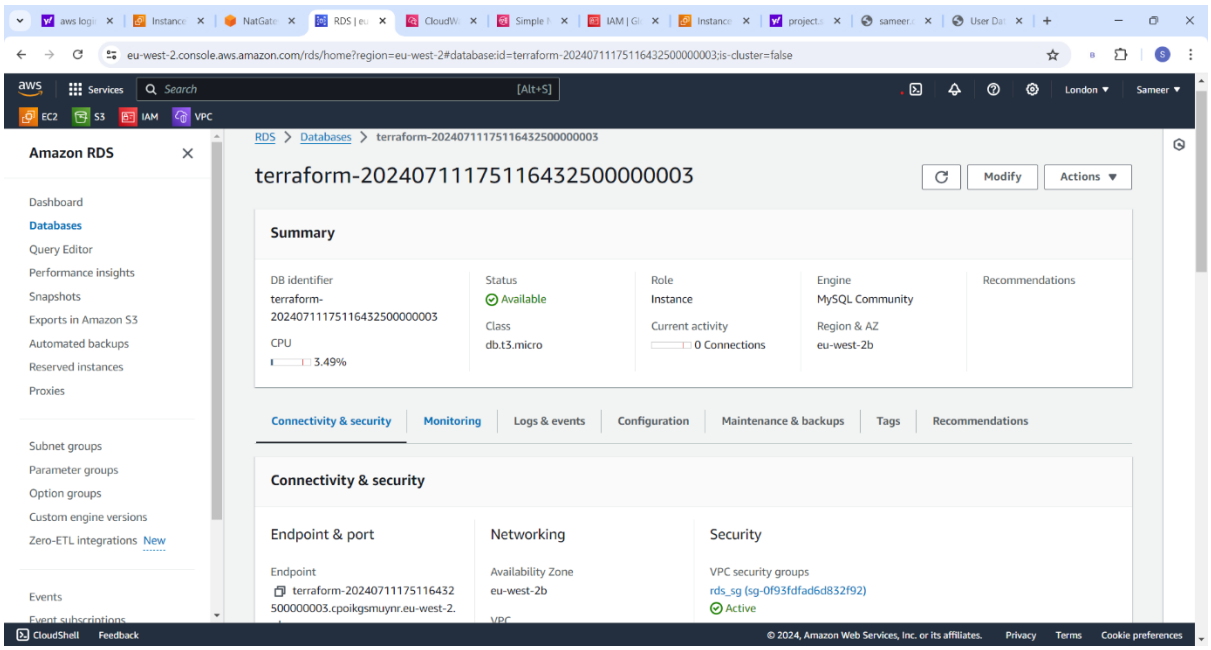
Create subnet groups for RDS



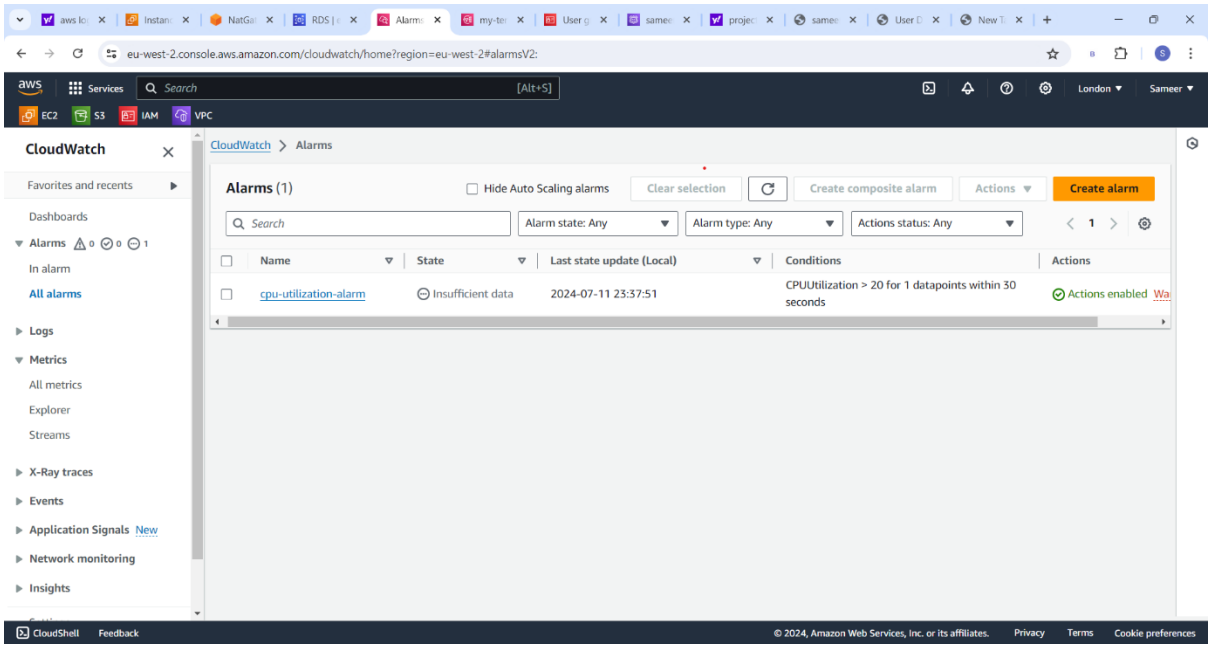
Create parameter groups .



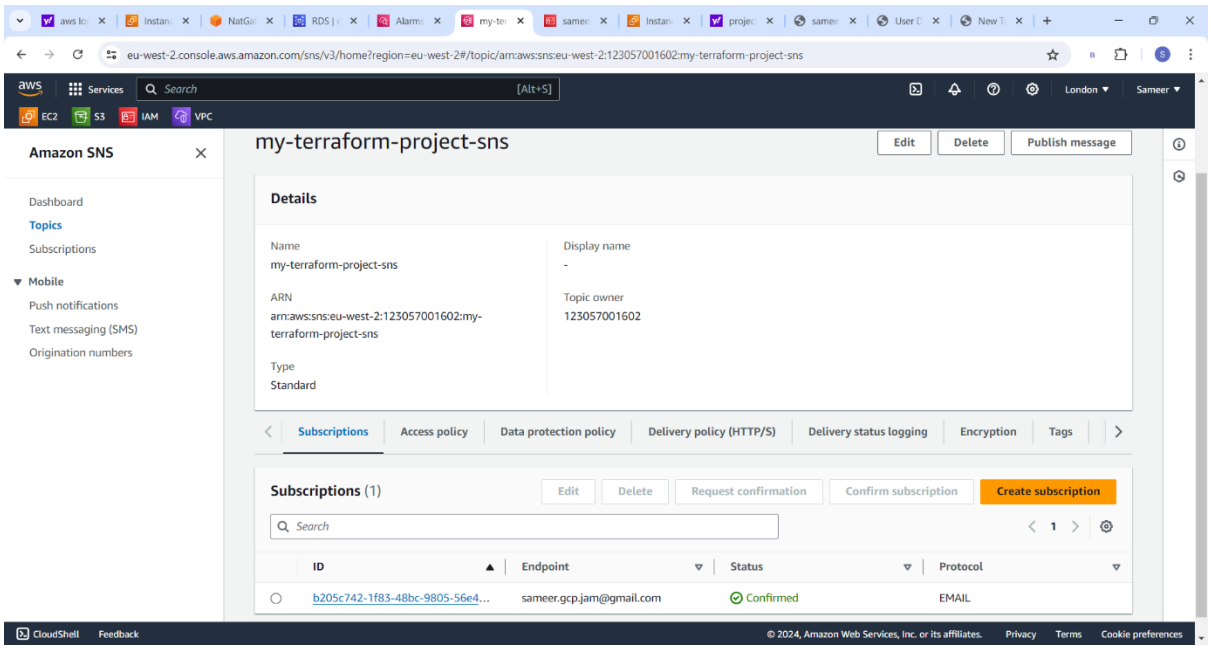
Launch an RDS instance using the created groups



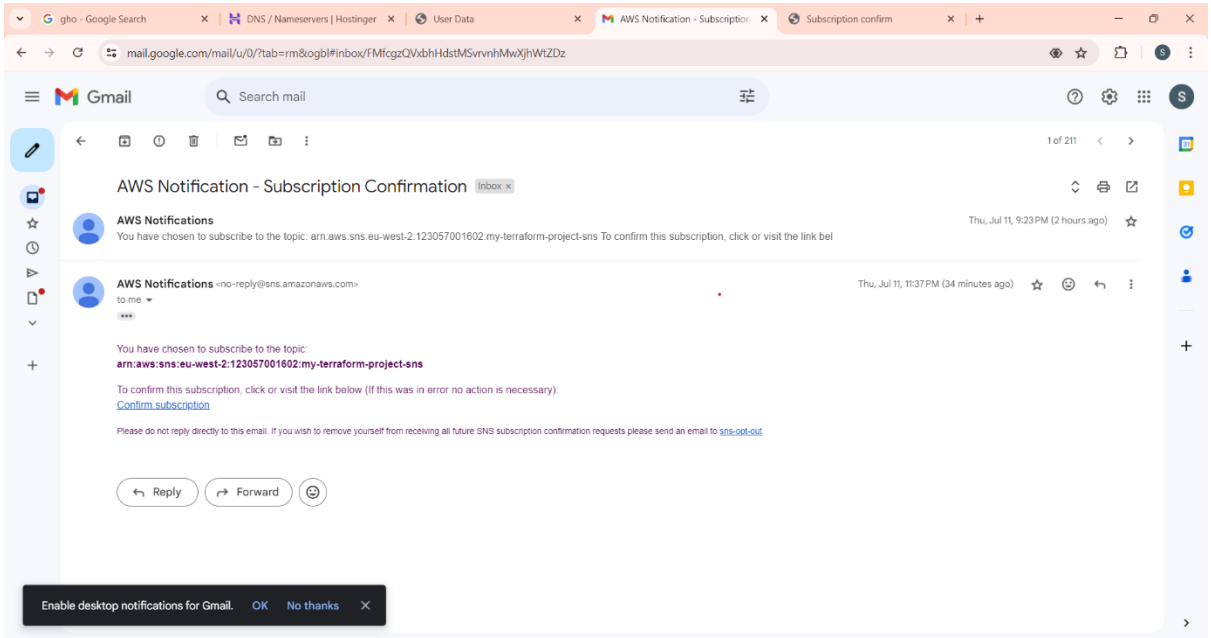
Create Alarm with CloudWatch



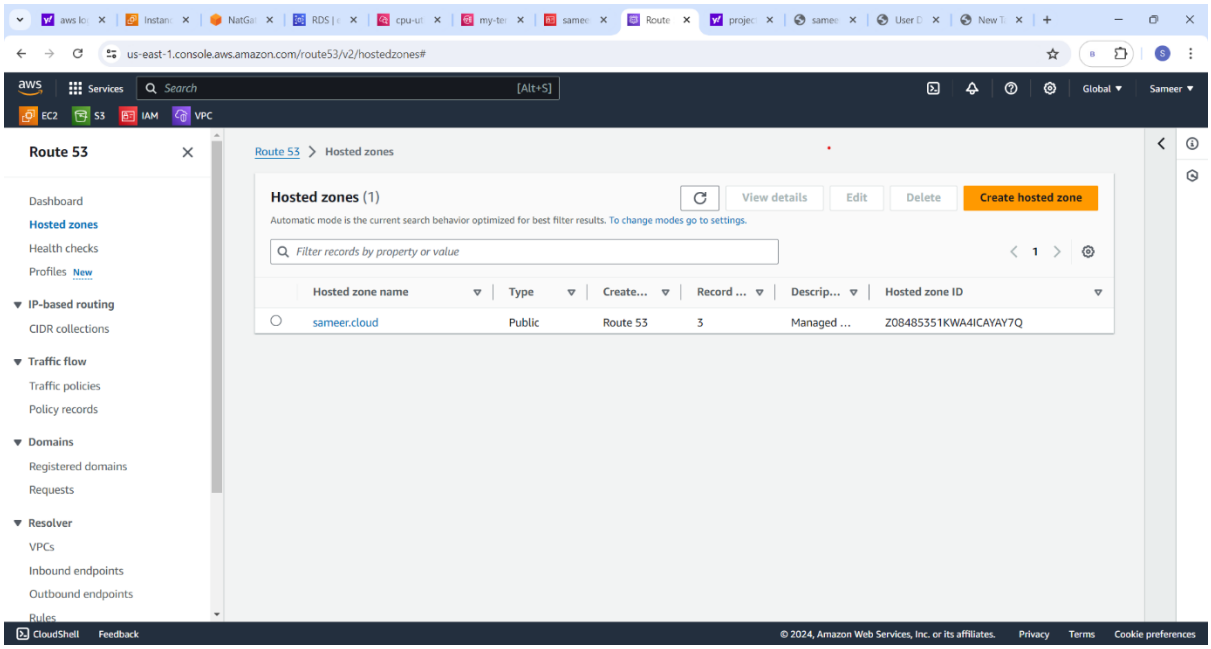
Create SNS topic



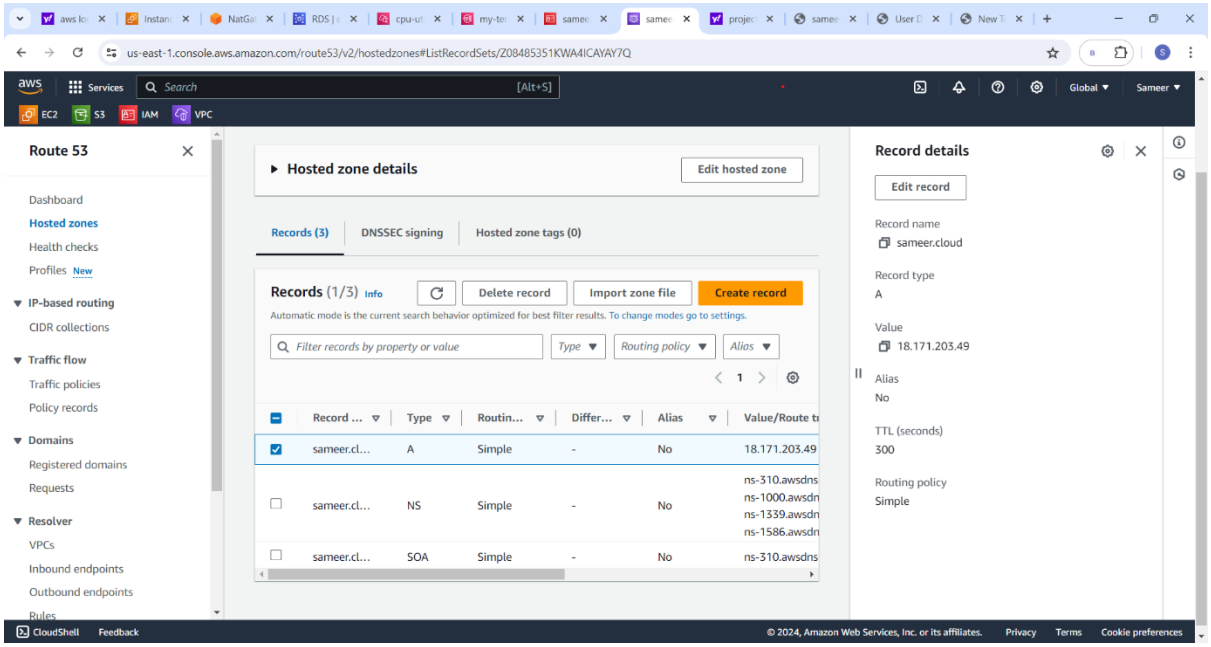
Gmail for confirmation from aws



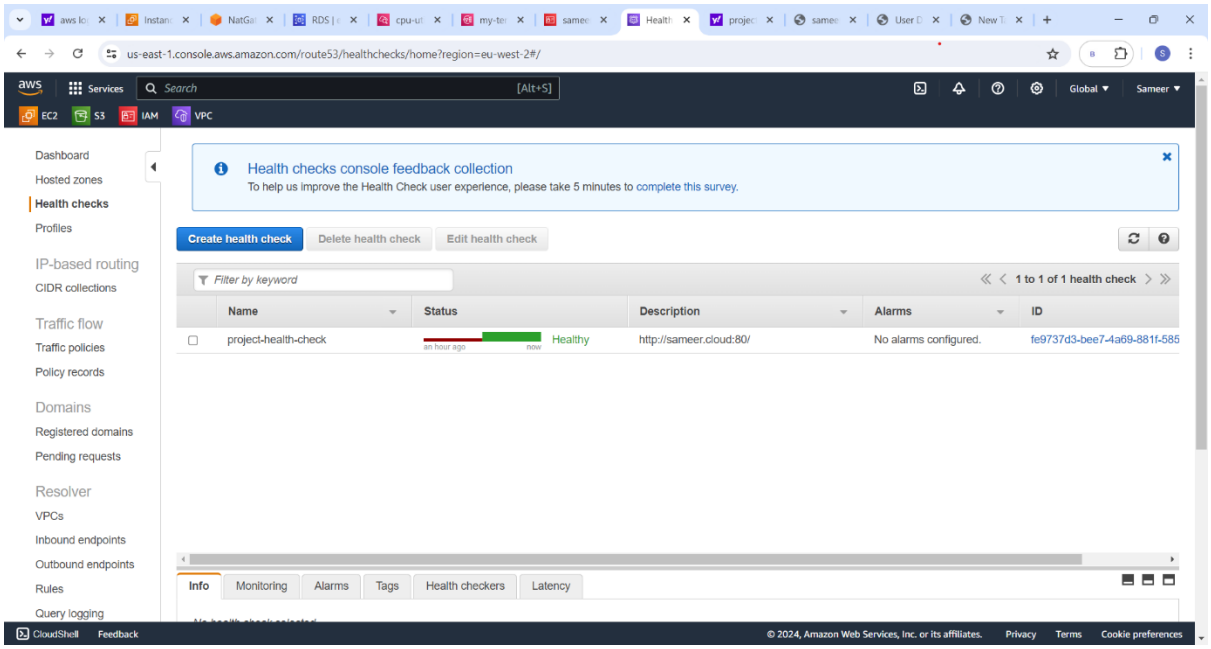
Create a Route 53 hosted zone.



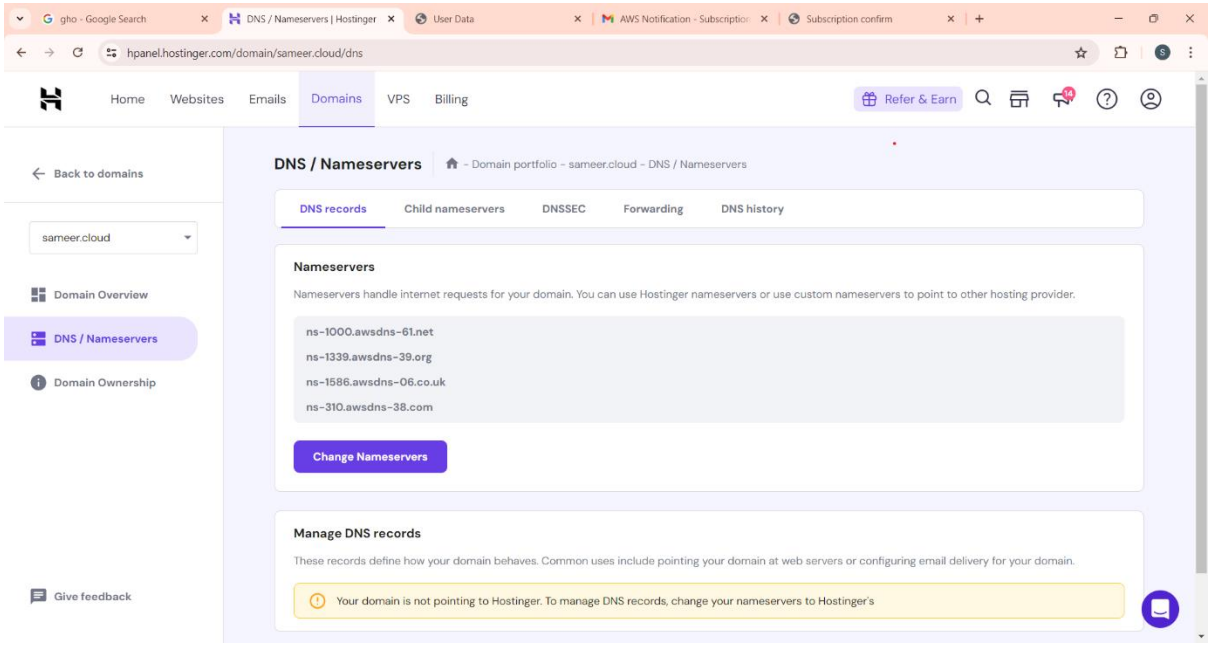
Add a new A record with a simple routing policy.



Create a health check for the DNS.



Add name server from route53 hoststage zone



Here we go, we see the application on internet.

Database is stored successfully.