ENPM818N - Mid Term Project Report

Group 4:

Aditi Lnu, Rishabh Goel, Solomon Bezabih and Tanvi Kanchan

Table of Contents

1. INTRODUCTION:

The group project report details the designing, developing and deployment phases of building our scalable and secure e-commerce platform site (***enpmprojectgroup4.shop)*** using Amazon Web Services (AWS). The project is developed in a way that the system can handle fluctuating user traffic, protect sensitive customer data and also deliver content of our e-commerce website efficiently to users globally.

After thorough research and exploring AWS resources, we have implemented the best practices of several AWS services, each of which contribute to the e-commerce website's reliability, security and performance. The services incorporated for our project are AutoScaling, Load Balancer, Amazon RDS, Amazon WAF (Web Application Firewall), CloudFront for CDN, Amazon KMS and Cloud Watch to name a few.

Auto-scaling is used to dynamically adjust the EC2 instances based on the traffic spike to maintain optimal performance. Load balancers distribute traffic amongst the EC2 instances thus enhancing fault tolerance. Amazon RDS hosts our MySQL database which helps with efficient customer and product management. WAF is set up to protect the website against common web threats like Cross-Site Scripting and SQL injection.

Further, to decrease the latency in delivering static assets to users worldwide, Amazon's CDN service, Cloudfront, is implemented. Encryption is one of the core features of the project. To encrypt data at rest, Amazon KMS feature is used and SSL is enabled to secure connections between the e-commerce platform and the RDS MySQL instance.

So to summarize, the objectives we accomplished through the project are as follows:

- Deployed a secure and scalable e-commerce platform using AWS services.
- Implemented auto-scaling to efficiently handle varying levels of traffic.
- Ensured data security through encryption and secure database management.
- Optimized content delivery for global users.
- Implemented robust security measures to protect against common web vulnerabilities.

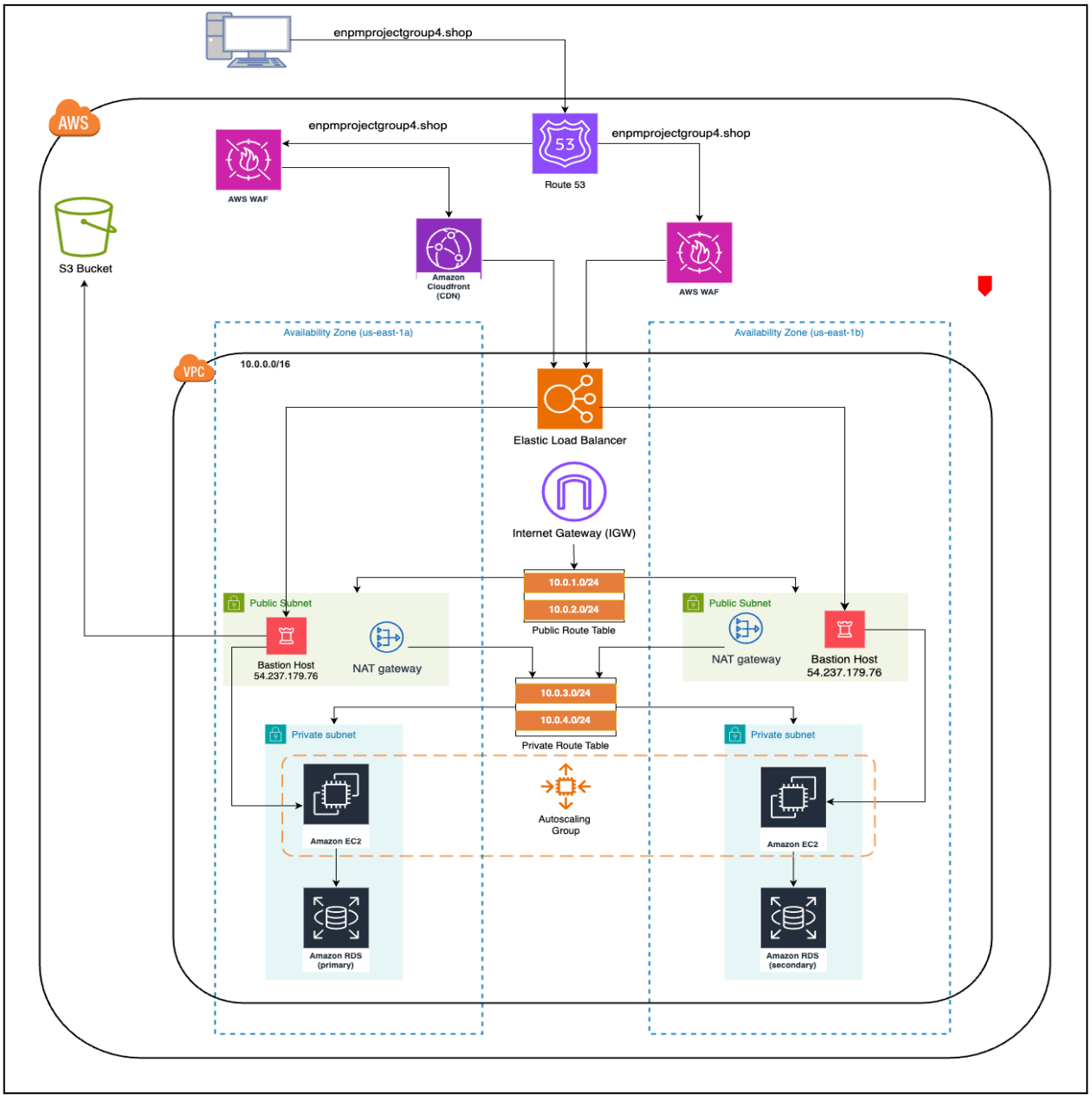
Through rigorous testing using Apache Jmeter, it can be verified that our platform fulfills the project objectives, ensuring reliability, security, and optimized performance.

2. ARCHITECTURE OF THE PROJECT:

The cloud-based architecture is designed to support a highly available, secure and scalable application on AWS for our e-commerce platform: ***enpmprojectgroup4.shop***. The architecture makes use of various foundational components like Virtual Private Cloud (VPC), Elastic Load Balancing, Auto Scaling, Route 53, Web Application Firewall (WAF), and a Cloudfront Content Delivery Network (CDN) for optimized global performance.

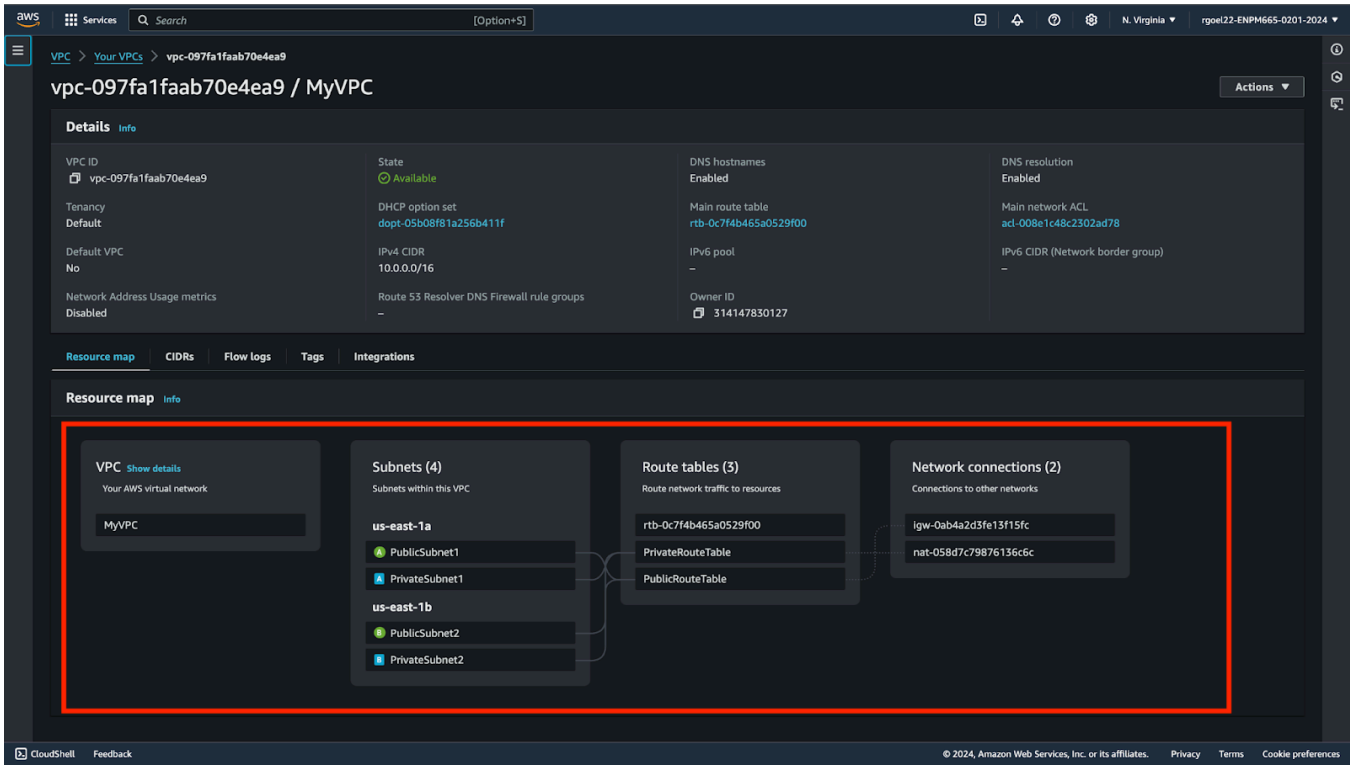
The VPC provides an isolated network resource, the Load Balancing allows for even traffic distribution across the available resources to maintain availability, an Auto Scaling group for dynamic resource management, and a relational database service (RDS) for data storage. Resources are distributed across public and private subnets within multiple Availability Zones (AZs) for fault tolerance and effective resource management. The Route53 service is also enabled to manage the DNS settings for ***enpmprojectgroup4.shop***. Additionally, the WAF and Cloudfront services are also utilized for security and optimization purposes.

The following diagram illustrates how the above mentioned services are integrated to form our architecture.



As seen in the diagram above, the workflow of our architecture can be explained as follows:
An internet-user will search for ***enpmprojectgroup4.shop*** on their web browser, triggering a DNS lookup through ***Route53***. Route53 then directs the request to ***CloudFront***, which, integrated with ***WAF***, analyzes the traffic for any malicious pattern before allowing it to proceed. Requests for static assets are directly served through CloudFront's cache, while dynamic requests are forwarded to the ***Load Balancer*** (LB). The LB distributes these incoming requests to the EC2 instances. These instances scale up or down based on demand. For any data-related operations required by the backend EC2 instances, the requests are processed through ***Amazon RDS***. The instance is configured to handle primary database operations in the primary availability zone, with a secondary availability zone instance for failover support. ***CloudWatch*** monitors instance CPU utilization and triggers scaling policies in the ***Auto Scaling Group*** as needed.

2.1 Virtual Private Cloud (VPC):



The VPC has been allocated the IPv4 CIDR block of ***10.0.0.0/16***, which provides a large address space for assigning IP addresses to various subnets within the VPC. The DNS hostname and resolution options are enabled to facilitate smooth internal name resolution, enhancing connectivity among instances.

VPC includes four subnets, split across two Availability Zones (us-east-1a and us-east-1b). This approach provides high availability and fault tolerance:

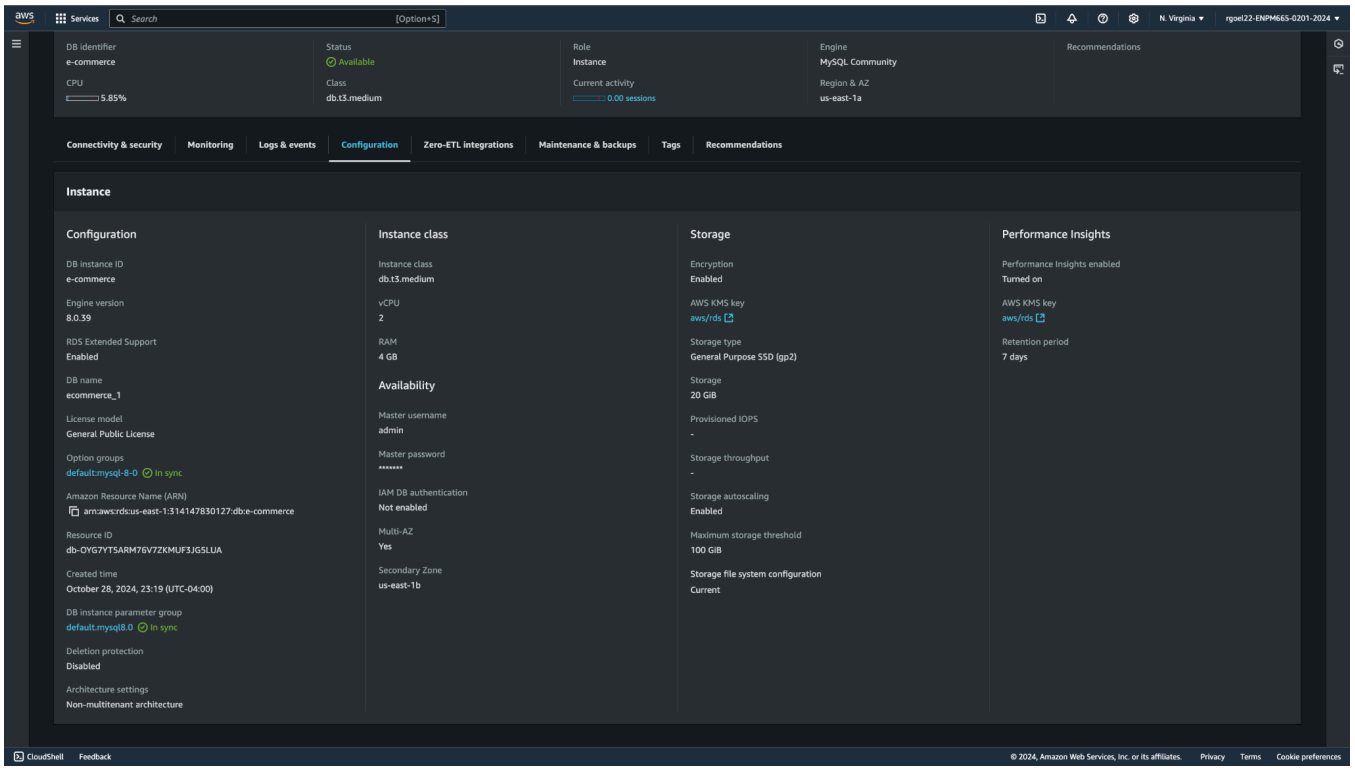
- Public Subnet 1 (***10.0.1.0/24***) and Public Subnet 2 (***10.0.2.0/24***): These subnets are intended for resources that require internet access, such as the Application Load Balancer (ALB) and the Bastion Host. The public subnets are configured to automatically assign public IP addresses to resources upon launch.
- Private Subnet 1 (***10.0.3.0/24***) and Private Subnet 2 (***10.0.4.0/24***): These are internal subnets that have sensitive resources such as EC2 instances within the Auto Scaling Group (ASG) and the RDS databases. These subnets are isolated from direct internet access to ensure data security.

The VPC consists mainly of three route tables to manage traffic routing within the network:

- Public Route Table: This table routes outbound traffic from public subnets to the internet through the Internet Gateway (***igw-0ab4a2d3fe13f15fc***). This allows resources in public subnets to communicate with external networks.

- Private Route Table: This table directs outbound internet traffic from public subnets through a Network Address Translation (***nat-058d7c79876136c6c***) Gateway, which provides a secure way for internal resources to access the internet without exposing them to public traffic.
- Main Route Table (***rtb-0c7f4b465a0529f00***): This route table handles default network routing for the VPC and is linked with the primary network ACL (Access Control List) for additional security control.

2.2 Relational Database Service (RDS):



The image illustrates the configuration details for our RDS instance, tailored for our e-commerce platform. With features like autoscaling, multi-AZ deployment, and performance insights enabled, our database is optimized for high availability, reliability, and efficient performance under varying load conditions.

2.2.1 RDS Configuration Details:

- DB Instance ID: e-commerce
- Engine and Version: MySQL engine, version 8.0.39
- Instance Class: db.t3.medium, this instance has 2 vCPUs and 4 GB of RAM

2.2.2 Storage and Scalability:

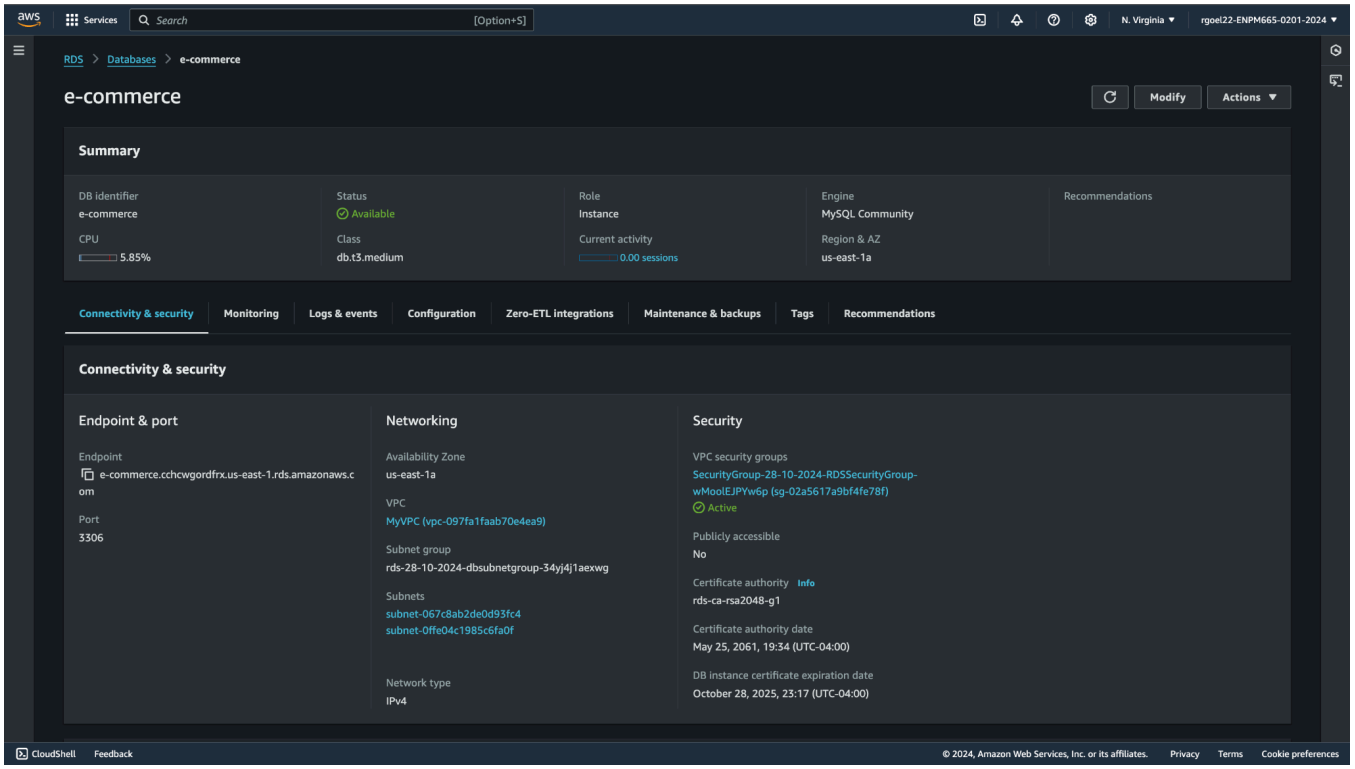
- Storage Type and Size: General Purpose SSD (gp2) storage with an allocated size of 20 GiB
- Storage Autoscaling: Autoscaling is enabled, allowing the storage capacity to increase automatically up to a maximum threshold of 100 GiB
- Encryption: AWS Key Management Service (KMS)

2.2.3 High Availability and Reliability:

- Multi-AZ Deployment: primary instance in us-east-1a and a secondary standby instance in us-east-1b
- Backup and Retention: Backups are enabled with a default retention period of 7 days, safeguarding data recovery options in case of unexpected data loss.

2.2.4 Access and Monitoring:

- Master User: Accessible by admin user
- Performance Insights: enabled



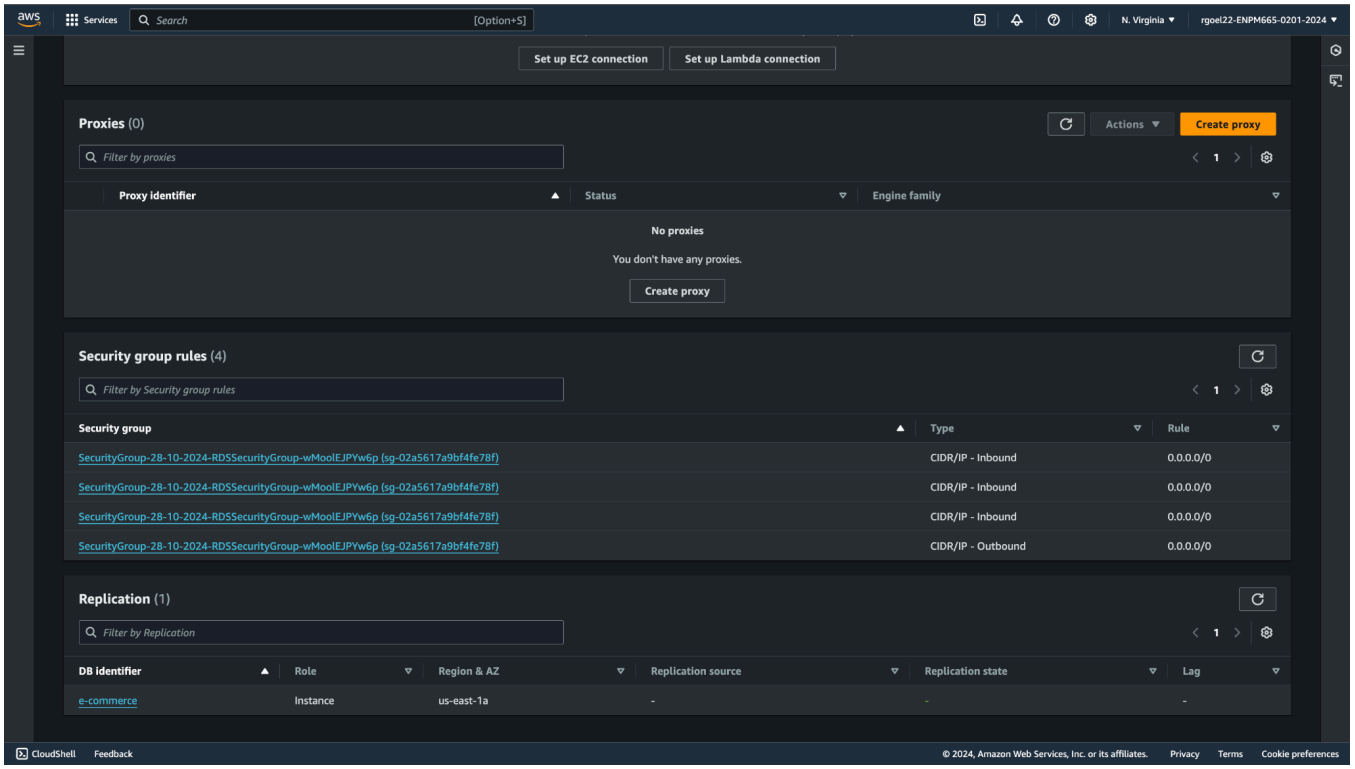
The above image outlines the connectivity and security settings for our RDS instance, designed to ensure secure and controlled access within our ecommerce platform's infrastructure. It highlights critical components like the VPC configuration, endpoint accessibility, and security measures, ensuring robust protection and optimal performance.

2.2.5 Connectivity:

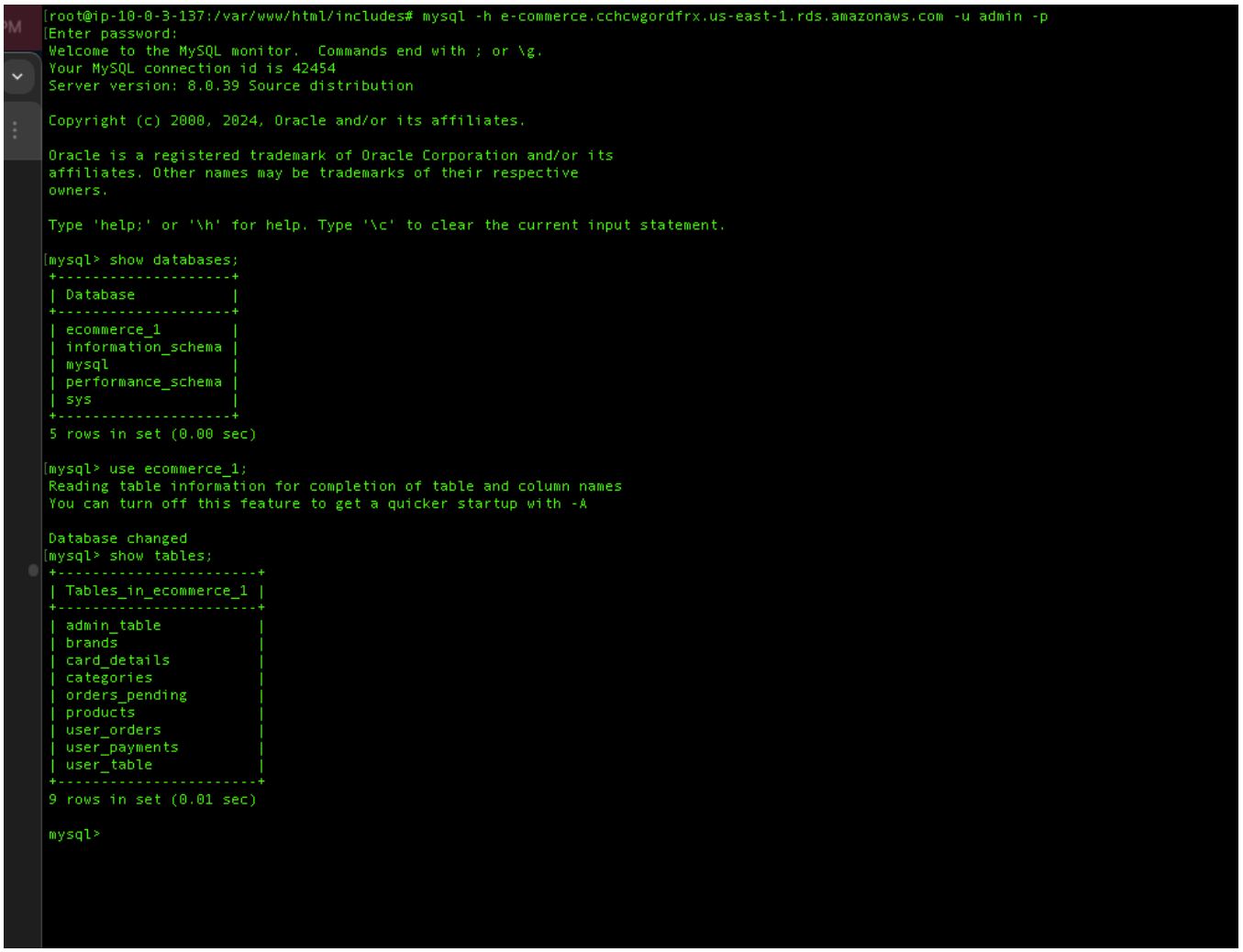
- Endpoint and Port: The database can be accessed using the endpoint (***e-commerce.cchwcgordfrx.us-east-1.rds.amazonaws.com***) on port ***3306***.
- Networking: The RDS instance is hosted within the VPC (***vpc-097fa1faab70e4ea9***) in the us-east-1a Availability Zone.
- Subnet Group: The database is associated with a subnet group (***rds-28-10-2024-dbsubnetgroup-34yj4j1aexwg***), which includes private subnets.

2.2.6 Security:

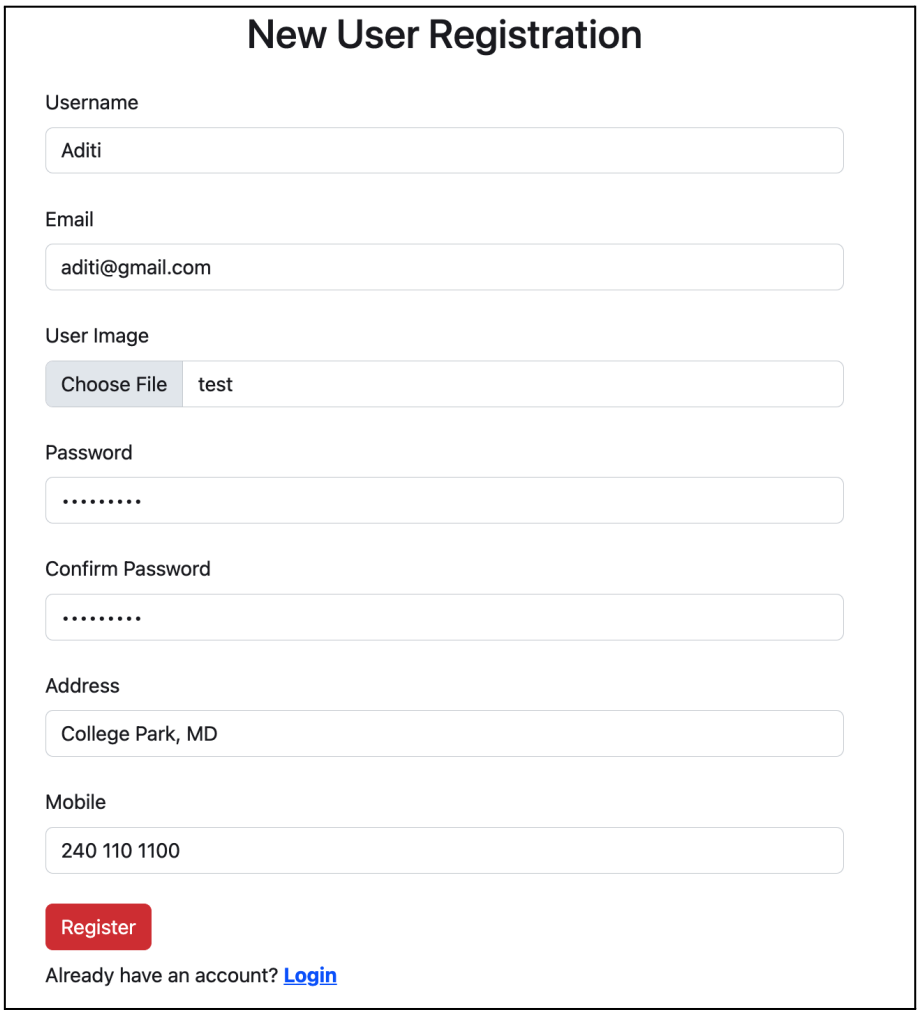
- VPC Security Groups: RDS instance is protected by a dedicated security group (***SecurityGroup-28-10-2024-RDSSecurityGroup-wMooEJPvW6p***), which strictly controls inbound and outbound traffic.
- Access Control: The database is configured as "***Not Publicly Accessible***", which implies that it is only accessible to resources within the VPC.
- Encryption: This layer supports secure data transmission, with the endpoint being managed within the VPC and leveraging security groups.



The above image shows that the instance has been configured in us-east-1a as a standalone instance for this setup, without cross-region replication.



The above image provides an overview of the database structure within the Amazon RDS instance for the **ecommerce_1** database, demonstrating the tables that form the core of the e-commerce application. This database schema includes essential tables such as **admin_table**, **brands**, **card_details**, **categories**, **orders_pending**, **products**, **user_orders**, **user_payments**, and **user_table**.



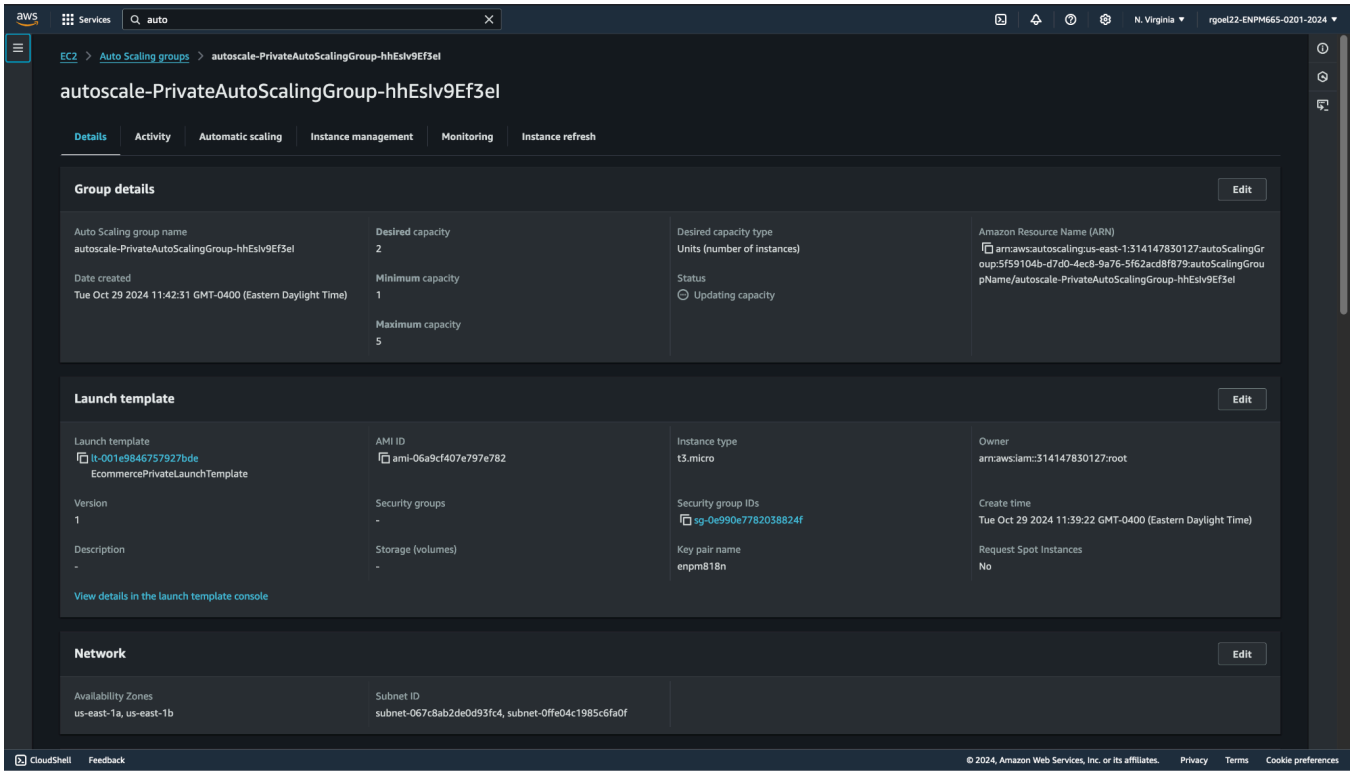

```
Database changed
mysql> select * from user_table;
+---------+----------+---------------------+--------------------------------------------------------------+-----------------------------------+-------------+----------------+-------------+
| user_id | username | user_email          | user_password                                                | user_image                        | user_ip     | user_address   | user_mobile |
+---------+----------+---------------------+--------------------------------------------------------------+-----------------------------------+-------------+----------------+-------------+
|       1 | abdo     | abdo@gmail.com      | $2y$10$5ynby9fq7vf22wHlkvehu.JGbK6r7zZLtLzuJz9Jt5FP03rGZ9Mj. | new logo after Edit1920.png       | ::1         | Cairo          | 123456789   |
|       2 | test     | test@test.com       | $2y$10$1fParoAjj.EPZd7OFhrnXuQ2L3hHXb5FpRYAQKrwuxqmrVxKZiybO | Screenshot 2024-10-13 at 7.03.11 PM.png | 50.237.13.62 | 21122121       | 2404670423  |
|       3 | testuser | user@test.com       | $2y$10$1u5b40LFtnlbcNTV6NUHqOxkelCU4BzZvPjnpja1hjZOxGc.Gt/hW | Screenshot 2024-10-13 at 7.03.11 PM.png | 50.237.13.62 | adsasd         | 1212122121  |
|       4 | tanvitest | test@gmail.com      | $2y$10$bGWvS7mljSVrRQ.62Mz/3OAY1h7oa79Qoplem.wnS/7IYdqF.8PPO | image.jpg                         | 69.137.233.24 | College Park, MD | 1234567890  |
|       5 | test1234 | rayiv86796@runtan.com | $2y$10$MSpb0/eBFlpL2n7j6Yx3zOmlbPKz7Xszk4bjCDZoOBKwtLoUK5WT6 | image.jpg                         | 69.137.233.24 | College Park, MD | 1234567890  |
|       6 | test1212 | sdhdsjh@dhsjds.com  | $2y$10$wQNUYT5Bb4rC1FRArdeG6errKMYTB4oJ6Fompl2JyOTthQk5DG3rC | bg.png                            | 50.237.13.62 | dasd           | 123123123   |
|       7 | Aditi    | aditi@gmail.com     | $2y$10$6OEKv.GM0t8u4NDRxxSCZeUrzZED.60.JTrkghENyRVV1mVO2vEcC | test.png                          | 50.237.13.57 | College Park, MD | 240 110 1100 |
+---------+----------+---------------------+--------------------------------------------------------------+-----------------------------------+-------------+----------------+-------------+
7 rows in set (0.00 sec)

mysql>
```

Basic create, read, update and delete (CRUD) operations on the database via the e-commerce platform.

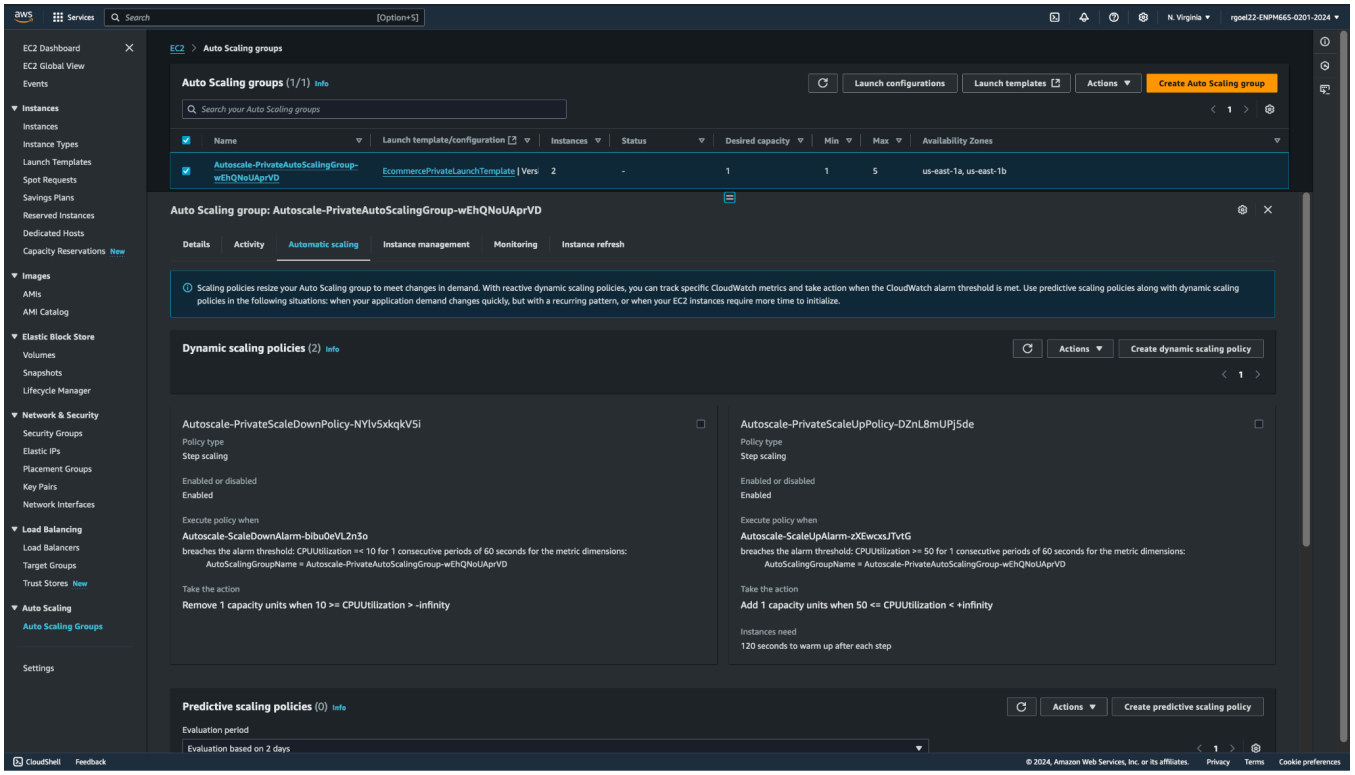
2.3 Auto Scaling Groups:

The Auto-Scaling group for the private instances. Our application is based on the private instances and we route it through Bastion Host for added security, hence the private instances need to be scaled based on the traffic spike. Based on our requirements, we have defined the parameters as following:

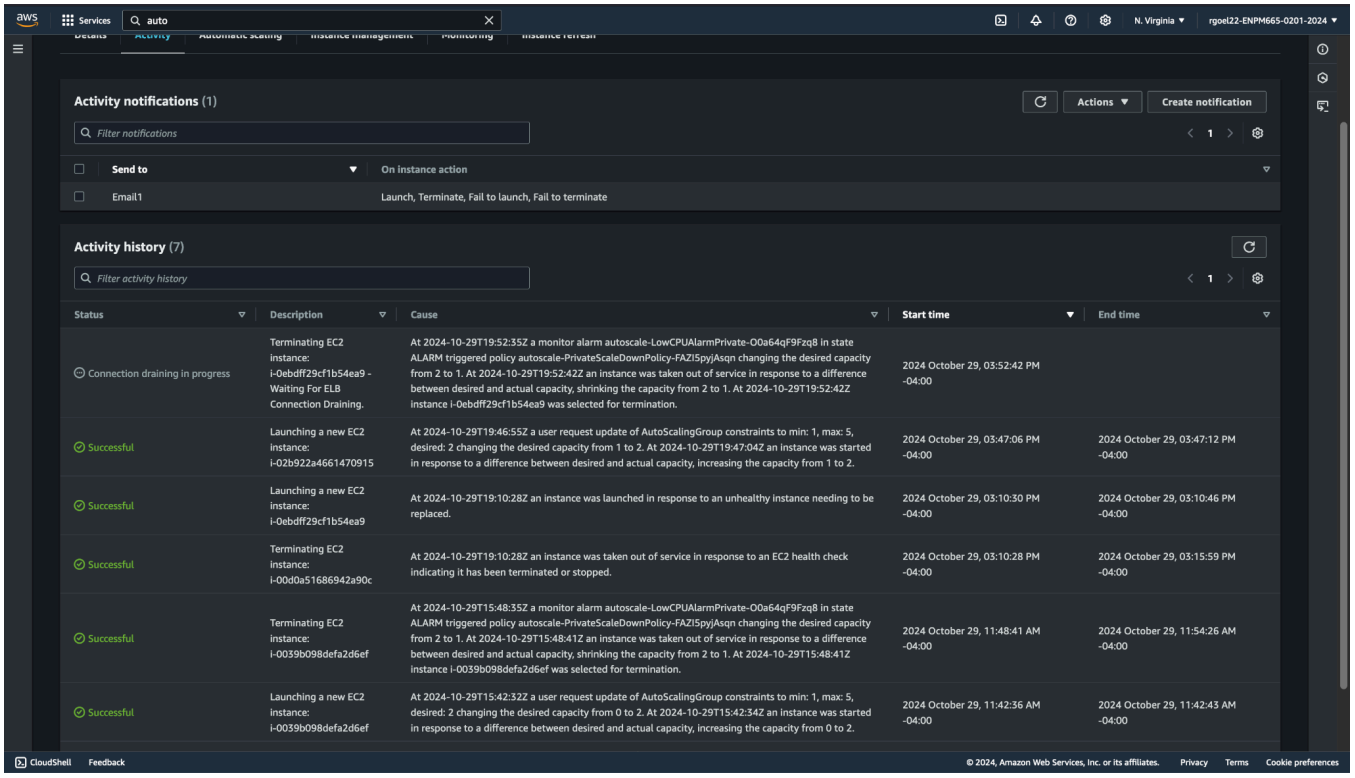
- ***Desired capacity - 2***
- ***Minimum Capacity -1***
- ***Maximum Capacity - 5***



The Auto-scaling group follows two simple dynamic scaling policies. If the CPU utilization ***exceeds the 50%*** threshold for ***1 consecutive period*** of ***60 seconds*** , that is 1 minute, and an ***instance is added*** to the available resources. Similarly, if the CPU utilization is ***less than 10%*** for 1 minute, an ***instance is removed*** from the pool of available resources. We have ensured a cooldown period of 2 minutes after each scaling action to prevent flapping[].

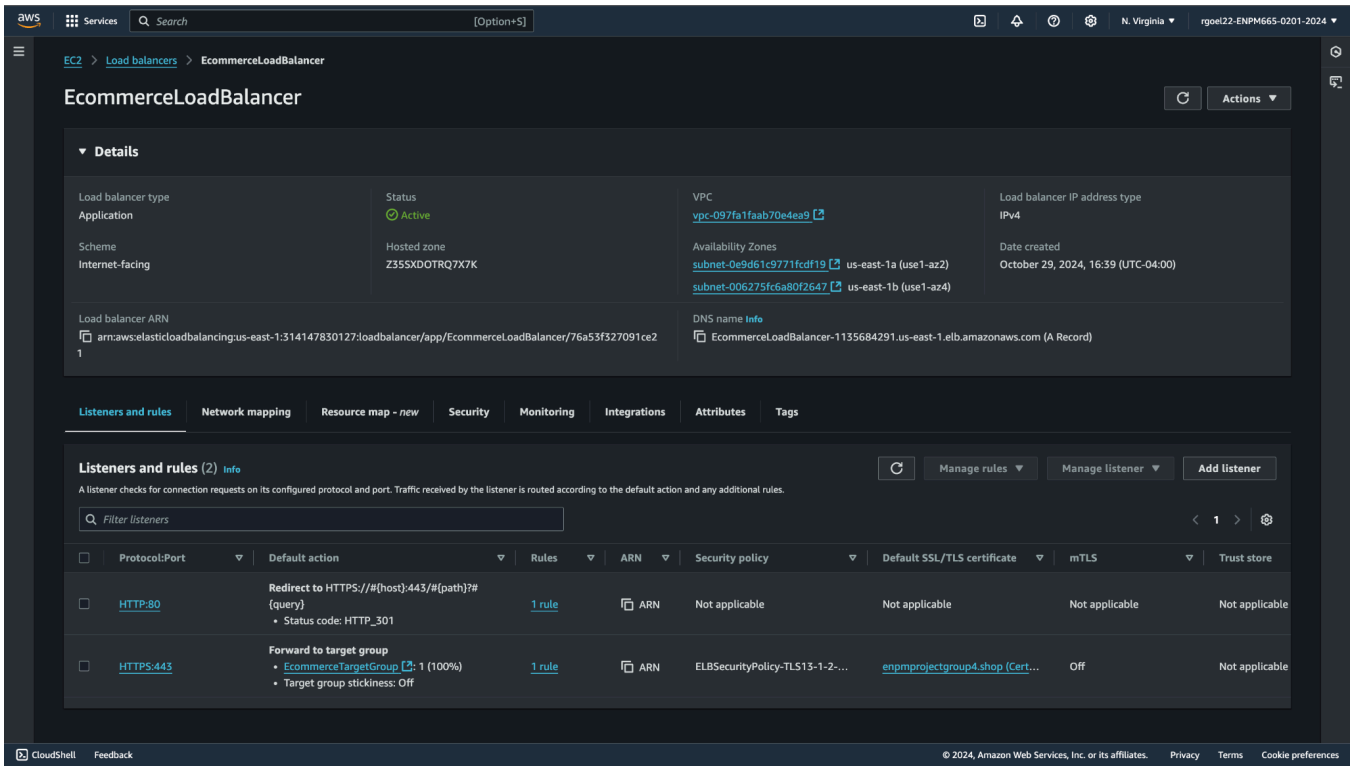


The scaling policies can be checked in real-time through the activity history. The instances scale up when an unhealthy instance is detected and it scales down when traffic detected is low on the instance.



2.4 Load Balancer:

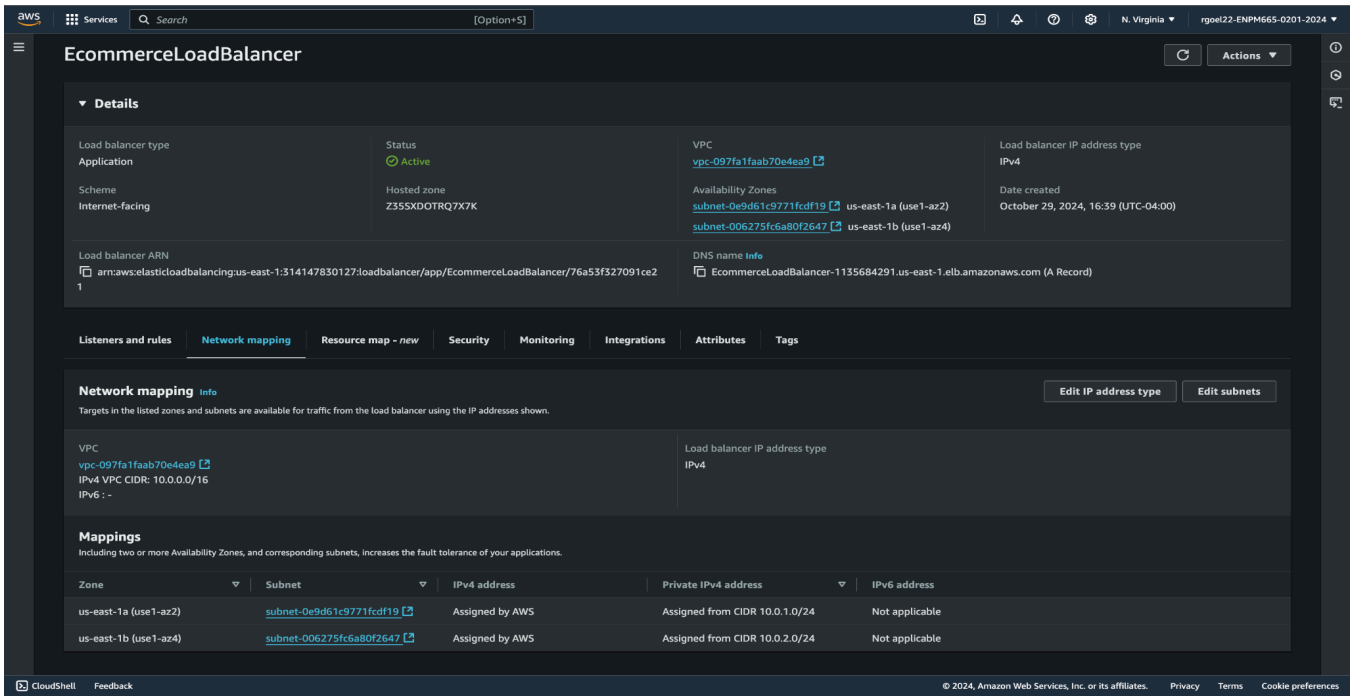
The image below presents the EcommerceLoadBalancer, designed to increase the availability and performance of our e-commerce platform by evenly distributing incoming traffic. It operates across two Availability Zones (***us-east-1a*** and ***us-east-1b***), further enhancing availability and resilience by routing traffic between these zones.



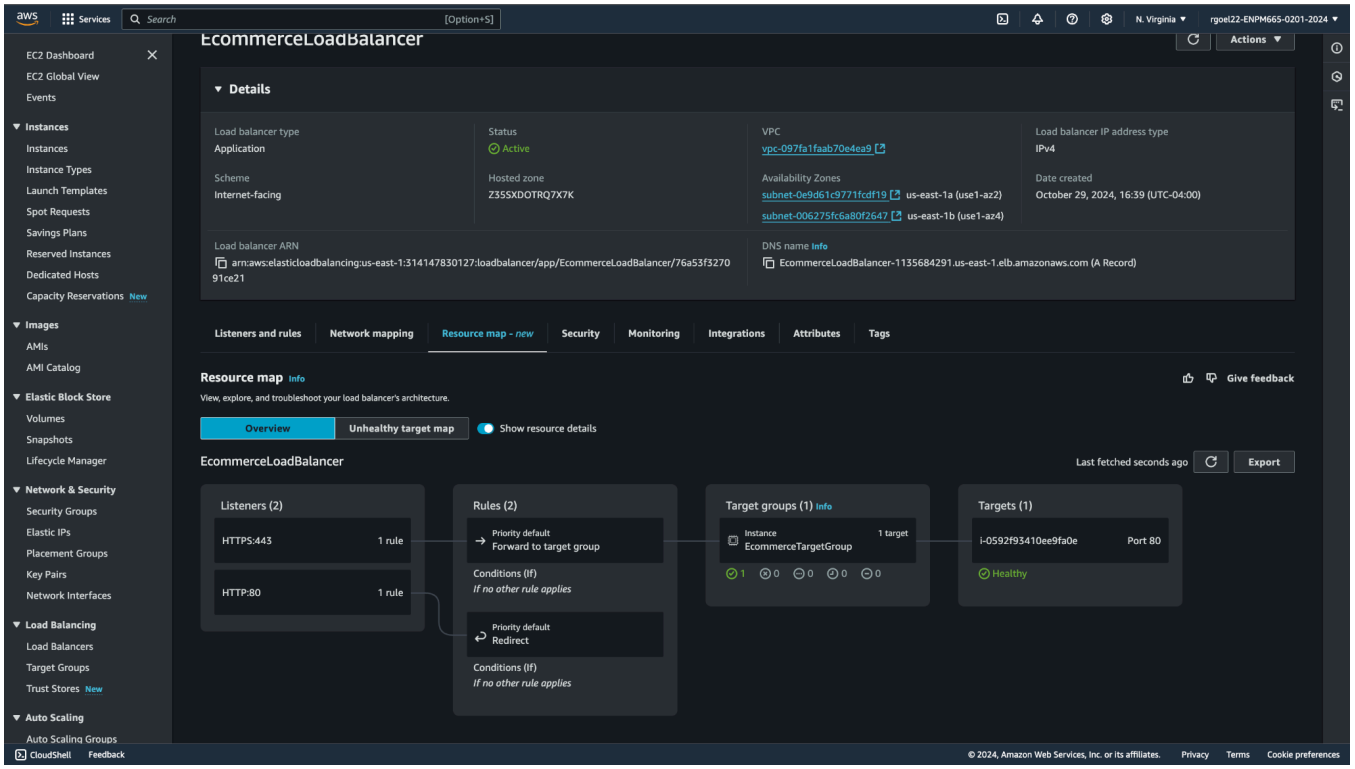
The load balancer has two listeners: an HTTP listener on ***port 80*** and an HTTPS listener on ***port 443***, respectively.

- HTTP listener captures unencrypted requests and permanently redirects them to HTTPS using a 301 status code, ensuring all traffic is secure.
- HTTPS listener handles encrypted requests, forwarding them to the ***EcommerceTargetGroup***, where they are distributed across EC2 instances in the Auto Scaling Group. Configured with SSL/TLS certificates for the ***enpmprojectgroup4.shop*** domain, the HTTPS listener ensures data privacy and integrity. Additionally, a security policy (***ELBSecurityPolicy-TLS-1-2-2021-07***) applies secure protocols supporting a secure connection for all user traffic.

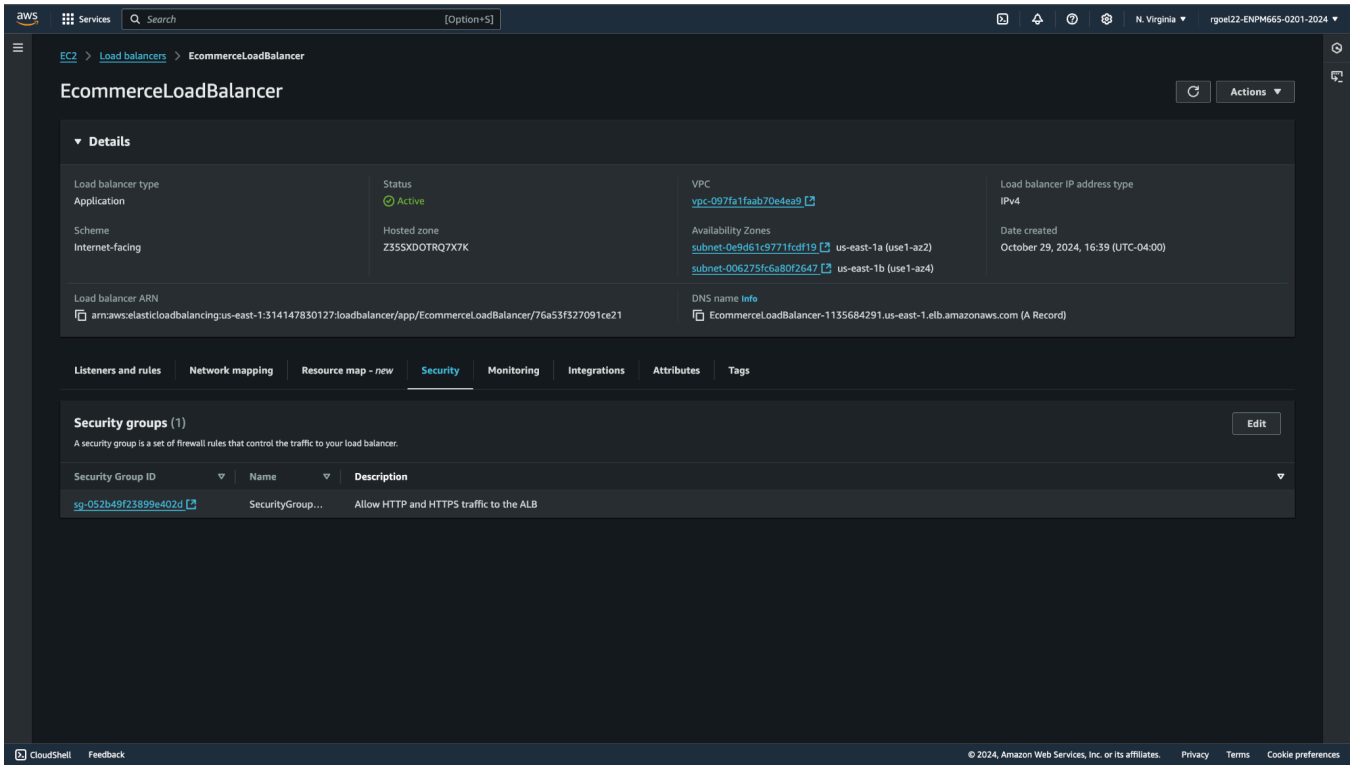
The load balancer, with the dual listener configuration and security policies, provides a robust approach to handling user traffic. It ensures that users access the application through secure connections, while efficiently balancing the load across multiple backend instances.



The load balancer operates in an internet-facing scheme within the ***vpc-097fa1faab70e4ea9*** VPC, which has an IPv4 CIDR block of ***10.0.0.0/16***. This setup utilizes two Availability Zones—us-east-1a and us-east-1b—each associated with a designated public subnet (***10.0.1.0/24*** for us-east-1a and ***10.0.2.0/24*** for us-east-1b).



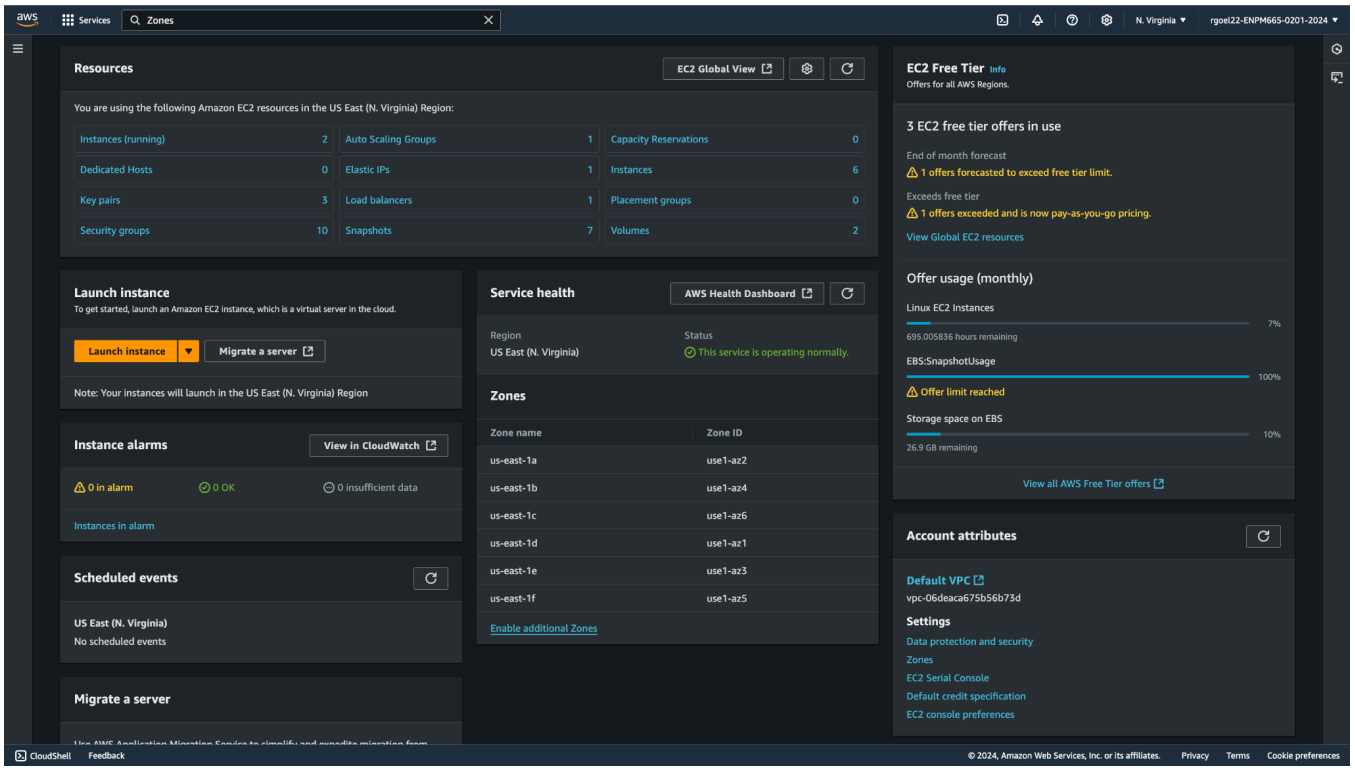
The HTTPS listener forwards the secure traffic to a designated target group, ***EcommerceTargetGroup***, which contains one registered EC2 instance on port 80. This instance is marked as "***Healthy***", indicating it is ready to handle incoming traffic.



The ***EcommerceLoadBalancer*** is secured by a dedicated security group (***sg-052b49f23999e402d***), which is configured to allow HTTP and HTTPS traffic. This security group plays the role in managing and restricting inbound traffic, ensuring that only web traffic on ***port 80*** (HTTP) and ***port 443*** (HTTPS) can reach the load balancer.

2.5 Availability Zones:

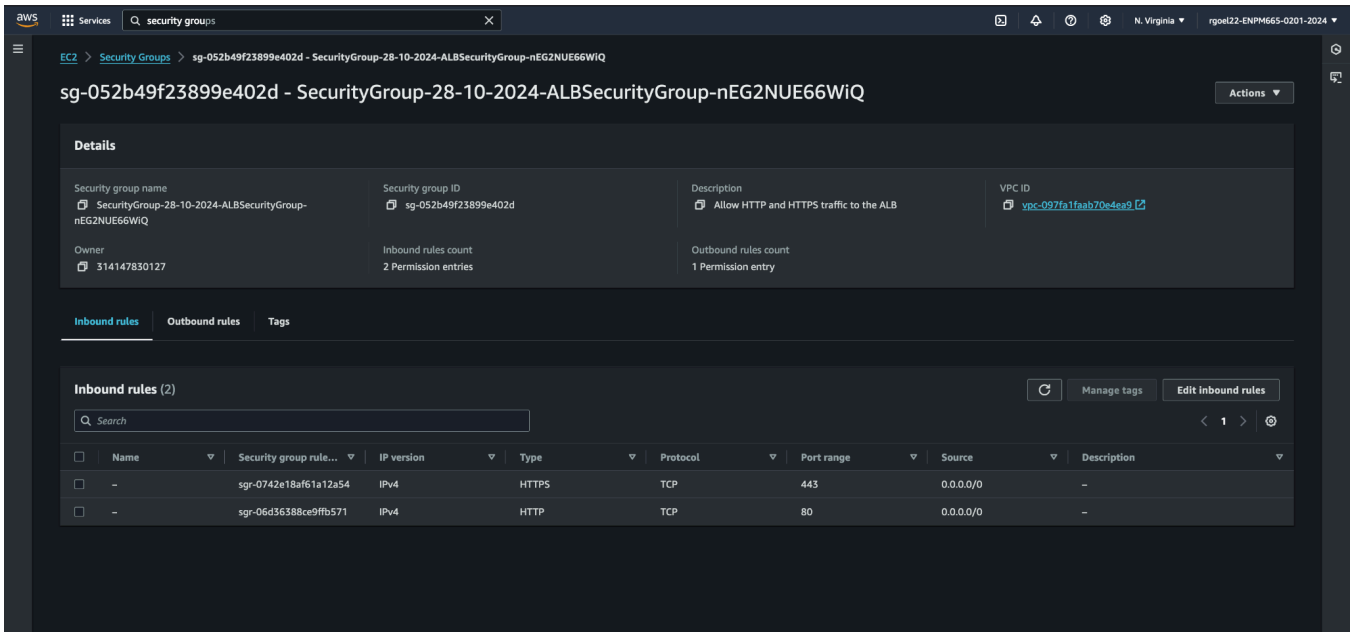
The Availability Zones in the US East (***N. Virginia***) region provide a reliable, distributed infrastructure for our application. We use zones ***us-east-1a*** and ***us-east-1b*** to enhance fault tolerance and ensure high availability.



2.6 Security Groups:

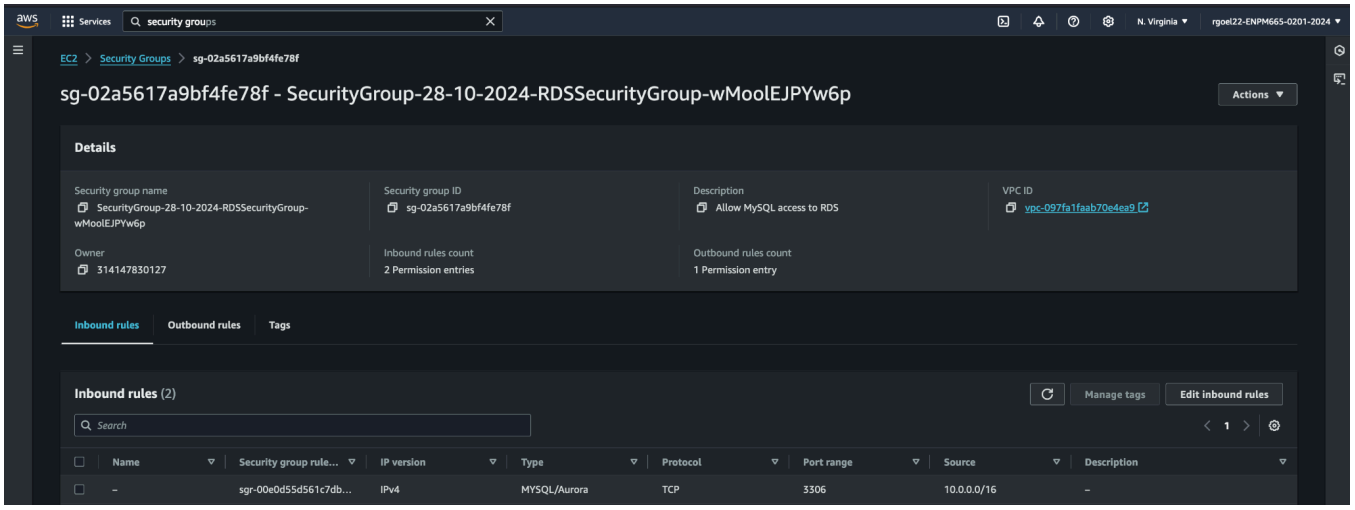
2.6.1 ALB

As the load balancer has to be internet-facing, in the Application Load Balancer security group, we allow ***port 80*** (HTTP) and ***port 443*** (HTTPS) traffic for any IP address (***0.0.0.0/0***). In this way, our application is available globally.



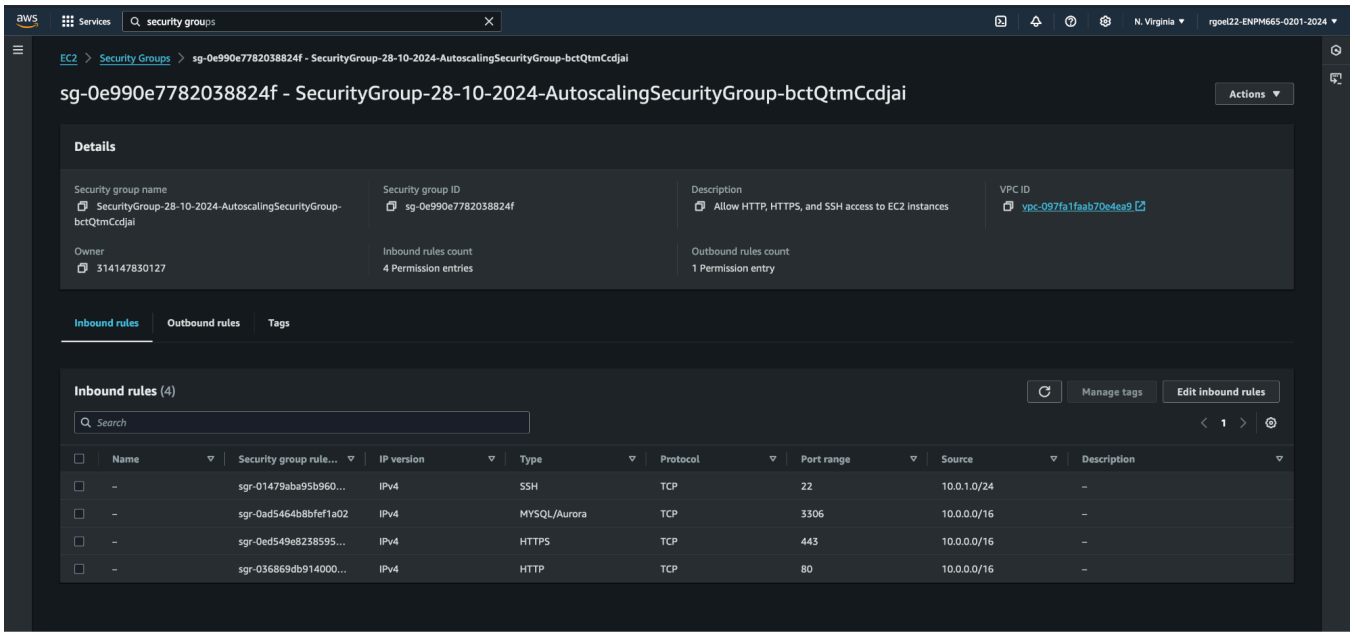
2.6.2 RDS:

In the RDS Security group, we only allow database connections from within the VPC. The inbound rule shown in the image indicates the same, all TCP connections from the VPC will be directed to ***port 3306***.



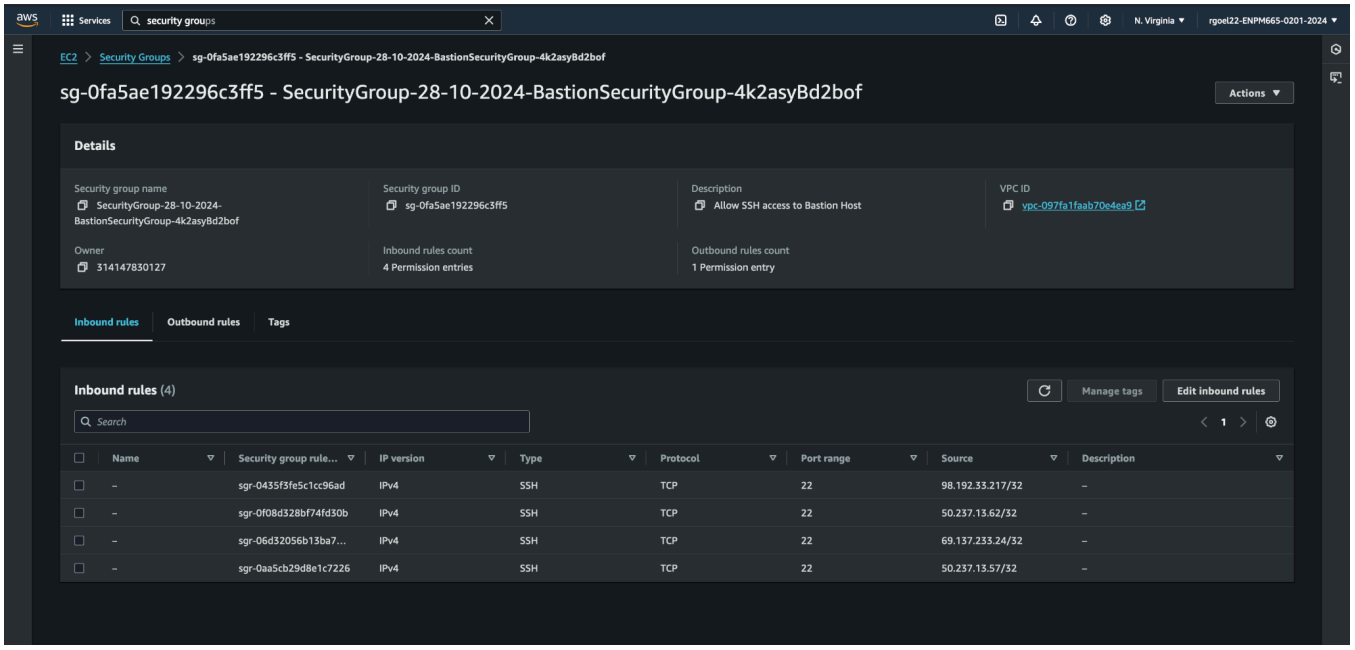
2.6.3 Auto-Scaling:

In the Auto Scaling security group, we allow all ***port 80*** (HTTP) and ***port 443*** (HTTPS) traffic from the VPC CIDR ***10.0.0.0/16*** that typically originates from the Application Load Balancer. Additionally, we only allow SSH (port 22) requests specifically from the Bastion Host subnet (CIDR ***10.0.1.0/24***) for secure administrative access.



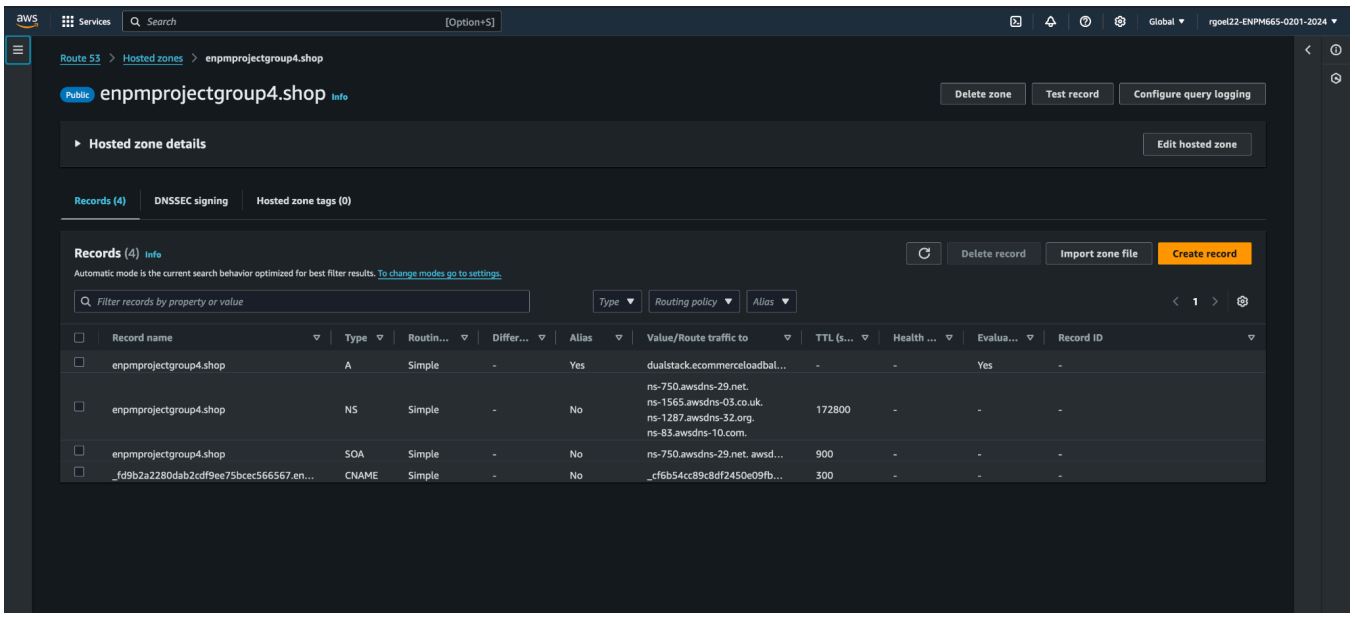
2.6.4 Bastion Host:

Bastion Host is used for secure remote management, hence only ***port 22*** (SSH) access is given to specific IPs which belong to the team members working on the project. In this way, even if there is any unauthorized SSH pem file access, the security group will still restrict access only to those four IPs.



2.7 Route 53:

Route53 is the service provided by AWS which allows us to manage DNS for our domain. This is achieved by simply pointing the domain to the AWS nameservers as mentioned in the hosted zone.
https://dnschecker.org/#NS/enpmprojectgroup4.shop



Additionally an A record (refer record 1) pointing to the Load Balancer and a CNAME record (refer record 4) needed to issue the SSL certificate for the domain.

3. CLOUD FORMATION TEMPLATES:

To ensure automated provisioning of infrastructure, consistent and repeatable deployment, we have made use of Cloud Formation features available at AWS. The cloud formation templates were deployed in the order in which there are listed below:

3.1 VPC Template:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: CloudFormation template to create a VPC with public and private subnets, route
tables, and a NAT gateway.

Resources:
 # VPC
 VPC:
   Type: AWS::EC2::VPC
   Properties:
     CidrBlock: 10.0.0.0/16
     EnableDnsSupport: true
     EnableDnsHostnames: true
     Tags:
       - Key: Name
         Value: "MyVPC"

 # Public Subnet 1
 PublicSubnet1:
   Type: AWS::EC2::Subnet
   Properties:
     VpcId: !Ref VPC
     CidrBlock: 10.0.1.0/24
     AvailabilityZone: !Select [ 0, !GetAZs '' ]
     MapPublicIpOnLaunch: true
     Tags:
       - Key: Name
         Value: "PublicSubnet1"

 # Public Subnet 2
 PublicSubnet2:
   Type: AWS::EC2::Subnet
   Properties:
     VpcId: !Ref VPC
     CidrBlock: 10.0.2.0/24
     AvailabilityZone: !Select [ 1, !GetAZs '' ]
     MapPublicIpOnLaunch: true
     Tags:
       - Key: Name
         Value: "PublicSubnet2"
```

```yaml
# Private Subnet 1
PrivateSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 10.0.3.0/24
    AvailabilityZone: !Select [ 0, !GetAZs '' ]
    Tags:
      - Key: Name
        Value: "PrivateSubnet1"

# Private Subnet 2
PrivateSubnet2:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
    CidrBlock: 10.0.4.0/24
    AvailabilityZone: !Select [ 1, !GetAZs '' ]
    Tags:
      - Key: Name
        Value: "PrivateSubnet2"

# Internet Gateway
InternetGateway:
  Type: AWS::EC2::InternetGateway
  Properties: {}

# Attach Internet Gateway to VPC
AttachGateway:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    VpcId: !Ref VPC
    InternetGatewayId: !Ref InternetGateway

# NAT Gateway EIP
NatEIP:
  Type: AWS::EC2::EIP
  Properties:
    Domain: vpc

# NAT Gateway
NatGateway:
  Type: AWS::EC2::NatGateway
  Properties:
    AllocationId: !GetAtt NatEIP.AllocationId
    SubnetId: !Ref PublicSubnet1  # Deploy in a public subnet for internet access

# Public Route Table
```

```yaml
PublicRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: "PublicRouteTable"

# Private Route Table
PrivateRouteTable:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: "PrivateRouteTable"

# Route for Public Subnet Internet Access
PublicRoute:
  Type: AWS::EC2::Route
  DependsOn: AttachGateway
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway

# Route for Private Subnet Internet Access via NAT
PrivateRoute:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PrivateRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    NatGatewayId: !Ref NatGateway

# Associate Public Subnet 1 with Public Route Table
PublicSubnet1RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet1
    RouteTableId: !Ref PublicRouteTable

# Associate Public Subnet 2 with Public Route Table
PublicSubnet2RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    SubnetId: !Ref PublicSubnet2
    RouteTableId: !Ref PublicRouteTable
```

```
  # Associate Private Subnet 1 with Private Route Table
  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PrivateSubnet1
      RouteTableId: !Ref PrivateRouteTable

  # Associate Private Subnet 2 with Private Route Table
  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PrivateSubnet2
      RouteTableId: !Ref PrivateRouteTable

Outputs:
  VPCId:
    Description: "VPC ID"
    Value: !Ref VPC
    Export:
      Name: VPCId

  PublicSubnet1:
    Description: "Public Subnet 1 ID"
    Value: !Ref PublicSubnet1
    Export:
      Name: PublicSubnet1

  PublicSubnet2:
    Description: "Public Subnet 2 ID"
    Value: !Ref PublicSubnet2
    Export:
      Name: PublicSubnet2

  PrivateSubnet1:
    Description: "Private Subnet 1 ID"
    Value: !Ref PrivateSubnet1
    Export:
      Name: PrivateSubnet1

  PrivateSubnet2:
    Description: "Private Subnet 2 ID"
    Value: !Ref PrivateSubnet2
    Export:
      Name: PrivateSubnet2

  NatGatewayId:
    Description: "NAT Gateway ID"
    Value: !Ref NatGateway
```

```
  Export:
    Name: NatGatewayId

 InternetGatewayId:
  Description: "Internet Gateway ID"
  Value: !Ref InternetGateway
  Export:
    Name: InternetGatewayId
```

3.2 Security Group Template:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: CloudFormation template for security groups for the VPC, including ALB, Auto
Scaling instances, RDS, and Bastion Host.

Resources:
 # Security Group for RDS (allows MySQL traffic on port 3306)
 RDSSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: "Allow MySQL access to RDS"
    VpcId: !ImportValue VPCId
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 3306
        ToPort: 3306
        CidrIp: 10.0.0.0/16  # Use the VPC CIDR block for internal traffic
    Tags:
      - Key: Name
        Value: "RDS Security Group"

 # Security Group for Auto Scaling instances (allow HTTP/HTTPS traffic and SSH from Bastion
Host)
 AutoscalingSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: "Allow HTTP, HTTPS, and SSH access to EC2 instances"
    VpcId: !ImportValue VPCId
    SecurityGroupIngress:
      # Allow HTTP traffic from ALB
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 10.0.0.0/16  # Internal traffic from ALB within the VPC
      # Allow HTTPS traffic from ALB
      - IpProtocol: tcp
        FromPort: 443
```

```
        ToPort: 443
        CidrIp: 10.0.0.0/16  # Internal HTTPS traffic from ALB within the VPC
      # Allow SSH traffic from the Bastion Host's subnet or private IP
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 10.0.1.0/24  # Replace this with the Bastion Host subnet CIDR or specific IP
    Tags:
      - Key: Name
        Value: "Autoscaling Security Group"

# Security Group for ALB (allow HTTP/HTTPS traffic)
ALBSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: "Allow HTTP and HTTPS traffic to the ALB"
    VpcId: !ImportValue VPCId
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 80
        ToPort: 80
        CidrIp: 0.0.0.0/0  # Allow public HTTP traffic from the internet
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0  # Allow public HTTPS traffic from the internet
    Tags:
      - Key: Name
        Value: "ALB Security Group"

# Security Group for Bastion Host (allow SSH access from the team's IPs)
BastionSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: "Allow SSH access to Bastion Host"
    VpcId: !ImportValue VPCId
    SecurityGroupIngress:
      # Your IP 1
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 50.237.13.57/32  # IP 1
      # Your IP 2
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 50.237.13.62/32  # IP 2
      # Teammate's IP 3
```

```
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 69.137.233.24/32  # IP 3
      # Teammate's IP 4
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 98.192.33.217/32  # IP 4
    Tags:
      - Key: Name
        Value: "Bastion Security Group"

Outputs:
 RDSSecurityGroupId:
   Description: "Security Group ID for RDS"
   Value: !Ref RDSSecurityGroup
   Export:
     Name: RDSSecurityGroupId

 AutoscalingSecurityGroupId:
   Description: "Security Group ID for Auto Scaling instances"
   Value: !Ref AutoscalingSecurityGroup
   Export:
     Name: AutoscalingSecurityGroupId

 ALBSecurityGroupId:
   Description: "Security Group ID for ALB"
   Value: !Ref ALBSecurityGroup
   Export:
     Name: ALBSecurityGroupId

 BastionSecurityGroupId:
   Description: "Security Group ID for Bastion Host"
   Value: !Ref BastionSecurityGroup
   Export:
     Name: BastionSecurityGroupId
```

3.3 RDS Template:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: CloudFormation template to create a highly available, encrypted MySQL RDS
instance with storage auto-scaling and default KMS encryption at rest.

Parameters:
 DBInstanceIdentifier:
   Description: "The name of the RDS instance"
```

```
    Type: String
    Default: "e-commerce"

  DBPassword:
    Description: "The password for the RDS instance master user."
    Type: String
    NoEcho: true
    MinLength: 8
    MaxLength: 41
    AllowedPattern: "[a-zA-Z0-9]*"
    ConstraintDescription: "Must contain only alphanumeric characters."

  DBAllocatedStorage:
    Description: "The allocated storage for the RDS instance in GB."
    Type: Number
    Default: 20

  DBInstanceClass:
    Description: "The compute and memory capacity of the DB instance."
    Type: String
    Default: db.t3.medium

Resources:
  # Create a DB Subnet Group for RDS (for Multi-AZ support)
  DBSubnetGroup:
    Type: AWS::RDS::DBSubnetGroup
    Properties:
      DBSubnetGroupDescription: "Subnet group for RDS instance in private subnets"
      SubnetIds:
        - !ImportValue PrivateSubnet1  # Use imported value from VPC template
        - !ImportValue PrivateSubnet2  # Use imported value from VPC template
      Tags:
        - Key: Name
          Value: "RDS Subnet Group"

  MyDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBInstanceIdentifier: !Ref DBInstanceIdentifier
      AllocatedStorage: !Ref DBAllocatedStorage
      MaxAllocatedStorage: 100
      DBInstanceClass: !Ref DBInstanceClass
      Engine: MySQL
      EngineVersion: '8.0'
      MasterUsername: admin
      MasterUserPassword: !Ref DBPassword
      DBName: ecommerce_1
      MultiAZ: true
```

```
      StorageType: gp2
      StorageEncrypted: true
      BackupRetentionPeriod: 7
      VPCSecurityGroups:
        - !ImportValue RDSSecurityGroupId  # Imported RDS Security Group
      DBSubnetGroupName: !Ref DBSubnetGroup
      PubliclyAccessible: false
    DeletionPolicy: Snapshot

Outputs:
 DBInstanceEndpoint:
    Description: "RDS MySQL Endpoint"
    Value: !GetAtt MyDBInstance.Endpoint.Address

 DBInstancePort:
    Description: "RDS MySQL Port"
    Value: !GetAtt MyDBInstance.Endpoint.Port
```

3.4 AutoScaling and Load Balancer Template:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: CloudFormation template for an E-commerce platform with autoscaling support for
instances in private subnets and a bastion host.

Parameters:

 KeyName:
    Type: AWS::EC2::KeyPair::KeyName
    Default: enpm818n
    Description: "The name of the EC2 Key Pair to enable SSH access"

 AMIId:
    Type: AWS::EC2::Image::Id
    Description: "The AMI ID for the EC2 instances"

Resources:
 # Bastion Host in Public Subnet
 BastionHost:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t3.micro
      KeyName: !Ref KeyName
      ImageId: ami-08c40ec9ead489470  # Ubuntu 20.04 LTS AMI ID for us-east-1, replace for your
region
```

```
    SubnetId: !ImportValue "PublicSubnet1"  # Bastion Host in Public Subnet 1
    SecurityGroupIds:
      - !ImportValue "BastionSecurityGroupId"  # Use the Bastion Security Group
    Tags:
      - Key: Name
        Value: "BastionHost"

# Application Load Balancer
EcommerceLoadBalancer:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Name: "EcommerceLoadBalancer"
    Subnets:
      - !ImportValue "PublicSubnet1"  # Import Public Subnet 1 from the VPC template
      - !ImportValue "PublicSubnet2"  # Import Public Subnet 2 from the VPC template
    SecurityGroups:
      - !ImportValue "ALBSecurityGroupId" # Import Load Balancer Security Group
    Scheme: "internet-facing"
    LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: '60'

# Target Group for Load Balancer
EcommerceTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Name: "EcommerceTargetGroup"
    VpcId: !ImportValue "VPCId"  # Ensure you import the correct VPC
    Port: 80
    Protocol: HTTP  # Specify the protocol
    TargetType: instance
    HealthCheckPath: "/"
    HealthCheckProtocol: HTTP
    HealthCheckIntervalSeconds: 30
    HealthyThresholdCount: 3
    UnhealthyThresholdCount: 3
    HealthCheckPort: "80"
    HealthCheckTimeoutSeconds: 5
    Matcher:
      HttpCode: "200"

# Listener for Load Balancer
EcommerceListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    LoadBalancerArn: !Ref EcommerceLoadBalancer
    Port: 80
    Protocol: HTTP
```

```
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref EcommerceTargetGroup

# Launch Template for Private Instances
PrivateLaunchTemplate:
  Type: AWS::EC2::LaunchTemplate
  Properties:
    LaunchTemplateName: "EcommercePrivateLaunchTemplate"
    LaunchTemplateData:
      ImageId: !Ref AMIId  # Use the AMI parameter
      InstanceType: t3.micro
      SecurityGroupIds:
        - !ImportValue "AutoscalingSecurityGroupId"  # Import Auto Scaling Security Group
      KeyName: !Ref KeyName  # Shared key name
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          set -e  # Exit on any error

          # Go to the application directory
          cd /var/www/html
          sudo su
          git checkout -- .
          git pull origin main

          # Ensure permissions are correct for the web server
          chown -R www-data:www-data /var/www/html

          # Restart Apache to apply the latest code
          systemctl restart apache2

      TagSpecifications:
        - ResourceType: instance
          Tags:
            - Key: Name
              Value: "EcommercePrivateInstance"

# Auto Scaling Group for Private Instances
PrivateAutoScalingGroup:
  Type: AWS::AutoScaling::AutoScalingGroup
  DependsOn: EcommerceLoadBalancer
  Properties:
    VPCZoneIdentifier:
      - !ImportValue "PrivateSubnet1"
      - !ImportValue "PrivateSubnet2"
    LaunchTemplate:
      LaunchTemplateId: !Ref PrivateLaunchTemplate
```

```
      Version: !GetAtt PrivateLaunchTemplate.LatestVersionNumber
    MinSize: 1
    MaxSize: 5
    DesiredCapacity: 2
    TargetGroupARNs:
      - !Ref EcommerceTargetGroup
    HealthCheckType: ELB
    HealthCheckGracePeriod: 300

# Scale-Up Policy for Private Instances
PrivateScaleUpPolicy:
  Type: AWS::AutoScaling::ScalingPolicy
  DependsOn: PrivateAutoScalingGroup  # Ensure Auto Scaling Group is created first
  Properties:
    AutoScalingGroupName: !Ref PrivateAutoScalingGroup
    PolicyType: StepScaling
    StepAdjustments:
    - MetricIntervalLowerBound: 0
      ScalingAdjustment: 1
    AdjustmentType: ChangeInCapacity
    Cooldown: 120
    EstimatedInstanceWarmup: 120

# Scale-Down Policy for Private Instances
PrivateScaleDownPolicy:
  Type: AWS::AutoScaling::ScalingPolicy
  DependsOn: PrivateAutoScalingGroup
  Properties:
    AutoScalingGroupName: !Ref PrivateAutoScalingGroup
    PolicyType: StepScaling
    StepAdjustments:
    - MetricIntervalUpperBound: 0
      ScalingAdjustment: -1
    AdjustmentType: ChangeInCapacity
    Cooldown: 120
    EstimatedInstanceWarmup: 120

# Scale-Up Alarm (>= 30% CPU)
ScaleUpAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmDescription: "Scale up when CPU Utilization is >= 50% for 50 seconds"
    MetricName: "CPUUtilization"
    Namespace: "AWS/EC2"
    Statistic: "Average"
    Period: 60
    EvaluationPeriods: 1
    Threshold: 50
```

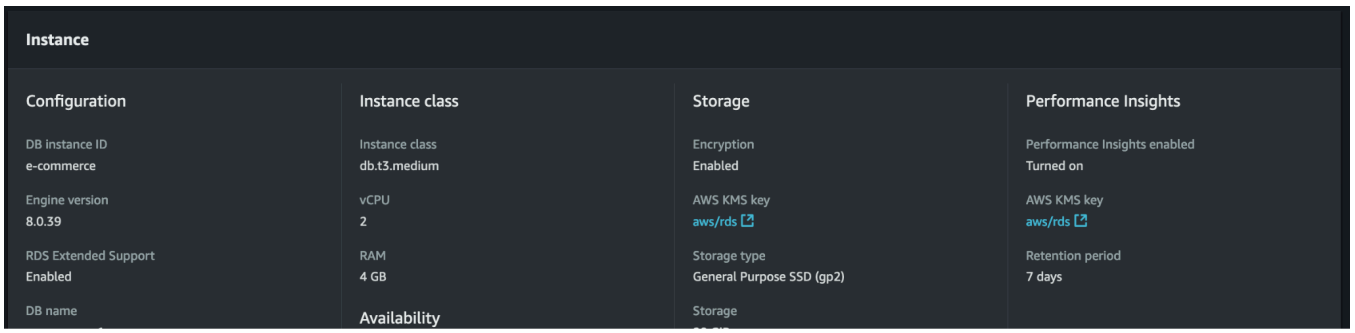
```
    ComparisonOperator: "GreaterThanOrEqualToThreshold"
    AlarmActions:
      - !Ref PrivateScaleUpPolicy
    Dimensions:
      - Name: "AutoScalingGroupName"
        Value: !Ref PrivateAutoScalingGroup

# Scale-Down Alarm (<= 10% CPU)
ScaleDownAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmDescription: "Scale down when CPU Utilization is <= 10% for 60 seconds"
    MetricName: "CPUUtilization"
    Namespace: "AWS/EC2"
    Statistic: "Average"
    Period: 60
    EvaluationPeriods: 1
    Threshold: 10
    ComparisonOperator: "LessThanOrEqualToThreshold"
    AlarmActions:
      - !Ref PrivateScaleDownPolicy
    Dimensions:
      - Name: "AutoScalingGroupName"
        Value: !Ref PrivateAutoScalingGroup

Outputs:
 PrivateAutoScalingGroupName:
   Description: "Name of the Private Auto Scaling Group"
   Value: !Ref PrivateAutoScalingGroup

 LoadBalancerDNSName:
   Description: "DNS Name of the Load Balancer"
   Value: !GetAtt EcommerceLoadBalancer.DNSName

 BastionHostPublicIP:
   Description: "Public IP of the Bastion Host"
   Value: !GetAtt BastionHost.PublicIp
```

All these templates can also be easily accessible on our github repository:
https://github.com/tan-vi-k/enpm818NyamlFiles.git

4. SECURITY MEASURES:

4.1 Amazon Key Management Service (KMS):

Amazon KMS is a fully managed service provided by AWS, which makes it easy to manage and use. We have implemented it to enhance the security of our database infrastructure. This provides robust encryption capabilities, ensuring the confidentiality and integrity of our sensitive data. The advantage of using the KMS service is that it is highly scalable, so if the instances need to be scaled up or down, it automatically does it for us. Also, KMS integrates with AWS CloudTrail, providing a comprehensive audit trail of all key usage for monitoring and compliance purposes.

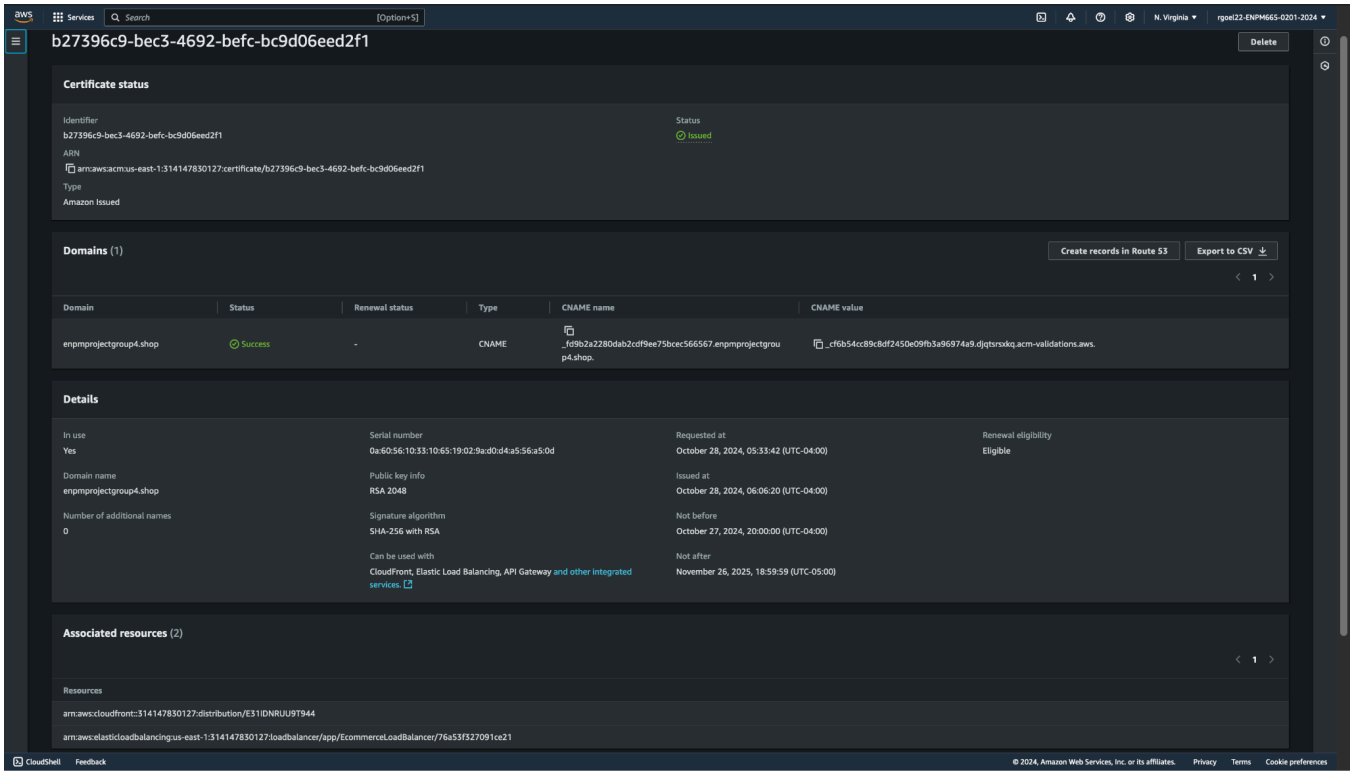


Using KMS, the data encryption is satisfied at rest.

4.2 SSL for domain:

As per the workflow of our architecture, once the our domain enpmprojectgroup4.shop is requested, Route53 directs a request through Cloudfront and WAF, and sends it to the Load Balancer. As our website is an e-commerce site, sensitive data needs to be transferred to and fro, such as login details, order details, payment details, etc, so it is important for the data in transit to be encrypted.

To achieve this, we have enabled a SSL certificate from AWS Certificate Manager (ACM) on our domain, which is also associated with the Load balancer and CloudFront services that we have used.

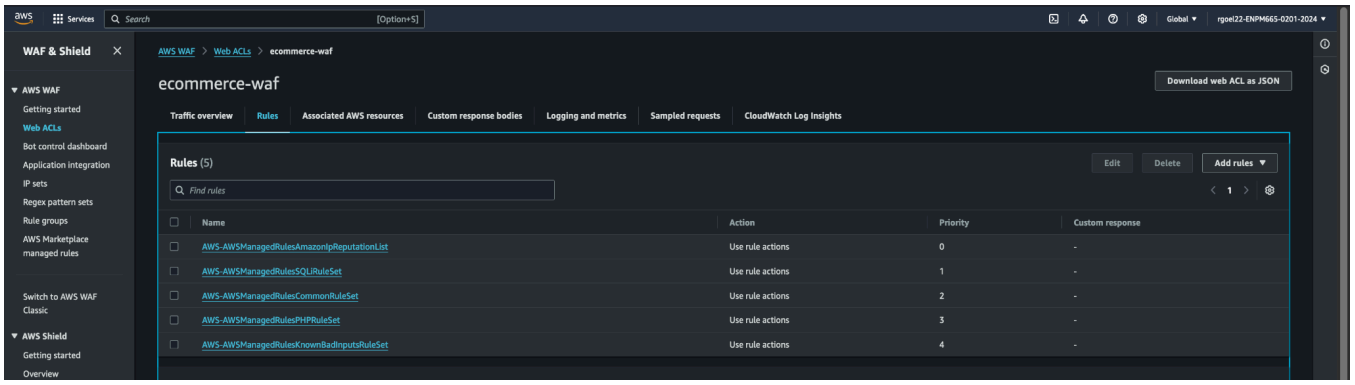


4.3 SSL between database and web connections:

The image below shows the connect.php file in which an SSL certificate, located at ***/var/www/html/global-bundle.pem***, is specified to authenticate the connection. The ***mysqli_ssl_set*** function is used to enforce SSL, ensuring that all communication between the web server and the RDS instance is encrypted.

```
 GNU nano 7.2                                                                                                  connec
<?php
// Database connection parameters
$host = 'e-commerce.cchcvgordfrx.us-east-1.rds.amazonaws.com'; // RDS endpoint
$user = 'admin'; // Database admin user
$password = 'rootenpm818n'; // Admin password
$dbname = 'ecommerce_1'; // Database name

// Create connection

$ssl_cert_path = '/var/www/html/global-bundle.pem';
$con = new mysqli($host, $user, $password, $dbname);
$con->ssl_set(NULL, NULL, $ssl_cert_path, NULL, NULL);


// Check connection and provide detailed error reporting
if ($con->connect_error) {
    die("Connection failed: " . $con->connect_error . "<br>Error Number: " . $con->connect_errno);
}

?>
```

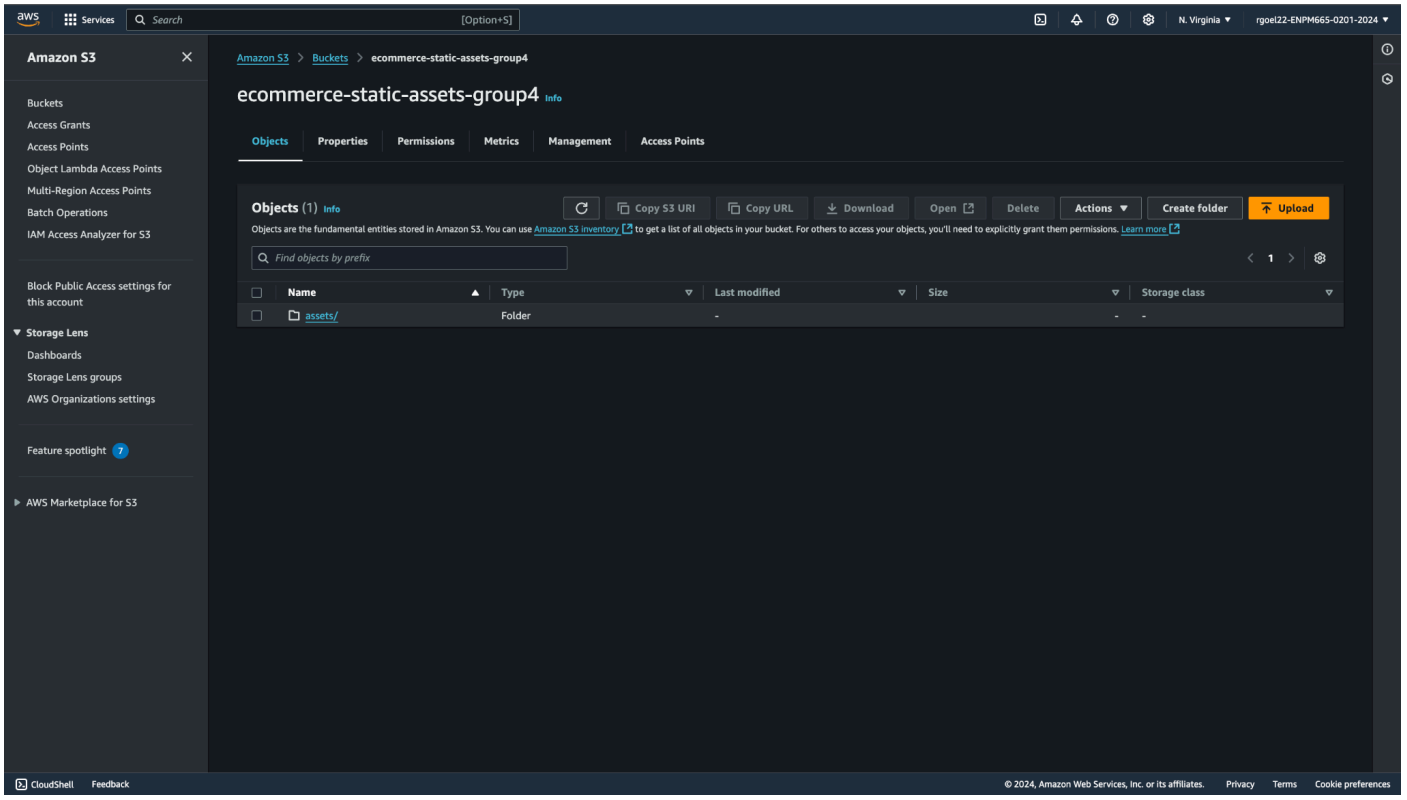
4.4 WAF rules configured:

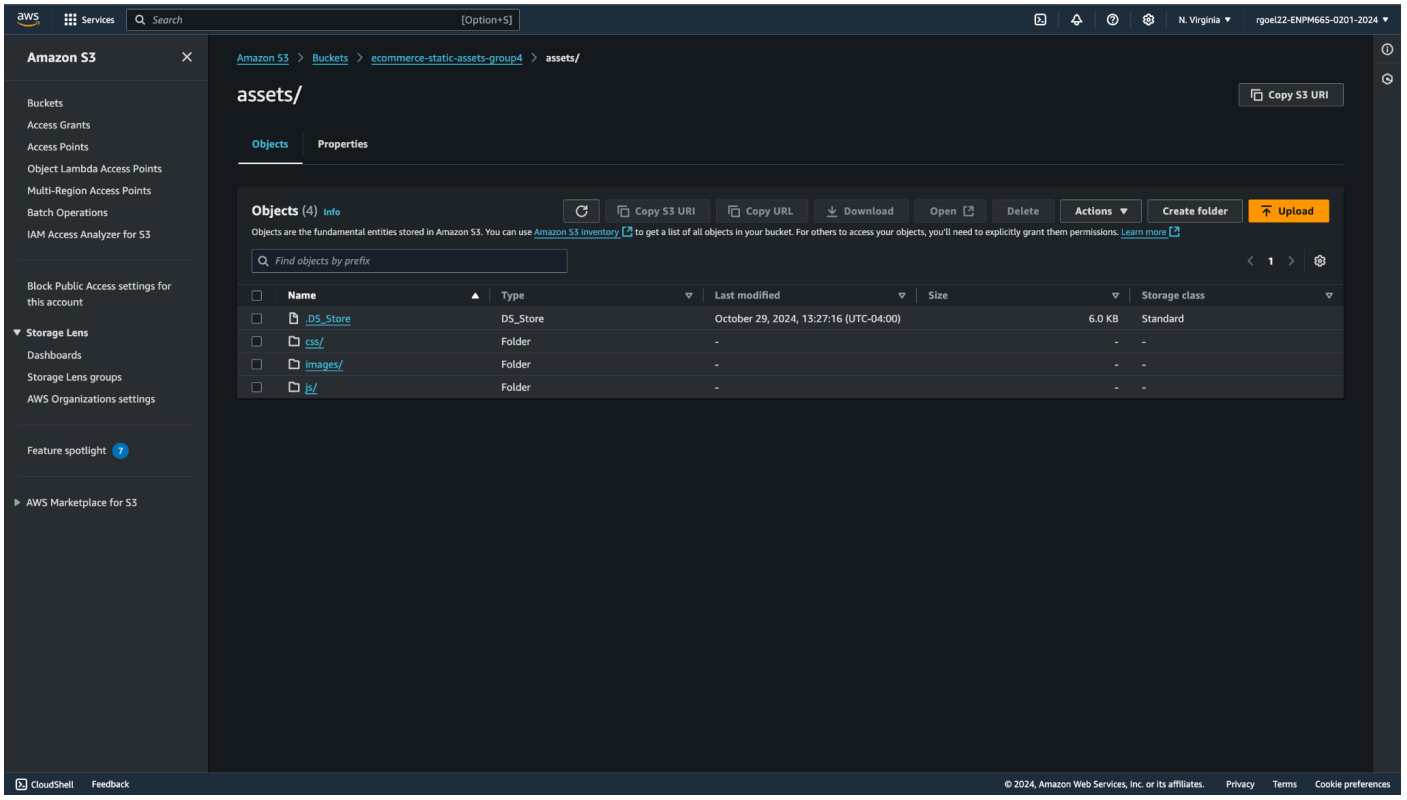


The WAF configuration for the "***ecommerce-waf***" includes a comprehensive set of managed rule groups, such as the ***AWSManagedRulesAmazonIpReputationList***, ***SQLiRuleSet***, ***CommonRuleSet***, ***PHPRuleSet***, and ***KnownBadInputsRuleSet***. With a Web ACL capacity usage of ***1225*** out of the allowable 5000 WCUs, the setup is well within limits, optimizing security without compromising performance. The default action is set to "***Allow***" for requests that do not match any specific rules, maintaining a balanced security approach while permitting necessary traffic.

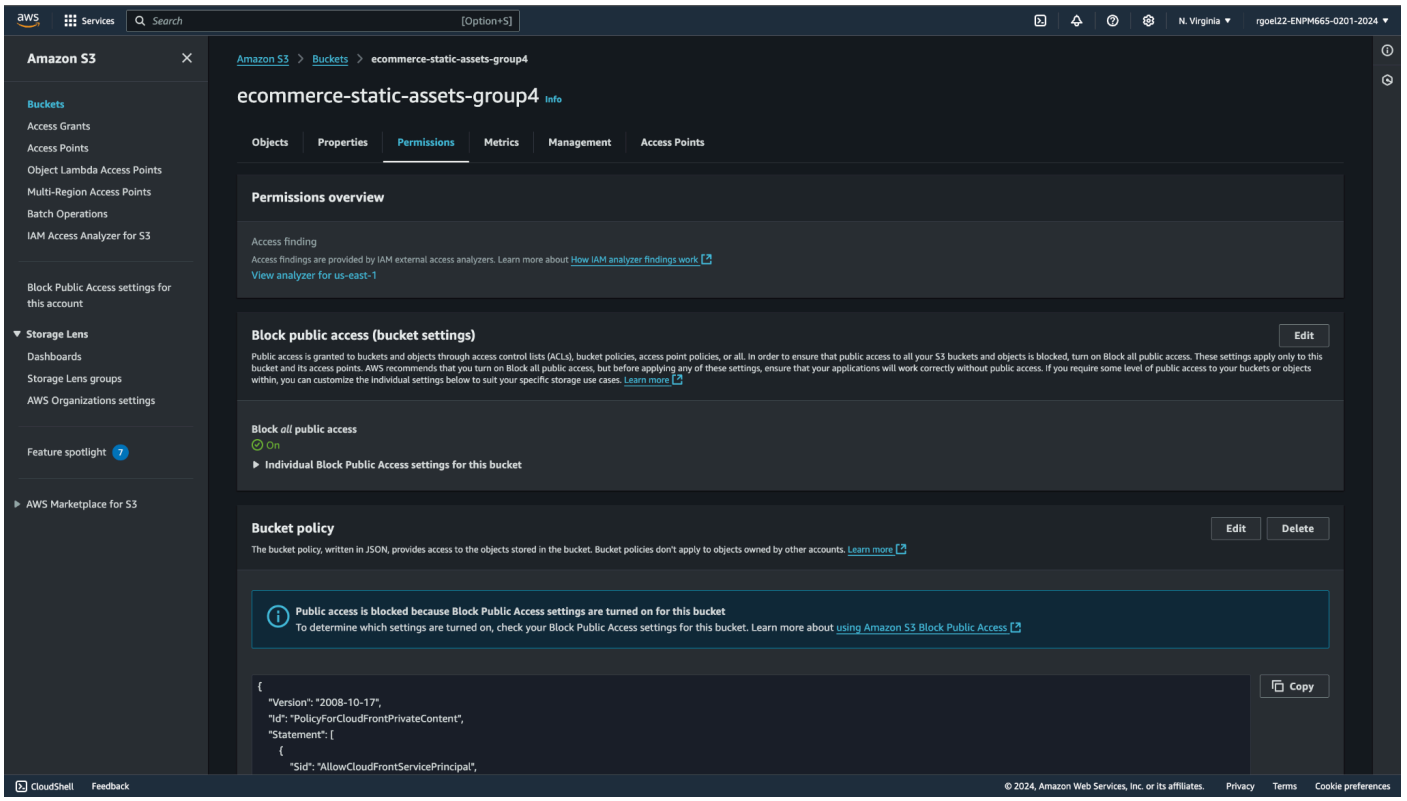
5. OPTIMIZATION MEASURES:

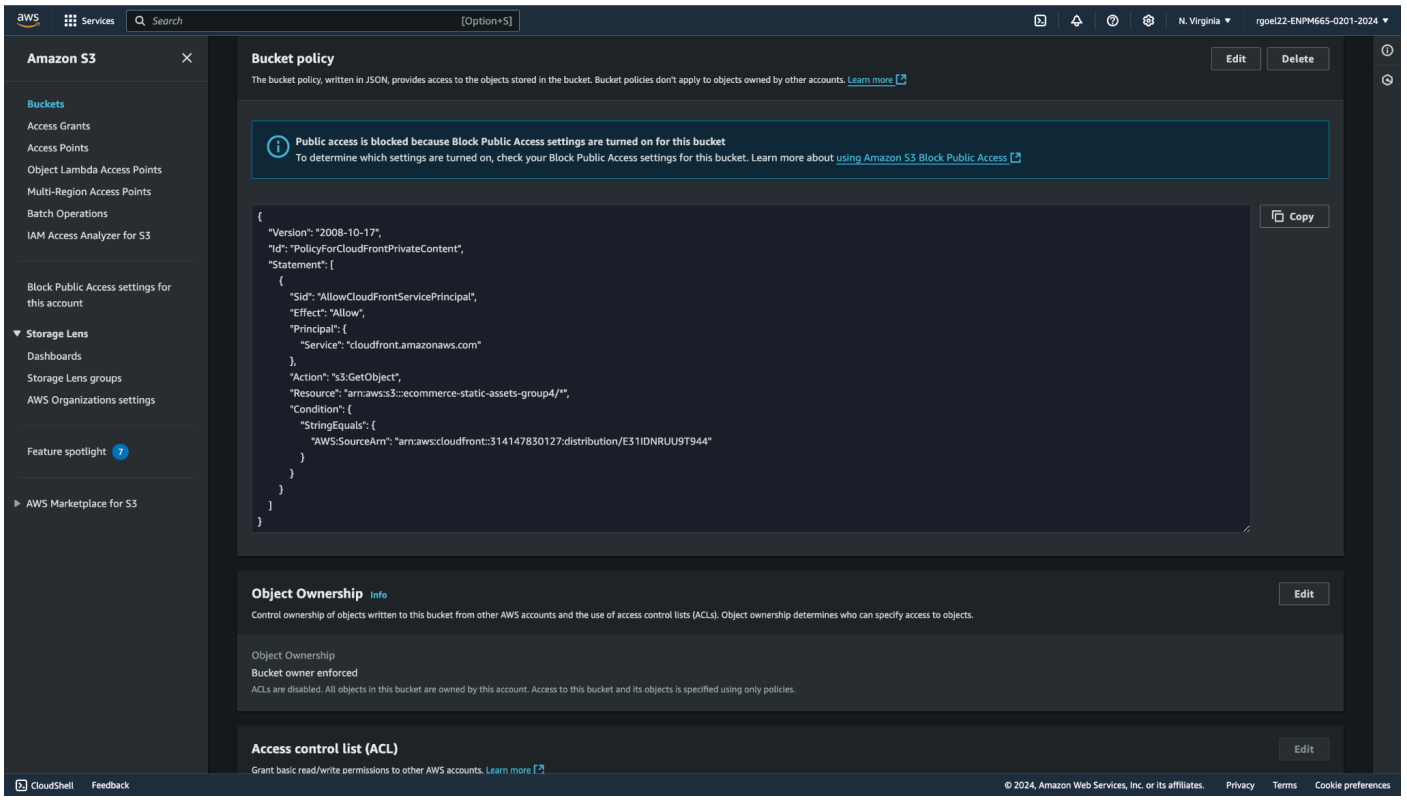
5.1 CloudFront:



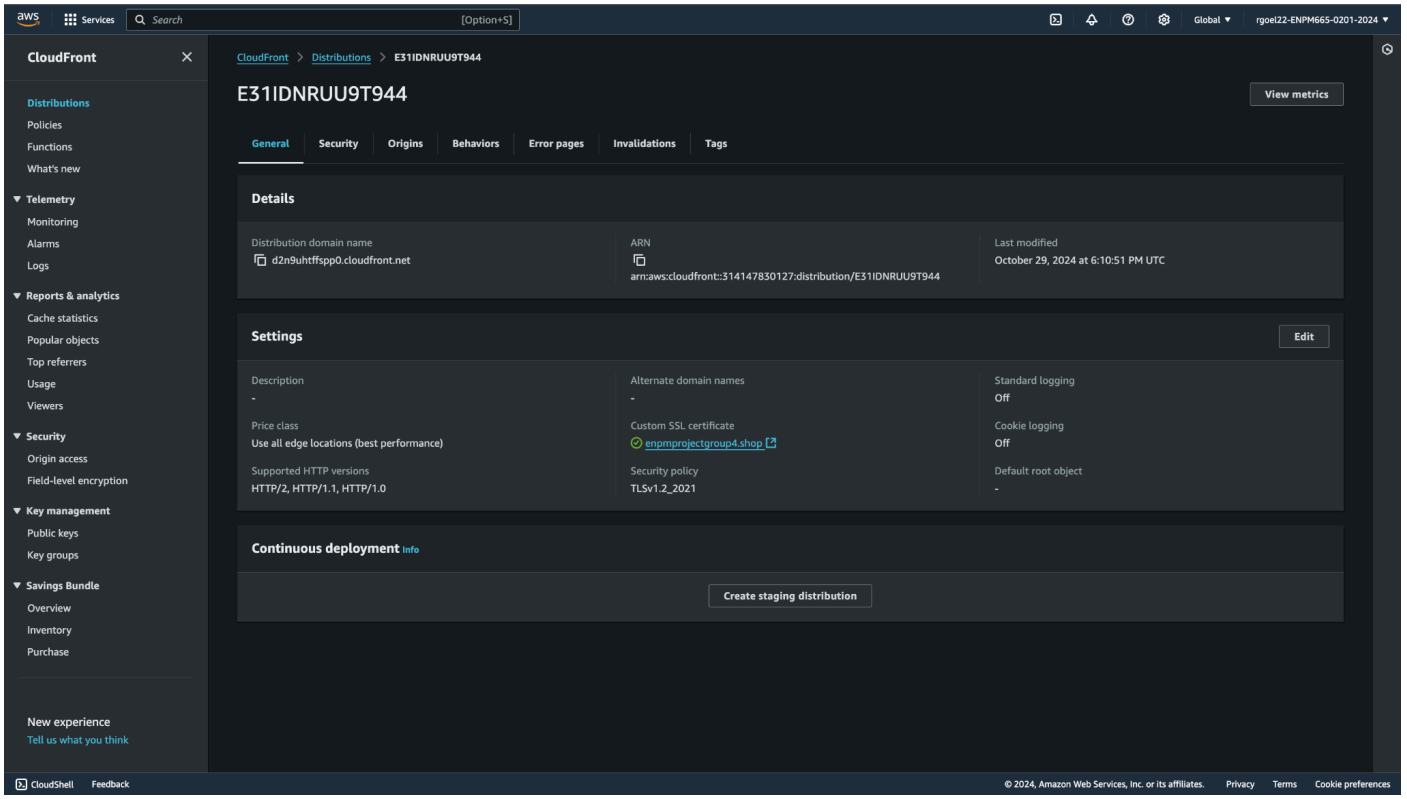


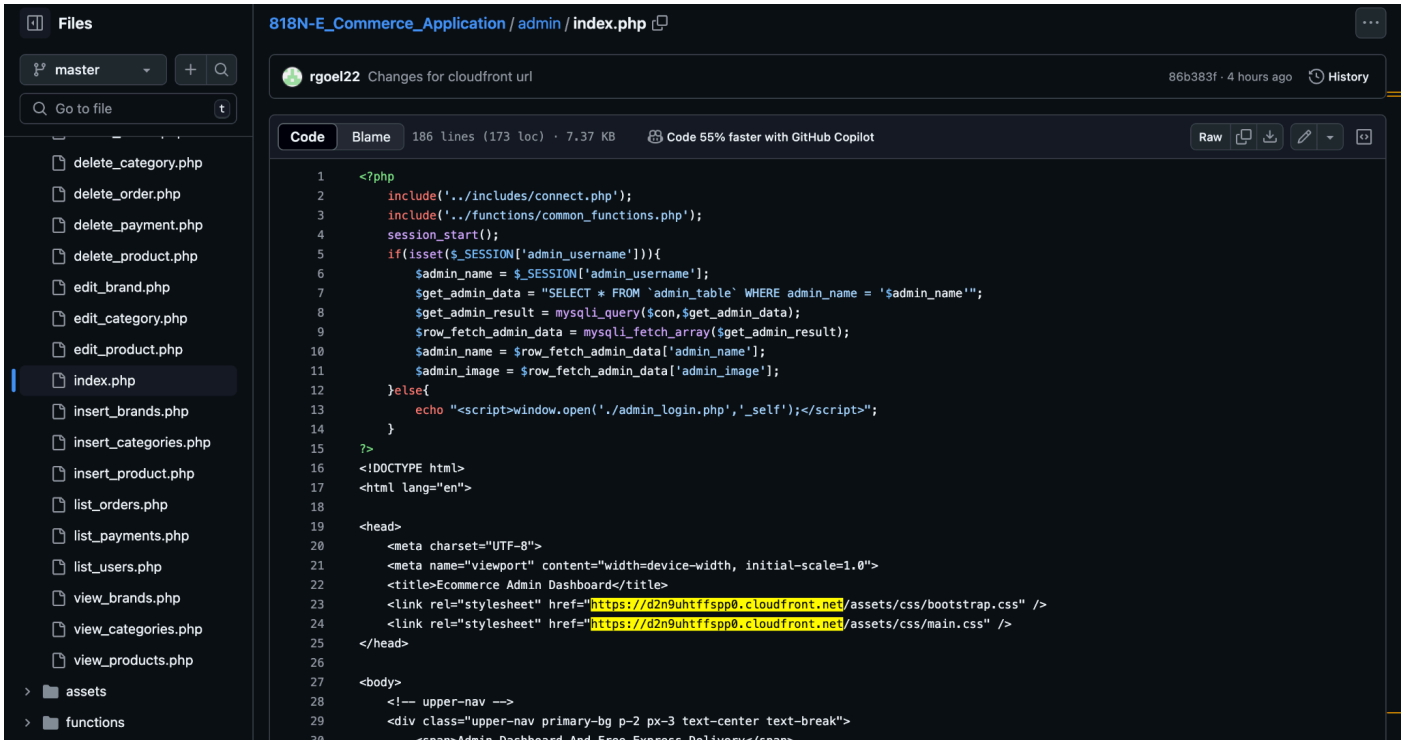
The S3 bucket, named ***ecommerce-static-assets-group4***, was configured to serve as the origin for CloudFront, providing a centralized storage for files like CSS, JavaScript, and images. By organizing assets within designated folders—css, js, and images—this structure ensures easier file management and retrieval, as seen in the organization under the "***assets***" directory in the S3 configuration.





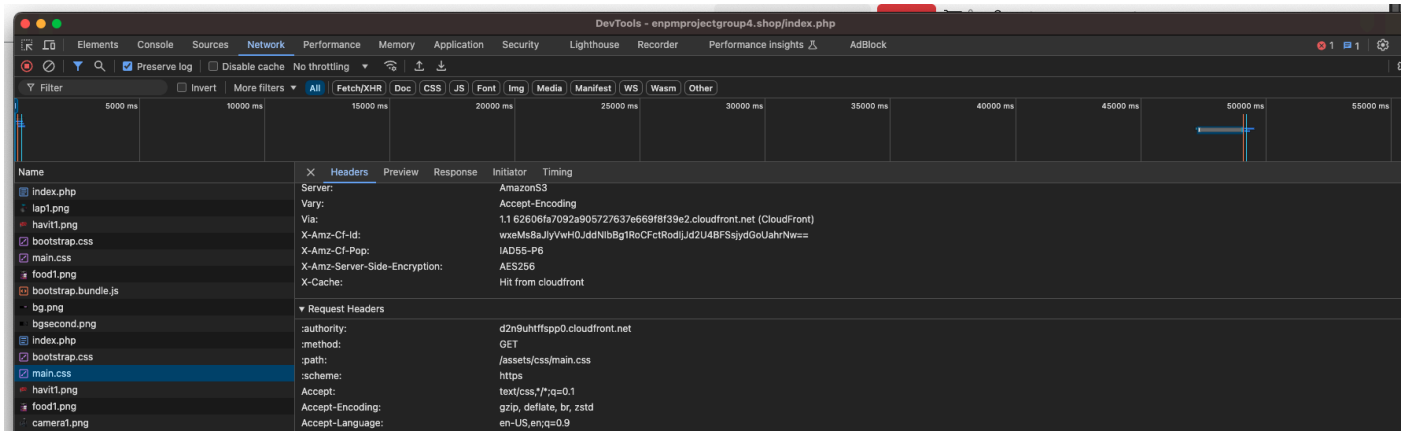
The permissions configuration for the ***ecommerce-static-assets-group4*** S3 bucket prioritizes security and controlled access, with public access fully blocked to prevent unauthorized access. To enable secure integration with CloudFront, a specific bucket policy was applied, allowing only the designated CloudFront distribution to retrieve objects from the bucket. This policy authorizes the CloudFront service principal (***cloudfront.amazonaws.com***) to perform the ***s3:GetObject*** action, restricted by a condition that ensures access is granted exclusively through the specified CloudFront distribution.





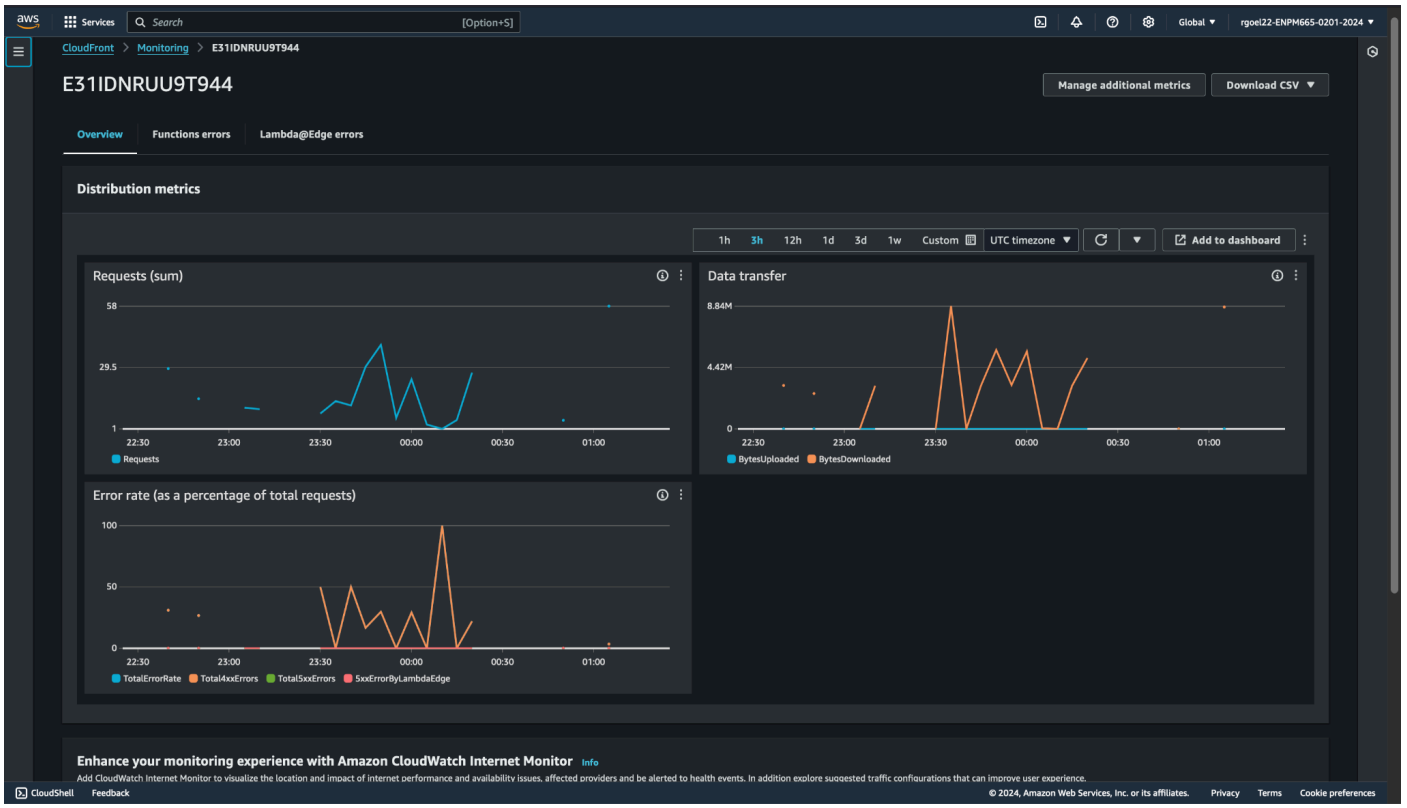
CloudFront distribution (***E31IDNRUU9T944***) was created to cache assets across its global network, enhancing load times by delivering content from the edge location closest to the user. The CloudFront URL (***d2n9uhtffspp0.cloudfront.net***) was then integrated in the code, replacing direct links to S3.

5.2 Gzip compression:



The image displays the effective deployment of GZIP compression on static assets in our web application, focusing on the ***main.css*** file distributed via CloudFront with compression activated. As visible, ***Content-Encoding: gzip*** response header, CloudFront delivers the CSS file with GZIP compression, which is vital for optimizing web performance.

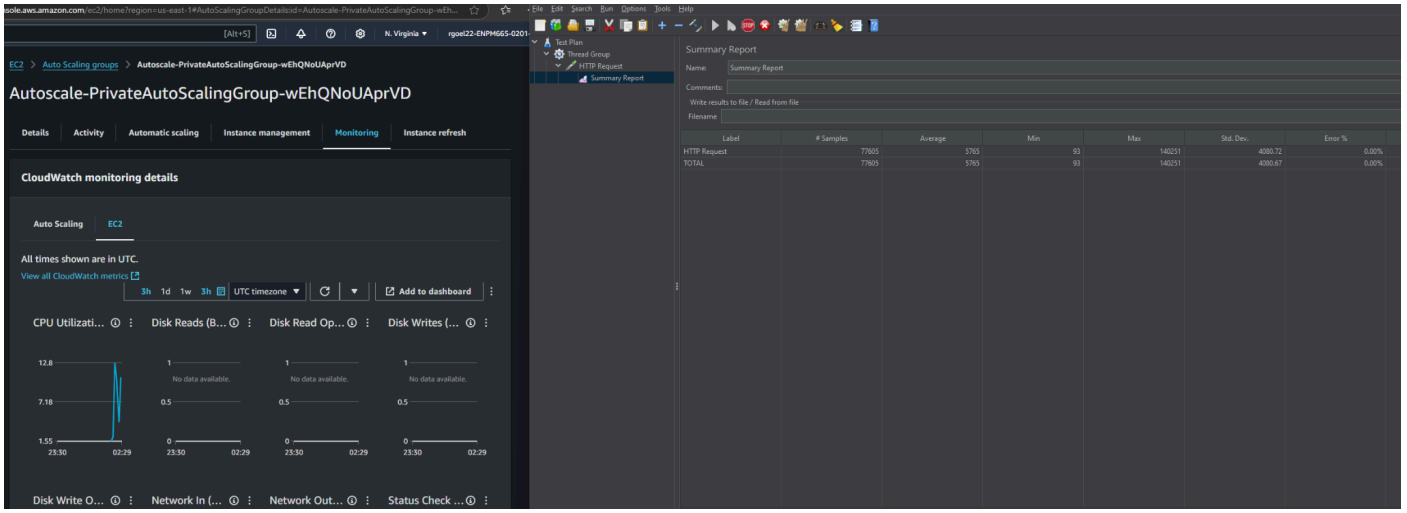
5.3 Cloudwatch Logs:

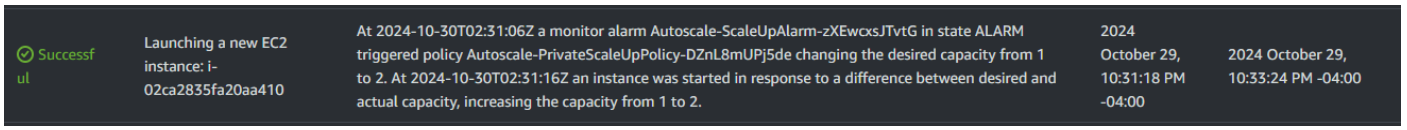


6. TESTING AND MONITORING:

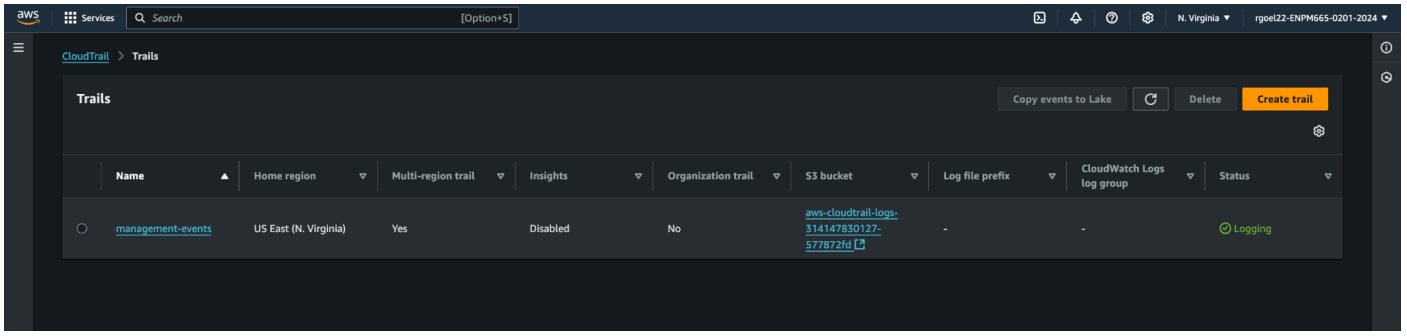
6.1 Stress Testing:

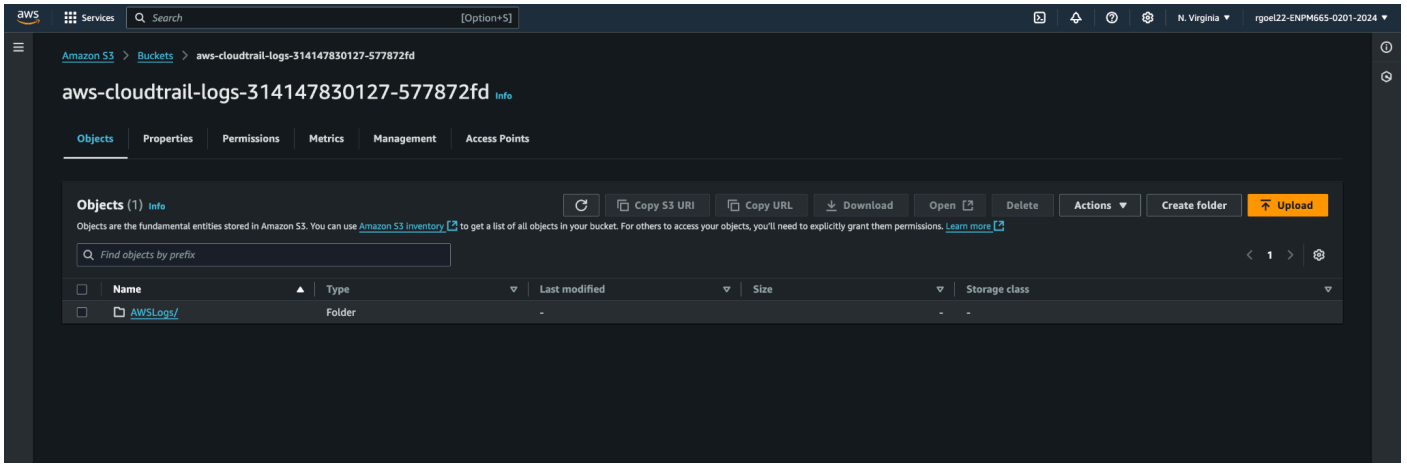
The stress testing was simulated using Apache JMeter (5.6.3). To see auto scaling in action, we needed to temporarily dial down the upscale CPU utilization threshold to 10 % from the production's 50%. A total of 1000 individual users/threads with a ramp-up period of 20 secs accomplished that.

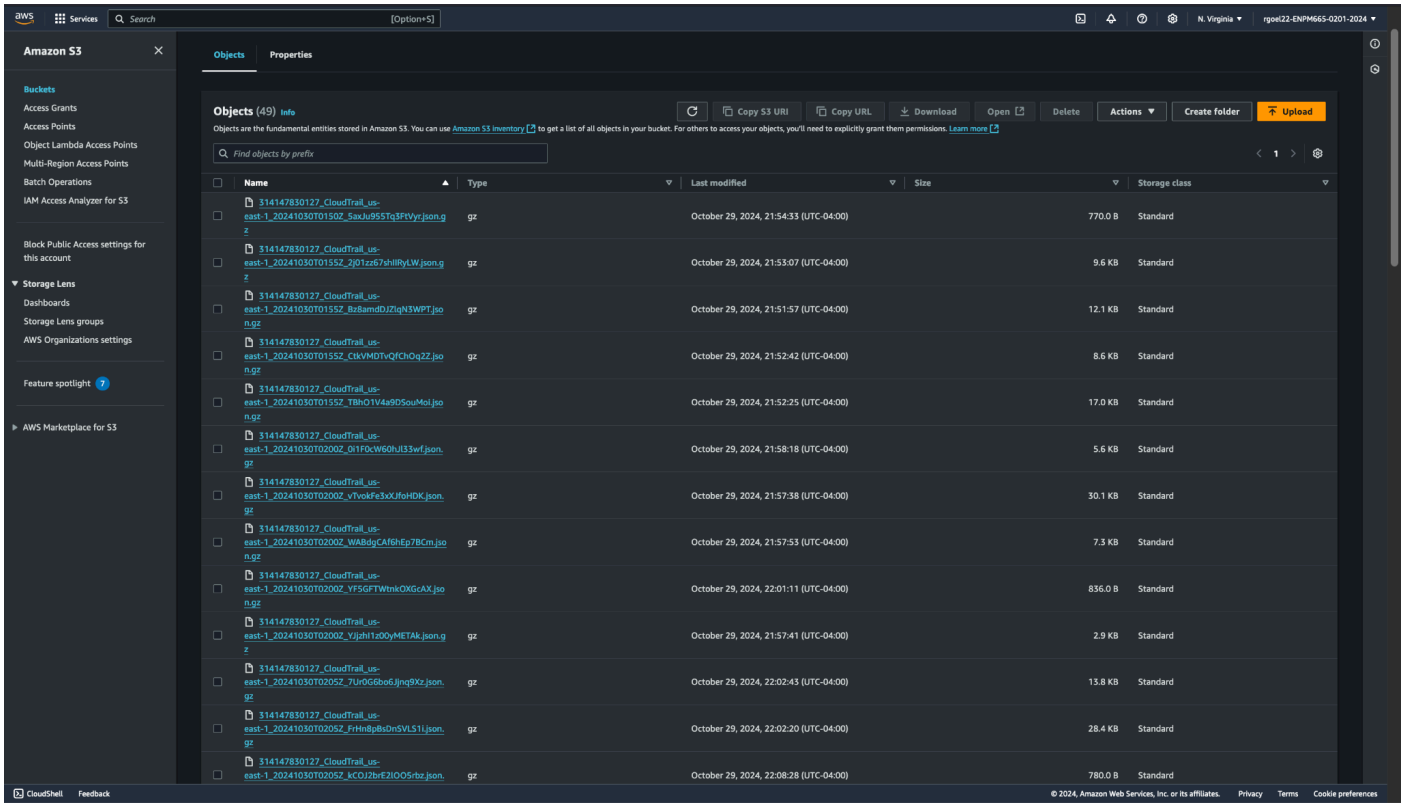




6.2 Monitoring:



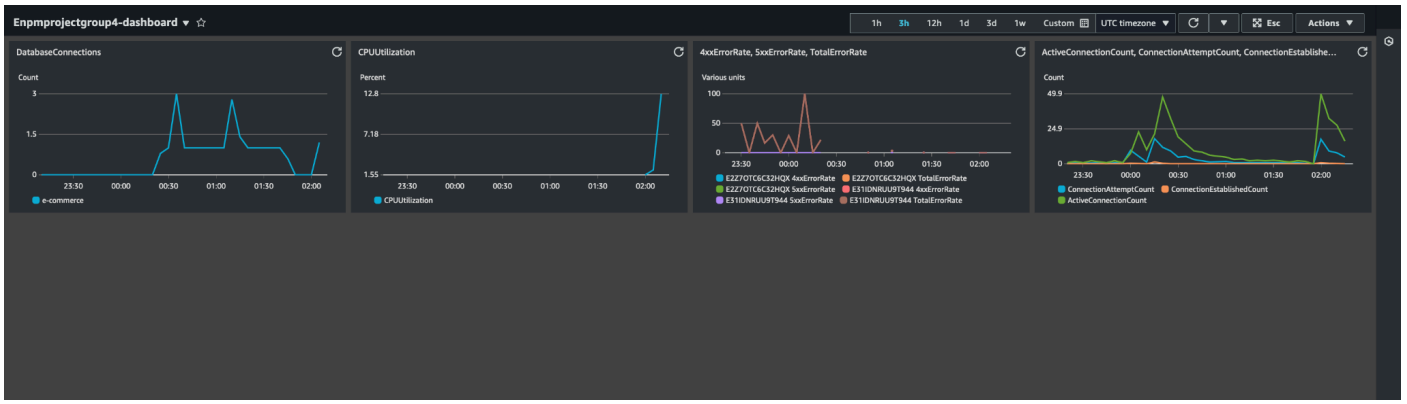




The CloudTrail, includes a "***management-events***" trail to log security and management events across all regions. The logs are stored in the S3 bucket "***aws-cloudtrail-logs-314147830127-577872fd***", as shown in the above images. This setup centralizes event records securely, ensuring that all critical activities are logged and stored for future analysis and auditing.

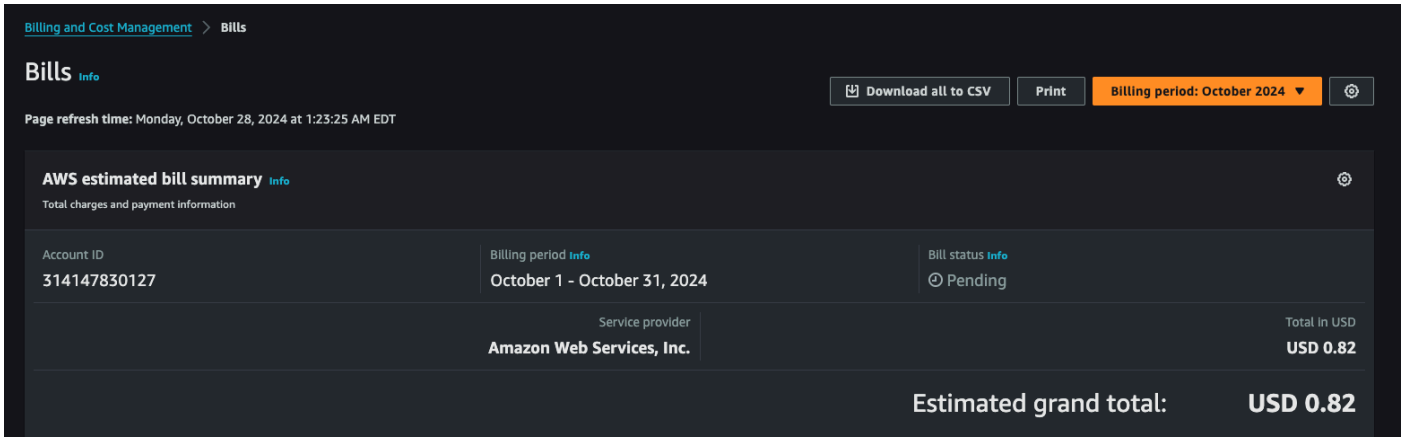
6.3 Cloudwatch Dashboard:

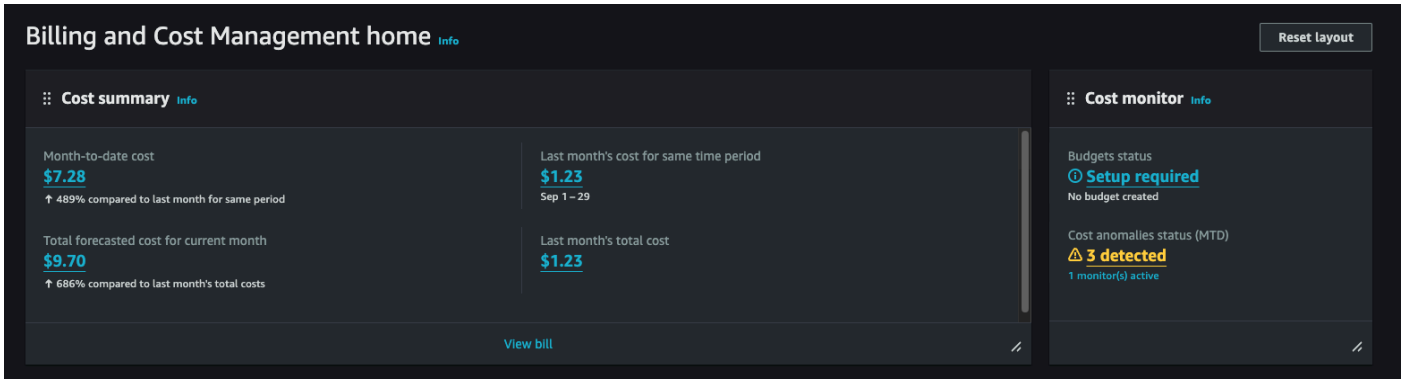
The following is the custom dashboard created to monitor database connections, CPU utilization, error rates and active connections to the website.

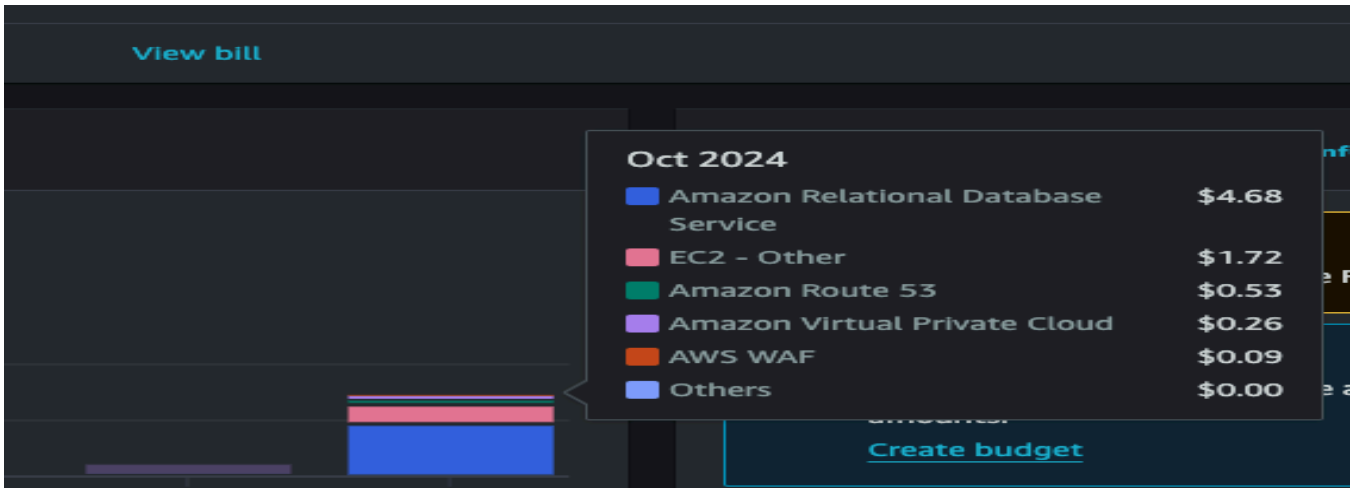


7. COST ANALYSIS:

Given that the increase in AWS costs is due to recent usage for the assignment, it's crucial to focus on optimizing future expenses. The current month-to-date cost is $7.28, with a forecasted total of $9.70, compared to last month's minimal usage of $1.23. The primary costs stem from Amazon RDS and EC2 services. To manage and potentially reduce costs, we recommend adopting Reserved Instances (RIs) or Compute Savings Plans, which can provide up to 72-75% savings for long-term, consistent usage. Specifically, for Amazon RDS, using RDS Reserved Instances or Savings Plans would significantly lower ongoing database costs. Additionally, optimizing VPC usage by reviewing and eliminating unnecessary resources like unused Elastic IPs or NAT gateways can further reduce costs. Regularly tracking AWS usage and costs using AWS Cost Explorer will help identify any unexpected spikes and allow for timely resource adjustments. These measures will ensure efficient use of AWS services while minimizing long-term expenses.







8. CONCLUSION:

In conclusion, our project successfully met its core objectives of designing and deploying a secure, scalable, and high-performance e-commerce platform on AWS. By leveraging AWS's cloud infrastructure and managed services, including Auto Scaling, Load Balancers, RDS, WAF, and CloudFront, we created an architecture that supports fluctuating user demands, optimizes performance, and secures sensitive data. The implementation of encryption for data at rest and in transit, along with strict access controls through security groups and WAF rules, has ensured a high level of data security and compliance with industry standards. CloudFront's CDN capabilities further enhanced user experience by providing low-latency content delivery across global locations.

Our cost analysis highlights the value of AWS's Reserved Instances and Savings Plans for long-term cost efficiency. The adoption of these financial strategies, along with continuous monitoring using AWS Cost Explorer, will help manage the platform's operational costs effectively. Rigorous testing through tools like Apache JMeter validated our system's scalability and resilience under various load conditions, affirming that the platform can adapt to real-world traffic variations while maintaining stable performance.

This project not only achieved its technical goals but also served as a practical demonstration of AWS best practices for cloud-based e-commerce platforms. The foundational architecture and configurations developed here provide a strong basis for future enhancements, including the potential integration of advanced analytics, AI-driven insights, and additional security layers. This report showcases the thoughtful design and diligent implementation process undertaken to ensure both immediate functionality and long-term sustainability of our e-commerce solution on AWS.

9. RESOURCES:

[1] https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html

[2] https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html

[3] https://docs.aws.amazon.com/waf/latest/developerguide/what-is-aws-waf.html

[4] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html

[5] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/best-practices-security.html

[6] https://docs.aws.amazon.com/config/latest/developerguide/WhatIsConfig.html

[7] https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/ServingCompressedFiles.html

[8] https://aws.amazon.com/premiumsupport/support-cloud-cost-optimization/