ENPM 818N - Cloud Computing

Midterm Group Project

Scalable and Secure E-Commerce Platform on AWS

https://group18ecommerce.com

Submitted By – Group 18

Name	Directory ID	University ID	Section
Aryan Kulshrestha	ak17	120342035	0201
Aashay Mehta	amehta01	119577028	0201
Long Le	lle38	116223775	0201
Gideon Marcus Jayakumar	gideonmm	121182067	0201

INDEX

Introduction

The aim of our project is to set up an e-commerce platform using AWS. We ran the web application on EC2 instances and used RDS MYSQL database to store the data. We also added auto scaling to make the system handle changes in traffic. We used AWS WAF and added SSL encryption for security. To make the system load faster we used CloudFront. To keep track of everything we used Cloud Watch. WE also ran some load tests and looked into ways to reduce costs by using reserved instances and savings plan towards the end. Overall, the idea was to build something that's similar to how a real cloud-based app would work.

1.1 Project Overview

The goal of the project was to deploy a fully functional e-commerce platform on AWS. It contains feature like hosting the web app on EC2 Instances, storing data in an RDS MySQL database and using auto scaling to handle the traffic. Later security measures like AWS WAF and SSL encryption were added. We also used CloudFront to speed up the content delivery and monitored everything using CloudWatch. Finally, we tested the system under load and looked into cost saving options like reserved instances and savings plan.

1.2 Objective

The main objective of this project was to see how a complete web application can be built and managed using AWS tools. We focused on making the system scalable, secure and cost effective. Through this we learned how different AWS services connect to support a web app and how to monitor and optimize it. In terms of scope, this project can be extended by adding features like CI/CD pipelines, serverless components (like Lambda), better user authentication, or even containerizing the app using docker and deploying it on ECs and EKs. There's also scope to build analytics or recommendation systems on top of the platform in the future.

Architectural Diagram

To represent the infrastructure designed and deployed for our cloud-based eCommerce application, the following architecture diagram captures the complete flow from user access to backend data management. It integrates core AWS services that ensure availability, scalability, performance, and security. This diagram reflects our implementation across various phases, showing how services like EC2, RDS, CloudFront, and WAF interact to deliver a seamless and secure user experience.

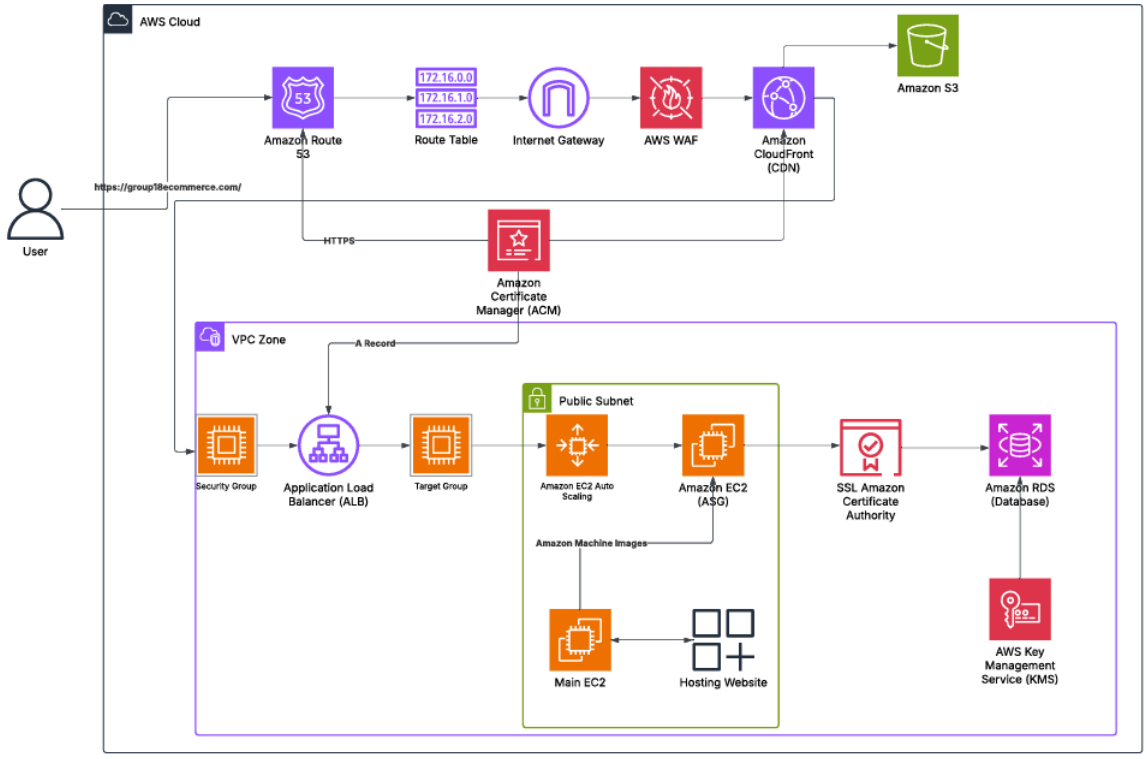

Figure 1: Main Architectural Diagram of the eCommerce Platform

Lucid Chart Link: https://lucid.app/lucidchart/b4230840-95e0-4576-9085-9e3fb15404ee/edit?invitationId=inv_115c4f0f-be50-498d-aa1a-f47c983b31b2&page=0_0#

Overall Workflow:

1. Domain Routing and Content Delivery: The application begins at Amazon Route 53, which manages domain resolution for the custom DNS (e.g., group18ecommerce.com). AWS CloudFront acts as a CDN, caching static content from Amazon S3 to improve load times and global accessibility. An SSL certificate from AWS Certificate Manager (ACM) ensures secure HTTPS communication with users.

2. Security and Load Balancing: Incoming traffic is filtered through AWS WAF to block malicious requests such as SQL injection and cross-site scripting. Requests are then routed to an Application Load Balancer (ALB), which distributes traffic to backend EC2 instances. The ALB is integrated with ACM to serve traffic over HTTPS securely.

3. Compute and Auto Scaling: The main application is hosted on an EC2 instance (Midterm-EC2) deployed in a public subnet of the default VPC. To ensure high availability under variable load, an Auto Scaling Group (ASG) is configured using a custom AMI of the midterm EC2. This allows AWS to automatically scale in or out based on CPU utilization or traffic patterns.

4. Database Layer and Encryption: Amazon RDS (MySQL) serves as the backend database. SSL/TLS encryption is enabled to secure data in transit. The RDS instance is publicly accessible with tightly controlled Security Groups to allow connections only from the EC2 instance. This setup ensures secure and efficient communication between the application and the database.

5. Monitoring, Logging, and Cost Management: Amazon CloudWatch is used to monitor system metrics like CPU utilization and memory usage. It also captures logs for performance tuning and debugging. AWS CloudTrail provides visibility into user activity and service API calls. AWS Cost Explorer helps track resource usage and spending, while JMeter was used to stress test the infrastructure under simulated load conditions.

Phase 1: Infrastructure Setup

This phase lays down the foundation for our cloud-based E-Commerce web application. We started by setting up the core compute and database resources in AWS. The infrastructure is designed to ensure modularity, scalability, and security. Service like EC2 for hosting the application, RDS for managing the database layer, and Application Load Balancer for distributing traffics were deployed. Other resources like VPC, subnet, route tables, internet gateway, and security groups were configured to enable secure communication between components and control traffic flow across the layers of the infrastructure.

3.1 Architecture Overview:

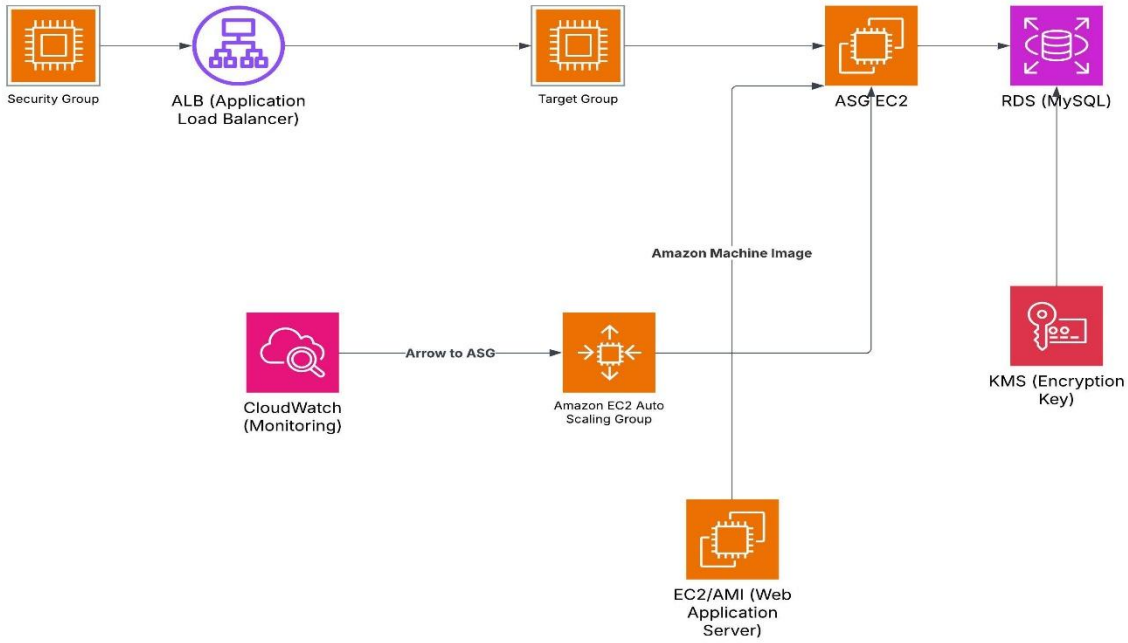

Figure 2: High-Level Architecture of Phase 1

Lucid Chart Link: https://lucid.app/lucidchart/b6ecad76-6d34-43d0-8eb4-6f5f5398d04f/edit?invitationId=inv_0d133ed2-c055-4e0c-a968-18a24569f5a1&page=0_0#

This architecture visually represents the setup:

- The Application Load Balancer (ALB) acts as the entry point, routing user requests to EC2 instances based on availability and health.
- EC2 instances within an Auto Scaling Group ensure scalability and high availability for the application tier.

- The Amazon RDS (MySQL) database is decoupled from the application layer to enhance performance and manageability.
- CloudWatch is configured to monitor metrics like CPU usage, and Auto Scaling Policies respond to traffic changes dynamically.

3.2 EC2 Instance and Load Balancer Configuration

The EC2 instance serves as the primary compute environment to host and run the eCommerce web application. It allows us to deploy the codebase, install required software, and serve traffic to users. To ensure high availability, fault tolerance, and even distribution of traffic, the EC2 instance was integrated with an Application Load Balancer (ALB). The ALB acts as the single point of entry, intelligently routing user requests across one or more backend instances, and supporting both HTTP and HTTPS traffic with automatic redirection and SSL offloading.

EC2 Instance Creation:

- An EC2 instance was launched using the Ubuntu 22.04 LTS AMI. The EC2 instances, configured as t2.micro, are the virtual servers where the e-commerce application backend runs. The t2.micro instance type is cost-effective and provides sufficient CPU and memory for small-to-intermediate medium tasks.
- Security groups were configured to allow inbound HTTP (80), HTTPS (443), and SSH (22) traffic.
- Apache and PHP were installed on the EC2 instance to host the web application.
- The GitHub repository containing the eCommerce web application was cloned onto the EC2.
- The EC2 was provisioned in a public subnet within a default VPC with internet access enabled via an internet gateway.

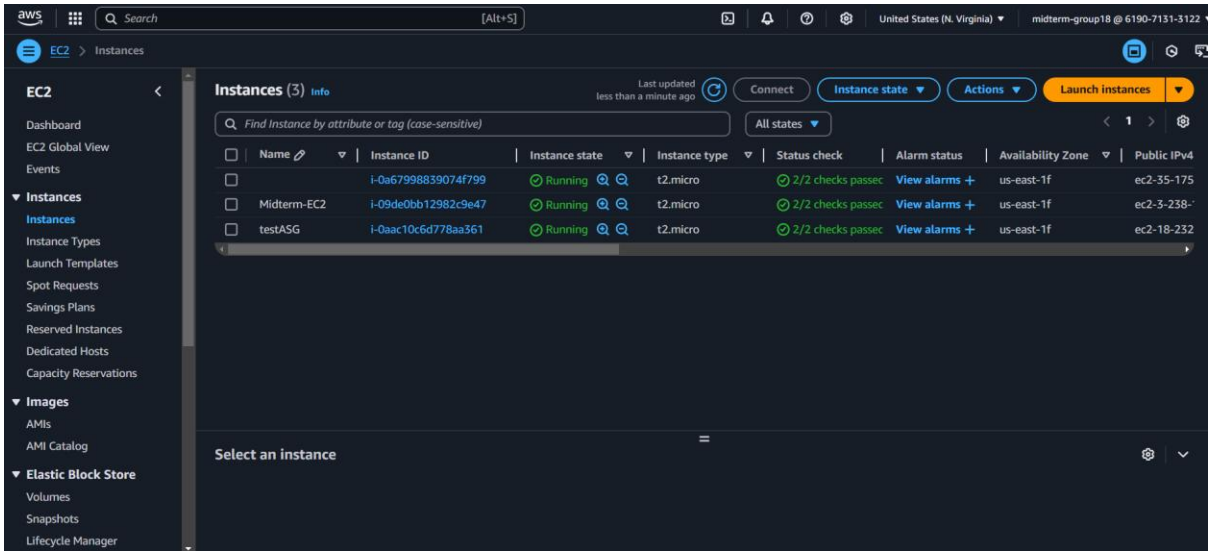

Figure 3: EC2 Instances

Application Load Balancer (ALB):

- An ALB was created to distribute traffic across instances and ensure high availability.
- Two listeners were configured:
  - Port 80 (HTTP) with a redirect rule to HTTPS (443).
  - Port 443 (HTTPS) with SSL certificate from AWS Certificate Manager.
- Target group is created and the EC2 instance is registered to receive traffic. The ALB organizes EC2 instances into target groups, which determine where traffic is sent. This configuration allows the ALB to manage multiple instances properly, allowing the redirection of traffic based on instance health and capacity.

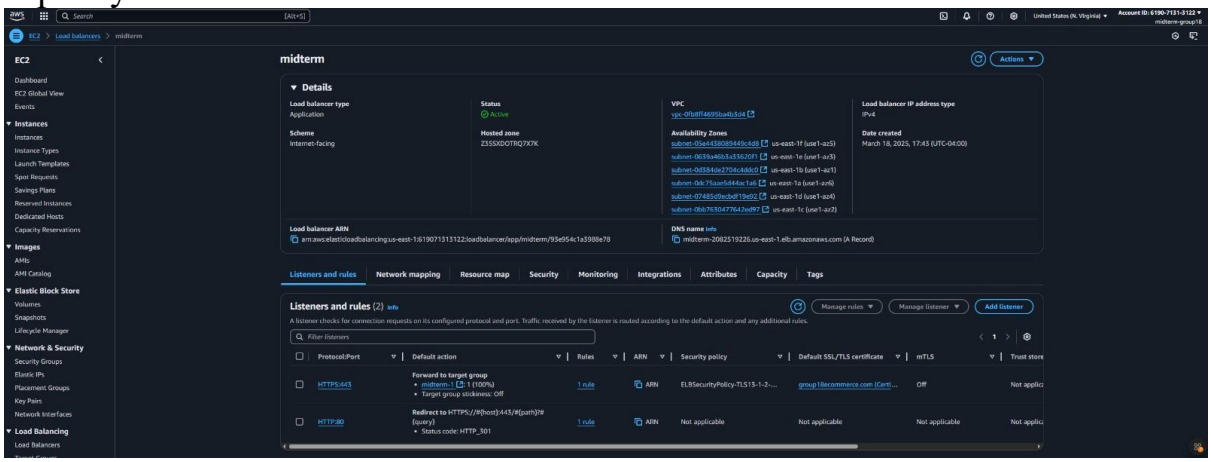

Figure 4: Application Load Balancers (ALB)

3.3 RDS Setup for MySQL and Database Encryption

Amazon RDS was used to decouple the database layer from the application layer, enabling better manageability, scalability, and performance. It eliminates the need to manage database servers directly on EC2 and provides built-in features like automated backups, encryption, and monitoring.

RDS Instance Configuration:

- Amazon RDS was used to launch a MySQL database instance.
- The database was configured with publicly accessible set to "Yes" for initial connectivity.
- Security group for RDS allowed inbound traffic on port 3306 from the EC2 instance.
- Database credentials and endpoint were added in the ‘connect.php’ file of the web app.
- The RDS instance was launched in a private subnet for added security and accessed only through the application.

Encryption:

- RDS storage encryption was enabled using AWS-managed KMS keys.
- For SSL in-transit encryption (Phase 2.2b), an Amazon RDS SSL certificate was optionally used, and the database connection was modified with MYSQLI_CLIENT_SSL ` (later disabled for debugging).

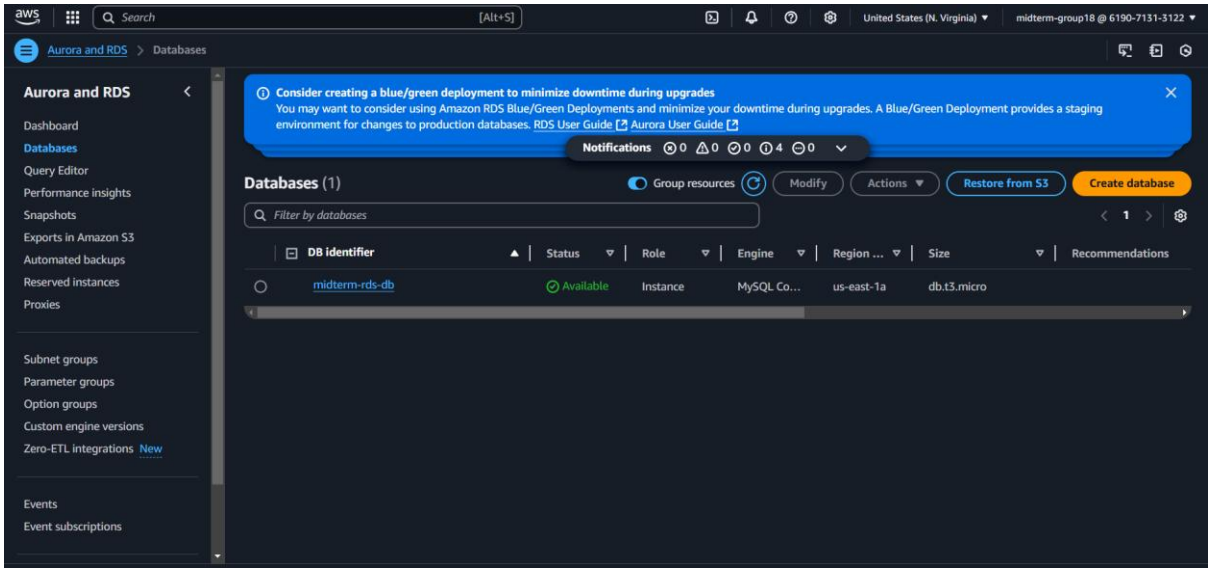

Figure 5: RDS database

3.4 Auto Scaling Policies and Cloud Watch Alarm

To ensure the web application can handle varying levels of incoming traffic, Auto Scaling was configured using a Amazon Machine Image (AMI) and tied to monitoring metrics through Amazon CloudWatch. This approach ensures the application remains highly available and responsive under stress, while also optimizing infrastructure costs during periods of low activity.

AMI Creation:

- An Amazon Machine Image (AMI) was created from the fully configured and working EC2 instance (Midterm-EC2), that hosted the deployed eCommerce web application.
- This AMI serves as a base image to launch identical instances automatically when scaling is triggered.

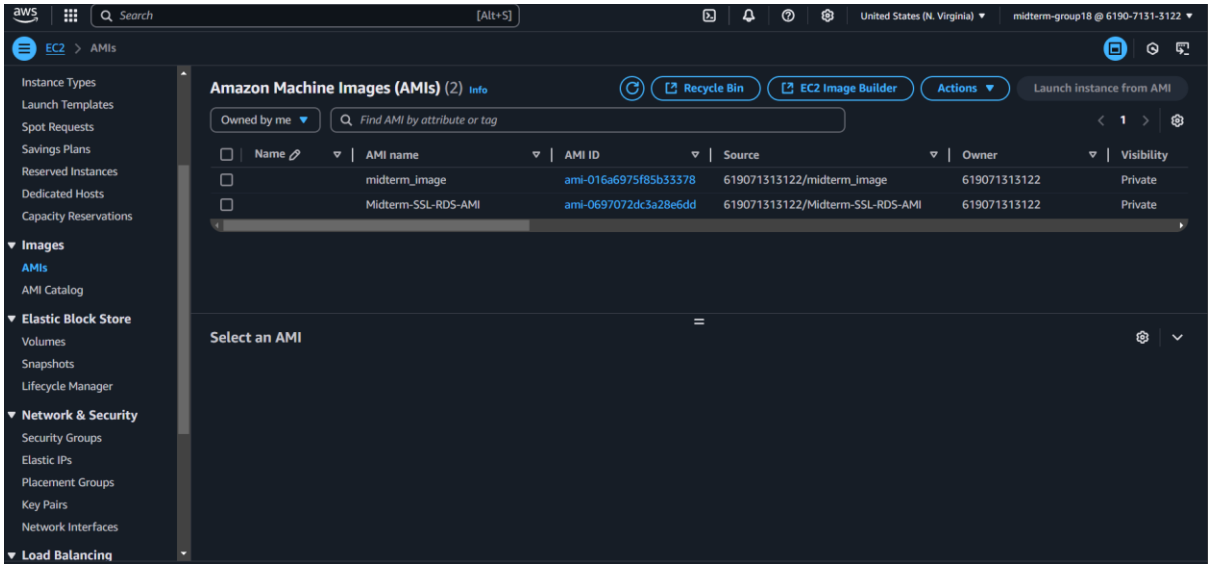

Figure 6: Amazon Machine Image (AMI)

Auto Scaling Group (ASG):

- An Auto Scaling Group was configured using the AMI, with defined minimum, desired, and maximum capacity values to manage the number of running EC2 instances based on demand.
- The ASG was associated with the Application Load Balancer (ALB), ensuring that all incoming user traffic is evenly distributed across the active EC2 instances, thereby supporting fault tolerance and high availability.

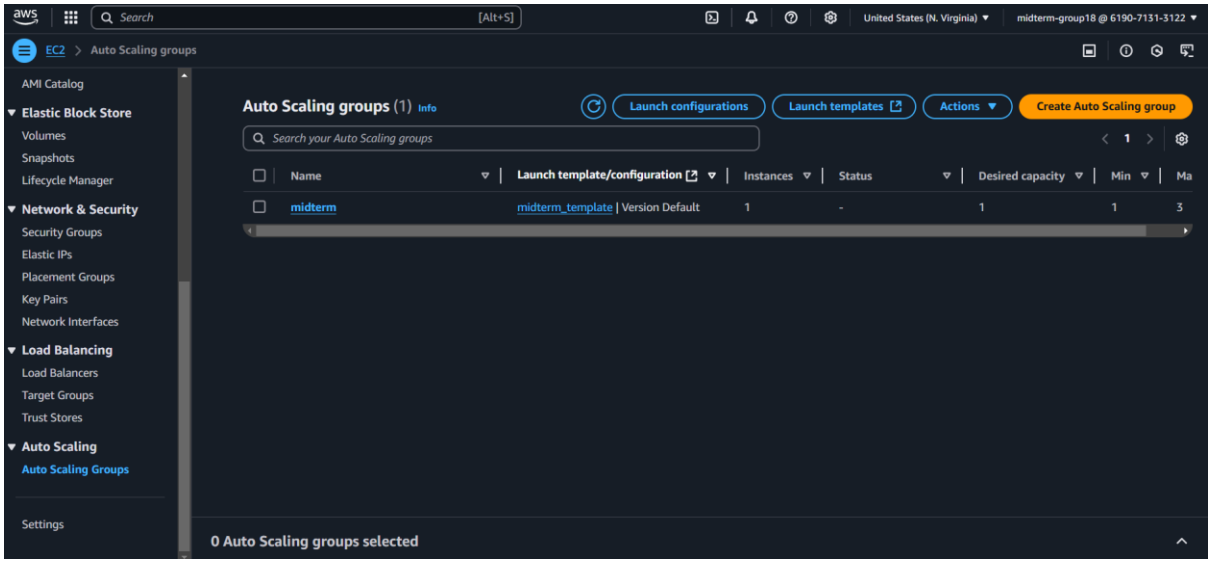

Figure 7: Auto Scaling Group (ASG)

CloudWatch Alarm and Scaling Policy:

- Amazon CloudWatch was used to create an alarm that monitors CPU utilization across instances within the Auto Scaling Group.
- When the average CPU usage exceeds the defined threshold of 50%, the scaling policy automatically triggers the launch of additional EC2 instances to handle the increased load.
- Conversely, when the traffic subsides and CPU utilization drops below a safe threshold, the scaling policy terminates unnecessary instances to optimize costs.
- These dynamic adjustments allow the application to scale in and out based on real-time performance metrics.

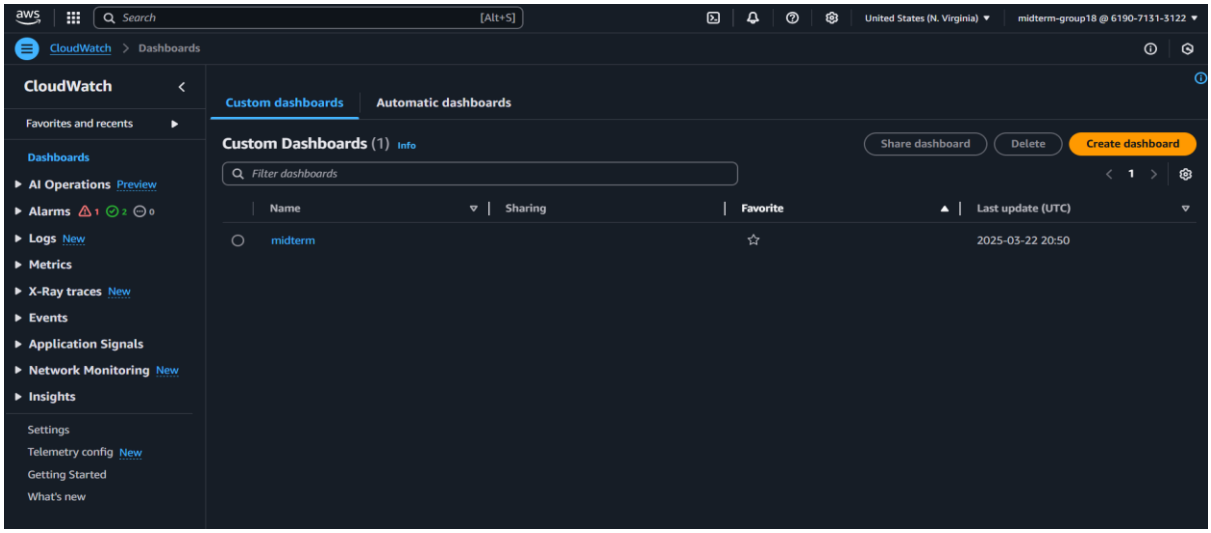

Figure 8: CloudWatch

Phase 2: Securing the Application

The purpose of this phase is to enhance the security of the E-commerce Web Application by implementing Web Application Firewall (WAF) with Application Load Balancer (ALB) to prevent SQL injection and XSS. Furthermore, encrypt data in transit by registering and applying SSL/TLS certificates through AWS Certificate Manager (ACM) for HTTPS communication and also enable SSL for database connections between web application and RDS.

4.1 Architecture Overview

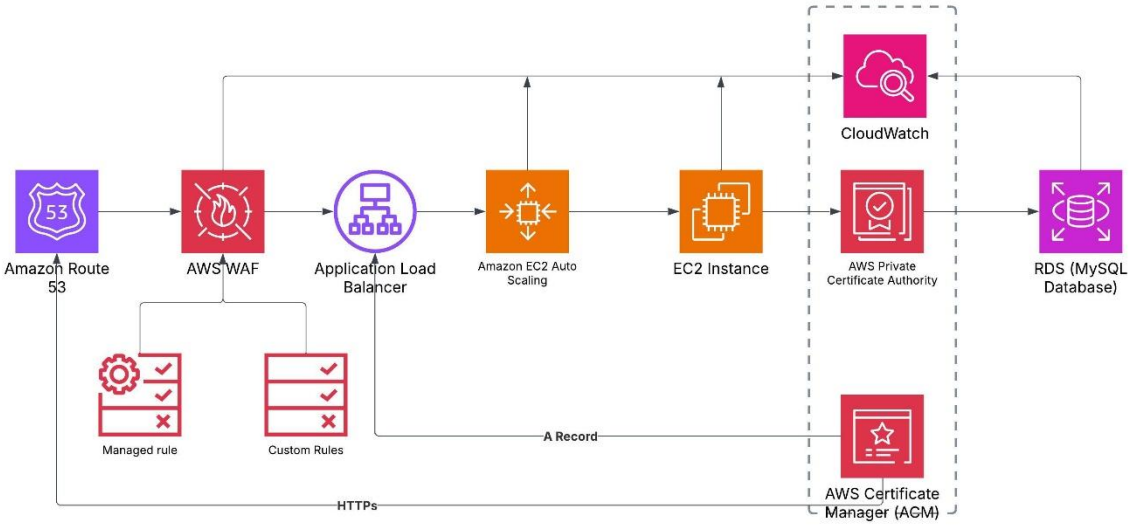

Figure 9: High-Level Architecture of Phase 2

Lucid Chart Link: https://lucid.app/lucidchart/e4c145b4-cdad-406b-9a63-d4b74d50377b/edit?invitationId=inv_e22fb24d-2534-42c6-80ae-eea56e948533&page=0_0#

This diagram represents the enhancements added to secure and optimize the application:

- Amazon Route 53 handles DNS routing to direct traffic toward the web application.
- AWS WAF (Web Application Firewall) provides protection against common exploits like SQL injection and XSS attacks. WAF logs are streamed into CloudWatch Logs for real-time monitoring and incident analysis.
- AWS Certificate Manager (ACM) is used to provision SSL/TLS certificates that enable HTTPS for encrypted data transmission.

- The Application Load Balancer (ALB) continues to route user traffic intelligently to the EC2 backend.
- Amazon RDS (MySQL) remains the backend database, supporting encrypted in-transit connections and secure access from the application layer.

4.2 Setup AWS WAF

AWS WAF protects the e-commerce web application from common web attacks such as SQL Injection and Cross-Site Scripting (XSS) by filtering and blocking malicious requests from the internet.

Configuration AWS WAF with the Load Balancer:
Created a Web ACLs in AWS WAF named ecomerce-waf-acl with some common rules like AWSManagedRulesCommonRuleSet, AWSManagedRulesSQLiRuleSet, AWSManagedRulesAdminProtectionRuleSet, AWSManagedRulesAnonymousIpList, and AWSManagedRulesAmazonIpReputationList, so that the WAF can help the web application from SQL Injection and XSS.

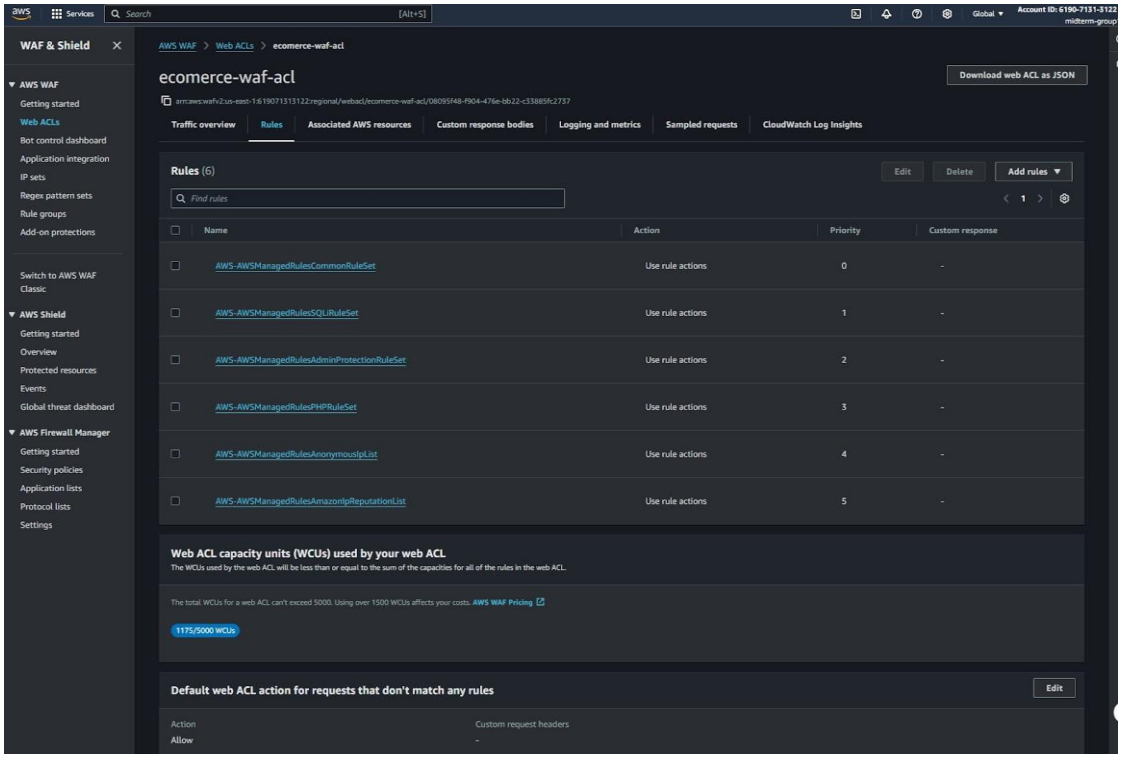

Figure 10: AWS WAF Config with Managed Rules and Custom Rules

Next, we associated this Web ACLs with AWS resource which is our Application Load Balancer named midterm so that it will filter out all the traffic before reaching to our EC2 instance which is hosting our website.

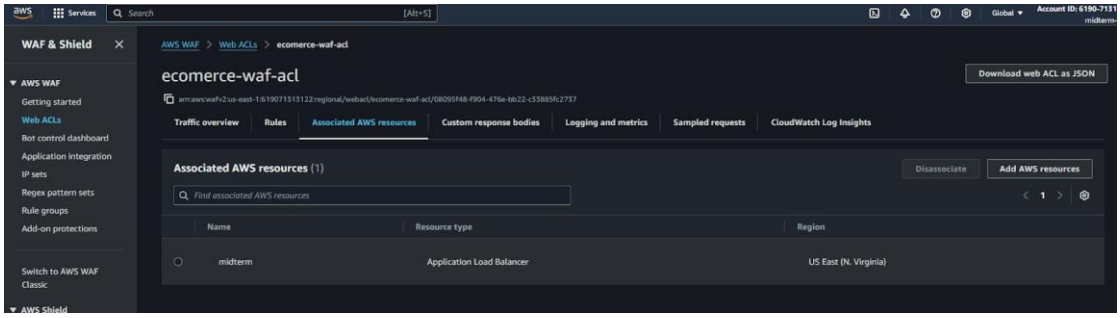

Figure 11: AWS WAF associated with the Application Load Balancer midterm

Monitoring block request:

For monitoring, we are using AWS CloudWatch to monitor all the traffic going through the website. The CloudWatch Metric logs include the block request.

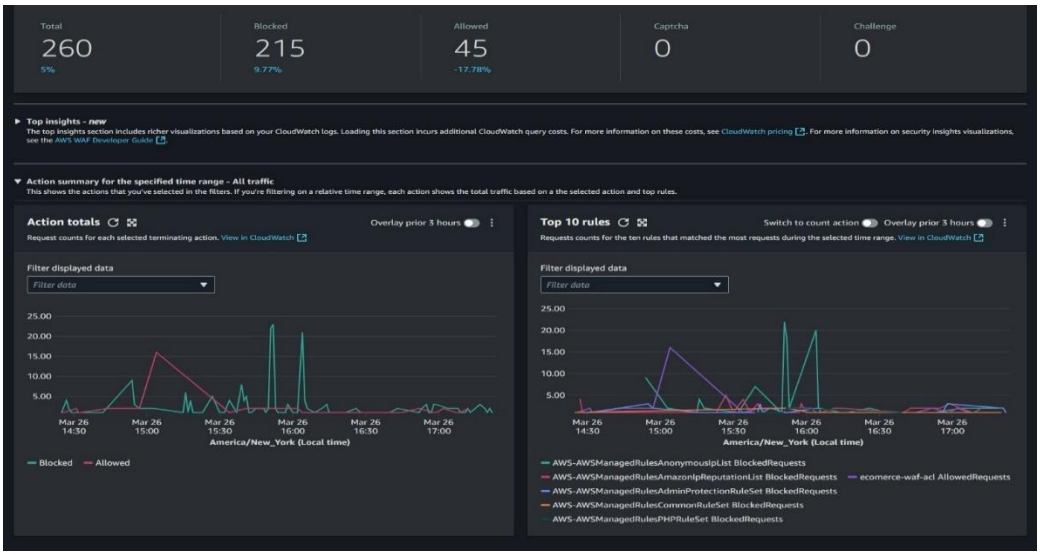

Figure 12: Monitoring Block and Allowed request base on each rule

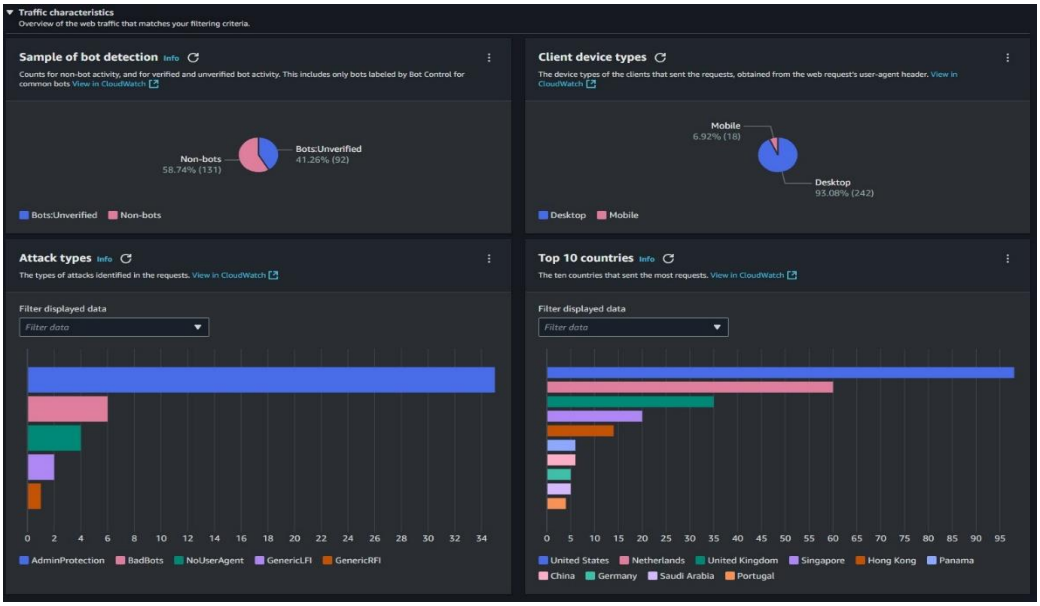

Figure 13: Monitoring top 10 countries accessing the website and attack types

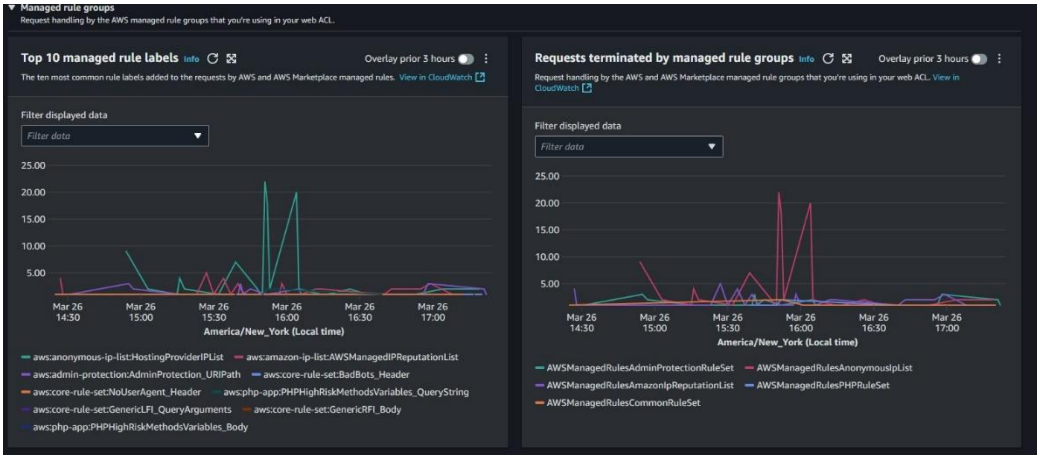

Figure 14: Top 10 managed rules and requests terminated by managed rules

CloudWatch logs enable continuous monitoring for the web traffic, detecting suspicious patterns. By applying basic rules in WAF we can prevent false positives and missed threats.

4.3 DNS and SSL Configuration with Route 53

Enabling SSL/TLS Certificates through AWS Certificate Manager (ACM) and AWS Route53 for HTTPs ensures that the data is encrypted, when transmitted between the users and the web application, protecting all the sensitive data like login, users' credentials and payment details.

Steps performed were:

Step 1: Registering a domain: We have to register a domain name for the web application, so we registered a domain named group18ecommerce.com directly through AWS Route53

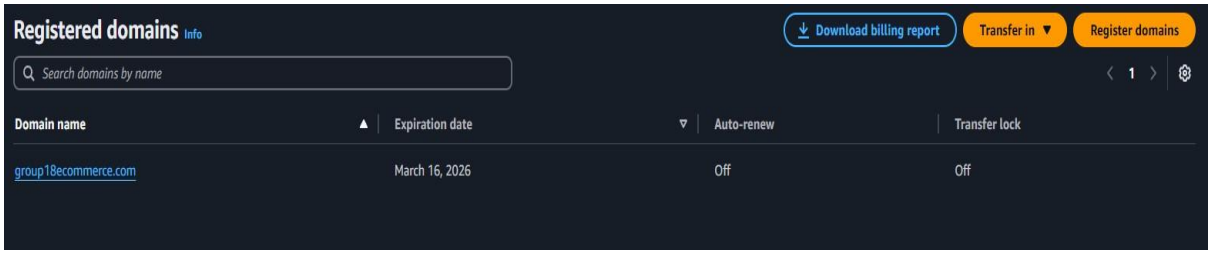

Figure 15: Domain name registered in Route53

Step 2: Create a certificate on ACM: Requested a public certificate in ACM with the domain name "group18ecommerce.com"

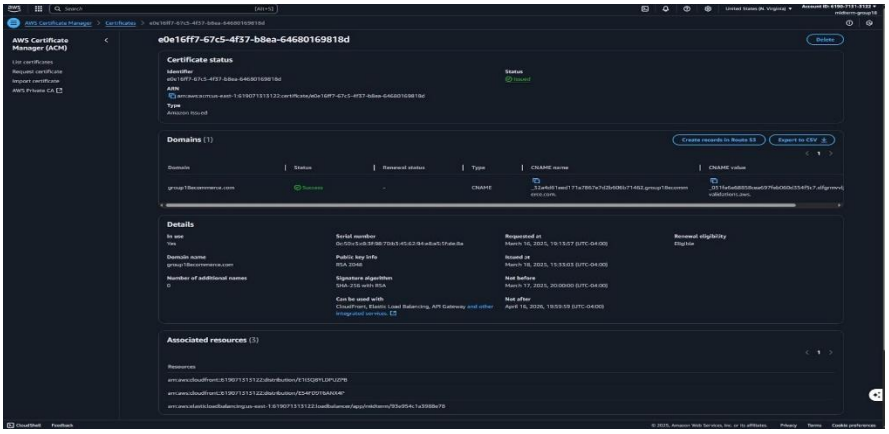

Figure 16: Created certificate on ACM using DNS name group18ecommerce.com

Step 3: Using Route53 for DNS -

- Created a public Hosted Zone in Route53 with the domain name group18ecommerce.com
- Assigned Canonical Name Record (CNAME) from ACM to Route53
- Create a A Record to route traffic to an IPv4 address and AWS resource which is our Application Load Balancer (ALB) midterm.

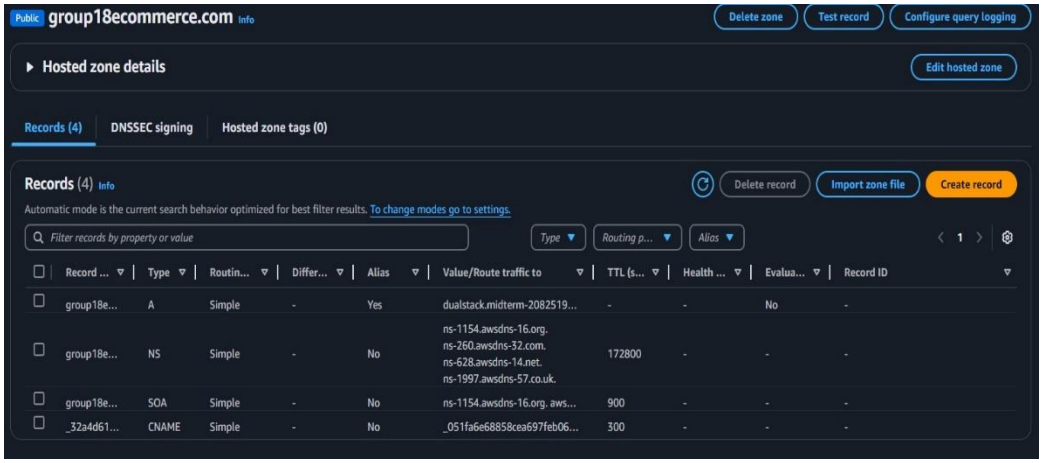

Figure 17: AWS Route 53 Hosted Zone with A Record and CNAME

Step 4: Redirect HTTP port 80 to HTTPS port 443: We went into midterm load balancer and edit HTTP rule so that every time someone try to access the website through HTTP port 80 it will automatically redirect to HTTPS port 443.

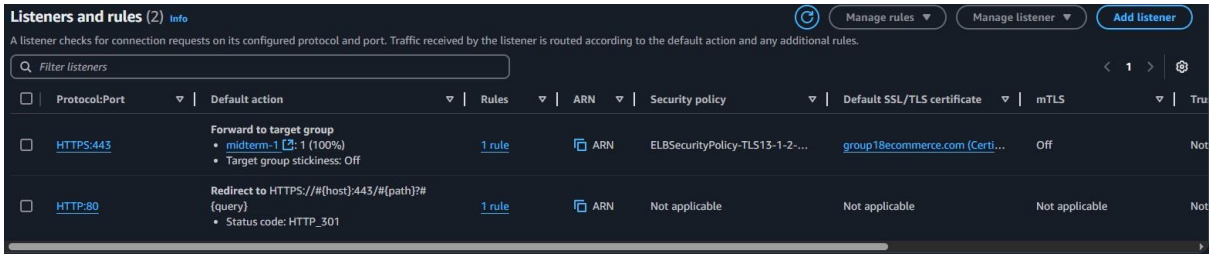

Figure 18: Redirect HTTP to HTTPS

The outcome of the above performed steps lead to successfully access the website running on the HTTPS protocol.

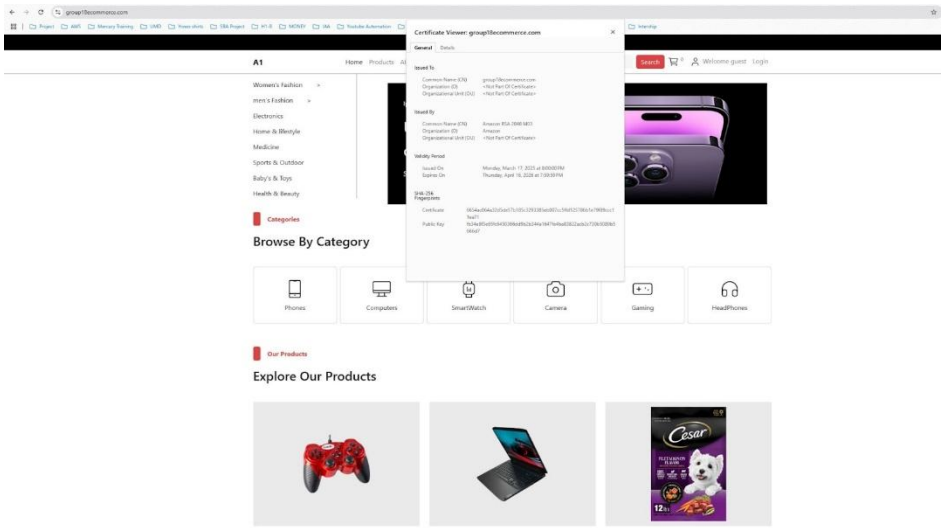

Figure 19: Website worked with DNS and SSL

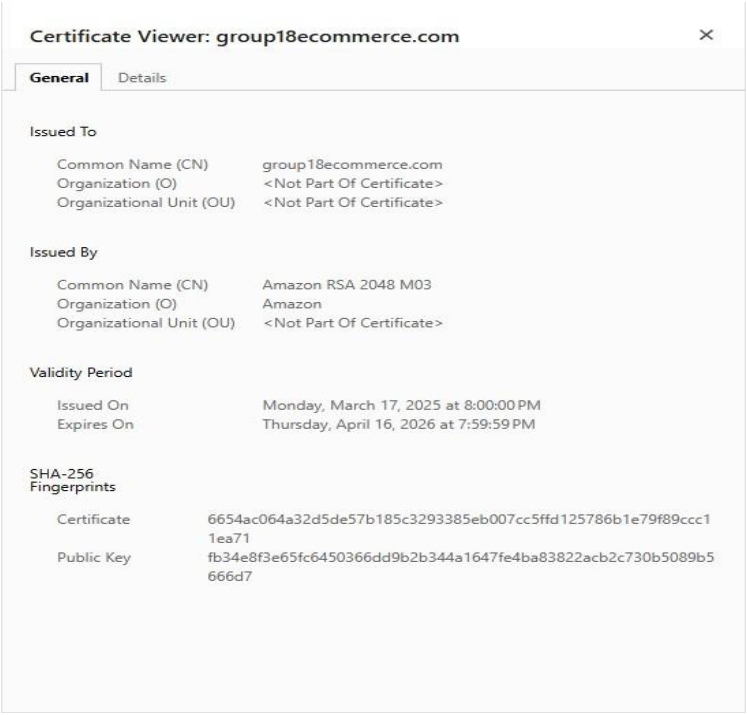

Figure 20: DNS registered to the website

4.4 Encryption of Data in Transit

To ensure secure communication between the web application and the Amazon RDS database, SSL encryption was implemented for data in transit. By enabling SSL, all information exchanged over the network such as user inputs, transactions, and database queries are encrypted, mitigating the risk of data interception or tampering. The SSL certificate provided by Amazon RDS was integrated into the EC2 instance, and the database connection was updated to use MYSQLI_CLIENT_SSL flags to enforce encrypted connections.

Steps performed were:

Step 1: Download the SSL Certificate for Amazon RDS. AWS provides a Certificate Authorities (CA) to establish a trusted SSL connection.

Step 2: Modify the RDS Parameter Group to Require SSL

```
                      Get server public key
  --ssl-mode=name     SSL connection mode.
  --ssl-ca=name       CA file in PEM format.
  --ssl-capath=name   CA directory.
  --ssl-cert=name     X509 cert in PEM format.
  --ssl-cipher=name   SSL cipher to use.
  --ssl-key=name      X509 key in PEM format.
  --ssl-crl=name      Certificate revocation list.
  --ssl-crlpath=name  Certificate revocation list path.
  --tls-version=name  TLS version to use, permitted values are: TLSv1.2,
                      TLSv1.3
  --ssl-fips-mode=name
                      SSL FIPS mode (applies only for OpenSSL); permitted
                      values are: OFF, ON, STRICT
  --tls-ciphersuites=name
                      TLS v1.3 cipher to use.
  --ssl-session-data=name
                      Session data file to use to enable ssl session reuse
  --ssl-session-data-continue-on-failed-reuse
                      If set to ON, this option will allow connection to
                      succeed even if session data cannot be reused.
```

Figure 21: List of parameters to be checked while configuring SSL

Step 3: Verify if SSL has deployed or not by running the command --ssl-mode = VERIFY_CA

```
ubuntu@ip-172-31-68-158:/var/www/html/includes$ mysql -h midterm-rds-db.czgmsqsesmff.us-east-1.rds.amazonaws.com -u admin -p --ssl-ca=/var/www/html/rds-combined-ca-bundle.pem --ssl-mode=VERIFY_CA
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 238037
Server version: 8.0.40 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
```

i-09de0bb12982c9e47 (Midterm-EC2)
PublicIPs: 3.238.116.190 PrivateIPs: 172.31.68.158

Figure 22: MYSQL Access Granted via SSL

Step 4: Verify SSL is active by checking the variable name SSL_Cipher if it has a value, then the SSL is enabled

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 238037
Server version: 8.0.40 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
+---------------+-------------------------+
| Variable_name | Value                   |
+---------------+-------------------------+
| Ssl_cipher    | TLS_AES_256_GCM_SHA384 |
+---------------+-------------------------+
1 row in set (0.05 sec)

mysql>
```

i-09de0bb12982c9e47 (Midterm-EC2)
PublicIPs: 3.238.116.190 PrivateIPs: 172.31.68.158

Figure 23: Verifying SSL Cipher after getting into MYSQL DB

Step 5: Modify the Web Application's Database Connection String: In the connect.php file, we added the downloaded Certified Authority (CA) file here by providing the path $ssl_cert = "/var/www/html/rds-combined-ca-bundle.pem".

Figure 24: Providing path to CA file in the connect.php file.

Step 6: Since we added SSL to RDS, the Amazon Machine Images (AMIs) of the EC2 instance that host our website will also change as well, so we created new AMIs named Midterm-SSL-RDS-AMI in the same launch template midterm_template to update the changes in RDS. Then we changed the default version of the launch template to the newly created version which is 2.

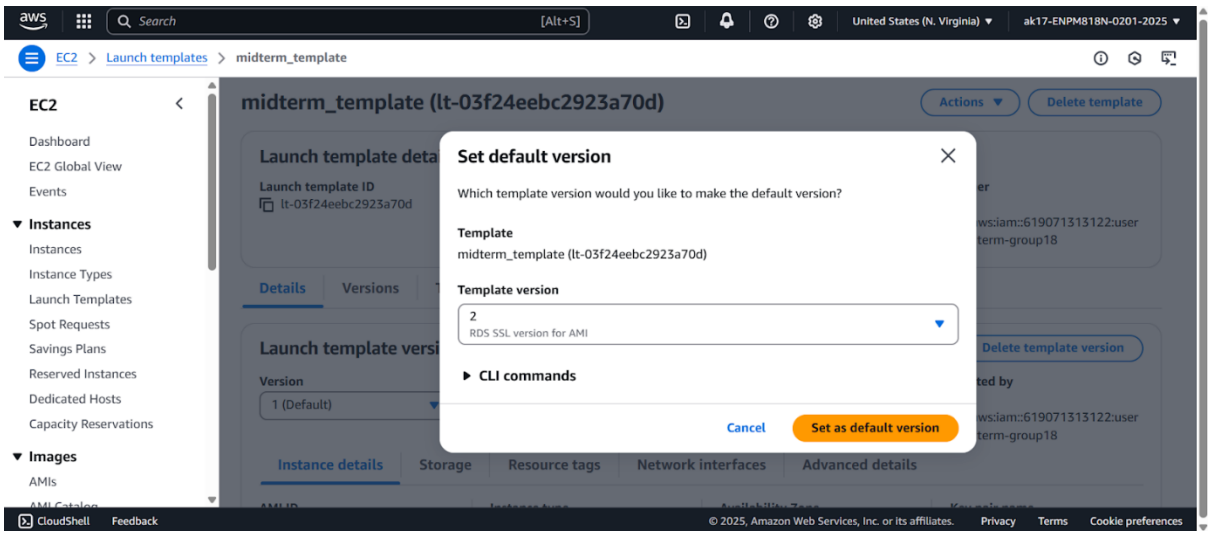

Figure 25: Launch-Template Version changed to 2

After going through all the steps, SSL has been enabled for RDS and all the database connections are now secured and encrypted.

Phase 3: Content delivery and Performance Optimization

This phase focuses on improving the performance, scalability, and user experience of the web application by leveraging AWS content delivery and monitoring services. Amazon CloudFront is used as a CDN to cache and deliver static assets from S3 with low latency. To further enhance performance, GZIP compression and custom cache policies are applied. Real-time monitoring is achieved through Amazon CloudWatch, allowing visibility into traffic patterns, cache efficiency, and potential bottlenecks.

5.1 Architecture Overview

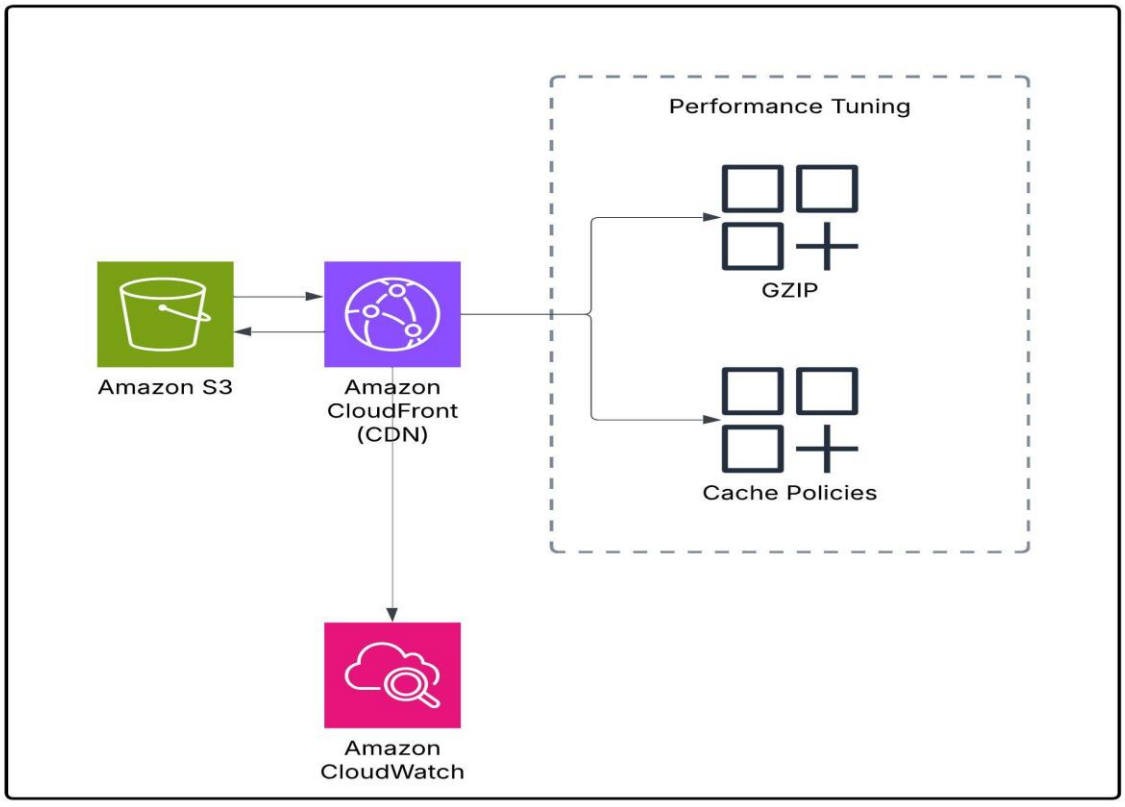

Figure 26: High-level architecture of Phase 3

Lucid Chart Link: https://lucid.app/lucidchart/a29d5213-1b83-4056-a376-4799ac053c9d/edit?invitationId=inv_78f6dbcf-43cc-4c4e-be44-467d770e57a4&page=0_0#

This diagram represents the enhancements added to improve content delivery and performance:

- Amazon CloudFront delivers static content globally with low latency using cached assets from S3.
- Amazon S3 stores static files and integrates with CloudFront for scalable delivery.
- GZIP compression reduces file size for faster page loads and lower bandwidth use.
- Cache policies control how assets are cached and refreshed to improve performance.
- Amazon CloudWatch tracks latency, request rates, and caching efficiency for optimization.

5.2 Cloud Front CDN Setup

To improve content delivery and load times globally, a CloudFront distribution was created with our S3 bucket as the origin source. We used a custom domain name linked via Route 53 and attached an SSL/TLS certificate from AWS Certificate Manager for secure HTTPS delivery. Logging was enabled and integrated with CloudWatch for request tracking and performance analysis.

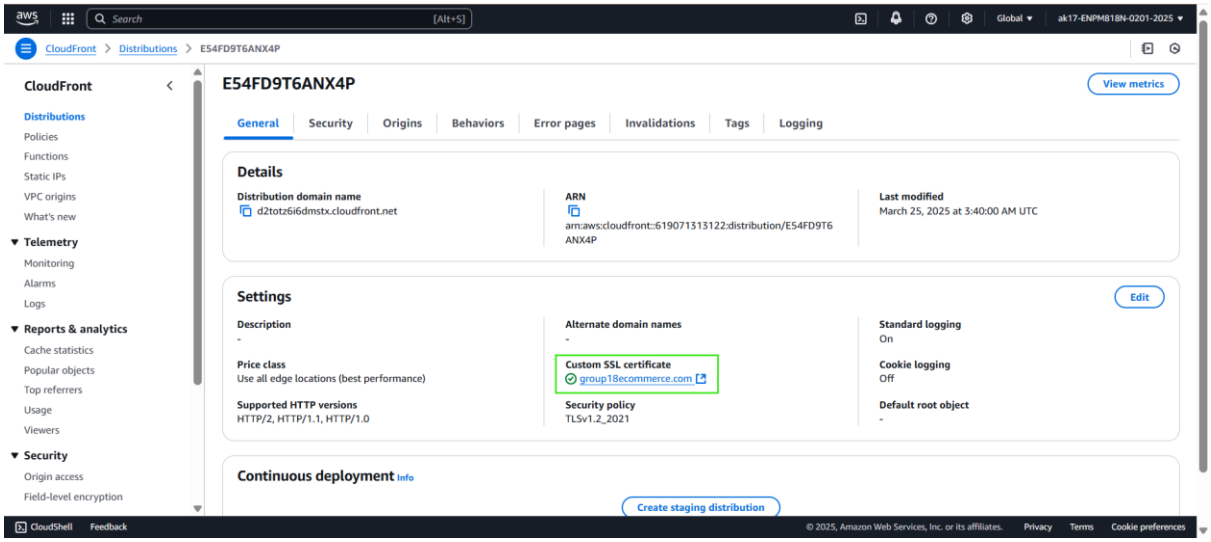

Figure 27: CloudFront setup

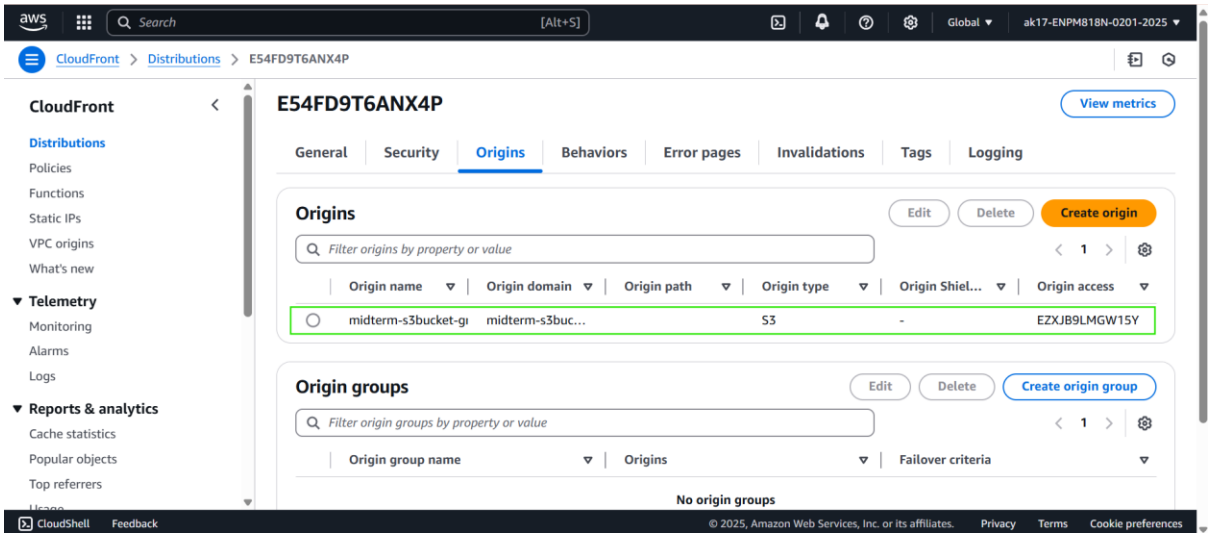

Figure 28: CloudFront setup

5.3 Performance Tuning and Cache Policies

To enhance performance and reduce page load times, GZIP compression was enabled in the cache policy applied to the CloudFront distribution. This compresses the size of static assets like CSS and JavaScript before serving them to the user, reducing bandwidth usage and improving client-side performance.

The cache policy was configured with optimized TTL (Time-to-Live) values to strike a balance between content freshness and performance. A low minimum TTL allows frequent updates for dynamic files, while a high maximum TTL helps cache static content efficiently, reducing repeated requests to the S3 origin.

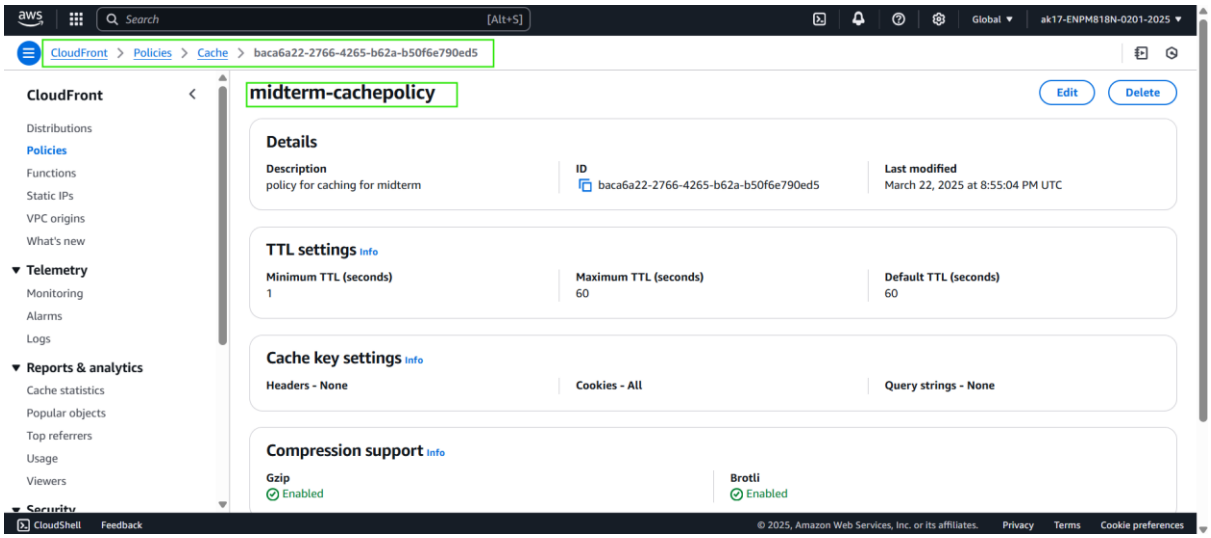

Figure 29: Cache Policy

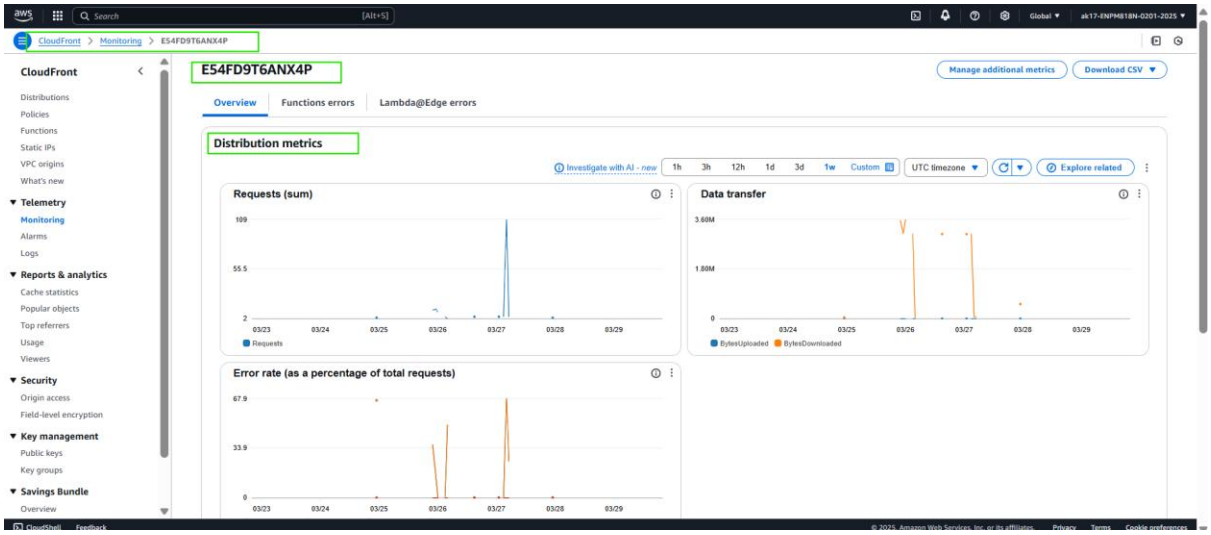

Figure 30: Distribution Metrics

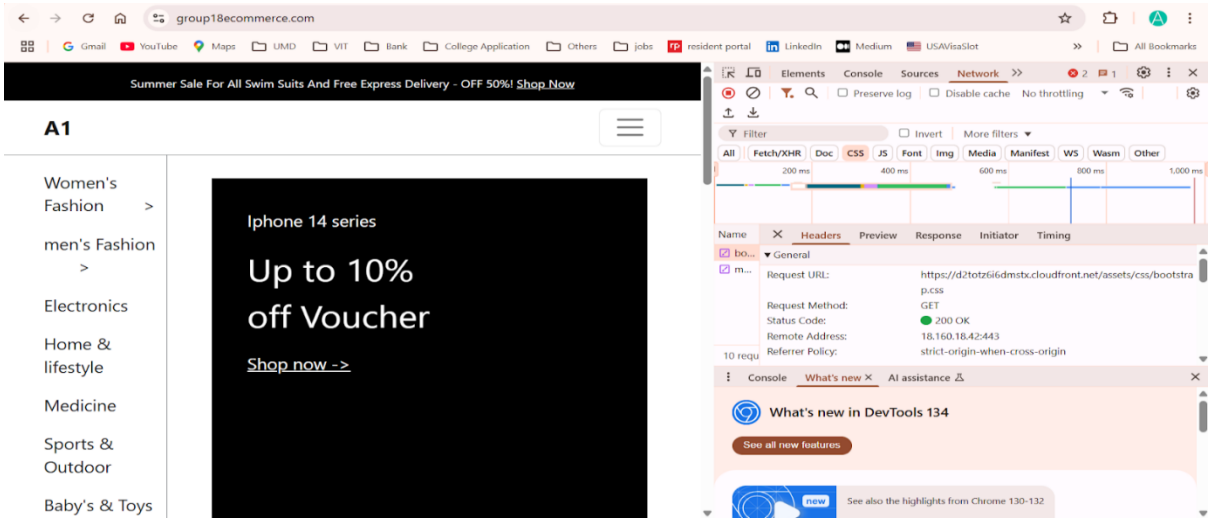

Figure 31: Request URL hits href

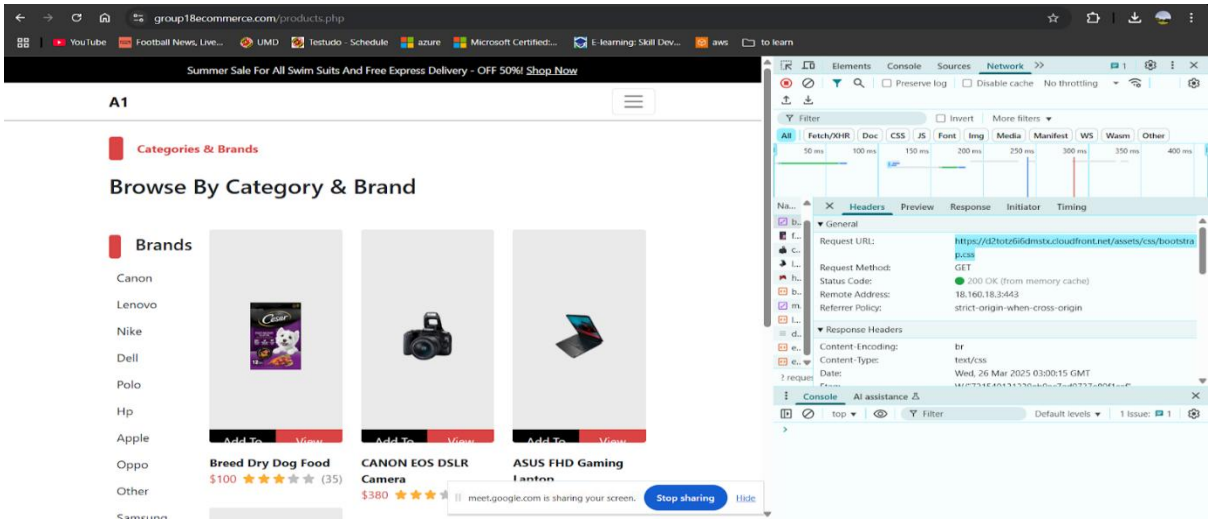

Figure 32: href works to get cache memory and load the image

5.4 CloudWatch Alarm Monitoring

Amazon CloudWatch was configured to monitor CloudFront request rates, error responses, and cache hit/miss ratios. This setup helps in real-time monitoring of content delivery performance and identifying potential issues like latency spikes or caching inefficiencies. Additionally, the logs from AWS WAF are integrated to enhance visibility of blocked threats.

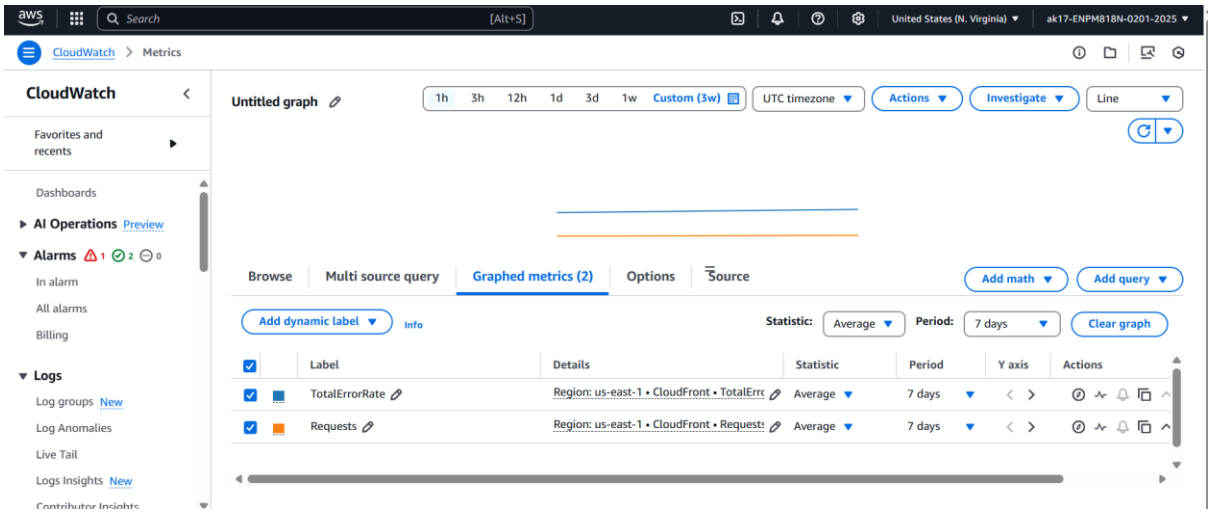

Figure 33: Request and Total Error Rate Metrics in CloudWatch

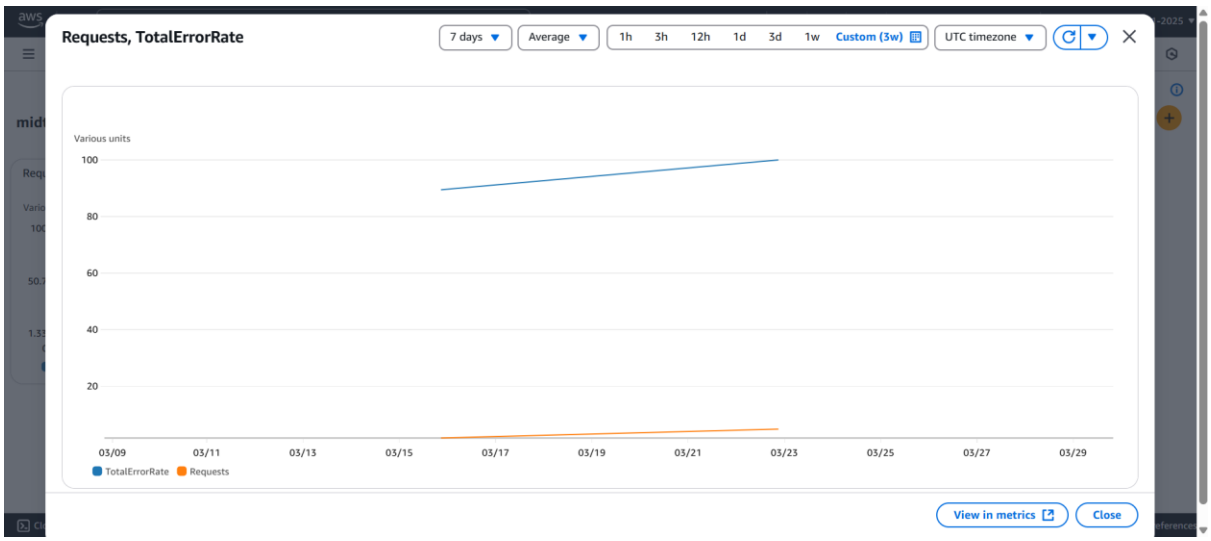

Figure 34: CloudWatch Dashboard Graph of requests & error rate for past 7 days

Phase 4: Testing and Monitoring

This phase focuses on validating the scalability, performance, and cost-efficiency of the deployed cloud infrastructure. To simulate real-world traffic and stress test the system, Apache JMeter was used to generate load on both the web application and the RDS backend. Metrics like CPU utilization and database performance were monitored using Amazon CloudWatch and CloudTrail to ensure that the Auto Scaling Group reacts appropriately under varying loads. AWS Cost Explorer was also leveraged to analyze usage patterns and evaluate cost optimization opportunities. This end-to-end monitoring helped fine-tune the infrastructure.

6.1 Architecture Overview

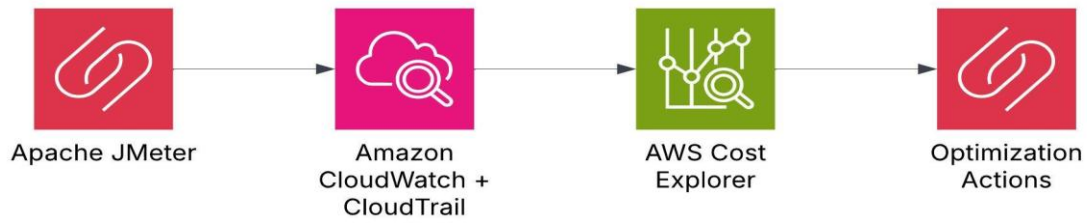

Figure 35: High-level architecture of Phase 4

Lucid Chart Link: https://lucid.app/lucidchart/34431140-08dd-438b-ba4c-6bb81aa17669/edit?invitationId=inv_841895cc-f8eb-466f-8ce3-bdc464bccadd&page=0_0#

This diagram illustrates the tools used for testing, monitoring, and optimizing the application infrastructure:

- Apache JMeter generates traffic to stress test the EC2 instances and RDS, ensuring scalability and performance under load.
- Amazon CloudWatch tracks key metrics like CPU usage and request rates to validate Auto Scaling behaviour and system health.
- AWS CloudTrail logs API activities during testing, enabling traceability and auditability of all triggered events.
- AWS Cost Explorer provides insights into cost impact during testing, helping identify expensive resources and usage spikes.
- Optimization Actions are based on monitoring and cost data to fine-tune scaling policies, reduce spend, and improve efficiency.

6.2 Stress Testing Database and Auto-scaling Group

Stress Testing ASG verifies the ability of the ASG to handle the unexpected traffic spikes and maintain the website performance and availability.

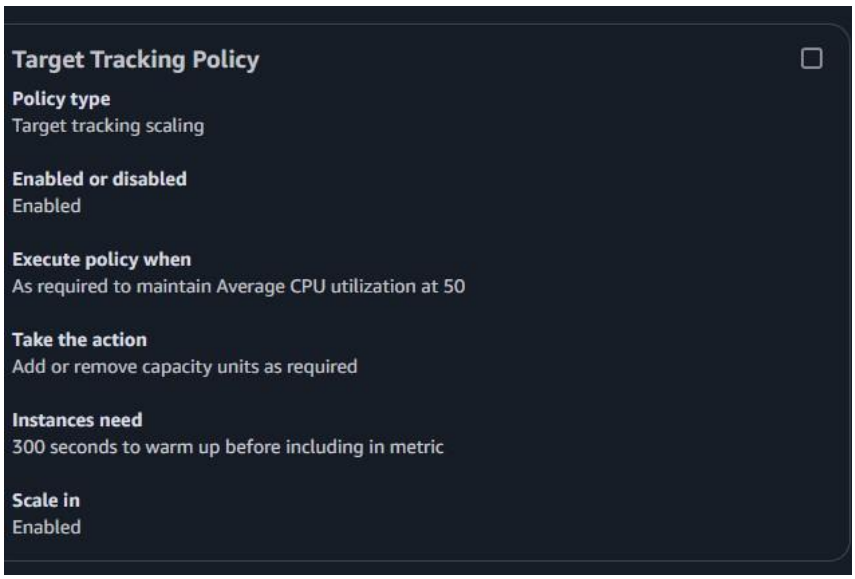

Figure 36: ASG will scale if average CPU utilization at 50%

Step 1: Download JMeter and set up test plan configuration.

Parameter explanation:

- **Number of Threads (Users):** 500+ users is more than enough for this website. It will gradually increase users to find the GPU threshold.
- **Ramp-Up Period:** Set to 60s. It will spread out the user's requests over time to simulate realistic traffic.
- **Loop Count:** Uncheck Infinite, this will let the test run for a specific duration instead of looping.
- **Duration (seconds):** Set to 300 seconds which is 5 minutes, and this will be long enough to trigger CPU in ASG to scale up to or more than 50%.
- **Startup Delay:** Set to 5 seconds will allow initials set up before the test starts.

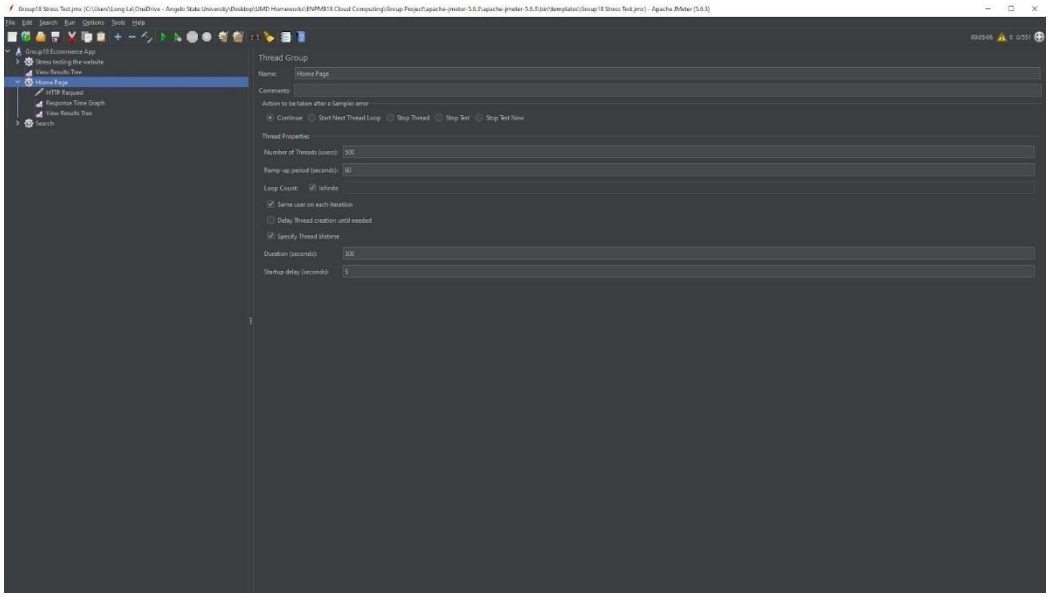

Figure 37: JMeter Configuration

Step 2: Start the test and let it run for 5 minutes according to our test plan configuration.
This will make the website go down, and when the CPU utilization goes above 50%, ASG will create another instance to handle the load, and then the website goes back up again.

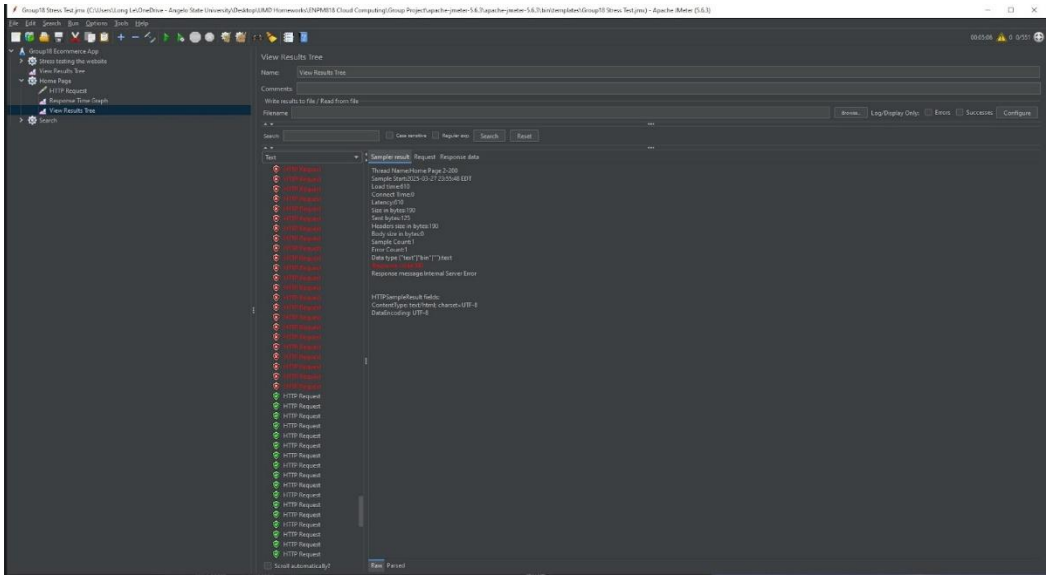

Figure 38: Stress test makes website go down and back up, when ASG is triggered

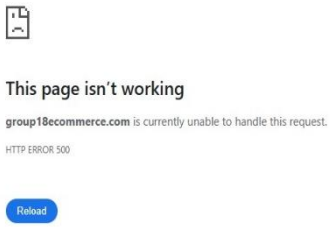

Figure 39: Website went down because of the load spike up during load testing

Step 3: Monitoring CloudWatch metrics

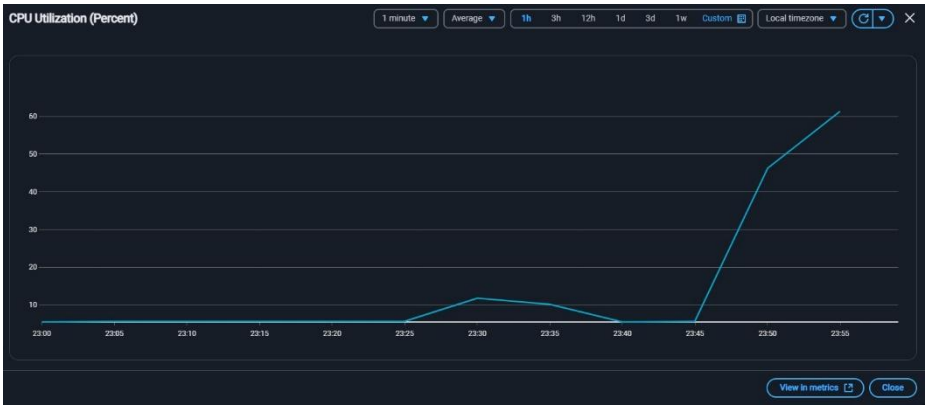

Figure 40: CPU Utilization went above 50%

This will trigger ASG to scale and create another EC2 instance.

Figure 41: ASG triggered and created another instance to handle the load

After ASG is scaled up and creates new instances to handle the load, the website starts working again.

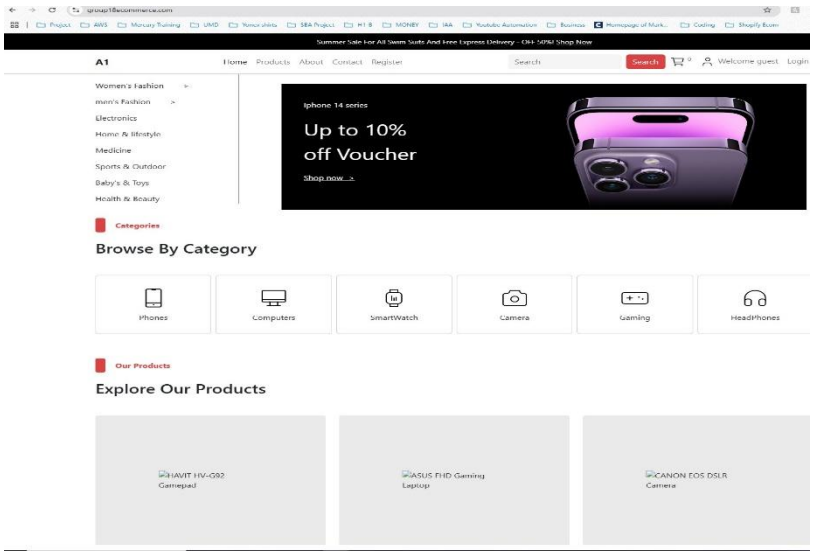

Figure 42: Website is back to work

Next, we move on to test the RDS. Queries load testing an Amazon RDS database helps ensure that it can handle the heavy workloads efficiently while maintaining performance, stability, and scalability.

Step 4: Set up Thread Group in Jmeter
Number of Thread: 200 (users)
Ramp-up period: 10 seconds
There will be 200 users sending queries to stress test RDS.

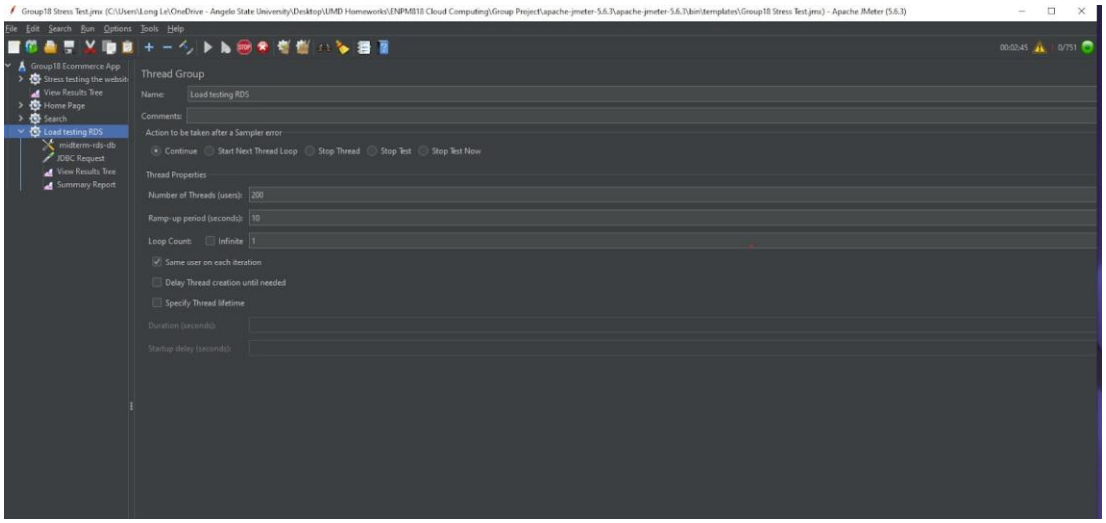

Figure 43: Thread Group Config for RDS load testing

Step 5: Set up JMeter Database Connection Configuration to RDS
Variable name for created pool: midterm-rds-db
To connect to our database, we use Database URL: jdbc:mysql://midterm-rds-db.czgmsqsesmff.us-east-1.rds.amazonaws.com:3306/ecommerce_1.

This includes:
RDS endpoint: midterm-rds-db.czgmsqsesmff.us-east-1.rds.amazonaws.com
Port: 3306
Database name: ecommerce_1
JDBC Driver: com.mysql.jdbc.Driver.
Username and Password: We used the database master username and password.

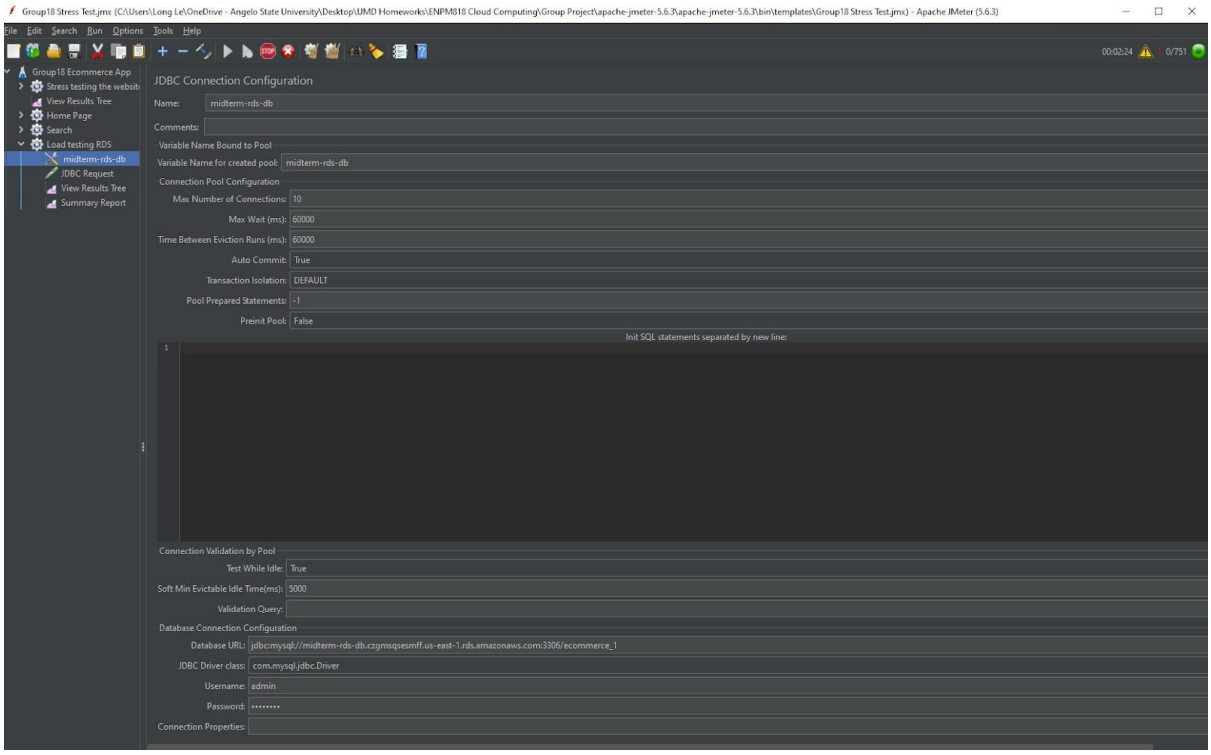

Figure 44: JMeter Database Connection Configuration

Step 6: Set up JDBC Request

Variable name of Pool declared in JDBC Connection Configuration: midterm-rds-db

Query Type: We chose Select Statement (SELECT * FROM products;) to get the “products” table in ecommerce_1 database

Successfully connect to the database and get the products table in the database.

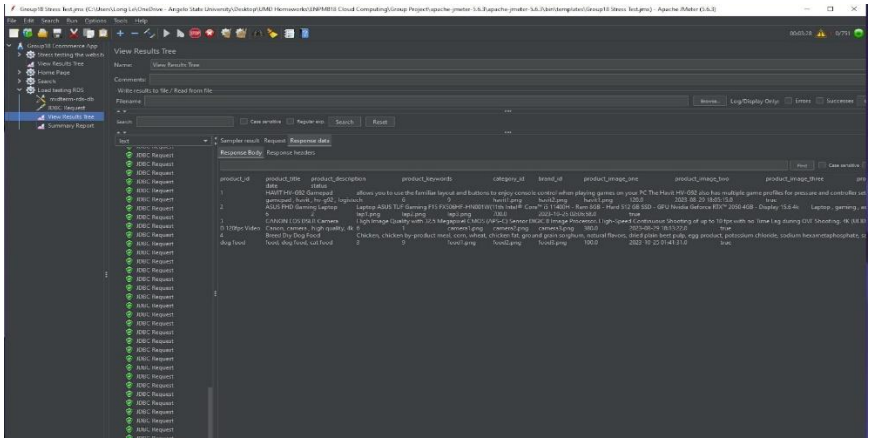

Figure 45: Successfully sent queries and got the products table

Step 7: Monitoring CloudWatch

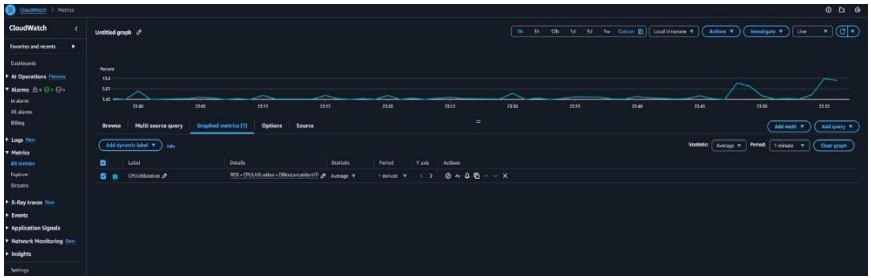

Figure 46: RDS CPU Utilization Graph under stress test

Sending heavy queries made EC2 CPU utilization went up as well and the ASG also had to scale up to handle the load.

Figure 47: EC2 CPU went above 90%

Under heavy load, the EC2 CPU utilization went above 50% and this made website go down, however, the ASG created another instance to handle the load. The website is now working.

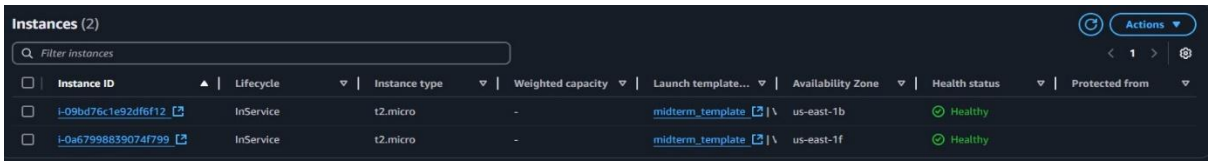

Figure 48: ASG scales up to handle the heavy load

6.3 Monitoring CPU utilization in AWS CloudWatch

To ensure efficient scaling under load, we used Amazon CloudWatch to monitor the CPU utilization of EC2 instances within the Auto Scaling Group. Custom thresholds were set to trigger alarms and automatically scale instances based on usage. In addition to CPU metrics, we monitored CloudFront request rates and latency to validate performance improvements. A CloudWatch Log Group was created to capture real-time logs from AWS services, allowing us to track traffic behavior and application activity during JMeter stress testing. We also enabled AWS CloudTrail for auditing API-level actions, helping us identify resource usage patterns, user activity, and confirm whether scaling events or anomalies occurred during the test

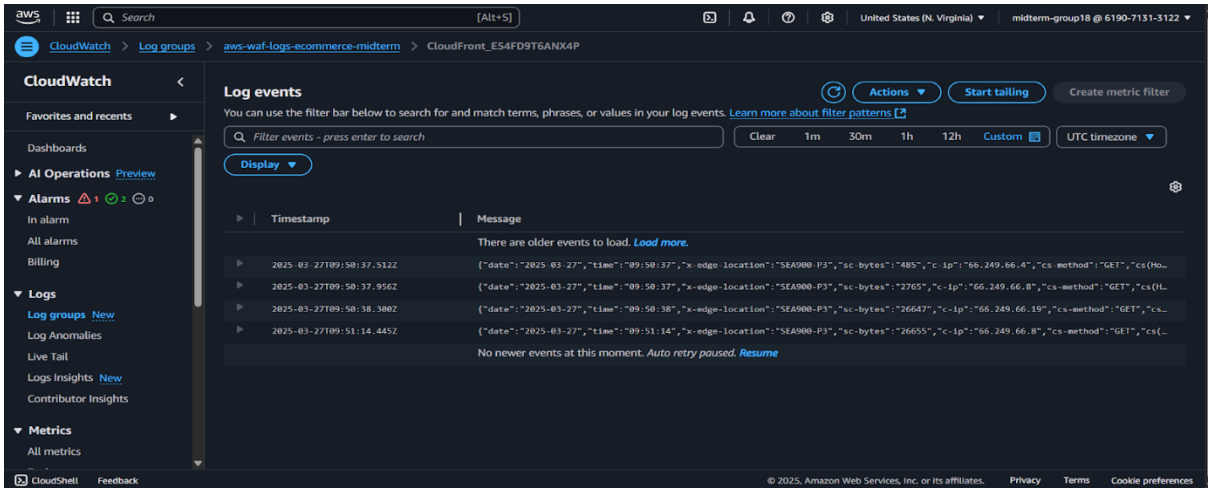

Figure 49: CloudWatch Event Logs

We configured a CloudWatch Log Group to include our resources and capture event logs in real-time. This allows us to monitor key activities and events across the infrastructure. The screenshot below displays detailed log entries, including user data, which helps us analyze incoming traffic and user behavior on the website.

Figure 50: CloudWatch Log Entries

In this screenshot, the selected metrics for our CloudFront distribution are clearly visible. We tracked parameters such as request count and data transfer, which are displayed in the monitoring chart.

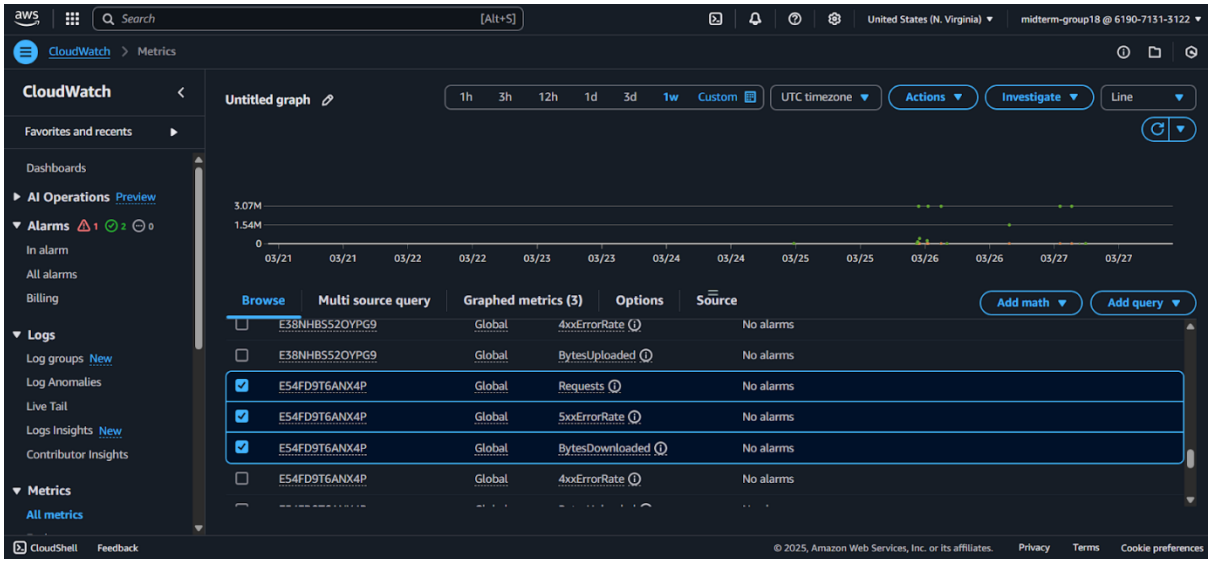

Figure 51: CloudFront Distributions

Further, CloudTrail was configured for event monitoring. The log entries provide detailed insights into user activity, including who logged in and what actions were performed across the AWS environment.

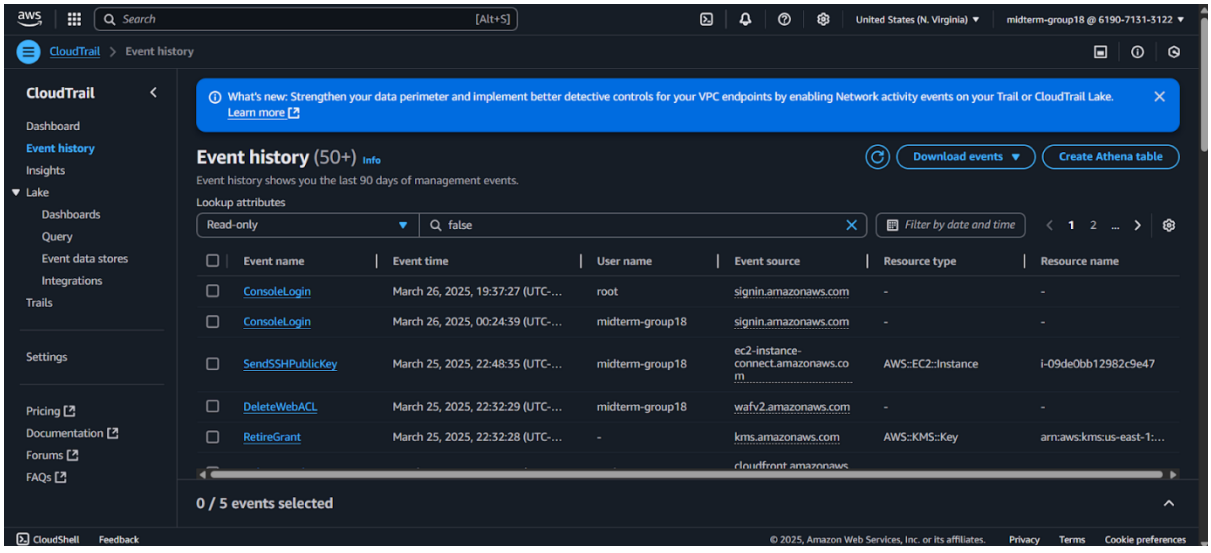

Figure 52: CloudTrail Event Logs

6.4 Cost Optimization

To manage expenses while ensuring performance and scalability, we used AWS Cost Explorer to analyze service-wise spending. The total cost reached $54.29, a significant increase compared to earlier phases. The largest contributor was again the Domain Registrar service, accounting for $14.00, followed by Amazon RDS at $10.52, likely due to persistent database uptime. WAF and Route 53 contributed $2.71 and $0.51 respectively. Notably, EC2 usage remained low, because of the use of one main EC2 instance in the free tier for short test durations.

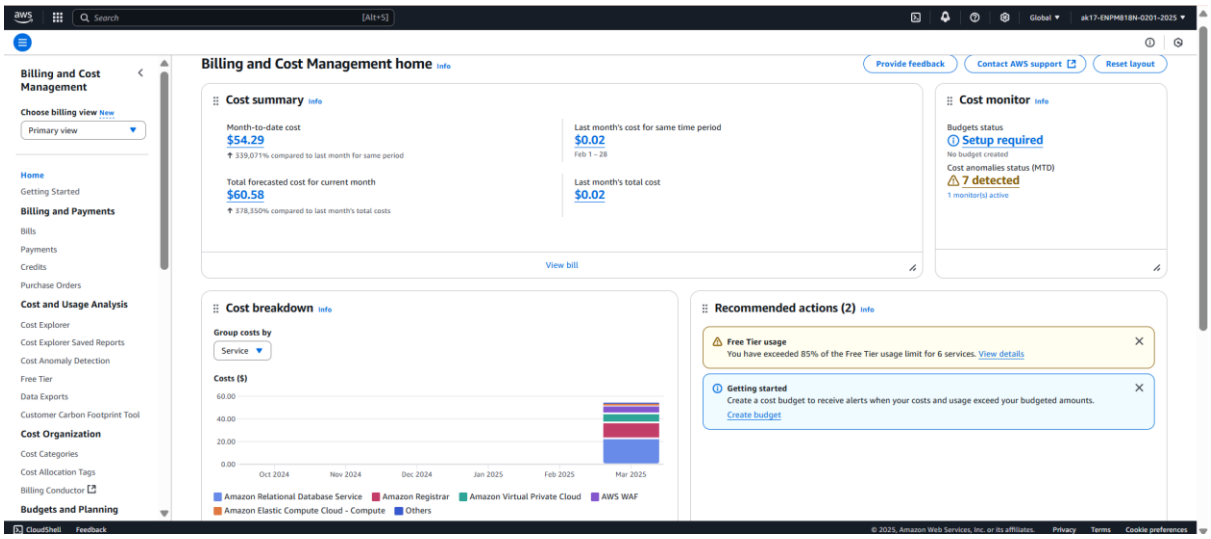

Figure 53: Billing and Cost Dashboard

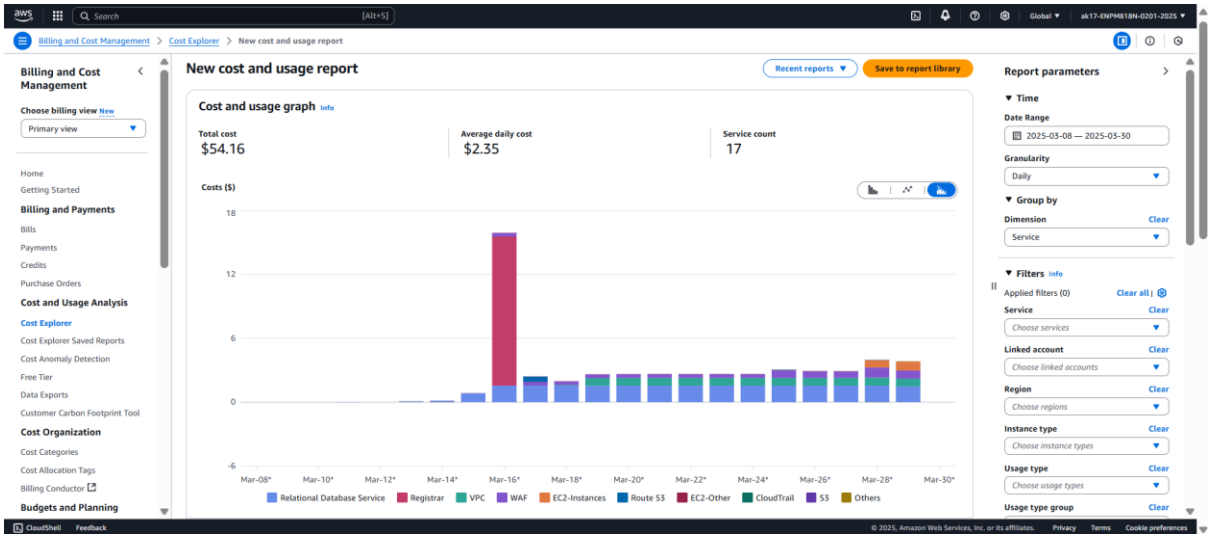

Figure 54: Cost Explorer Usage Report

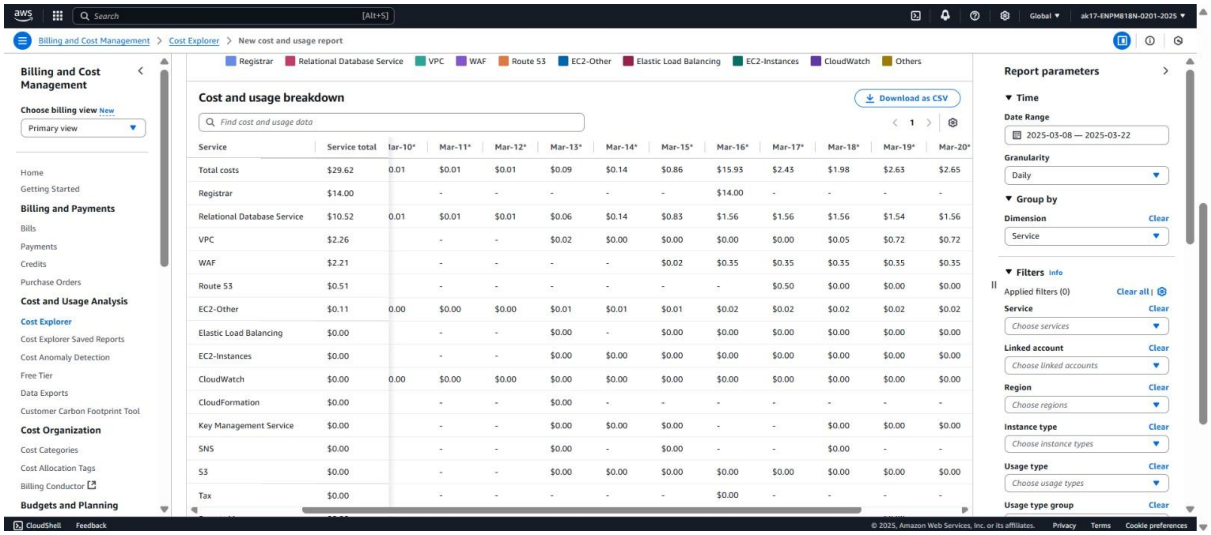

Figure 55: Cost breakdown of Services in Cost Explorer

To further optimize cost:

- Reserved Instances for RDS can be considered if long-term use is planned.
- Savings Plans can reduce EC2-related costs during consistent workloads.
- Unused services and idle resources can be identified through CloudTrail and CloudWatch Logs, allowing cleanup.
- Services like CloudFront, S3, and CloudWatch remained free or minimal, showing effective free-tier utilization.
- AWS Budget can be used for the cost anomalies detected by AWS, indicating irregular spending spikes.

Conclusion

7.1 Lesson Learned

1. **How to Build on AWS:** We learned how different AWS services like EC2, RDS, S3, and CloudFront work together to support a real web application.
2. **Security Best Practices:** We got hands-on experience setting up security layers using AWS WAF, SSL certificates, private/public subnets, and IAM roles to protect the system from external threats.
3. **Scalability and Auto Scaling:** Setting up Auto Scaling taught us how to make applications handle changes in traffic automatically, which is critical for real-world apps.
4. **Monitoring and Troubleshooting:** Services like CloudWatch and CloudTrail helped us understand how to monitor the system and catch issues early through logs and metrics.
5. **Performance Optimization:** Using CloudFront and S3 showed us how CDNs and static storage can speed up websites, especially for global users.
6. **Cost Awareness:** We became more aware of cloud costs and how to analyse them using AWS Cost Explorer. We also understood the benefits of Reserved Instances and Savings Plans.
7. **End-to-End Deployment Skills:** From DNS setup to load balancing, database configuration, and cost optimization. we now have a clearer understanding of deploying a web application in the cloud

7.2 Recommendations

1. Security

- Use IAM roles and policies to give least-privilege access across services.
- Enable Multi-Factor Authentication (MFA) on AWS accounts for extra protection.
- Rotate access keys regularly and avoid hardcoding them in applications.
- Set up AWS Shield for protection against DDoS attacks (especially for public-facing apps).

2. Scalability & Performance

- Implement a caching layer using Amazon ElastiCache (Redis/Memcached) to reduce database load and improve performance.
- Use S3 Transfer Acceleration if users are uploading content from distant locations.

- Enable GZIP compression at the server level to minimize file sizes and improve load times.
- Use Amazon CloudFront custom caching behaviours for frequently accessed dynamic content.

3. Database Optimization

- Enable automated backups and set a suitable retention period.
- Use Read Replicas in RDS to offload read-heavy queries and improve scalability.
- Enable RDS performance insights to monitor and tune SQL queries.
- Optimize database indexes based on the most frequent queries.

4. Cost Efficiency

- Use Reserved Instances or Savings Plans for EC2 and RDS once usage patterns are stable.
- Set up CloudWatch Budgets and Cost Anomaly Detection to alert on unexpected spending.
- Review the AWS Trusted Advisor regularly for cost-saving and performance tips.

5. Monitoring & Maintenance

- Set up detailed CloudWatch dashboards to monitor traffic, CPU, memory, and DB connections.
- Enable AWS Config to track configuration changes across your environment.
- Create lifecycle rules in S3 to move older objects to lower-cost storage tiers like Glacier.

6. Future Enhancements

- Implement CI/CD pipeline using AWS CodePipeline and CodeDeploy.
- Add user authentication using Amazon Cognito.
- Explore serverless options like AWS Lambda and API Gateway for certain functionalities.
- Containerize the app and deploy using ECS or EKS for better resource management.

References

https://www.youtube.com/watch?v=_bEPuvrjB5Y.
https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/domain-register.html
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html#UsingWithRDS.SSL.CertificatesAllRegions
https://www.youtube.com/watch?v=qneVPQtBaYs.
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.SSL.html#UsingWithRDS.SSL.CertificatesAllRegions
https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/GettingStarted.SimpleDistribution.html
https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-trails.html