ENPM818N
Mid-Term Project

Scalable and Secure E-commerce Platform on AWS

Bhanu Teja Panguluri
Bharat Gurbaxni
Yijie Bao
Palak Gupta

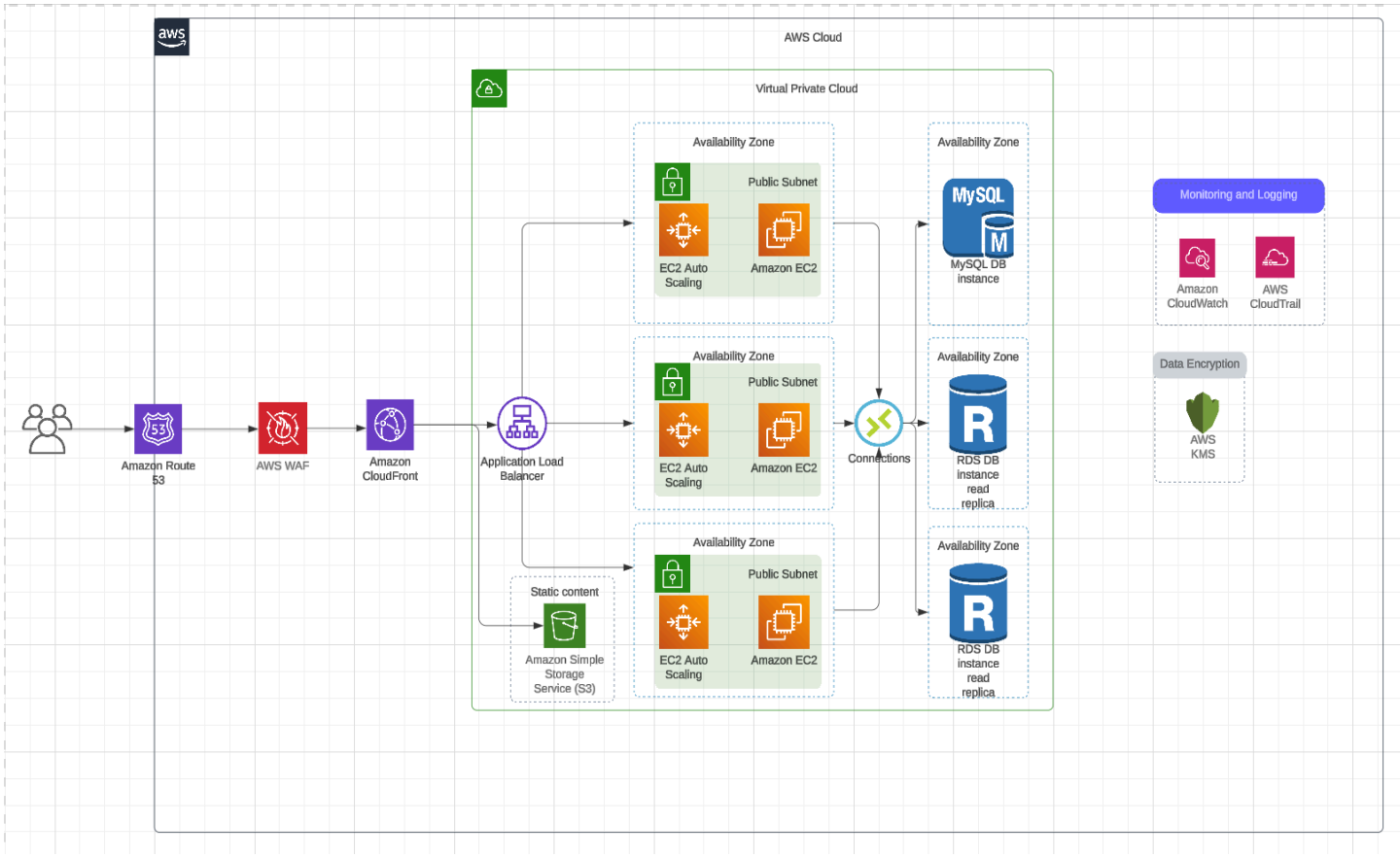
INDEX

Project Objectives

1. Deploy a secure and scalable e-commerce platform using AWS services.
2. Implement Auto Scaling to automatically adjust infrastructure based on traffic.
3. Set up RDS for managing product and customer(users) databases, ensuring data encryption.
4. Protect the application using WAF to mitigate common web vulnerabilities.
5. Use CloudFront CDN to accelerate static content delivery globally.
6. Enable encryption for sensitive data, both in transit and at rest.
7. Design CloudWatch dashboard monitoring key metrics.
8. Provide recommendations for improvement and cost saving.

Architecture Design

- Diagram:

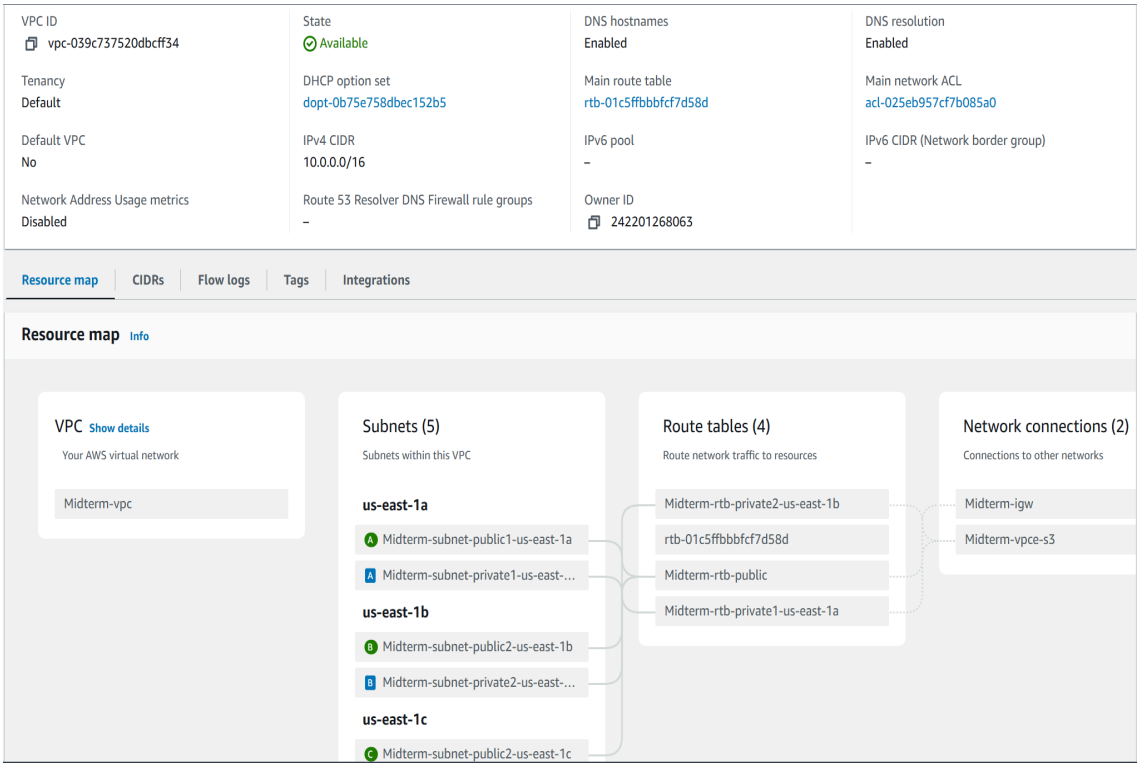


Implementation Details:

1. Infrastructure Setup

a. VPC Configuration:

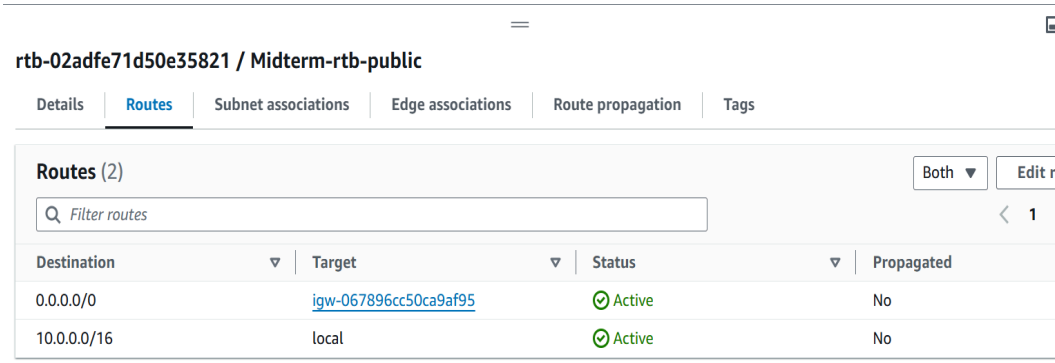
i. **Adding a new VPC** for the implementation of the project with CIDR notation - /16. This is done to add flexibility to the architecture and provide room for auto-scaling.

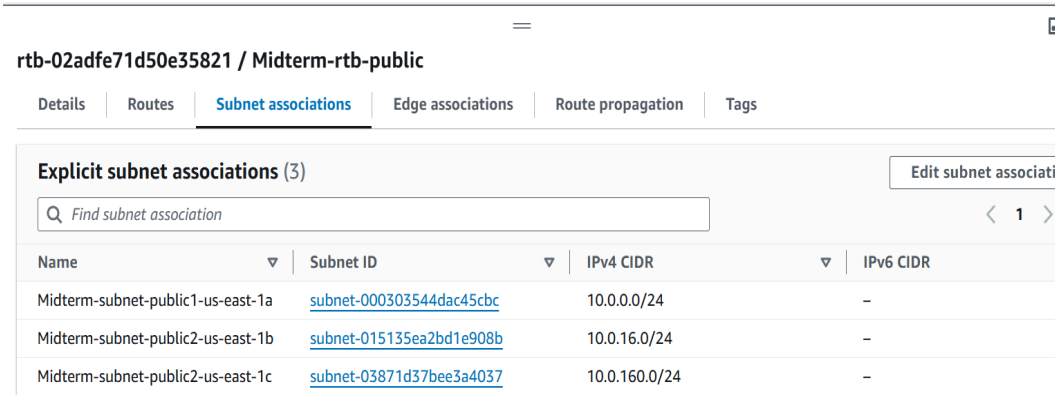


ii. **Adding subnets** with CIDR notation /24 in multiple availability zones to distribute resources overall thus giving high availability to architecture.

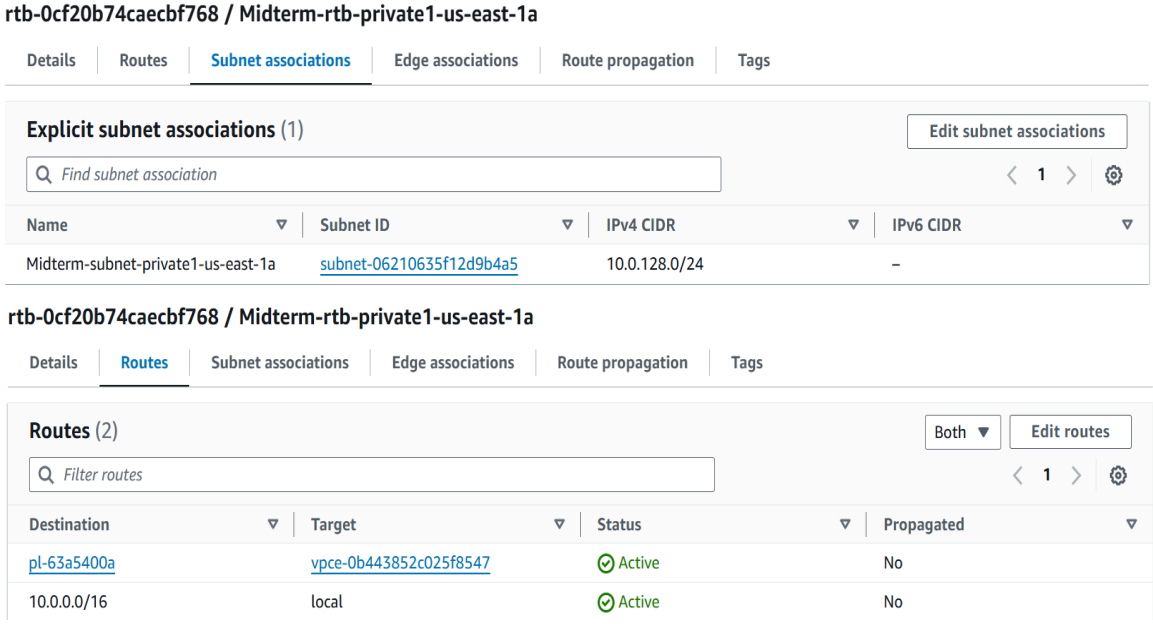
iii. **Add 3 route tables** between public and private tables. Public route tables are supposed to have access to the internet for resources like EC2 and private subnets are used to isolate sensitive resources/data.

a. Public route table:

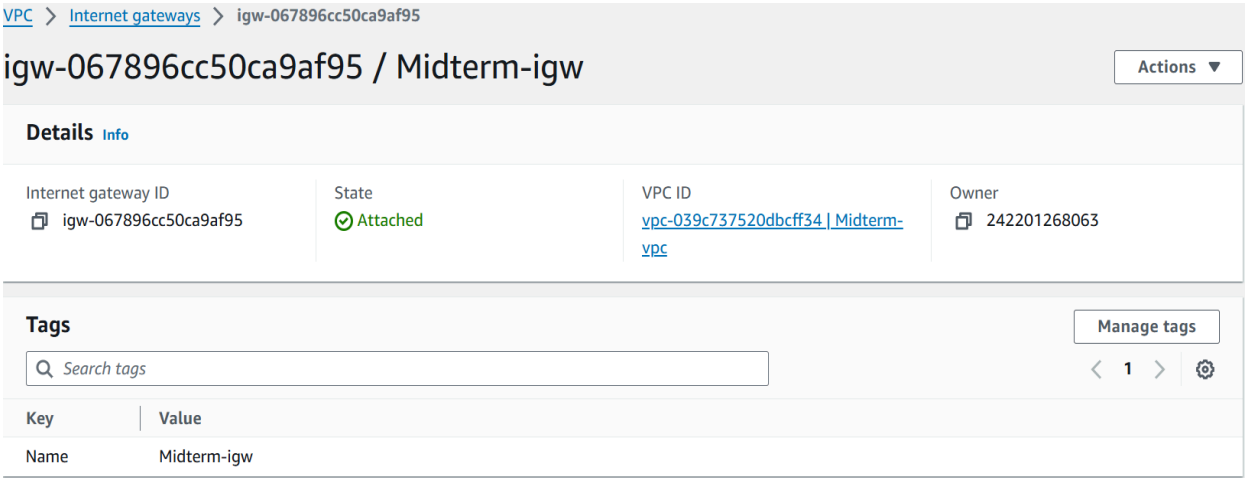




2. Private route table:

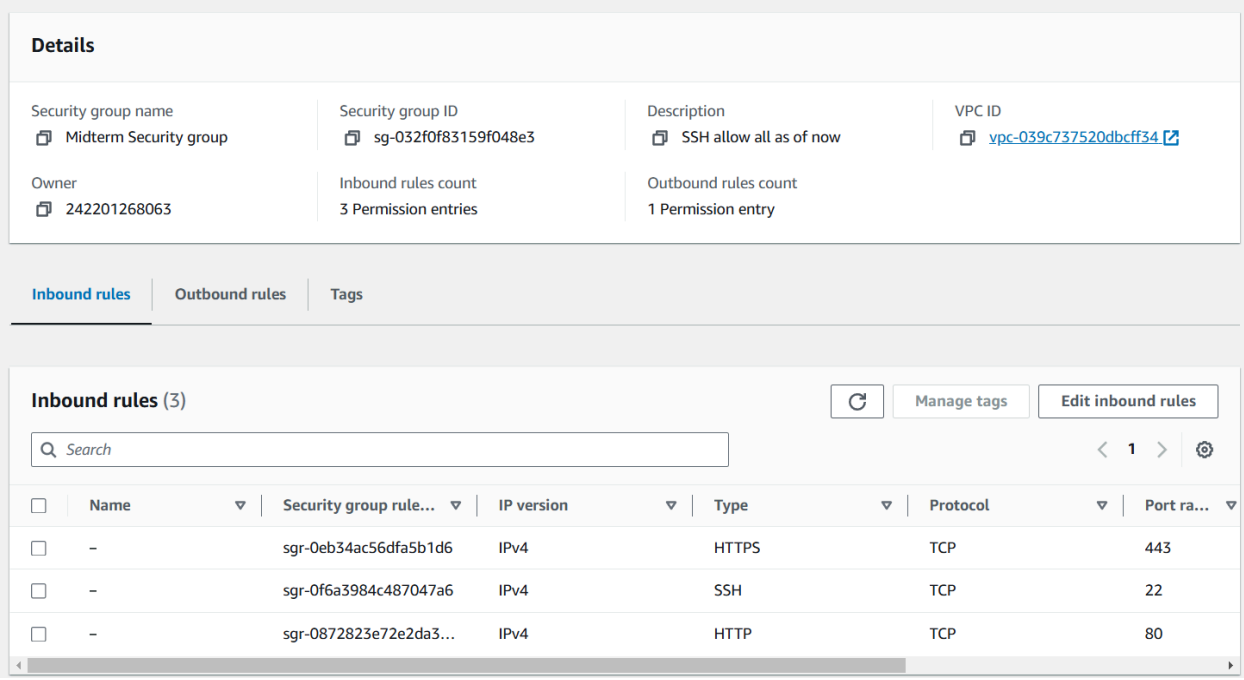


iv. Adding internet gateway:

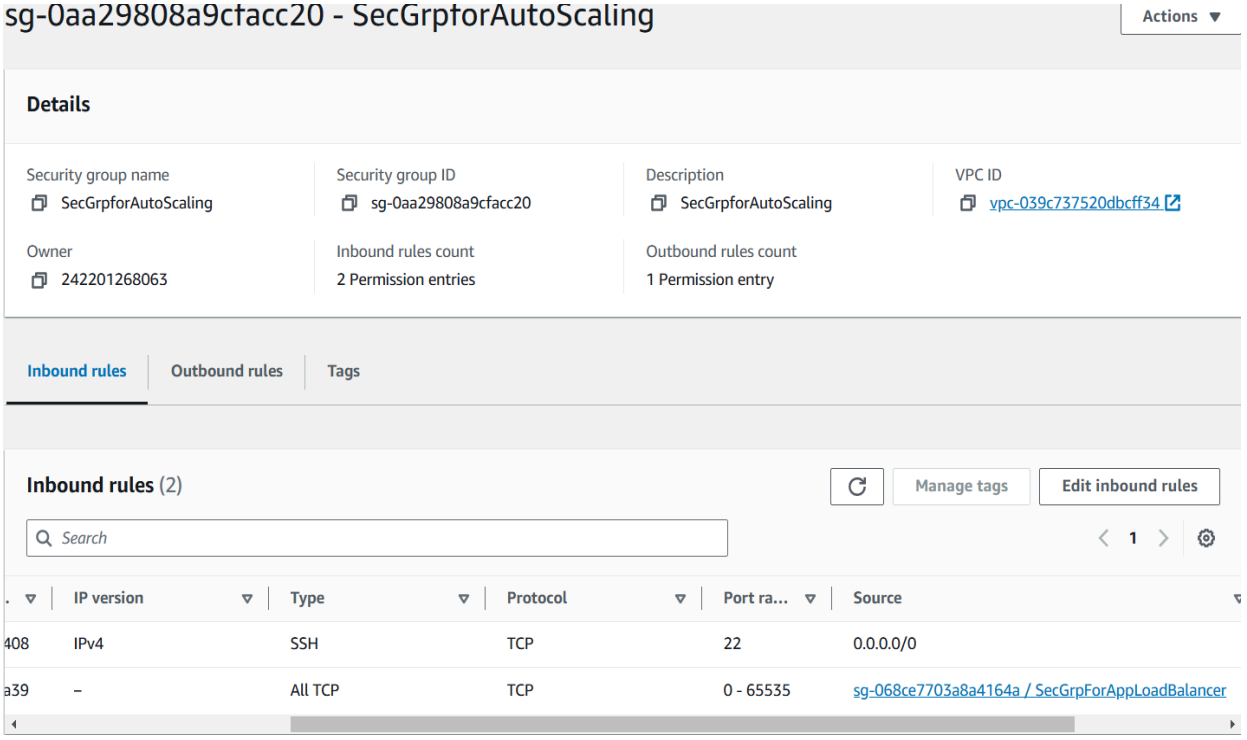


b. Creating Security group

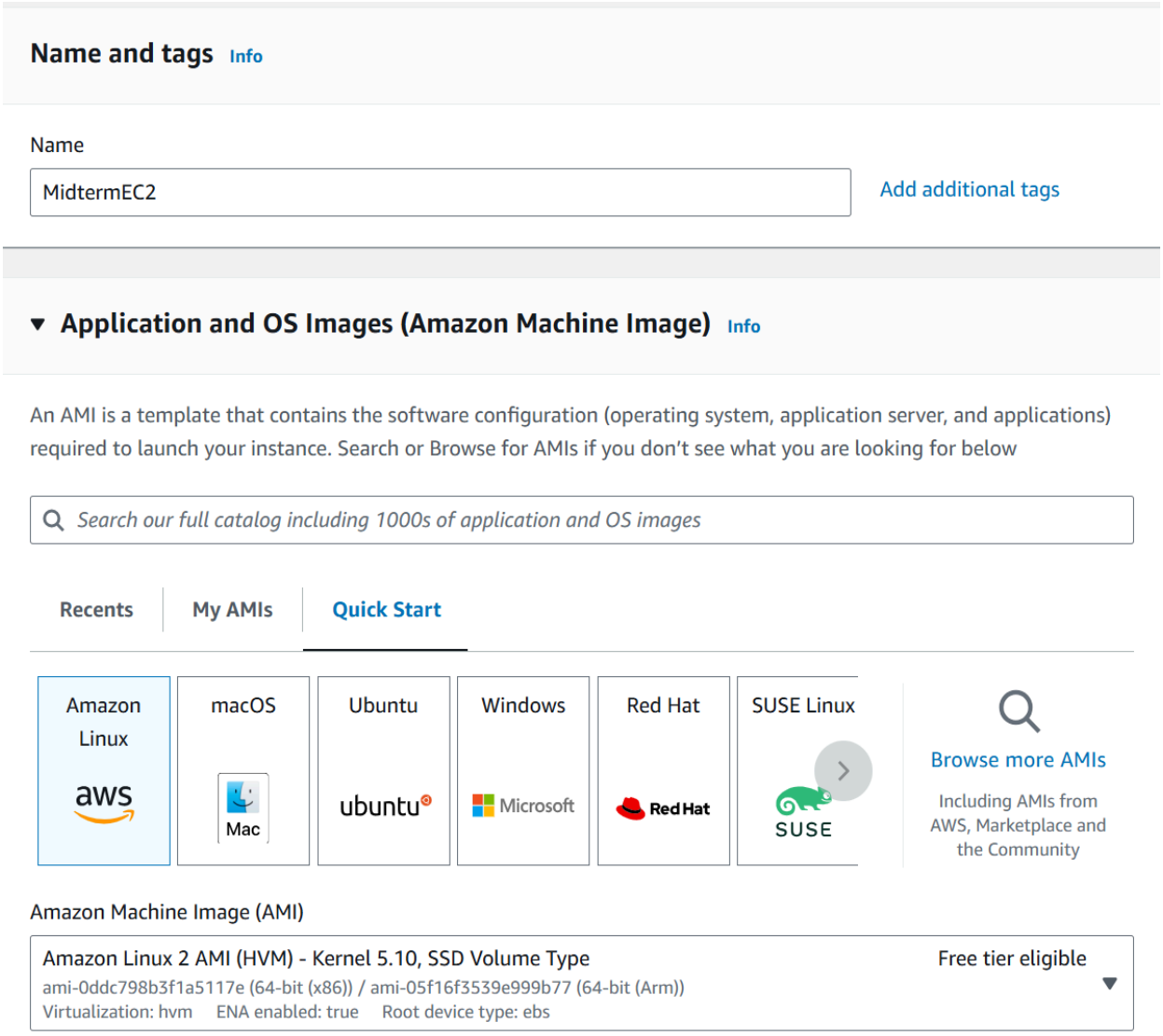
i. **For EC2** to allow HTTP, HTTPS and SSH traffic:
This would enable the httpd and mysql connection testing.



ii. **For Auto Scaling**:
This security group is designed to ingest traffic only from load balancer. This helps secure unwanted access from internet.

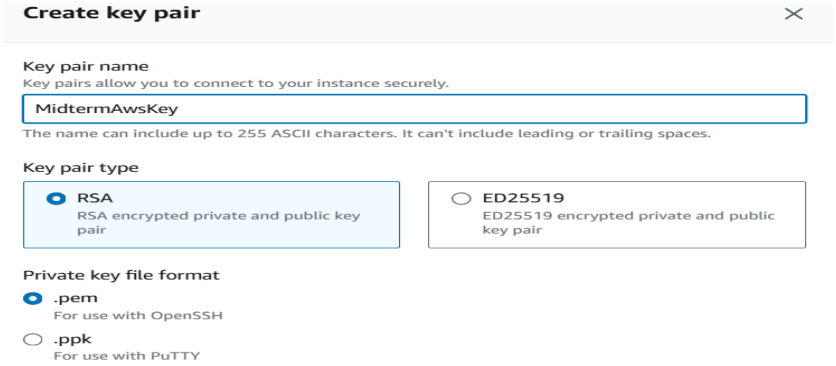


c. EC2 Instances:
   i. Create an EC2 instance using “Midterm-VPC”.

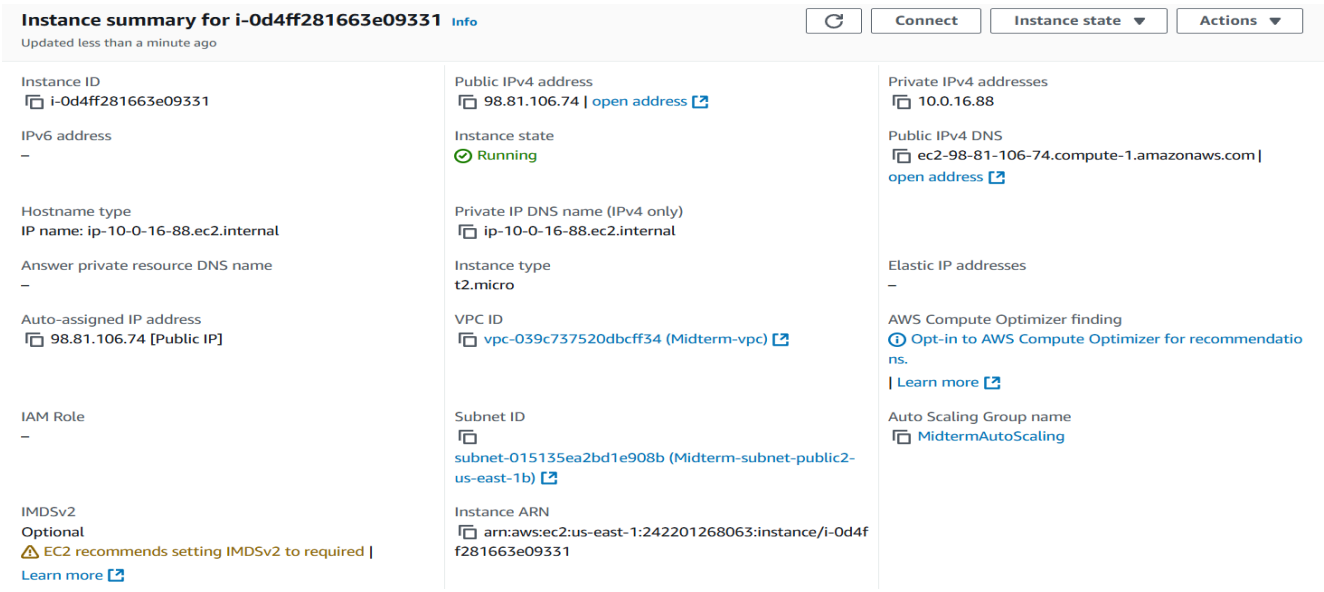


All of the selection made during the configuration of EC2 are done to save cost by using the free tier resources as much as possible. Since the scale can be handled using the Auto Scaling feature presented by AWS.

ii. Create new kay-pair.



iii. Launch the instance.

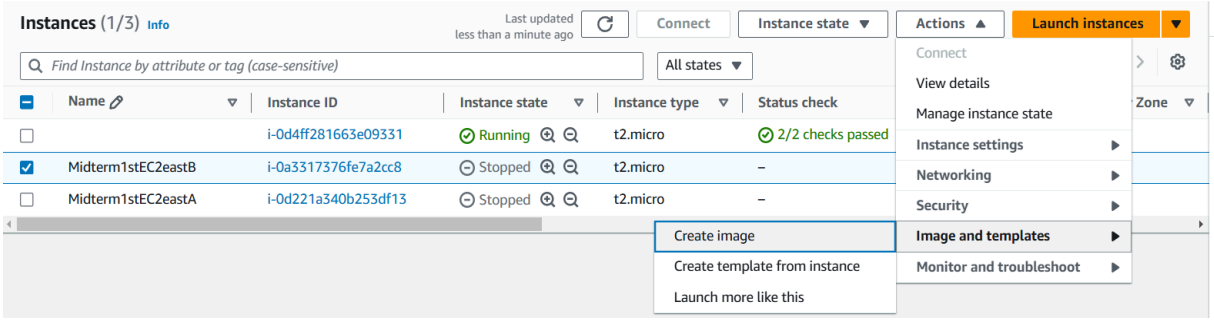


iv. Installing necessary software packages.

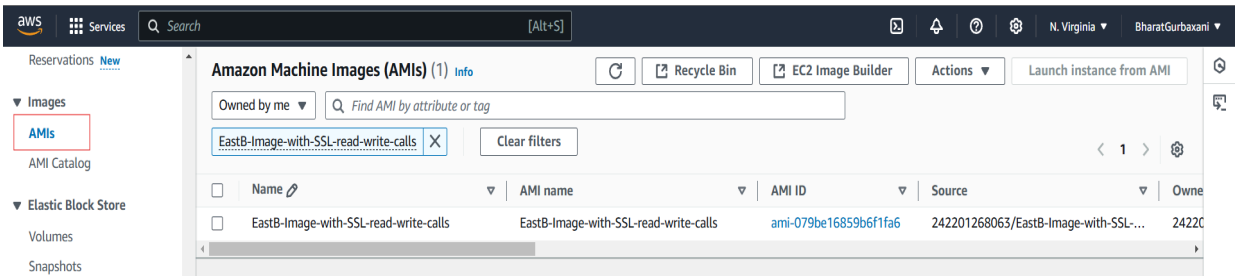
1. Commands used:
    a. ssh -i “MidtermAwsKeys.pem ec2-user@<ec2-public-IP>
    b. sudo yum update -y
    c. sudo yum install httpd -y
    d. sudo systemctl start httpd
    e. sudo systemctl enable httpd
    f. sudo amazon-linux-extras enable php8.0
    g. sudo yum clean metadata
    h. sudo yum install -y php-cli php-fpm php-opcache php-common
    i. sudo yum install git -y
    j. git clone https://github.com/Jocelyn1267/818N-E_Commerce_Application.git
    k. cd 818N-E_Commerce_Application
    l. sudo mv * /var/www/html/
    m. scp -i MidtermAwskey.pem us-east-1-bundle.pem ec2-user@<ec2-public-IP>

d. Creating Image for AutoScaling Configuration

i. Select “Actions” for the desired EC2 instance.



ii. Validate the AMI, under the images tab on the left.

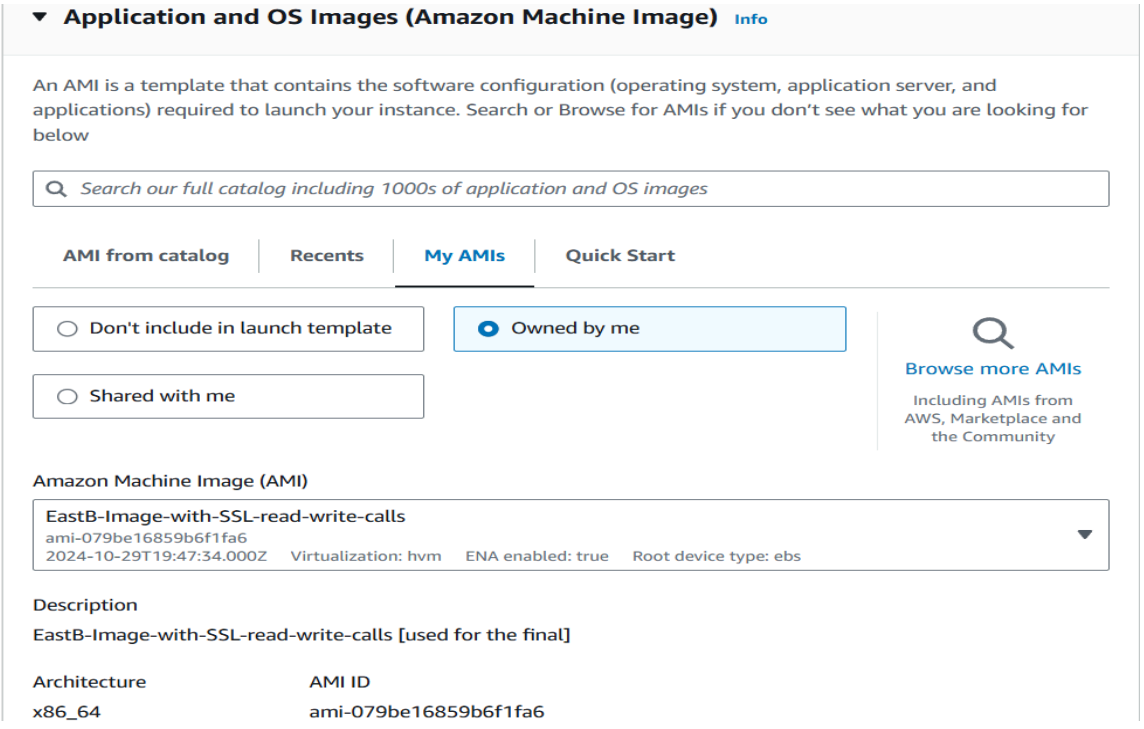


e. Configuring Auto Scaling

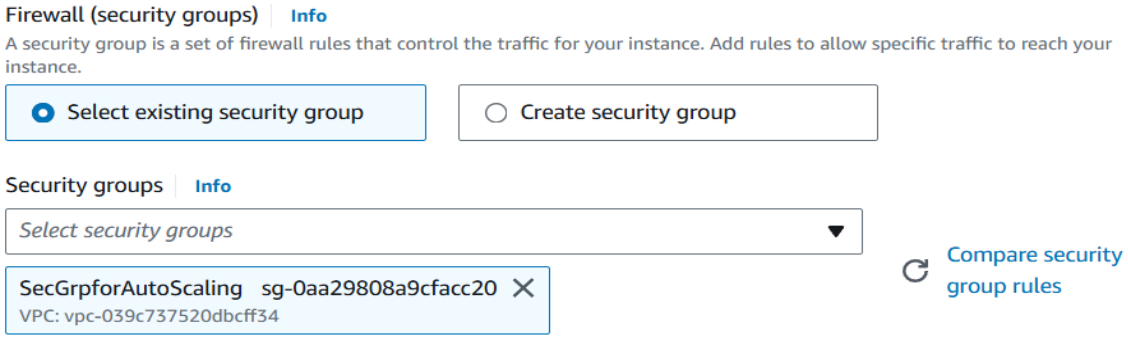
i. Creating launch template

Key steps include:

1. Selecting the AMI image generated in the last step.



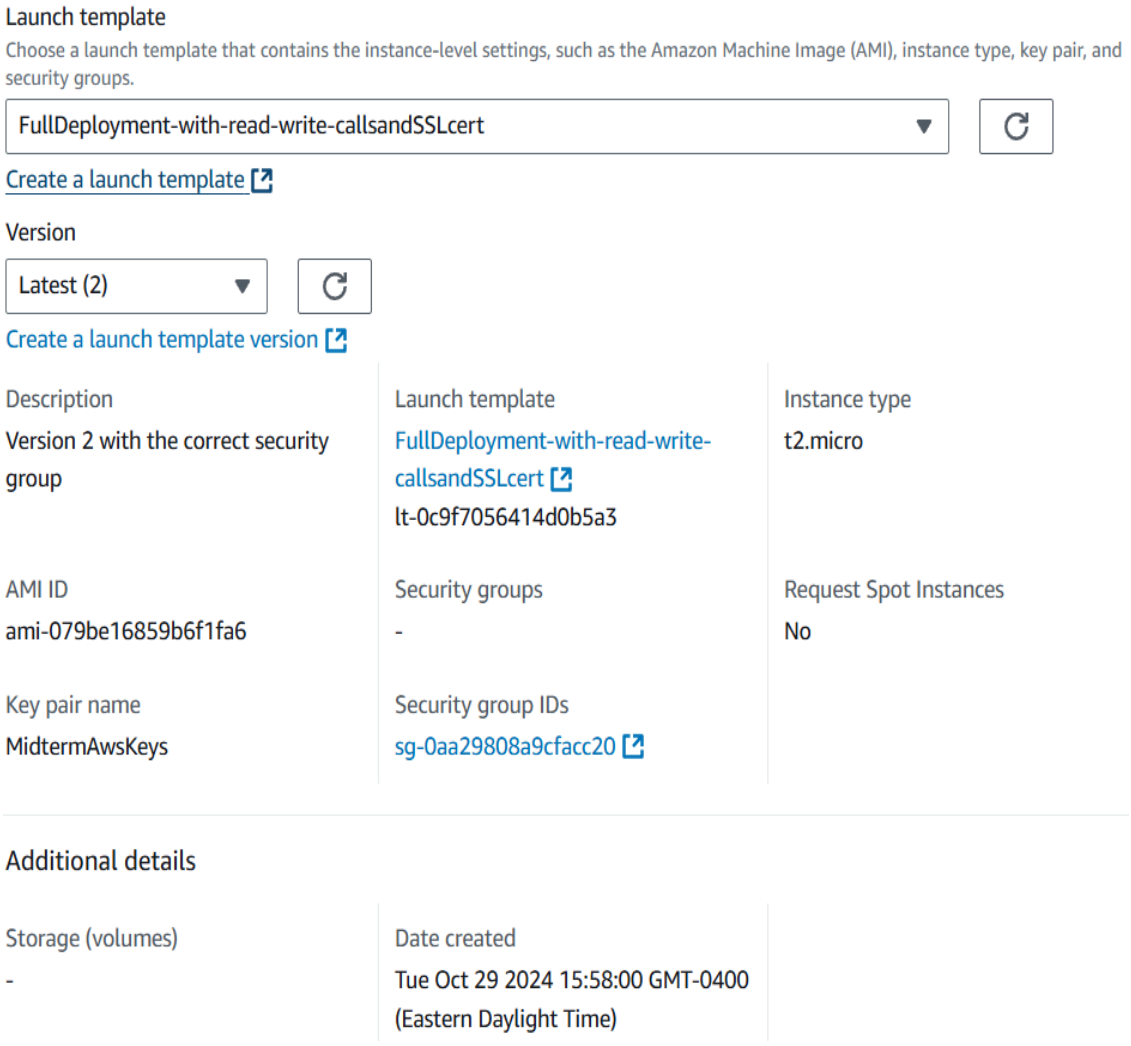
2. Assigning the correction security group.



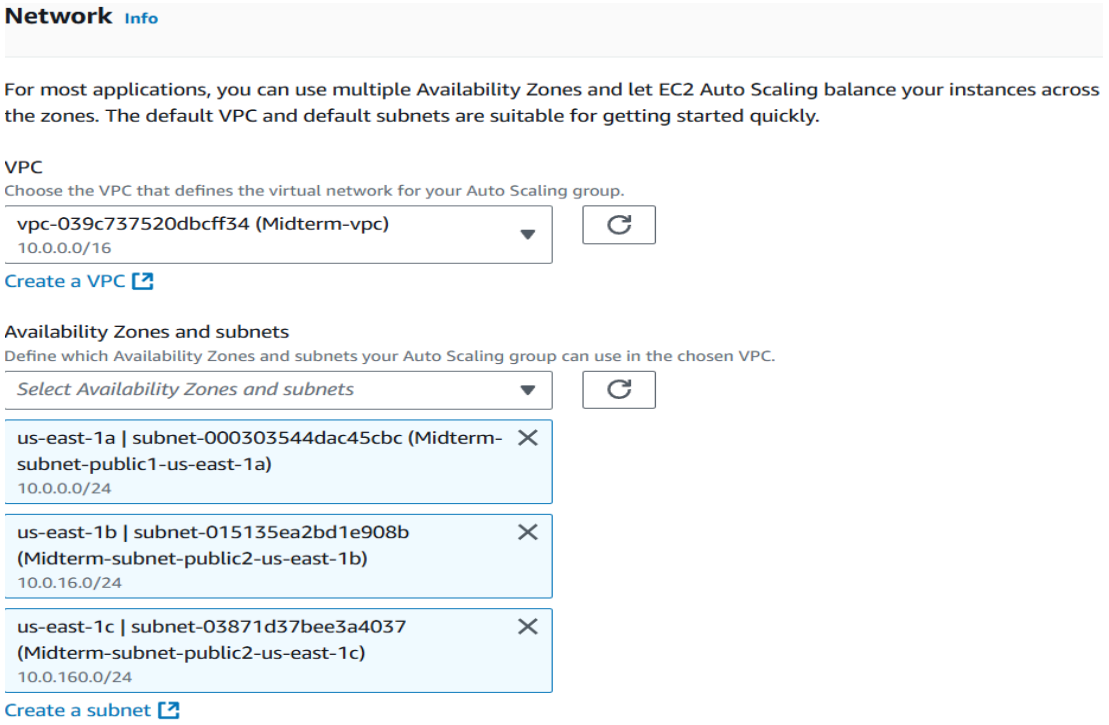
ii. Configuring Auto scaling group & Application load balancer:

Key steps include:

1. Select the configured “Launch Template”

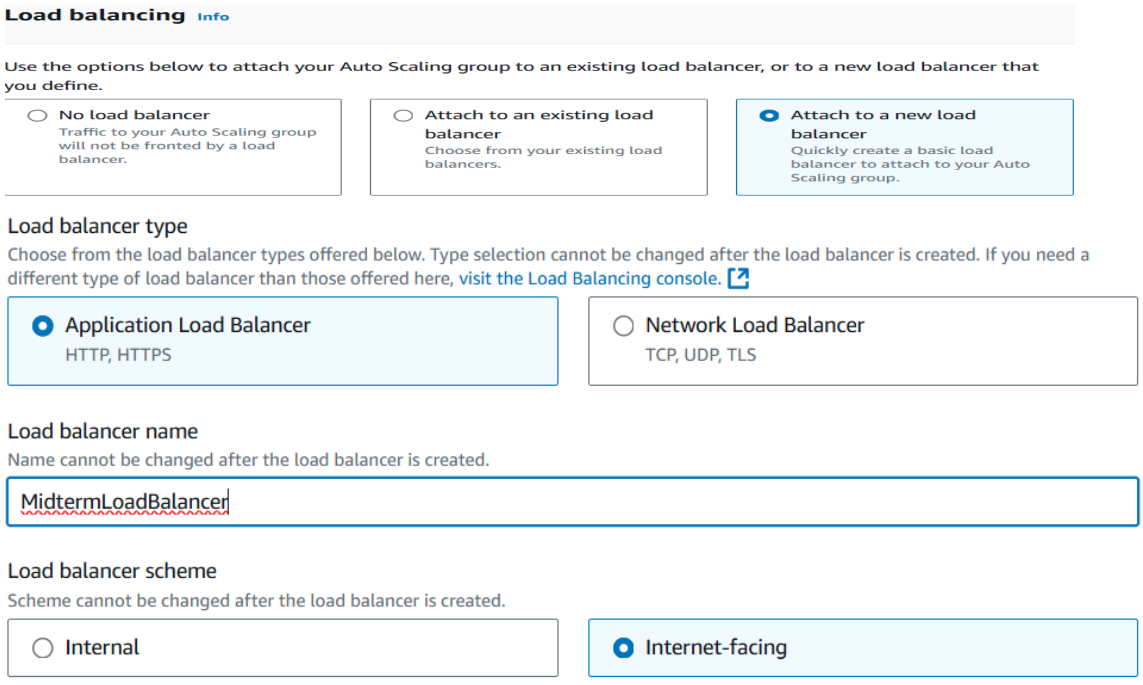


2. Select the VPC created for the assignment, along with availability zones where you want instances to be deployed.



Adding multiple availability zones ensures that your infrastructure is highly available and reduces the latency. This also improves user experience. Although, there can be diminishing returns if too many AZs are used since the infrastructure is still not fault tolerant since all AZs are from the same region.

3. Create application load balancer by selecting the following options:



Network mapping

Your new load balancer will be created using the same VPC and Availability Zone selections as your Auto Scaling group. You can select different subnets and add subnets from additional Availability Zones.

VPC

vpc-039c737520dbcff34 Midterm-vpc

Availability Zones and subnets

You must select a single subnet for each Availability Zone enabled. Only public subnets are available for selection to support DNS resolution.

us-east-1b	subnet-015135ea2bd1e908b
us-east-1a	subnet-000303544dac45cbc
us-east-1c	subnet-03871d37bee3a4037

Listeners and routing

If you require secure listeners, or multiple listeners, you can configure them from the Load Balancing console after your load balancer is created.

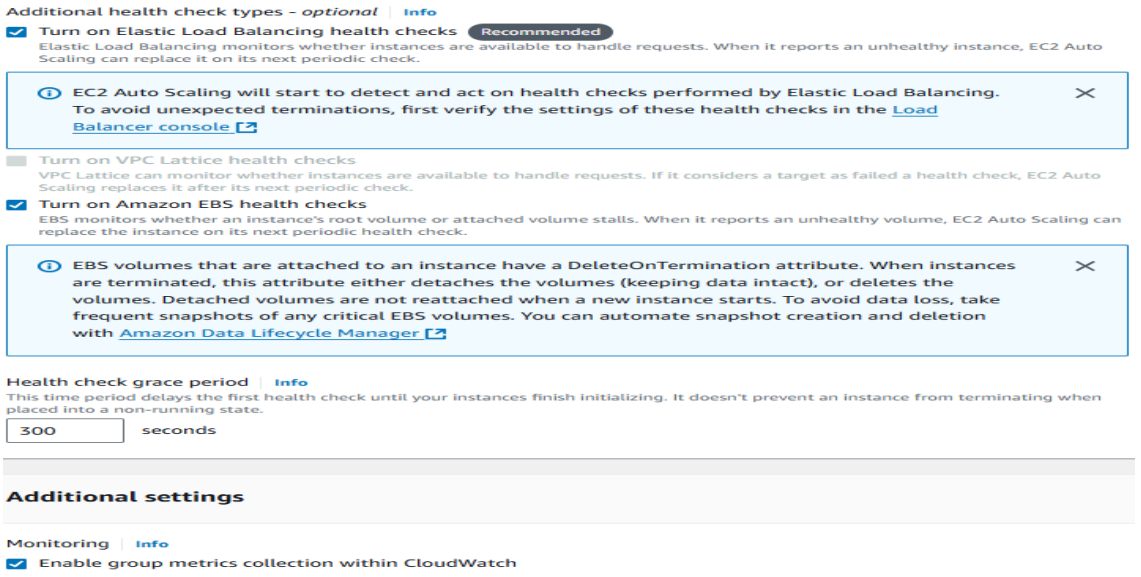
Protocol	Port	Default routing (forward to)
HTTP	80	Create a target group

New target group name

An instance target group with default settings will be created.

MidtermLoadBalancer

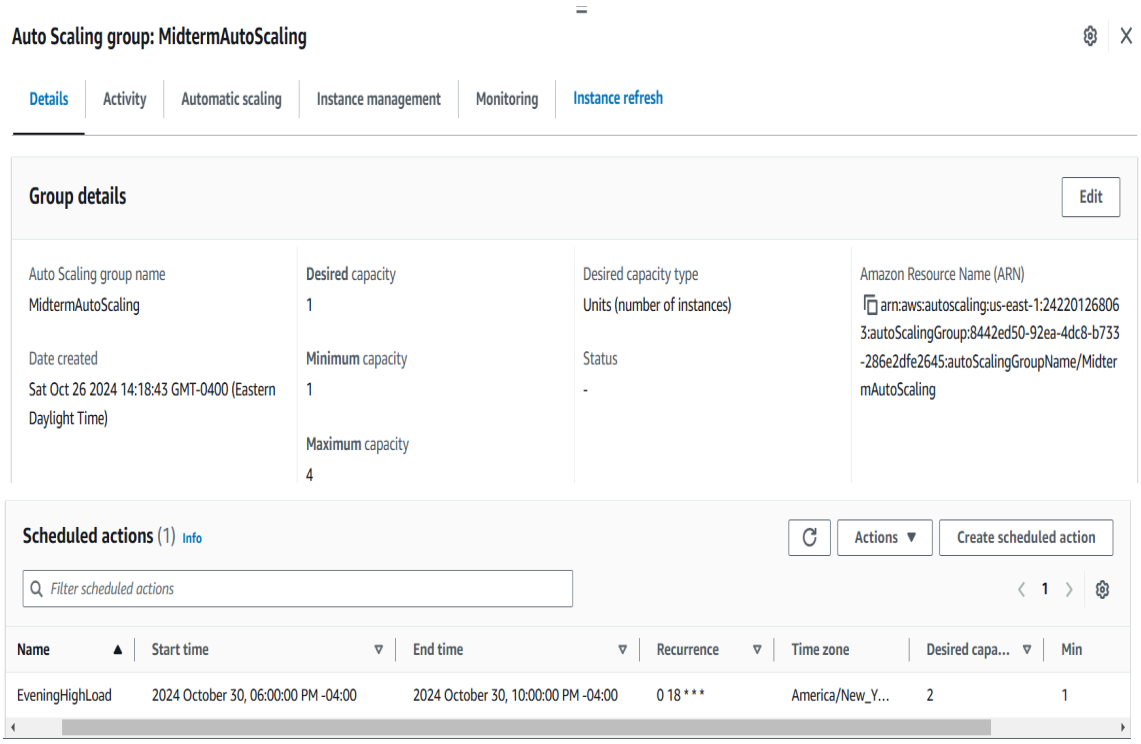
Since we opted for three AZs during the launch template creation, we need to make sure that we pass the same information to load balancer. This way we can make sure all the instances receive traffic and traffic is handled efficiently.



Make sure to enable health check and monitoring, this will later enable us to monitor all attributes of the architecture from a single dashboard.

Add the desired and range of instances that you would like to be deployed and define rules for the same.

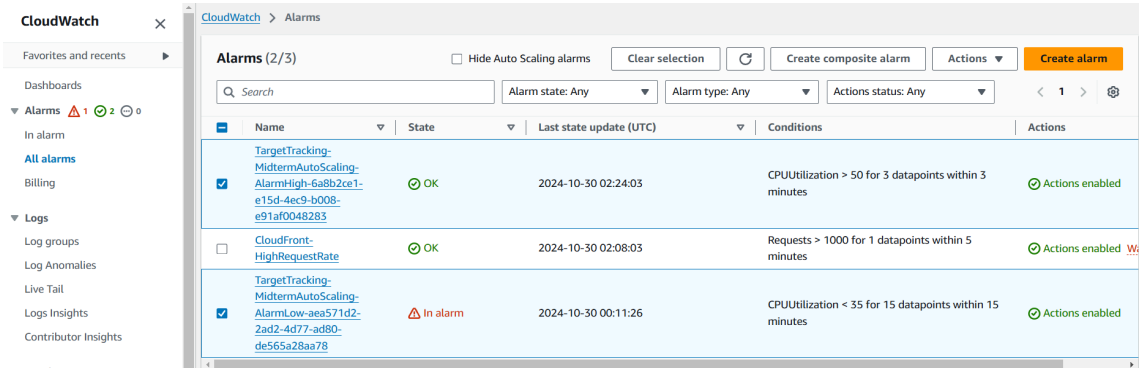
4. Define rules for auto scaling



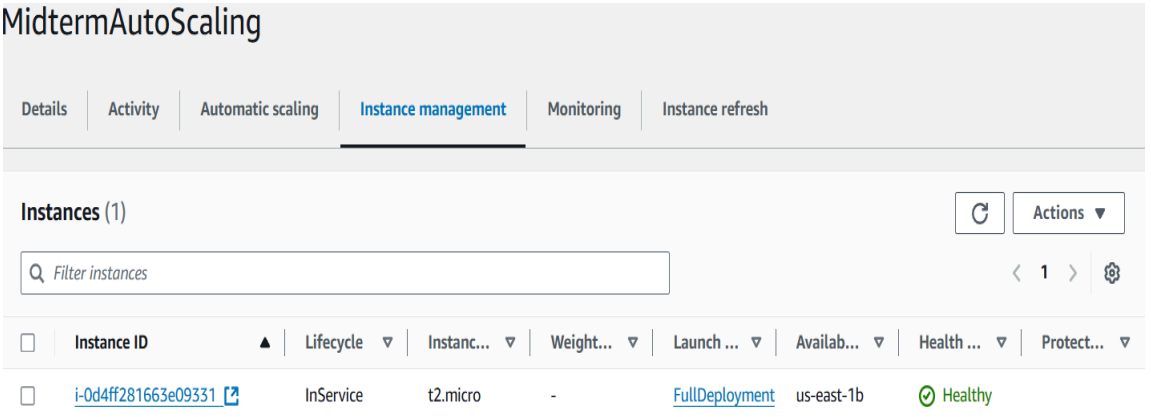
Its notice that the e-commerce platform gets their majority traffic in the evening therefore this policy is to preemptively handle that load before the system is under stress.

These rules look for the stress on the EC2 instances running and trigger cloudwatch alarm based on the set conditions.

5. Cloud watch Alarm:

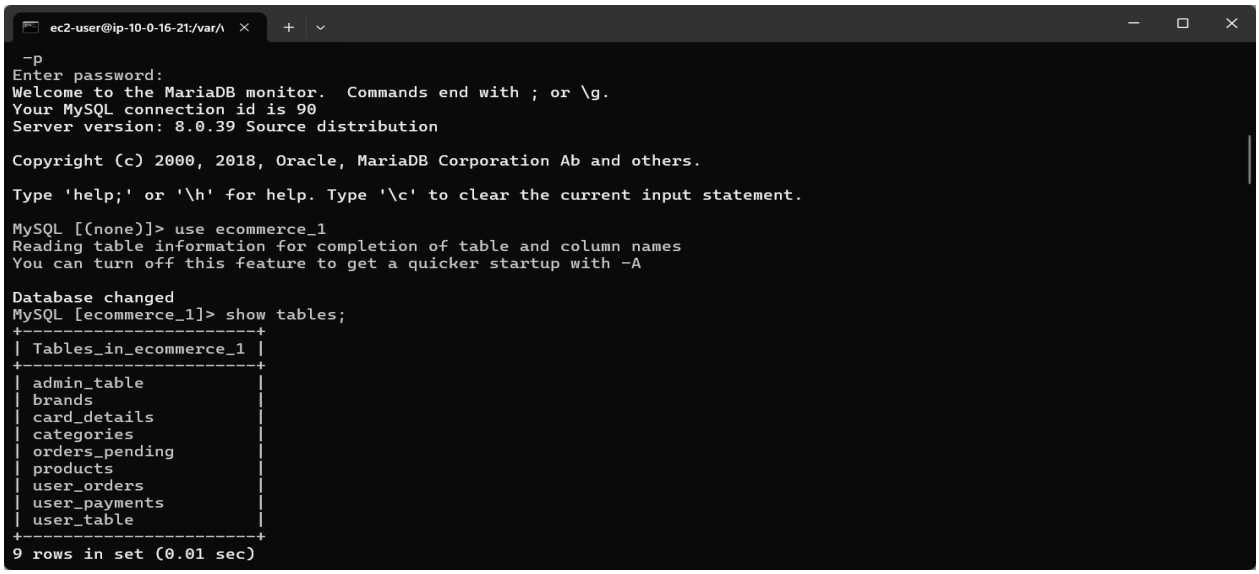


The status “in alarm” acts as a trigger for the Auto-scaling group to take action.



2. Database Setup

Our Amazon RDS configuration was pivotal in establishing a secure, high-availability backend for managing the e-commerce platform's critical data, including product information, user accounts, and order details.

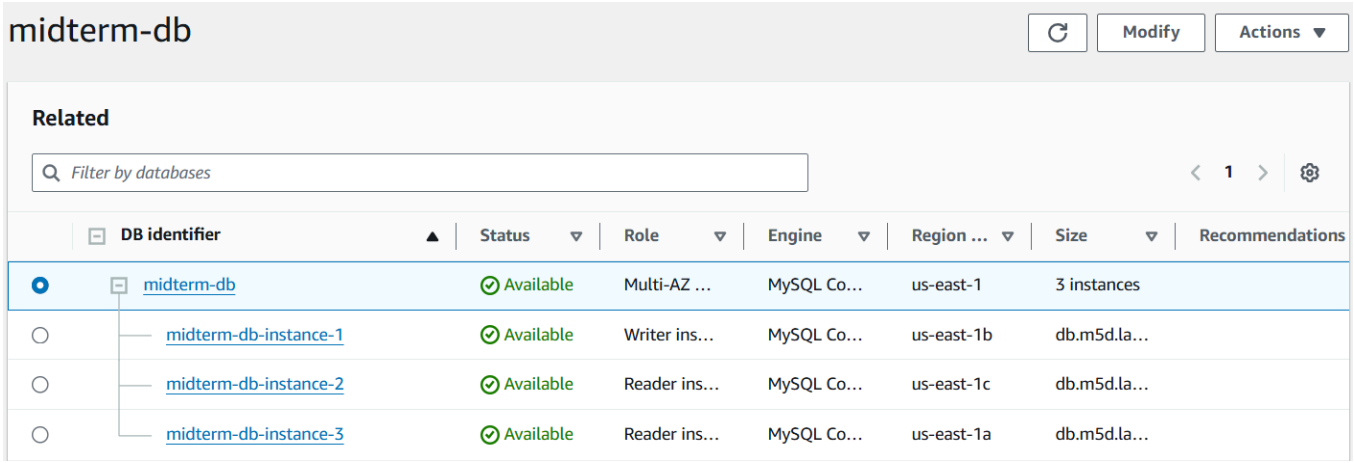


a. Initial Setup with a Single RDS Instance

To validate the database functionality and configurations, we first deployed a single RDS instance. This initial setup allowed us to ensure that all database-related operations, queries, and schema requirements were correctly implemented, laying a reliable foundation before expanding to a Multi-AZ architecture.

b. Multi-AZ Deployment for High Availability

After verifying that the single-instance configuration functioned well, we upgraded to a Multi-AZ RDS instance to enhance fault tolerance and minimize potential downtime. This Multi-AZ configuration provided synchronous replication across availability zones, ensuring seamless failover capabilities in the event of hardware failures or other disruptions. The setup included separate read and write endpoints to optimize load balancing and performance across the application.



Configuration

DB cluster ID
midterm-db

DB cluster role
Multi-AZ DB cluster

Engine version
8.0.39

RDS Extended Support
Disabled

Amazon Resource Name (ARN)
arn:aws:rds:us-east-1:242201268063:cluster:midterm-db

Resource ID
cluster-ZDTXUD3QGKZHUPJVZGVAXKFQYA

Created time
October 29, 2024, 14:17 (UTC-04:00)

DB instance parameter group
default.mysql8.0

Instance class

Instance class
db.m5d.large

vCPU
2

RAM
8 GB

Instance Store Info
Instance storage: 75 GiB NVMe SSD

Availability

Master username
admin

Master password
****

IAM DB authentication
Not enabled

Multi-AZ
3 Zones

Storage

Encryption
Enabled

AWS KMS key
midterm-db-cmk

Storage type
General Purpose SSD (gp3)

Storage
20 GiB

Provisioned IOPS
3000 IOPS

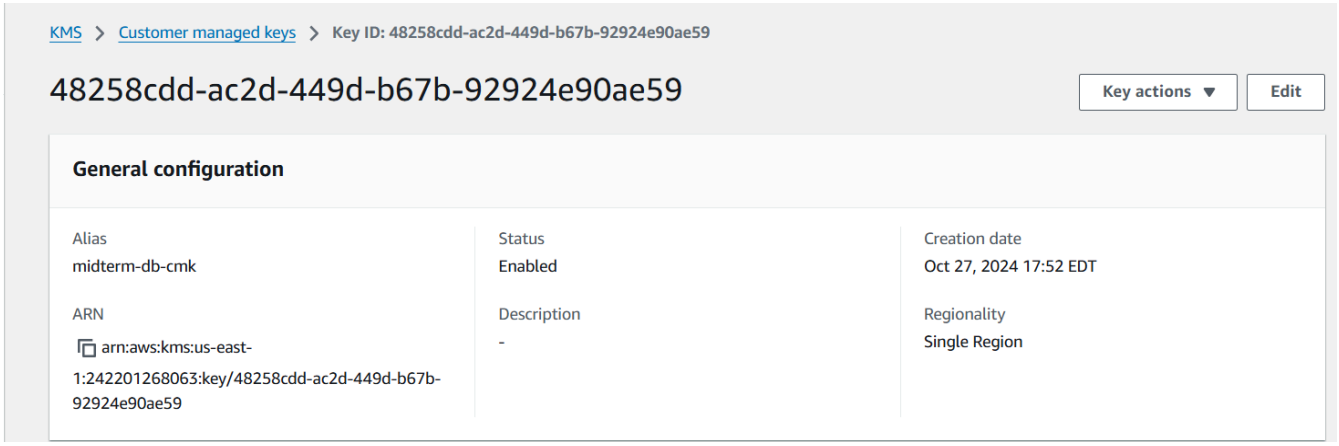
Storage throughput
125 MiBps

Storage autoscaling
Disabled

Performance Insights

Performance Insights enabled
Turned off

We implemented data encryption at rest using Amazon Key Management Service (KMS) with a customer-managed key (midterm-db-cmk). This approach not only safeguarded sensitive information but also ensured compliance with industry data protection standards by restricting unauthorized access. Access to the database and encryption keys was further protected by implementing IAM role-based policies, allowing only authorized users to manage database access and encryption key permissions, thereby reinforcing data security.



Transitioning to a Multi-AZ RDS cluster brought some architectural differences compared to the single-instance setup.The Multi-AZ configuration provides dedicated endpoints for read and write operations,which is very different from the single DB instance. Consequently, we updated the connection file (connect.php) and all relevant PHP files within the application. This adjustment enabled efficient utilization of the RDS's read-only endpoint for data retrieval and the primary endpoint for transactional operations, enhancing load distribution and reducing latency.

There is a small modification that needs to be mentioned is that the provided SQL files did not define primary keys in table creation statements at the first stage, which led to compatibility issues when migrating to the Multi-AZ setup. We

modified these SQL scripts to define primary keys where necessary, resolving schema integrity requirements and ensuring optimal performance and query accuracy.

connect.php

```php
<?php

// Specify the path to the SSL CA file
$ssl_ca = '/home/ec2-user/us-east-1-bundle.pem';

// Write connection (primary endpoint)
$con = new mysqli('midterm-db.cluster-c9466gmgsshe.us-east-1.rds.amazonaws.com', 'admin', 'password', 'ecommerce_1');
if ($con->connect_error) {
    die("Write Connection failed: " . $con->connect_error);
}

// Enable SSL for the write connection
$con->ssl_set(NULL, NULL, $ssl_ca, NULL, NULL);
$con->real_connect(
    'midterm-db.cluster-c9466gmgsshe.us-east-1.rds.amazonaws.com',
    'admin',
    'password',
    'ecommerce_1'
);

// Read connection (read-only endpoint)
$readCon = new mysqli('midterm-db.cluster-ro-c9466gmgsshe.us-east-1.rds.amazonaws.com', 'admin', 'password', 'ecommerce_1');
if ($readCon->connect_error) {
    die("Read Connection failed: " . $readCon->connect_error);
}

// Enable SSL for the read connection
$readCon->ssl_set(NULL, NULL, $ssl_ca, NULL, NULL);
$readCon->real_connect(
    'midterm-db.cluster-ro-c9466gmgsshe.us-east-1.rds.amazonaws.com',
    'admin',
    'password',
    'ecommerce_1'
);

?>
```

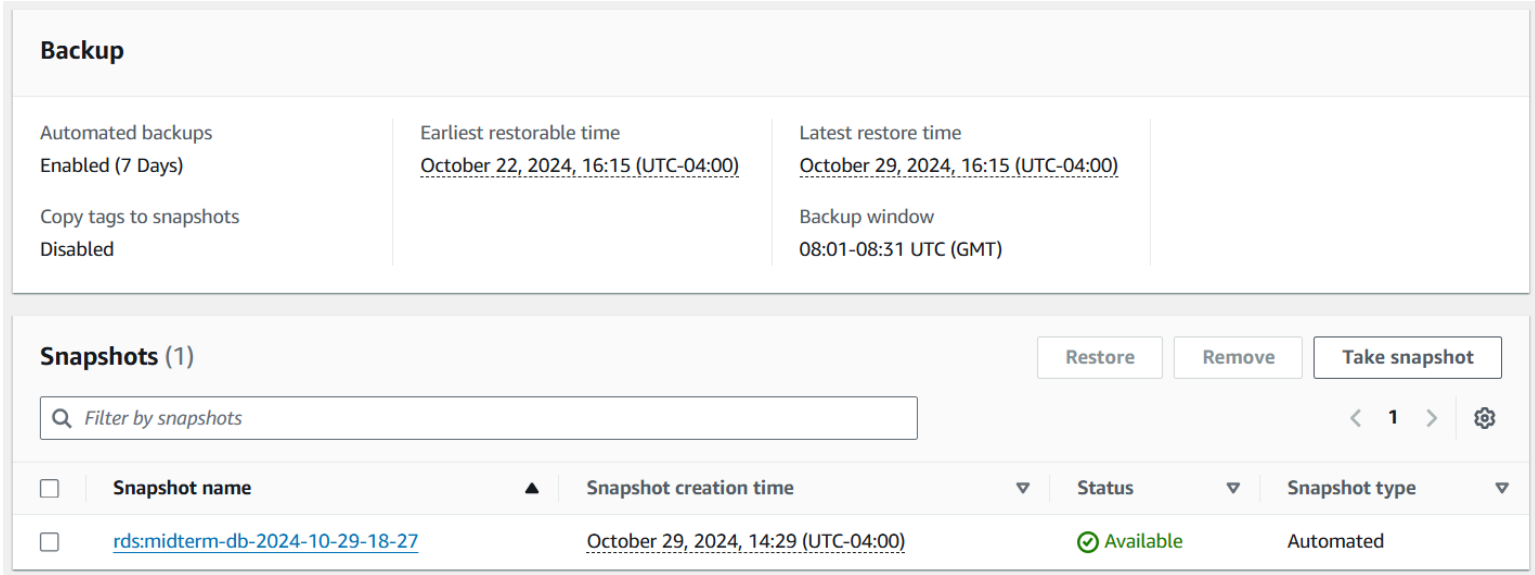
c. Securing Data in Transit with SSL/TLS

To further protect data as it moves between the web application and the RDS instance, we enabled SSL/TLS encryption. This encryption ensures that sensitive information, such as user credentials and transaction details, remains protected against unauthorized interception during transmission. The SSL configuration is defined in the `connect.php` file, as shown above, using a region-specific CA certificate (`us-east-1-bundle.pem`) to authenticate and secure each database connection.

```
[ec2-user@ip-10-0-16-21 ~]$ mysql -h midterm-db.cluster-c9466gmgsshe.us-east-1.rds.amazonaws.com -u admin -p --ssl-ca=/h
ome/ec2-user/us-east-1-bundle.pem --ssl
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 88
Server version: 8.0.39 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MySQL [(none)]> SHOW STATUS LIKE 'Ssl_cipher';
+---------------+----------------------------+
| Variable_name | Value                      |
+---------------+----------------------------+
| Ssl_cipher    | ECDHE-RSA-AES256-GCM-SHA384 |
+---------------+----------------------------+
1 row in set (0.00 sec)
```

d. Automated Backups and Recovery

To support data continuity, we configured automated daily backups with a point-in-time recovery option. This backup configuration allows the database to be restored to any point within the retention period, mitigating data loss risks and enhancing platform resilience.

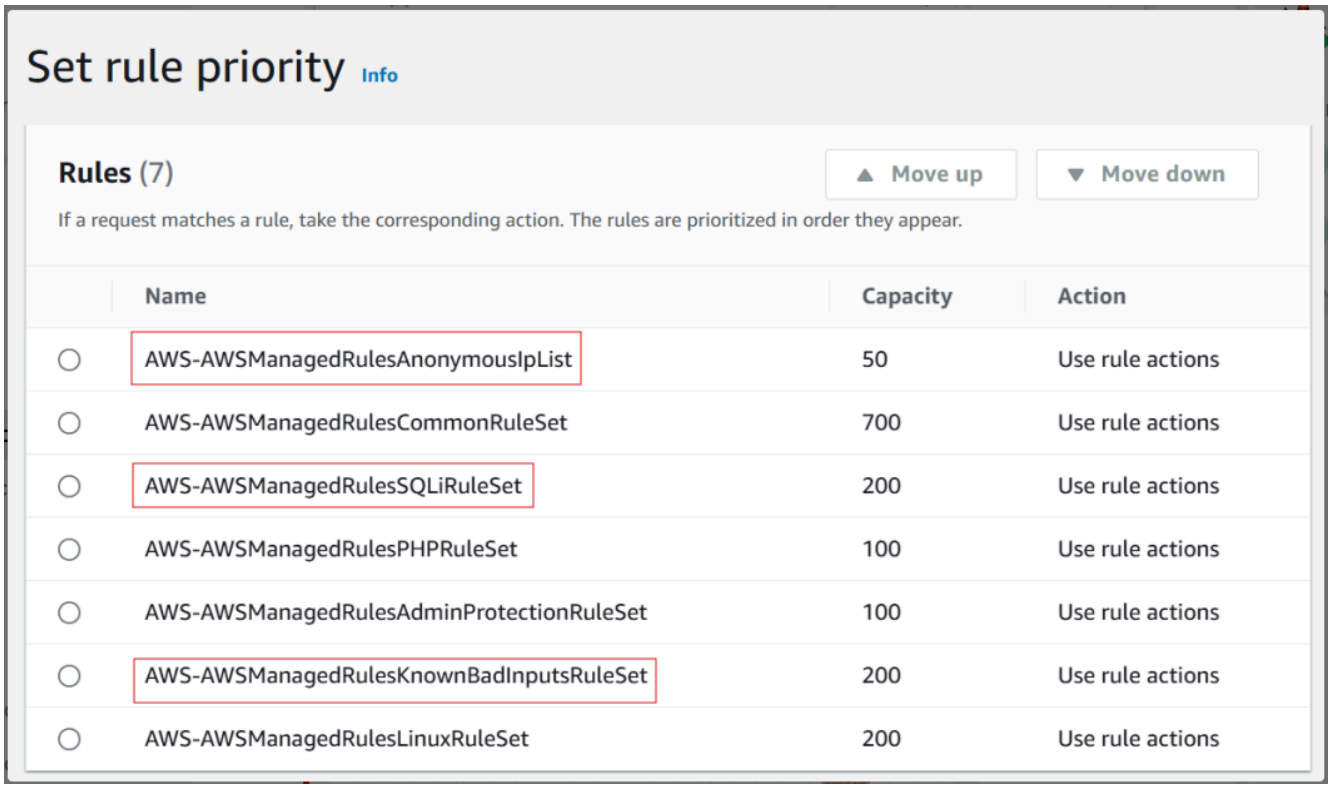


This robust Amazon RDS configuration underpins the e-commerce platform's data management requirements, balancing performance, security, and high availability. By utilizing Multi-AZ deployment, KMS encryption, IAM role-based access, SSL/TLS encryption, and automated backups, the setup meets stringent data protection standards and provides a dependable backend infrastructure for handling critical e-commerce data.

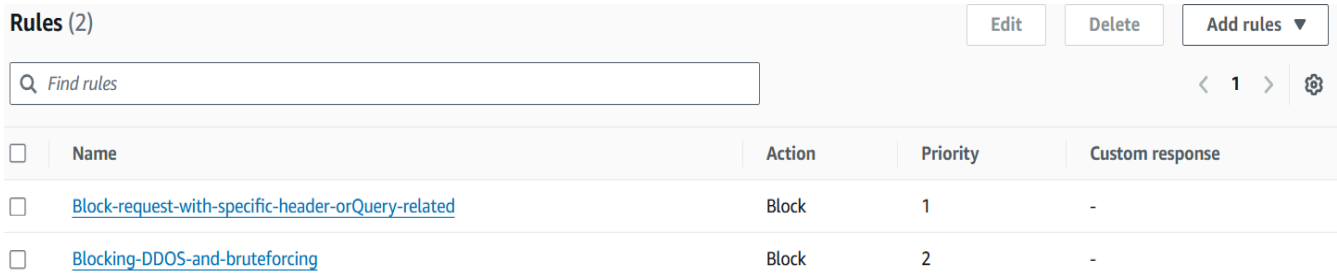
3. Security Measures

3.1 AWS WAF:

To protect from the host of commonly known attacks like SQLi, XSS and DDOS attacks, we needed to deploy the WAF. To achieve this following rules were added for WAF to implement (highlighted below).

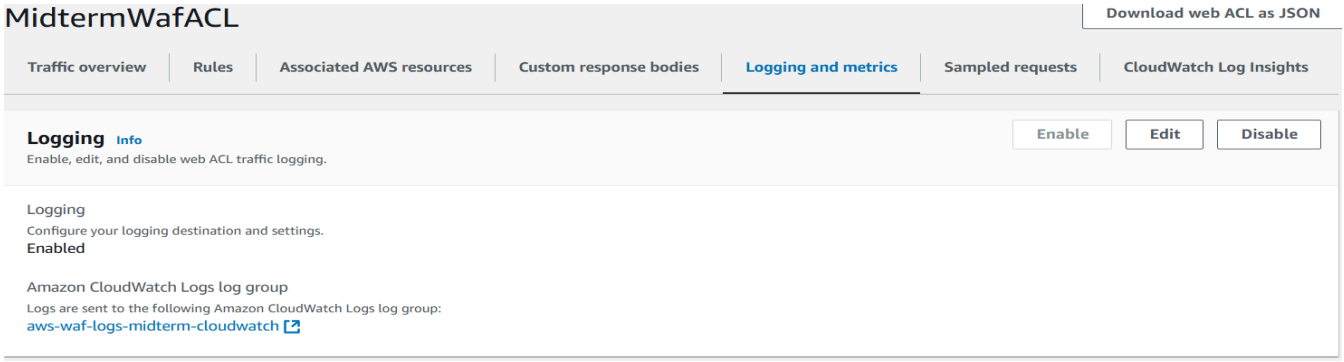


Additionally, 2 custom rules were added to make sure that no query or packets with specific header indicating attack-like scenarios can pass through.

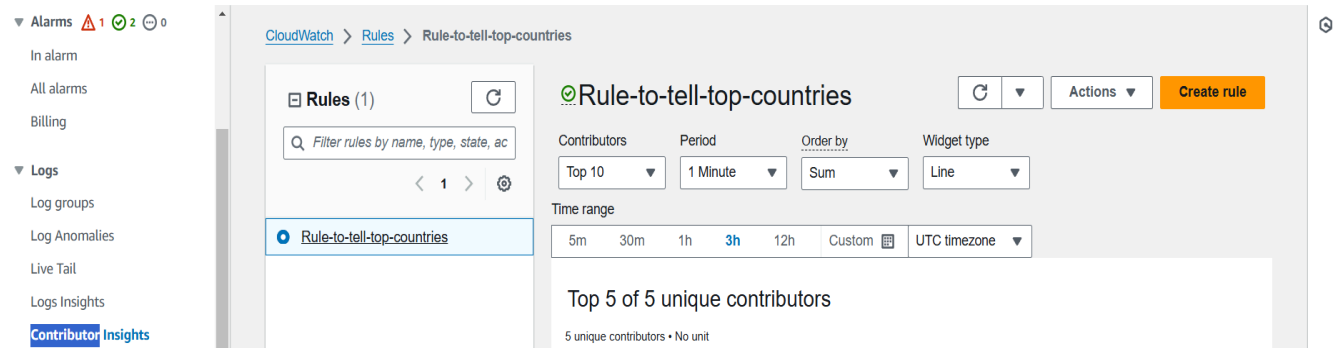


In addition to the above rule, we also made a decision to deploy a static web page(CDN) before the ecommerce application. This helps in avoiding a number of generic automated attacks which are launched over a big part of the internet to get high-value targets.

Since we can now block such requests, we also enabled the logging feature in cloudwatch based on the WAF interactions. Cloud watch can be used to parse through the WAF logs and understand the approval and rejection of the connections in much more granularity.



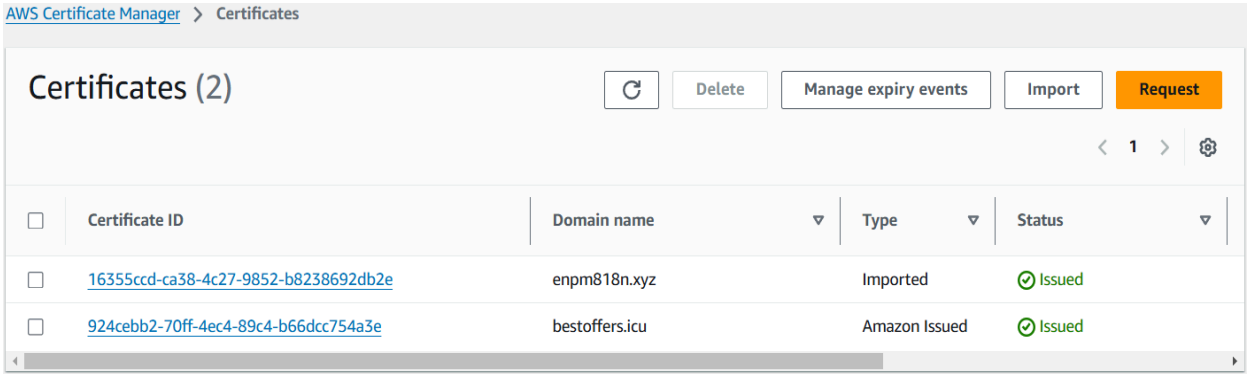
In addition to that, it can also offer Contributor insights where results can be generated based on the log analysis performed for a predefined criteria. For example, we have added Contributor insight to understand the countries where data is coming from.



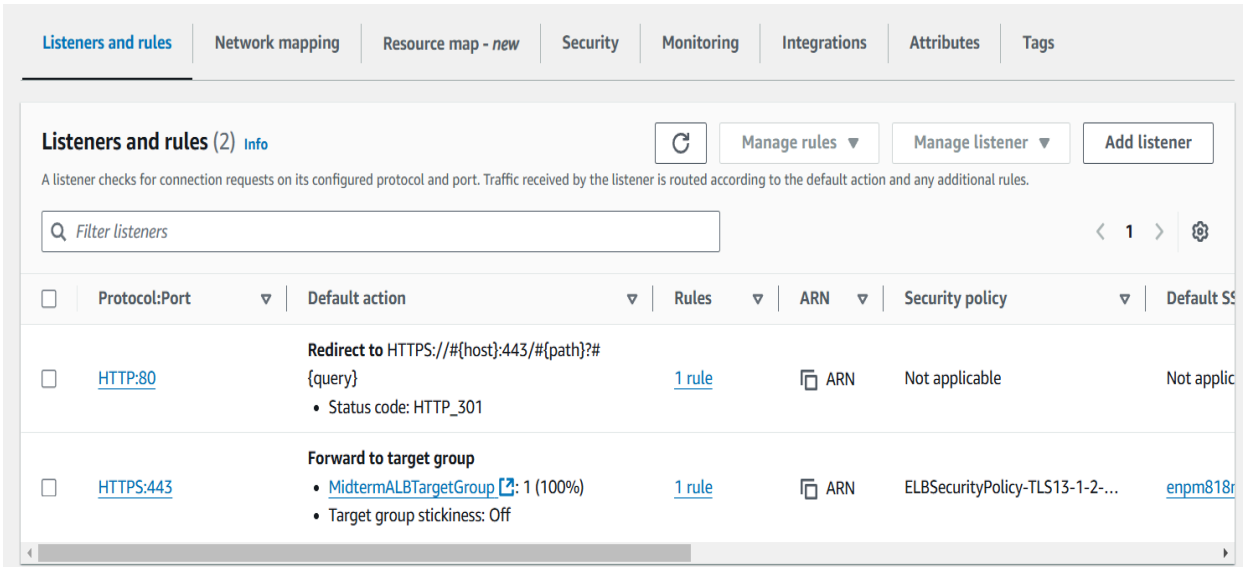
#	$.httpRequest.country	Sum of data points
1	US	18
2	SA	5
3	RS	3
4	DE	1
5	NL	1

3.2 Encryption:

- Between Client and WebServer:
  This can be implemented with the help of AWS CMS, which can generate certificates for the domain that is being used. We enable HTTPS to secure the flow of information between client and WebServer.

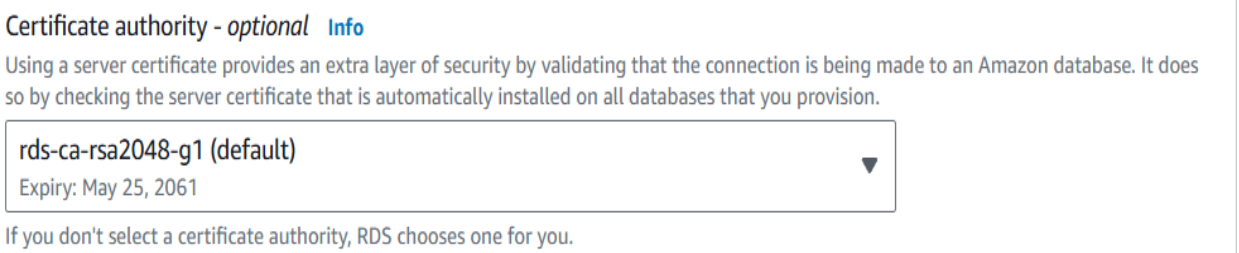


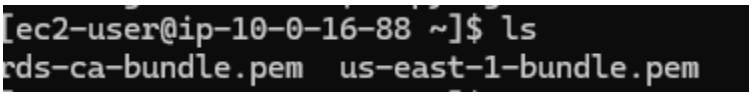
To accept the HTTPS traffic, we need to make sure that the HTTPS listener is active for load balancer. Also, redirecting HTTP traffic to HTTPS, to enforce safer communication.



- Between Webserver and RDS

SSL can be enabled for this communication channel with the help of certificate authority. It can generate a key-pair that can be used to connect over SSL. Thus enforcing the encryption on the communication channel.





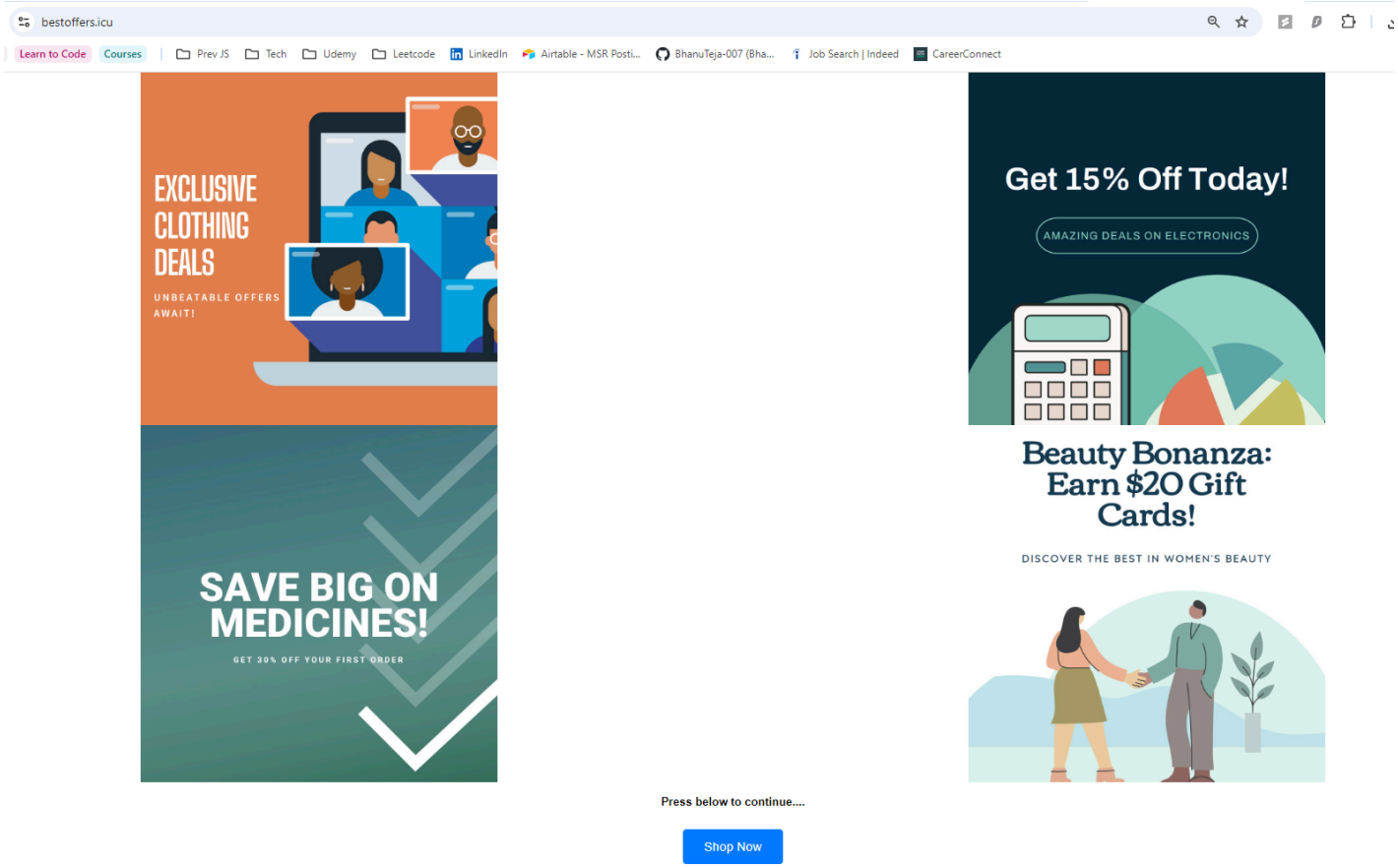
```
[ec2-user@ip-10-0-16-21 ~]$ mysql -h midterm-db.cluster-c9466gmgsshe.us-east-1.rds.amazonaws.com -u admin -p --ssl-ca=/h
ome/ec2-user/us-east-1-bundle.pem --ssl
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 114
Server version: 8.0.39 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

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

MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| ecommerce_1        |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

```
MySQL [ecommerce_1]> show status like 'ssl_cipher';
+---------------+-----------------------------+
| Variable_name | Value                       |
+---------------+-----------------------------+
| Ssl_cipher    | ECDHE-RSA-AES256-GCM-SHA384 |
+---------------+-----------------------------+
1 row in set (0.00 sec)
```

4. Content Delivery

- CloudFront CDN: Detail how CloudFront was set up to accelerate content delivery.

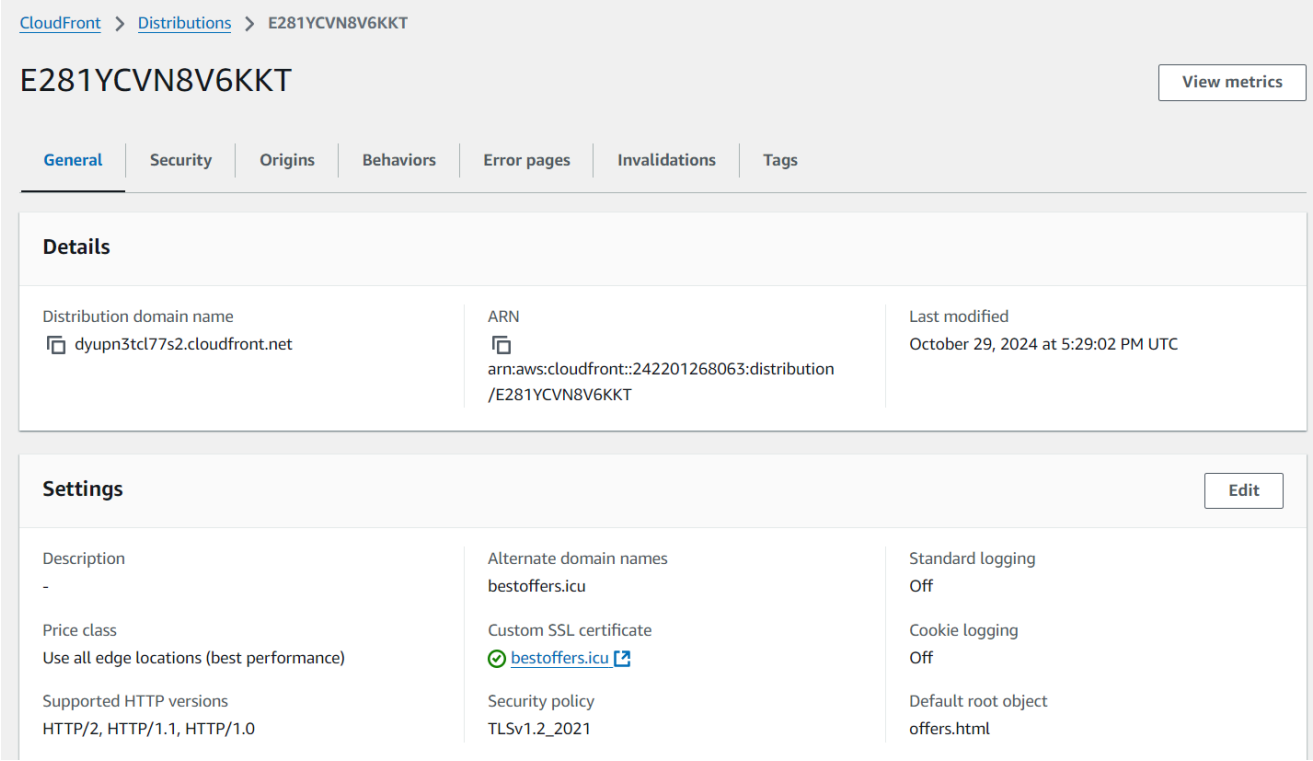
  To improve the speed and security of our website, we implemented a Content Delivery Network (CDN) using AWS CloudFront. The CDN serves as a front-facing layer for our site, allowing us to host a static landing page(https://bestoffers.icu), which acts as a gateway to our main e-commerce site. The landing page includes a "Shop Now" button, guiding users seamlessly to our main site and creating a structured and user-friendly first impression.



We have performed the following steps:

1.CloudFront Distribution

We started by setting up a CloudFront distribution, which essentially distributes our content globally, using Amazon's network of data centers. With this distribution, visitors are served content from the data center closest to them, which reduces load times significantly. This is especially important for static content like landing pages, as it ensures that users can access it quickly from anywhere.

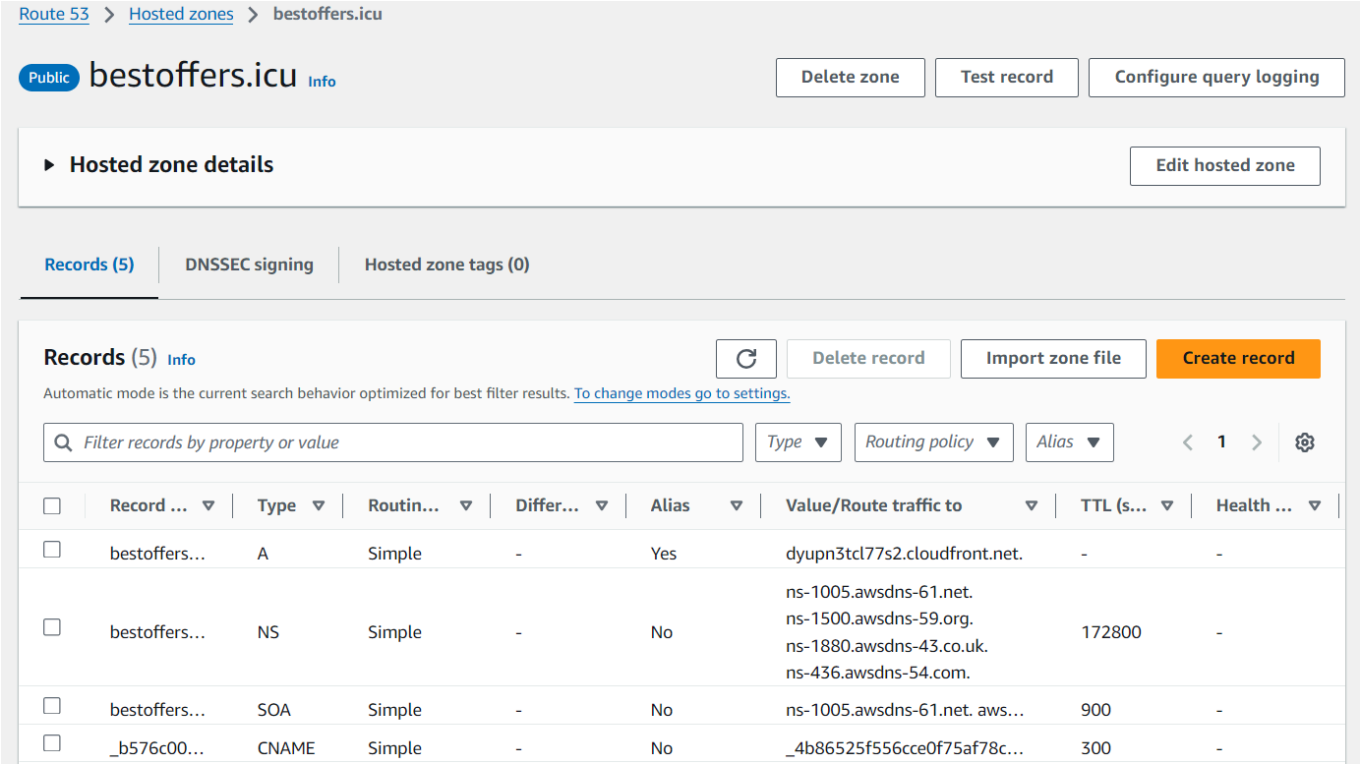


2. SSL Certification

To ensure secure connections to our landing page, we applied an SSL certificate through AWS Certificate Manager. This certificate enables HTTPS, which protects the communication between our users and the CDN. The SSL setup reassures users that our website is trustworthy, as their interactions and data are protected with encryption. It also aligns with industry standards and SEO best practices, as most modern browsers now expect HTTPS for user security.

3. Domain Configuration with Route 53:

For easy accessibility, we wanted users to reach our CDN-backed landing page through a custom domain, ***https://bestoffers.icu***. We used AWS Route 53, Amazon's domain name system (DNS) service, to create a hosted zone and configure the domain settings. By mapping the CloudFront distribution's DNS to **bestoffers.icu**, we ensured that visitors are automatically directed to the CDN-hosted landing page when they enter this URL. This setup provides a professional, branded domain and simplifies navigation for our users.



Redirection to Main Site

To provide a seamless transition from our landing page to the main shopping site, we configured the "**Shop Now**" button to redirect users to ***https://www.enpm818n.xyz***. For this, we created a separate hosted zone within AWS Route 53 specifically for **enpm818n.xyz**. This setup allowed us to manage DNS settings and ensure that traffic is directed appropriately.

We also secured this main domain with an SSL certificate via AWS Certificate Manager, enabling HTTPS for all interactions on **enpm818n.xyz**. By doing so, we maintained security and user trust as they navigate from the landing page to the main shopping experience. This configuration not only creates a smooth user journey but also ensures consistent performance and security across both domains.

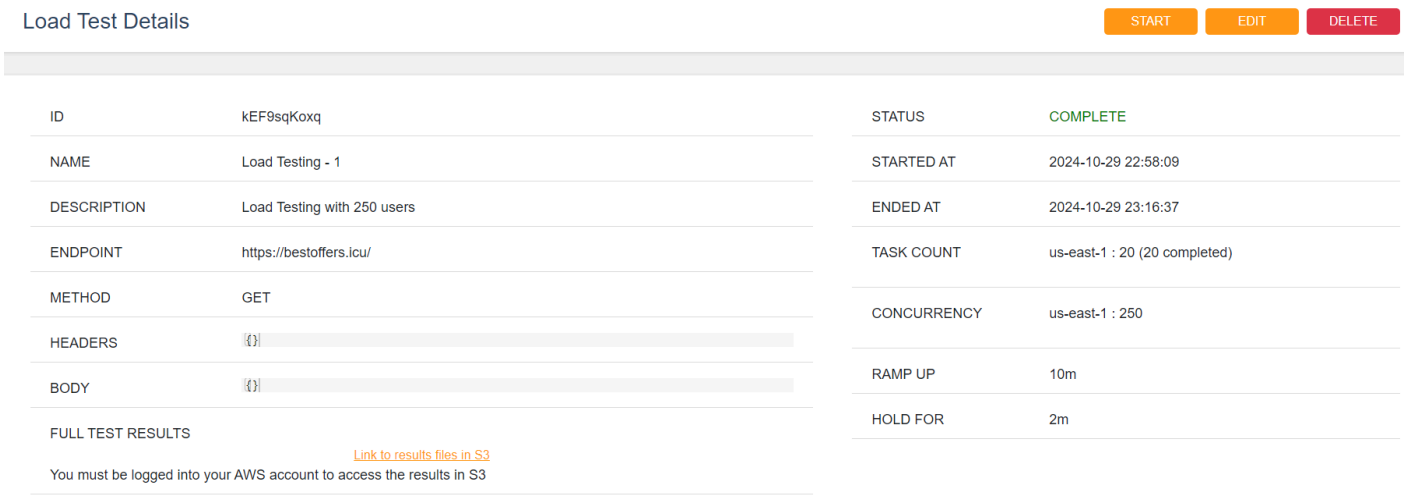
5. Testing and Monitoring

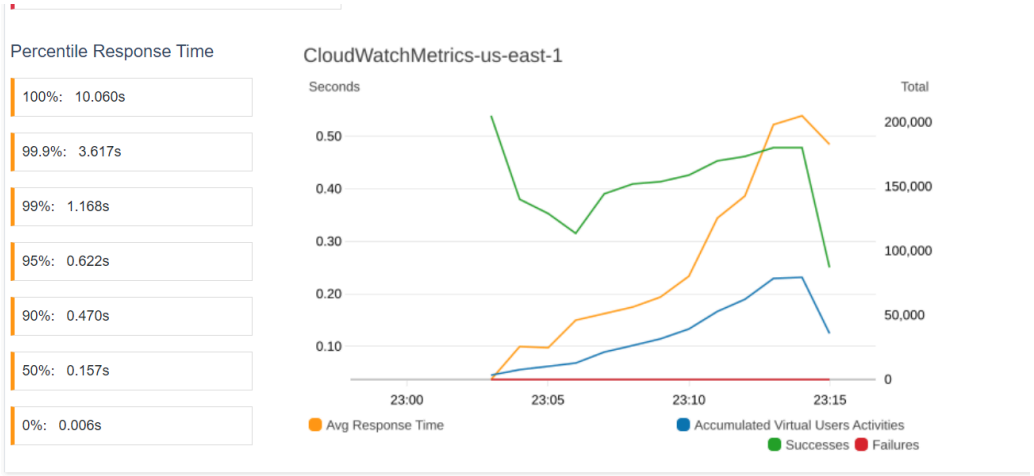
5.1 Stress Testing:

We have performed stress testing using AWS Distributed Load Testing to evaluate the reliability and performance of our e-commerce website under various traffic loads. This tool helped us simulate high numbers of concurrent users and assess how well the system handled different traffic conditions. Our goal was to observe how the website responded to increased demand, identify potential bottlenecks, and ensure a smooth user experience even during peak times.

Test Scenarios:

- **Scenario 1**: In this initial test, we simulated **20 tasks** with **250 concurrent users** accessing the site. The website, hosted at *https://bestoffers.icu (Landing page)*, successfully handled the load with a **100% success rate**, indicating strong system stability and effective handling of multiple requests simultaneously.

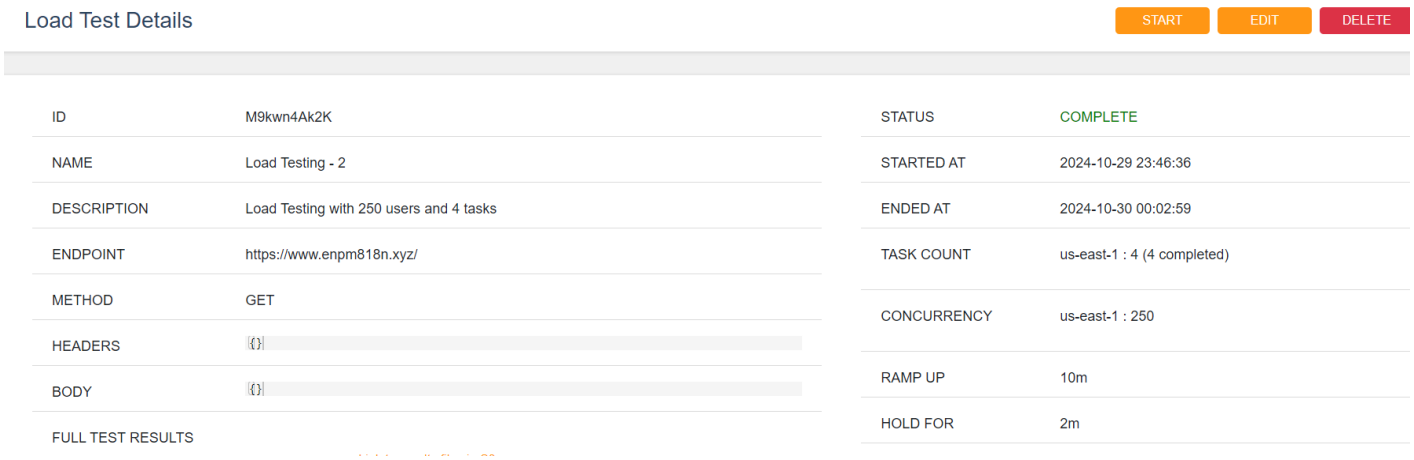


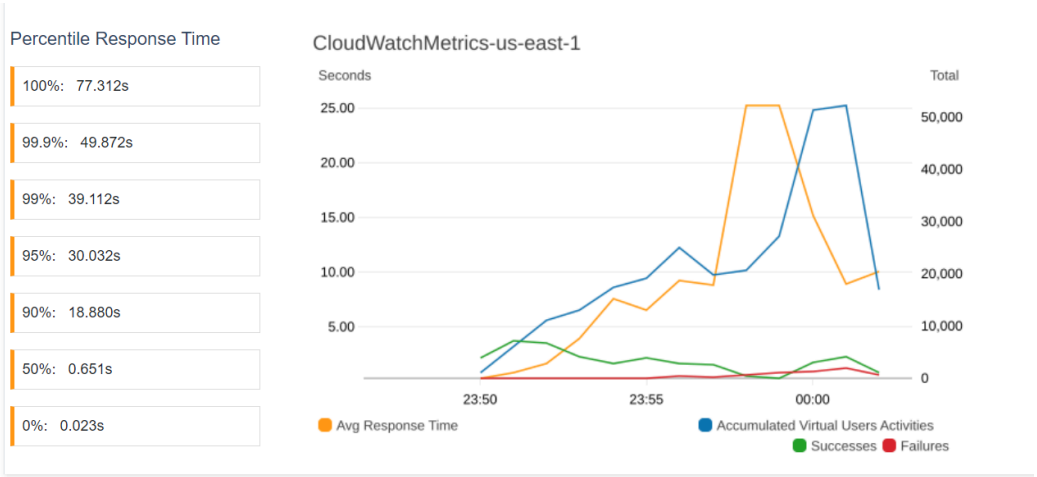


Results History

Run Time	Total Task Count	Total Concurrency	Average Response Time	Success %	
2024-10-29 23:16:37	20	250	0.22819s	100.00%	View details

- **Scenario 2**: We reduced the task count to **4 tasks** while maintaining **250 concurrent users** for the site hosted at *https://www.enpm818n.xyz (Main Website)*.This test yielded an **84.1% success rate**, signalling that although the system coped well with a majority of requests, and needed an increase in the maximum capacity of EC2 instances.

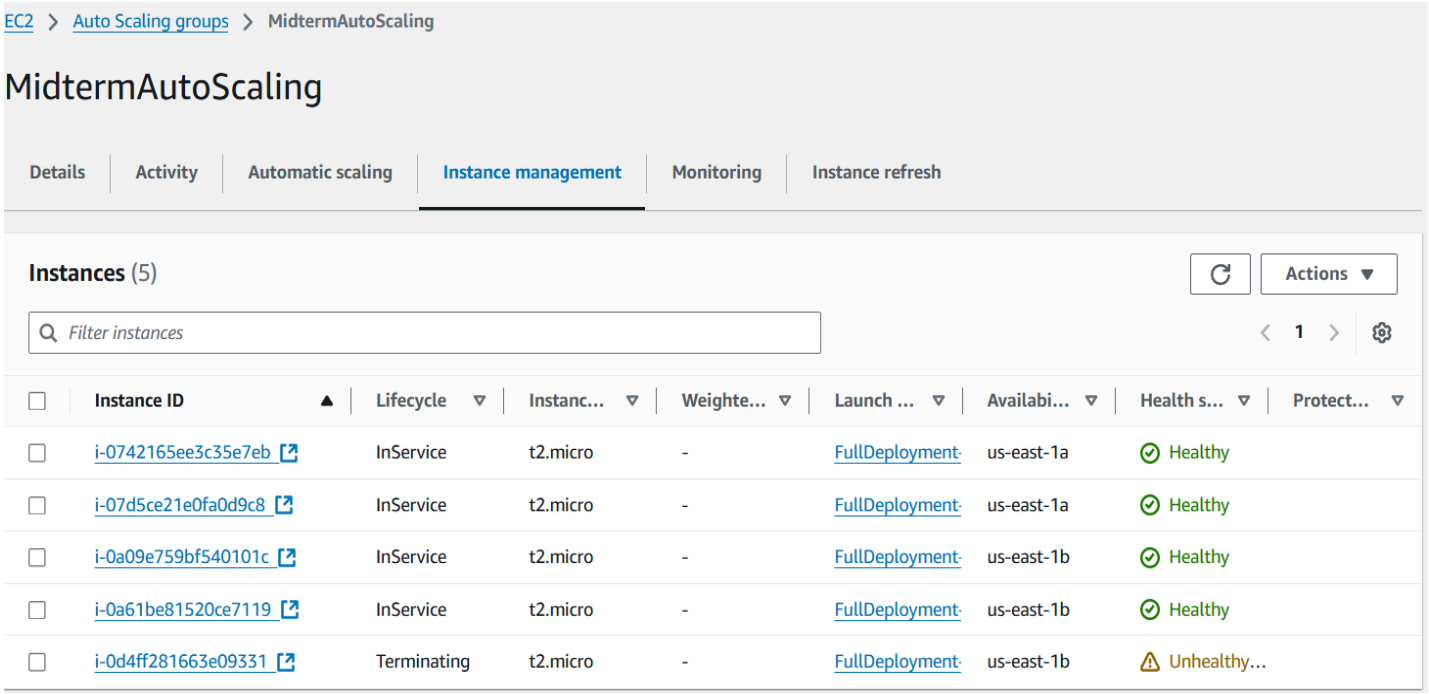




Results History

Run Time	Total Task Count	Total Concurrency	Average Response Time	Success %	
2024-10-30 00:02:59	4	250	6.37928s	84.10%	View details

We could see the Auto Scaling and RDS working with EC2 instances scaling up to maximum capacity and Multi-AZ DB triggering when the load increased.



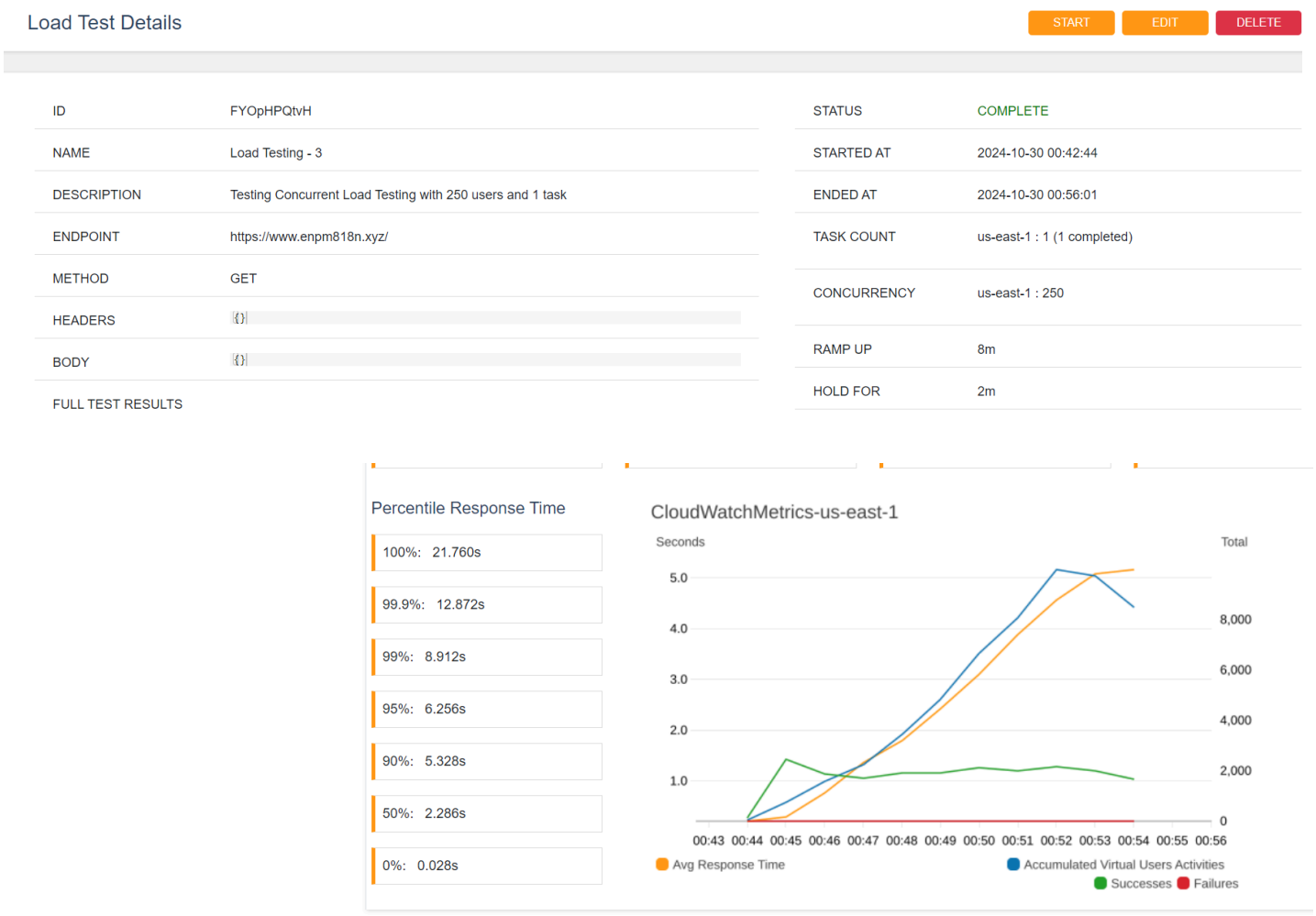
Status	Description	Cause	Start time
Successful	Launching a new EC2 instance: i-0a09e759bf540101c	At 2024-10-30T03:55:43Z an instance was launched in response to an unhealthy instance needing to be replaced.	2024 Octobe 11:55:4 -04:00
Connection draining in progress	Terminating EC2 instance: i-0d4ff281663e09331 - Waiting For ELB Connection Draining.	At 2024-10-30T03:55:43Z an instance was taken out of service in response to an ELB system health check failure.	2024 Octobe 11:55:4 -04:00
Waiting for instance warmup	Launching a new EC2 instance: i-07d5ce21e0fa0d9c8	At 2024-10-30T03:55:03Z a monitor alarm TargetTracking-MidtermAutoScaling-AlarmHigh-6a8b2ce1-e15d-4ec9-b008-e91af0048283 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 4. At 2024-10-30T03:55:12Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 4.	2024 Octobe 11:55:1 -04:00
Waiting for instance warmup	Launching a new EC2 instance: i-0a61be81520ce7119	At 2024-10-30T03:55:03Z a monitor alarm TargetTracking-MidtermAutoScaling-AlarmHigh-6a8b2ce1-e15d-4ec9-b008-e91af0048283 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 2 to 4. At 2024-10-30T03:55:12Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 2 to 4.	2024 Octobe 11:55:1 -04:00
Successful	Launching a new EC2 instance: i-0742165ee3c35e7eb	At 2024-10-30T03:49:03Z a monitor alarm TargetTracking-MidtermAutoScaling-AlarmHigh-6a8b2ce1-e15d-4ec9-b008-e91af0048283 in state ALARM triggered policy Target Tracking Policy changing the desired capacity from 1 to 2. At 2024-10-30T03:49:13Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 1 to 2.	2024 Octobe 11:49:1 -04:00

Databases (5) Group resources | Modify | Actions | Restore from S3 | Create database

Filter by databases

DB identifier	Status	Role	Engine	Region ...	Size	Recommendations	CPU	Current...	Maint
midterm-db	Available	Multi-AZ ...	MySQL Co...	us-east-1	3 instances	2 Informational	-	-	none
midterm-db-instance-1	Available	Writer inst...	MySQL Co...	us-east-1b	db.m5d.la...		9.97%	11 Con	none
midterm-db-instance-2	Available	Reader ins...	MySQL Co...	us-east-1c	db.m5d.la...		10.41%	9 Conn	none
midterm-db-instance-3	Available	Reader ins...	MySQL Co...	us-east-1a	db.m5d.la...		12.10%	9 Conn	none
midtermdb	Stopped tem	Instance	MySQL Co...	us-east-1a	db.t3.micro	2 Informational	-		none

- **Scenario 3**: In this final test, only **1 task** was used with **250 concurrent users** on *https://www.enpm818n.xyz (Main Website)*. This configuration yielded a **99.64% success rate**, a significant improvement from the previous test, showcasing how even a minimal task load with high user concurrency can still offer near-complete success with optimized settings.



Results History

Run Time	Total Task Count	Total Concurrency	Average Response Time	Success %	
2024-10-30 00:56:01	1	250	2.68187s	99.64%	View details

** After the 3 stress tests, we have concluded that the AWS functionalities associated are working as expected and configured under high-stress conditions.*

5.2 Monitoring Setup:

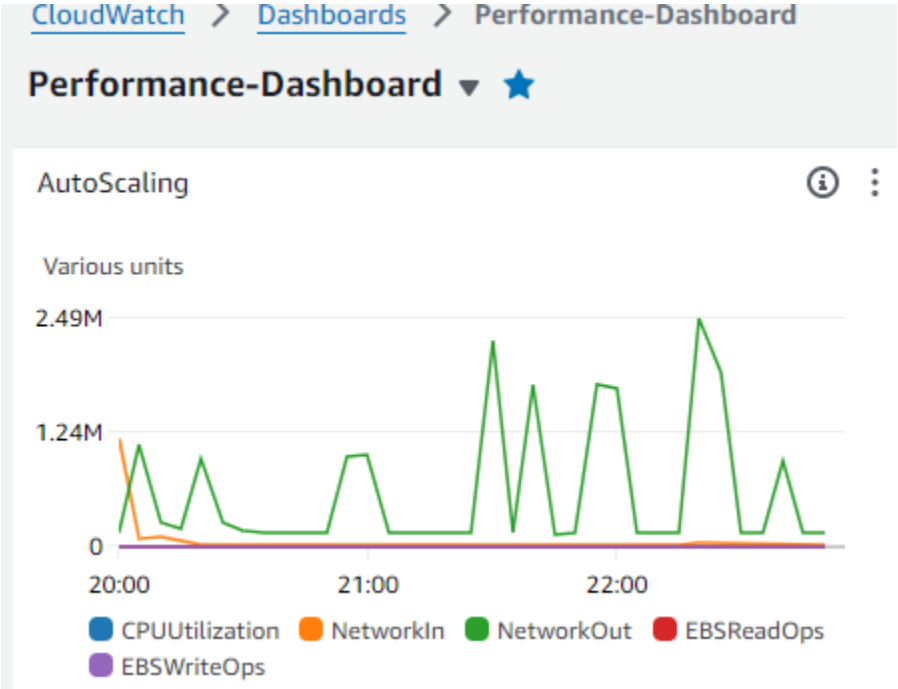
5.2.1 AWS CloudWatch:

To keep our e-commerce website running efficiently and proactively manage performance, we set up a CloudWatch Performance Dashboard. This dashboard provides a real-time view of our system's health by tracking essential metrics such as CPU utilization, database connections, error rates, and traffic patterns. With these insights, we can understand how our site responds to different levels of user demand and quickly address any issues. We also used the CloudWatch dashboard to observe specific metrics during stress testing, and we'll include visual examples of these metrics in the report.

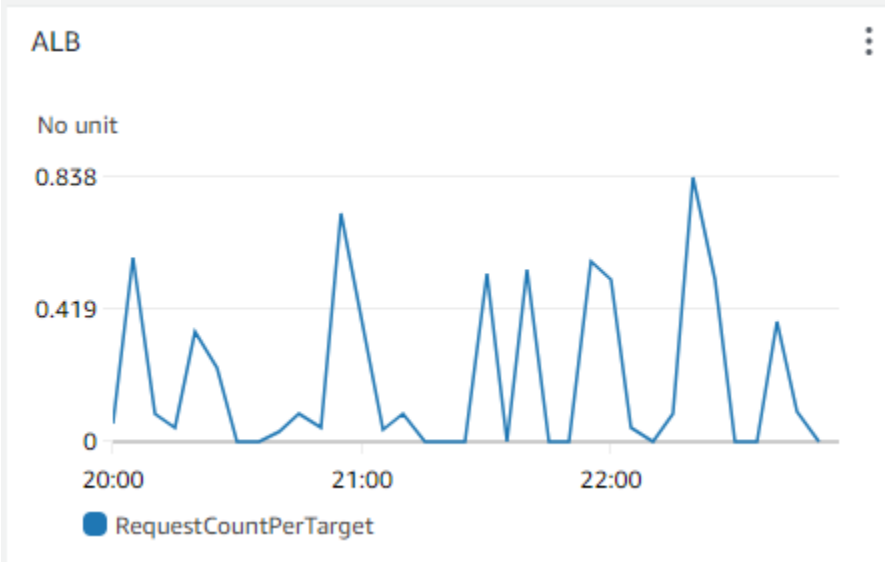
Dashboard Overview

We designed four main dashboards in CloudWatch to capture detailed metrics across key components of our infrastructure:

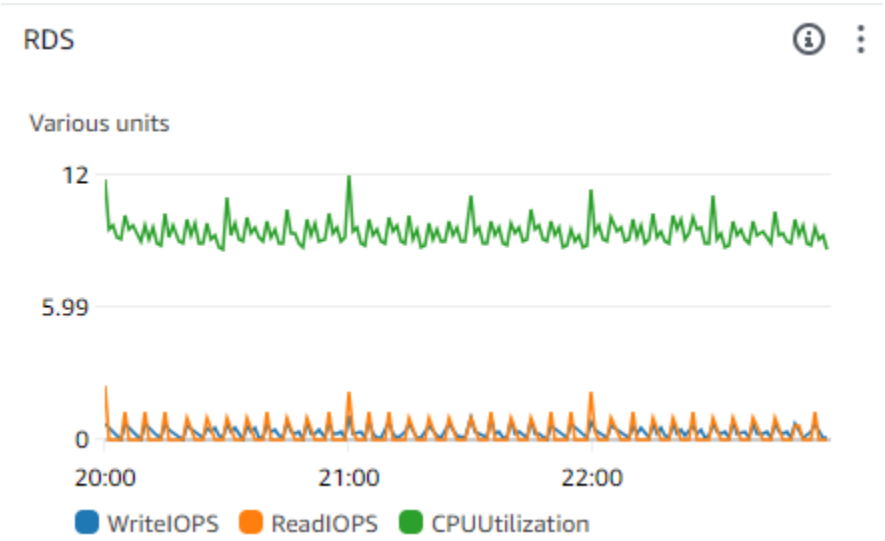
1. **Auto Scaling Dashboard**: This dashboard helps us monitor metrics for our Auto Scaling group. We track **CPU Utilization**, **Network In**, **Network Out**, **EBS ReadOps**, and **EBS WriteOps** to assess resource usage. By reviewing these, we ensure that our system scales effectively based on traffic levels, allowing us to maintain performance as demand shifts.



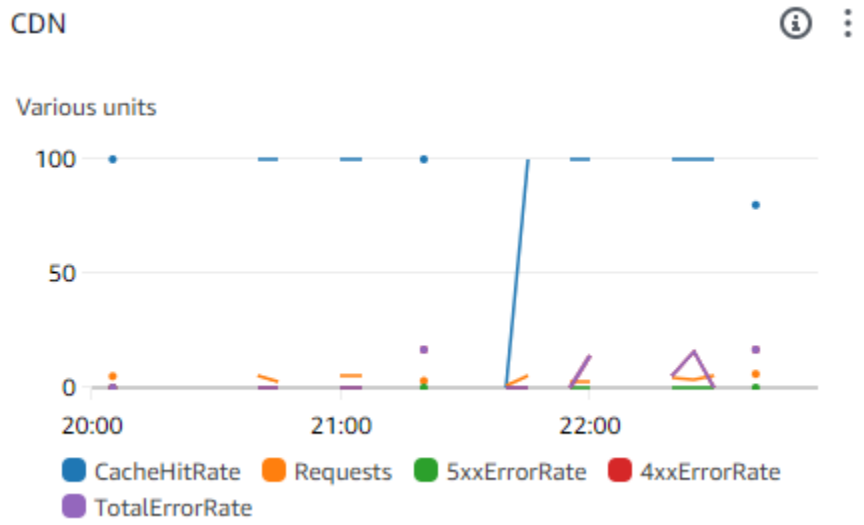
2. **Application Load Balancer (ALB)**: For our load balancer, we focus on **RequestCountPerTarget** to measure incoming requests to each server. Monitoring this metric helps us distribute the traffic efficiently and identify any irregularities in request distribution that might affect user experience.



3. **Relational Database Service (RDS)**: In our database dashboard, we keep track of **WriteIOPS**, **ReadIOPS**, and **CPU Utilization**. These metrics show how well our database handles read and write operations, as well as its overall processing load. Keeping an eye on these indicators helps us maintain database performance and avoid potential bottlenecks that might slow down the site.



4. **Content Delivery Network (CDN)**: For the CDN, we monitored **CacheHitRate**, **Requests**, and error rates, specifically **5xx** and **4xx** errors. We also keep track of **Total Error Rates**. This dashboard provides insights into how well our content is cached and distributed, reducing load times for users and ensuring smooth content delivery. By identifying and resolving any high error rates, we aim to maintain a high-quality user experience across regions.



Alarms

In addition to these dashboards, we set up three key CloudWatch alarms to alert us when certain conditions arise:

1. **CloudFront High Request Rate:** This alarm activates when the request count exceeds 1,000 within a five-minute window. It helps us identify unexpected traffic spikes and respond quickly if they impact performance.
2. **AutoScaling High Alarm:** We've configured this alarm to trigger if CPU Utilization exceeds 50% for three data points within three minutes. This alert helps us catch any unexpected load on the servers and make adjustments to maintain performance.
3. **AutoScaling Low Alarm:** To manage resource efficiency, this alarm activates if CPU Utilization drops below 35% for three data points within 15 minutes. By monitoring this threshold, we can optimize resource usage when demand is low, reducing unnecessary costs.

5.2.2 AWS CloudTrail:

To enhance security and maintain a detailed record of actions within our AWS environment, we implemented a Multi regional AWS CloudTrail for logging security-related events. This setup enables us to track and review various activities across our resources, helping us monitor for unauthorized or unusual actions.

We configured CloudTrail to capture two types of events: **Management Events** and **Data Events**. Management Events log high-level actions such as creating, modifying, or deleting resources, providing visibility into changes made to our AWS infrastructure. Data Events, in contrast, track specific access to objects within our storage resources, offering a closer look at user interactions with critical data.

By logging both event types, CloudTrail delivers a comprehensive record of activity, supporting security oversight and compliance needs. This setup allows us to swiftly identify potential issues and respond appropriately, safeguarding the integrity and security of our environment.

5.2.3 AWS Config Setup for Configuration Monitoring:

To maintain a secure and well-governed cloud environment, we implemented AWS Config to continuously monitor and log configuration changes across our AWS resources. This setup allows us to ensure that all resources adhere to our compliance and security standards and enables quick identification of any deviations from desired configurations.

Key Monitoring Categories and Rules

1. **Security and Identity Monitoring**:
   To strengthen identity management, we set up rules to verify critical security configurations. These include checks for multi-factor authentication (MFA) on IAM users, password policy enforcement, restrictions on root access keys, and monitoring of CloudTrail configurations. Specifically:
   - **IAM_USER_MFA_ENABLED** ensures that MFA is enabled for user accounts.
   - **IAM_PASSWORD_POLICY** checks for strong password policies.
   - **IAM_ROOT_ACCESS_KEY_CHECK** confirms the secure handling of root access keys.
   - **CLOUDTRAIL_ENABLED** and **CLOUDTRAIL_S3_DATAEVENTS_ENABLED** verify that CloudTrail is logging management and data events, respectively, helping us maintain visibility into resource access and modifications.

2. **Network Configuration and Monitoring**:
   For network security, we set rules to monitor our Virtual Private Cloud (VPC) settings and secure access points. Specifically:

   - **VPC_FLOW_LOGS_ENABLED** to track network traffic.
   - **VPC_DEFAULT_SECURITY_GROUP_CLOSED** ensures that the default security group has restricted access.
   - **RESTRICTED_SSH** and **RESTRICTED_RDP** rules limit SSH and RDP access to authorized IPs only, reducing exposure to potential threats.

3. **Data Protection and S3 Configuration**:
   We prioritized data security by enforcing encryption and restricting public access on our S3 buckets. Key rules include:
   - **S3_BUCKET_PUBLIC_READ_PROHIBITED** and **S3_BUCKET_PUBLIC_WRITE_PROHIBITED** to prevent unauthorized public access.
   - **S3_BUCKET_SERVER_SIDE_ENCRYPTION_ENABLED** ensures data encryption at rest.
   - **S3_BUCKET_VERSIONING_ENABLED** supports data recovery by enabling versioning on buckets.

4. **Logging and Monitoring Rules**:
   To ensure that logging is in place across critical resources, we included:
   - **CLOUDWATCH_LOG_GROUP_ENCRYPTED** to secure CloudWatch log data.
   - **RDS_LOGGING_ENABLED** to track database logs, aiding in database activity monitoring and troubleshooting.

5. **Compute and Resource Configuration Rules**:
   For compute resources, we established checks to verify resource usage and management:
   - **EC2_VOLUME_INUSE_CHECK** ensures all EC2 volumes are attached to instances, reducing unused resources.
   - **EC2_INSTANCE_MANAGED_BY_SSM** confirms that instances are managed by AWS Systems Manager for easier monitoring and automation.
   - **RDS_INSTANCE_PUBLIC_ACCESS_CHECK** prevents public accessibility of RDS instances, enhancing database security.

6. **Compliance and Governance Rules**:
   To ensure compliance, we implemented tagging rules and encryption standards. Key checks include:
   - **TAG_COMPLIANCE** and **REQUIRED_TAGS** to ensure resources are properly labeled, aiding in organization and cost management.
   - **EBS_SNAPSHOT_PUBLIC_RESTORABLE_CHECK** restricts public access to EBS snapshots.
   - **ENCRYPTED_VOLUMES** mandates that all volumes are encrypted for data protection.

6. Cost Analysis

Based on our detailed tracking and analysis using AWS Cost Explorer, we've provided an estimated breakdown of costs associated with running our scalable and secure e-commerce platform. The primary expenses are attributed to EC2 instances, RDS, CloudFront, and other AWS services that form the backbone of our infrastructure.

EC2 Instances: Given the variable traffic, the on-demand instances initially seemed cost-effective; however, consistent usage patterns suggested potential savings. Estimated monthly cost: $500.

RDS: Utilizing RDS for MySQL with Multi-AZ deployment ensures high availability but increases costs due to the replication feature. Estimated monthly cost: $300.

CloudFront: Costs are incurred based on the amount of data transferred out to the internet. Estimated monthly cost: $150.

Additional Services: Includes WAF, S3, Route 53, and other managed services contributing to security and performance. Estimated monthly cost: $50.

Total Estimated Monthly Cost: $1,000.

To manage and potentially reduce these costs, we suggest the following strategies:

1. Reserved Instances: By purchasing Reserved Instances for EC2 and RDS, we can significantly lower our expenses. A 1-year Standard Reserved Instance can offer up to 40% savings compared to on-demand instance pricing. For predictable usage, a 3-year term could save up to 60%, aligning with our long-term operational goals.
2. Savings Plans: Computing Savings Plans are another excellent option, offering up to 66% savings. They provide flexibility in terms of instance families, AWS services, and geographic regions without upfront costs if chosen on a "no upfront" payment term.
3. Spot Instances: For non-critical workloads such as batch processing or development environments, utilizing Spot Instances can reduce costs by up to 90%. This approach leverages unused EC2 capacity at significantly lower prices.
4. Optimizing CloudFront Usage: By adjusting cache behaviors and expiration settings, we can decrease the number of requests that reach the origin server, thus reducing CloudFront costs.
5. Right-Sizing Resources: Regularly analyze performance and usage metrics to right-size instances and databases, ensuring we are not over-provisioning resources. AWS Trusted Advisor and Cost Explorer provide insights and recommendations for optimal configurations.
6. Budget Alarms and Cost Management Tools: Utilize AWS Budgets to set custom cost and usage budgets that send alerts when service costs exceed your thresholds. This proactive measure helps avoid unexpected expenses.

By implementing these cost optimization strategies, we anticipate not only reducing our monthly AWS expenses but also improving our overall financial governance as we scale our e-commerce platform. These measures will ensure that we continue to use AWS resources efficiently, aligning costs with our business objectives without compromising on performance and security.

7. Future Improvements

There are some key area which can be improved in future such as:

1. Security:
    a. AWS Shield: Implement AWS Shield for DDoS protection to safeguard against large-scale attacks.
    b. AWS Secrets Manager: Store and manage sensitive information like database credentials securely.
2. Performance optimization:
    a. Auto Scaling for RDS: Auto scaling can be implemented for RDS as well, this will improve performance while saving cost.
    b. Optimize Caching: Using Amazon ElastiCache to cache frequently accessed data and reduce database load.
3. Disaster Recovery
    a. Set Up Cross-Region Replication: Implement cross-region replication for S3 and RDS to enhance disaster recovery capabilities.
    b. Periodic Backups: Ensuring automated backups are created for all critical data.
4. Implement CI/CD Pipelines: Use AWS CodePipeline and CodeDeploy to automate application deployments, ensuring faster and more reliable updates.