**ENPM 818N Mid - Group 9  
Scalable and Secure E-commerce Platform on AWS**

**By:**

| **Name** | **Directory ID** | **University ID** | **Section** |
| --- | --- | --- | --- |
| Ananda Kshitij | akshitij | 121131072 | 0101 |
| Nikhil Vishnu Bonageri | nikhilvb | 121007810 | 0101 |
| Rincy Mariam Thomas | rmt21 | 120124044 | 0101 |
| Sreelekshmy Rengith | srengith | 119458815 | 0101 |

**A logo of a university of maryland

Description automatically generated**

University of Maryland

7950 Baltimore Avenue

College Park, MD 20742

10-29-2024

**Table of Contents**

[**1. Introduction 4**](#_r7f5a0evosus)

[1.1 Project Overview 4](#_29gn3hc9ofx7)

[1.2 Objective and Scope 4](#_yzr741j0xooa)

[**2. Phase 1: Infrastructure Setup 6**](#_uhc0dzeq0st9)

[2.1 VPC and Subnet Planning 6](#_hs76gs3yprvb)

[2.2 EC2 Instances and Load Balancer Configuration 8](#_w3lzxydczqei)

[2.3 RDS Setup and Database Encryption 9](#_gac41kmgvwxg)

[2.4 Auto Scaling Policies and CloudWatch Alarms 12](#_g0jmzws70d9f)

[**3. Phase 2: Securing the Application 15**](#_oz1rxk8hp2br)

[3.1 Set Up AWS WAF 15](#_fb1297hcf8he)

[3.2 Domain Management and SSL Configuration with Route 53 15](#_12me7igsetr1)

[3.3 Encrypt Data in Transit 17](#_uj9z5y9ggcfr)

[3.4 Database Connection Encryption: 17](#_3kd5vn5ej8qx)

[**4. Phase 3: Content Delivery and Performance Optimization 18**](#_bz2ojshapgfw)

[4.1 CloudFront CDN Integration 18](#_one47rmrz4hj)

[4.2 Performance Tuning and GZIP Compression 19](#_1tbrql8lvfvp)

[4.3 Cache Configuration 19](#_2lheoo97flpv)

[4.4 CloudWatch Monitoring and Alarms 19](#_d2u4asmz4ntw)

[4.5 Security in CloudFront 21](#_lol0j42bude)

[**5. Phase 4. Testing and Monitoring 22**](#_yezk8rkqvsve)

[5.1. DNS and Domain Setup 22](#_w1dz97wdkjh1)

[5.2. Load Testing with JMeter 23](#_9i0mihejp1za)

[5.3. AWS WAF (Web Application Firewall) 25](#_p5xoeaceuvdi)

[5.4. Monitoring with AWS CloudWatch 26](#_ho0hy3rx2hbe)

[5.5 Cost Analysis and Savings Recommendations 27](#_1zm26bj10c4e)

[**6. Architecture Design 29**](#_16tyc92rk9jo)

[**7. Conclusion 31**](#_17vsqvod0u9k)

[7.1 Lessons Learned 31](#_pis5apj9a6uu)

[7.2 Potential Improvements 31](#_p5cbdl65l6i7)

[**8. Appendices 32**](#_618nk2xgidx)

[**9. References 32**](#_injrredj65uf)

**Figures**

[Figure 1: VPC Networking Architecture 6](#_oprqldacmszi)

[Figure 2: RDS MySQL Instance with KMS Encryption and Secrets Manager Integration 9](#_zgpw7h9mx1y0)

[Figure 3: DB CPU Utilization 10](#_9op2qbx88z9l)

[Figure 4: DB Config 11](#_m95k55hxj5zt)

[Figure 5: Summary of RDS MySQL Instance Configuration and Status 12](#_s450bm5bmh61)

[Figure 6: RDS Backup and Snapshot Management 12](#_alwk3w3tb8ah)

[Figure 7: Auto Scaling Group with Application Load Balancer and Target Tracking Scaling Policy 13](#_mjejbwm3bhpw)

[Figure 8: WAF SetUp 15](#_s9wx2hkvtd4p)

[Figure 9: CDN Setup with CloudFront, S3, and ALB for Static and Dynamic Content 18](#_9pgla0pgfd5n)

[Figure 10: CDN CloudWatch Dashboard 20](#_kyq9nmd3uhwg)

[Figure 11: Sample CloudWatch Alarm Statistics 20](#_o0x4kmpac3q3)

[Figure 12: Route 53 nameservers in the hosted zone 23](#_pusb8li2ffj8)

[Figure 13: GoDaddy nameserver settings with the nameservers copied from Route 53 23](#_jteqvnd9fuzp)

[Figure 14: JMeter Test 24](#_9lh48ytyt9xn)

[Figure 15: WAF Dashboard 25](#_7fyiqdpz9crb)

[Figure 16: CloudWatch Dashboard 26](#_93oynet4b4ad)

[Figure 17: CloudTrail Log 27](#_virpx8vbtmox)

[Figure 18: AWS Config Usage Metrics 27](#_ezbr88svd0vz)

[Figure 19: Savings Plan Recommendations 28](#_22my0240soqm)

[Figure 20: Architecture Diagram 29](#_yyt3syq0ogdn)

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 1. Introduction

In this group project, the objective is to design, develop, and deploy a **Scalable and Secure E-commerce Platform** on **AWS**. This platform is intended to serve as a practical exercise in leveraging AWS’s suite of cloud services to create an e-commerce solution that meets the demands of scalability, security, and performance. With modern e-commerce platforms needing to handle dynamic, high-volume user traffic, secure sensitive customer data, and deliver content quickly and reliably, AWS offers an ideal environment through its managed services.

The platform will be built using core AWS components such as **Auto Scaling** to dynamically manage computing resources, **RDS (Relational Database Service)** for a secure and scalable database, **WAF (Web Application Firewall)** to protect against common security threats, and **CloudFront** for accelerated content delivery across global regions. These services will work together to assist us in developing a strong, safe, and highly available architecture that can manage the demands of actual e-commerce.

## 1.1 Project Overview

This project will focus on building a robust, cloud-native e-commerce platform that can effectively handle varying levels of user load, ensure data security for sensitive customer information, and optimize content delivery for a global audience. The AWS architecture we are developing will encompass all essential layers needed for a full-scale e-commerce solution:

**Compute Layer:** Making use of EC2 instances, which automatically scale in and out in response to traffic demand, guaranteeing cost-effective resource allocation.

**Database Layer**: Setting up a MySQL RDS instance with high availability and encryption capabilities to safeguard data while it's in transit and at rest in order to keep critical data (such as customer information, product details, and order histories).

**Security Layer:** Using encryption protocols for safe communication, implementing AWS Identity and Access Management (IAM) policies for stringent access control, and deploying WAF to counteract web-based attacks like SQL Injection and Cross-Site Scripting (XSS).

**Content Delivery Layer:** Setting up CloudFront CDN to cache content at edge locations near users around the world, optimize load times, and serve static assets (such product photos and CSS files).

To support real-world e-commerce demands, this project’s architecture will also include monitoring and logging tools for performance analysis, cost management, and ongoing optimization.

## 1.2 Objective and Scope

#### 

#### **Objective**

Our primary goal is to implement an e-commerce platform on AWS which is scalable, safe, and has the capacity to handle fluctuating traffic loads while guaranteeing data security and quick content delivery.

#### **Scope**

The scope of this project covers the full deployment of an e-commerce application, from infrastructure setup to performance monitoring. Key components include:

1. **Auto Scaling and Load Balancing:** To save money, EC2 instances will be grouped together in an Auto Scaling group that adapts to predetermined rules (such CPU utilization). This will allow the application to scale up during periods of high traffic and down during periods of low traffic. To ensure balanced load distribution for high availability, an Application Load Balancer (ALB) will divide traffic among EC2 instances.
2. **RDS Database Configuration:** The relational database for the platform will be a MySQL RDS instance, which will offer long-term storage for client and product information. RDS will be set up with KMS encryption to protect data while it is at rest and Multi-AZ deployment to facilitate failover.
3. **Web Application Firewall (WAF) Setup:** To safeguard the application against typical threats like SQL Injection and XSS that could jeopardize user data or application integrity, WAF will be set up with both AWS Managed Rules and custom rules.
4. **CloudFront Content Delivery Network (CDN):** The CloudFront CDN helps to cache the static assets like CSS, images and Javascript files thus speeding up loading items by serving cached content from AWS edge locations worldwide. This reduces the wait time for loading these resources, creating a smoother, more responsive experience for users.
5. **Security and Encryption:** To protect data while it is being transferred between users and the program, we will use SSL/TLS certificates. Data at every level of the stack will be protected by the encryption of database connections between the application and RDS.
6. **Monitoring and Cost Optimization:** To guarantee seamless operations and efficient issue response, CloudWatch will be used to track the platform's performance and security parameters.

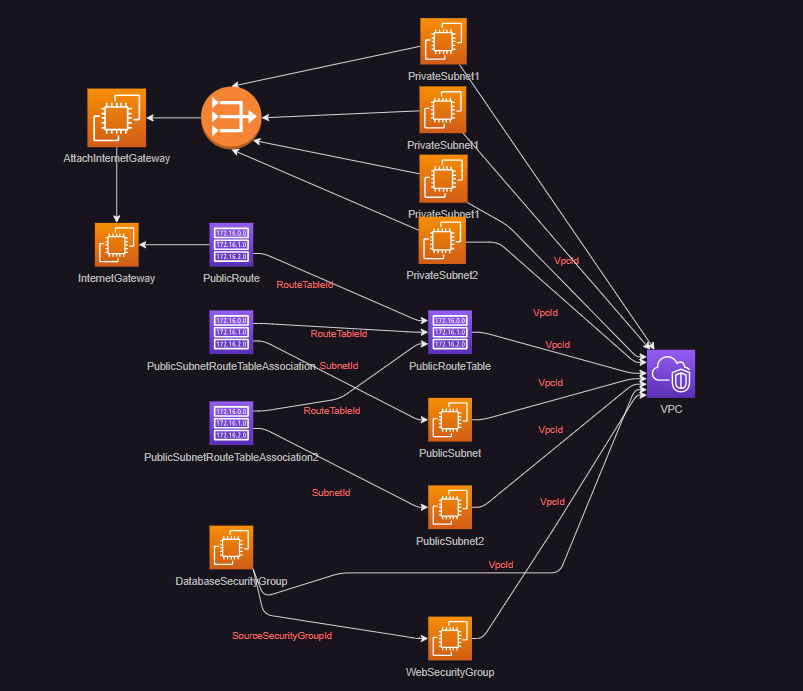
Our goal is to launch a cloud-based e-commerce platform at the project's conclusion that satisfies performance, security, and scalability standards, setting the groundwork for future improvements and practical use.

# 2. Phase 1: Infrastructure Setup

## 2.1 VPC and Subnet Planning

### **Overview of Resources**

The CloudFormation template provisions the following key networking resources:

1. **VPC**
2. **Subnets**
3. **Internet Gateway (IGW)**
4. **Route Tables and Routes**
5. **Security Groups**  
     
    

###### Figure 1: VPC Networking Architecture

### **Resource Breakdown and Configuration**

#### **VPC (Virtual Private Cloud)**

* **Purpose**: The VPC provides a secure and isolated network environment for the microservices to manage IP addressing and routing within the VPC.
* **Configuration**:
  + **CidrBlock** is set to 10.0.0.0/16 which allows for a broad IP address range for internal use.
  + **EnableDnsSupport** and **EnableDnsHostnames** are both enabled thus allowing instances within the VPC to resolve domain names and use hostnames.
* **Security Note**: The VPC serves as the main network boundary which controls the IP range for resources and isolating traffic from other VPCs.

#### **Subnets**

* **Purpose**: The subnets divide the VPC into logically separate areas for public and private resources which inturn enhances security and organization.
* **Configuration**:
  + **PrivateSubnet1** is configured in us-east-2a with CidrBlock set to 10.0.1.0/24.
  + **PrivateSubnet2** is also configured in us-east-2b with CidrBlock set to 10.0.2.0/24.  
    **Additional Private Subnets for redundancy:** *PrivateSubnet1Ec2 (us-east-2a)* and *PrivateSubnet2Ec2 (us-east-2b)* provide enhanced isolation for backend instances in separate availability zones.
  + **PublicSubnet** is configured in us-east-2a with CidrBlock set to 10.0.3.0/24 and MapPublicIpOnLaunch enabled for internet-accessible resources.
  + **Internet Gateway (MyInternetGateway):** Attached to the public subnets, enabling internet access for resources within these subnets.
  + **NAT Gateways:** Configured in each availability zone to allow private subnet resources to access the internet for updates and patches without exposing them to inbound internet traffic.
* **Usage** : The private subnets are isolated from direct internet access making it ideal for databases or backend services, while the public subnet is suited for load balancers and other internet-facing resources.

#### **Internet Gateway (IGW)**

* **Purpose**: The IGW provides internet access to resources within the public subnet.
* **Configuration**:
  + **Attachment**: The IGW is attached to the MicroservicesVPC, enabling communication between public resources and the internet.
* **Security Note**: Only resources in the public subnet can access the internet, and further access control is managed via route tables and security groups.

#### **Route Tables and Routes**

* **Purpose**: To direct traffic between subnets within the VPC and manage traffic flow to and from the internet.
* **Configuration**:
  + **Public Route Table**: Configures routes to the internet through the IGW for resources in the public subnet.
  + **Private Route Table**: Routes are isolated, restricting internet access, thus securing internal resources.
* The routing setup ensures traffic flows appropriately within the VPC while maintaining network isolation for sensitive resources in private subnets.

#### **Security Groups**

* **Purpose**: SG’s controls inbound and outbound traffic for resources in the VPC, adding a layer of security for application instances.
* **Configuration**:
  + **WebSecurityGroup**: Limits access to only necessary ports (e.g., HTTP, HTTPS) for web-facing resources.
  + **Ec2SecurityGroup:** Limits access only to HTTP and HTTPS from web security group and SSH from security group of jump server.
  + **DatabaseSecurityGroup**: Allows sql connection to RDS MYSQL.
* Security groups provide fine-grained control, ensuring only trusted traffic reaches application instances and restricting unwanted access.

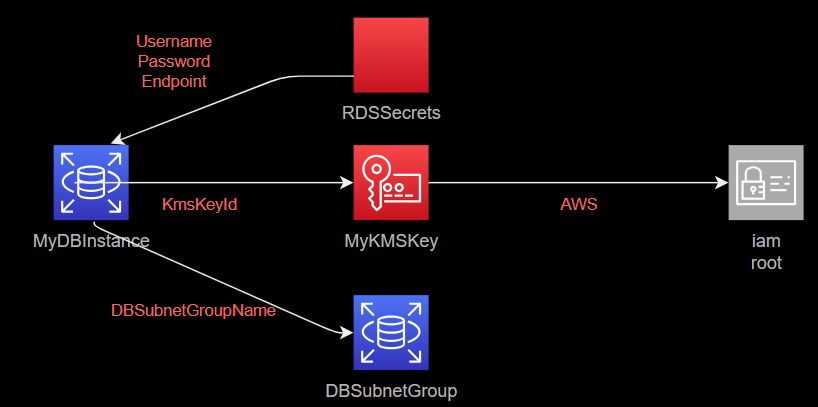
## 2.2 EC2 Instances and Load Balancer Configuration

* **EC2 Instances**:
  + **Instance Type**: 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 resources (CPU, memory) for a small-to-medium workload.
  + **Image ID (AMI)**: The specific Amazon Machine Image (AMI) ID ami-09483685933501e93 pre-installs necessary software and dependencies, ensuring each instance has a consistent environment. This reduces setup time and eliminates potential configuration errors.
  + **Security Group**: EC2 instances are secured with the WebSecurityGroup, which controls inbound and outbound traffic, allowing only specific types of connections (e.g., HTTP, HTTPS) to reach the instances.
* **Load Balancer Setup**: The Application Load Balancer (ALB) evenly distributes incoming requests among EC2 instances, ensuring smooth operation even as demand increases.
  + **Listener Configuration**: The ALB listens for traffic on HTTPS (port 443), which encrypts data transmitted between users and the application, safeguarding sensitive information.
  + **Target Groups**: The ALB organizes EC2 instances into target groups, which determine where traffic is sent. This configuration allows the ALB to manage multiple instances effectively, redirecting traffic as needed based on instance health and capacity.

## 2.3 RDS Setup and Database Encryption

### **Overview of Resources**

The CloudFormation template provisions the following key resources:

1. **DBSubnetGroup**: Defines a logical grouping of subnets for the database deployment.
2. **KMS Key**: A key for encrypting database storage.
3. **Secrets Manager Secret**: Manages database credentials securely.
4. **RDS MySQL Instance**: The core database resource.
5. **Outputs**: Exports the ARN of the RDS secret for integration with other AWS services.  
     
    

###### Figure 2: RDS MySQL Instance with KMS Encryption and Secrets Manager Integration

### **Resource Breakdown and Configuration**

#### **DBSubnetGroup**

* **Purpose**: Specifies the subnets within the VPC where the RDS instance will reside. This ensures the database is isolated within the private subnets of the VPC for enhanced security.
* **Configuration**:
  + The **DBSubnetGroup** is associated with two subnets (PrivateSubnet1 and PrivateSubnet2), imported from the existing AWS environment, and referenced by their IDs.
  + The **DBSubnetGroupName** is set to "MyDBSubnetGroup" for easy identification and tagging.
* **Security Note**: By deploying the RDS instance in private subnets, we ensure that the database is not accessible directly from the internet, significantly enhancing security.

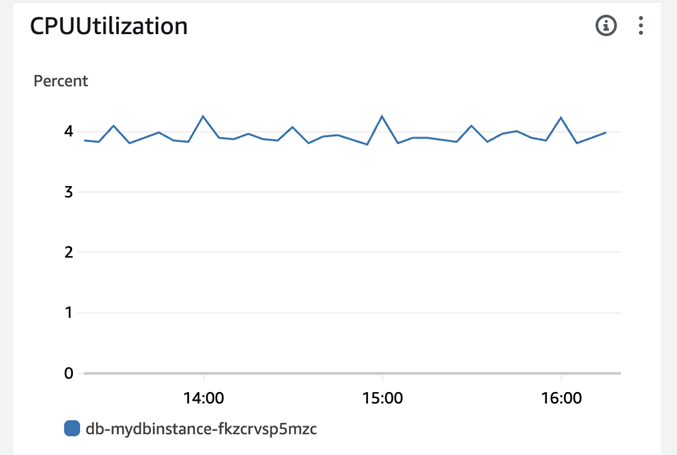
#### **MyKMSKey**

* **Purpose**: Provides encryption for the RDS MySQL instance storage, safeguarding sensitive data at rest.
* **Configuration**:
  + **Key Policy**: Grants the AWS account’s root user full control over the key.
  + **Key Rotation**: Automatically rotates the key to maintain security over time.
* **Security Note**: This KMS (Key Management Service) key will be used to encrypt all data stored in the RDS instance, helping prevent unauthorized access to sensitive information.

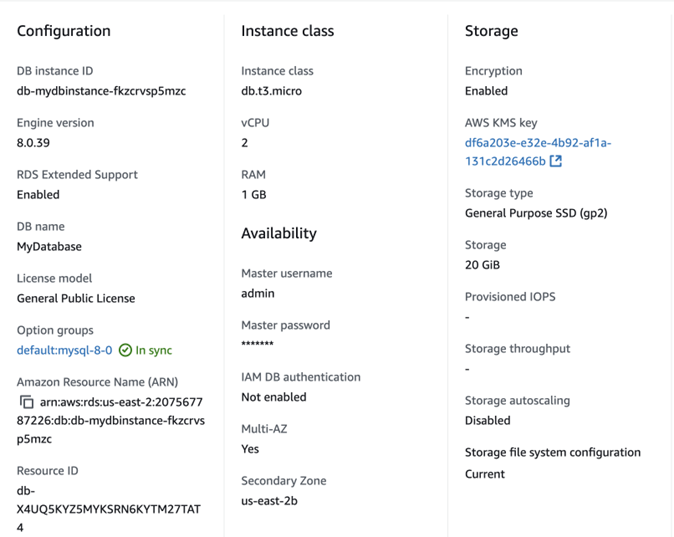
#### **RDSSecrets**

* **Purpose**: Stores and manages database credentials securely in AWS Secrets Manager, replacing hard-coded credentials.
* **Configuration**:
  + **GenerateSecretString**: Automatically generates a secure password (16 characters, excluding characters like /@" for compatibility).
  + **Secret Template**: Predefines the username as "admin" while the generated password remains hidden.
* **Security Note**: Storing credentials in **Secrets Manager** allows secure retrieval of the username and password without hardcoding them in the template, enhancing overall security.

#### **MyDBInstance**

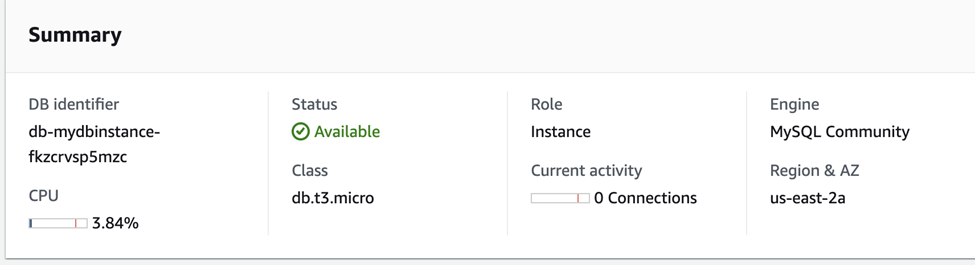
* **Purpose**: Creates and configures the MySQL database instance, leveraging the security and management practices outlined above.
* **Configuration**:
  + **DBInstanceClass**: A db.t3.micro instance type is chosen for cost-effective, lightweight workloads.  
    

###### Figure 3: DB CPU Utilization

* + **AllocatedStorage**: Provides 20 GB of storage for database needs.
  + **Engine**: Specifies MySQL as the chosen database engine.
  + **MasterUsername & MasterUserPassword**:
    - Rather than hard coding credentials, the template retrieves them securely from **Secrets Manager** using the syntax {{resolve:secretsmanager:MyRDSSecret:SecretString:username}}, which enhances security.
  + **StorageEncrypted**: Encrypts the storage with the previously created **KMS key** (MyKMSKey) for data protection.
  + **MultiAZ**: Enables high availability by deploying the RDS instance across multiple Availability Zones (AZs).
  + **VPCSecurityGroups**: Attaches the DatabaseSecurityGroup (imported from elsewhere) to manage access permissions to the database.
  + **DBSubnetGroupName**: Assigns the subnet group MyDBSubnetGroup to control the subnets where the database can be deployed within the VPC.  
    

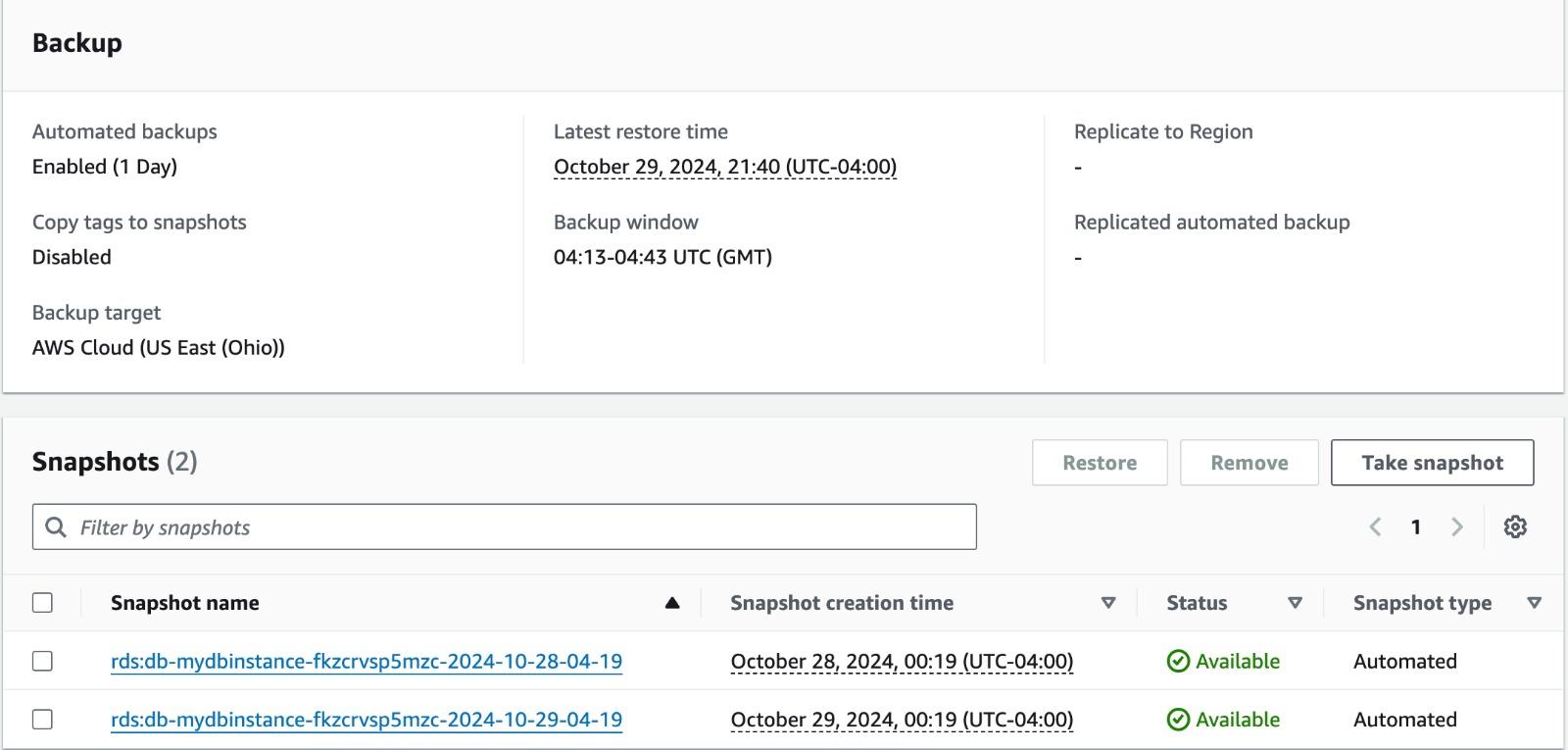
###### Figure 4: DB Config

* **Availability and Security Note**: Multi-AZ deployment, combined with encryption, private subnets, and controlled access via security groups, ensures high availability, resilience, and data protection for the database.



###### Figure 5: Summary of RDS MySQL Instance Configuration and Status

* **RDS Backup Configuration:** The RDS instance is configured for automated daily backups in the US East (Ohio) region to provide protection of data and support for disaster recovery. Automated backups are retained for 1 day, with the backup window set between 04:13 and 04:43 UTC. Snapshots are taken daily with the latest restore time on October 29, 2024, at 21:40 (UTC-04:00). And, of course, two recent automated snapshots from October 28 and October 29 are available in order to restore data in case some go missing or become corrupted, since a dependable recovery mechanism would be needed. Presently, tags are not copied to snapshots, and backups are not replicated to other regions.



###### Figure 6: RDS Backup and Snapshot Management

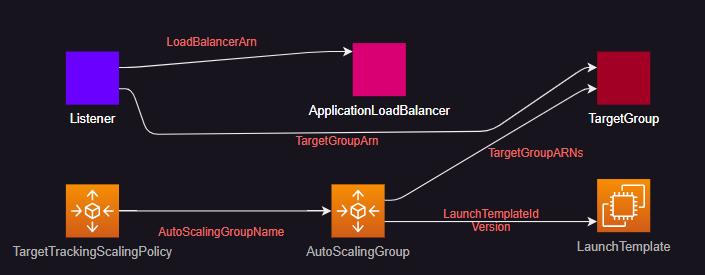
### **Outputs**

* **RDSSecretArn**:
  + The template outputs the **Amazon Resource Name (ARN)** of the RDS secret stored in **Secrets Manager**. This output can be used by other CloudFormation stacks or AWS services to securely access the database credentials when needed.
* **Usage Note**: Exporting the ARN simplifies integrating the RDS instance with other AWS resources without exposing sensitive information.

## 2.4 Auto Scaling Policies and CloudWatch Alarms

### **Overview of Resources**

The cloud formation template provisions the following main resources:

1. **Launch Template**: Defines EC2 instance configurations for the Auto Scaling Group.
2. **Application Load Balancer (ALB)**: Distributes traffic to instances across multiple Availability Zones.
3. **Target Group**: Routes incoming requests to instances.
4. **Auto Scaling Group (ASG)**: Automatically scales EC2 instances based on traffic demands.
5. **Networking Components**: Includes subnets, internet gateways, and security groups.  
     
    

###### Figure 7: Auto Scaling Group with Application Load Balancer and Target Tracking Scaling Policy

### **Resource Breakdown and Configuration:**

#### **LaunchTemplate**

* **Purpose**: Configures the EC2 instances used in the Auto Scaling Group (ASG).
* **Configuration**:
  + **InstanceType**: t2.micro instance type, which is cost-effective for small workloads.
  + **KeyName**: Specifies an existing key pair (my-key-pair) for SSH access.
  + **SecurityGroupIds**: Uses an imported security group (WebSecurityGroup) to manage access.
  + **ImageId**: Specifies the Amazon Machine Image (AMI) ID (ami-071f488fe1f5b3cd9) for the instances.
  + **IamInstanceProfile**: Attaches an imported IAM role (AutoScalingInstanceProfile1) to manage permissions.
* **Usage Note**: The Launch Template allows for consistent instance configurations across the Auto Scaling Group, ensuring instances meet specific security and application requirements.

#### **Application Load Balancer (ALB)**

* **Purpose**: Distributes incoming traffic across multiple EC2 instances, ensuring high availability.
* **Configuration**:
  + **Subnets**: ALB is deployed in multiple subnets (PublicSubnet and PublicSubnet2), which are imported and enable public accessibility.
  + **SecurityGroups**: Attaches the WebSecurityGroup to control inbound traffic rules.
  + **Scheme**: Configured as internet-facing, making it accessible from the internet.
* **Availability Note**: Using an ALB ensures traffic is evenly distributed and can direct traffic away from any failed instance, improving uptime.

#### **TargetGroup**

* **Purpose**: Defines a group of instances that the ALB can route traffic to, based on health checks and load-balancing rules.
* **Configuration**:
  + **Target Type**: Configures instances as targets.
  + **Health Check**: Periodically verifies the health of instances, ensuring that only healthy instances receive traffic.
* **Reliability Note**: Target groups enhance reliability by redirecting traffic only to healthy instances, avoiding disruptions in the user experience.

#### **AutoScalingGroup**

* **Purpose**: Automatically adjusts the number of EC2 instances based on demand, ensuring the application can handle variable loads.
* **Configuration**:
  + **Launch Template**: Uses the specified LaunchTemplate for instance configuration.
  + **VPCZoneIdentifier**: Attaches to multiple subnets, allowing scaling across Availability Zones.
  + **DesiredCapacity, MinSize, and MaxSize**: Sets the desired, minimum, and maximum instance counts to control scaling behavior.
  + **HealthCheckType**: Configures EC2 and ALB health checks, ensuring instances are replaced if unhealthy.
  + **Subnets:** Deployed in PrivateSubnet1Ec2 and PrivateSubnet2Ec2 to maintain high availability and mitigate point failure.
* **Scalability Note**: The ASG ensures that the application remains responsive by adjusting resources in response to traffic changes, reducing costs during low-demand periods and enhancing performance during peak times.
* Target Tracking Policy: Initially we configured auto scaling with CPU utilization > 50%, but after experimenting with different parameters we found out that **RequestCountperTarget** was the best parameter to handle scale in and scale out. The threshold we have configured for this is **30.**

### **Security and Networking Components**

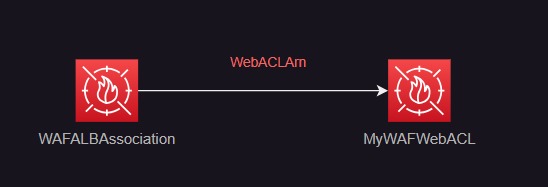
The template references multiple networking resources, including:

* **Subnets**: Public subnets to make the ALB accessible, ensuring the application can serve external traffic.
* **Security Groups**: The imported WebSecurityGroup controls access to the EC2 instances and ALB, enabling secure network boundaries.

# 3. Phase 2: Securing the Application

## 3.1 Set Up AWS WAF

* **Purpose**: AWS Web Application Firewall (WAF) is implemented to shield the application from common web attacks, including SQL Injection and Cross-Site Scripting (XSS), ensuring security and resilience against malicious traffic.



###### Figure 8: WAF SetUp

* **Configuration**:
  + **Attach WAF to the Load Balancer**: The WAF is associated with the Application Load Balancer (ALB), which directs internet traffic to the application. This setup allows the WAF to monitor and filter incoming requests based on the predefined rules.
  + **Use of AWS Managed Rules**:
    - **SQL Injection Protection**: AWSManagedRulesSQLiRuleSet is applied to detect and block SQL Injection attempts, which can potentially harm the database.
    - **XSS Protection**: AWSManagedRulesCommonRuleSet is used to prevent Cross-Site Scripting (XSS) attacks, which could inject malicious scripts into the application.
  + **Custom Rules**: A custom rate-limiting rule is implemented to limit requests from any single IP address to a specified threshold (e.g., 2,000 requests), reducing the risk of Denial of Service (DoS) attacks.
* **Monitoring**:
  + **WAF Logs in CloudWatch**: CloudWatch Metrics are enabled for the WAF WebACL, allowing detailed tracking of requests that are blocked or allowed. Specific CloudWatch alarms can be set to notify administrators of unusual activity, providing visibility and insight into security events.

## 3.2 Domain Management and SSL Configuration with Route 53

To provide a seamless, secure, and branded domain experience for our e-commerce platform, we integrated Amazon Route 53 with a custom domain purchased directly through Route 53. The domain,<https://shoppingcart-team9.xyz/>, enables AWS to handle DNS routing, SSL certificate management, and secure traffic flow to our Application Load Balancer (ALB).

1. **Domain Purchase in Route 53**:
   * The custom domain **shoppingcart-team9.xyz** was purchased directly from [**godaddy.com**](https://www.godaddy.com/)to serve as the primary domain for the e-commerce platform.
2. **Setting Up DNS Records in Route 53**:
   * After purchasing the domain, we created an **A Record (Alias Record)** in Route 53 and updated the nameservers in the domain in **godaddy** to route all traffic for **shoppingcart-team9.xyz** to the Application Load Balancer (ALB) associated with the platform. This setup directs all incoming traffic to our domain through the load balancer, which then distributes the traffic across EC2 instances within the Auto Scaling group.
3. **SSL Certificate Configuration using AWS Certificate Manager**:
   * To ensure secure data transmission, we requested an **SSL/TLS certificate** through **AWS Certificate Manager (ACM)**. During the certificate request, we specified **shoppingcart-team9.xyz** as the domain to validate ownership and enable HTTPS.
   * ACM provided a **CNAME record** for domain validation, which was automatically managed in Route 53, allowing ACM to verify our domain ownership.
4. **Attaching SSL Certificate to Load Balancer**:
   * After validation, the SSL certificate was attached to the Application Load Balancer, allowing for secure **HTTPS traffic** to the e-commerce platform. This configuration ensures that all communications between clients and the application are encrypted, protecting user data.
5. **Security Group Update for HTTPS Access**:
   * We updated the **security group** linked to the load balancer to allow inbound HTTPS traffic on port **443**. This ensures that users can securely connect to the platform using HTTPS, safeguarding sensitive data such as login credentials and payment information.

**Outcome**:

* **Branded Domain Experience**: The custom domain<https://shoppingcart-team9.xyz/> provides a professional and easily accessible URL for the e-commerce platform, enhancing brand recognition.
* **Enhanced Security**: By configuring an SSL certificate, all client-server communications are encrypted, protecting sensitive data.
* **Efficient Traffic Management**: With Route 53 managing DNS, we benefit from AWS’s scalable, low-latency, and reliable routing, ensuring the domain is accessible and performs well.

## 3.3 Encrypt Data in Transit

* **SSL/TLS Certificates for the Load Balancer**:
  + **AWS Certificate Manager (ACM)**: SSL/TLS certificates are issued via AWS Certificate Manager, securing all communications between clients and the web application hosted on the ALB. SSL/TLS encryption is configured on the ALB, which listens on HTTPS (port 443), ensuring that sensitive information, such as user credentials and payment details, is protected.
  + **CloudFormation Configuration**: The ACM certificate is attached to the ALB through the CloudFormation template by referencing the certificate’s Amazon Resource Name (ARN), ensuring consistent and automated security setup across deployments.

## 3.4 Database Connection Encryption:

* **SSL for RDS Connections**: SSL is enabled for connections between the web application and the RDS MySQL instance, ensuring that data is encrypted as it travels between the application and the database.
* **Implementation**:
  + **RDS CA Certificate**: The appropriate RDS CA certificate is downloaded and included in the application configuration.
  + **Database Connection Settings**: SSL is configured in the application code, specifying the path to the RDS CA certificate, ensuring that only secure connections are established with the database.

# 

# 

# 

# 

# 

# 

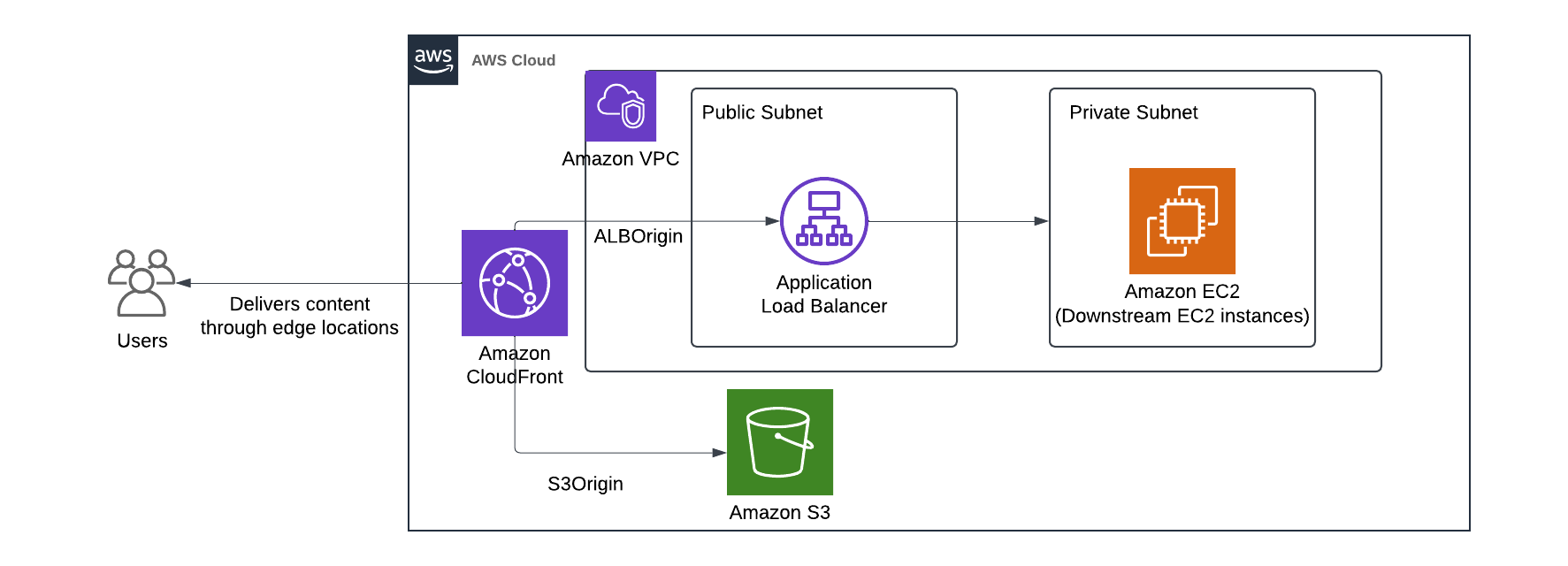
# 

# 

# 

# 4. Phase 3: Content Delivery and Performance Optimization

## 4.1 CloudFront CDN Integration

Amazon CloudFront was manually configured as a Content Delivery Network (CDN) for our application in order to improve content delivery performance and accessibility worldwide. By caching material at global edge locations, the integration reduces latency and speeds up content delivery.  


###### Figure 9: CDN Setup with CloudFront, S3, and ALB for Static and Dynamic Content

* **Distribution Details**:
  + **Distribution Domain**: The primary CloudFront distribution domain name is d2ley31395op8x.cloudfront.net.
  + **ARN**: The unique Amazon Resource Name (ARN) for this CloudFront distribution is arn:aws:cloudfront::207567787226:distribution/E325W6OZZ10YL6.
  + **Price Class**: Configured to use all edge locations for the best performance, ensuring optimal load times for users globally.
  + **Supported HTTP Versions**: The distribution supports HTTP/2, HTTP/1.1, and HTTP/1.0, which ensures compatibility with various client devices and maximizes delivery speed.
* **Origin Configuration**:
  + **Origins**: The CDN sources content from two origins:
    1. **S3 Bucket (cdn-s3-version-v1.s3.us-east-1.amazonaws.com)**: Stores static content, such as images, CSS, and JavaScript files, which can be cached for quick delivery.
    2. **Application Load Balancer (autosc-appli-wekvoax5rrv-1306503148.us-east-2.elb.amazonaws.com)**: Handles dynamic content generated by the backend application hosted on EC2 instances.
  + **Origin Type**: While dynamic material is delivered via the Elastic Load Balancer origin, which offers users flexibility in how content is presented to them, static files are handled by the S3 origin.

## 4.2 Performance Tuning and GZIP Compression

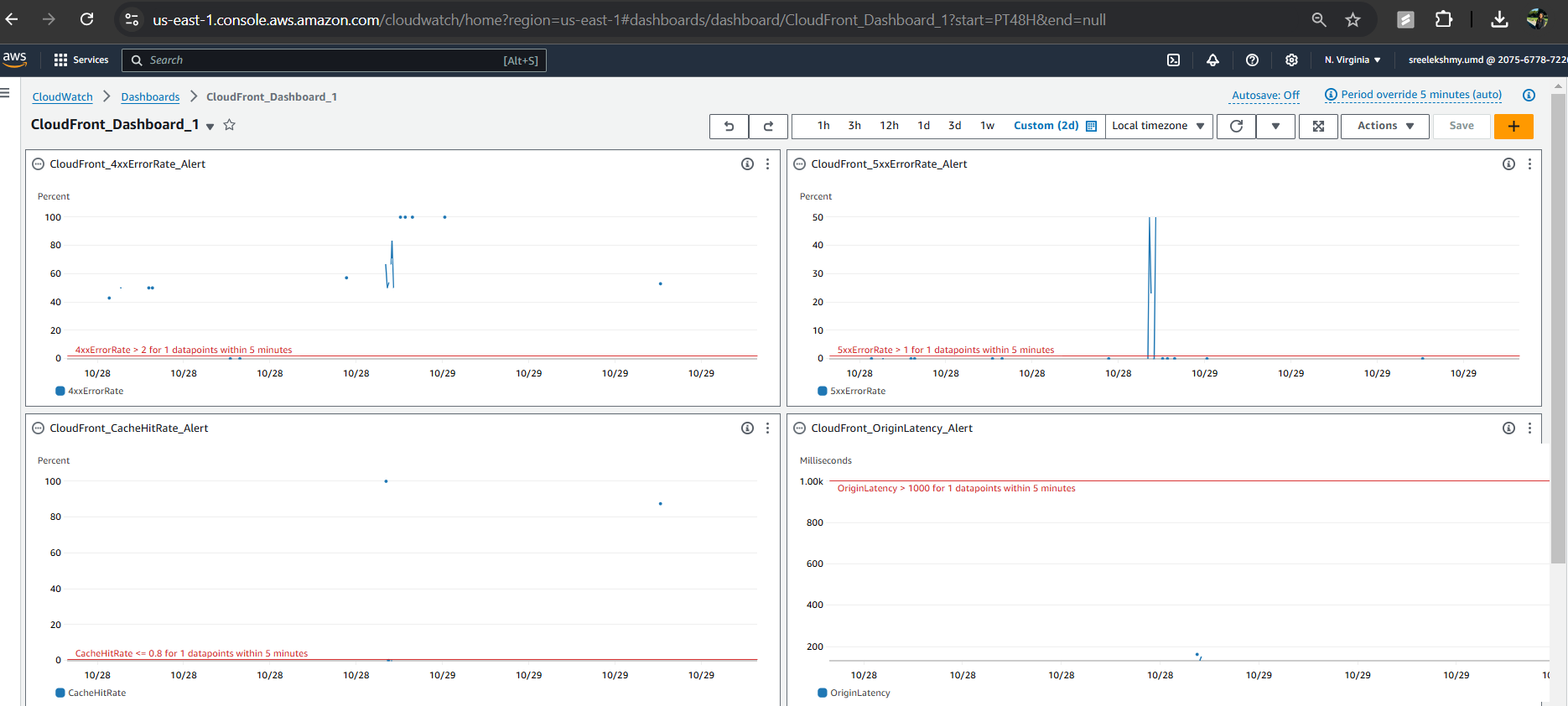
* **Dynamic Content**: We've set up CloudFront behaviors to route requests to the ALB origin and, when practical, cache dynamically created content in order to maximize the speed of dynamic pages.
* **GZIP Compression**: One can compress any text-based resource, including HTML, CSS, and JavaScript files, using GZIP compression; this reduces file sizes, which in turn speeds up load times and requires less bandwidth. Performance is noticeably improved, especially for customers with slower Internet connections.

## 4.3 Cache Configuration

* **Behavior Settings**:
  + **Path Patterns**: Specific path patterns are defined to handle different types of content, allowing for tailored caching strategies. For example:
    - Paths such as /admin/\*, /functions/\*, /includes/\*, and /users-area/\* are configured to redirect HTTP requests to HTTPS, ensuring secure data transfer.
    - Other paths like /product\_details.php, /products.php, and /cart.php are set with caching policies that optimize performance by caching dynamic pages while maintaining accuracy.
  + **Cache Policies**:
    - **Managed-AllViewer**: The majority of path patterns are covered by this policy, which lets cache-control headers from the origin determine edge caching behavior to maintain freshness and reduce latency.
    - **Managed-CachingOptimized**: Used to lessen the strain on the origin of requests for frequently requested resources with longer cache lifetimes.

## 4.4 CloudWatch Monitoring and Alarms

A number of CloudWatch alarms are configured in order to track CDN performance and proactively identify problems:



###### Figure 10: CDN CloudWatch Dashboard

* **CloudWatch Alarms**:
  + **4xx and 5xx Error Rates**: These alarms are set up to sound when error rates are above specified levels, signifying problems that could impact the user experience on the client or server side.
  + **High Request Volume Alert**: Notifies us of traffic spikes that might necessitate adjusting scalability when requests surpass 10,000 during a 5-minute period.
  + **Cache Hit Rate Alert**:This feature lets us adjust caching strategies for improved performance by alerting us if the cache hit rate drops below 80%.
  + **Origin Latency Alert**:This feature, which provides information about possible backend delays, is set to activate if the origin response time over 1,000 ms.

###### Figure 11: Sample CloudWatch Alarm Statistics

**Explanation of CloudWatch Alarm Statistics**

* **Current Status**:  
  The current **CacheHitRate** at the time shown is around **87.5%**, which means that 87.5% of requests are being served from the cache. The metric status is marked as **OK** in green, indicating that the CacheHitRate is above the threshold, so no alarm is currently triggered.
* **Brief Alarm Trigger**:  
  There was a brief period where the alarm was triggered, showing that the CacheHitRate dipped below the threshold of **80%** for a moment but quickly recovered.
* **Interpretation**:  
  This graph indicates that your CloudFront distribution is effectively caching content, as most requests are served from the cache, enhancing performance by reducing load on the origin and minimizing latency for end-users.

## 4.5 Security in CloudFront

**Access Control and Permissions:**

* **S3 Bucket Policies:** Set up a bucket policy that limits access to the S3 bucket so that only CloudFront may access and retrieve items. Because CloudFront offers a secure front layer, this configuration makes sure that users are unable to access the S3 bucket directly.
* **Origin Access Control (OAC)**: To restrict access to the S3 bucket solely via CloudFront, use Origin Access Control (OAC), also known as the previous Origin Access Identity (OAI). This stops users from accessing S3-stored material directly and avoiding CloudFront.

**HTTPS for Secure Data Transfer**:

* To ensure safe data transmission, set the Viewer Protocol Policy in CloudFront to "Redirect HTTP to HTTPS" or "HTTPS only" to require end users to connect via encrypted connections.

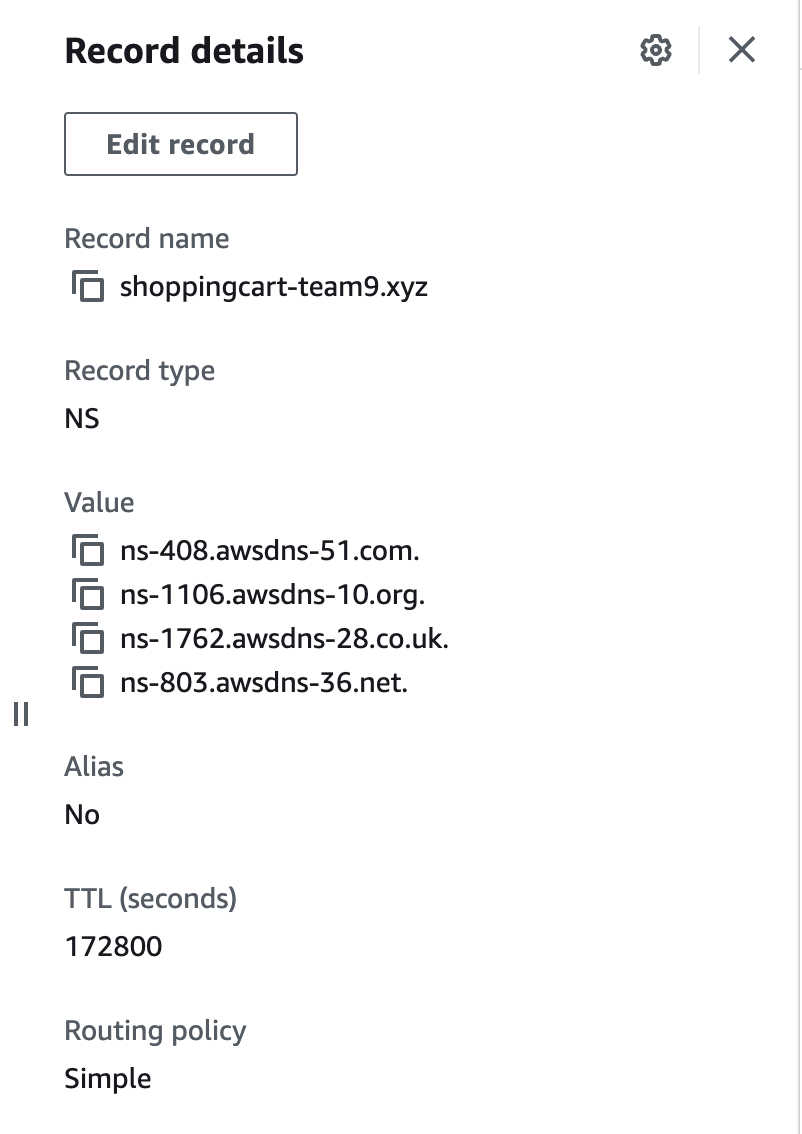
# 5. Phase 4. Testing and Monitoring

In this phase, we will cover the steps involved in testing the configuration and monitoring the performance of our deployed system to ensure it meets the expected requirements. We performed various tests using tools like **JMeter** to simulate load and stress on the system, as well as ensured that proper monitoring is in place using **AWS Web Application Firewall (WAF)** and **AWS CloudWatch**.

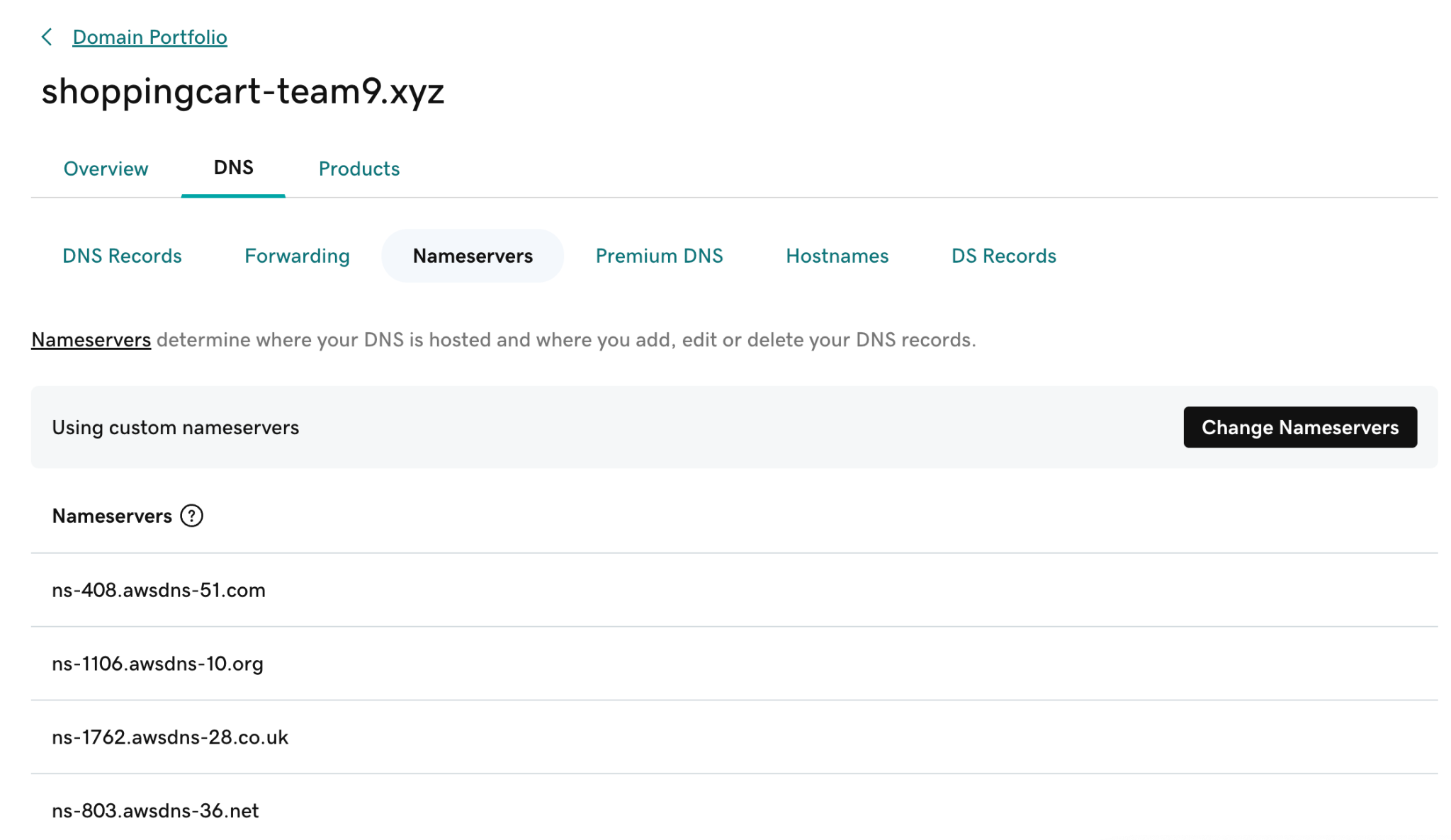
## 5.1. DNS and Domain Setup

After creating the **Route 53** hosted zone, we configured the DNS to point our domain to Route 53. Here are the key steps we took:

* **Route 53 Hosted Zone Created**: After setting up the hosted zone in Route 53, we received a set of four nameservers (NS records).
* **GoDaddy Domain Update**: These nameservers were copied and added to our GoDaddy domain settings, which ensures that the domain URL (e.g., https://shoppingcart-team9.xyz) points to the Route 53 DNS configuration.



###### Figure 12: Route 53 nameservers in the hosted zone



###### Figure 13: GoDaddy nameserver settings with the nameservers copied from Route 53

## 5.2. Load Testing with JMeter

We performed load testing using **Apache JMeter** to evaluate how well the application handles simultaneous user requests. Here is the setup we used:

* **JMeter Test Setup**:
  + **Thread Group Name**: "Thread Group"
  + **Number of Threads**: 250 (to simulate 250 users).
  + **Ramp-up Period**: 15 seconds (JMeter gradually increases the load to 250 users over this time).
  + **Loop Count**: 20 (each simulated user runs through 20 iterations of the test).
  + **Same User On Each Iteration**: Enabled.

##### 

###### Figure 14: JMeter Test

##### **Test Execution Results:**

The JMeter test generated 5000 entries in total (250 users \* 20 iterations). Below is a sample of the output showing a few rows:

| **timeStamp** | **elapsed** | **label** | **responseCode** | **responseMessage** | **threadName** | **success** |
| --- | --- | --- | --- | --- | --- | --- |
| 1.73015E+12 | 219 | HTTP Request | 200 | OK | Thread Group 1-3 | TRUE |
| 1.73015E+12 | 193 | HTTP Request | 200 | OK | Thread Group 1-4 | TRUE |
| 1.73015E+12 | 134 | HTTP Request | 200 | OK | Thread Group 1-4 | TRUE |
| 1.73015E+12 | 209 | HTTP Request | 200 | OK | Thread Group 1-7 | TRUE |
| 1.73015E+12 | 287 | HTTP Request | 200 | OK | Thread Group 1-8 | TRUE |

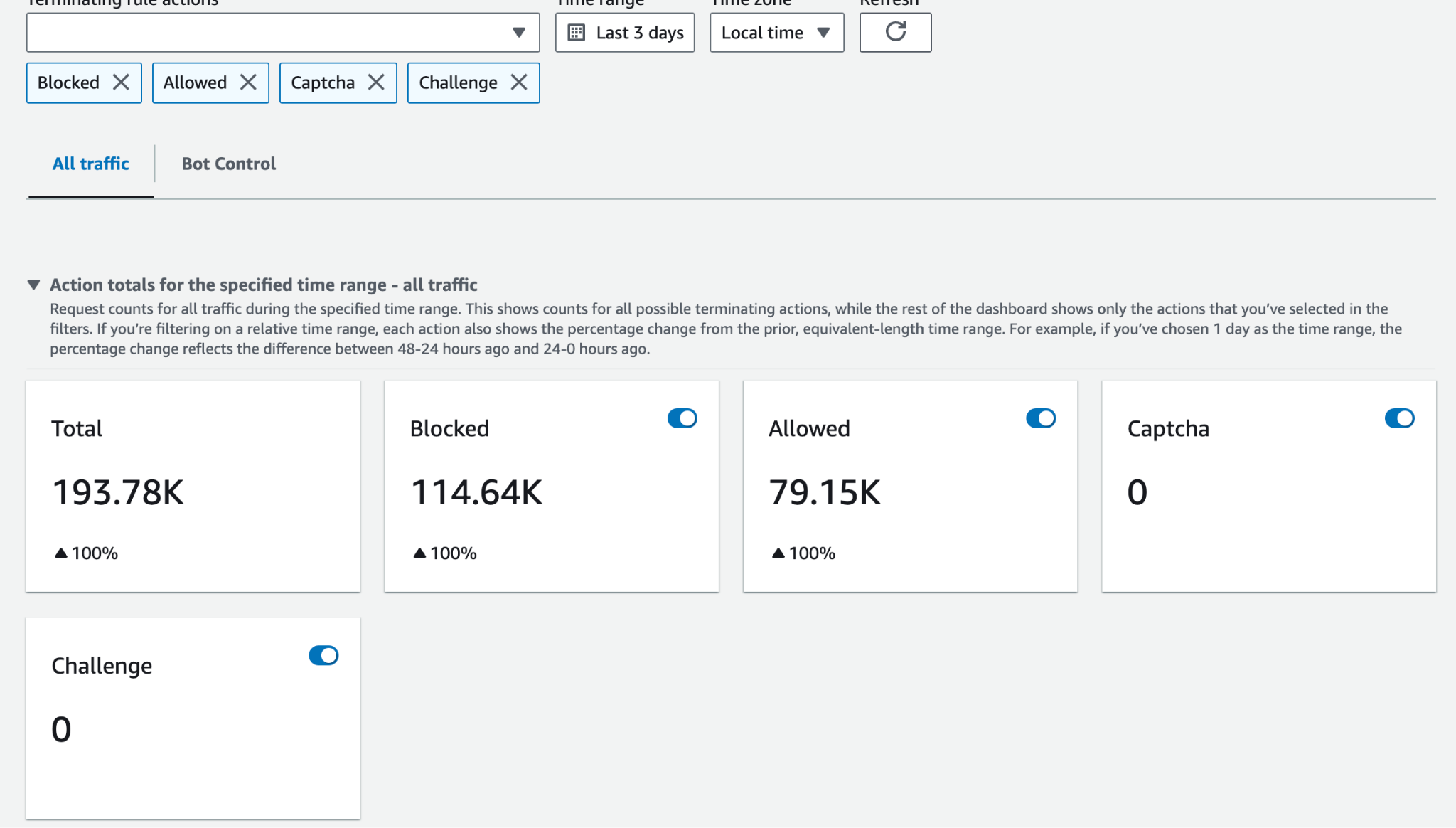
Each entry shows that the HTTP requests were successful (responseCode: 200 OK) and the response times (in milliseconds) were acceptable under the simulated load.

Output file of JMeter test results: [JMeter Output](https://drive.google.com/file/d/1FoctTwEO-RSWZXA11ysCvx2imo416--s/view?usp=sharing)

## 5.3. AWS WAF (Web Application Firewall)

To further secure our application and monitor traffic, we integrated AWS **Web Application Firewall (WAF)** with our system. This setup helps in filtering and monitoring incoming traffic to protect against common web attacks, such as SQL injection, cross-site scripting (XSS), and DDoS.

* **WAF Setup**:
  + Custom rules were created to block specific malicious patterns.
  + Regular traffic monitoring was enabled to capture any unusual traffic patterns and ensure the health of the application.

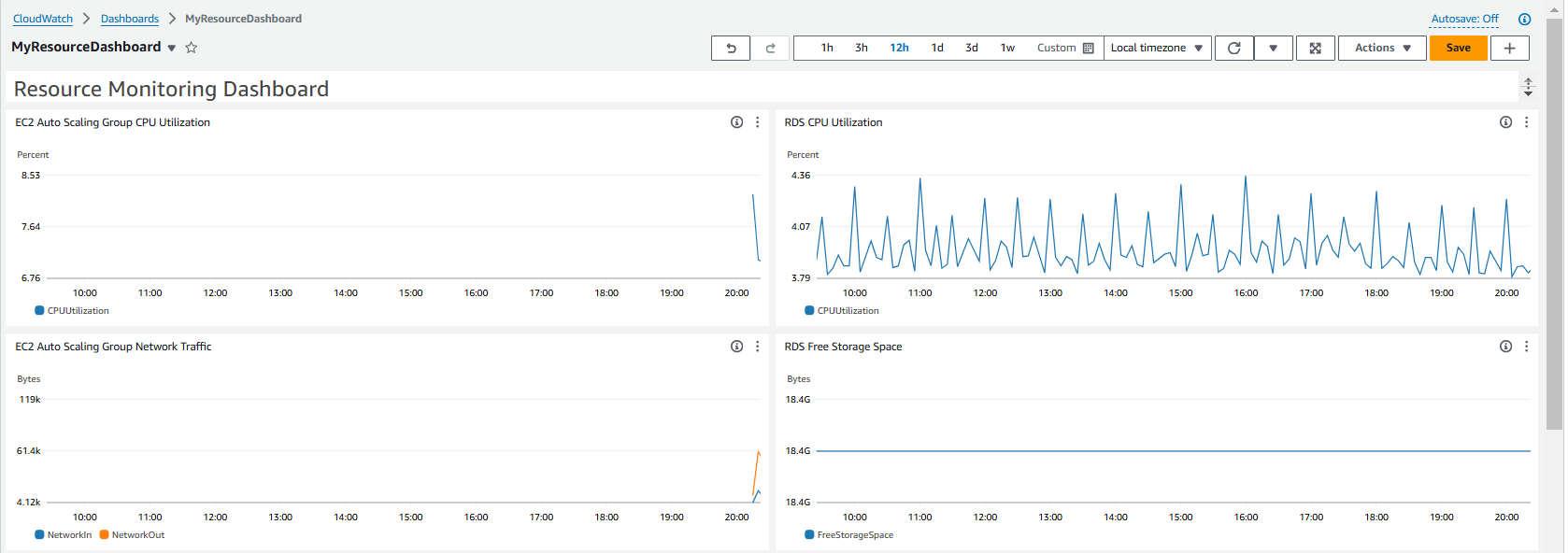


###### Figure 15: WAF Dashboard

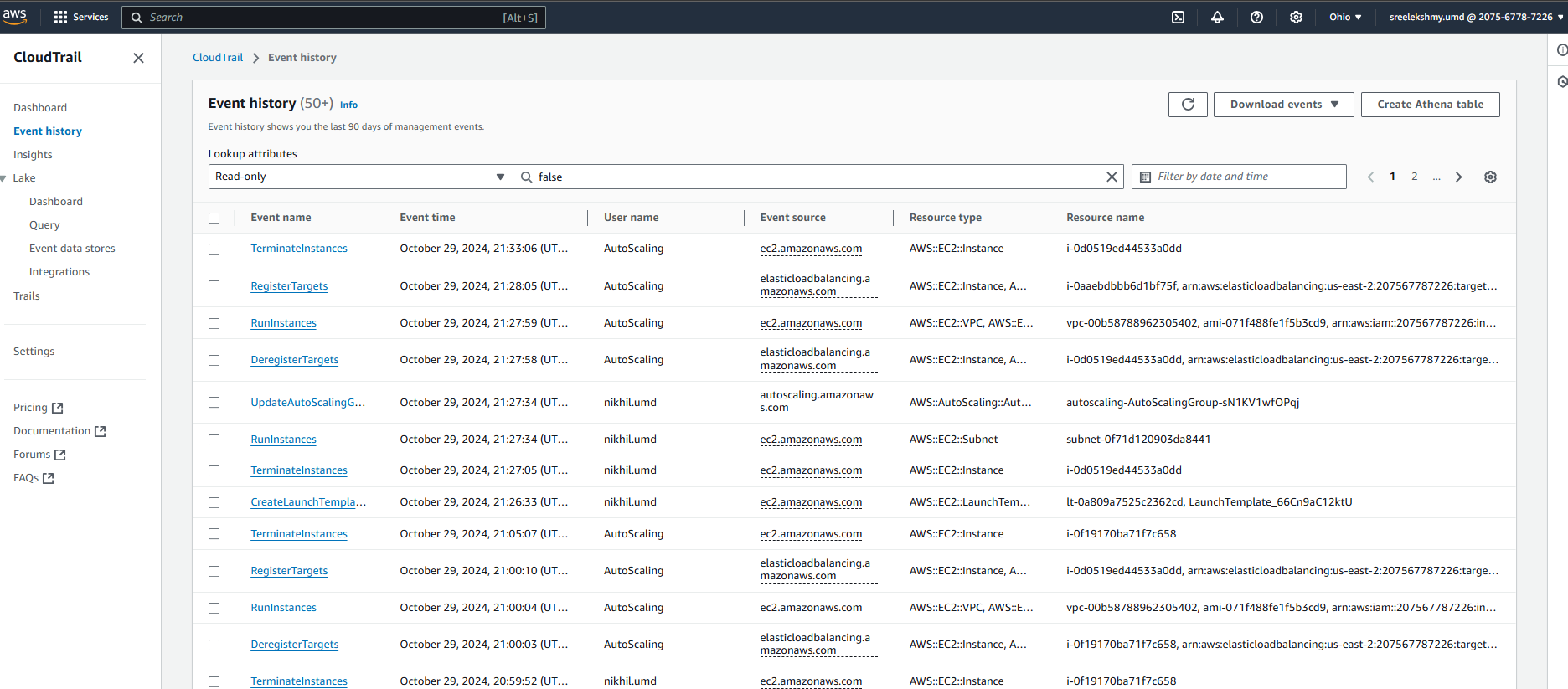
Above AWS WAF dashboard showing traffic monitoring or security rules in place. Various types of connections were tested to verify the effectiveness and functionality of the WAF.

## 5.4. Monitoring with AWS CloudWatch

To ensure continuous monitoring of the system's health and performance, **AWS CloudWatch** was used to track application metrics, such as CPU usage, memory consumption, and HTTP request latency. Alarms were set up to notify the team if any metric exceeds predefined thresholds, helping us stay ahead of potential issues.

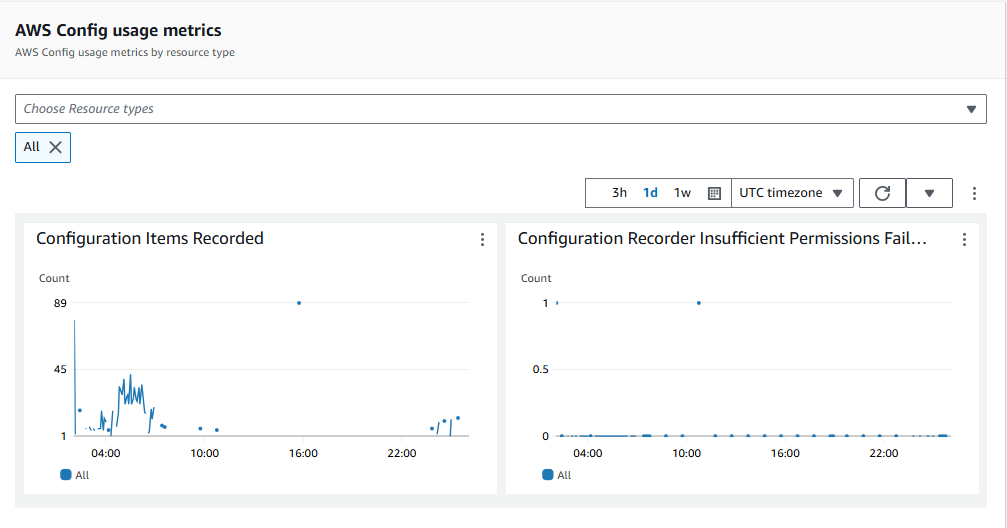


###### Figure 16: CloudWatch Dashboard

By using a combination of testing tools (JMeter) and AWS services (WAF and CloudWatch), we ensured that our application is secure, resilient, and performs well under significant load.  
  
To enhance the security and compliance of the e-commerce platform, **AWS CloudTrail** is configured to capture API activity across all AWS services. This enables logging, continuous monitoring, and auditing of security-related events. Key logs, such as authentication attempts, resource creation, modification, and deletion, are stored for tracking user activity and identifying any unauthorized actions.  


###### Figure 17: CloudTrail Log

In parallel, **AWS Config** is enabled to monitor and record all configuration changes to AWS resources. It provides continuous evaluation of configuration compliance, helping ensure resources maintain adherence to security and operational best practices. **AWS Config Rules** are applied to enforce secure configurations, generating alerts when any non-compliant changes occur, allowing for proactive responses to potential configuration drift.



###### Figure 18: AWS Config Usage Metrics

## 

## 5.5 Cost Analysis and Savings Recommendations

We used AWS Cost Explorer to review the usage of resources on the platform for possible savings. Based on current spending patterns, no Savings Plan recommendations are available since the average on-demand spend is below $0.10 per hour; substantial savings opportunities were not found. Even without recommendations, proactive usage review and thinking about Savings Plans or Reserved Instances are necessary in case the usage patterns grow so that we are prepared to apply long-term cost savings strategies as the platform grows.

###### Figure 19: Savings Plan Recommendations

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

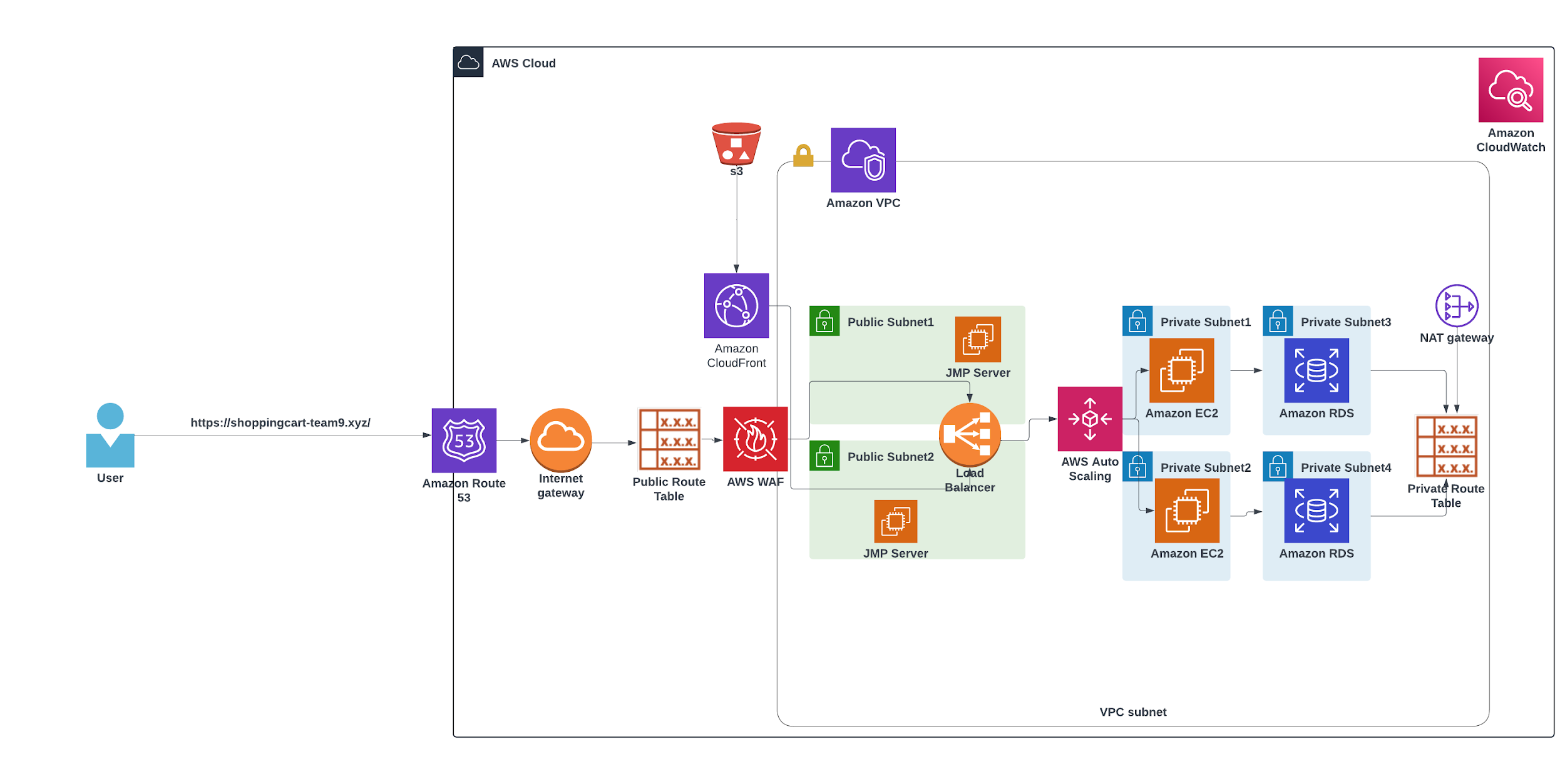
# 

# 

# 

# 

# 6. Architecture Design



###### Figure 20: Architecture Diagram

**AWS Architecture Diagram-** [**LucidChart Link**](https://lucid.app/lucidchart/f2bcc890-58ae-4260-992e-21dbfe091539/edit?view_items=DTdmXFA9P9-a%2CtVdmhKJ7n3e0%2Cf7dmtrFQDyz2%2Cy8dmEsThXbdX%2CxNem-IOTobxT%2C2zfmcBGF8xqX%2ChAfmnVqz61x_%2CJAfmHwCly.b~%2CrOdm2V5dcR4j%2CQOdmHDxoe.Hg%2CkQdm26ROlcQo%2CtRdm-2zHufeG%2CVUdmUnJiD1EC%2Cc3dm85u2e2pl%2CB4dmsBSqXvhA%2CP4dmhF-COJgX%2CGeemQ9zlYJui%2CGoemOPAbGx4A%2CBqemSf0ZpJDJ%2CmdgmSkXtWV5W%2CcuemESzLqRjb%2CLwemiDKBZXwg%2CZwemG8z7kd2m%2C2tfmRBPCkA9d%2C1Bfm4Ufx4wP~%2CjDfm.BDrxnh-%2CKDfmzU5RXmHY%2CtEfmnTS-B27c%2CcHfmAVd2Wfec%2CZIfmvYbwNU1h%2CeLfmwPHwRxVO%2CRSfm6o2SlvQp%2CTTfmeVzwy_x3%2CuUfmCcaB_gfb%2CjYfmNpXt~nqU%2Cx0fmP8apqXAV%2Ci1fmdee1Cvz~%2CE1fmV6MYyjJP%2CLkwmYQ1HOMTj%2CLkwmGBGZ3Ra0%2C4mwmmCYFi3W~%2Cmnwmpxi1HU_U%2CW5wmLr5CHRsQ&invitationId=inv_c4923e61-aaaf-4068-9cc9-17aa6cdbd10c)

The architecture for our e-commerce platform is designed to balance scalability, security, and high availability, leveraging several core AWS services. Below is an explanation of how the architecture components fit together.

1. **User and DNS Routing**
   * Users access the platform through the custom domain https://shoppingcart-team9.xyz, which is managed using **Amazon Route 53**. Route 53 handles DNS routing, directing user traffic to the appropriate resources, including **Application Load Balancer (ALB)**.
2. **Web Layer and Load Balancer**
   * Incoming requests from users are first routed through **AWS WAF** (Web Application Firewall) attached to the ALB. The WAF protects against common threats such as SQL Injection and Cross-Site Scripting (XSS) by filtering malicious traffic.
   * The **Application Load Balancer (ALB)**, positioned in the public subnets, evenly distributes traffic across **Amazon EC2 instances** in multiple private subnets for high availability and fault tolerance. The ALB also handles SSL/TLS termination, ensuring secure communication between the client and the backend using certificates managed by **AWS Certificate Manager** (ACM).
3. **Content Delivery Network (CDN) and Static Assets**
   * To optimize performance and reduce latency, static content such as images and CSS files are stored in **Amazon S3** and served via **Amazon CloudFront**, which acts as the CDN. This setup ensures faster delivery of static assets by caching them at AWS edge locations closer to users.
4. **Virtual Private Cloud (VPC) and Networking**
   * The platform is hosted within a **VPC** to ensure a secure and isolated environment. The VPC is subdivided into multiple subnets:
     + **Public Subnets** (Subnet 1 & 2): These contain the ALB and **Jump Servers (JMP Servers)**, which are used for secure administrative access to instances in private subnets.
     + **Private Subnets** (Subnet 1, 2, 3, and 4): These subnets host the EC2 instances and **Amazon RDS** (Relational Database Service) instances. The private subnets are not directly accessible from the internet, enhancing security.
5. **Compute Layer with Auto Scaling**
   * EC2 instances are configured in **Auto Scaling Groups**, which dynamically adjust the number of instances based on traffic patterns to meet the demand. Auto Scaling ensures high availability while optimizing resource utilization during peak and off-peak hours.
   * These EC2 instances are placed in multiple private subnets across different availability zones to ensure fault tolerance and disaster recovery.
6. **Database Layer**
   * The e-commerce platform uses **Amazon RDS** for database management, with instances deployed in private subnets (Subnet 3 and Subnet 4) for increased security. RDS is configured with **Multi-AZ** deployments to ensure high availability, and data is encrypted both at rest using **KMS** (Key Management Service) and in transit using SSL.
7. **Security and Traffic Management**
   * All outgoing internet traffic from instances in private subnets is routed through a **NAT Gateway**, allowing them to download updates and communicate with external services securely without exposing them directly to the internet.
   * **Security Groups** and **Route Tables** are configured to control traffic between the various components of the architecture, ensuring that only authorized traffic is allowed.
8. **Monitoring and Performance**
   * **Amazon CloudWatch** is used to monitor application performance and resource usage. Custom CloudWatch alarms are set up to trigger auto-scaling events and send alerts for any unusual activity or performance issues, ensuring smooth operation of the platform.

# 7. Conclusion

## 7.1 Lessons Learned

One important thing we learned was being able to see how AWS managed services simplify painful processes such as security, database administration, and scaling. It was rather valuable to gain experience on how these services should be correctly configured and connected.

One of the reasons for deploying an e-commerce application to AWS Cloud was to demonstrate how security configuration would be considered, especially when e-commerce data protection is supposed to be the biggest factor. Tools such as AWS WAF, encryption techniques, and access management securely followed the best practices of risk reduction.

Improvements and Insights: We implemented monitoring using CloudWatch and obtained immediate insights into user behaviors and application performance in real time. This gave us a far greater understanding of the application and better anticipation of scaling requirements. The cost management technologies also made it easy to plan resources and control costs.

## 7.2 Potential Improvements

While the current setup uses AWS Auto Scaling to dynamically adjust resources according to traffic demand, there are still areas for optimization. During scaling time, some requests may just be delayed, or even worse—cause an HTTP 500 error if they reach the instances when the capacity is fully utilized before new instances become operational. It would improve the user experience and overall platform reliability to reduce the downtime experienced during scaling. The following is a list of possible strategies to achieve the above:

**Decrease Cooldown Periods**: The cooldown period in Auto Scaling can be reduced to make instances scale up or down more quickly because of demand changes. This would potentially shorten the time for which capacity might not be sufficient.

**Use Predictive Scaling**: AWS Predictive Scaling allows the forecasting of demand according to past traffic trends, so resources can be changed before the demand increase occurs. Such proactive reconfiguration can prevent capacity shortfalls and diminish the likelihood of request failures.

**Configuring Warm Pools for EC2 Instances**: Introducing warm pools can significantly reduce startup times for new instances. By keeping instances in a ready state, they can be launched and begin serving requests faster, effectively minimizing the delay during scaling events.

**Implement Amazon ElastiCache**: Adding a caching layer using ElastiCache can offload some of the demand from the EC2 instances by serving frequently requested data directly from memory. This can help reduce the load on the backend, especially during traffic spikes, ensuring that more requests are successfully processed without depending on full scaling.

Furthermore, although AWS CloudFormation has been a vital tool in automating our infrastructure, Terraform may be used in its place. Terraform is more flexible, has a larger number of supported providers, and is, therefore, more fitting for multi-cloud environments in case further expansions or integrations with other cloud services become necessary. Its modular approach also provides more control and reusability across different environments.

A true fault tolerance, complementary to the current high availability setup, would demand redundancy not only across availability zones but also across multiple AWS regions. This would mean that EC2 and database replicas are set up in different regions in order to ensure that even if there is an outage of an entire region, mission-critical parts of the platform can still be kept up and running by the critical services.

These will enhance the scalability, availability, and fault tolerance of the platform, which will prepare it to handle bigger traffic surges with less disruption and ensure that access to key services is uninterrupted.

# 8. Appendices

* [**Architecture Diagram**](https://lucid.app/lucidchart/f2bcc890-58ae-4260-992e-21dbfe091539/edit?view_items=DTdmXFA9P9-a%2CtVdmhKJ7n3e0%2Cf7dmtrFQDyz2%2Cy8dmEsThXbdX%2CxNem-IOTobxT%2C2zfmcBGF8xqX%2ChAfmnVqz61x_%2CJAfmHwCly.b~%2CrOdm2V5dcR4j%2CQOdmHDxoe.Hg%2CkQdm26ROlcQo%2CtRdm-2zHufeG%2CVUdmUnJiD1EC%2Cc3dm85u2e2pl%2CB4dmsBSqXvhA%2CP4dmhF-COJgX%2CGeemQ9zlYJui%2CGoemOPAbGx4A%2CBqemSf0ZpJDJ%2CmdgmSkXtWV5W%2CcuemESzLqRjb%2CLwemiDKBZXwg%2CZwemG8z7kd2m%2C2tfmRBPCkA9d%2C1Bfm4Ufx4wP~%2CjDfm.BDrxnh-%2CKDfmzU5RXmHY%2CtEfmnTS-B27c%2CcHfmAVd2Wfec%2CZIfmvYbwNU1h%2CeLfmwPHwRxVO%2CRSfm6o2SlvQp%2CTTfmeVzwy_x3%2CuUfmCcaB_gfb%2CjYfmNpXt~nqU%2Cx0fmP8apqXAV%2Ci1fmdee1Cvz~%2CE1fmV6MYyjJP%2CLkwmYQ1HOMTj%2CLkwmGBGZ3Ra0%2C4mwmmCYFi3W~%2Cmnwmpxi1HU_U%2CW5wmLr5CHRsQ&invitationId=inv_c4923e61-aaaf-4068-9cc9-17aa6cdbd10c)
* [**Github project link**](https://github.com/clockno143/N-E_Commerce_Application)

# 9. References

* **Setting up Route 53:** [**https://www.youtube.com/watch?v=tXgOSt80Mtg**](https://www.youtube.com/watch?v=tXgOSt80Mtg)
* **AWS Documentation on CDN:** [**https://docs.aws.amazon.com/mediastore/latest/ug/cdns.html**](https://docs.aws.amazon.com/mediastore/latest/ug/cdns.html)
* **Setting up AWS CloudFront:** [**https://stackoverflow.com/questions/42531643/amazon-s3-static-web-hosting-caching**](https://stackoverflow.com/questions/42531643/amazon-s3-static-web-hosting-caching)