Report: Cloud-Based DDoS Mitigation Simulation Using AWS WAF and Shield

1. Introduction

Distributed Denial-of-Service (DDoS) attacks pose a significant threat to the availability and performance of cloud-hosted applications. These attacks aim to overwhelm network resources, rendering services inaccessible to legitimate users. As organizations increasingly migrate to cloud infrastructures, implementing robust DDoS mitigation strategies becomes critical to ensure business continuity and customer trust.

This project focuses on simulating a volumetric DDoS attack against an AWS-hosted web application and deploying mitigation mechanisms using AWS Web Application Firewall (WAF) and AWS Shield. The simulation is designed to emulate real-world attack patterns, enabling a practical assessment of the effectiveness of rate limiting, IP blocking, and bot control rules. Furthermore, the project integrates monitoring through Amazon CloudWatch and automated response using AWS Lambda to dynamically respond to anomalous traffic.

By systematically testing and refining these controls, this project demonstrates how AWS native services can be leveraged to safeguard web applications against evolving DDoS threats.

2. Objective

The primary objective of this project is to simulate a volumetric Distributed Denial-of-Service (DDoS) attack on an AWS-hosted web application and implement a layered mitigation strategy using AWS-native services. The simulation targets a web application deployed behind an Application Load Balancer (ALB), with the attack traffic generated using the **hping3** tool to emulate high-volume TCP-based flooding. The project involves configuring AWS Web Application Firewall (WAF) with rate-based rules, IP blocking mechanisms, and bot mitigation to detect and block malicious traffic. It also includes the use of Amazon CloudWatch for real-time monitoring and visualization of traffic trends through custom dashboards. Additionally, AWS Lambda functions are deployed to automatically ban IP addresses exhibiting suspicious behavior. The overarching goal is to evaluate the effectiveness of these mitigation strategies and optimize the configuration to enhance the application's resilience against DDoS attacks.

Key goals include:

- **Simulate Realistic DDoS Attack Traffic:**
  Generate high-volume TCP flood traffic using the *hping3* tool to replicate a realistic volumetric DDoS attack scenario targeting an AWS-hosted web application. This allows for a controlled environment to assess the application's behavior under stress and identify potential vulnerabilities in the current network configuration.

- **Implement Scalable WAF Rules:**
  Configure AWS WAF with rate-based rules, IP blocking, and bot mitigation to identify and block malicious traffic without affecting legitimate users. The rules are designed to adapt to traffic anomalies and protect the application layer from both volumetric and logic-based attacks.

- **Establish Real-Time Monitoring and Analytics:**
  Set up Amazon CloudWatch to capture, analyze, and visualize traffic patterns and security events through custom dashboards and metrics. This visibility enables rapid detection of anomalies, performance bottlenecks, and attack indicators, supporting timely incident response.

- **Enable Automated Threat Response:**
  Utilize AWS Lambda functions to dynamically detect and block IP addresses associated with suspicious or abnormal request behavior. Automation minimizes response time and reduces the operational burden of manual intervention during active attack phases.

- **Evaluate and Optimize Mitigation Strategy:**
  Continuously assess the effectiveness of the implemented controls, refine rule sets, and improve system resilience based on observed performance during the attack simulation. This iterative process ensures that the security posture remains aligned with evolving threat landscapes and operational requirements.

3. Tools And Techniques

This project utilized a combination of open-source utilities and AWS-native services to simulate, detect, and mitigate Distributed Denial-of-Service (DDoS) attacks. Each tool was selected to fulfill a specific functional requirement within the attack lifecycle and defense strategy.

- **hping3:**
  Used to simulate high-volume TCP flood traffic by crafting and sending packets to replicate a volumetric DDoS attack.

- **Amazon EC2:**
  Provided scalable virtual servers to host the web application and run the attack simulation environment.

- **Application Load Balancer (ALB):**
  Distributed incoming traffic across backend instances while integrating with AWS WAF for edge security enforcement.

- **AWS WAF:**
  Enabled configuration of rate-based rules, IP blocking, and bot mitigation to filter and block malicious HTTP(S) requests.

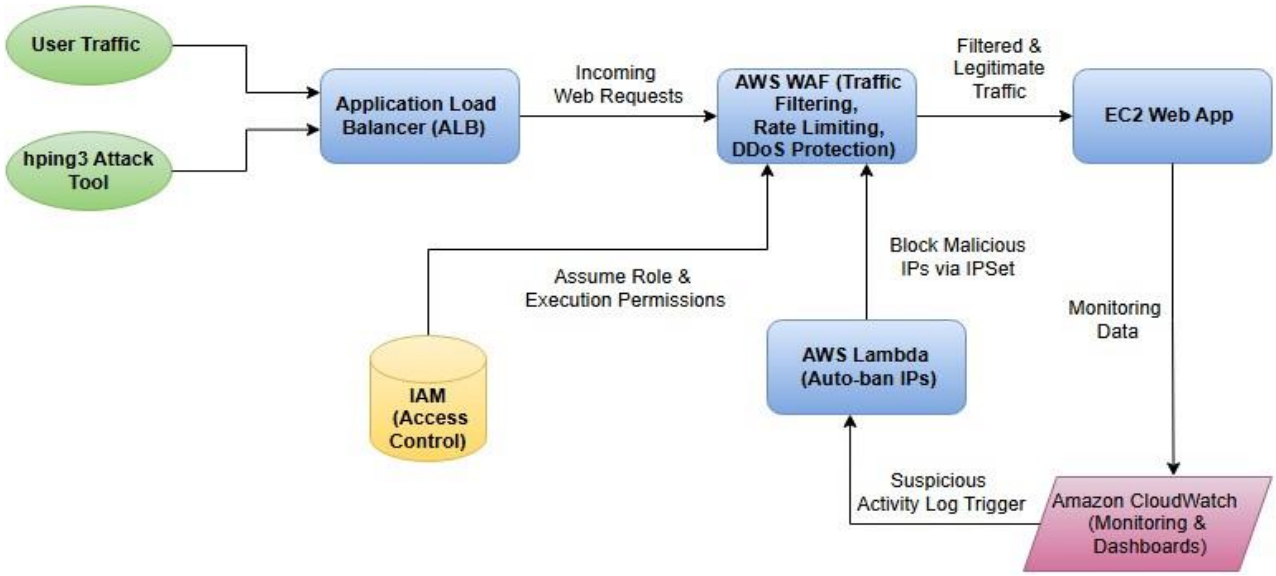
- **AWS Shield:**
  Offered infrastructure-level DDoS protection, with options for standard baseline defense or advanced analytics and support.

- **Amazon CloudWatch:**
  Monitored network traffic and system metrics in real time, supporting visualization through custom dashboards.

- **AWS Lambda:**
  Automated the detection and blocking of suspicious IP addresses by triggering security actions based on CloudWatch alerts.

- **IAM (Identity and Access Management):**
  Managed and enforced least privilege access controls for all users and services involved in the simulation.

4. Workflow Diagram



5. Methodology

This section outlines the systematic approach followed to simulate a volumetric DDoS attack on an AWS-hosted web application and to implement layered mitigation strategies using AWS services. Each phase—from environment setup to attack simulation, defense configuration, and real-time response—was designed to replicate real-world DDoS scenarios while evaluating the effectiveness of AWS-native protections such as WAF, Shield, CloudWatch, and Lambda automation. The methodology ensures a practical and measurable assessment of cloud-based DDoS defense capabilities.

3.1 Environment Setup

The initial phase focused on establishing a foundational infrastructure necessary for the DDoS mitigation simulation. This involved provisioning an EC2 instance to host the web application and configuring network components to ensure secure and efficient traffic flow.

An Application Load Balancer (ALB) was deployed to distribute incoming traffic, providing fault tolerance and scalability. The environment was validated for accessibility to ensure readiness for subsequent attack simulation and mitigation steps.

- **Launch EC2 Instance for Web Application**

  The process began with provisioning an Amazon Linux 2 EC2 instance named **DDoS-Test-WebApp** within the pre-configured Virtual Private Cloud (**MyVPC**). The instance was deployed in the public subnet **MySubnet** to ensure internet accessibility for web traffic. During instance creation, a dedicated security group was configured and attached to enforce strict network access controls: SSH (port 22) was permitted from any IP address to allow secure administrative access, while HTTP traffic (port 80) was allowed from all sources to enable public access to the hosted web application. This integrated setup of compute resources and network security established a robust foundation for subsequent DDoS simulation and mitigation activities.

  EC2 Configuration Details:

  i) **VPC Selection:**

  The EC2 instance **DDoS-Test-WebApp** was launched within the pre-existing Virtual Private Cloud (VPC) named **MyVPC** to ensure network isolation and controlled routing.

  ii) **Subnet Assignment:**

  The instance was assigned to the public subnet **MySubnet** within **MyVPC**, providing direct internet access through an associated Internet Gateway.

  iii) **Security Group Creation and Attachment:**

  A new security group was created with the following inbound rules:

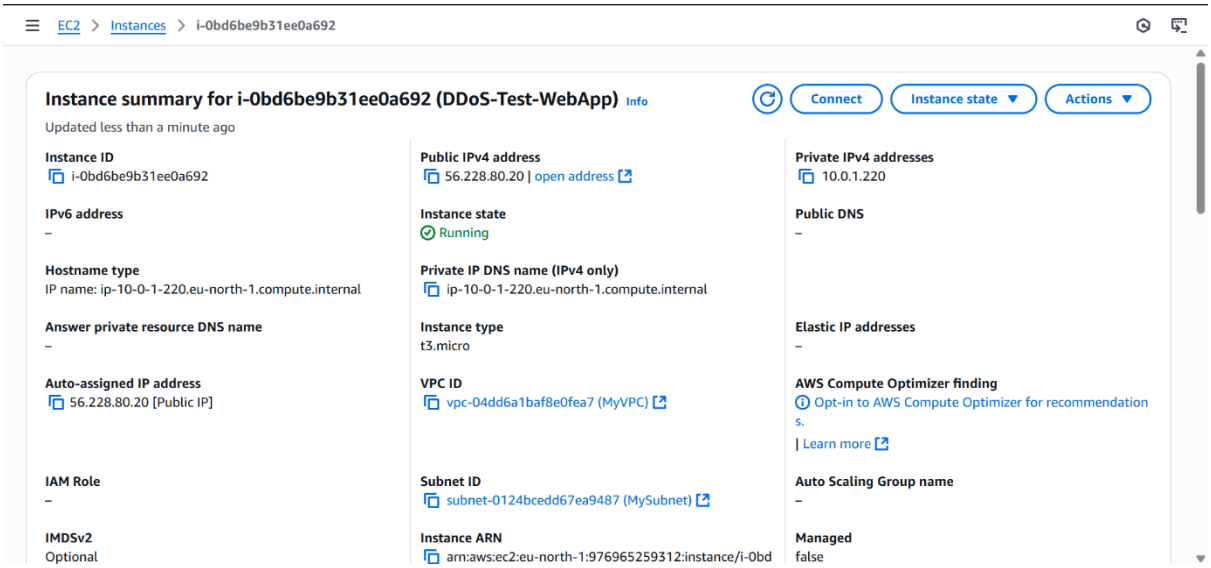
  (a) SSH (Port 22): Allowed from any IP address (0.0.0.0/0) to enable remote administrative access.

  (b) HTTP (Port 80): Allowed from any IP address (0.0.0.0/0) to permit public web traffic to the instance.

iv) **Security Group Association:**

This custom security group was attached to the **DDoS-Test-WebApp** instance at launch to enforce the defined network access controls.

(Deploy EC2 instance with the specified VPC, subnet, and security group settings.)



- **Configure Target Group and Application Load Balancer (ALB)**

  To route incoming traffic efficiently and support load balancing functionality, a Target Group was first created and configured to register the EC2 instance. This was followed by the provisioning of an Application Load Balancer (ALB) within the same VPC. The ALB serves as the primary entry point for client traffic, forwarding requests to healthy targets defined in the target group.

  Purpose: The objective of this step is to simulate a scalable, production-like environment where traffic flows through a load balancer and is intelligently directed based on health status and target availability. Creating the target group first allows seamless registration of backend instances, while the ALB facilitates traffic inspection and enforces WAF-based protections for HTTP-layer DDoS mitigation.

  Target Group Configuration

  A target group named yash-target-group was created to enable backend traffic routing for the Application Load Balancer (ALB). The configuration details are as follows:

i) **Target Type: Instance**

The target group was configured to register EC2 instances directly using their instance IDs, allowing direct communication with the compute resource.

ii) **Name: yash-target-group**

The group was named appropriately for identification and management purposes.

iii) **Port: 80**

The target group forwards HTTP traffic on port 80, aligning with the default port for the deployed web application.

iv) **VPC: MyVPC**

The group operates within the Virtual Private Cloud (VPC) named MyVPC, ensuring isolated and controlled networking.

v) **Health Check Configuration:**

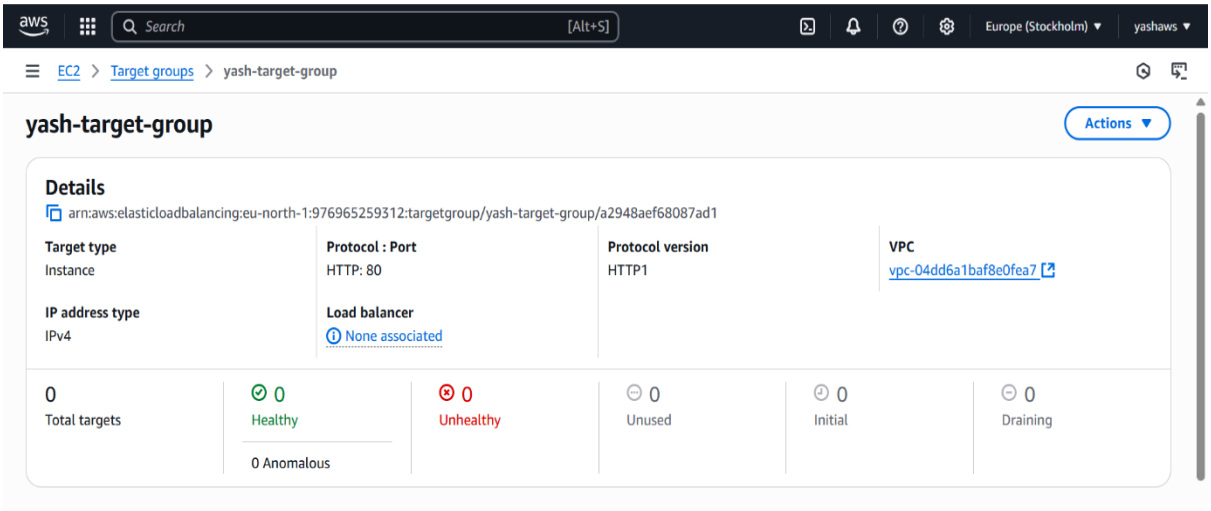
(a) Protocol: HTTP

(b) Path: /

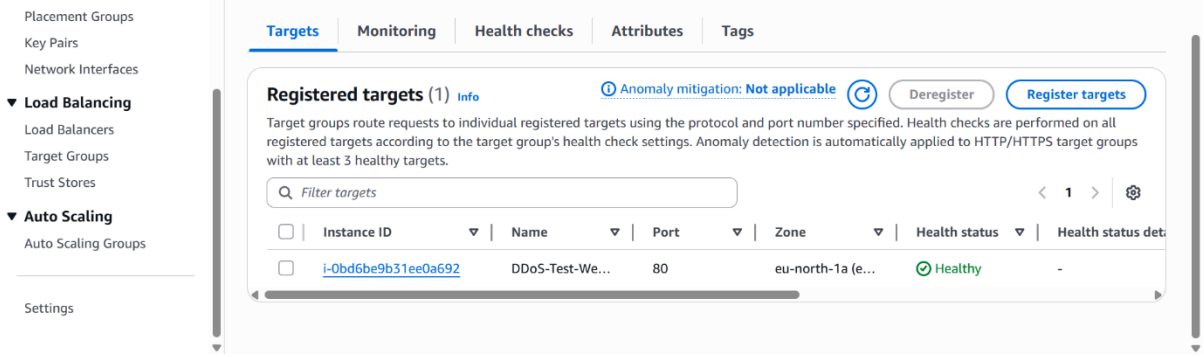
vi) **Registered Target:**

The EC2 instance with ID i-0bd6be9b31ee0a692 (DDoS-Test-WebApp) was successfully registered to the target group. This enables the ALB to forward incoming requests directly to the instance while monitoring its health status.

(Target group 'yash-target-group' configured for HTTP port 80 in VPC 'MyVPC')



(EC2 instance i-0bd6be9b31ee0a692 registered as active in the target group)



Application Load Balancer Configuration

An Application Load Balancer named yash-alb was provisioned with the following specifications**:**

i) Scheme: Internet-facing

Configured to accept incoming traffic from the internet, allowing public access to the web application.

ii) VPC: MyVPC

The ALB was deployed within the same Virtual Private Cloud to maintain network segmentation and control.

iii) Subnets:

Two subnets from different availability zones were selected to ensure high availability and fault tolerance.

iv) Security Group:

The existing security group launch-wizard-1 was associated, which permits inbound traffic on necessary ports while maintaining security controls.

v) Listener:

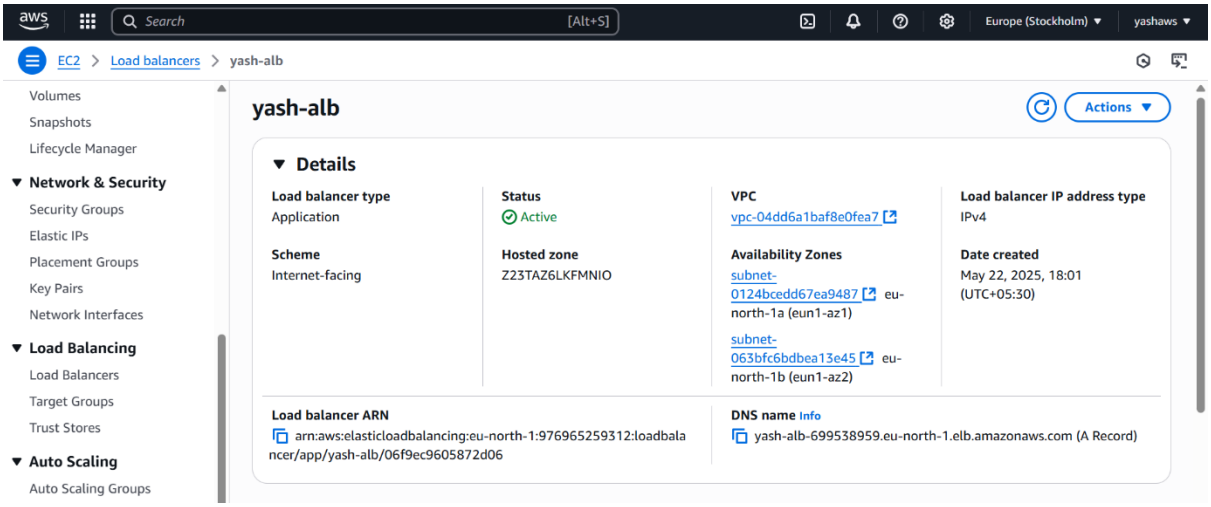
Configured to listen for HTTP traffic on port 80, enabling the ALB to accept and route web requests.

vi) Target Group:

The previously created target group yash-target-group was attached to route incoming requests to the registered backend EC2 instance(s).

This setup ensures the ALB can distribute incoming traffic efficiently, maintain high availability, and integrate with AWS WAF and Shield for DDoS mitigation.

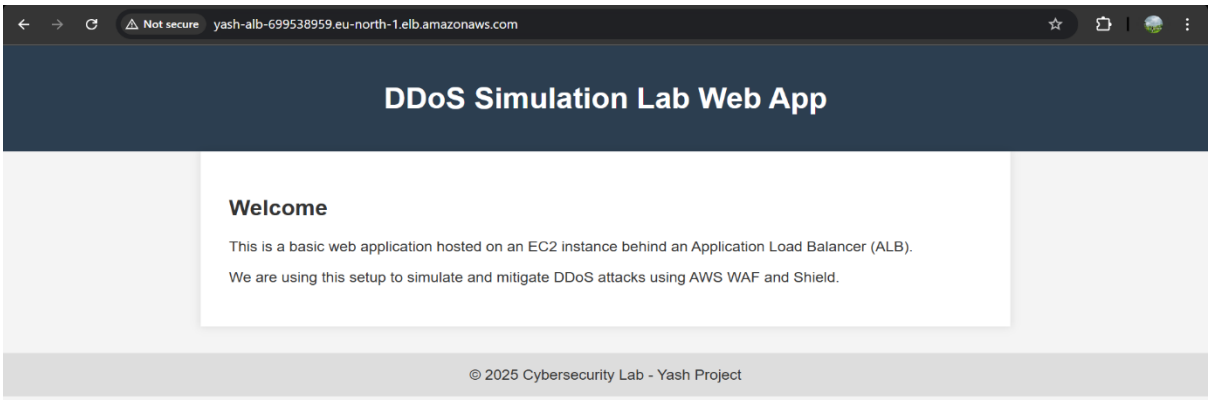
(Internet-facing ALB 'yash-alb' created with HTTP listener on port 80 and linked to target group.)



- **Verify Load Balancer DNS Accessibility**

To confirm the successful deployment and configuration of the Application Load Balancer, the ALB's DNS endpoint was accessed via a web browser or HTTP client. This step validates that the ALB is publicly reachable, correctly forwarding incoming requests to the registered EC2 instance, and serving the deployed web application.

(Web page successfully served via ALB DNS endpoint accessed in browser.)



3.2 Attack Simulation using hping3

This phase involved simulating a volumetric DDoS attack on the deployed web application using **hping3** to generate high-rate traffic. The purpose was to monitor the Application Load Balancer's performance metrics under attack, including response time, request volume, new connection count, peak LCU usage, and overall consumed capacity, to evaluate the system's resilience.

- **Simulate DDoS Attack Using hping3**

  The purpose of this step is to generate controlled, high-volume network traffic to emulate a real-world DDoS attack scenario. This enables testing of the system's capacity to handle malicious traffic and stresses the infrastructure for evaluation.

  sudo hping3 -S --flood -p 80 yash-alb-699538959.eu-north-1.elb.amazonaws.com

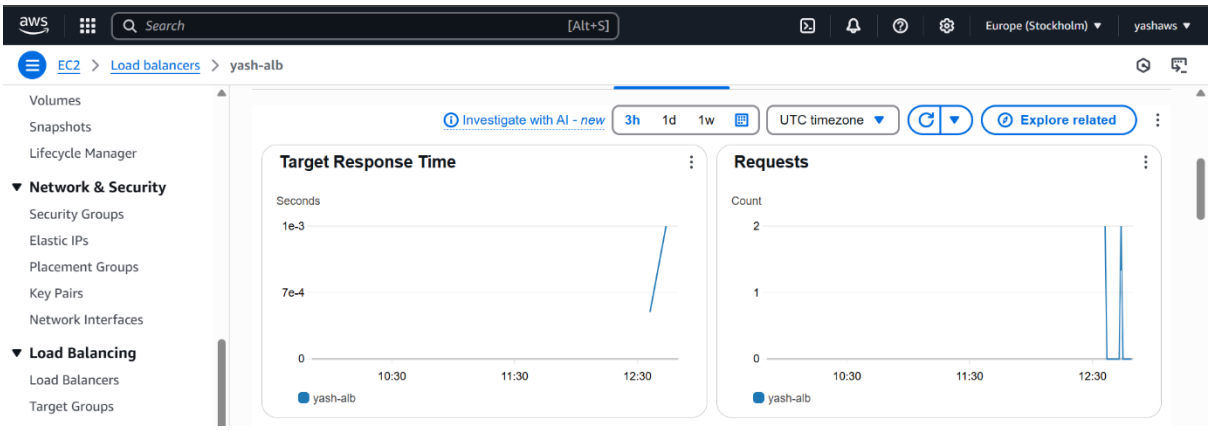
  The command sends a continuous flood of TCP SYN packets (-S --flood) to port 80 (-p 80) of the Application Load Balancer (yash-alb-699538959.eu-north-1.elb.amazonaws.com). Running with sudo allows raw packet transmission. This simulates a SYN flood attack, overwhelming the load balancer with connection requests to test its resilience.

  (hping3 tool simulating SYN flood attack on ALB DNS endpoint over port 80.)

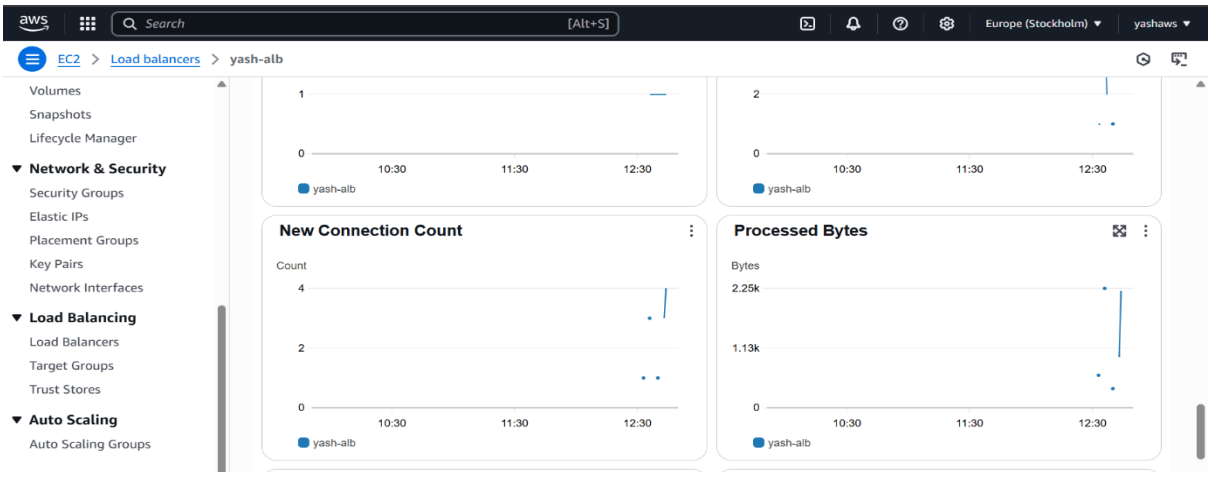
- **Monitor Load Balancer Metrics and Target Response**

  This step involves continuous observation of the Application Load Balancer's performance indicators—including response time, request count, new connection rate, peak Load Balancer Capacity Units (LCU), and overall capacity utilization. The objective is to quantify the impact of the attack and identify any performance degradation or potential points of failure.

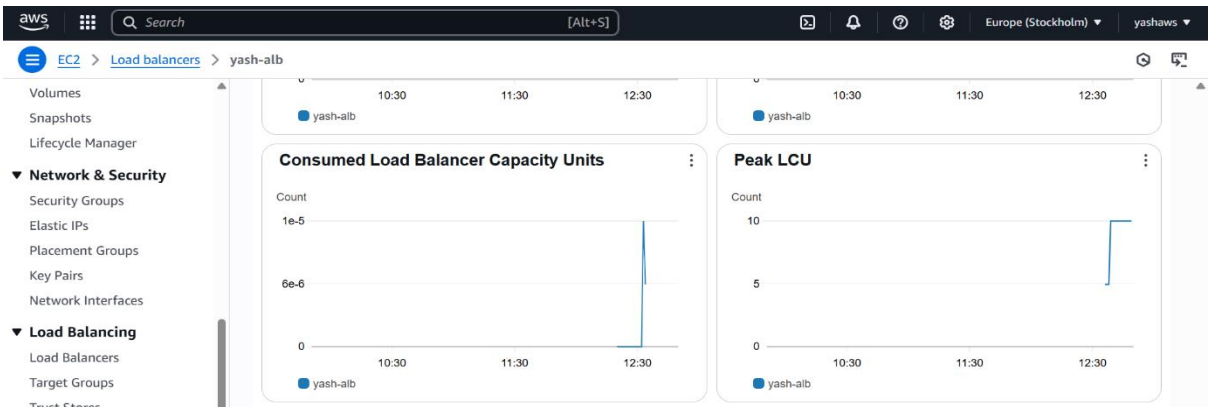
(CloudWatch metrics showing increased request count and target response time during simulated attack.)



(Display of increased **new connection count**, reflecting the abnormal volume of incoming traffic.)



(Metrics highlighting spike in **peak LCU** and **consumed load balancer capacity**, confirming resource stress under simulated DDoS conditions.)



3.3 Deployment of AWS WAF and Shield

This phase focuses on deploying AWS WAF to actively mitigate the volumetric DDoS attack simulated in the previous phase. AWS WAF was configured with custom rulesets—including rate-based rules and IP blocking—to detect and respond to malicious traffic. Additionally, WAF logging was enabled with Amazon CloudWatch Logs as the logging destination to capture sampled web requests, enabling detailed traffic analysis and continuous monitoring. This combination enhanced the application's resilience and supported proactive threat detection and mitigation.

- **Create and Associate Web ACL with Load Balancer, configure rules**

  In this initial step, an AWS Web Application Firewall (WAF) Web Access Control List (Web ACL) named DDoS-WebACL was created and associated with the Application Load Balancer yash-alb. The Web ACL was configured with a comprehensive set of managed and custom rules designed to filter and block malicious traffic before it reaches the application layer.

  Purpose:

  The primary objective of this step is to establish a foundational security layer that enforces traffic filtering and threat mitigation at the edge. By associating the Web ACL with the load balancer and implementing rules such as rate limiting, IP reputation filtering, SQL injection prevention, and manual IP blocking, this step significantly reduces the risk of volumetric and application-layer DDoS attacks impacting the web application.

  Technical Details:

  1. **Web ACL**

     Name: DDoS-WebACL

     Resource Type: Regional (to support ALB in the selected AWS region)

     Associated Resource: Application Load Balancer named yash-alb

  2. **WAF Rules**

     The DDoS-WebACL was configured with multiple AWS Managed Rule Groups alongside custom rules to provide layered protection against various attack vectors:

a. **ManualBlockList (Custom Block Rule):**
   Blocks specific IP addresses identified as malicious or suspicious based on manual threat intelligence.

b. **RateLimitRule (Rate-Based Rule):**
   Automatically blocks IP addresses that exceed a predefined request threshold within a 5-minute window, effectively throttling potential volumetric attacks.

c. **AWSManagedRulesCommonRuleSet:**
   Provides broad protection against common web exploits such as cross-site scripting (XSS), SQL injection, and other OWASP Top 10 vulnerabilities.

d. **AWSManagedRulesAmazonIpReputationList:**
   Blocks IP addresses with a history of suspicious or malicious behavior as maintained by AWS threat intelligence.

e. **AWSManagedRulesAnonymousIpList:**
   Mitigates risks from anonymous proxies and VPNs often used to obfuscate attacker identity.

f. **AWSManagedRulesAdminProtectionRuleSet:**
   Protects administrative endpoints from unauthorized access and common exploitation attempts.

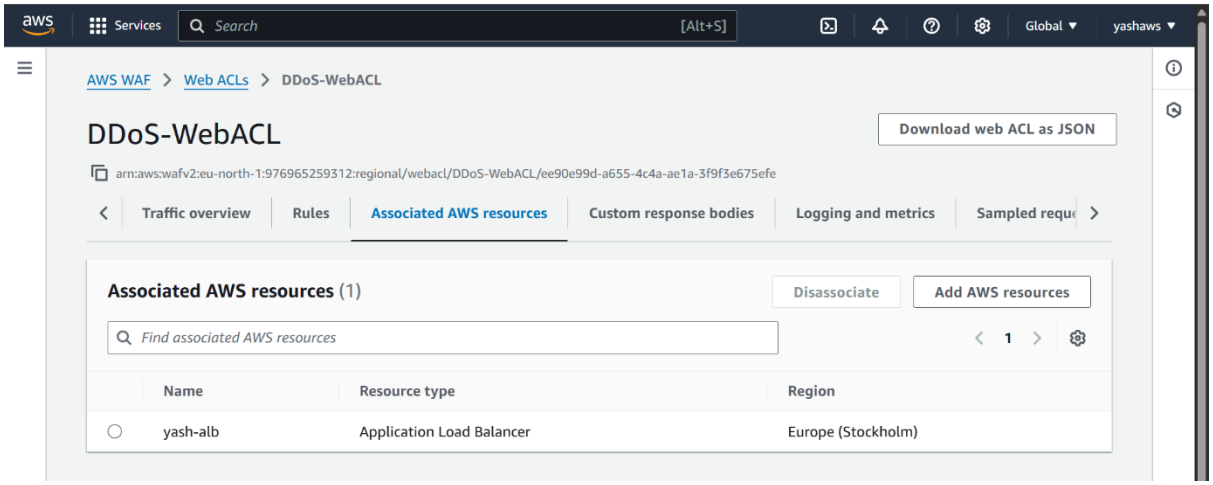
g. **AWSManagedRulesSQLiRuleSet:**
   Specifically targets and blocks SQL injection attack patterns.

h. **AWSManagedRulesKnownBadInputsRuleSet:**
   Filters out requests with known malicious input patterns, enhancing security posture against known exploits.

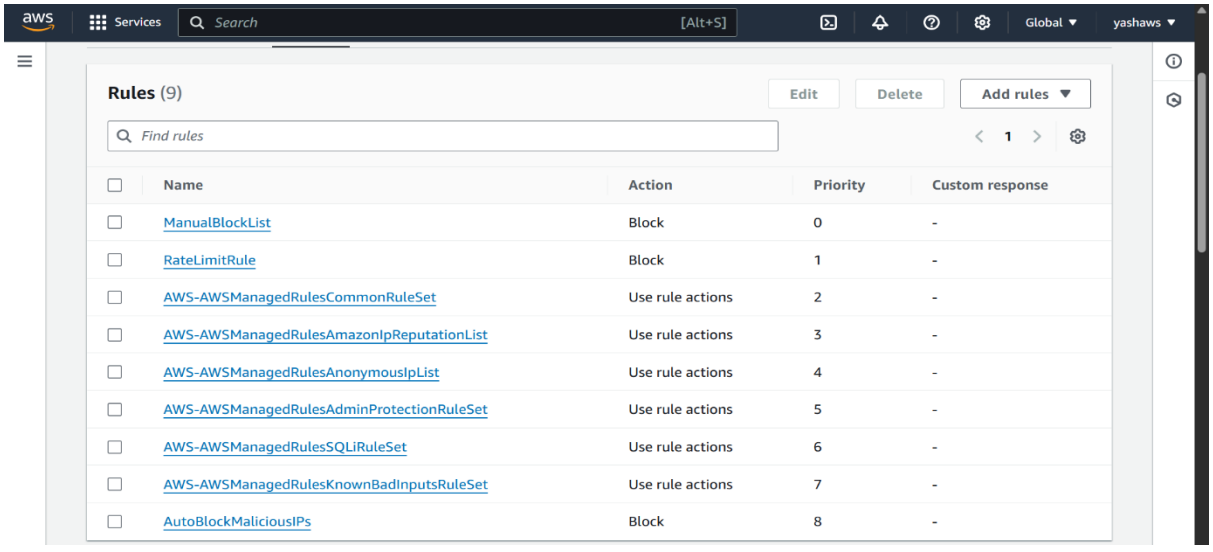
i. **AutoBlockMaliciousIPs (Custom Block Rule):**
Custom automation-driven rule that blocks IPs identified through Lambda-based detection logic.

This combination of managed and custom rules enables a robust defense-in-depth strategy, reducing attack surface and mitigating potential DDoS and application layer threats before they impact the infrastructure.

(Shows the DDoS-WebACL Web ACL associated with the Application Load Balancer yash-alb, enforcing security policies at the edge.)



(Displays the comprehensive set of AWS Managed and custom WAF rules configured within the DDoS-WebACL to mitigate various attack vectors.)



- **Configure WAF Sampled Requests Logging**

  In this step, AWS WAF logging was enabled to capture sampled web requests for the DDoS-WebACL, with logs being sent to Amazon CloudWatch Logs. This setup allows real-time collection and centralized monitoring of web traffic data for analysis and threat detection.

  Purpose:

  The primary purpose of directing WAF logs to CloudWatch Logs is to facilitate detailed traffic monitoring, enable proactive security analysis, and support automated alerting or response mechanisms based on observed traffic patterns. This enhances the ability to fine-tune WAF rules and improve overall attack mitigation effectiveness.

  a) **Logging Destination:**

  AWS WAF logging was configured to send sampled web request logs to the Amazon CloudWatch Logs group named aws-waf-logs-ddos-webacl.

  b) **Log Configuration:**

  This setup enables continuous capture of detailed request data, including source IP, HTTP method, URI, headers, and WAF rule evaluations, allowing granular inspection of web traffic patterns.

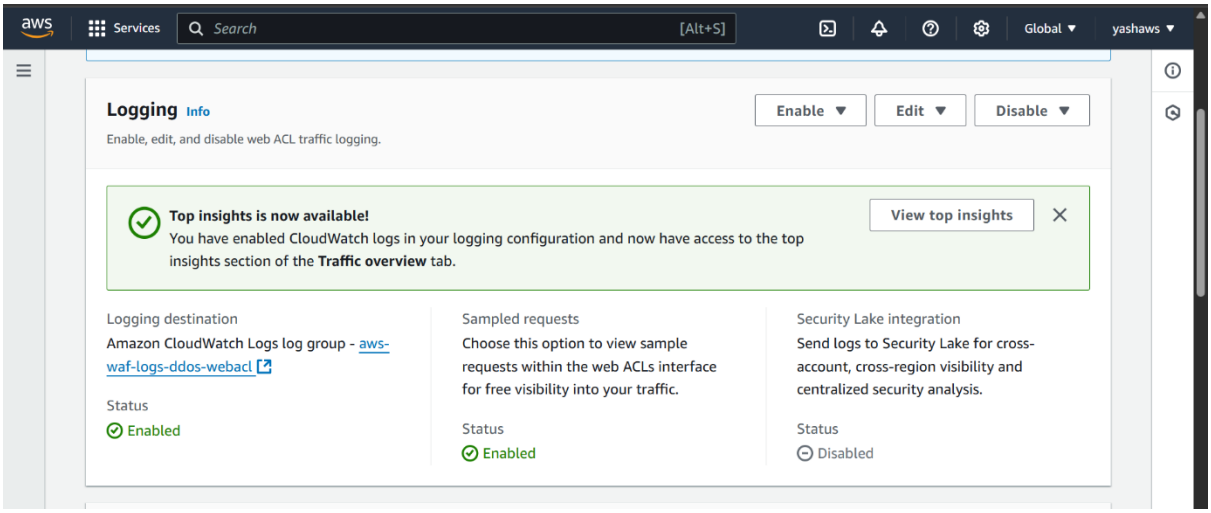
  c) **Sampling Rate:**

  A configurable sampling rate was applied to balance log volume and detail, ensuring meaningful data is collected without overwhelming storage or analysis capabilities.

  d) **Benefits:**

  The integration with CloudWatch Logs supports real-time monitoring, custom metric filters, and automated alerting to detect and respond promptly to suspicious traffic, improving the overall security posture of the application.

(WAF logging configured to stream sampled requests to CloudWatch Logs group aws-waf-logs-ddos-webacl.)



3.4 Monitoring and Dashboard Setup using CloudWatch

In this phase, Amazon CloudWatch was utilized to monitor application traffic patterns and key metrics in real time. Custom dashboards were created to visualize parameters such as request count, latency, and error rates, enabling effective tracking of traffic behavior and early detection of anomalies. This setup enhances situational awareness and supports informed decision-making for ongoing DDoS mitigation efforts.

- **Monitor Key Metrics in CloudWatch**

  This step focuses on monitoring critical performance indicators of the Application Load Balancer (ALB) using Amazon CloudWatch. Key metrics such as RequestCount, TargetResponseTime, PeakLCUs, and ConsumedLCUs were selected to evaluate the impact of the simulated DDoS traffic. These metrics help assess resource utilization and service availability in real time, providing actionable insight for response tuning and capacity planning.

  Technical Details:

  Service Used: Amazon CloudWatch

  Monitored Resource: Application Load Balancer (yash-alb)

Metrics Selected:

RequestCount: Total number of HTTP requests received by the ALB during each monitoring interval.

TargetResponseTime: Measures the time taken by the target to respond to a request—an indicator of backend responsiveness.

PeakLCUs (Load Balancer Capacity Units): Reflects the maximum number of LCUs consumed in a period, indicating ALB stress level.

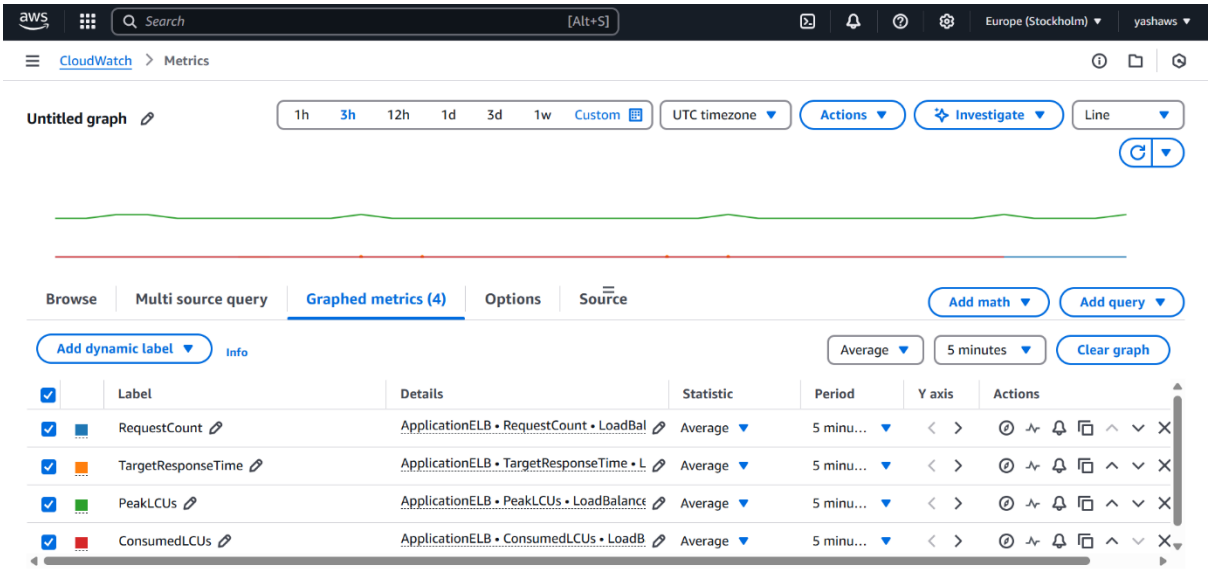
ConsumedLCUs: Represents the total ALB resource usage, calculated from new connections, active connections, and rule evaluations.

Statistic Applied: Average

Monitoring Interval: 5 minutes

Graph View: Line chart under "Graphed metrics" tab for trend analysis.

(Graph displaying real-time CloudWatch metrics for the ALB.)



- **Create Custom CloudWatch Dashboards**

  In this step, custom CloudWatch dashboards are created to visualize and correlate critical metrics related to the Application Load Balancer and AWS WAF in a centralized and intuitive manner. These dashboards enable real-time situational awareness and allow security teams to detect anomalies such as traffic spikes, response time delays, or LCU saturation. By consolidating key indicators like **RequestCount**, **TargetResponseTime**, and **ConsumedLCUs**, these dashboards provide a proactive monitoring interface to assess the system's resilience under DDoS attack conditions.

  Key Metrics Displayed:

  BlockedRequests and AllowedRequests: Show how many web requests are being blocked or allowed by the WAF, helping to understand how well security rules are working.

  RequestCountPerTargetGroup and CPUUtilization: Track the amount of traffic hitting each backend service and the CPU load on servers to monitor the impact on infrastructure.

  Pie Chart of Blocked IPs: Visualizes which IP addresses are being blocked most frequently, providing insight into traffic sources**.**

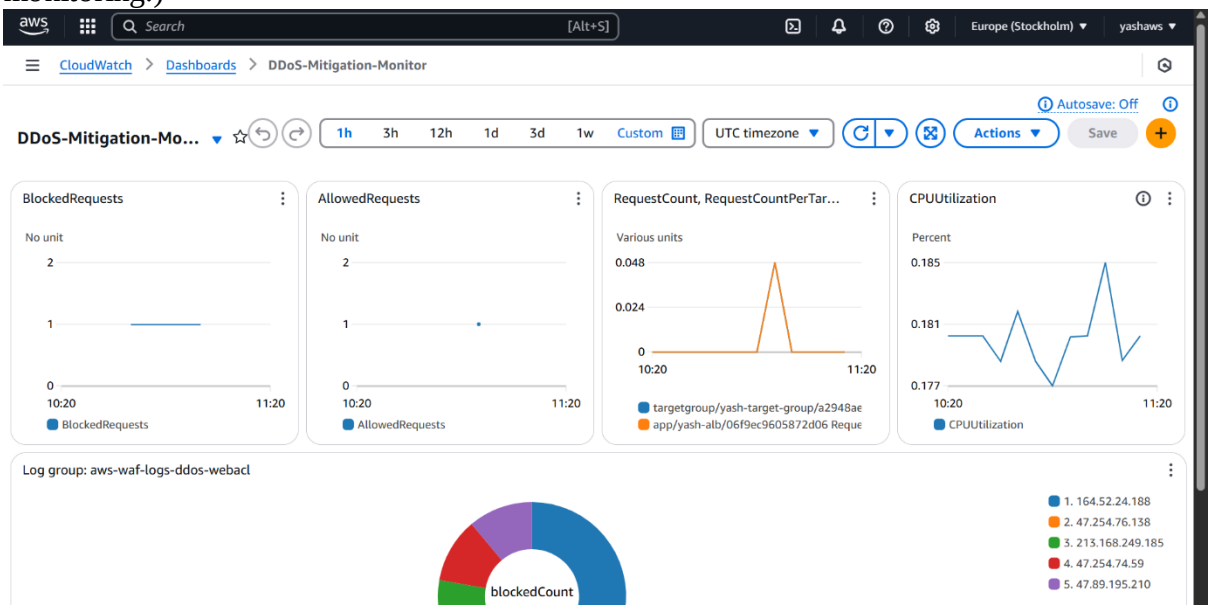
  Log Connection:

  The dashboard is connected to the AWS WAF logs (in the log group aws-waf-logs-ddos-webacl) so analysts can quickly investigate any unusual activity by reviewing the detailed logs.

  Benefits:

  This dashboard enables quick detection of unusual traffic spikes or delays, helping security and operations teams respond promptly to threats like DDoS attacks and maintain application performance.

(The dashboard shows WAF activity, traffic load, CPU use, and blocked IPs for quick monitoring.)



3.5 Automated Threat Response with AWS Lambda

In this phase, an automated threat response mechanism is implemented using AWS Lambda to enhance the security posture by enabling real-time, programmatic mitigation of detected threats. Leveraging Lambda's serverless compute capabilities, security events identified through monitoring and alerting systems trigger predefined response workflows without manual intervention. This automation accelerates incident containment, reduces operational overhead, and ensures consistent enforcement of security policies across the cloud environment. By integrating AWS Lambda with CloudWatch alarms, AWS WAF, and other security tools, this phase delivers a proactive and scalable approach to threat management.

AWS Lambda Function: AutoBlocksIPs

Function Name: AutoBlocksIPs

Runtime Environment: Python 3.12

Execution Role: A new IAM role created with the AWS-managed policy **AWSLambdaBasicExecutionRole** to grant the function necessary permissions for execution and logging.

Trigger: The function is configured to be invoked automatically by **CloudWatch Logs**, specifically from the AWS WAF log group (aws-waf-logs-ddos-webacl). This enables real-time detection and automated blocking of malicious IP addresses based on log events.

Below is the Lambda function code used to automatically block malicious IPs based on AWS WAF log events:

```
import json
import boto3
import base64
import gzip
import logging
import ipaddress
from botocore.exceptions import ClientError

waf = boto3.client('wafv2')
ip_set_id = '32ed6fb2-4827-412b-90b7-0844aae938fb'
ip_set_name = 'MaliciousIPsSet'
scope = 'REGIONAL'

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def is_public_ip(ip):
    """Return True if IP is public, False if private/reserved."""
    try:
        return ipaddress.ip_address(ip).is_global
    except ValueError:
        return False

def lambda_handler(event, context):
    try:
        cw_data = event['awslogs']['data']
```

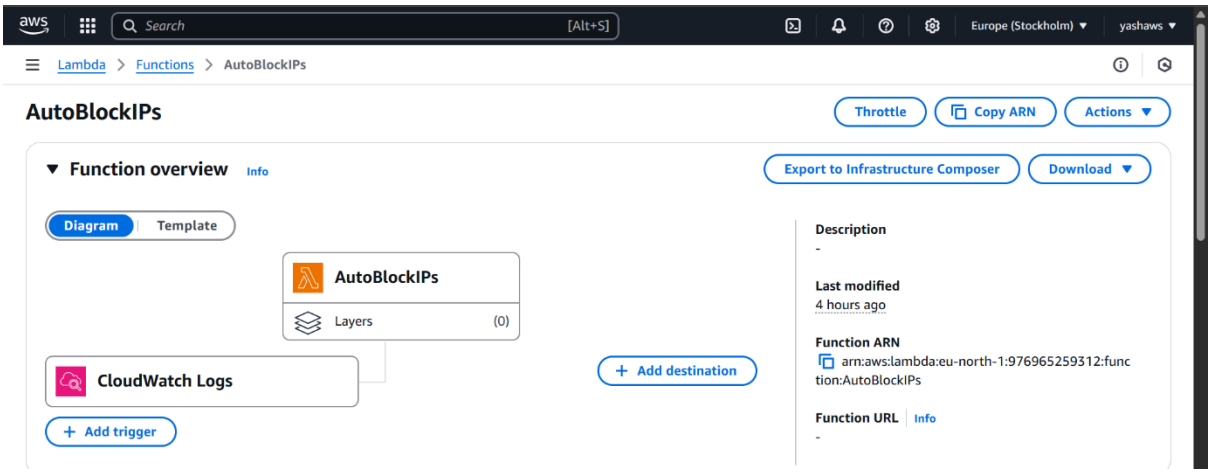
```
decoded_data = base64.b64decode(cw_data)
decompressed_data = gzip.decompress(decoded_data)
log_data = decompressed_data.decode('utf-8')
logs = json.loads(log_data)

malicious_ips = set()
for record in logs.get('logEvents', []):
    try:
        msg = json.loads(record.get('message', '{}'))
        if msg.get('action') == 'BLOCK':
            ip = msg.get('httpRequest', {}).get('clientIp')
            if ip and is_public_ip(ip):
                malicious_ips.add(ip + "/32")
    except Exception as e:
        logger.warning(f"Failed to parse log message: {e}")

if not malicious_ips:
    logger.info("No new malicious IPs found.")
    return {"status": "No new IPs", "count": 0}

ip_set = waf.get_ip_set(Name=ip_set_name, Scope=scope, Id=ip_set_id)
addresses = set(ip_set['IPSet'].get('Addresses', []))

new_ips = malicious_ips - addresses
if not new_ips:
    logger.info("No new IPs to add; all malicious IPs already blocked.")
    return {"status": "No update needed", "count": 0}

if len(addresses) + len(new_ips) > 10000:
    logger.warning("IP set address limit exceeded. Update skipped.")
    return {"status": "Limit exceeded", "count": 0}

updated_ips = addresses.union(new_ips)
```

```
            waf.update_ip_set(
                Name=ip_set_name,
                Scope=scope,
                Id=ip_set_id,
                LockToken=ip_set['LockToken'],
                Addresses=list(updated_ips)
            )

            logger.info(f"IP set updated with {len(new_ips)} new IP(s).")
            return {"status": "IPs updated", "count": len(new_ips)}

        except ClientError as e:
            logger.error(f"AWS ClientError: {e}")
            return {"status": "Error", "message": str(e)}

        except Exception as e:
            logger.error(f"Unexpected error: {e}")
            return {"status": "Error", "message": str(e)}
```

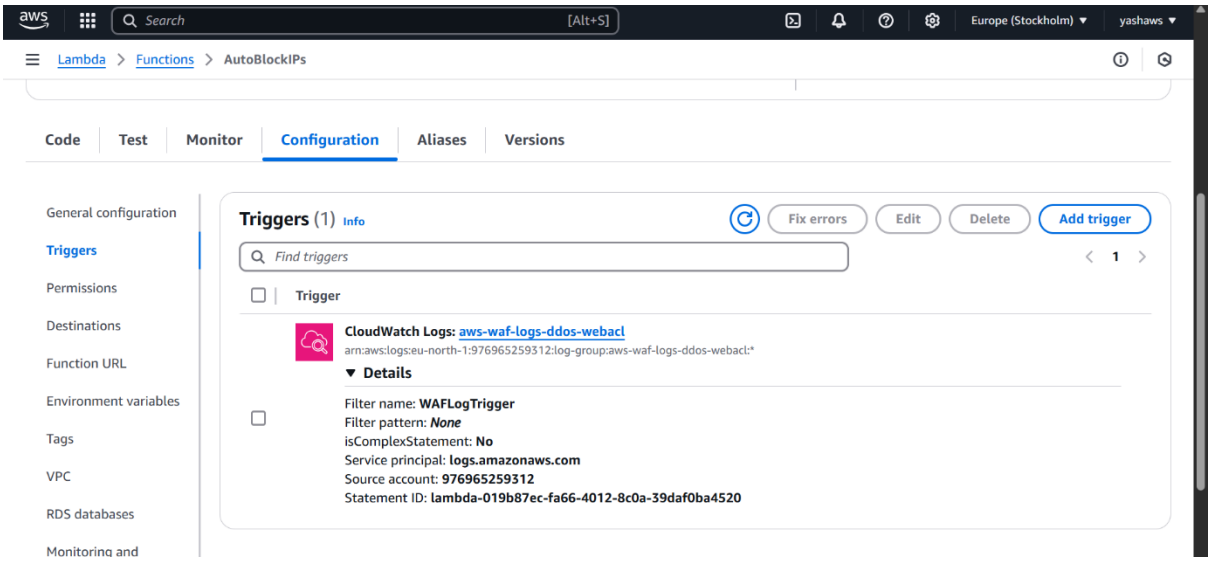
This Lambda function automates the process of updating an AWS WAF IP set by extracting blocked malicious IPs from CloudWatch Logs and dynamically adding them to the WAF block list to enhance security.

a) Initializes AWS WAFv2 client for IP set management.
b) Decodes and decompresses incoming CloudWatch Logs data.
c) Extracts log events and parses blocked IP addresses.
d) Filters IPs to include only public addresses.
e) Retrieves current IP addresses from the specified WAF IP set.
f) Identifies new malicious IPs not already blocked.
g) Ensures total IPs do not exceed the AWS WAF limit of 10,000.
h) Updates the IP set with new malicious IP addresses.
i) Logs key actions and handles exceptions for errors.

(Lambda function AutoBlocksIPs created with Python 3.12 runtime.)



(Trigger set to CloudWatch Logs group aws-waf-logs-ddos-webacl.)



IAM Policy for AutoBlocksIPs Lambda

CloudWatch Logs:

Allows the Lambda to create and write logs to its own log group.

Allows reading and filtering logs from the AWS WAF log group to detect threats.

AWS WAF:

Allows the Lambda to get and update IP sets to block malicious IP addresses automatically.

Scope:

Logging permissions are limited to the Lambda's log group and the WAF logs in the specified AWS region and account.

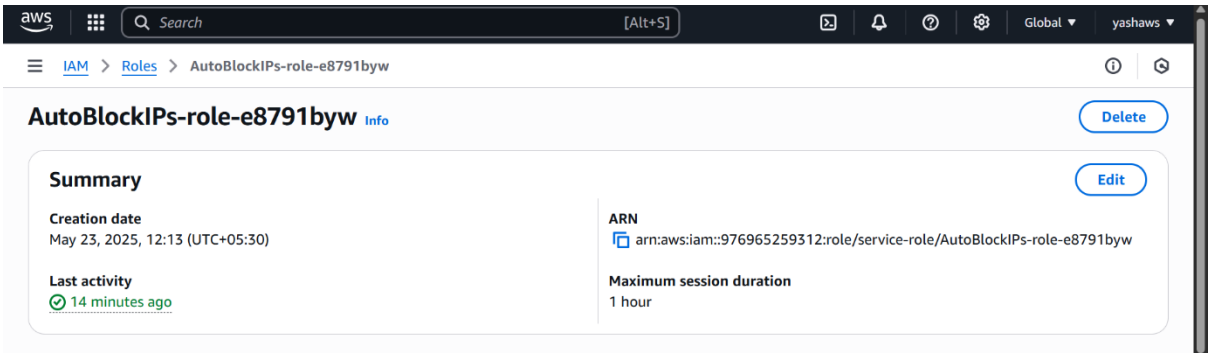
WAF permissions apply to all IP sets to enable updating them as needed.

Policy:

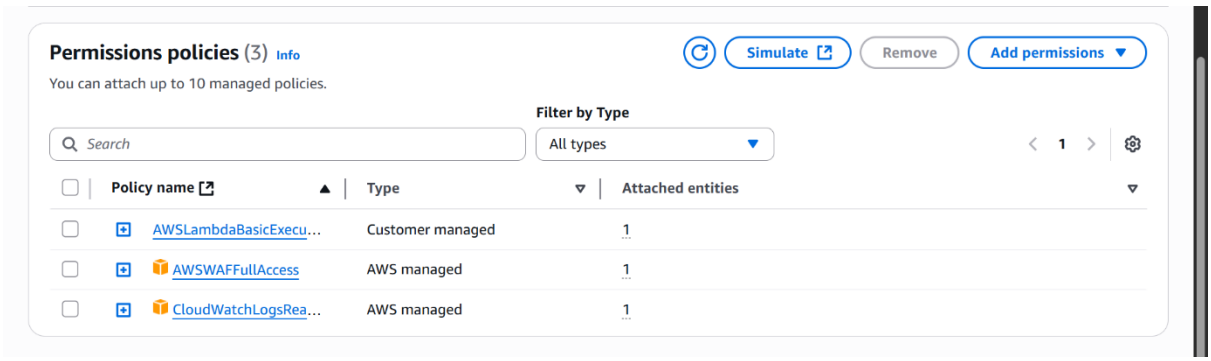

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": "arn:aws:logs:eu-north-1:976965259312:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:eu-north-1:976965259312:log-group:/aws/lambda/AutoBlockIPs:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents",
        "logs:DescribeLogStreams"
      ],
```

```
            "Resource": [
                "arn:aws:logs:eu-north-1:976965259312:log-group:aws-waf-logs-ddos-
webacl:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "wafv2:GetIPSet",
                "wafv2:UpdateIPSet"
            ],
            "Resource": "*"
        }
    ]
}
```

(IAM policy created to allow log access and WAF IP set updates.)



(Execution role assigned with basic Lambda permissions and the custom policy.)



3.6 Post-Mitigation Analysis and Tuning

This phase entails a comprehensive evaluation of mitigation strategies through detailed analysis of security metrics, dashboards, and logs. It focuses on optimizing protective measures by adjusting rate limits and updating IP blocklists to ensure effective threat prevention while maintaining application performance and reliability. Continuous monitoring and fine-tuning during this phase are critical for sustaining a robust security posture.

- **Analyze metrics, dashboards, logs, and malicious IPs.**

  This step involves monitoring and analyzing security metrics, dashboards, and logs to gain insights into the effectiveness of current mitigation measures. The purpose is to identify traffic patterns, detect potential threats, and verify that no legitimate traffic is being blocked, ensuring the security controls are functioning as intended.

  a) Monitor CloudWatch metrics to track the volume, frequency, and trends of blocked IP addresses, identifying anomalous spikes indicative of potential threats.

  b) Analyze custom dashboards to correlate key performance metrics such as RequestCount, TargetResponseTime, and ConsumedLCUs with mitigation actions, ensuring system stability.

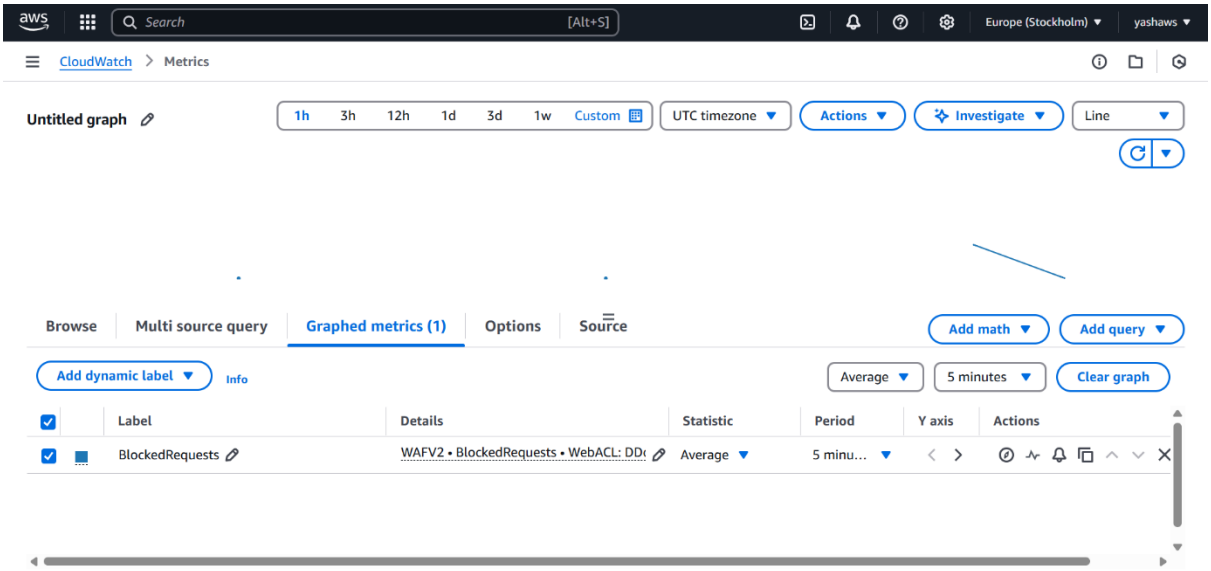
  c) Conduct thorough log reviews from AWS WAF and application sources to validate the accuracy of blocking events and confirm the exclusion of legitimate traffic.

  d) Identify patterns of recurring malicious IPs and assess their impact on network performance and security posture.

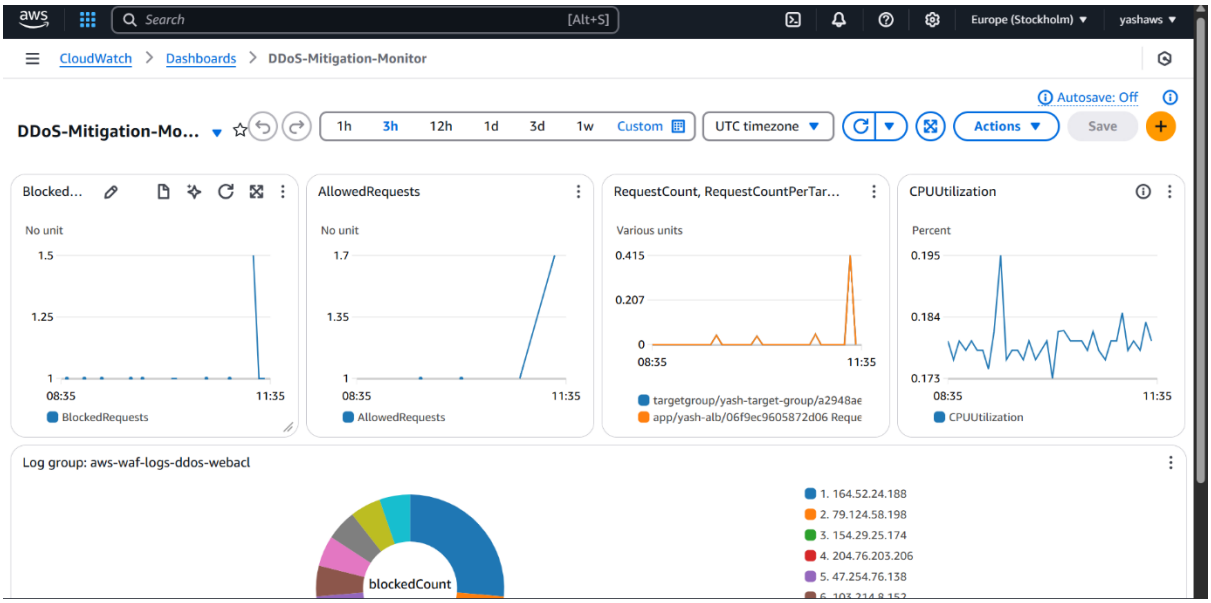
  e) Detect and investigate potential false positives or misconfigurations contributing to unnecessary blocks.

  f) Leverage insights derived from metrics and logs to optimize rate limits, refine IP blocklists, and enhance overall mitigation effectiveness.

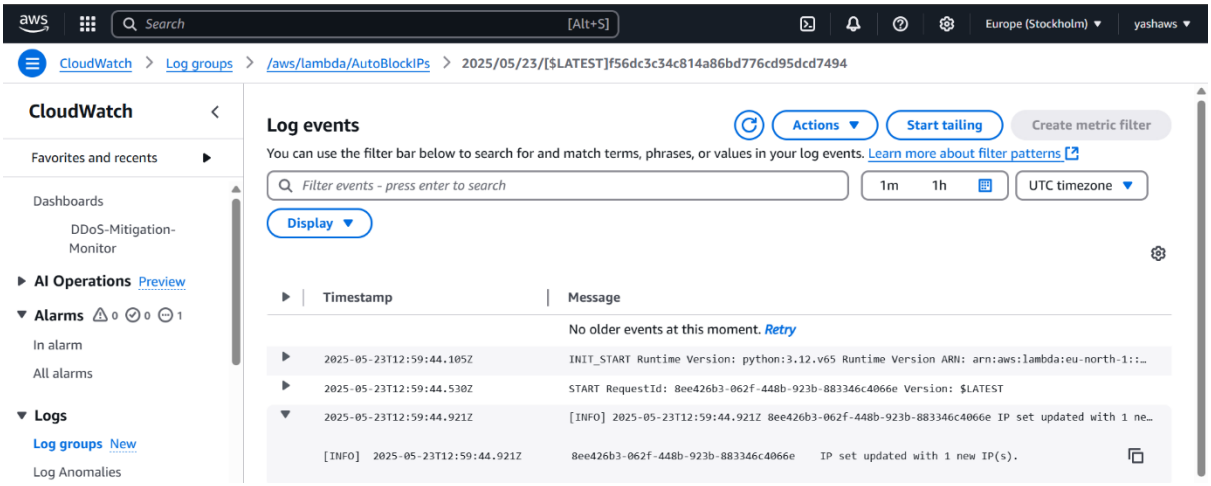
(Reviewed CloudWatch metrics to monitor the count and trends of blocked IP addresses.)



(Examined the custom dashboard to correlate key security and performance indicators for comprehensive analysis.)



(Inspected Lambda logs to verify the automatic blocking of malicious IPs detected in real-time.)



- **Tune rate limits and update IP blocks accordingly.**

  Focuses on interpreting the visualized data from dashboards to gain a comprehensive understanding of system performance and security events. The purpose is to correlate key metrics and identify anomalies or trends, enabling informed decisions to optimize defense mechanisms and maintain operational resilience.

  a) Configure the AWS WAF rate-based rule to limit requests to 2000 per 5 minutes per IP, based on observed traffic patterns.

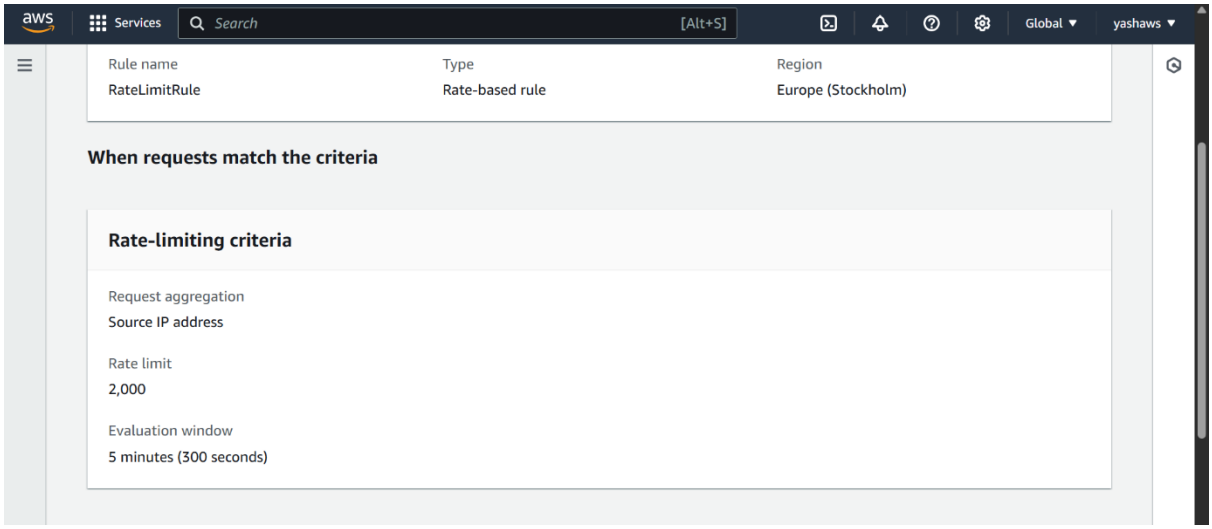
  b) Apply the updated rate limit rule to the Web ACL protecting critical resources.

  c) Review the malicious IP set to confirm effective blocking of identified threats.

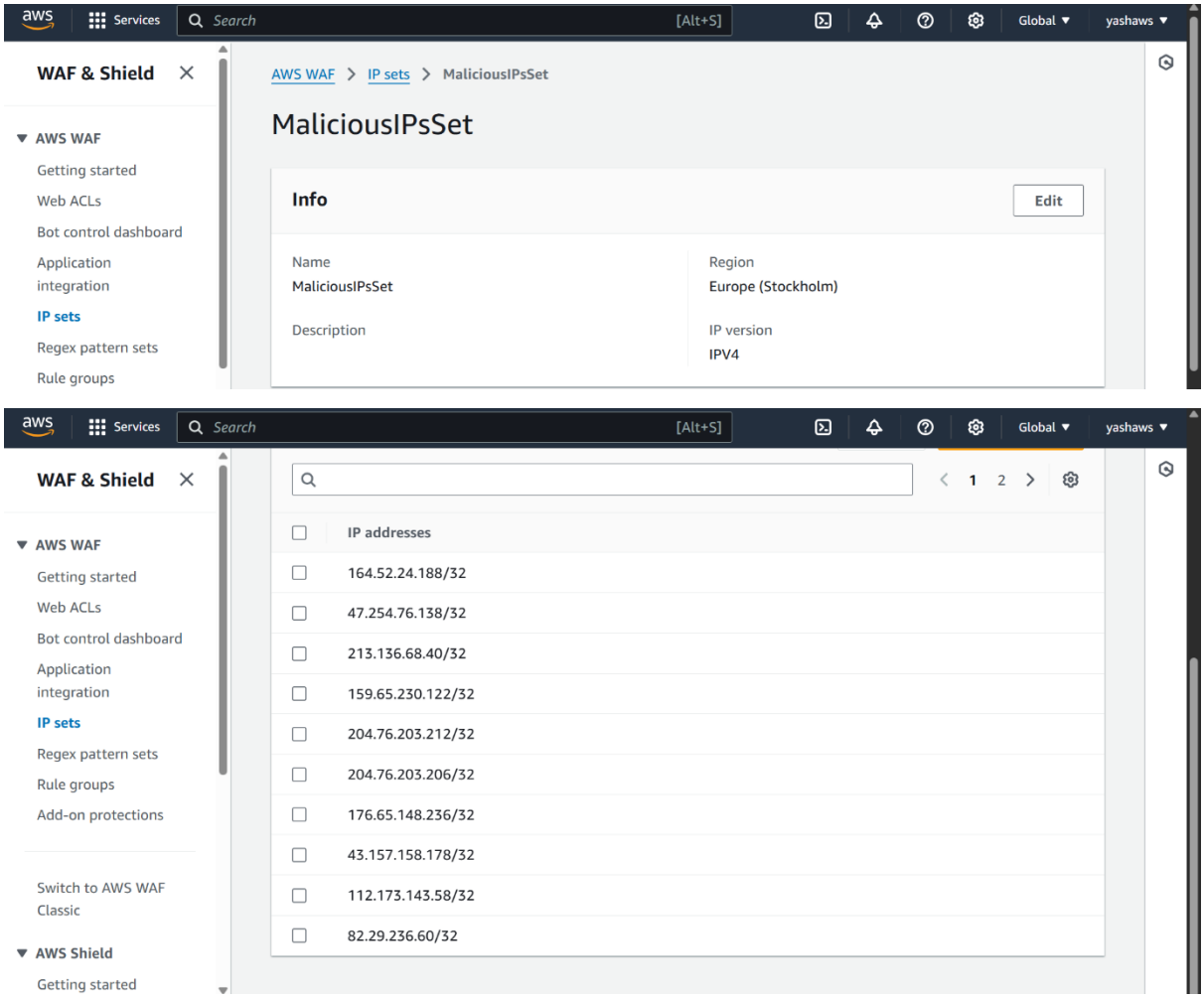
  d) Monitor logs and metrics to verify enforcement and minimize impact on legitimate traffic.

  e) Fine-tune the rate limit threshold as needed according to ongoing traffic analysis.

(Configured the AWS WAF rate-based rule to enforce a request limit of 2000 per IP within a 5-minute window.)



(Validated the updated list of malicious blocked IPs to ensure effective threat mitigation.)



6. Results And Findings

This section details the outcomes observed after implementing the DDoS mitigation measures using AWS WAF, Shield, and Lambda automation. It highlights the effectiveness, operational insights, and any limitations found during the simulation and monitoring phases.

- **Effective Traffic Filtering**
  The AWS WAF rate-based rules and IP blocking mechanisms successfully filtered out a large portion of malicious traffic generated during the simulated DDoS attack, significantly reducing unwanted requests reaching the application server.

- **CloudWatch Metrics Corroboration**
  CloudWatch dashboards and metrics clearly showed a sharp decline in requests from IPs identified as malicious, confirming the functionality of the WAF and Lambda-triggered updates to the IP sets.

- **Automated IP Blocking Efficiency**
  The Lambda function, triggered by CloudWatch Logs, accurately identified IP addresses associated with blocked requests and updated the WAF IP set dynamically, enabling real-time threat response without manual intervention.

- **Low False Positive Rate**
  Through careful tuning of rate limits and rule conditions, there was minimal impact on legitimate users, which suggests the mitigation strategy was well-calibrated to avoid blocking normal traffic.

- **System Performance Stability**
  Despite the high volume of attack traffic, the infrastructure maintained stable performance metrics, including low latency and uninterrupted availability, demonstrating the resilience of the deployed mitigation strategy.

7. Recommendations

Based on the results and lessons learned from the simulation, this section provides actionable advice to enhance and maintain the effectiveness of DDoS mitigation in the AWS environment.

- **Ongoing Monitoring and Dynamic Tuning**
  Continuously monitor CloudWatch metrics and adjust WAF rate limits and IP blocking rules to respond to evolving traffic behaviors and attack vectors.

- **Broaden Defensive Ruleset**
  Incorporate additional WAF managed rule groups and custom rules targeting known bot traffic, HTTP anomalies, and geo-restrictions to bolster security layers.

- **Refine Lambda Automation and IP Management**
  Regularly review and optimize Lambda functions for efficient processing, ensure the IP set size does not exceed AWS limits, and implement clean-up mechanisms to remove outdated IPs.

- **Utilize AWS Shield Advanced (If Applicable)**
  For environments with high risk or sensitive applications, adopting Shield Advanced can provide enhanced detection, mitigation, and 24/7 DDoS response support from AWS.

- **Periodic Attack Simulations**
  Conduct routine penetration testing and simulated DDoS attacks to validate existing protections and identify new vulnerabilities before they can be exploited.

8. Conclusion

The deployment of AWS WAF and Shield Advanced in conjunction with custom Lambda functions proved to be a robust solution for mitigating volumetric DDoS attacks on cloud-hosted web applications. By implementing rate-based rules, IP blocking, and bot mitigation, the system effectively filtered out malicious traffic while maintaining service availability for legitimate

users. This layered defense strategy is crucial in the cloud environment where scalability and flexibility must be balanced with security.

Continuous monitoring through CloudWatch and custom dashboards provided real-time insights into traffic patterns and attack behaviors. This visibility was instrumental in understanding the attack dynamics and allowed for rapid tuning of WAF rules to improve mitigation effectiveness. The metrics and logs offered detailed data, helping identify false positives and ensuring that legitimate traffic was not unintentionally blocked, thus maintaining user experience.

The automation of threat response through Lambda functions, which dynamically updated the WAF IP sets based on suspicious IP addresses detected in logs, significantly enhanced the operational efficiency of the mitigation strategy. This automation reduced manual workload and shortened the response time to emerging threats, which is vital in the fast-paced nature of DDoS attacks. It also enabled a proactive security posture, where the system could adapt and respond without human intervention.

Moreover, the simulation highlighted the importance of iterative analysis and tuning of security controls. Attackers continually evolve their tactics, making it imperative that mitigation strategies are regularly reviewed and adjusted based on new data. The ability to analyze blocked IPs and rate limits helped optimize the balance between security and accessibility, preventing unnecessary blocking while ensuring robust protection.

In conclusion, the integration of AWS native security services with custom automation and monitoring delivers a comprehensive approach to DDoS defense. Organizations can leverage such a framework to safeguard critical applications, minimize downtime, and protect their reputation. Continued investment in monitoring, automation, and adaptive rule tuning will be essential to stay ahead of sophisticated attack techniques and maintain resilience in an increasingly hostile cyber environment.