CCS6344 T2510 Assignment 2 Submission

Group Name: Group 26

SUNTERRESAA SANKAR	**1211102415**
AHMAD HYKAL HAKIMI BIN YUSRY	**1221305344**
PRAVIN KUNASEGRAN	**1221303877**

Chapter 1: Introduction

The shift towards cloud-native architecture has been motivated by the growing need to have scalable and safe web applications. This report describes how a PHP + MySQL sample application can be migrated out of a local XAMPP environment to AWS, with the intention of improving security, maintainability and performance.

The initial application using PHP and MySQL did not have production capabilities in terms of secure networking, access control and monitoring. In order to fill such gaps, the architecture was redesigned and built on AWS services, such as EC2 (web server), RDS (database), IAM (control plane), ALB and WAF (security solutions), CloudTrail (audit trail), and CloudWatch (event logging).

The migration follows Infrastructure as Code (IaC) guidelines and takes advantage of AWS-native tooling to enhance security, scalability, observability, and maintain CRUD capabilities.

Chapter 2: Traditional Application Security Risk Assessment

The original To-Do application hosted on XAMPP introduced several critical security vulnerabilities:

- **Unencrypted communication**: HTTP-based data transfer left user data exposed to interception.
- **Exposed services**: Default MySQL and admin panels were publicly accessible, increasing attack surface.
- **Lack of patching**: Manual updates meant outdated software with exploitable vulnerabilities.
- **Weak access control**: Admin credentials stored in plaintext with no granular permission control.
- **No logging or monitoring**: The absence of real-time visibility made it impossible to detect attacks.
- **No firewall protection**: XAMPP provided no native protection against port scanning or DDoS attempts.

These risks justified the migration to AWS, where security mechanisms could be enforced through cloud-native services.

Chapter 3: AWS Architecture Design

The new architecture was designed on AWS with clear separation of layers and security controls. The application runs on an EC2 instance (Apache + PHP) within a public subnet, while the RDS (MySQL) instance resides in a private subnet. A VPC, security groups, and IAM policies enforce network segmentation and least privilege access.

An Application Load Balancer (ALB) distributes traffic and integrates with AWS WAF for web-layer protection. CloudWatch monitors performance, while CloudTrail logs management events

to S3 for auditing. All infrastructure components were defined using CloudFormation to ensure repeatability through Infrastructure as Code.

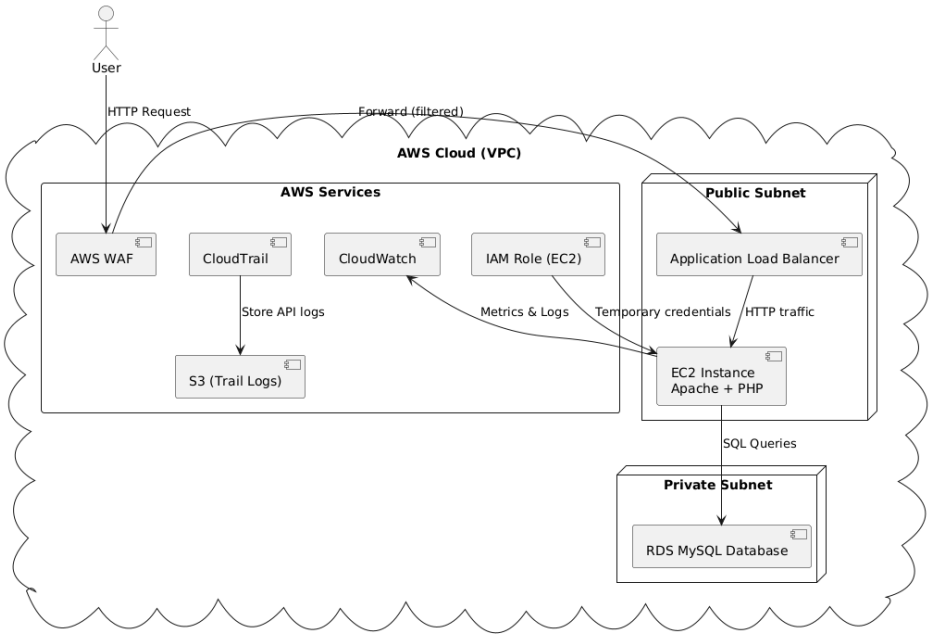


Figure 3.1: AWS Cloud Architecture of the Migrated PHP To-Do Application

The diagram represents the cloud-native design of the application using EC2, RDS, ALB, security groups, IAM roles, and application health routing. This infrastructure was defined using AWS CloudFormation and adheres to secure design principles.

Chapter 4: Security Migration Strategy

The migration strategy focused on addressing the core vulnerabilities of the original XAMPP-based system by leveraging AWS services to build a secure and scalable architecture. Each identified risk was mitigated through specific cloud-native features.

To reduce **network exposure**, the application was restructured with the EC2 instance hosted in a public subnet and the RDS database placed in a private subnet. Strict security groups were applied to limit access HTTP and SSH only to EC2, and MySQL access restricted to EC2's security group.

Encryption at rest was enabled on RDS using AWS KMS to protect sensitive data. IAM roles were attached to the EC2 instance to avoid hardcoded credentials and to enforce the Principle of Least Privilege (PoLP).

For **monitoring and auditing**, CloudWatch was configured to track CPU usage with alarm thresholds, while CloudTrail recorded API activities and stored logs in an S3 bucket for auditing.

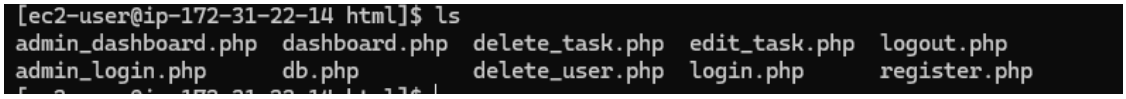
To guard against **web-based attacks**, the system was placed behind an Application Load Balancer integrated with AWS WAF. Managed rule sets for SQL injection, XSS, and PHP-specific threats were enabled. Simulated attack inputs confirmed that malicious payloads were effectively blocked with 403 status responses.

These improvements transformed the application into a more secure, reliable, and observable system, addressing the limitations of the original deployment while aligning with AWS security best practices.

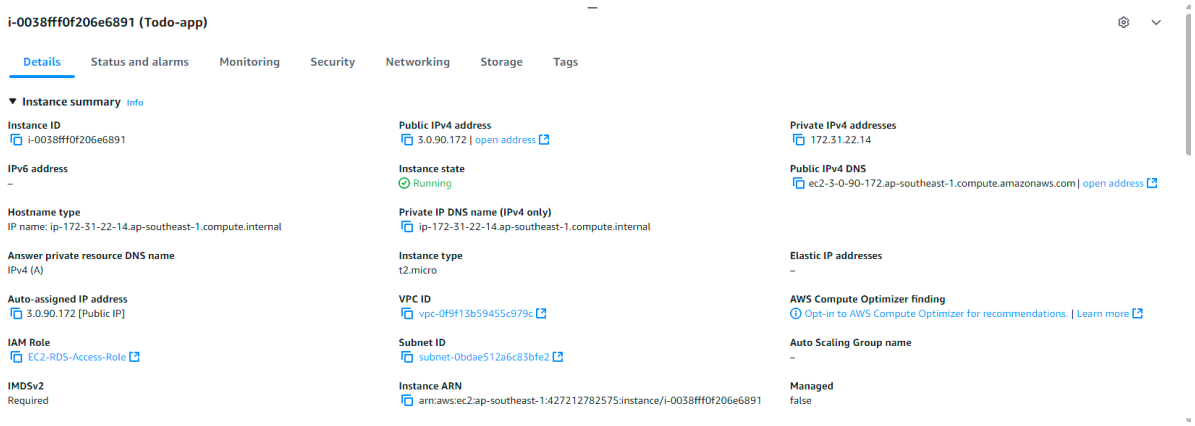
Chapter 5: Implementation

This chapter presents the technical implementation of the secure migration of the legacy PHP-based To-Do application from a local XAMPP environment to a fully cloud-native AWS architecture. The migration followed a phased approach, starting with infrastructure provisioning, application deployment, database migration, and concluding with the integration of security and monitoring services.

The implementation began with the deployment of a virtual server using **Amazon EC2**. An Amazon Linux 2 AMI was selected for compatibility and cost-efficiency. Apache HTTP Server, PHP, and relevant MySQL extensions were installed using the yum package manager. The application source code, originally hosted on a local XAMPP stack, was packaged and securely transferred to the EC2 instance using scp. After transferring the packaged source code from the local development environment to the EC2 instance using `scp`, the files were extracted into the `/var/www/html` directory. The Apache web server was configured to serve content from this location. The file listing confirms that all application modules, including login, registration, task management, and admin control, were successfully migrated to the cloud-hosted environment.

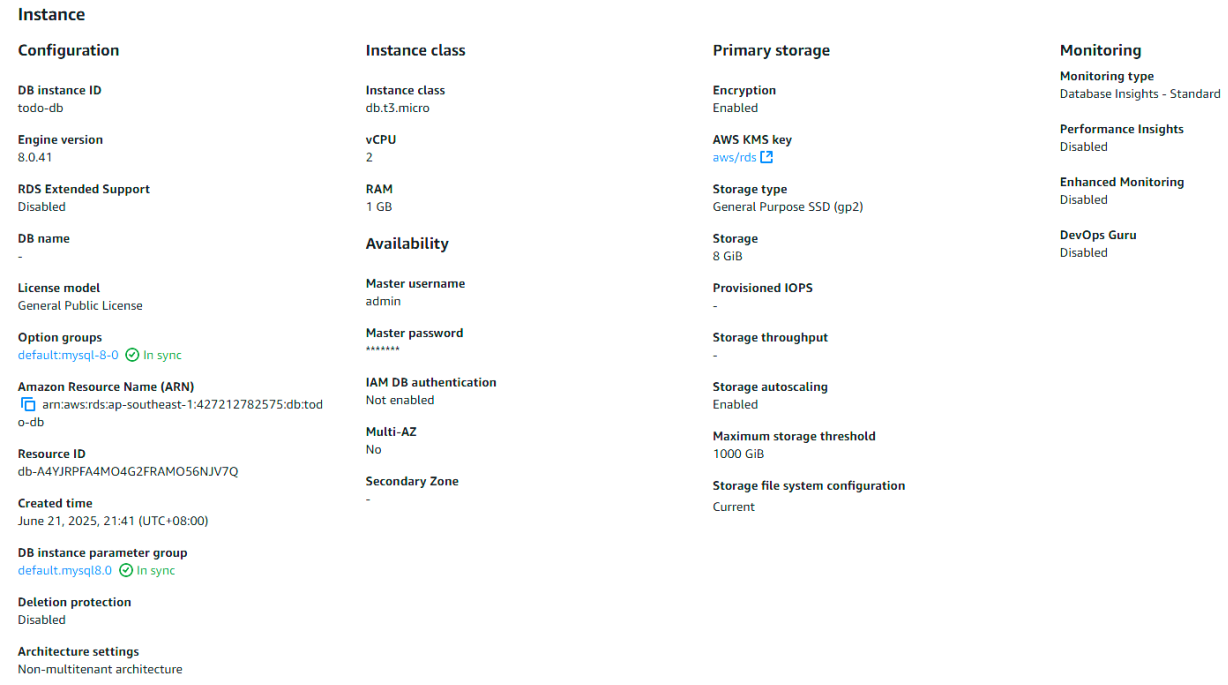


Screenshot 5.1a: PHP application files visible in EC2 `/var/www/html` after migration from local system



Screenshot 5.1: EC2 instance dashboard showing public IPv4, AMI ID, and instance type

The relational database component was migrated to **Amazon RDS** using MySQL 8.0. A new RDS instance was provisioned in a private subnet, with security group rules allowing inbound traffic only from the EC2 instance. A MySQL client was installed on the EC2 instance to connect to RDS and import the application's existing schema and data using the database.sql dump. The PHP application's db.php configuration file was updated with the RDS endpoint and credentials. Encryption at rest was enabled using AWS Key Management Service (KMS).



Screenshot 5.2: RDS instance configuration showing engine, endpoint, and encryption status

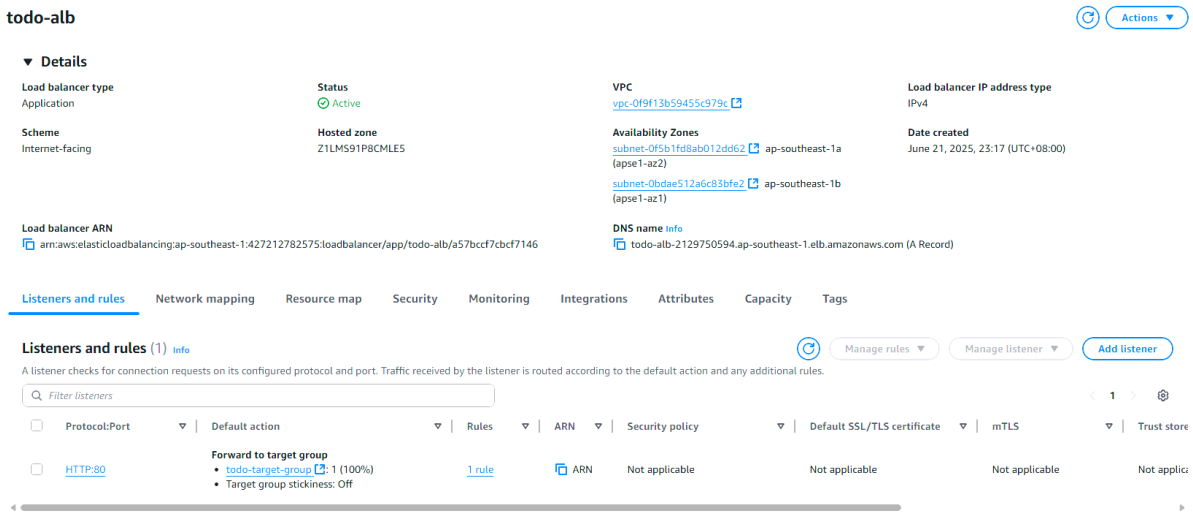
```php
<?php
$host = 'todo-db.c90koim8kxpr.ap-southeast-1.rds.amazonaws.com';
$user = 'admin';
$pass = 'your_secure_password123';
$dbname = 'todo_db';

$conn = new mysqli($host, $user, $pass, $dbname);
$conn->set_charset("utf8mb4");

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

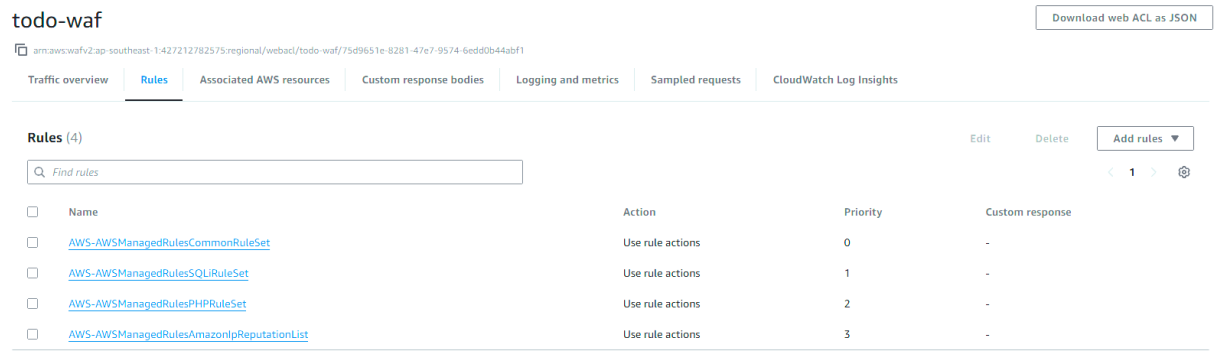
Screenshot 5.3: db.php file showing mysqli connection code to RDS

To ensure high availability and decouple direct traffic from the EC2 instance, an **Application Load Balancer (ALB)** was configured. The ALB listens on port 80 and forwards incoming traffic to a target group that includes the EC2 instance. A health check was configured on the login endpoint (/login.php) to ensure that only healthy instances receive traffic.

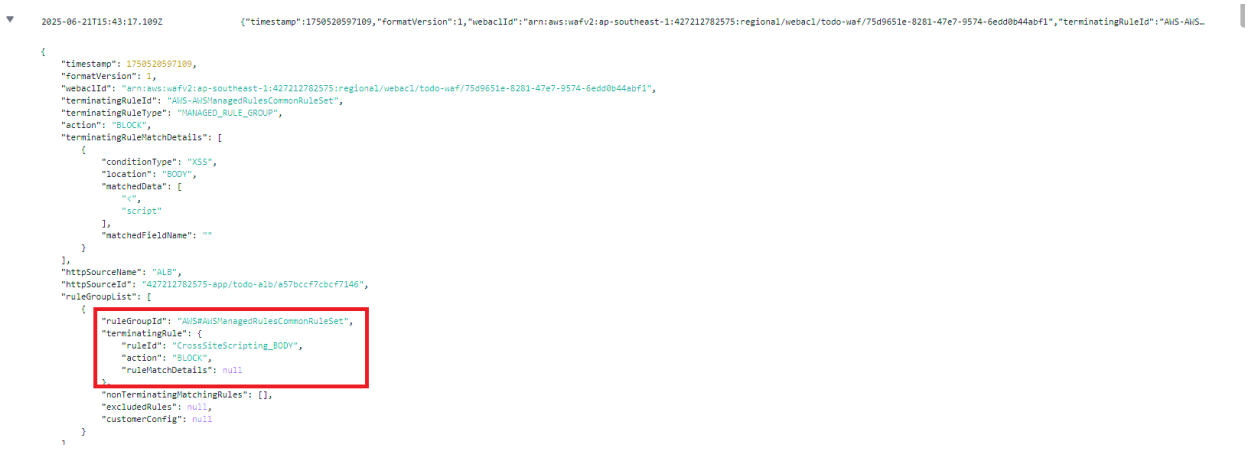


Screenshot 5.4: ALB configuration screen showing DNS name, listeners, and health check path

Application-layer protection was added using **AWS WAF**, which was associated with the ALB. AWS-managed rule sets including SQL injection, cross-site scripting (XSS), and PHP application threats were applied. Simulated attacks using malicious payloads (<script>alert(1)</script>, ' OR '1'='1) confirmed that WAF rules were active and effectively blocked such inputs, returning HTTP 403 status codes.

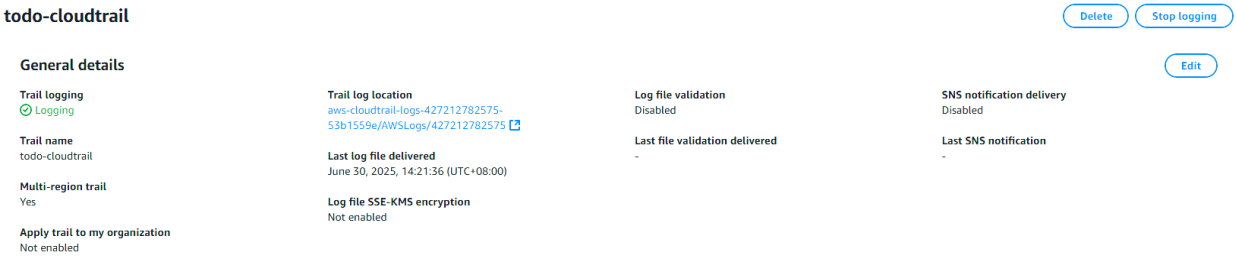


Screenshot 5.5: WAF ACL showing managed rules attached and logging enabled

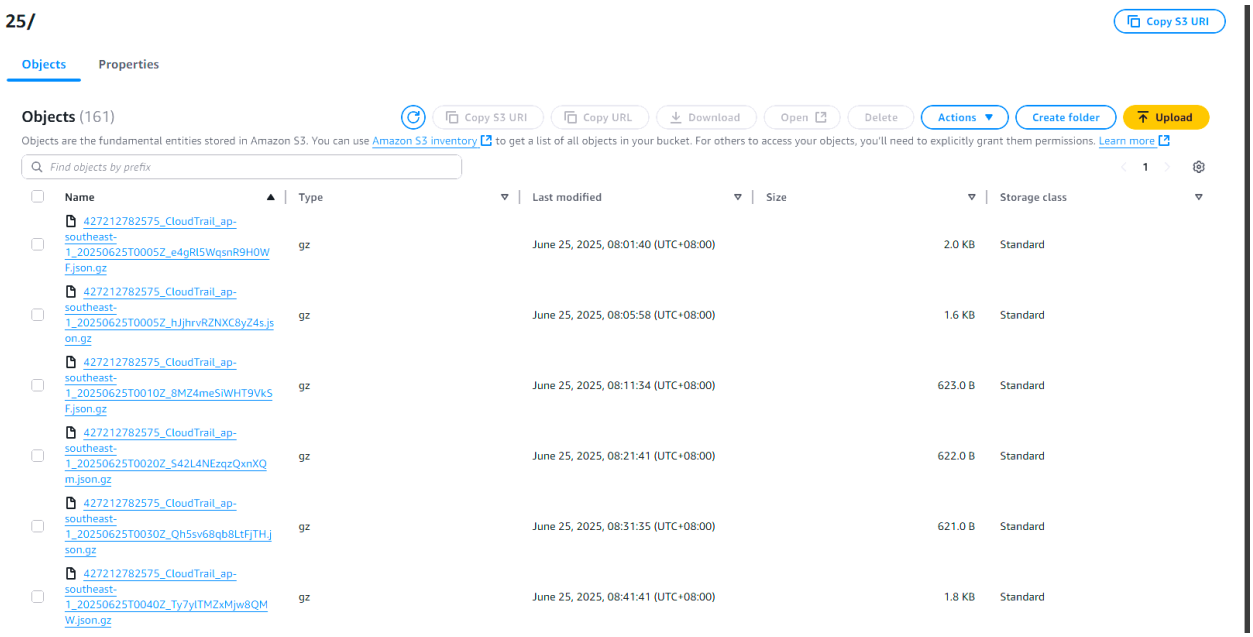


Screenshot 5.6: CloudWatch Logs or WAF logs showing a blocked request with 403 code

CloudTrail was enabled across all regions to record management events and API calls. A new S3 bucket was created as a log destination. Log files were verified to include EC2 launch events, IAM actions, and RDS access configuration changes. These logs serve both auditing and incident response purposes.

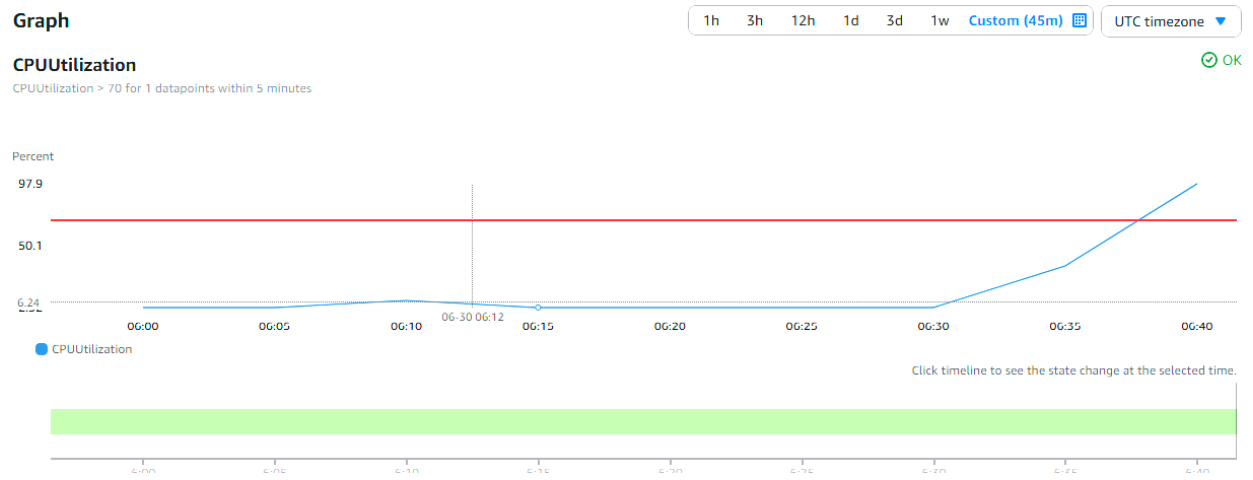


Screenshot 5.7: CloudTrail console showing trail name, S3 bucket, and region



Screenshot 5.8: S3 folder structure with log files

In addition, **CloudWatch** was configured to monitor EC2 CPU utilization. A threshold-based alarm was created to notify administrators if usage exceeded 70% for more than five minutes. The notification system was set up via Amazon SNS, and test alerts were generated to confirm functionality.

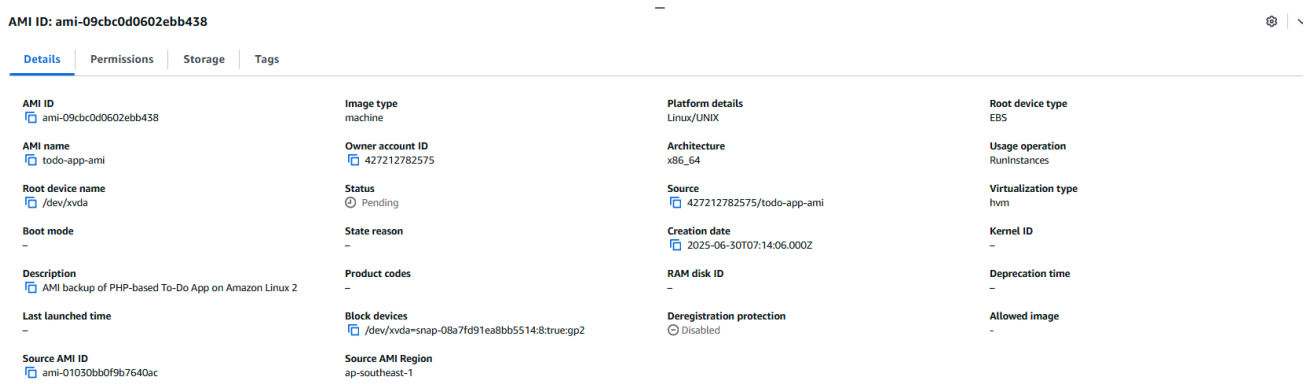


Screenshot 5.9: CloudWatch alarm configuration showing metric, threshold, and action

As part of the Infrastructure as Code (IaC) strategy, a **CloudFormation YAML template** was created to codify the entire environment. This template provisions EC2, RDS, ALB, security groups, IAM roles, and network configurations. It ensures that the infrastructure can be deployed consistently and efficiently. The visual system diagram generated via AWS Application Composer was based on this YAML file.

The template (todo-app`.yaml`) is available in the project's GitHub repository, complete with inline comments for maintainability and reuse.

To enable disaster recovery and fast redeployment, an **Amazon Machine Image (AMI)** of the EC2 instance was created. The AMI captures the application state, including all installed packages and uploaded code.



Screenshot 5.10: EC2 AMI created for backup and redeployment

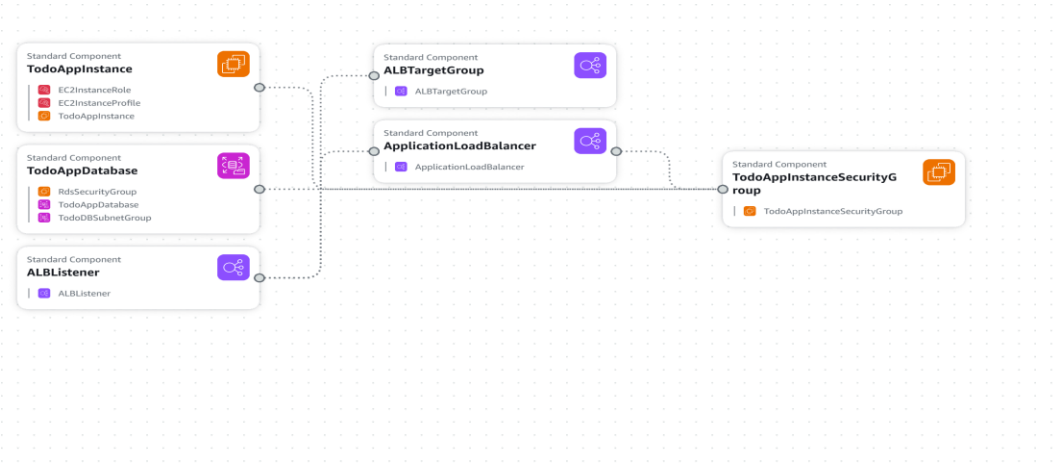


Diagram 5.1: AWS Architecture Diagram from AWS Application Composer

Through the implementation process, all components of the original system were securely migrated to AWS, with security, observability, and scalability greatly enhanced. The completed system supports full CRUD operations, WAF filtering, alerting, and logging in alignment with cloud security best practices.

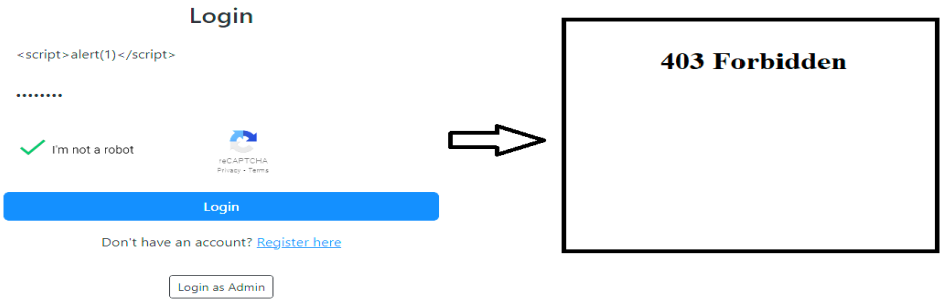
Chapter 6: Security Validation & Testing

To verify the effectiveness of the security controls applied during the cloud migration of the To-Do application, a series of validation tests were conducted. These tests focused on simulating common web-based attack vectors, evaluating system monitoring effectiveness, and auditing infrastructure changes. The goal was to ensure that the migrated system adheres to cloud security best practices and maintains availability, confidentiality, and integrity.

The primary layer of application-layer protection is provided by **AWS WAF**, which was integrated with the Application Load Balancer (ALB). To test its configuration, simulated **SQL injection** and **Cross-Site Scripting (XSS)** attacks were carried out through the login form hosted at:

http://todo-alb-2129750594.ap-southeast-1.elb.amazonaws.com/login.php

In the first test, a SQL injection payload (' OR 1=1 --) was entered in the username field. In the second test, an XSS payload (<script>alert(1)</script>) was submitted. In both cases, the application returned a **403 Forbidden** status code, confirming that the WAF successfully intercepted and blocked the requests based on its managed rule sets.

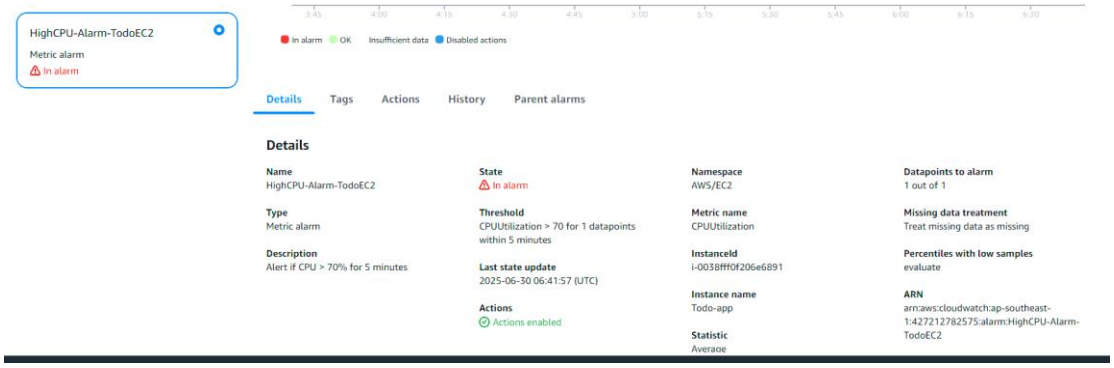


Screenshot 6.1: Browser view showing 403 error when submitting SQLi or XSS input

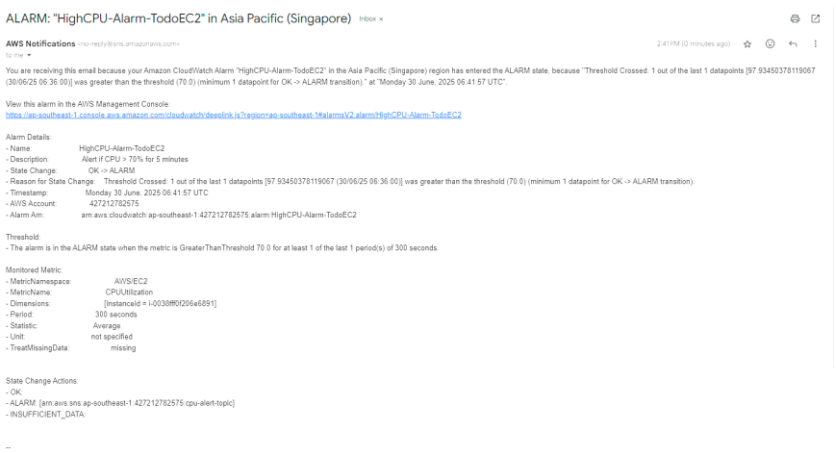
```
"action": "BLOCK",
"terminatingRuleMatchDetails": [
    {
        "conditionType": "XSS",
        "location": "BODY",
        "matchedData": [
            "<",
            "script"
        ],
        "matchedFieldName": ""
    }
],
"httpSourceName": "ALB",
"httpSourceId": "427212782575-app/todo-alb/a57bccf7cbcf7146",
"ruleGroupList": [
    {
        "ruleGroupId": "AWS#AWSManagedRulesCommonRuleSet",
        "terminatingRule": {
            "ruleId": "CrossSiteScripting_BODY",
            "action": "BLOCK",
            "ruleMatchDetails": null
        },
        "nonTerminatingMatchingRules": [],
        "excludedRules": null,
        "customerConfig": null
    }
```


Screenshot 6.2: AWS WAF Logs (via CloudWatch) showing the blocked request with terminatingRuleId

In addition to real-time attack prevention, the application was configured to be monitored by **Amazon CloudWatch**. An alarm was set to trigger when the EC2 instance CPU utilization exceeds 70% for 5 consecutive minutes. A load simulation was performed using repeated refreshes and test data inputs. Upon triggering the threshold, the alarm changed status to "ALARM," and an email notification was sent via Amazon SNS to the configured recipient.

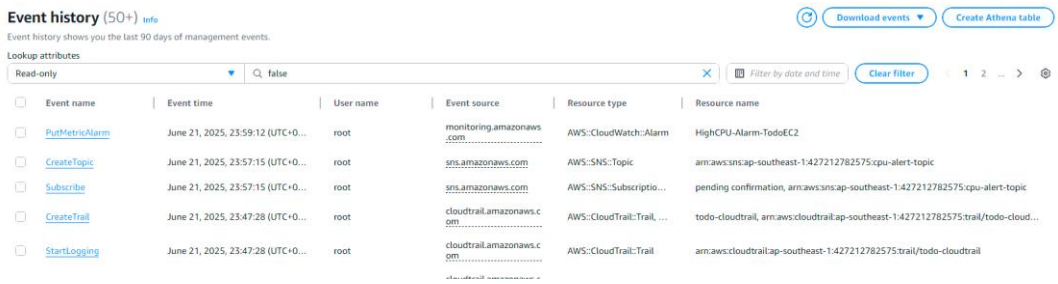


Screenshot 6.3: CloudWatch alarm screen showing state = ALARM with triggered metric

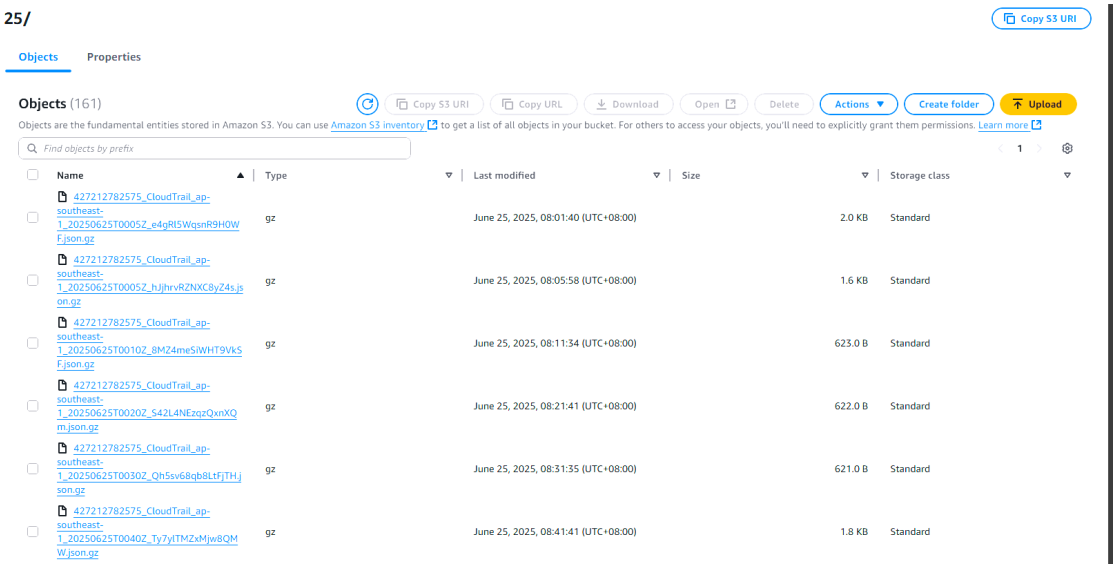


Screenshot 6.4: Email notification from SNS showing alert details

To track infrastructure-level changes and user actions, **AWS CloudTrail** was enabled and configured to send logs to an S3 bucket. Several test actions were performed, such as modifying the security group, restarting the EC2 instance, and viewing IAM policies. These events were successfully captured by CloudTrail and verified through log files stored in the specified S3 path.



Screenshot 6.5: CloudTrail console showing recent API activity timeline for EC2/RDS/IAM



Screenshot 6.6: S3 folder structure with date/time-based folders containing log files

Screenshot 6.7: Sample JSON log entry from CloudTrail showing user, action, timestamp

Furthermore, **nmap** was used externally to scan the public IP of the EC2 instance (via ALB) to confirm that only necessary ports (HTTP and SSH) were open. The results validated that port 80 was accessible and that port 22 access was restricted as expected.

```
C:\Users\User>nmap todo-alb-2129750594.ap-southeast-1.elb.amazonaws.com
Starting Nmap 7.97 ( https://nmap.org ) at 2025-06-30 14:53 +0800
Nmap scan report for todo-alb-2129750594.ap-southeast-1.elb.amazonaws.com (54.255.123.38)
Host is up (0.015s latency).
Other addresses for todo-alb-2129750594.ap-southeast-1.elb.amazonaws.com (not scanned): 3.0.32.15
rDNS record for 54.255.123.38: ec2-54-255-123-38.ap-southeast-1.compute.amazonaws.com
Not shown: 997 filtered tcp ports (no-response)
PORT     STATE  SERVICE
22/tcp   closed ssh
80/tcp   open   http
3306/tcp closed mysql

Nmap done: 1 IP address (1 host up) scanned in 7.27 seconds
```

Screenshot 6.8: nmap scan output showing open ports: 80 only (via ALB)

Collectively, these tests confirm that the migrated AWS infrastructure is protected against common vulnerabilities, actively monitored for abnormal behavior, and supports traceability and auditability. These controls align with the shared responsibility model and demonstrate adherence to AWS cloud security principles.

Chapter 7: Conclusion

The migration of the legacy PHP-based To-Do application to Amazon Web Services (AWS) successfully addressed the key security and scalability limitations of the original XAMPP-based deployment. Through a cloud-native architecture leveraging services such as EC2, RDS, ALB, WAF, CloudTrail, CloudWatch, and IAM, the application was reengineered to meet modern standards for availability, monitoring, and threat mitigation.

The use of Infrastructure as Code (IaC) through CloudFormation provides a reproducible, version-controlled method for provisioning resources. The integration of AWS WAF and managed rule sets ensured proactive protection against common web vulnerabilities including SQL injection and cross-site scripting (XSS). CloudTrail and CloudWatch further enhanced observability, enabling real-time alerting and comprehensive audit logging.

Security validation tests confirmed the efficacy of the implemented controls, with blocked malicious requests, triggered alarms, and traceable logs demonstrating the robustness of the deployed solution. The architecture adheres to AWS's shared responsibility model and incorporates best practices such as the Principle of Least Privilege, encryption at rest, and secure network segmentation.

In conclusion, the migration project not only met its objectives but also provided a practical framework for future migrations of monolithic applications to the cloud. It highlights the importance of secure-by-design thinking, infrastructure automation, and continuous monitoring in cloud-based deployments.

APPENDIX A – SOURCE CODE

GITHUB : https://github.com/Sunteresa7/Todo-app_v2