AUTO SCALING GROUP WITH APPLICATION LOAD BALANCER ON AWS

A MINIPROJECT REPORT

Submitted by

NARMADHA R (73772221169)

NIKITHA S (73772221171)

VIGNENH K (73772221207)

in partial fulfillment of the requirement

for the award of the degree

of

B.TECH

in

INFORMATION TECHNOLOGY

K.S. RANGASAMY COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

TIRUCHENGODE – 637 215

MAY 2025

K.S. RANGASAMY COLLEGE OF TECHNOLOGY
TIRUCHENGODE - 637 215

BONAFIDE CERTIFICATE

Certified that this project report titled **"AUTO SCALING GROUP WITH APPLICATION LOAD BALANCER ON AWS"** is the bonafide work of **NARMADHA R(73772221169)**, **NIKITHA S(73772221171)**, **VIGNESH K (73772221207)** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
S. B. THAMARAI SELVI
SUBJECT HANDLER
Assistant Professor
Department of Computer Science and Engineering
K.S. Rangasamy College of Technology
Tiruchengode - 637 215

SIGNATURE
DR.S. MADHAVI M.E., PH.D.,
HEAD OF THE DEPARTMENT
Professor
Department of Computer Science and Engineering
K.S. Rangasamy College of Technology
Tiruchengode - 637 215

DECLARATION

We jointly declare that the project report on **"AUTO SCALING GROUP WITH APPLICATION LOAD BALANCER ON AWS** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of B.Tech. This project report is submitted on the partial fulfilment of the requirement of the award of Degree of B.Tech.

Signature

_____

NARMADHA R

_____

NIKITHA S

_____

VIGNESH K

Place :Tiruchengode

Date :

ABSTRACT

In today's digital landscape, web applications must be designed for high availability, scalability, and fault tolerance to meet unpredictable user demands. This project focuses on deploying a web application on Amazon Web Services (AWS) using Elastic Compute Cloud (EC2) instances managed by an Auto Scaling Group (ASG) and distributed through an Application Load Balancer (ALB). The ALB ensures that incoming traffic is efficiently routed to healthy instances, while the ASG automatically adjusts the number of instances based on CPU usage to handle traffic spikes and reduce costs during low usage. The infrastructure spans multiple Availability Zones, providing resilience against zone failures. The solution is cost-effective, self-healing, and fully managed, making it ideal for production-grade applications that require reliability and performance without manual intervention. This mini project provides hands-on experience with AWS infrastructure services and demonstrates how to build a scalable and highly available architecture.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Modern web applications are expected to handle varying levels of traffic while maintaining high availability, performance, and fault tolerance. Traditionally, scaling an application to meet fluctuating demands required manual intervention or over-provisioning of resources, both of which are inefficient and costly. Cloud computing platforms like Amazon Web Services (AWS) provide automated solutions that address these challenges through scalable infrastructure and managed services.

This mini project explores the use of Amazon EC2 Auto Scaling Groups (ASG) in combination with an Application Load Balancer (ALB) to deploy a highly available and scalable web application. The Auto Scaling Group ensures that the number of running EC2 instances automatically adjusts based on the application's real-time resource usage, such as CPU utilization. Meanwhile, the ALB distributes incoming traffic across healthy instances, improving fault tolerance and performance.

By setting up this architecture across multiple Availability Zones, the project demonstrates how to build a resilient web infrastructure that can self-heal during instance or zone failures and scale seamlessly based on user demand. This project provides practical experience with key AWS services and emphasizes the importance of automation in cloud-based application deployment.

CHAPTER 2

TOOL USED - AWS SERVICES

This project utilized a variety of AWS services and supporting technologies to deploy a scalable and highly available web application. Below is a brief description of each tool and its role in the project:

1. Amazon EC2 (Elastic Compute Cloud)

Amazon EC2 provides resizable compute capacity in the cloud. It was used to host the web application instances. The instances were launched using a pre-configured Launch Template with a user data script that installed the Apache web server.

2. Auto Scaling Group (ASG)

The Auto Scaling Group automatically manages the number of EC2 instances running based on demand. It ensures that the application can scale out during high traffic and scale in during low traffic to optimize cost and performance.

3. Application Load Balancer (ALB)

The ALB distributes incoming HTTP traffic across multiple EC2 instances in different Availability Zones, improving fault tolerance and performance. It listens on port 80 and routes traffic to healthy targets registered in the target group.

4. Target Group

The target group is associated with the ALB and contains the EC2 instances that handle requests. It performs health checks on each instance to ensure traffic is only directed to healthy and active servers.

5. Amazon CloudWatch

CloudWatch monitors the health and performance of AWS resources. It was used to track CPU utilization and trigger scaling policies in the Auto Scaling Group.

6. AWS IAM (Identity and Access Management)

IAM was used to create roles and assign permissions securely. The EC2 instances and Auto Scaling needed IAM roles to interact with other AWS services such as CloudWatch.

7. Amazon Linux 2 (AMI)

This Amazon Machine Image was selected as the base operating system for EC2 instances. It is optimized for performance and compatibility with AWS infrastructure and supports the installation of common software packages like Apache.

8. Apache Web Server

Apache was installed via the EC2 instance’s user data script to serve a simple HTML web page. It acts as the front-end of the web application for demonstration purposes.

CHAPTER 3

IMPLEMENTATION STEPS

This section outlines the step-by-step process followed to deploy a web application using AWS services including EC2, Auto Scaling Group (ASG), and Application Load Balancer (ALB). The goal was to build a fault-tolerant and scalable architecture.

3.1 Launch Template Creation

A Launch Template defines the EC2 instance configuration used by the Auto Scaling Group.

AMI Used: Amazon Linux 2 (latest version)

Instance Type: t2.micro (chosen for its eligibility under the AWS Free Tier and sufficient for testing)

Key Pair: A new or existing key pair was selected for SSH access.

User Data Script: A script was added to automatically install Apache Web Server and create a sample web page on startup.

```
#!/bin/bash
yum update -y
yum install -y httpd
systemctl start httpd
systemctl enable httpd
echo "<h1>Welcome to Auto Scaling Web Server - $(hostname)</h1>" > /var/www/html/index.html
```

3.2 Create Target Group

A target group is necessary to register EC2 instances and perform health checks.

- **Target Type**: Instance
- **Protocol/Port**: HTTP/80
- **VPC**: Selected from existing ones
- **Health Check Path**: /
- **Health Check Protocol**: HTTP
- **Healthy Threshold**: 3

- **Unhealthy Threshold**: 2

This ensures only healthy instances receive traffic.

3.3 Configure Application Load Balancer

An ALB was configured to distribute traffic evenly among the EC2 instances registered in the target group.

- **Scheme**: Internet-facing (publicly accessible)
- **IP Type**: IPv4
- **Listener Configuration**:
  - Protocol: HTTP
  - Port: 80
  - Forward to: Target Group created earlier

- **Availability Zones**: Two or more zones selected for high availability

Security groups were adjusted to allow inbound HTTP (port 80) traffic from anywhere.

3.4 Create Auto Scaling Group

The Auto Scaling Group (ASG) was set up to use the launch template and automatically manage EC2 instances.

- **Launch Template**: Previously created
- **VPC/Subnet**: Selected multiple subnets in different Availability Zones
- **Desired Capacity**: 2
- **Minimum Capacity**: 2
- **Maximum Capacity**: 5
- **Load Balancing**: Linked to the ALB's target group

This configuration ensures the system maintains at least two instances running at all times and can scale up to five if needed.

3.5 Configure Scaling Policies

To automatically adjust capacity based on system load, dynamic scaling policies were added.

- **Metric Type**: Average CPU Utilization
- **Scale Out Policy**:
  - Trigger when CPU > 60% for 5 minutes

    - Add 1 instance
- **Scale In Policy**:
    - Trigger when CPU < 30% for 5 minutes
    - Remove 1 instance

These thresholds were tested using load generation (Apache Benchmark or stress tools) to simulate traffic.

3.6 Final Validation

- The setup was verified by accessing the ALB's DNS in a browser.
- The web page displayed the instance's hostname to differentiate each server.
- When CPU usage was increased, Auto Scaling triggered a new instance.
- The new instance was registered in the target group and started serving traffic within minutes.

CHAPTER 4

TESTING

This section outlines the various testing procedures performed to validate the configuration and functionality of the AWS Auto Scaling Group integrated with the Application Load Balancer. The objective was to ensure high availability, fault tolerance, and scalability of the deployed application.

4.1 Load Balancer Accessibility Test

After setting up the Application Load Balancer (ALB), its **DNS name** was used to test public accessibility.

- Opened the ALB DNS in a browser (e.g., http://<alb-dns-name>).
- Successfully loaded the web page hosted on EC2 instances.
- Each refresh returned responses from different instances (confirmed via hostname in the HTML).

This confirmed that:

- The ALB was distributing traffic evenly.
- EC2 instances were correctly registered and healthy.

4.2 Health Check Validation

The Target Group's health check settings were verified by:

- Temporarily stopping one EC2 instance.
- Observing that the instance was marked as **unhealthy** in the Target Group.
- Ensuring traffic was no longer routed to that instance.

Once the instance was restarted and passed health checks, it was automatically added back to the traffic pool.

4.3 Auto Scaling Trigger Test

To test scaling behavior:

- A **CPU stress tool** (such as stress or Apache Benchmark) was installed on the EC2 instance via SSH.
- Simulated high CPU load using:
  ```
  //bash
   sudo yum install stress -y
   stress --cpu 4 --timeout 300
  ```
- **CloudWatch** was monitored:

    - CPU usage crossed **60%**.
    - Auto Scaling launched a new EC2 instance.
    - The instance was registered in the Target Group and began serving traffic.
- After the load test ended:
    - CPU usage dropped below **30%**.
    - Auto Scaling **terminated** the extra instance.

This confirmed that dynamic scaling policies were functioning correctly.

4.4 High Availability Verification

To test fault tolerance:

- Manually terminated an active EC2 instance.
- Auto Scaling detected the failure and automatically launched a **replacement instance**.
- The new instance joined the Target Group and began handling requests within minutes.

This validated that the system was **highly available** and **self-healing**.

CHAPTER 5

RESULTS

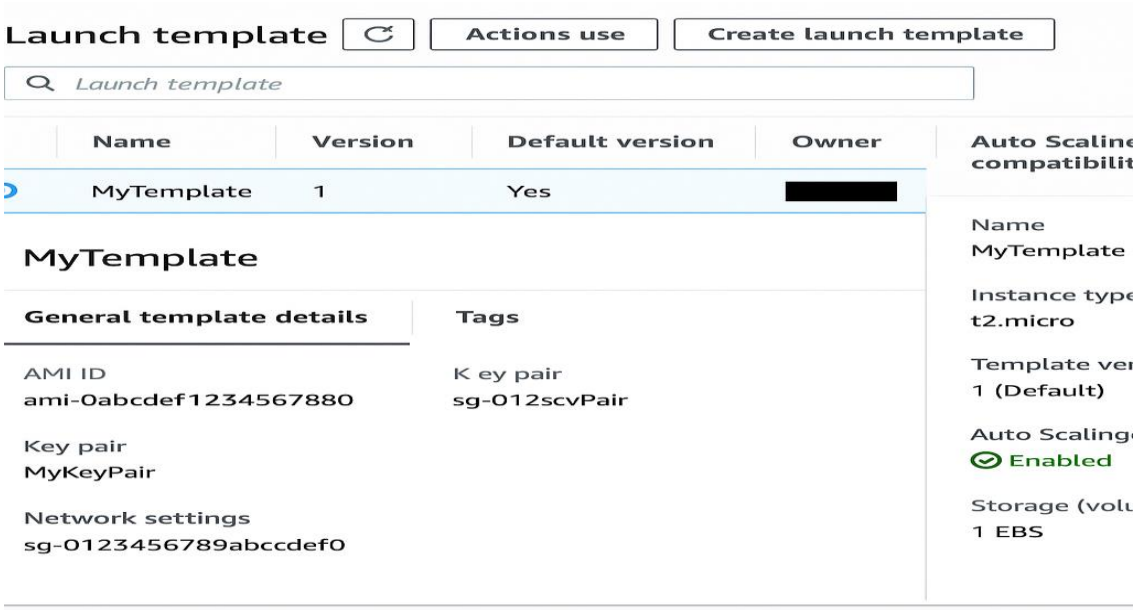

Figure 5.1 Launch Template

Figure 5.2 Target Groups

Figure 5.3 Auto Scaling Groups

CHAPTER 6

FUTURE SCOPE

While the current project demonstrates a solid foundation for deploying scalable and highly available web applications using AWS services, there are several enhancements and extensions that can be explored in future work to improve functionality, security, and automation.

1. Enable HTTPS for Secure Communication

- Use AWS Certificate Manager (ACM) to provision SSL/TLS certificates.
- Configure HTTPS listeners on the Load Balancer to ensure encrypted communication.

2.Implement CI/CD Pipelines

- Integrate services like AWS CodePipeline and CodeDeploy to automate application deployment.
- This ensures faster, consistent, and version-controlled updates to the EC2 instances.

3.Add a Backend Database

- Use Amazon RDS or DynamoDB to store persistent application data.
- This enables dynamic and data-driven web applications

4.Enhanced Monitoring and Alerts

- Set up advanced monitoring using Amazon CloudWatch dashboards and custom metrics.
- Configure alarms to notify administrators in case of unusual behavior or failures

5.Use Infrastructure as Code (IaC)

- Use AWS CloudFormation or Terraform to automate resource provisioning.

- This improves repeatability and makes the infrastructure more manageable

6.Global Load Balancing and Multi-Region Support

- Expand deployment to multiple AWS regions using Route 53 and Global Accelerator.
- Ensures better latency and disaster recovery readiness.

7. Security Enhancements

- Apply security best practices like private subnets, NAT gateways, and IAM role restrictions.
- Use AWS WAF and Shield to protect against web-based attacks.

CONCLUSION

This project successfully demonstrated the deployment of a highly available and scalable web application infrastructure using core AWS services such as EC2, Auto Scaling Group (ASG), and Application Load Balancer (ALB). By integrating these components, we were able to build a robust environment capable of automatically adjusting its capacity based on real-time demand while ensuring uninterrupted access to end users.

Through practical implementation and testing, the project highlighted how AWS Auto Scaling helps maintain application performance and reduce costs by dynamically managing compute resources. The Application Load Balancer played a crucial role in distributing traffic evenly and maintaining high availability across multiple Availability Zones.

In addition, the project provided hands-on experience with critical AWS tools including CloudWatch for monitoring, IAM for secure access management, and Amazon Linux 2 for server configuration. The successful execution of load tests and auto scaling events validated the effectiveness of the setup.

In conclusion, this mini project deepened the understanding of cloud-native architecture and reinforced the importance of automation, scalability, and fault tolerance in modern web application deployments.

REFERENCES

1. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/create-launch-template.html
2. https://docs.aws.amazon.com/elasticloadbalancing/latest/application/create-application-load-balancer.html
3. https://aws.amazon.com/free
4.