CAPSTONE PROJECT REPORT

AWS INFRASTRUCTURE DESIGN

Learn how to build and deploy scalable web applications in the AWS environment

LinkedIn
www.linkedin.com/in/gkmensah_

X (Formerly twitter)
@gkmensah_

CAPSTONE PROJECT REPORT

Designing and Maintaining a Scalable Web Application in AWS

TABLE OF CONTENTS

1. Introduction--- page 2

2. Project Outcomes-- page 2

3. Basic Terminologies--- page 2

4. AWS Architecture Design and Implementation-- page 3

- VPC and Subnet Configuration-- page 4

- Internet Gateway and NAT Gateway --- page 4

- Route Tables-- page 4

- Security Groups and NACLs -- page 4

- Launching EC2 Instances -- page 4

- Installing Web Servers -- page 4

- Creating AMI and Launch Templates --- page 4

- Load Balancer and Auto Scaling Group -- page 4

5. Monitoring and Optimization-- page 5

6. DynamoDB Integration --- page 5

7. IAM Role Management-- page 5

8. Challenges and Resolutions--- page 6

9. Areas for Improvement--- page 6

10. Conclusion-- page 7

1. INTRODUCTION

This report presents the first phase of a capstone project completed as part of an 18-week AWS Cloud Solutions Architect training program with Generation Ghana in partnership with Azubi Africa. The project focused on architecting, deploying, and securing a scalable and highly available web application infrastructure using Amazon Web Services (AWS).

2. PROJECT OUTCOMES

- Design and implement a secure and scalable web infrastructure in AWS.
- Gain hands-on experience with core AWS services.
- Understand the deployment of dynamic web applications in a production-like environment.
- Learn how to monitor, troubleshoot, and optimize AWS resources.
- Explore database integration and user access control.

3. BASIC TERMINOLOGIES

Key terms such as **VPC** (Virtual Private Cloud), **CIDR** (Classless Inter-Domain Routing), subnets, **NAT gateway** (Network Address Translation Gateway), **AMI** (Amazon Machine Image), **EC2** (Elastic Compute Cloud), and **Launch templates**, were explored and applied throughout the project. These form the backbone of cloud networking and compute services in AWS. Visit this AWS page to for an overview of the AWS services.

4. AWS ARCHITECTURE DESIGN AND IMPLEMENTATION

4.1 VPC AND SUBNET CONFIGURATION

- ✓ A VPC was created with a CIDR block of 10.0.0.0/16, which provides over 65,000 IP addresses, allowing room for subnetting and scalability.
- ✓ Four subnets (two public and two private) were created using CIDR blocks like 10.0.1.0/24, 10.0.2.0/24 to avoid IP overlap and segment the VPC.
- ✓ A pair of private and public subnet each were put in different Availability zones (AZs)

4.2 INTERNET GATEWAY AND NAT GATEWAY

- ✓ An Internet Gateway (IGW) was attached to the VPC to allow internet access for public subnets.
- ✓ A NAT Gateway was set up with an Elastic IP for private subnet instances to securely access the internet without being exposed.

NB: See recommended best practice for NAT gateway implementation in the areas for improvement section here.

4.3 ROUTE TABLES

- ✓ Public and private route tables were created to manage traffic flow to and from subnets.

4.4 SECURITY GROUPS AND NACLs

- ✓ Security Groups (SGs) were created to manage inbound and outbound rules per instance.
- ✓ Network ACLs (NACLs) were configured to manage subnet-level access control.

4.5 LAUNCHING EC2 INSTANCES

- ✓ EC2 instances were launched into subnets with configured SGs.
- ✓ SSH connections were established using PEM key pairs and CLI tools.

NB: *PEM key pair is a file that contains cryptographic key pair (public and private) often for SSH authentication, SSL/TLS encryption or secure access to cloud servers. In the context of AWS, it is the private key part of an SSH key pair (AWS keeps the public key pair). It is used to connect to your EC2 instances via SSH.*

4.6 INSTALLING WEB SERVERS

- ✓ Apache and NGINX were installed on the EC2 instances to host web applications.

4.7 CREATING AMI AND LAUNCH TEMPLATES

- ✓ AMIs were created as reusable blueprints of EC2 configurations.
- ✓ Launch Templates were used to standardize instance launches based on the AMI.

4.8 LOAD BALANCER AND AUTO SCALING GROUP

- ✓ An Application Load Balancer (ALB) distributed traffic to healthy instances.
- ✓ Target Groups and health checks ensured traffic was routed only to operational instances.
- ✓ Auto Scaling Group (ASG) adjusted instance count based on load.

5. MONITORING AND OPTIMIZATION

- ✓ Stress tests simulated CPU load to validate scaling policies.
- ✓ AWS CloudWatch monitored metrics like CPUUtilization.
- ✓ Alarms and SNS notifications were configured to alert on performance thresholds.

6. DYNAMODB INTEGRATION

- ✓ DynamoDB was used for its scalability, flexibility, and seamless integration with EC2 via IAM roles.
- ✓ Unlike MySQL, DynamoDB is fully managed, serverless, and doesn't require database administration, making it ideal for lightweight, scalable apps as used in this project.

7. IAM ROLE MANAGEMENT

- ✓ IAM roles were created and attached to EC2 instances to securely access DynamoDB.
- ✓ This approach avoids the need to embed access keys in application code.

8. CHALLENGES AND RESOLUTIONS

- ✓ Region mismatch caused EC2-DynamoDB connection failure. Resolved by ensuring resources were in the same region.
- ✓ Misconfigured NACL and SG rules initially blocked traffic. Reviewed and corrected accordingly.
- ✓ Insufficient CPU stress prevented CloudWatch alarms from triggering. Adjusted test parameters.

9. AREAS FOR IMPROVEMENT

9.1 High Availability with Multiple NAT Gateways

- ✓ Use of a single NAT Gateway created a single point of failure. Deploying multiple NAT Gateways in different AZs can increase redundancy.

9.2 Optimizing Network ACL Configuration

- ✓ Define stricter, clearer NACL rules based on the principle of least privilege.

9.4 Database Placement and Access

- ✓ Ensure region consistency and adopt AWS Secrets Manager for secure database credential management.

9.5 Enhanced Monitoring

- ✓ Expand CloudWatch metrics beyond CPU to include disk I/O, memory, and logs.

9.6 Deployment Automation

- ✓ Future improvements should include using CloudFormation or Terraform for repeatable infrastructure provisioning.

10. CONCLUSION

- This project demonstrated the practical application of AWS services in building secure, reliable, and scalable infrastructure.
- It highlighted the importance of careful configuration, security awareness, and proactive monitoring in real-world deployments.
- The successful execution of the project demonstrates a solid understanding of cloud-based application deployment and data management with DynamoDB