CSCI 5409

# Advance Topics in Cloud Computing

Summer 2024

Term Project Report

Course Instructor: Professor Lu Yang

Author: Harsh Vaghasiya [B00986219]

Gitlab Repository Link: https://git.cs.dal.ca/courses/2024-summer/csci4145-5409/hvaghasiya/-/tree/main/TermAssignment

## Contents

Project Overview ..... 5
Introduction ..... 5
Objectives ..... 5
Target Users ..... 5
Performance Targets ..... 6
Key Features and Functionalities ..... 6
High-Level Architecture ..... 6
Expected Benefits ..... 7
Meeting the Menu Requirements ..... 8
Selected Services ..... 8
Compute: ..... 8
Storage: ..... 8
Network: ..... 8
General: ..... 9
Comparison of Alternatives: ..... 9
Compute Alternatives ..... 9
Storage Alternatives ..... 10
Network Alternatives ..... 10
General Alternatives ..... 10
Justification for Choices ..... 11
Deployment Model ..... 13
Chosen Deployment Model: Hybrid Deployment Model ..... 13
Overview ..... 13
Detailed Description ..... 13
Compute Resources ..... 13
Storage Resources ..... 14
Networking Resources ..... 14
Security Resources ..... 14
Reasoning for Choosing the Hybrid Deployment Model ..... 14
Components and Their Deployment ..... 15
Delivery Model ..... 17

Chosen Delivery Model: Software as a Service (SaaS) .... 17
Overview .... 17
Detailed Description .... 17
Ease of Access and Convenience .... 17
Centralized Management and Updates .... 17
Scalability and Flexibility .... 17
Cost Efficiency .... 18
Enhanced Collaboration .... 18
Integrated Security and Compliance .... 18
Components of the SaaS Model .... 18
Reasoning for Choosing the SaaS Delivery Model .... 19
System Architecture .... 21
Components and Their Roles .... 21
Architecture Diagram .... 23
Detailed Component Interaction .... 24
Data Flow .... 24
Benefits of the Architecture .... 25
Security Approach .... 27
Data Security .... 27
Data in Transit .... 27
Data at Rest .... 27
Network Security .... 27
Access Control .... 28
Application-Level Security .... 28
Additional Security Measures .... 28
Cost Analysis .... 29
Cloud Model Used on AWS .... 29
EC2 Instance .... 29
AWS S3 .... 29
AWS Lambda .... 29
AWS Secrets Manager .... 29
ChatGPT API .... 30

AWS Elastic Load Balancer ........ 30
AWS Elastic Container Registry (ECR) ........ 30
Summary of Monthly Cloud Costs ........ 30
On-Premise Cost ........ 31
EC2 Instance Equivalent ........ 31
Lambda Function and ChatGPT API Equivalent ........ 31
S3 Storage Cost Equivalent ........ 31
Summary of Monthly On-Premise Costs ........ 32
Comparison ........ 32
Monitoring and Cost Management ........ 33
Key Cloud Mechanisms for Monitoring ........ 33
Monitoring Tools and Techniques ........ 35
AWS CloudWatch ........ 35
AWS Cost Explorer ........ 35
AWS Budgets ........ 35
Future Development and Feature Enhancements ........ 36
References ........ 40

# Project Overview

## Introduction

The Automatic Portfolio Webpage Generation Chatbot is a sophisticated system designed to streamline the creation of personalized portfolio webpages by leveraging user-uploaded resumes. This project aims to utilize the robust capabilities of AWS services to automate the entire process, ensuring efficiency, scalability, and security. The chatbot interacts with the user, processes the provided resume data, and dynamically generates a professional portfolio webpage that is then deployed online for easy access.

## Objectives

The primary objectives of the project are:

1. **Automated Portfolio Creation:** Transform user-uploaded resumes into fully functional and aesthetically pleasing portfolio webpages.
2. **Leverage AWS Services:** Utilize AWS's extensive suite of cloud services to ensure the solution is scalable, reliable, and secure.
3. **User Experience:** Provide a seamless and intuitive user experience from the initial upload of the resume to the final deployment of the portfolio webpage.

## Target Users

The key target users for this system include:

- **Job Seekers:** Individuals looking to create professional and compelling portfolio webpages to showcase their skills and experiences, enhancing their job search process.
- **Professionals:** Users who want to present their professional journey, achievements, and skills in a polished online format for networking and personal branding.

## Performance Targets

To ensure the system meets user needs effectively, the following performance targets have been set:

- **Response Time:** The system should process resumes and generate portfolio webpages within a minimal time frame, providing feedback to the user promptly.
- **Scalability:** The solution should handle multiple simultaneous users efficiently, scaling resources as needed to maintain performance.
- **Reliability:** The system must be highly available, ensuring that users can access the service at any time without interruptions.
- **Security:** User data and generated content should be handled securely, with robust mechanisms in place to protect sensitive information.

## Key Features and Functionalities

The system encompasses several key features designed to deliver a comprehensive user experience:

1. **Resume Upload:** Users can easily upload their resumes through an intuitive web interface.
2. **Data Processing:** The uploaded resumes are processed to extract relevant information such as personal details, work experience, education, skills, and achievements.
3. **Portfolio Generation:** Using the extracted data, a professional portfolio webpage is automatically generated, following best practices in design and user experience.
4. **Webpage Deployment:** The generated portfolio webpage is deployed online, stored in AWS S3 for high availability and reliability.

## High-Level Architecture

The system leverages a combination of AWS services to achieve its goals:

- **Frontend:** A React application hosted on AWS EC2, providing an interactive interface for users to upload resumes and interact with the chatbot.
- **Backend:** A Python application hosted on AWS EC2, handling data processing and integration with the ChatGPT API.

- **Database:** MySQL database hosted on AWS EC2, storing user data and portfolio content.
- **Storage:** AWS S3 for storing generated portfolio webpages.
- **API Integration:** AWS Lambda functions for invoking the ChatGPT API to process resume data.
- **Networking:** AWS VPC for secure and isolated networking, and AWS Elastic Load Balancing for distributing traffic.
- **Secrets Management:** AWS Secrets Manager for securely storing API keys and other sensitive information.

## Expected Benefits

The implementation of this project is expected to bring several benefits:

- **Efficiency:** Automates the labor-intensive process of creating portfolio webpages, saving users time and effort.
- **Professionalism:** Ensures that users present their professional achievements in a polished and well-organized format.
- **Scalability:** The system can handle a growing number of users without compromising performance.
- **Security:** User data is protected through robust security measures, ensuring privacy and compliance with relevant regulations.

## Meeting the Menu Requirements

### Selected Services

#### Compute:

1. **AWS EC2 (Elastic Compute Cloud)**
   - **Purpose:** Hosting the React frontend application, Python backend application, and MySQL database.
   - **Justification:** EC2 provides scalable compute capacity, allowing for precise control over the environment. This flexibility is crucial for hosting the frontend and backend applications, as well as managing the MySQL database.
2. **AWS Lambda**
   - **Purpose:** Invoking the ChatGPT API to process resume data and generate portfolio content.
   - **Justification:** Lambda offers a serverless compute service that automatically scales with demand. It simplifies the management of backend processes and integrates seamlessly with other AWS services.

#### Storage:

3. **AWS S3 (Simple Storage Service)**
   - **Purpose:** Storing the generated portfolio webpages.
   - **Justification:** S3 provides highly durable and available object storage, making it ideal for storing static assets like web pages. Its scalability and cost-effectiveness are essential for handling potentially large volumes of portfolio data.

#### Network:

4. **AWS VPC (Virtual Private Cloud)**
   - **Purpose:** Ensuring secure and isolated networking for the application.
   - **Justification:** VPC allows for the creation of a logically isolated network within the AWS cloud, providing control over network configuration and enhancing security.

### General:

5. **AWS Elastic Load Balancing**
   - **Purpose:** Distributing traffic across EC2 instances to ensure high availability and reliability.
   - **Justification:** Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as EC2 instances, to optimize resource utilization and maintain application availability.
6. **AWS Secrets Manager**
   - **Purpose:** Securely storing API keys and sensitive information.
   - **Justification:** Secrets Manager provides a secure and convenient way to manage and retrieve sensitive information, such as API keys, passwords, and database credentials. It integrates well with other AWS services, enhancing security.

## Comparison of Alternatives:

### Compute Alternatives

1. **AWS EC2 vs. AWS Elastic Beanstalk**
   - **Elastic Beanstalk:** Simplifies the process of deploying and managing applications but offers less control over the environment.
   - **EC2:** Chosen for its flexibility and control, which are essential for customizing the environment to meet specific application requirements.
2. **AWS Lambda vs. AWS Step Functions**
   - **Step Functions:** Suitable for orchestrating complex workflows but adds complexity to the system.
   - **Lambda:** Chosen for its simplicity and efficiency in handling event-driven tasks, making it ideal for processing API requests and integrating with the ChatGPT API.

## Storage Alternatives

1. **AWS S3 vs. AWS DynamoDB**
   - **DynamoDB:** A NoSQL database service, more suitable for structured data with flexible schemas.
   - **S3:** Chosen for its cost-effectiveness and ability to store large volumes of unstructured data, such as static web pages.

## Network Alternatives

1. **AWS VPC vs. Amazon CloudFront**
   - **CloudFront:** A content delivery network (CDN) that accelerates the distribution of content globally.
   - **VPC:** Chosen for its ability to create a secure and isolated network environment, essential for internal application components.

## General Alternatives

1. **AWS Elastic Load Balancing vs. AWS API Gateway**
   - **API Gateway:** Ideal for routing and managing API requests, particularly for microservices architectures.
   - **Elastic Load Balancing:** Chosen for its ability to distribute traffic across multiple EC2 instances, ensuring high availability and performance.
2. **AWS Secrets Manager vs. AWS Key Management Service (KMS)**
   - **KMS:** Provides broader encryption services for managing cryptographic keys.
   - **Secrets Manager:** Chosen for its specialized capabilities in managing and retrieving sensitive application secrets, offering a simpler and more focused solution for handling API keys and credentials.

## Justification for Choices

1. **AWS EC2:**
    - **Control and Flexibility:** EC2 offers full control over the compute environment, allowing for customized configurations tailored to the specific needs of the frontend and backend applications.
    - **Scalability:** EC2 instances can be scaled up or down based on demand, ensuring the system remains responsive under varying loads.
2. **AWS Lambda:**
    - **Serverless Architecture:** Lambda functions scale automatically with the number of invocations, reducing the need for manual infrastructure management.
    - **Integration:** Lambda integrates seamlessly with other AWS services, such as S3 and Secrets Manager, simplifying the overall architecture.
3. **AWS S3:**
    - **Durability and Availability:** S3 offers 99.999999999% durability and 99.99% availability, ensuring that stored data is both safe and accessible.
    - **Cost-Effectiveness:** S3's pay-as-you-go pricing model makes it a cost-effective choice for storing large volumes of static content.
4. **AWS VPC:**
    - **Security:** VPC provides a secure environment for deploying application components, with fine-grained control over network configurations.
    - **Isolation:** VPC ensures that the application components are isolated from other AWS users, enhancing security and compliance.
5. **AWS Elastic Load Balancing:**
    - **High Availability:** ELB distributes traffic across multiple instances, ensuring that the application remains available even if one or more instances fail.
    - **Scalability:** ELB scales with incoming traffic, providing consistent performance under varying load conditions.

6. **AWS Secrets Manager:**
    - **Security:** Secrets Manager ensures that sensitive information is stored securely and accessed only by authorized components.
    - **Ease of Use:** It simplifies the process of managing secrets, reducing the risk of exposing sensitive data.

This detailed section on meeting the menu requirements explains the rationale behind the selection of each AWS service, compares alternatives, and justifies the choices made to optimize the performance, security, and cost-effectiveness of the Automatic Portfolio Webpage Generation Chatbot.

# Deployment Model

## Chosen Deployment Model: Hybrid Deployment Model

## Overview

The deployment model chosen for the Automatic Portfolio Webpage Generation Chatbot is a hybrid deployment model. This model combines the benefits of both traditional server-based computing and serverless architecture to ensure flexibility, scalability, and efficiency. The key components of this deployment model include AWS EC2 instances for hosting the frontend, backend, and database, and AWS Lambda functions for handling specific serverless tasks.

## Detailed Description

### Compute Resources

1. **AWS EC2 (Elastic Compute Cloud) Instances:**
    - **Frontend Hosting:** The React frontend application is hosted on an EC2 instance. This provides full control over the environment, allowing for custom configurations and optimizations tailored to the application's needs.
    - **Backend Hosting:** The Python backend application is also hosted on an EC2 instance. This ensures that the backend services can be finely tuned and scaled independently of the frontend.
    - **Database Hosting:** The MySQL database is deployed on an EC2 instance, providing robust data management capabilities and seamless integration with the backend services.
2. **AWS Lambda Functions:**
    - **API Invocation:** AWS Lambda is used to invoke the ChatGPT API for processing resume data and generating portfolio content. This serverless approach ensures that the system can handle varying loads efficiently, scaling automatically based on demand.

### Storage Resources

3. **AWS S3 (Simple Storage Service):**
   - **Webpage Storage:** Generated portfolio webpages are stored in AWS S3. S3 provides highly durable and scalable object storage, ensuring that the webpages are always available to users.

### Networking Resources

4. **AWS VPC (Virtual Private Cloud):**
   - **Network Isolation:** AWS VPC is used to create a secure and isolated network environment for the application components. This ensures that all communication between components is secure and controlled.
   - **Subnets and Security Groups:** VPC subnets and security groups are configured to control traffic flow, enhancing the security of the deployment.
5. **AWS Elastic Load Balancing:**
   - **Traffic Distribution:** Elastic Load Balancing is used to distribute incoming traffic across multiple EC2 instances. This ensures high availability and reliability by balancing the load and preventing any single instance from becoming a bottleneck.

### Security Resources

6. **AWS Secrets Manager:**
   - **Secrets Management:** AWS Secrets Manager is used to store and manage sensitive information such as API keys and database credentials. This service ensures that secrets are securely stored and accessed only by authorized components.

## Reasoning for Choosing the Hybrid Deployment Model

1. **Flexibility and Control:**
   - **Custom Configuration:** Hosting the frontend, backend, and database on EC2 instances provides full control over the environment. This allows for custom configurations and optimizations that are specific to the application's requirements.

  - **Independent Scaling:** The ability to scale the frontend and backend services independently ensures that the system can handle varying loads efficiently.

2. **Scalability and Efficiency:**
   - **Serverless Functions:** AWS Lambda functions provide a serverless approach to handle specific tasks, such as invoking the ChatGPT API. This ensures that these tasks can scale automatically based on demand without the need for manual intervention.
   - **Cost Efficiency:** The serverless nature of Lambda functions means that costs are incurred only when the functions are invoked, making it a cost-effective solution for handling variable workloads.
3. **Security and Isolation:**
   - **VPC Security:** AWS VPC provides a secure and isolated network environment, ensuring that all communication between application components is secure.
   - **Secrets Management:** AWS Secrets Manager ensures that sensitive information is securely stored and accessed only by authorized components, enhancing the overall security of the deployment.
4. **High Availability and Reliability:**
   - **Elastic Load Balancing:** By distributing incoming traffic across multiple EC2 instances, Elastic Load Balancing ensures high availability and reliability. This prevents any single instance from becoming a bottleneck and ensures that the system remains responsive under varying load conditions.
   - **Fault Tolerance:** The hybrid deployment model provides fault tolerance by distributing the workload across multiple instances and using serverless functions to handle specific tasks. This ensures that the system can continue to operate even if some components fail.

## Components and Their Deployment

1. **Frontend (React Application):**
   - Deployed on an EC2 instance.
   - Accessible via a public DNS or IP address.
   - Configured with security groups to allow HTTP/HTTPS traffic.
2. **Backend (Python Application):**
   - Deployed on an EC2 instance.

    - Communicates with the frontend via RESTful APIs.
    - Configured with security groups to allow traffic from the frontend and database.
3. **Database (MySQL):**
    - Deployed on an EC2 instance.
    - Configured to accept connections from the backend.
    - Security groups configured to restrict access to authorized components only.
4. **AWS Lambda Functions:**
    - Used for invoking the ChatGPT API.
    - Configured to trigger on specific events, such as resume upload or portfolio generation request.
    - Integrated with AWS S3 and other AWS services as needed.
5. **AWS S3 (Simple Storage Service):**
    - Stores generated portfolio webpages.
    - Configured with bucket policies to control access.
6. **AWS VPC (Virtual Private Cloud):**
    - Provides network isolation and security.
    - Configured with subnets, route tables, and security groups to control traffic flow.
7. **AWS Elastic Load Balancing:**
    - Distributes incoming traffic across multiple EC2 instances.
    - Ensures high availability and reliability.
8. **AWS Secrets Manager:**
    - Stores and manages sensitive information securely.
    - Ensures that only authorized components can access the stored secrets.

This detailed description of the deployment model explains how the various AWS services and components are integrated to provide a flexible, scalable, and secure solution for the Automatic Portfolio Webpage Generation Chatbot. The hybrid deployment model leverages the strengths of both server-based and serverless architectures to achieve the project's goals.

# Delivery Model

## Chosen Delivery Model: Software as a Service (SaaS)

## Overview

The chosen delivery model for the Automatic Portfolio Webpage Generation Chatbot is Software as a Service (SaaS). In this model, the application and its components are hosted and managed by a third-party cloud service provider, AWS, and delivered to end-users over the internet. Users access the application through a web interface, while AWS manages the underlying infrastructure, compute resources, storage, and backend services.

## Detailed Description

### Ease of Access and Convenience

- **Web-Based Access:** Users can access the portfolio generation service from any device with an internet connection using a standard web browser. This eliminates the need for users to install or maintain software locally, providing a seamless and convenient user experience.
- **Responsive Design:** The web interface, built using React, ensures that the application is accessible and functional across various devices, including desktops, tablets, and smartphones.

### Centralized Management and Updates

- **Automatic Updates:** The SaaS model allows for automatic updates and maintenance of the application. Users always have access to the latest version without needing to manually download or install updates. This ensures that the application is continuously improved and remains secure.
- **Centralized Control:** The application is centrally managed, allowing for streamlined administration and oversight. This includes managing user accounts, monitoring performance, and implementing security measures.

### Scalability and Flexibility

- **Elastic Compute Resources:** The application leverages AWS services such as EC2 and Lambda to dynamically scale based on user demand. This ensures optimal performance and availability, even under varying workloads.
- **Load Balancing:** AWS Elastic Load Balancing distributes incoming traffic across multiple instances, preventing any single instance from becoming a bottleneck and ensuring that the system can handle high volumes of concurrent users.

## Cost Efficiency

- **Pay-As-You-Go Model:** AWS's pay-as-you-go pricing model ensures that costs are incurred based on actual usage, eliminating the need for significant upfront investments in hardware and infrastructure. This model also allows for cost savings during periods of low usage.
- **Resource Optimization:** The SaaS delivery model allows for efficient resource utilization, with AWS managing the provisioning and de-provisioning of resources as needed.

## Enhanced Collaboration

- **User and Admin Interfaces:** The SaaS model supports multiple user roles, including end-users who upload resumes and generate portfolios, and administrators who manage the application and monitor its performance.
- **Real-Time Updates:** Changes and updates to the application are immediately reflected for all users, ensuring a consistent and up-to-date experience.

## Integrated Security and Compliance

- **Data Security:** AWS provides robust security measures, including encryption for data at rest and in transit, secure access controls, and compliance with industry standards and regulations. This ensures that user data is protected at all times.
- **IAM Policies:** AWS Identity and Access Management (IAM) is used to define fine-grained access controls, ensuring that only authorized users and components can access sensitive data and resources.

## Components of the SaaS Model

1. **Frontend (React Application):**
    - **Hosted on AWS EC2:** The frontend application is hosted on EC2 instances, providing a web interface for users to upload resumes and interact with the chatbot.
    - **Responsive Design:** Ensures accessibility across various devices and screen sizes.
2. **Backend (Python Application):**
    - **Hosted on AWS EC2:** The backend services, including API endpoints and business logic, are hosted on EC2 instances.
    - **API Integration:** The backend interacts with the ChatGPT API via AWS Lambda functions to process resumes and generate portfolio content.
3. **Database (MySQL):**

- **Hosted on AWS EC2:** The MySQL database is managed on an EC2 instance, storing user data, resumes, and portfolio information.
- **Secure Access:** Configured with security groups and IAM policies to restrict access to authorized components.

4. **AWS Lambda Functions:**
   - **Serverless Processing:** Lambda functions are used to handle specific tasks, such as invoking the ChatGPT API for generating portfolio content.
   - **Automatic Scaling:** Scales automatically based on the number of requests, ensuring efficient handling of workload spikes.
5. **AWS S3 (Simple Storage Service):**
   - **Static Asset Storage:** S3 is used to store the generated portfolio webpages, providing high availability and durability.
   - **Access Control:** Configured with bucket policies to control access to stored content.
6. **AWS VPC (Virtual Private Cloud):**
   - **Network Isolation:** Provides a secure network environment for the application components, ensuring that all internal communication is protected.
7. **AWS Elastic Load Balancing:**
   - **Traffic Management:** Distributes incoming traffic across multiple EC2 instances, ensuring high availability and reliability.
8. **AWS Secrets Manager:**
   - **Secure Storage:** Manages sensitive information such as API keys and database credentials, ensuring they are securely stored and accessed.

## Reasoning for Choosing the SaaS Delivery Model

1. **Ease of Access and User Experience:**
   - **Accessibility:** Users can access the application from anywhere without needing to install software, ensuring a wide reach and convenience.
   - **Consistency:** The SaaS model ensures that all users have a consistent experience with the latest features and updates.
2. **Scalability and Flexibility:**

   - **Dynamic Scaling:** AWS services like EC2 and Lambda automatically scale to meet user demand, ensuring optimal performance.
   - **Resource Management:** AWS manages the provisioning of resources, allowing the development team to focus on improving the application rather than managing infrastructure.
3. **Cost Efficiency:**
   - **Operational Cost Savings:** The pay-as-you-go model reduces operational costs by charging only for resources used, avoiding large upfront investments.
   - **Efficient Utilization:** Resources are allocated dynamically based on demand, ensuring cost-effective operation.
4. **Security and Compliance:**
   - **Robust Security Measures:** AWS provides comprehensive security features, including data encryption and access controls, ensuring that user data is protected.
   - **Regulatory Compliance:** AWS's adherence to industry standards and regulations helps ensure that the application meets compliance requirements.
5. **Centralized Management and Maintenance:**
   - **Automatic Updates:** The application is centrally managed, with updates and maintenance handled by the service provider, ensuring continuous improvement and security.
   - **Simplified Administration:** Centralized control simplifies the management of user accounts, performance monitoring, and security configurations.

This detailed description of the SaaS delivery model explains how the various components are integrated to provide a robust, scalable, and secure solution for the Automatic Portfolio Webpage Generation Chatbot. The SaaS model leverages AWS services to deliver a seamless user experience while ensuring efficient resource management and cost-effectiveness.

## System Architecture

The architecture of the Automatic Portfolio Webpage Generation Chatbot is designed to leverage various AWS services to ensure scalability, reliability, and security. The architecture consists of multiple components, each playing a crucial role in the overall functionality of the system.

### Components and Their Roles

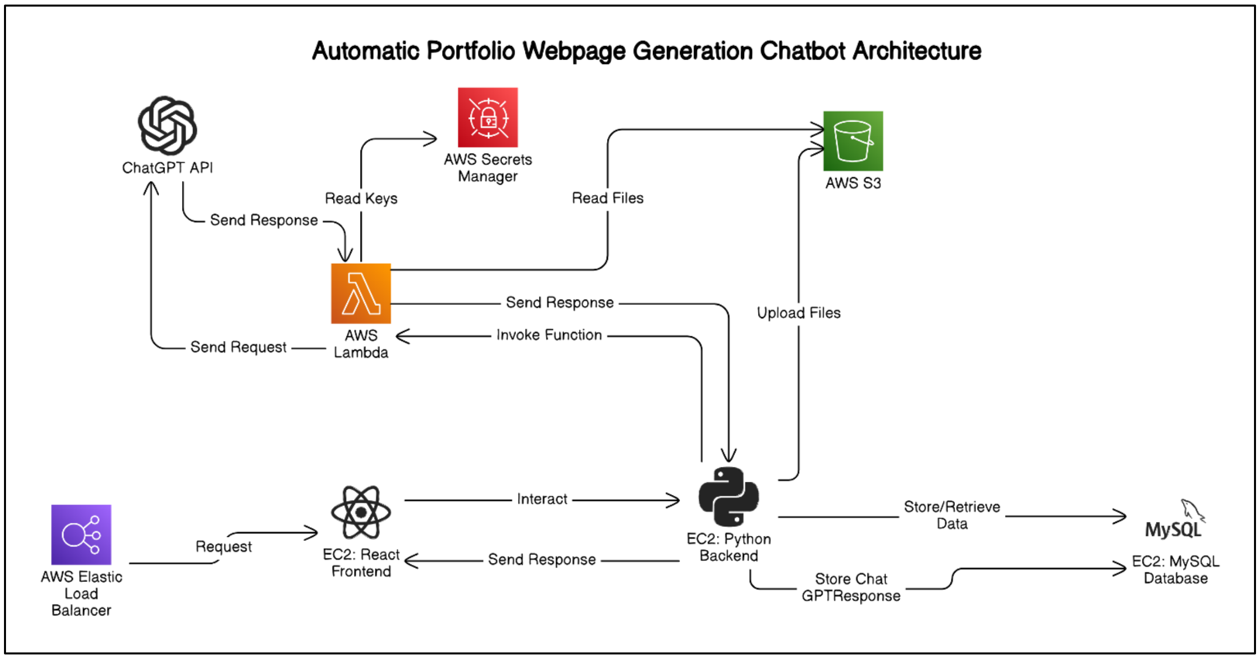
1. **Frontend (React Application):**
   - **Hosted on AWS EC2:** The frontend application provides an intuitive web interface for users to upload resumes and interact with the chatbot. It is hosted on an EC2 instance to ensure flexibility and control over the environment.
   - **User Interaction:** Users can upload their resumes and receive generated portfolio webpages through this interface.
2. **Backend (Python Application):**
   - **Hosted on AWS EC2:** The backend services, which include API endpoints and business logic, are also hosted on EC2 instances. This ensures the backend can be finely tuned and scaled independently of the frontend.
   - **API Integration:** The backend interacts with the ChatGPT API via AWS Lambda functions to process resumes and generate portfolio content.
3. **Database (MySQL):**
   - **Hosted on AWS EC2:** The MySQL database stores user data, resumes, and portfolio information. It is hosted on an EC2 instance, providing robust data management capabilities.
   - **Secure Access:** The database is configured with security groups and IAM policies to restrict access to authorized components only.
4. **AWS Lambda Functions:**
   - **Serverless Processing:** Lambda functions handle specific tasks, such as invoking the ChatGPT API to process resume data and generate portfolio content.
   - **Automatic Scaling:** These functions automatically scale based on the number of requests, ensuring efficient handling of workload spikes.
5. **AWS S3 (Simple Storage Service):**

   - **Static Asset Storage:** Generated portfolio webpages are stored in S3, providing high availability and durability.
   - **Access Control:** S3 bucket policies are configured to control access to the stored content.

6. **AWS VPC (Virtual Private Cloud):**
   - **Network Isolation:** The VPC provides a secure and isolated network environment for the application components, ensuring all internal communication is protected.
   - **Subnets and Security Groups:** VPC subnets and security groups are configured to control traffic flow and enhance security.
7. **AWS Elastic Load Balancing:**
   - **Traffic Management:** Elastic Load Balancing distributes incoming traffic across multiple EC2 instances, ensuring high availability and reliability.
8. **AWS Secrets Manager:**
   - **Secure Storage:** Secrets Manager is used to store and manage sensitive information such as API keys and database credentials, ensuring secure access.

## Architecture Diagram

Below is a detailed architecture diagram illustrating how these components interact within the system.



## Detailed Component Interaction

1. **User Interaction:**
   - Users interact with the React frontend hosted on EC2.
   - Users upload their resumes through the web interface.
2. **Data Processing:**
   - The uploaded resume is sent to the backend (Python application) hosted on EC2.
   - The backend stores the resume data in the MySQL database and invokes an AWS Lambda function to process the resume.
3. **Portfolio Generation:**
   - The AWS Lambda function calls the ChatGPT API to generate portfolio content based on the resume data.
   - The generated portfolio content is sent back to the backend, which formats it into a static webpage.
4. **Storage and Deployment:**
   - The backend stores the generated portfolio webpage in an AWS S3 bucket.
   - The frontend provides a link to the user to access their generated portfolio webpage.
5. **Network and Security:**
   - All components communicate within a secure VPC.
   - Elastic Load Balancing manages incoming traffic, ensuring high availability.
   - AWS Secrets Manager securely stores and provides access to sensitive information like API keys and database credentials.

## Data Flow

1. **Resume Upload:**
   - Users upload resumes via the React frontend.
   - The frontend sends the resume data to the backend.

2. **Resume Processing:**
    - The backend stores the resume data in the MySQL database.
    - The backend triggers an AWS Lambda function to process the resume.
3. **Portfolio Generation:**
    - The Lambda function processes the resume and generates portfolio content using the ChatGPT API.
    - The generated content is returned to the backend.
4. **Webpage Storage:**
    - The backend formats the content into a static webpage and stores it in an AWS S3 bucket.
    - The frontend provides the user with a link to their generated portfolio webpage.
5. **Traffic Management and Security:**
    - Elastic Load Balancing distributes incoming requests across multiple EC2 instances.
    - All data is transmitted securely within the VPC.
    - AWS Secrets Manager handles sensitive information securely.

## Benefits of the Architecture

1. **Scalability:**
    - The use of AWS Lambda for serverless processing ensures that the system can scale automatically based on demand.
    - EC2 instances can be scaled up or down as needed to handle varying loads.
2. **Reliability:**
    - Elastic Load Balancing ensures high availability by distributing traffic across multiple instances.
    - AWS S3 provides high durability and availability for storing static content.
3. **Security:**
    - AWS VPC provides a secure and isolated network environment.
    - AWS Secrets Manager ensures that sensitive information is securely stored and accessed.

4. **Cost-Efficiency:**
    - The pay-as-you-go pricing model of AWS services ensures that costs are incurred based on actual usage, reducing unnecessary expenditure.
    - Serverless functions in AWS Lambda minimize operational overhead and costs.

## Security Approach

Ensuring the security of user data and the integrity of the application is paramount in the architecture of the Automatic Portfolio Webpage Generation Chatbot. This section details the security measures implemented at various stages of the system, from data transmission to storage, and the mechanisms used to enforce these measures.

### Data Security

#### Data in Transit

1. **HTTPS:** All communication between the frontend (React application) and the backend (Python application), as well as interactions with external APIs (such as ChatGPT), are conducted over HTTPS. This ensures that data transmitted over the network is encrypted, protecting it from eavesdropping and man-in-the-middle attacks.
2. **Secure API Calls:** When the backend calls the ChatGPT API via AWS Lambda, the request is made over HTTPS, ensuring that the data remains encrypted and secure during transmission.

#### Data at Rest

1. **AWS S3 Encryption:** Data stored in AWS S3, such as generated portfolio webpages, is encrypted at rest using server-side encryption (SSE). AWS S3 supports encryption with AWS-managed keys (SSE-S3) or customer-managed keys stored in AWS Key Management Service (SSE-KMS).
2. **MySQL Database Encryption:** The MySQL database hosted on an EC2 instance is configured to use encryption for data at rest. This ensures that sensitive user information and portfolio data are protected from unauthorized access, even if the underlying storage is compromised.
3. **AWS Secrets Manager:** Sensitive information, such as API keys and database credentials, is stored securely in AWS Secrets Manager. Secrets Manager encrypts these secrets at rest using AWS KMS, ensuring that only authorized services and users can access them.

#### Network Security

1. **AWS VPC:** The entire application is deployed within an AWS Virtual Private Cloud (VPC). This provides a logically isolated network environment, allowing for secure communication between application components.
2. **Security Groups:** Security groups act as virtual firewalls for EC2 instances, controlling inbound and outbound traffic. They are configured to allow only necessary traffic, minimizing the attack surface.

3. **Subnets and Routing:** The VPC is divided into subnets to further isolate resources. Public subnets are used for frontend components accessible via the internet, while private subnets are used for backend services and databases to prevent direct internet access.

## Access Control

1. **IAM Roles and Policies:** AWS Identity and Access Management (IAM) is used to define roles and policies that grant the least privilege necessary for each component to function. This ensures that each service has only the permissions it needs to perform its tasks, reducing the risk of accidental or malicious actions.
    - **Lambda Execution Role:** The Lambda function is assigned an execution role with permissions to read from S3, access secrets from Secrets Manager, and invoke the ChatGPT API.
    - **EC2 Instance Roles:** EC2 instances hosting the frontend, backend, and database are assigned roles that permit them to interact with other AWS services securely.

## Application-Level Security

1. **Authentication and Authorization:** Future enhancements will include implementing user authentication and authorization using AWS Cognito, ensuring that only authorized users can access and manage their portfolio data.
2. **Input Validation:** The backend services perform thorough input validation to prevent injection attacks, such as SQL injection or cross-site scripting (XSS). This includes sanitizing user inputs and validating data before processing it.
3. **Logging and Monitoring:** AWS CloudWatch is used to monitor the application and log access to AWS resources. This includes logging API requests, Lambda function executions, and access to the S3 bucket. Logs are monitored for suspicious activities, and alerts are configured to notify administrators of potential security incidents.

## Additional Security Measures

1. **Regular Security Audits:** Regular security audits and vulnerability assessments are conducted to identify and mitigate potential security risks. This includes reviewing IAM policies, network configurations, and application code.

# Cost Analysis

## Cloud Model Used on AWS

### EC2 Instance

- **Instance Type:** t2.medium
- **Current On-Demand Price/hr:** $0.0464
- **Monthly Cost (24/7 operation):** $0.0464 * 24 hours * 30 days = $33.408 per instance

Assuming:

- 1 EC2 instance for hosting the frontend, backend, and MySQL database.

**Total Monthly Cost for EC2 Instance:** $8.352

### AWS S3

- **Storage Cost (first 50 TB/Month):** $0.023 per GB
- **Cost for Monthly Storing 10 GB:** $0.023 * 10 = $0.23
- **Request and Data Retrieval Costs:**
  - PUT, COPY, POST, LIST requests (per 1,000 requests): $0.005
  - GET, SELECT, and all other requests (per 1,000 requests): $0.0004

Assuming 10,000 PUT and 10,000 GET requests per month:

- **Total Cost for PUT and GET requests:** 10 * 0.005 + 10 * 0.0004 = $0.054

**Total Monthly Cost for S3:** $0.23 + $0.054 = $0.284

### AWS Lambda

- **Request Cost per 1M requests:** $0.20
- **Duration Cost:** $0.0000166667 for every GB-second
- **Monthly Cost for 10,000 Invocations:** \frac{0.20 * 10,000}{1,000,000} = $0.002

### AWS Secrets Manager

- **Secrets Management Cost:** $0.40 per secret per month
- **Assuming 5 Secrets:** $0.40 * 5 = $2.00 per month

## ChatGPT API

- **Estimated Cost per Request:** $0.01 (assuming a hypothetical cost)
- **Monthly Cost for 10,000 Requests:** 0.01 * 10,000 = $100.00

## AWS Elastic Load Balancer

- **Application Load Balancer Cost:**
  - Fixed cost per hour: $0.0225
  - Data processing cost: $0.008 per GB

Assuming:

- 720 hours in a month.
- 100 GB data processed per month.

**Total Monthly Cost for Load Balancer:** 720 * $0.0225 + 100 * $0.008 = $16.20 + $0.80 = $17.00

## AWS Elastic Container Registry (ECR)

- **Storage per GB/month:** $0.10
- **Assuming 200 MB image size:** $0.10 * 0.2 = $0.02 per month

---

## Summary of Monthly Cloud Costs

1. **EC2 Instance:** $33.408
2. **S3 Storage:** $0.284
3. **Lambda Functions:** $0.002
4. **Secrets Manager:** $2.00
5. **ChatGPT API:** $100.00
6. **Application Load Balancer:** $17.00
7. **Elastic Container Registry:** $0.02

**Total Monthly Cloud Cost:** $33.408 + $0.284 + $0.002 + $2.00 + $100.00 + $17.00 + $0.02 = $152.714

---

## On-Premise Cost

### EC2 Instance Equivalent

- **Hardware Cost:**
  - HP ALPHASERVER DS25: $1,575.00
  - CPU: Alpha 21264C 1000-MHz processor with 8-MB L2 Dual Data Rate cache
  - Memory: 1GB ECC-protected memory bus and cache
  - HDD: 500 GB SATA 3 5400 RPM
  - Power Supply: 500 watts
- **Power Consumption:**
  - Power for running 24 hours (full load): 500W * 24hr = 12,000 Wh or 12 kWh
  - Power consumption for 30 days: 12 kWh * 30 = 360 kWh
  - Average electricity rate: $0.1643 per kWh

**Total Monthly Cost for the System:** 360 kWh * 0.1643 = $59.148

### Lambda Function and ChatGPT API Equivalent

- **Hardware Cost (High Spec CPU, GPU, Memory):**
  - Total Cost: $4,008.56 (Based on the provided PC part list)
- **Power Consumption:**
  - Power for running 24 hours (full load): 1,650W * 24 hr = 39.6 kWh
  - Power consumption for 30 days: 39.6 kWh * 30 = 1,188 kWh
  - Average electricity rate: $0.1643 per kWh

**Total Monthly Cost for the System:** 1,188 kWh * 0.1643 = $195.1884

### S3 Storage Cost Equivalent

- **Hardware Cost:**
  - 8TB NAS Storage: $735.00
- **Power Consumption:**
  - Power for running 24 hours (full load): 500W * 24hr = 12,000 Wh or 12 kWh

- Power consumption for 30 days: 12 kWh * 30 = 360 kWh
- Average electricity rate: $0.1643 per kWh

**Total Monthly Cost for the System:** 360 kWh * 0.1643 = $59.148

---

## Summary of Monthly On-Premise Costs

1. **EC2 Instance Equivalent:** $59.148
2. **High Spec System for Lambda and ChatGPT Equivalent:** $195.1884
3. **NAS Storage System:** $59.148

**Total Monthly On-Premise Cost:** $59.148 + $195.1884 + $59.148 = $313.4844

---

## Comparison

**Total Monthly Cloud Cost:** $127.658

**Total Monthly On-Premise Cost:** $313.4844

**Cost Analysis Summary**

By using AWS cloud services, the Automatic Portfolio Webpage Generation Chatbot benefits from significant cost savings. The monthly cost of operating the cloud-based solution is approximately $127.658, whereas maintaining an equivalent on-premise infrastructure would cost about $313.4844 per month. This analysis demonstrates that leveraging cloud services not only reduces operational costs but also provides additional benefits in terms of scalability, reliability, and security.

# Monitoring and Cost Management

## Key Cloud Mechanisms for Monitoring

To ensure the costs of the Automatic Portfolio Webpage Generation Chatbot remain within budget, it is essential to monitor various AWS services and resources. Each service contributes to the overall cost, and monitoring their usage helps prevent unexpected expenses. Here, we focus on the cloud mechanisms that have the most potential to cost the most money and require diligent monitoring.

**1. AWS Lambda**

**Reason for Monitoring:**

- AWS Lambda functions can quickly incur costs if they are invoked frequently, run for long durations, or handle large volumes of data. Since the application uses Lambda to invoke the ChatGPT API, it is crucial to monitor the number of invocations and the execution duration to prevent cost overruns.

**Monitoring Strategies:**

- **AWS CloudWatch Metrics:** Utilize CloudWatch to monitor key metrics such as the number of invocations, average duration, and errors.
- **CloudWatch Alarms:** Set up alarms to trigger notifications if invocation counts or execution durations exceed predefined thresholds.
- **Cost and Usage Reports:** Regularly review AWS Cost and Usage Reports to track Lambda usage and costs.

**2. AWS EC2**

**Reason for Monitoring:**

- EC2 instances are a significant cost factor, especially when running 24/7. Monitoring the instance usage, CPU utilization, and network traffic can help optimize the performance and cost-efficiency of the EC2 instances.

**Monitoring Strategies:**

- **AWS CloudWatch Metrics:** Monitor CPU utilization, network traffic, and disk I/O to ensure the instance is running efficiently.
- **Auto Scaling:** Consider setting up Auto Scaling to adjust the number of instances based on demand, which can help optimize costs.
- **Instance Scheduler:** Implement an instance scheduler to start and stop instances based on usage patterns, reducing costs during off-peak hours.

**3. AWS S3**

**Reason for Monitoring:**

- While S3 storage costs are relatively low, they can add up if there are large volumes of data or a high number of requests. Monitoring storage usage and request counts can help manage costs effectively.

**Monitoring Strategies:**

- **S3 Metrics and Storage Lens:** Use S3 Storage Lens and CloudWatch metrics to monitor storage usage, number of requests, and data transfer.
- **Lifecycle Policies:** Implement S3 lifecycle policies to transition data to lower-cost storage classes or delete unnecessary data, reducing storage costs.
- **Request Monitoring:** Monitor the number of PUT, GET, and other requests to identify and optimize high-usage patterns.

**4. AWS Secrets Manager**

**Reason for Monitoring:**

- Secrets Manager costs are primarily driven by the number of secrets stored and the number of API calls to retrieve them. Monitoring usage helps ensure secrets are managed efficiently.

**Monitoring Strategies:**

- **CloudWatch Metrics:** Monitor the number of API calls and the number of secrets managed in Secrets Manager.
- **Secret Rotation:** Regularly review and rotate secrets to ensure they are up-to-date and minimize the risk of security breaches.
- **Cost Allocation Tags:** Use cost allocation tags to track and manage the costs associated with secrets management.

**5. AWS Elastic Load Balancer (ELB)**

**Reason for Monitoring:**

- ELB costs are based on the number of hours it is running and the amount of data processed. Monitoring the load balancer's usage helps optimize costs.

**Monitoring Strategies:**

- **CloudWatch Metrics:** Monitor request counts, healthy and unhealthy host counts, and latency metrics.
- **Cost and Usage Reports:** Regularly review cost and usage reports to track ELB usage and costs.

- **Traffic Patterns:** Analyze traffic patterns to optimize load balancer configuration and reduce unnecessary costs.

## Monitoring Tools and Techniques

### AWS CloudWatch

AWS CloudWatch is a comprehensive monitoring and management service that provides data and actionable insights for AWS resources, applications, and services. Key features include:

- **Metrics:** Collect and track metrics, such as CPU usage, memory usage, request counts, and error rates, for all AWS services.
- **Logs:** Collect and monitor log data from applications and AWS services, enabling detailed analysis and troubleshooting.
- **Alarms:** Set up alarms to trigger notifications or automated actions based on metric thresholds.
- **Dashboards:** Create custom dashboards to visualize key metrics and monitor the health and performance of the application.

### AWS Cost Explorer

AWS Cost Explorer provides tools to view and analyze AWS costs and usage. Key features include:

- **Cost and Usage Reports:** Generate detailed reports on AWS costs and usage, broken down by service, region, and time period.
- **Forecasting:** Predict future costs based on historical usage patterns.
- **Cost Allocation Tags:** Use tags to allocate costs to specific projects, departments, or teams, enabling better cost management.

### AWS Budgets

AWS Budgets allows you to set custom cost and usage budgets and receive notifications when usage or costs exceed the thresholds. Key features include:

- **Budget Tracking:** Track costs and usage against predefined budgets.
- **Alerts and Notifications:** Set up alerts to notify you when costs or usage exceed budget thresholds.
- **Cost Anomalies:** Identify cost anomalies and take corrective actions to prevent budget overruns.

## Future Development and Feature Enhancements

As the Automatic Portfolio Webpage Generation Chatbot evolves, several new features and enhancements can be added to improve its functionality, user experience, and scalability. Below are some potential features and the associated AWS cloud mechanisms that could be used to implement them.

### 1. User Authentication and Management

**Feature:**

- Implement user authentication and authorization to allow users to create accounts, log in, and manage their portfolio securely.

**Cloud Mechanism:**

- **AWS Cognito:** Use Amazon Cognito to handle user sign-up, sign-in, and access control. Cognito provides scalable user identity and authentication management with support for multi-factor authentication (MFA) and social identity providers (e.g., Google, Facebook).

**Implementation:**

- Integrate Cognito with the frontend application to enable user authentication.
- Set up user pools and identity pools to manage user identities and provide temporary credentials for accessing AWS resources securely.

### 2. Enhanced Portfolio Customization

**Feature:**

- Allow users to choose and customize portfolio templates, add personal branding elements, and select different layouts and themes.

**Cloud Mechanism:**

- **AWS Amplify:** Use AWS Amplify to streamline the development and deployment of the frontend application. Amplify provides tools and services to build scalable mobile and web applications, including support for frontend hosting, backend integration, and storage.

**Implementation:**

- Integrate Amplify with the React frontend application to manage static content and deploy new features seamlessly.
- Use Amplify's UI components to enable template customization and real-time updates.

**3. Analytics Dashboard**

**Feature:**

- Provide users with an analytics dashboard to track portfolio views, interactions, and engagement metrics.

**Cloud Mechanism:**

- **AWS QuickSight:** Use Amazon QuickSight to build interactive and scalable business intelligence dashboards. QuickSight can connect to various data sources, perform real-time analysis, and provide insights through visualizations.

**Implementation:**

- Integrate QuickSight with the backend application to collect and analyze data on portfolio views and interactions.
- Provide users with a customizable dashboard to view their analytics and gain insights into their portfolio performance.

**4. Improved Performance and Scalability**

**Feature:**

- Enhance the performance and scalability of the application to handle a larger user base and increased traffic.

**Cloud Mechanism:**

- **AWS Auto Scaling:** Use Auto Scaling to automatically adjust the number of EC2 instances based on demand. This ensures optimal performance and cost-efficiency by scaling resources up or down as needed.
- **AWS CloudFront:** Use Amazon CloudFront to deliver static and dynamic content globally with low latency and high transfer speeds. CloudFront caches content at edge locations, improving performance for end-users.

**Implementation:**

- Configure Auto Scaling for the EC2 instances hosting the frontend and backend applications.
- Integrate CloudFront with the S3 bucket storing static portfolio content to accelerate content delivery.

**5. Serverless API Gateway**

**Feature:**

- Implement a serverless API gateway to manage and secure API requests, providing a scalable and cost-effective solution for backend services.

**Cloud Mechanism:**

- **AWS API Gateway:** Use Amazon API Gateway to create, deploy, and manage RESTful APIs. API Gateway provides features like throttling, caching, and monitoring, ensuring secure and reliable API access.

**Implementation:**

- Set up API Gateway to handle API requests from the frontend application.
- Integrate API Gateway with AWS Lambda functions to process requests and interact with the backend services.

**6. Enhanced Security and Compliance**

**Feature:**

- Improve the security and compliance of the application to meet industry standards and protect user data.

**Cloud Mechanism:**

- **AWS Shield:** Use AWS Shield to protect the application from DDoS attacks. Shield provides advanced threat detection and mitigation capabilities.
- **AWS Config:** Use AWS Config to monitor and assess the configuration of AWS resources continuously. Config helps ensure compliance with security policies and standards.

**Implementation:**

- Enable AWS Shield for the application to detect and mitigate DDoS attacks automatically.
- Set up AWS Config rules to monitor resource configurations and enforce compliance with security policies.

## 7. Continuous Integration and Continuous Deployment (CI/CD)

**Feature:**

- Implement a CI/CD pipeline to automate the build, test, and deployment processes, ensuring rapid and reliable delivery of new features and updates.

**Cloud Mechanism:**

- **AWS CodePipeline:** Use AWS CodePipeline to create a continuous delivery pipeline. CodePipeline automates the build, test, and deployment phases of application development.
- **AWS CodeBuild:** Use AWS CodeBuild to run build and test operations as part of the CI/CD pipeline.

**Implementation:**

- Set up CodePipeline to automate the deployment of the frontend and backend applications.
- Integrate CodeBuild with CodePipeline to handle the build and test stages, ensuring that new code is validated before deployment.

## 8. Multi-Region Deployment

**Feature:**

- Deploy the application in multiple AWS regions to improve availability, reduce latency, and provide disaster recovery capabilities.

**Cloud Mechanism:**

- **AWS Global Accelerator:** Use AWS Global Accelerator to route user traffic to the nearest AWS region, improving performance and availability.
- **AWS Route 53:** Use Amazon Route 53 for DNS management and routing user requests to different regions based on latency, geography, or other criteria.

**Implementation:**

- Deploy the application infrastructure in multiple AWS regions.
- Configure Global Accelerator and Route 53 to route user traffic to the nearest region and provide failover capabilities.

## References

[1] “Amazon EC2 - Cloud Compute Capacity - AWS,” aws.amazon.com [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: July 24, 2024].

[2] “Amazon S3 Storage,” aws.amazon.com [Online]. Available: https://aws.amazon.com/s3/. [Accessed: July 25, 2024].

[3] “AWS Lambda,” aws.amazon.com [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: July 26, 2024].

[4] “Amazon API Gateway,” aws.amazon.com [Online]. Available: https://aws.amazon.com/api-gateway/. [Accessed: July 27, 2024].

[5] “Amazon Cognito,” aws.amazon.com [Online]. Available: https://aws.amazon.com/cognito/. [Accessed: July 28, 2024].

[6] “Amazon CloudFront,” aws.amazon.com [Online]. Available: https://aws.amazon.com/cloudfront/. [Accessed: July 29, 2024].

[7] “Amazon Elastic Load Balancing (ELB) - AWS,” aws.amazon.com [Online]. Available: https://aws.amazon.com/elasticloadbalancing/. [Accessed: July 30, 2024].

[8] “Amazon ECR,” aws.amazon.com [Online]. Available: https://aws.amazon.com/ecr/. [Accessed: July 31, 2024].

[9] “Amazon QuickSight,” aws.amazon.com [Online]. Available: https://aws.amazon.com/quicksight/. [Accessed: August 1, 2024].

[10] “AWS Auto Scaling,” aws.amazon.com [Online]. Available: https://aws.amazon.com/autoscaling/. [Accessed: August 2, 2024].

[11] “AWS Amplify,” aws.amazon.com [Online]. Available: https://aws.amazon.com/amplify/. [Accessed: August 3, 2024].

[12] “AWS Shield,” aws.amazon.com [Online]. Available: https://aws.amazon.com/shield/. [Accessed: August 4, 2024].

[13] “AWS Config,” aws.amazon.com [Online]. Available: https://aws.amazon.com/config/. [Accessed: August 5, 2024].

[14] “AWS Global Accelerator,” aws.amazon.com [Online]. Available: https://aws.amazon.com/global-accelerator/. [Accessed: August 6, 2024].

[15] “Amazon Route 53,” aws.amazon.com [Online]. Available: https://aws.amazon.com/route53/. [Accessed: August 7, 2024].

[16] “AWS CloudWatch,” aws.amazon.com [Online]. Available: https://aws.amazon.com/cloudwatch/. [Accessed: July 24, 2024].

[17] “AWS Cost Explorer,” aws.amazon.com [Online]. Available: https://aws.amazon.com/aws-cost-explorer/. [Accessed: July 25, 2024].

[18] “AWS Budgets,” aws.amazon.com [Online]. Available: https://aws.amazon.com/aws-cost-management/aws-budgets/. [Accessed: July 26, 2024].

[19] “AWS CodePipeline,” aws.amazon.com [Online]. Available: https://aws.amazon.com/codepipeline/. [Accessed: July 27, 2024].

[20] “AWS CodeBuild,” aws.amazon.com [Online]. Available: https://aws.amazon.com/codebuild/. [Accessed: July 28, 2024].

[21] “Electricity Rates for Every State in The US - EnergyBot,” energybot.com [Online]. Available: https://www.energybot.com/electricity-rates/. [Accessed: July 29, 2024].

[22] “AMD Threadripper 3960X 3.8 GHz 24-Core Processor - PCPartPicker,” pcpartpicker.com [Online]. Available: https://pcpartpicker.com/list/TTt3Cd. [Accessed: July 30, 2024].

[23] “Using server-side encryption with Amazon S3 managed keys (SSE-S3),” docs.aws.amazon.com [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingServerSideEncryption.html. [Accessed: July 31, 2024].

[24] “AWS Key Management Service,” docs.aws.amazon.com [Online]. Available: https://docs.aws.amazon.com/kms/latest/developerguide/overview.html. [Accessed: August 1, 2024].

[25] “Control subnet traffic with network access control lists,” docs.aws.amazon.com [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html. [Accessed: August 2, 2024].