



CSCI 5411  
Advance Cloud Architecting  
Fall 2024

Term Project Report

Course Instructor: Professor Lu Yang

Author: Harsh Vaghasiya [B00986219]

Gitlab Repository Link: [https://github.com/HarshVagh/portfolio\\_ai\\_chatbot](https://github.com/HarshVagh/portfolio_ai_chatbot)

## Contents

|  |    |
|--|----|
| Introduction.....  | 4  |
| Overview of the Project Objective.....                                   | 4  |
| Importance of Hosting Applications on AWS.....                           | 4  |
| Brief Explanation of the Selected Application and Its Significance ..... | 4  |
| Application Selection .....  | 6  |
| Details of the Chosen Open-Source Application .....                      | 6  |
| Justification for Selecting the Application .....                        | 6  |
| Cloud Architecture Design.....   | 7  |
| Description of the Overall Architecture .....                            | 7  |
| Architecture Diagram.....  | 8  |
| Diagram Description: .....   | 9  |
| Explanation and Detailed Justification for AWS Services Chosen .....     | 10 |
| Summary of Architectural Decisions .....                                 | 13 |
| AWS Well-Architected Framework .....                                     | 14 |
| Brief Explanation of the Six Pillars.....                                | 14 |
| How These Pillars Guided the Architectural Decisions .....               | 14 |
| Summary of AWS Well-Architected Framework Alignment.....                 | 16 |
| Infrastructure as Code (IaC) Implementation .....                        | 17 |
| Description of the Tools Used.....                                       | 17 |
| Explanation of the IaC Script and Its Functionality .....                | 17 |
| Challenges Faced and Solutions Implemented .....                         | 22 |
| Deployment on AWS .....  | 23 |
| Description of the Deployment Process.....                               | 23 |
| Restrictions and Challenges Due to Service Limitations .....             | 24 |
| Hosted Application's Public URL .....                                    | 25 |
| Evidence of Deployment: .....  | 25 |
| Frontend Interface: .....  | 25 |
| VPC: .....   | 29 |
| Load Balancer: .....   | 29 |
| ECS: .....   | 30 |
| Cost Estimation .....  | 31 |
| Detailed Cost Breakdown .....  | 31 |

|  |    |
|--|----|
| Overall Estimated Monthly Cost .....                                   | 34 |
| Challenges and Solutions.....  | 35 |
| Challenges Encountered During Architecture Design and Deployment ..... | 35 |
| Conclusion .....   | 37 |
| Summary of the Project Outcome .....                                   | 37 |
| Reflection on Learning and Skills Developed During the Project.....    | 37 |
| References .....   | 39 |

# Introduction

## Overview of the Project Objective

The Automatic Portfolio Webpage Generation Chatbot is an innovative application designed to revolutionize the process of creating professional portfolio webpages. By leveraging user-uploaded resumes, this system automates the transformation of raw resume data into polished, professional portfolio websites. The chatbot interacts with users to extract relevant information from resumes and dynamically generates portfolio webpages using the ChatGPT API. These webpages are subsequently deployed online, ensuring accessibility and usability. The project aims to deliver a seamless, scalable, and secure solution for automating portfolio creation while optimizing cloud resource utilization.

## Importance of Hosting Applications on AWS

Amazon Web Services (AWS) offers a robust platform for deploying and managing applications, with unparalleled scalability, security, and reliability. The platform's pay-as-you-go model ensures cost-efficiency, making it accessible for projects of various scales. Hosting applications on AWS allows developers to leverage advanced services such as Amazon ECS for container orchestration, AWS Lambda for serverless functions, and Amazon S3 for scalable storage [1][2][3]. These features are essential for ensuring that the chatbot can process multiple user requests simultaneously, store user data securely, and scale automatically based on traffic demands. Additionally, AWS services like CloudWatch provide real-time monitoring and alerting, enabling proactive system management [4].

## Brief Explanation of the Selected Application and Its Significance

The selected application, a GenAI-powered portfolio webpage generator chatbot, builds on a previously developed project from CSCI 5409 Advanced Topics in Cloud Computing. This application has been enhanced to include several new features and optimizations:

1. **Backend Modernization:** The backend was migrated from a Python Flask application to a Node.js Express application to improve performance and streamline integration with the frontend.
2. **Lambda Conversion:** AWS Lambda functions were converted to Node.js utility functions and integrated into the backend for improved efficiency and maintainability.

3. **Automation and AI Integration:** The application leverages the ChatGPT API to process resume data and generate visually appealing portfolio webpages, combining automation with generative AI.

This application is significant for its ability to address a critical need for job seekers and professionals who require an impactful online presence. It empowers users to create customized, professional portfolios with minimal effort, streamlining their journey toward personal branding and career growth.

# Application Selection

## Details of the Chosen Open-Source Application

The selected application for this project is a GenAI-powered portfolio webpage generator chatbot. Originally developed as part of the coursework for CSCI 5409 Advanced Topics in Cloud Computing, this application was enhanced to incorporate more robust features for automated portfolio creation. The updated version includes a backend migrated from Python Flask to Node.js Express, improved integration of utility functions derived from AWS Lambda, and additional optimizations for leveraging AWS cloud services. The application utilizes the ChatGPT API to transform user-uploaded resumes into professional, fully functional portfolio webpages [3][4].

## Justification for Selecting the Application

The selection of this application was driven by several key factors:

1. **Popularity and Relevance:** The chatbot addresses a highly relevant need for job seekers and professionals, offering a streamlined way to showcase skills and experiences online. Given the increasing importance of online branding in professional networking, this application caters to a broad audience with practical utility.
2. **Functionality:** The application stands out for its unique capability to automate portfolio creation using AI. By integrating resume processing with generative AI technologies like ChatGPT, it delivers a polished user experience that simplifies an otherwise complex process. The automated deployment feature ensures that users can access their webpages without additional technical expertise [3].
3. **Adaptability for Cloud Architecture:** The application's design aligns seamlessly with AWS cloud architecture principles. Key features such as its serverless backend, containerized frontend, and use of AWS services like S3, ECS, and DynamoDB ensure that the application is scalable, secure, and cost-efficient [1][2]. These attributes made it a strong candidate for this term project, where cloud architecting is a central focus.
4. **Alignment with Project Goals:** The application leverages advanced cloud technologies and generative AI, meeting the objectives of the term project. It also provided an opportunity to demonstrate the effective migration of a Python-based backend to Node.js while integrating modern cloud-native design principles [4].

By choosing this application, the project not only fulfills its academic objectives but also produces a practical solution that can be extended for real-world use.

# Cloud Architecture Design

## Description of the Overall Architecture

The architecture of the GenAI Portfolio Webpage Generator Chatbot is designed to be highly scalable, secure, and efficient, leveraging various AWS services to meet the application's requirements. The system is hosted within an Amazon Virtual Private Cloud (VPC) that spans three Availability Zones (AZs) to ensure high availability and fault tolerance. The architecture consists of both public and private subnets in each AZ, enabling isolation of resources and controlled network access.

An Application Load Balancer (ALB) distributes incoming HTTP requests to the frontend and backend services based on path-based routing. The frontend is a React application, and the backend is a Node.js Express app. Both services are containerized and run on AWS Fargate within Amazon Elastic Container Service (ECS), eliminating the need to manage servers.

The backend services reside in private subnets and require outbound internet access to interact with external APIs, such as the ChatGPT API, for portfolio generation. To facilitate this, NAT Gateways are deployed in each AZ's public subnet, allowing resources in private subnets to access the internet securely. Amazon DynamoDB is used as a NoSQL database to store user data, chats, and messages. Amazon Simple Storage Service (S3) buckets are utilized for storing user-uploaded resumes and generated portfolio webpages.

AWS Secrets Manager securely stores sensitive information like API keys, and Amazon CloudWatch monitors resource utilization and application performance. Auto Scaling policies are implemented to adjust the number of running tasks based on demand, ensuring cost-efficiency and optimal performance.

# Architecture Diagram

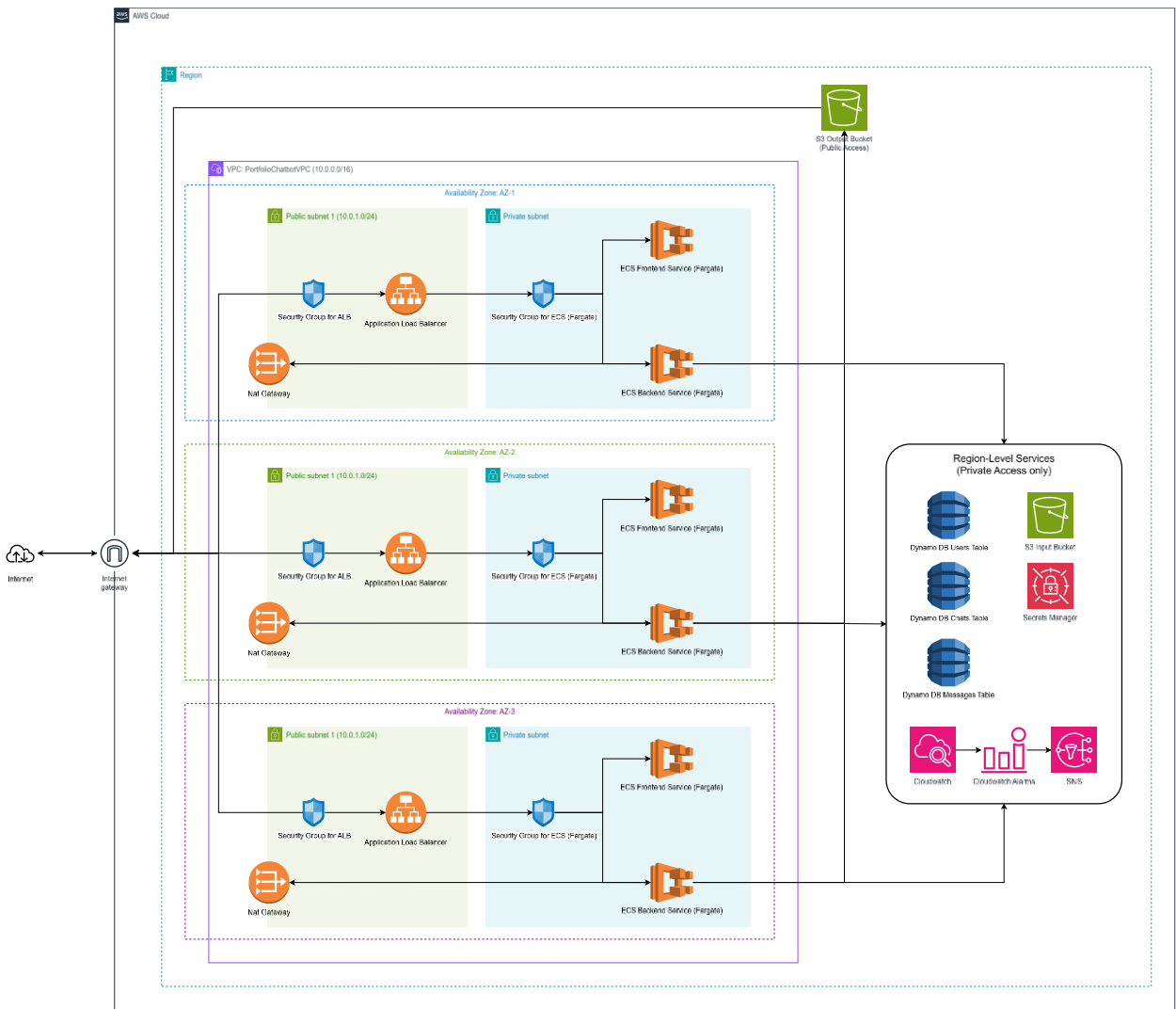


Figure 1: Cloud Architecture Diagram of AWS services



## Diagram Description:

- **VPC Configuration:**
  - **Public Subnets:** Contain the ALB and NAT Gateways in each of the three Availability Zones.
  - **Private Subnets:** Host ECS Fargate tasks for the frontend and backend services.
- **Application Load Balancer:**
  - Routes incoming HTTP requests to the appropriate service based on URL paths (e.g., /api/\* routes to the backend).
- **ECS Cluster:**
  - **Frontend Service:** Runs the React application in Docker containers managed by Fargate.
  - **Backend Service:** Runs the Node.js Express application, also containerized and managed by Fargate.
- **NAT Gateways:**
  - Provide outbound internet access for resources in private subnets, essential for the backend to communicate with the ChatGPT API.
- **Amazon DynamoDB:**
  - Stores user profiles, chat sessions, and messages.
- **Amazon S3 Buckets:**
  - **Input Bucket:** Stores user-uploaded resumes.
  - **Output Bucket:** Stores generated portfolio webpages, configured for static website hosting.
- **AWS Secrets Manager:**
  - Securely holds API keys and other sensitive configuration data.
- **Monitoring and Notifications:**
  - **Amazon CloudWatch:** Monitors system performance and resource utilization.
  - **Amazon SNS:** Sends email notifications when CloudWatch alarms are triggered.

# Explanation and Detailed Justification for AWS Services Chosen

## 1. Compute - AWS Fargate (Amazon ECS)

- **Why AWS Fargate?**

AWS Fargate eliminates the need to manage servers, providing a serverless compute platform for containerized applications [1]. The portfolio generator's React frontend and Node.js backend are deployed as containers, allowing the architecture to remain scalable and cost-efficient. Fargate ensures automatic scaling to handle fluctuations in user load, which is essential for a system that processes resource-intensive tasks like interacting with ChatGPT APIs.

- **Why Ideal for Portfolio Generator?**

The portfolio generator relies on the backend's ability to process uploaded resumes, generate portfolio content using the ChatGPT API, and handle real-time user requests. Fargate's support for microservice architectures makes it an excellent fit for separating concerns between the frontend and backend while maintaining performance and availability. This separation ensures a responsive user experience even under high traffic.

## 2. Storage - Amazon S3

- **Why Amazon S3?**

Amazon S3 offers highly durable and scalable storage with a pay-as-you-go pricing model [2]. The application uses two buckets: an input bucket for storing user-uploaded resumes and an output bucket for hosting generated portfolio webpages. Lifecycle management policies reduce costs by moving data to infrequent access or archival storage tiers over time.

- **Why Ideal for Portfolio Generator?**

Portfolio webpages need to be accessible as static content, which S3 supports natively through static website hosting. The separation of input and output data ensures efficient data handling while maintaining security and performance. S3's durability (99.999999999%) guarantees that user data and generated content are securely preserved and highly available.

## 3. Database - Amazon DynamoDB

- **Why Amazon DynamoDB?**

DynamoDB is a fully managed NoSQL database service offering single-digit millisecond latency [5]. It stores structured data like user profiles, chat histories, and messages, allowing for seamless retrieval during user interactions. The global secondary indexes enhance performance by enabling efficient queries based on user email or chat session IDs.

- **Why Ideal for Portfolio Generator?**

The portfolio generator processes real-time requests and retrieves historical data, such as previous user interactions and portfolio versions. DynamoDB's scalability ensures consistent performance even with increasing users and concurrent sessions. Its managed nature reduces operational overhead, letting developers focus on improving application features.

#### **4. Networking & Content Delivery - Amazon VPC, NAT Gateways, and Application Load Balancer (ALB)**

- **Why These Services?**

- **Amazon VPC:** Ensures network isolation and secure communication between components [6].
- **NAT Gateways:** Provide internet access for backend services residing in private subnets, allowing them to connect to external APIs like ChatGPT [7].
- **ALB:** Distributes incoming requests between frontend and backend services based on path-based routing [8].

- **Why Ideal for Portfolio Generator?**

The backend heavily relies on secure internet access for interacting with the ChatGPT API, making NAT Gateways indispensable. Deploying NAT Gateways in all three AZs ensures high availability and minimizes latency for API interactions. The ALB simplifies routing by handling requests like / for frontend resources and /api/\* for backend logic, ensuring efficient traffic management and improving user response times.

#### **5. Security, Identity, and Compliance - AWS Secrets Manager**

- **Why AWS Secrets Manager?**

AWS Secrets Manager securely stores and rotates sensitive information, such as ChatGPT API keys, reducing the risk of accidental exposure [9]. It integrates seamlessly with other AWS services, allowing the application to retrieve secrets without embedding them in code.

- **Why Ideal for Portfolio Generator?**

The portfolio generator requires secure API key storage for interacting with the ChatGPT API. With automatic rotation, Secrets Manager ensures that these keys remain secure while minimizing manual updates in the application. This is critical in maintaining compliance and protecting user data in a system dealing with personal information.

## 6. Monitoring & Logging - Amazon CloudWatch

- **Why Amazon CloudWatch?**

CloudWatch provides real-time monitoring of system performance and usage metrics, such as CPU utilization and memory usage, enabling proactive management of resources [4]. Alarms notify administrators when thresholds are breached, facilitating quick responses to potential issues.

- **Why Ideal for Portfolio Generator?**

The application processes data-intensive tasks, and CloudWatch ensures visibility into resource utilization for both frontend and backend services. Scaling decisions are based on CloudWatch metrics, ensuring optimal performance under varying loads. For instance, high CPU utilization can trigger auto-scaling, maintaining responsiveness during peak usage.

## 7. Management & Governance - AWS CloudFormation

- **Why AWS CloudFormation?**

CloudFormation enables Infrastructure as Code (IaC), ensuring consistent deployment of resources across environments [10]. It simplifies resource management by automating provisioning and updates.

- **Why Ideal for Portfolio Generator?**

The portfolio generator's architecture comprises numerous interconnected resources, including ECS clusters, ALBs, DynamoDB tables, and S3 buckets. CloudFormation streamlines deployment and reduces the likelihood of misconfigurations, ensuring a reliable and repeatable setup process.

## 8. Application Integration - Amazon SNS

- **Why Amazon SNS?**

SNS facilitates reliable message delivery to notify administrators about critical events, such as CloudWatch alarms for high CPU or memory usage. Notifications can be sent via email, ensuring timely alerts.

- **Why Ideal for Portfolio Generator?**

With multiple moving parts in the architecture, real-time notifications are essential to maintain system health. SNS enables proactive monitoring by alerting administrators of potential issues, such as resource overutilization, preventing application downtime.

## Summary of Architectural Decisions

The portfolio generator relies on the synergy of these AWS services to create a secure, scalable, and efficient cloud-based system. Each service was carefully chosen to address specific requirements of the use case:

- AWS Fargate for efficient compute resources.
- S3 for durable storage and static website hosting.
- DynamoDB for low-latency, scalable data access.
- VPC, NAT Gateways, and ALB for secure networking and traffic management.
- Secrets Manager for robust security.
- CloudWatch for monitoring and auto-scaling.
- CloudFormation for consistent resource provisioning.
- SNS for real-time notifications.

# AWS Well-Architected Framework

## Brief Explanation of the Six Pillars

1. **Operational Excellence:** Focuses on running and monitoring systems to deliver business value and continuously improve supporting processes and procedures. Emphasizes automation, infrastructure as code (IaC), and observability [1].
2. **Security:** Ensures the protection of data, systems, and assets while delivering business value. It includes identity and access management, data protection, and threat detection and response [2].
3. **Reliability:** Focuses on ensuring a workload performs its intended function consistently. It emphasizes fault-tolerant architectures, automatic recovery, and reliable infrastructure design [3].
4. **Performance Efficiency:** Involves using resources efficiently to meet system requirements, emphasizing scalability, monitoring, and adaptable technologies [4].
5. **Cost Optimization:** Centers on delivering business value at the lowest price point by avoiding unnecessary expenses and maximizing resource utilization [5].
6. **Sustainability:** Aims to minimize the environmental impact of workloads by reducing energy consumption and leveraging shared resources in the cloud [6].

## How These Pillars Guided the Architectural Decisions

### 1. Operational Excellence

- **Implementation:**  
The architecture uses AWS CloudFormation to implement infrastructure as code, ensuring consistent deployments and enabling rapid recovery in case of failures. CloudWatch monitors system health and performance, while SNS sends notifications for proactive issue resolution.
- **Guidance:** Automating deployments and implementing monitoring ensures the system runs smoothly, reduces operational overhead, and supports continuous improvement.

### 2. Security

- **Implementation:**  
AWS Secrets Manager securely stores sensitive credentials like ChatGPT API keys, and IAM roles restrict access to AWS resources based on the principle

of least privilege. S3 buckets enforce access control policies, encrypt stored data, and restrict public access where necessary.

- **Guidance:** Protecting user data and API credentials is critical, as the portfolio generator handles personal information and interacts with third-party APIs.

### 3. Reliability

- **Implementation:**  
The architecture spans multiple availability zones (AZs) with redundant resources like NAT Gateways and ALB for fault tolerance. DynamoDB ensures high availability and data durability, while ECS tasks scale based on traffic load to maintain consistent performance.
- **Guidance:** Ensuring high availability and automatic recovery protects against service disruptions, meeting user expectations for a reliable system.

### 4. Performance Efficiency

- **Implementation:**  
AWS Fargate dynamically scales ECS tasks to handle varying workloads, ensuring optimal resource usage. CloudWatch metrics inform auto-scaling policies to adapt to demand changes, while S3's content delivery capabilities provide fast access to static portfolio webpages.
- **Guidance:** Efficient resource utilization and dynamic scaling ensure that user requests are processed promptly without over-provisioning.

### 5. Cost Optimization

- **Implementation:**  
S3 lifecycle policies automatically transition older data to lower-cost storage classes. DynamoDB uses on-demand pricing to scale costs with usage, and ECS Fargate removes the need for dedicated EC2 instances, charging only for actual usage.
- **Guidance:** Minimizing costs while meeting application needs ensures long-term viability, especially for a system with variable workloads.

### 6. Sustainability

- **Implementation:**  
By leveraging AWS's shared infrastructure, the architecture minimizes environmental impact compared to traditional data centers. S3's lifecycle

policies reduce storage energy use, and ECS Fargate avoids over-provisioning compute resources.

- **Guidance:** Efficient use of cloud resources aligns with sustainable computing practices, reducing the carbon footprint of the portfolio generator.

## Summary of AWS Well-Architected Framework Alignment

The architectural decisions for the portfolio generator align closely with the principles of the AWS Well-Architected Framework:

- **Automation:** Ensures operational excellence with CloudFormation and monitoring tools.
- **Security:** Protects sensitive user data and API credentials with Secrets Manager and access control policies.
- **Fault Tolerance:** Ensures reliability with redundant resources across multiple AZs.
- **Scalability:** Delivers performance efficiency with auto-scaling and serverless solutions.
- **Cost Management:** Optimizes costs with pay-as-you-go services and lifecycle policies.
- **Environmentally Friendly:** Promotes sustainability by leveraging AWS's energy-efficient infrastructure.



# Infrastructure as Code (IaC) Implementation

## Description of the Tools Used

AWS CloudFormation was used as the primary Infrastructure as Code (IaC) tool for this project. CloudFormation provides a declarative way to define and provision AWS resources through templates [10]. It simplifies infrastructure management by enabling automated, consistent, and repeatable deployments across environments.

## Explanation of the IaC Script and Its Functionality

The CloudFormation script provisions a complete infrastructure for the portfolio generator chatbot. Below is a detailed breakdown of each component and its role:

### 1. Networking Configuration

- **VPC (Virtual Private Cloud):**
  - Creates a logically isolated network (CIDR block 10.0.0.0/16) for the application.
  - Enables DNS support and hostnames to facilitate resource communication [2].
- **Subnets (Public and Private):**
  - Public subnets (10.0.1.0/24, 10.0.2.0/24, 10.0.3.0/24): Host internet-facing resources, including NAT Gateways and Application Load Balancer (ALB).
  - Private subnets (10.0.4.0/24, 10.0.5.0/24, 10.0.6.0/24): Host backend ECS tasks and database resources for added security.
  - Subnets are spread across three Availability Zones (AZs) to ensure high availability.
- **Internet Gateway and NAT Gateways:**
  - Internet Gateway enables public subnets to connect to the internet.
  - NAT Gateways in each AZ allow resources in private subnets to access external APIs securely, such as the ChatGPT API, without exposing them to public internet [7].

- **Route Tables and Associations:**

- Separate route tables for public and private subnets.
  - Public route tables direct internet-bound traffic through the Internet Gateway.
  - Private route tables route internet-bound traffic through respective NAT Gateways.
- 

## 2. Compute Resources

- **Amazon ECS Cluster:**

- Deploys an ECS cluster named PortfolioChatbotCluster for managing containerized workloads.
- This cluster hosts frontend (React) and backend (Node.js) services [3].

- **ECS Task Definitions:**

- **Frontend Task Definition:** Specifies container configurations for the React-based frontend. Includes:
  - CPU: 256 units and Memory: 512 MiB.
  - Container image: chatbot-frontend from Amazon ECR.
  - Port mapping: Maps container port 80 for HTTP traffic.
  - Logging: Configures CloudWatch logging for diagnostics.
- **Backend Task Definition:** Configures the backend (Node.js) container. Includes:
  - CPU: 256 units and Memory: 512 MiB.
  - Container image: chatbot-backend from Amazon ECR.
  - Port mapping: Maps container port 5000 for HTTP traffic.
  - Environment variables: Includes API URL for interaction with the frontend.
  - Logging: Configures CloudWatch logging for debugging [7].

- **ECS Services:**

- **Frontend Service:** Runs the frontend task on Fargate with path-based routing (/). Configured with:
    - Desired task count: 1.
    - Auto-scaling to handle traffic spikes.
    - Security group and private subnet for task isolation.
  - **Backend Service:** Runs the backend task with routing (/api/\*). Configured with:
    - Desired task count: 1.
    - Integrated with NAT Gateways for external API access.
    - Security group for backend service isolation.
- 

### 3. Storage

- **Amazon S3 Buckets:**

- Input bucket (portfolio-input-bucket): Stores uploaded resumes.
- Output bucket (portfolio-output-bucket): Hosts generated portfolio webpages with public access for static website hosting.
- Lifecycle rules reduce costs by transitioning data to infrequent access or archival storage after specified periods.
- Encryption ensures data protection [2].

- **Bucket Policy:**

- Configures public access to the output bucket, allowing generated portfolios to be accessed directly via a browser.
-

## 4. Load Balancing

- **Application Load Balancer (ALB):**
    - Distributes traffic between the frontend and backend services.
    - Uses listener rules to route / requests to the frontend and /api/\* requests to the backend [8].
    - ALB operates in the public subnets, ensuring external availability while keeping backend tasks in private subnets.
  - **Target Groups:**
    - Separate target groups for frontend and backend services ensure precise routing and health checks.
    - Health checks monitor endpoint availability, triggering service recovery if needed.
- 

## 5. Database

- **Amazon DynamoDB Tables:**
    - **Users Table:** Manages user profiles with attributes like id and email.
    - **Chats Table:** Tracks chat sessions using id and user\_id.
    - **Messages Table:** Stores individual chat messages, indexed by chat\_id.
    - Global secondary indexes enable efficient queries for session and user-related data.
    - Billing mode is set to on-demand for cost efficiency [5].
- 

## 6. Security

- **AWS Secrets Manager:**
  - Securely stores sensitive credentials, such as ChatGPT API keys.
  - Integrated with backend services to retrieve keys without exposing them in code [6].

- **IAM Roles and Policies:**
    - Assign least-privilege roles to ECS tasks and other AWS services, minimizing security risks.
  - **Security Groups:**
    - Frontend and backend services have distinct security groups, controlling access based on port and source.
- 

## 7. Monitoring and Notifications

- **Amazon CloudWatch Alarms:**
    - Monitors CPU and memory usage of ECS tasks.
    - Triggers auto-scaling or notifications upon breaching thresholds.
  - **Amazon Simple Notification Service (SNS):**
    - Sends email notifications to administrators for critical events, such as high resource utilization [9].
- 

## 8. Management and Governance

- **AWS CloudFormation Outputs:**
  - Provides key resource details, such as VPC ID, Subnet IDs, ALB DNS, and S3 bucket names.
  - Simplifies integration with other tools and troubleshooting.

## Challenges Faced and Solutions Implemented

### 1. Challenge: Multi-AZ High Availability

- **Issue:** Ensuring the system remains functional in case of an AZ failure.
- **Solution:** Distributed subnets, NAT Gateways, and ECS services across three AZs to maintain availability.

### 2. Challenge: Security of Sensitive Data

- **Issue:** Protecting API keys and sensitive configurations.
- **Solution:** Used AWS Secrets Manager for secure and dynamic credential management.

### 3. Challenge: Efficient Resource Scaling

- **Issue:** Handling traffic spikes while optimizing costs.
- **Solution:** Configured auto-scaling for ECS services based on CloudWatch metrics.

### 4. Challenge: Complex Resource Interdependencies

- **Issue:** Dependencies between resources (e.g., NAT Gateways, ALB, and ECS services).
- **Solution:** Leveraged CloudFormation's dependency management features to ensure correct provisioning order.

### 5. Challenge: Debugging Infrastructure Issues

- **Issue:** Identifying and resolving issues in complex deployments.
- **Solution:** Enabled detailed logging with CloudWatch for ECS tasks, ALB access, and system metrics.

# Deployment on AWS

## Description of the Deployment Process

The deployment process for the portfolio generator chatbot was automated and executed using AWS CloudFormation templates. Below is a detailed step-by-step outline:

### 1. CloudFormation Template Preparation

- A CloudFormation script defining all necessary AWS resources (networking, compute, storage, and security) was written and validated [10].

### 2. Resource Deployment

- The template was deployed in the AWS Learner Lab environment using the AWS Management Console. The CloudFormation stack provisioned:
  - Networking components (VPC, subnets, NAT Gateways, and route tables) [7].
  - Compute resources (ECS cluster, task definitions, and services for frontend and backend) [1].
  - Storage resources (S3 buckets for input resumes and generated portfolios) [2].
  - Security resources (IAM roles, Security Groups, and Secrets Manager) [6].

### 3. Docker Image Management

- Docker images for the frontend and backend were built locally and pushed to Amazon Elastic Container Registry (ECR). These images were specified in ECS task definitions.

### 4. ECS Service Configuration

- ECS services were configured to use Fargate as the launch type. The Application Load Balancer (ALB) was integrated with ECS services for traffic routing:
  - Path / routed to the frontend.
  - Path /api/\* routed to the backend [8].

### 5. Testing and Validation

- The ALB DNS name was used to test the application. Both the frontend interface and backend API endpoints were verified for functionality.

---

## Restrictions and Challenges Due to Service Limitations

### 1. Unavailability of HTTPS

- **Issue:** Services like AWS Certificate Manager, Route 53, and CloudFront were not available in the Learner Lab environment, preventing the use of HTTPS for secure communication.
- **Mitigation:** The application was deployed using an HTTP endpoint through the ALB. Security measures, such as restricted access to backend services via Security Groups, were implemented.

### 2. Limited IAM Policy Customization

- **Issue:** Restricted permissions in the Learner Lab environment limited the ability to customize IAM policies fully.
  - **Mitigation:** Default permissions provided by the Lab were used, ensuring the application adhered to the principle of least privilege.
-



## Hosted Application's Public URL

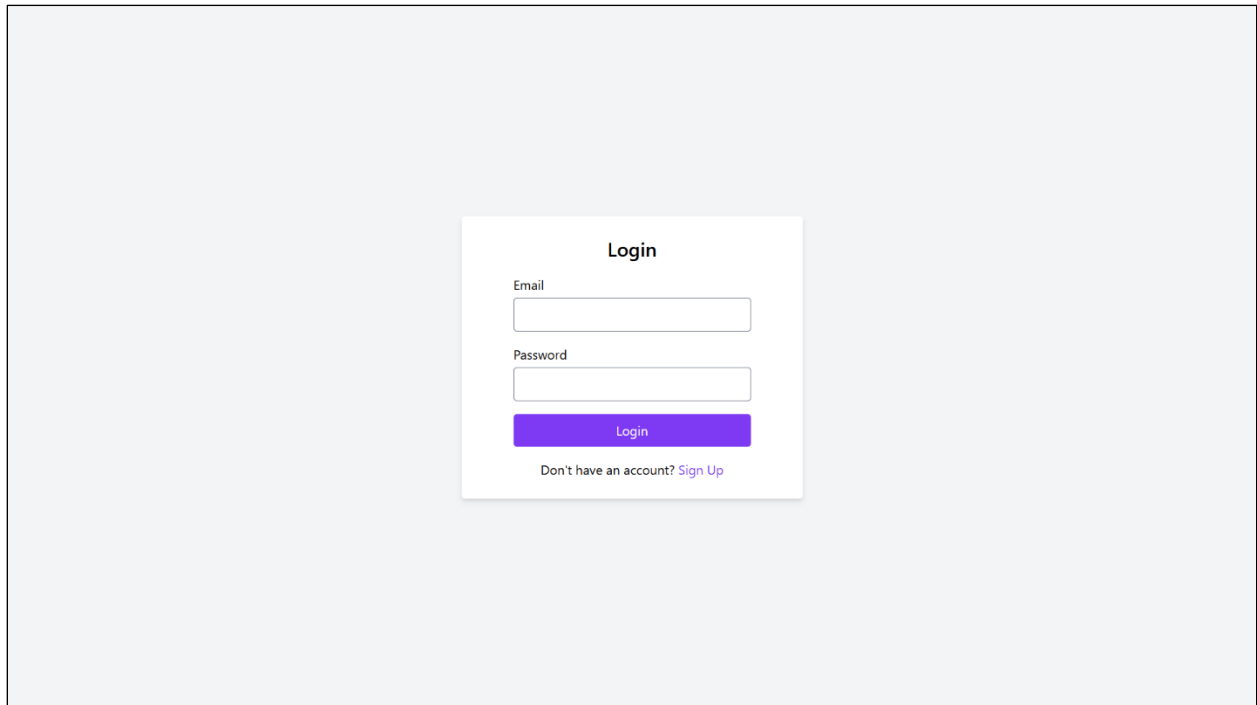
- **Public URL:**

The hosted application is accessible via the ALB DNS name:

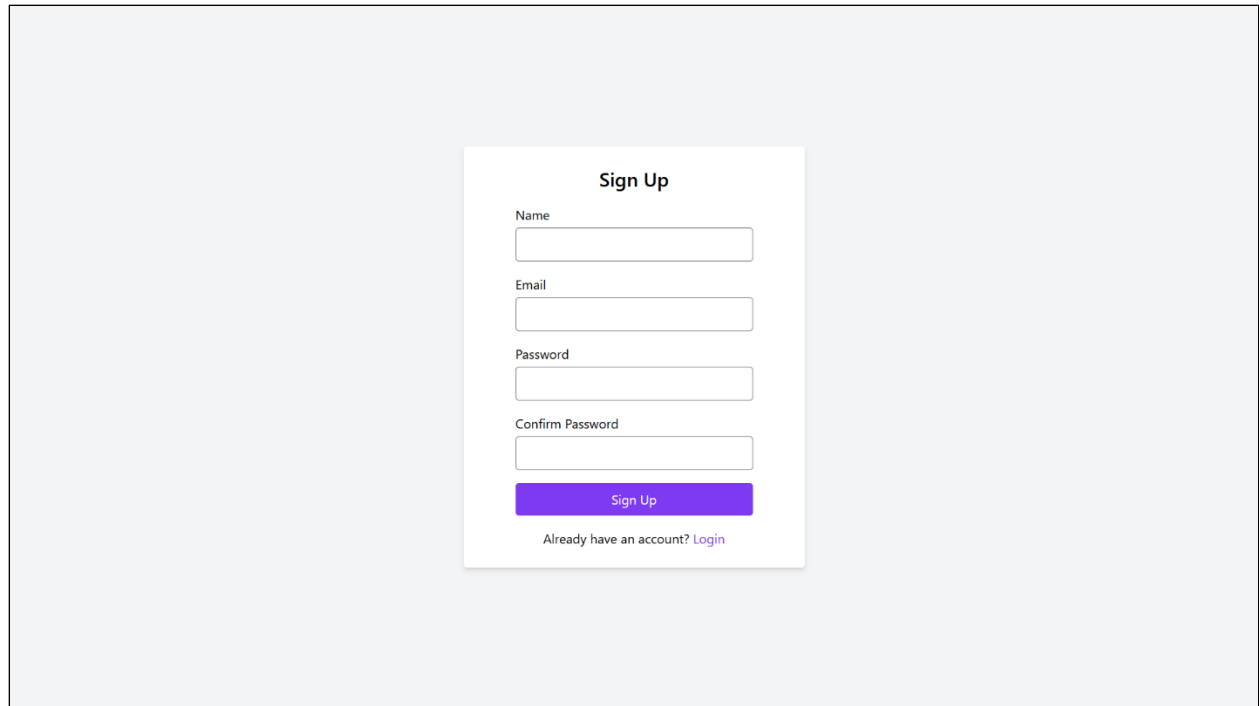
<http://portfoliochatbotalb-2070134174.us-east-1.elb.amazonaws.com/>

## Evidence of Deployment:

### Frontend Interface:

A screenshot of a web application's login page. The page has a light gray background. In the center, there is a white rectangular box with a thin gray border. Inside this box, the word "Login" is centered at the top in a bold, black font. Below it, there are two input fields: the first is labeled "Email" and the second is labeled "Password". Both labels are in a small, gray font. Below the "Password" field is a solid blue button with the word "Login" in white text. At the bottom of the white box, there is a link that says "Don't have an account? Sign Up" in a small, gray font, where "Sign Up" is a blue hyperlink.

*Figure 2: Login Page*

A sign-up form titled "Sign Up" is centered on a light gray background. The form is a white card with a subtle shadow. It contains four input fields: "Name", "Email", "Password", and "Confirm Password". Below the "Confirm Password" field is a purple "Sign Up" button. At the bottom of the card, there is a link that says "Already have an account? Login".

**Sign Up**

Name

Email

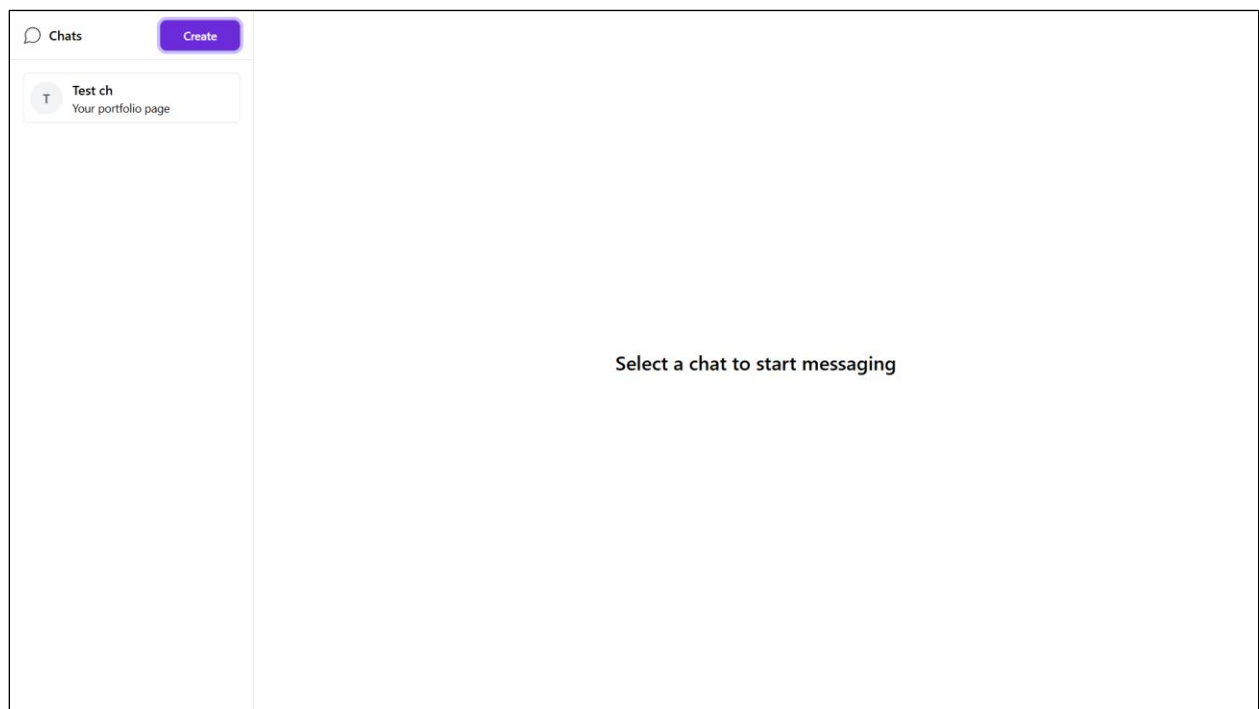
Password

Confirm Password

[Sign Up](#)

Already have an account? [Login](#)

Figure 3: Signup Page

A chat interface with a sidebar on the left and a main chat area on the right. The sidebar has a "Chats" header with a speech bubble icon and a purple "Create" button. Below this is a list of chat items; the first one is "Test ch" with a circular icon containing the letter "T" and the subtitle "Your portfolio page". The main chat area is empty and contains the text "Select a chat to start messaging".

Chats [Create](#)

T Test ch  
Your portfolio page

Select a chat to start messaging

Figure 4: Chat Page

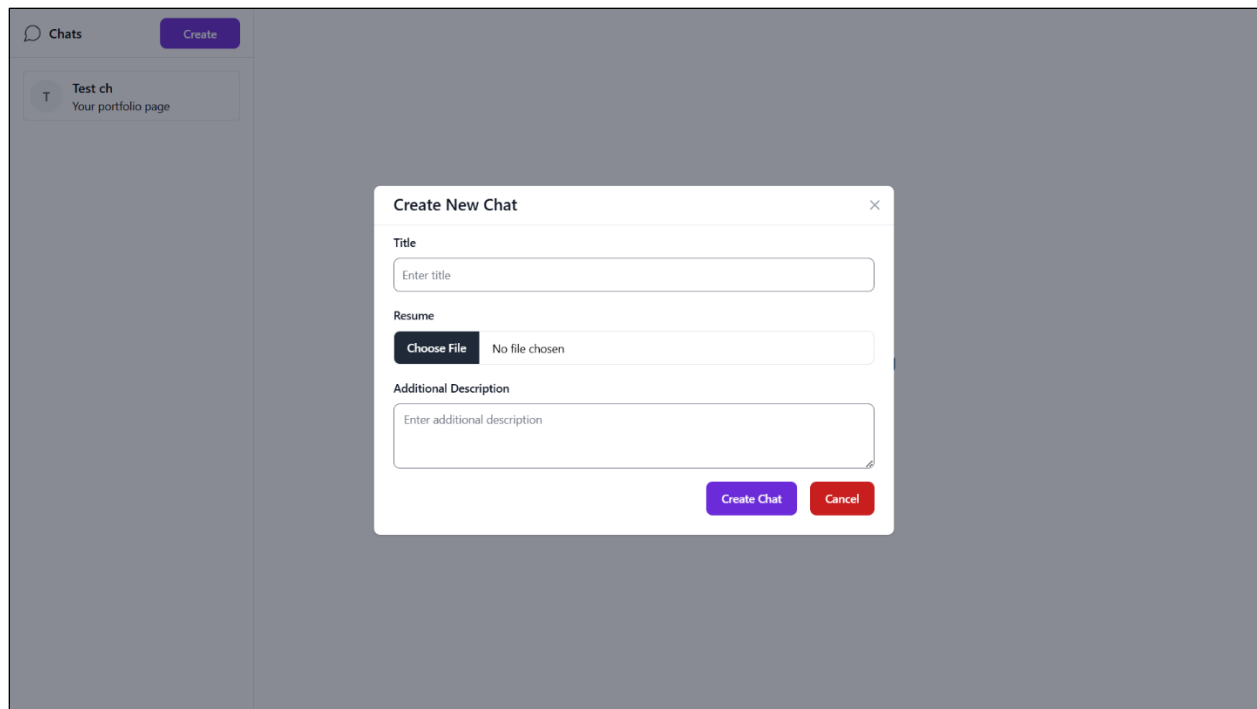


Figure 5: Create Chat

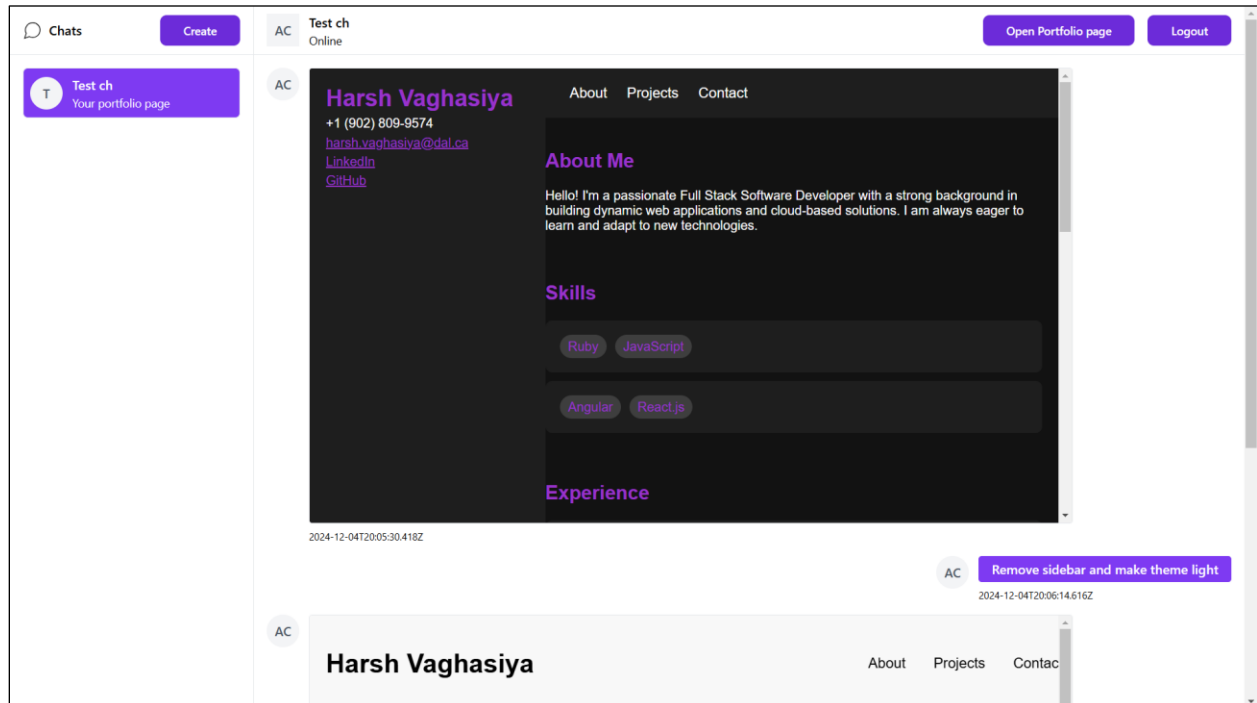


Figure 6: Generated Portfolio page chat

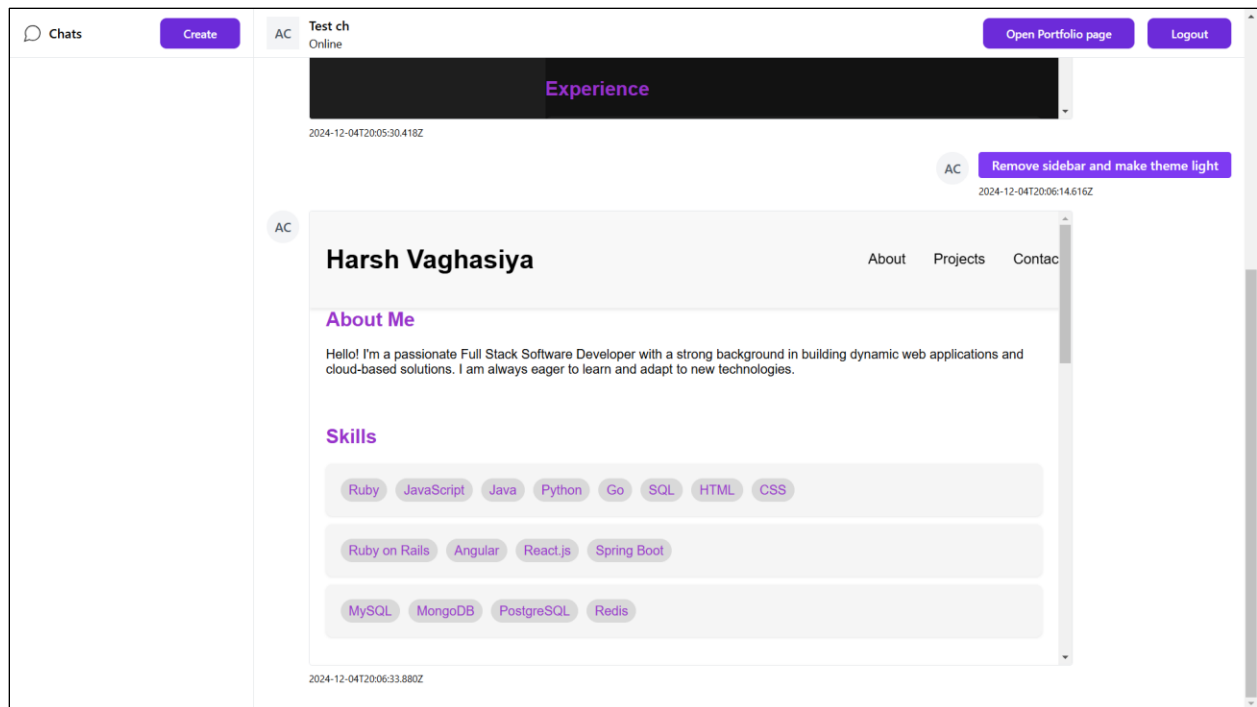


Figure 7: Generated Portfolio page chat

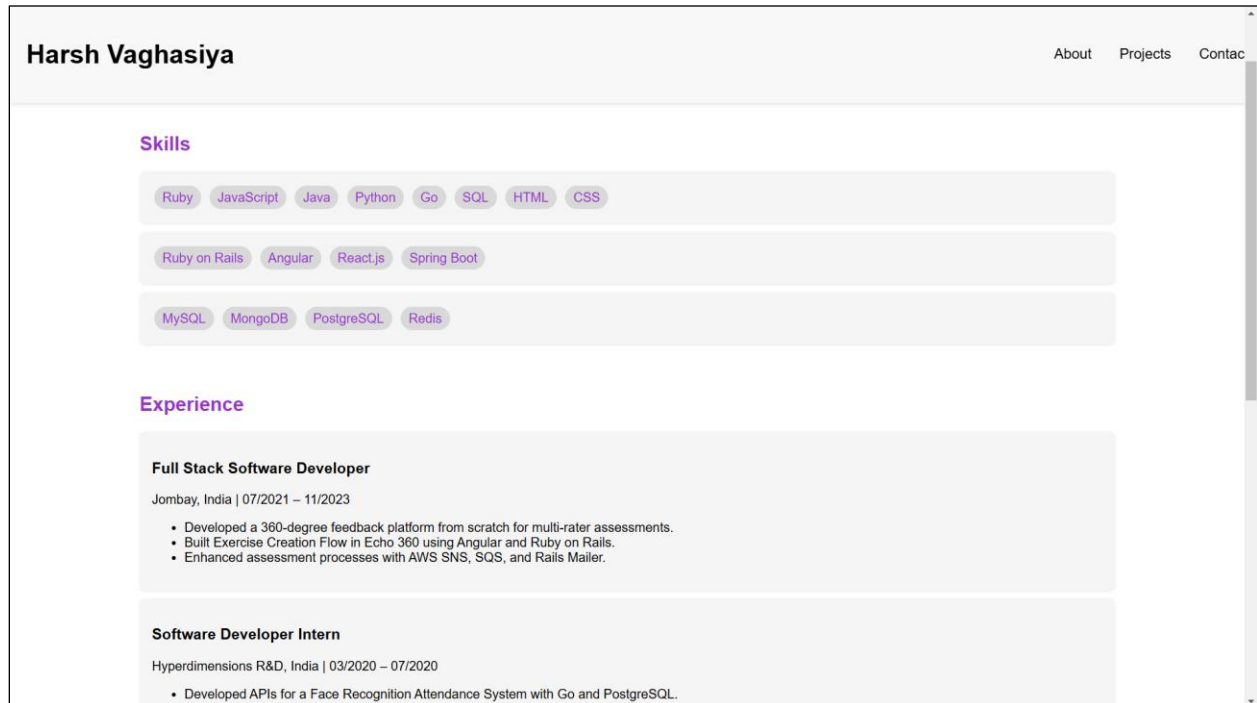


Figure 8: Hosted Portfolio page

## VPC:

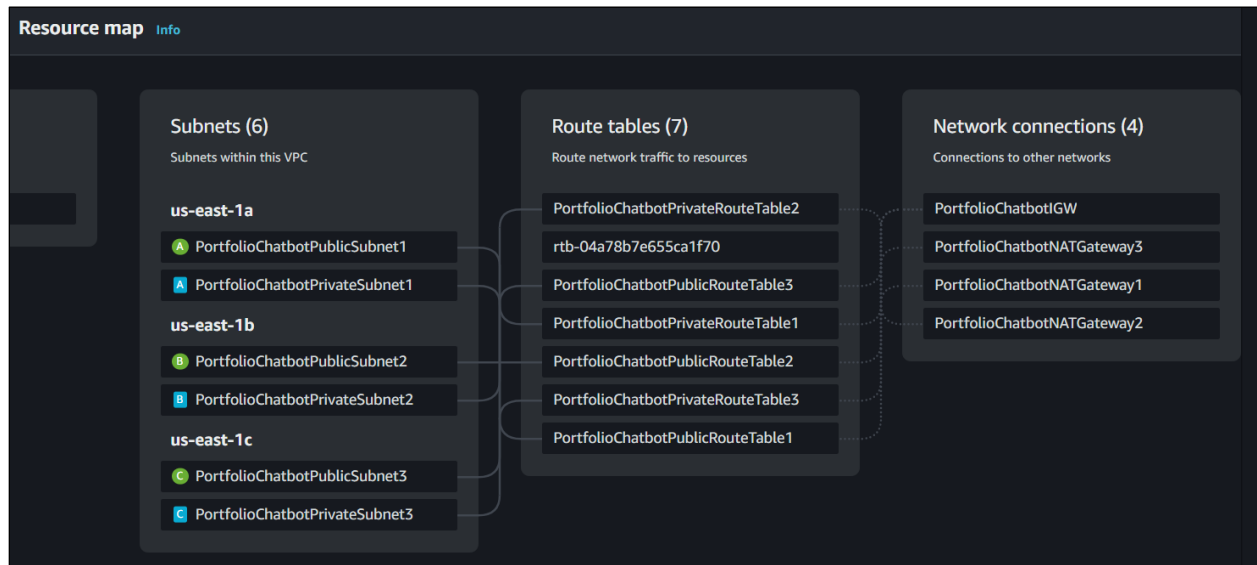


Figure 9: VPC Resource map

## Load Balancer:

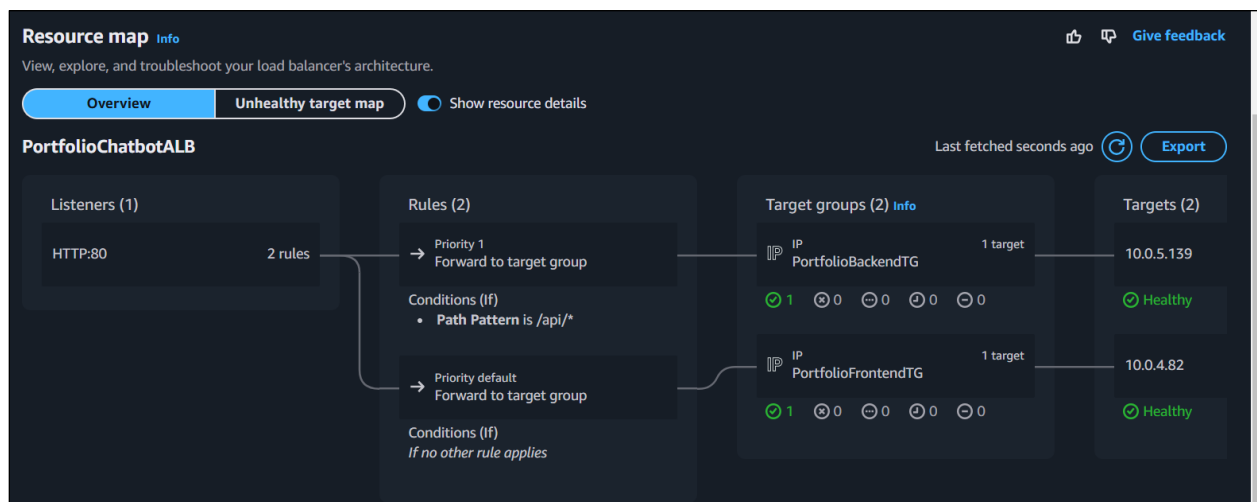


Figure 1: Application Load Balancer Resource map

## ECS:

The screenshot displays the AWS Management Console for the **PortfolioChatbotCluster** ECS cluster. The left sidebar shows the navigation menu for Amazon Elastic Container Service, including Clusters, Namespaces, Task definitions, Account settings, and various AWS services like ECR, ECR Repositories, and AWS Batch.

The main content area shows the **PortfolioChatbotCluster** overview. The cluster is **Active** and has been last updated on December 04, 2024, at 22:30 (UTC-4:00). It features CloudWatch monitoring (Default) and 2 registered container instances. The cluster is using Fargate ephemeral storage for managed storage.

Below the overview, the **Services** tab is selected, showing a list of 2 services. The services are **PortfolioBackendService** and **PortfolioFrontendService**, both in an **Active** state. Each service has 1/1 tasks running and a completed deployment.

| Service name                             | ARN   | Status | Service type | Deployments and tasks | Last deploy... |
|--|---|--------|--------------|-----------------------|----------------|
| <a href="#">PortfolioBackendService</a>  | arn:aws:ecs:us-east-1:333267225604:cluster/PortfolioChatbotCluster/service/PortfolioBackendService  | Active | REPLICA      | 1/1 Tasks running     | Completed      |
| <a href="#">PortfolioFrontendService</a> | arn:aws:ecs:us-east-1:333267225604:cluster/PortfolioChatbotCluster/service/PortfolioFrontendService | Active | REPLICA      | 1/1 Tasks running     | Completed      |

Figure 1: ECS services

# Cost Estimation

## Detailed Cost Breakdown

Below is the estimated cost breakdown for each AWS service used in the portfolio generator chatbot project. These estimates are based on typical usage patterns and AWS pricing for the us-east-1 region.

---

### 1. Compute Resources

- **AWS Fargate**
  - **Frontend Service:**
    - **Usage:** 1 task, 256 CPU units, 512 MiB memory, running continuously (30 days).
    - **Cost Calculation:**
      - vCPU:  $0.25 \text{ vCPU} \times \$0.04048 \text{ per vCPU/hour} = \$0.01012/\text{hour}$ .
      - Memory:  $0.5 \text{ GiB} \times \$0.004445 \text{ per GiB/hour} = \$0.002222/\text{hour}$ .
      - Total per hour:  $\$0.012342/\text{hour}$ .
      - Monthly Cost:  $\$0.012342/\text{hour} \times 730 \text{ hours} = \mathbf{\$9.00}$ .
  - **Backend Service:**
    - Same configuration and usage as the frontend.
    - Monthly Cost: **\$9.00**.

**Total Cost for Compute (Fargate): \$18.00/month.**

---

### 2. Storage

- **Amazon S3**
  - **Input Bucket:**
    - **Usage:** 10 GB stored per month with 1,000 PUT and GET requests.
    - **Cost Calculation:**
      - Storage:  $10 \text{ GB} \times \$0.023/\text{GB} = \$0.23$ .

- PUT, POST, and LIST requests:  $1,000 \times \$0.005/1,000 \text{ requests} = \$0.005$ .
- GET requests:  $1,000 \times \$0.0004/1,000 \text{ requests} = \$0.0004$ .
- Total Cost: **\$0.24/month**.
- **Output Bucket:**
  - **Usage:** 20 GB stored per month with 5,000 GET requests.
  - **Cost Calculation:**
    - Storage:  $20 \text{ GB} \times \$0.023/\text{GB} = \$0.46$ .
    - GET requests:  $5,000 \times \$0.0004/1,000 \text{ requests} = \$0.002$ .
    - Total Cost: **\$0.46/month**.

**Total Cost for Storage (S3): \$0.70/month.**

---

### 3. Networking

- **Amazon VPC (NAT Gateway)**
  - **Usage:** 3 NAT Gateways for redundancy, each processing 50 GB data per month.
  - **Cost Calculation:**
    - NAT Gateway hourly charge:  $3 \times \$0.045/\text{hour} = \$3.24/\text{day}$ .
    - Data processing:  $50 \text{ GB} \times \$0.045/\text{GB} = \$2.25$ .
    - Total Monthly Cost:  $(\$3.24/\text{day} \times 30) + (\$2.25 \times 3) = \mathbf{\$102.15/month}$ .
- **Elastic Load Balancer (ALB)**
  - **Usage:** 1 ALB with 1,000 hours/month and 20 GB data processed.
  - **Cost Calculation:**
    - ALB hourly charge:  $\$0.0225/\text{hour} \times 730 \text{ hours} = \$16.43$ .
    - Data processed:  $20 \text{ GB} \times \$0.008/\text{GB} = \$0.16$ .
    - Total Cost: **\$16.59/month**.

**Total Cost for Networking: \$118.74/month.**

---



## 4. Database

- **Amazon DynamoDB**

- **Usage:** 1 GB storage, 10,000 read requests, 5,000 write requests per month.
  - **Cost Calculation:**
    - Storage:  $1 \text{ GB} \times \$0.25/\text{GB} = \$0.25$ .
    - Read requests:  $10,000 \times \$0.00013/\text{read} = \$1.30$ .
    - Write requests:  $5,000 \times \$0.00065/\text{write} = \$3.25$ .
    - Total Cost: **\$4.80/month.**
- 

## 5. Monitoring and Notifications

- **Amazon CloudWatch**

- **Usage:** 10 custom metrics and 1,000 logs ingested per month.
- **Cost Calculation:**
  - Metrics:  $10 \text{ metrics} \times \$0.30/\text{metric} = \$3.00$ .
  - Logs:  $1,000 \text{ logs} \times \$0.50/\text{GB} = \$0.50$ .
  - Total Cost: **\$3.50/month.**

- **Amazon SNS**

- **Usage:** 100 notifications sent per month.
- **Cost Calculation:**
  - Notifications:  $100 \times \$0.50/100,000 = \$0.0005$ .
  - Total Cost: **\$0.01/month.**

**Total Cost for Monitoring and Notifications: \$3.51/month.**

---

## 6. Security

- **AWS Secrets Manager**

- **Usage:** 5 secrets stored.
- **Cost Calculation:**

- $\$0.40/\text{secret} \times 5 = \text{\$2.00/month.}$

## Overall Estimated Monthly Cost

| Service                      | Cost<br>(USD/month) |
|------------------------------|---------------------|
| Compute (Fargate)            | \$18.00             |
| Storage (S3)                 | \$0.70              |
| Networking (VPC + ALB)       | \$118.74            |
| Database (DynamoDB)          | \$4.80              |
| Monitoring and Notifications | \$3.51              |
| Security (Secrets Manager)   | \$2.00              |
| <b>Total</b>                 | <b>\$147.75</b>     |

# Challenges and Solutions

## Challenges Encountered During Architecture Design and Deployment

### 1. Challenge: Limited Access to HTTPS Configuration

- **Issue:** The AWS Learner Lab environment lacks access to certain services like AWS Certificate Manager, Route 53, and CloudFront, making it impossible to configure HTTPS for the application.
- **Solution:** Used HTTP via the Application Load Balancer (ALB) and implemented security best practices at the network level, such as restricting backend services to private subnets and using secure secrets management [8].

### 2. Challenge: Resource Limitations in Learner Lab

- **Issue:** Limited quotas for services, such as compute and storage, constrained the deployment of scalable resources.
- **Solution:** Optimized ECS task definitions by using lightweight container images and minimizing resource allocation for Fargate tasks [1].

### 3. Challenge: Multi-AZ Networking Complexity

- **Issue:** Configuring a reliable networking setup with NAT Gateways and subnets across three Availability Zones added complexity to the architecture.
- **Solution:** Deployed NAT Gateways and route tables specific to each AZ, ensuring seamless communication for backend services while adhering to high availability principles [7].

### 4. Challenge: Secure API Key Management

- **Issue:** Sensitive credentials like ChatGPT API keys needed to be securely managed without hardcoding them into the application.
- **Solution:** Used AWS Secrets Manager to securely store and access sensitive information dynamically during runtime [6].

### 5. Challenge: Load Balancing Path-Based Routing

- **Issue:** Configuring ALB listener rules for path-based routing to direct / to the frontend and /api/\* to the backend required precise definitions.
- **Solution:** Defined target groups and listener rules in the CloudFormation template to ensure proper routing [8].

## 6. Challenge: Debugging Deployment Failures

- **Issue:** Debugging issues during initial deployments, such as misconfigured ECS task definitions and security group rules, delayed progress.
- **Solution:** Enabled detailed logging with Amazon CloudWatch and reviewed logs for ECS tasks, ALB access, and API calls to identify and fix errors [4].

## 7. Challenge: Lack of Advanced Monitoring Tools

- **Issue:** Advanced monitoring tools were not available in the Learner Lab environment, limiting detailed insights into system performance.
- **Solution:** Set up CloudWatch alarms for resource usage and configured Amazon SNS to send alerts for high CPU or memory utilization [4][9].

## 8. Challenge: Manual Regional Configuration

- **Issue:** Services in the Learner Lab were restricted to the us-east-1 region, requiring manual adjustments to the CloudFormation template.
- **Solution:** Updated all region-specific configurations in the template and ensured compatibility for us-east-1 deployment [10].

# Conclusion

## Summary of the Project Outcome

The portfolio generator chatbot project successfully achieved its objectives by automating the creation of personalized portfolio webpages using user-uploaded resumes and ChatGPT API. The application was deployed on AWS using a well-architected infrastructure that leveraged various AWS services, including Amazon ECS, S3, DynamoDB, Secrets Manager, and CloudFormation.

Despite limitations in the AWS Learner Lab, such as the unavailability of HTTPS and resource constraints, the application was functional and demonstrated core capabilities, including secure data handling, reliable service delivery, and effective resource utilization. The public-facing application was hosted using an Application Load Balancer (ALB) and provided users with a seamless interface to upload resumes and generate portfolio webpages.

## Reflection on Learning and Skills Developed During the Project

This project provided valuable learning opportunities and allowed the development of key technical and professional skills, including:

### 1. Cloud Infrastructure Design and Implementation

- Gained hands-on experience in designing scalable, secure, and cost-effective cloud architectures.
- Learned how to use AWS services, such as ECS for container orchestration, DynamoDB for NoSQL database needs, and CloudFormation for IaC [1][4].

### 2. Infrastructure as Code (IaC)

- Developed skills in writing and managing CloudFormation templates to provision and maintain AWS resources.
- Learned to handle resource interdependencies and automate deployment processes efficiently [10].

### 3. Networking and Security

- Understood the principles of VPC configuration, NAT Gateway deployment, and securing resources through IAM roles and Security Groups.
- Implemented secure API key management using AWS Secrets Manager [6][7].

#### **4. Application Development and Containerization**

- Improved proficiency in containerizing applications using Docker and deploying them on Amazon ECS with Fargate.
- Worked with Amazon Elastic Container Registry (ECR) to manage container images [1].

#### **5. Monitoring and Troubleshooting**

- Gained experience in setting up CloudWatch for monitoring resource utilization and debugging deployment issues.
- Configured SNS notifications to streamline alerting and issue response workflows [4][9].

#### **6. Problem-Solving and Adaptability**

- Adapted to constraints in the Learner Lab environment, such as the lack of HTTPS and limited service quotas.
- Developed solutions to maintain functionality despite these limitations, demonstrating problem-solving and resilience.

#### **7. Collaboration and Documentation**

- Enhanced skills in documenting project architecture, deployment processes, and technical decisions.
- Communicated challenges and solutions effectively in a professional context.

## References

- [1] "AWS Fargate - Serverless Compute for Containers," aws.amazon.com [Online]. Available: <https://aws.amazon.com/fargate/>. [Accessed: November 30, 2024].
- [2] "Amazon S3 Storage," aws.amazon.com [Online]. Available: <https://aws.amazon.com/s3/>. [Accessed: December 1, 2024].
- [3] "AWS Lambda," aws.amazon.com [Online]. Available: <https://aws.amazon.com/lambda/>. [Accessed: December 2, 2024].
- [4] "Amazon CloudWatch," aws.amazon.com [Online]. Available: <https://aws.amazon.com/cloudwatch/>. [Accessed: December 3, 2024].
- [5] "Amazon DynamoDB," aws.amazon.com [Online]. Available: <https://aws.amazon.com/dynamodb/>. [Accessed: December 4, 2024].
- [6] "AWS Secrets Manager," aws.amazon.com [Online]. Available: <https://aws.amazon.com/secrets-manager/>. [Accessed: November 30, 2024].
- [7] "Amazon VPC," aws.amazon.com [Online]. Available: <https://aws.amazon.com/vpc/>. [Accessed: December 1, 2024].
- [8] "Elastic Load Balancing," aws.amazon.com [Online]. Available: <https://aws.amazon.com/elasticloadbalancing/>. [Accessed: December 2, 2024].
- [9] "Amazon Simple Notification Service (SNS)," aws.amazon.com [Online]. Available: <https://aws.amazon.com/sns/>. [Accessed: December 3, 2024].
- [10] "AWS CloudFormation," aws.amazon.com [Online]. Available: <https://aws.amazon.com/cloudformation/>. [Accessed: December 4, 2024].