

# Cloud-Based Web Application Deployment on AWS Infrastructure

Salman Ahmad (21i-0469)

Syed Haider Naqvi (20i-0816)

May 11, 2025

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                        | <b>2</b>  |
| 1.1      | Project Overview . . . . .                 | 2         |
| 1.2      | Live Demo URLs . . . . .                   | 2         |
| 1.3      | GitHub Repository . . . . .                | 2         |
| <b>2</b> | <b>Architecture Diagram</b>                | <b>3</b>  |
| 2.1      | Architecture Components . . . . .          | 3         |
| 2.2      | Network Flow . . . . .                     | 3         |
| <b>3</b> | <b>Implementation Details</b>              | <b>4</b>  |
| 3.1      | VPC Configuration . . . . .                | 4         |
| 3.2      | Backend Implementation . . . . .           | 4         |
| 3.3      | Frontend Implementation . . . . .          | 4         |
| 3.4      | Database Schema . . . . .                  | 4         |
| 3.5      | AWS S3 Integration . . . . .               | 4         |
| <b>4</b> | <b>Security Measures</b>                   | <b>5</b>  |
| 4.1      | IAM Roles and Policies . . . . .           | 5         |
| 4.2      | Security Groups . . . . .                  | 5         |
| 4.3      | Data Protection . . . . .                  | 5         |
| <b>5</b> | <b>Deployment Guide</b>                    | <b>5</b>  |
| 5.1      | Prerequisites . . . . .                    | 5         |
| 5.2      | VPC and Network Setup . . . . .            | 5         |
| 5.3      | Backend Deployment . . . . .               | 6         |
| 5.3.1    | Setting up RDS Database . . . . .          | 6         |
| 5.3.2    | Setting up S3 Bucket . . . . .             | 6         |
| 5.3.3    | Deploying Backend to EC2 . . . . .         | 6         |
| 5.4      | Frontend Deployment . . . . .              | 6         |
| <b>6</b> | <b>Screenshots and Evidence</b>            | <b>7</b>  |
| 6.1      | IAM Role Creation and Attachment . . . . . | 7         |
| 6.2      | IAM Policies . . . . .                     | 7         |
| 6.3      | Security Groups Configuration . . . . .    | 8         |
| 6.4      | EC2 Instance Deployment . . . . .          | 8         |
| 6.5      | RDS Database Configuration . . . . .       | 9         |
| 6.6      | S3 Bucket Configuration . . . . .          | 9         |
| 6.7      | Elastic Beanstalk Deployment . . . . .     | 10        |
| 6.8      | Application Screenshots . . . . .          | 10        |
| <b>7</b> | <b>Conclusion</b>                          | <b>11</b> |
| 7.1      | Future Improvements . . . . .              | 12        |

# 1 Introduction

This document details the implementation of a cloud-based web application deployed on AWS infrastructure. The application follows modern cloud-native architecture principles and consists of two separate components:

- A backend service that performs CRUD operations, handles user authentication, manages a PostgreSQL database, and supports secure file and image uploads to S3.
- A frontend client that consumes the backend via REST APIs, deployed on Elastic Beanstalk.

The project demonstrates the use of various AWS services including EC2, RDS, S3, Elastic Beanstalk, VPC, and IAM to create a secure and scalable application architecture.

## 1.1 Project Overview

- Application Type: Task Management System
- Backend Technologies: Node.js, Express, Sequelize ORM
- Frontend Technologies: React, Bootstrap, Axios
- Database: PostgreSQL on Amazon RDS
- Storage: Amazon S3 for file uploads
- Deployment: Backend on EC2 with Docker, Frontend on Elastic Beanstalk

## 1.2 Live Demo URLs

- Frontend: <http://taskapp-frontend-prod-env.eba-vupzncxu.ap-southeast-1.elasticbeanstalk.com/>
- Backend API: <http://122.248.202.13:5000/api>

## 1.3 GitHub Repository

- Repository Link: <https://github.com/YourUsername/cloud-task-manager>

## 2 Architecture Diagram

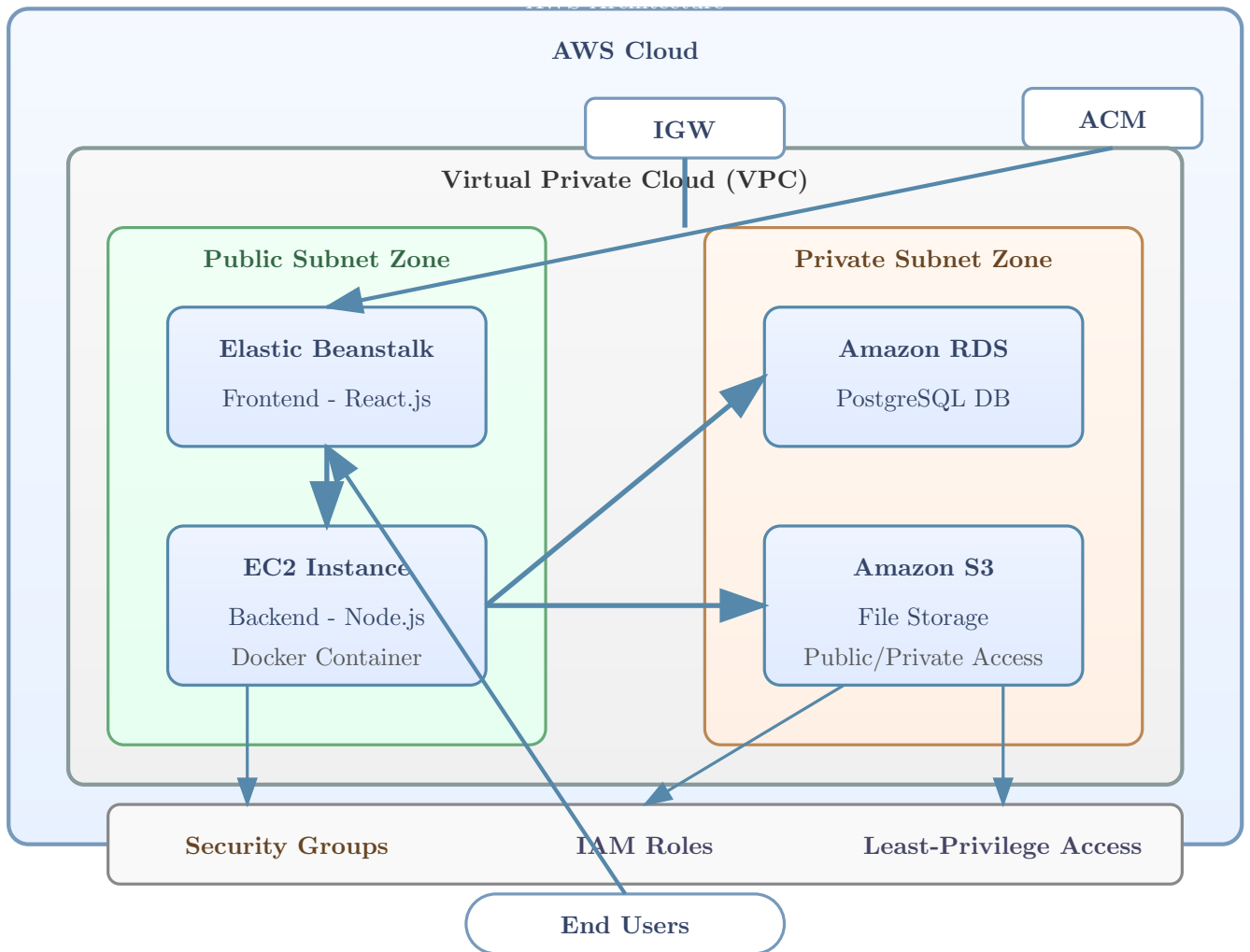


Figure 1: A diagram imported directly from SVG.

### 2.1 Architecture Components

- **VPC Configuration:** Custom VPC with public and private subnets across multiple Availability Zones for enhanced security and availability.
- **Frontend:** React application deployed on Elastic Beanstalk, providing a scalable and managed environment.
- **Backend:** Node.js/Express API deployed on an EC2 instance within a public subnet, using Docker for containerization.
- **Database:** PostgreSQL database hosted on Amazon RDS in a private subnet for enhanced security.
- **Storage:** Amazon S3 bucket configured with appropriate permissions for storing user file uploads.
- **Security:** IAM roles and policies implementing the principle of least privilege, security groups restricting traffic, and HTTPS for secure communication.

### 2.2 Network Flow

1. Users access the frontend application hosted on Elastic Beanstalk.

2. The frontend makes API requests to the backend service running on EC2.
3. The backend authenticates requests, processes data, interacts with the database, and manages file operations with S3.
4. Backend responses are returned to the frontend for display to the user.

## **3 Implementation Details**

### **3.1 VPC Configuration**

- Custom VPC (CIDR: 172.31.0.0/16)
- Public Subnets: 172.31.1.0/24, 172.31.2.0/24
- Internet Gateway for external connectivity
- Route Tables configured for proper traffic routing

### **3.2 Backend Implementation**

- Node.js/Express RESTful API
- JWT-based authentication system
- Sequelize ORM for database interactions
- Integration with AWS S3 for file storage
- Docker containerization for consistent deployment

### **3.3 Frontend Implementation**

- React single-page application
- Bootstrap for responsive design
- Axios for API communication
- React Router for client-side routing
- Context API for state management

### **3.4 Database Schema**

- Users Table: Stores user credentials and profile information
- Tasks Table: Stores task information with foreign key relationship to users

### **3.5 AWS S3 Integration**

- Bucket configured with appropriate CORS settings
- Secure file upload mechanism
- Direct file access URLs for authenticated users

## 4 Security Measures

### 4.1 IAM Roles and Policies

- **EC2 Instance Role (TaskApp-EC2-Role):** Provides permissions for the EC2 instance to access S3 and RDS.
- **Elastic Beanstalk Role (TaskApp-EB-Role):** Grants necessary permissions for Elastic Beanstalk to manage resources.
- Principle of least privilege implemented across all roles

### 4.2 Security Groups

- **EC2 Security Group:**
  - Inbound: Allow HTTP (80), HTTPS (443), and SSH (22) from specific IPs
  - Outbound: Allow all traffic
- **RDS Security Group:**
  - Inbound: Allow PostgreSQL (5432) traffic only from EC2 security group
  - Outbound: Allow all traffic
- **Elastic Beanstalk Security Group:**
  - Inbound: Allow HTTP (80) and HTTPS (443) from anywhere
  - Outbound: Allow all traffic

### 4.3 Data Protection

- Password hashing using bcrypt
- JWT-based authentication with expiration
- HTTPS for secure data transmission
- Database in private subnet, inaccessible from the internet

## 5 Deployment Guide

### 5.1 Prerequisites

- AWS Account with access to required services
- AWS CLI configured with appropriate credentials
- Node.js and npm installed locally
- Docker installed locally (for backend containerization)

### 5.2 VPC and Network Setup

1. Create a VPC with CIDR block 172.31.0.0/16
2. Create public subnets in different Availability Zones
3. Create an Internet Gateway and attach to VPC
4. Configure route tables to allow internet access

## **5.3 Backend Deployment**

### **5.3.1 Setting up RDS Database**

1. Create a DB subnet group with private subnets
2. Launch PostgreSQL RDS instance with appropriate settings
3. Configure security group to allow access only from backend

### **5.3.2 Setting up S3 Bucket**

1. Create an S3 bucket for file storage
2. Configure CORS settings to allow cross-origin requests
3. Set up bucket policy for appropriate access control

### **5.3.3 Deploying Backend to EC2**

1. Launch EC2 instance in public subnet
2. Attach IAM role with necessary permissions
3. Connect to instance using SSH
4. Install Docker and other dependencies
5. Clone repository and navigate to backend directory
6. Configure environment variables for database and S3 access
7. Build and run Docker container

## **5.4 Frontend Deployment**

1. Navigate to frontend directory locally
2. Configure API endpoint in environment variables
3. Build React application
4. Create necessary configuration files (.ebextensions, Procfile)
5. Create deployment package (ZIP file)
6. Create Elastic Beanstalk application and environment
7. Upload and deploy application code
8. Configure environment variables if needed

## 6 Screenshots and Evidence

### 6.1 IAM Role Creation and Attachment

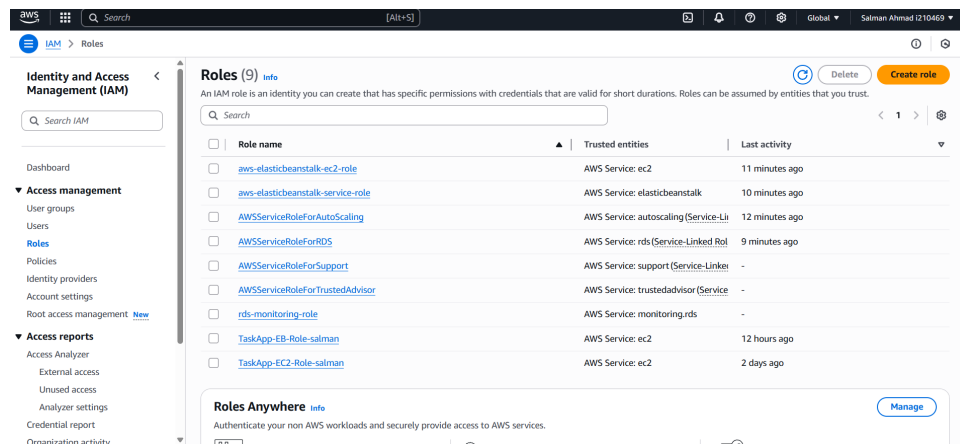


Figure 2: IAM Role Creation

### 6.2 IAM Policies

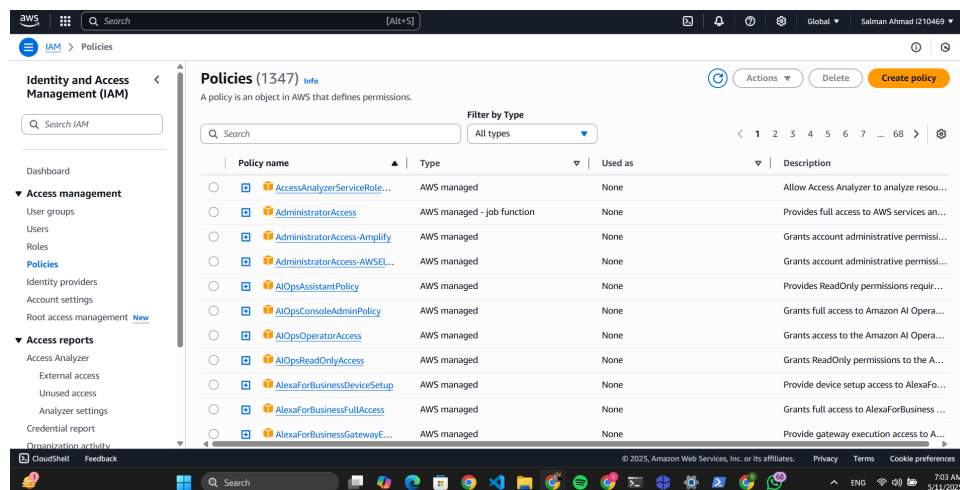


Figure 3: Configured IAM Policies

## 6.3 Security Groups Configuration

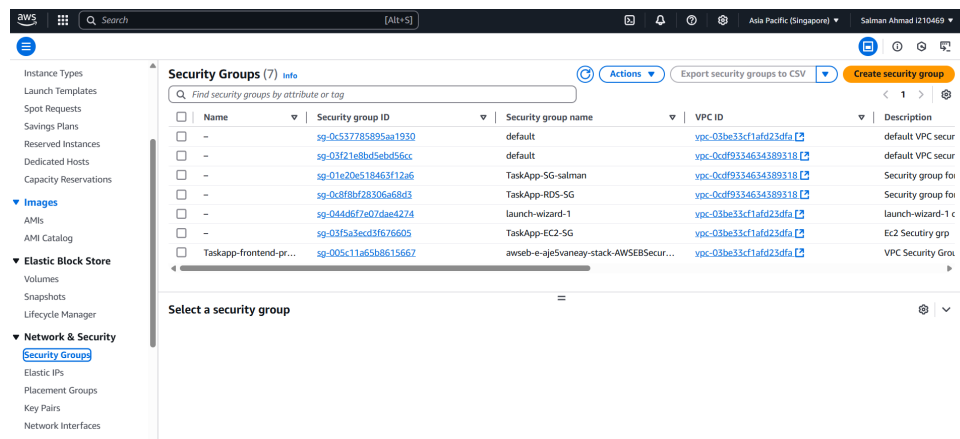


Figure 4: Security Groups Configuration

## 6.4 EC2 Instance Deployment

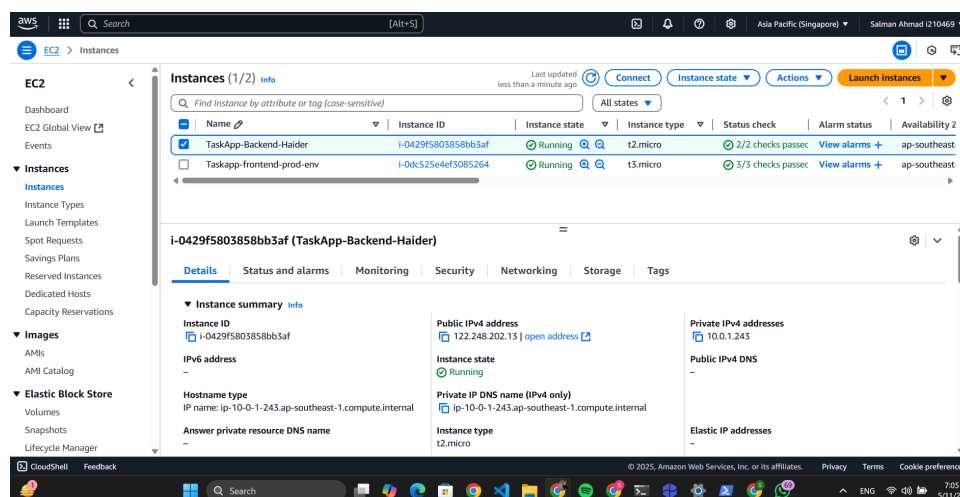


Figure 5: EC2 Instance for Backend



## 6.5 RDS Database Configuration

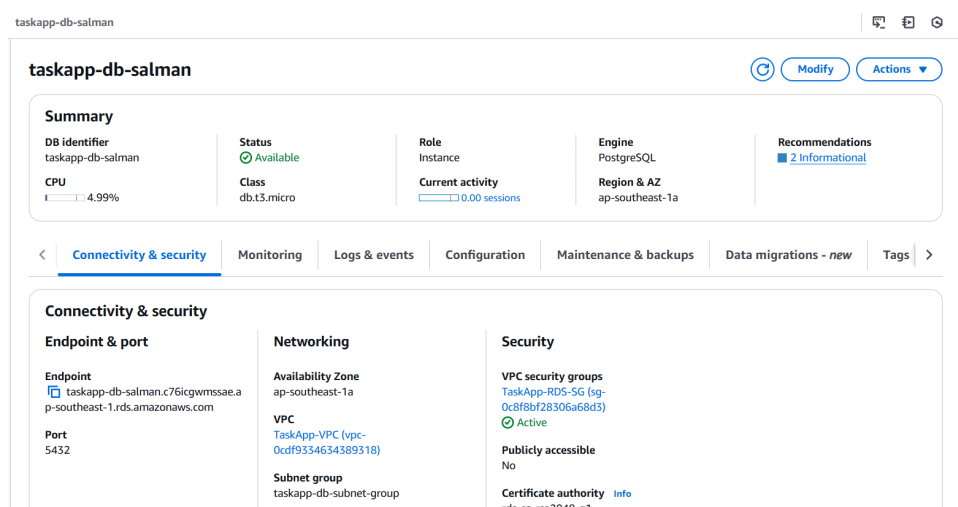


Figure 6: RDS Database Configuration

## 6.6 S3 Bucket Configuration

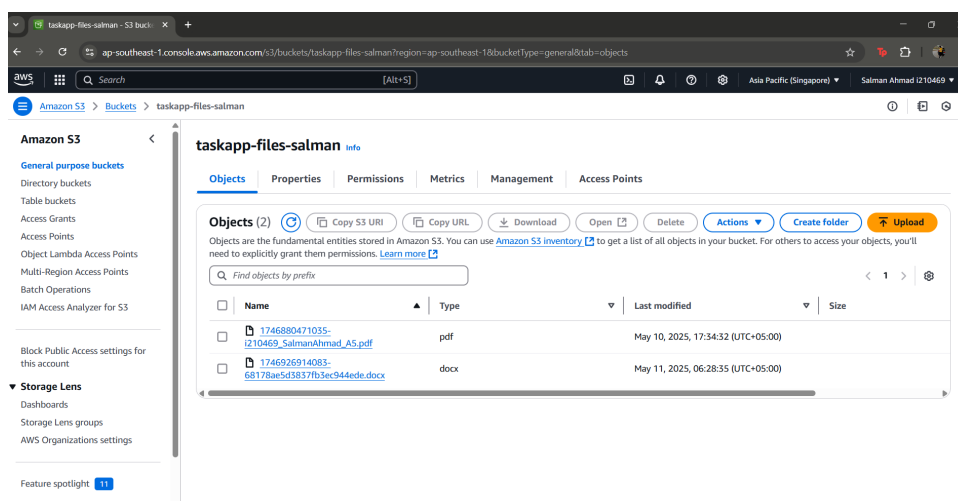


Figure 7: S3 Bucket for File Storage

## 6.7 Elastic Beanstalk Deployment

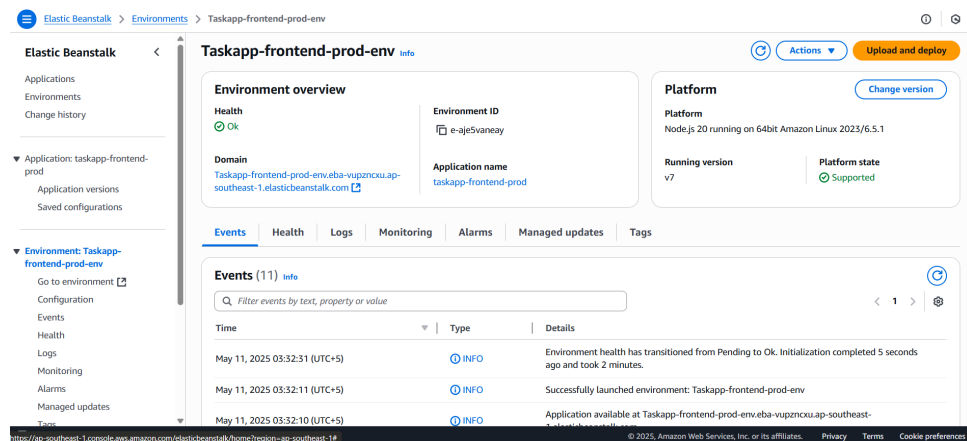


Figure 8: Elastic Beanstalk Deployment

## 6.8 Application Screenshots

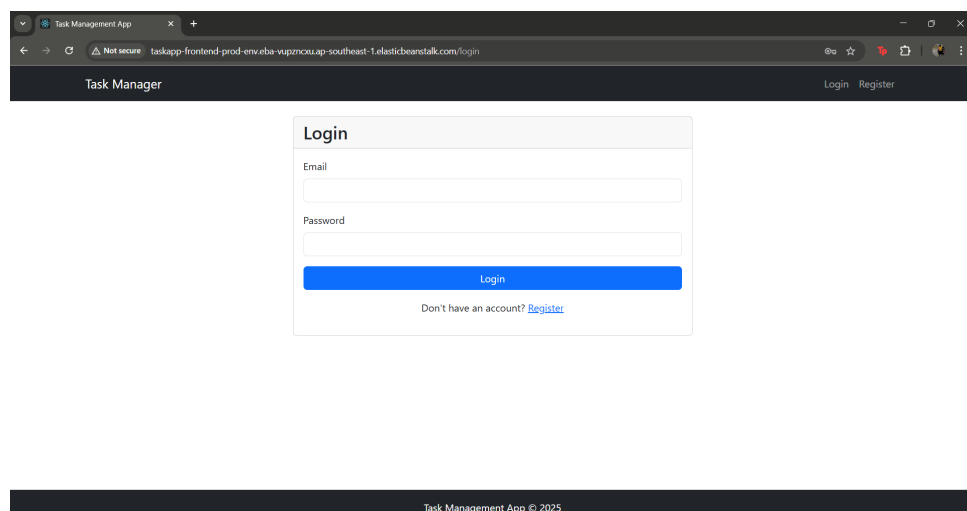


Figure 9: Application Login Screen

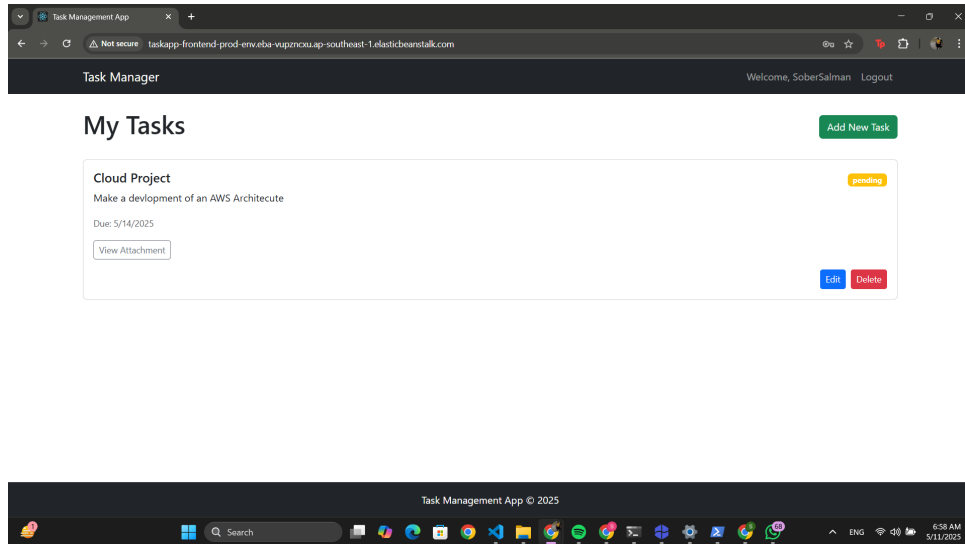


Figure 10: Application Dashboard

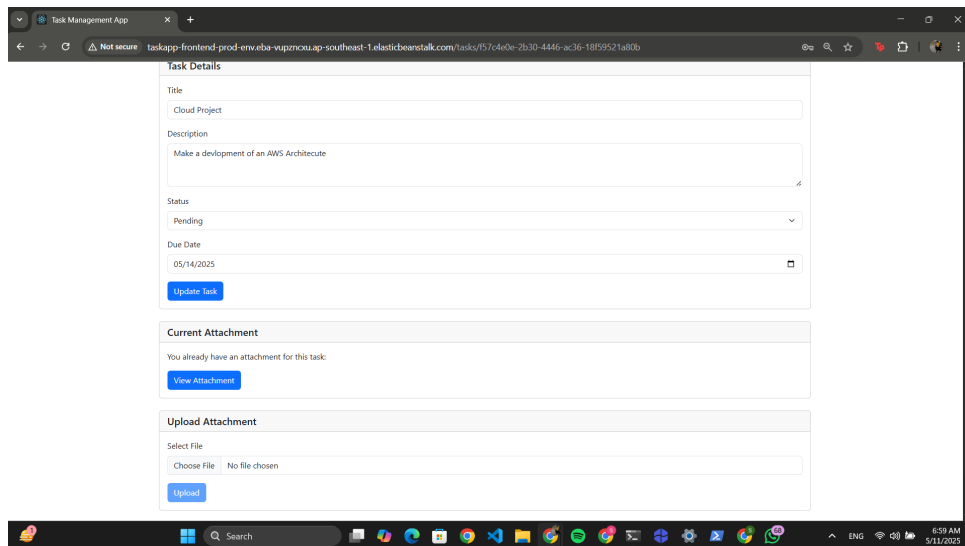


Figure 11: Task Detail with File Upload

## 7 Conclusion

This project successfully implements a cloud-based web application on AWS infrastructure following modern cloud-native architecture principles. The application includes all required components:

- User authentication system
- CRUD operations for task management
- Database integration using Amazon RDS
- File upload capabilities with Amazon S3
- Frontend and backend deployed as separate components
- Secure infrastructure with proper IAM roles and security groups

The deployment process demonstrates practical knowledge of AWS services and cloud architectural patterns. The application is secure, scalable, and follows best practices for cloud application development and deployment.

## 7.1 Future Improvements

- Implement HTTPS with AWS Certificate Manager
- Set up CI/CD pipeline for automated deployments
- Add monitoring and alerting with CloudWatch
- Implement auto-scaling for the application tier
- Enhance security with AWS WAF and Shield