UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

DEHRADUN, UTTARAKHAND

Project Report: Brew Haven Web Application & CI/CD Pipeline

Project: TechNova_CICDpipeline_devsecops

Submitted by:

Akash Chauhan

Khushi Chauhan

Dhruv Chaubey

Aryan Karnwal

Aanidhay Aggarwal

Ankur Sharma

Submitted to:

Utkarsh Agarwal

Table of Contents

Appendix D: style.css

Appendix E: sonar-project.properties

1. Introduction

1.1 Project Overview

This report details the development and deployment of the "Brew Haven" web application, a project designed to establish a vibrant online presence for a coffee shop. The project encompasses front-end and back-end development, integrated with a robust, automated DevSecOps pipeline for continuous integration, delivery, and monitoring. The application serves as a digital gateway for customers, providing information about the menu, services, and community events, while also featuring an interactive AI-powered chatbot for real-time customer engagement.

The core of the project is a Flask-based web application that presents a modern, single-page interface to users. The technology stack extends to include a sophisticated CI/CD pipeline managed by GitHub Actions, which automates infrastructure provisioning on AWS with Terraform, ensures code quality with SonarCloud, and deploys the application as a Docker container. A key feature of this pipeline is the integration of the AWS CloudWatch agent for comprehensive log monitoring, which adds a crucial layer of observability to the application's operations.

1.2 Project Objectives

The primary objectives of the Brew Haven project are:

Develop a dynamic and engaging web presence for the Brew Haven coffee shop to attract and inform customers.

Create an intuitive user interface that showcases the menu, tells the story of the brand, and facilitates customer orders and interaction.

Implement an AI-powered chatbot to provide instant, automated responses to customer inquiries, enhancing user experience and operational efficiency.

Build a fully automated CI/CD pipeline to streamline the development lifecycle from code commit to deployment.

Integrate security best practices into the pipeline (DevSecOps) through automated code quality analysis.

Establish a scalable and resilient infrastructure using Infrastructure as Code (IaC) principles with Terraform on AWS.

Implement a monitoring solution to capture application and system logs for operational insight and troubleshooting.

1.3 Scope of the Project

The scope of this project includes:

Front-End: A single-page HTML/CSS website with multiple sections, including Home, About, Menu, Reviews, Order, and Team information.

Back-End: A Python Flask server that renders the webpage and provides a backend API for the chatbot functionality.

AI Chatbot: Integration with the Google Generative AI API to power a "BrewBot" capable of answering questions based on a predefined business context.

CI/CD Pipeline: A GitHub Actions workflow that automates the entire build, test, and deploy process.

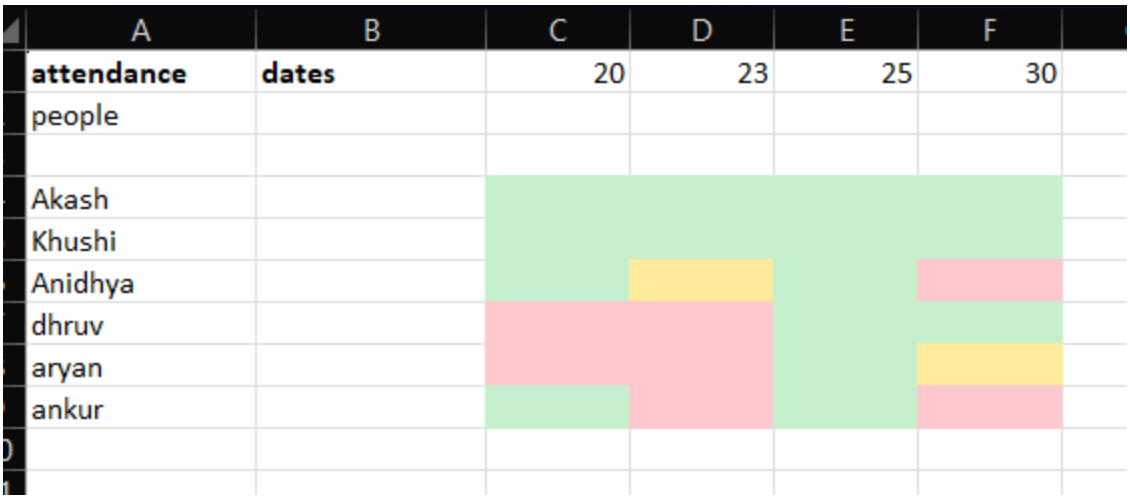
Infrastructure: AWS EC2 instance provisioning managed via Terraform.

Containerization: Dockerization of the Flask application for consistent deployment environments.

Code Analysis: Automated static code analysis using SonarCloud to identify bugs, vulnerabilities, and code smells.

Monitoring: Deployment of the AWS CloudWatch agent to collect and centralize application and security logs from the deployed container and host system.

2. Individual Contributions



	A	B	C	D	E	F
	attendance	**dates**	20	23	25	30
	people					
	Akash					
	Khushi					
	Anidhya					
	dhruv					
	aryan					
	ankur					

MOM’s

MOM 20
Work distribution in the meeting
Akash - cicd pipeline
Khushi - monitoring
Anidhya - website building '
ankur - docker

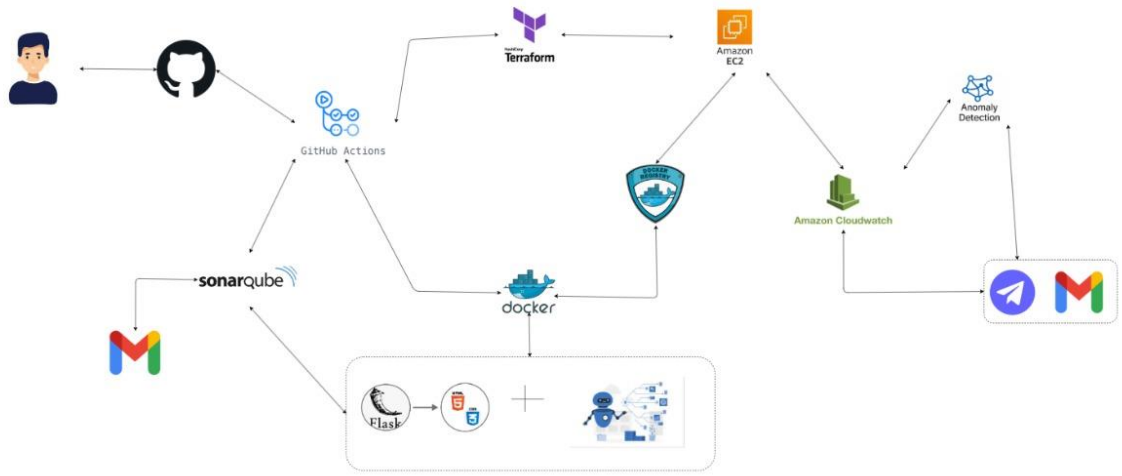
MOM 23
Akash - drawn the DFD
Khushi - architecture discussed

MOM 25
all members contribiuted workflows and their progress .

3. System Architecture

2.1 Architectural Diagram

The project's architecture is visualized in the diagram below, which outlines the flow from the developer to the end-user, encompassing the CI/CD pipeline and the cloud infrastructure.



2.2 Component Overview

The architecture consists of the following key components:

Version Control System (GitHub): The central repository hosting the application source code and the GitHub Actions workflow definitions.

CI/CD Pipeline (GitHub Actions): The automation engine that orchestrates the entire process. It is triggered on every push to the main branch and includes jobs for provisioning, code analysis, and deployment.

Infrastructure as Code (Terraform): Used to provision and manage the AWS EC2 instance where the application is hosted. This ensures the infrastructure is reproducible and version-controlled.

Code Quality (SonarCloud): Integrated into the pipeline to perform static analysis on the codebase. It checks for bugs and security vulnerabilities, and a failed quality gate can block the deployment, enforcing a high standard of code quality.

Containerization (Docker): The Flask application is packaged into a Docker image, which is stored in DockerHub. This ensures a consistent runtime environment and simplifies the deployment process.

Cloud Provider (AWS): The application is hosted on an Amazon EC2 instance. The pipeline interacts with AWS to manage the instance and deploy the application.

Monitoring (AWS CloudWatch): The CloudWatch agent is installed on the EC2 instance to collect logs from the Docker container (application logs) and critical system files (auth.log, syslog). These logs are centralized in CloudWatch for analysis and alerting.

Web Application (Flask & Generative AI): The user-facing component, consisting of the Flask backend and the HTML/CSS/JS front-end, which includes the AI chatbot.

3. Front-End Development

3.1 User Interface and Experience (UI/UX)

The front-end of the Brew Haven website is designed to be visually appealing, modern, and user-friendly. It is a single-page application built with HTML5 and styled extensively with CSS3.

Layout and Design: The website uses a clean, section-based layout. The color palette is warm and inviting, dominated by shades of brown and cream, which aligns with the coffee shop theme. A fixed navigation bar allows for easy scrolling to different sections of the page.

Typography and Imagery: The design utilizes sans-serif fonts for readability and incorporates high-quality images of coffee, food, and the shop's ambiance to create an engaging visual experience. Images are used generously in the main, about, menu, and team sections.

Interactivity: The UI includes hover effects on navigation links, buttons, and menu cards, providing visual feedback to the user. The site also features icons from Font Awesome for social media links, navigation controls, and other UI elements.

3.2 Website Sections

The index.html file structures the website into several distinct sections, each serving a specific purpose:

Home: The landing section features a prominent headline, a main image, and a brief welcome message. It immediately communicates the brand's identity and value proposition.

About: This section provides more details about Brew Haven's mission and values, accompanied by an image to reinforce the brand story.

Menu: A grid-based layout showcases the various coffee and food items available. Each menu item is presented in a "card" with an image, name, description, price, and an "Order Now" button.

Review: This section displays customer testimonials in a card-based format, building social proof and trust. Each review includes the customer's name, photo, rating, and comments.

Order: A simple form allows users to place an order by providing their name, contact details, and order information.

Team: Introduces the staff with photos and brief bios, adding a personal touch to the brand.

Footer: Contains quick links, contact information, service highlights, and social media links.

3.3 Interactive Chatbot

A key feature of the front-end is the integrated chatbot, which enhances user engagement.

UI Implementation: The chatbot is presented as a pop-up widget that can be toggled by the user. It has a dedicated header, a message display area, and an input field for users to type their queries.

Functionality: When a user sends a message, a POST request is sent to the /chat endpoint of the backend Flask application. The bot's response is then dynamically appended to the chat window. This interactive element provides immediate assistance to users without them needing to leave the website.

4. Back-End Development

4.1 Flask Application

The back-end is powered by a lightweight Python web framework, Flask. The entire back-end logic is contained within the app.py file.

Dependencies: The application relies on Flask for the web server, and google-generativeai for the AI integration, as listed in requirements.txt.

Routing: The application defines two routes:

@app.route('/'): This is the main endpoint that renders the index.html template, serving the website to the user.

@app.route('/chat', methods=['POST']): This endpoint handles incoming messages from the front-end chatbot. It receives the user's message, processes it, and returns the AI-generated response as a JSON object.

4.2 AI-Powered Chatbot Integration

The chatbot, named "BrewBot," is the most complex feature of the backend. It uses Google's Generative AI to provide intelligent and context-aware responses.

Configuration: The application is configured with a Google AI API key and uses the gemini-pro model.

Business Context: A comprehensive BUSINESS_CONTEXT string is defined within app.py. This string serves as the foundational knowledge base for the AI model. It contains detailed information about:

Brew Haven's Identity: Mission, location, and personality.

Operational Details: Hours, contact info, and amenities.

Full Menu: Including descriptions and prices for all items.

Special Offers: Student discounts, loyalty programs, etc.

Community Events: Open mic nights, book clubs, etc.

Customer Service Guidelines: Defines the bot's role, capabilities, limitations, and tone of voice.

Prompt Engineering: For each user message, the application constructs a prompt that combines the predefined BUSINESS_CONTEXT with the user's query. This technique ensures that the AI's responses are grounded in the specific details of the Brew Haven coffee shop, making the chatbot a highly effective virtual assistant.

5. DevSecOps: CI/CD & Monitoring

The backbone of the project is its sophisticated DevSecOps pipeline, defined in .github/workflows/main.yml. This pipeline automates every step from code integration to production deployment and monitoring.

5.1 CI/CD Pipeline Overview

The pipeline, named TechNova CI/CD Pipeline, is triggered automatically on every push to the main branch. It consists of three sequential jobs: provision, sonarqube, and deploy.

5.2 Job 1: Infrastructure Provisioning with Terraform

Purpose: This job ensures that the necessary cloud infrastructure is in place.

Steps:

Checkout Code: Fetches the source code from the repository.

Setup Terraform: Initializes the Terraform environment.

Configure AWS Credentials: Uses secrets to securely authenticate with AWS.

Terraform Init & Apply: Runs terraform init and terraform apply -auto-approve to create or update the EC2 instance.

Ensure Instance is Running: A crucial script checks the state of the EC2 instance. If it is stopped, it starts it. If it is terminated, the pipeline fails with an error, preventing deployment to a non-existent server.

Get IP Address: Once the instance is confirmed to be running, its public IP address is retrieved and passed as an output to the deploy job.

5.3 Job 2: Code Quality with SonarQube

Purpose: This job integrates the "Sec" part of DevSecOps by performing static code analysis.

Steps:

Checkout Code: Fetches the code with full Git history for a more accurate analysis (fetch-depth: 0).

Setup Python & Install Dependencies: Prepares the environment by installing the project's dependencies from requirements.txt.

SonarCloud Scan: Runs the SonarCloud scanner. It uses the sonar.projectKey and other properties defined in sonar-project.properties and authenticates using a SONAR_TOKEN secret.

Quality Gate Check: After the scan, a script queries the SonarCloud API to check the project's Quality Gate status. If the status is not "OK," the job fails, which in turn blocks the deployment job.

Email Notification: Regardless of the outcome, an email is sent to a specified address with the results of the SonarQube analysis, ensuring that stakeholders are immediately informed of the code quality status.

5.4 Job 3: Build, Deploy, and Monitor

Purpose: This job is responsible for containerizing the application, deploying it to the EC2 instance, and setting up monitoring. It depends on the successful completion of the provision and sonarqube jobs.

Steps:

Checkout Code & Login to DockerHub: Prepares for the image build.

Build and Push Docker Image: Builds the Docker image from the Dockerfile in the repository and pushes it to DockerHub, tagging it as technova-app:latest.

Deploy to EC2: Uses an SSH action to connect to the EC2 instance (using the IP address from the provision job) and runs a deployment script. The script pulls the latest Docker image, stops and removes any existing container, and starts the new container, mapping port 80 on the host to port 80 in the container.

Install and Configure CloudWatch Agent: This is a critical monitoring step. The script connects to the EC2 instance again and:

Downloads and installs the Amazon CloudWatch agent.

Creates a config.json file for the agent. This configuration is set up to collect logs from two primary sources:

Application Logs: All log files from Docker containers located at /var/lib/docker/containers/*/*.log. These are sent to the TechNova-App-Logs log group.

Security & System Logs: Important system logs like /var/log/auth.log (authentication attempts) and /var/log/syslog (general system messages) are collected and sent to their own dedicated log groups in CloudWatch.

Starts the CloudWatch agent service, which begins streaming these logs to AWS CloudWatch in real-time.

6. Conclusion

6.1 Project Summary

The Brew Haven project successfully meets all its objectives. It delivers a feature-rich, customer-facing web application with a sophisticated AI chatbot. More importantly, it establishes a mature, fully automated DevSecOps pipeline that ensures rapid, reliable, and secure delivery of software. The integration of Terraform for IaC, SonarCloud for code quality, Docker for containerization, and AWS CloudWatch for monitoring represents a modern and robust approach to software development and operations. The pipeline not only automates deployment but also enforces quality and provides critical operational visibility.

6.2 Future Work

While the current implementation is robust, there are several areas for future improvement:

Enhanced Front-End: The "Gallary" section, currently commented out in the HTML, could be implemented to showcase more images of the coffee shop.

Database Integration: The "Order" form is currently a static element. A database (e.g., PostgreSQL or MongoDB) could be added to the backend to store orders, user information, and menu items, making the application fully dynamic.

Advanced Monitoring: The CloudWatch integration could be expanded to include metric collection (CPU/memory usage, disk space) and the creation of alarms to automatically notify the team of performance issues.

Separate Environments: The CI/CD pipeline could be enhanced to support multiple environments (e.g., development, staging, production) with different approval gates for each.

Chatbot Memory: The chatbot is currently stateless. Its functionality could be extended to have memory of past conversations with a user during a single session.