CI/CD Pipeline System Design Document
Project Name: RecipeConnect
Team: DevourDevs
Assignment 2

1. Introduction

This document describes the design and implementation of the Continuous Integration and Continuous Deployment (CI/CD) pipeline for RecipeConnect. The CI/CD pipeline automates the testing, building, and deployment of the application's containerized frontend and backend. It leverages GitHub Actions to run tests on multiple Node.js versions (CI) and to build, tag, and push Docker images to Docker Hub (CD). In production, these images are deployed using Docker Compose with a production configuration file.

2. System Overview

RecipeConnect is structured as follows:

- Frontend (Client): A React-based application (located in the client folder) that handles UI, routing, and API interactions.
- Backend (Server): A Node.js/Express application (located in the server folder) that manages authentication, business logic, and database operations.
- Database: A MongoDB instance (locally during development, with deployment instructions provided for production).
- External APIs: Interfaces with Spoonacular, Gemini, and Roboflow for recipe data, AI assistant features, and ingredient detection.

For containerization:

- Docker is used to build container images.
- docker-compose orchestrates multi-container builds (and is later used in production to deploy containers).
- GitHub Actions automates the CI/CD workflow by running tests, building images, and pushing them to Docker Hub.

3. CI Pipeline Design

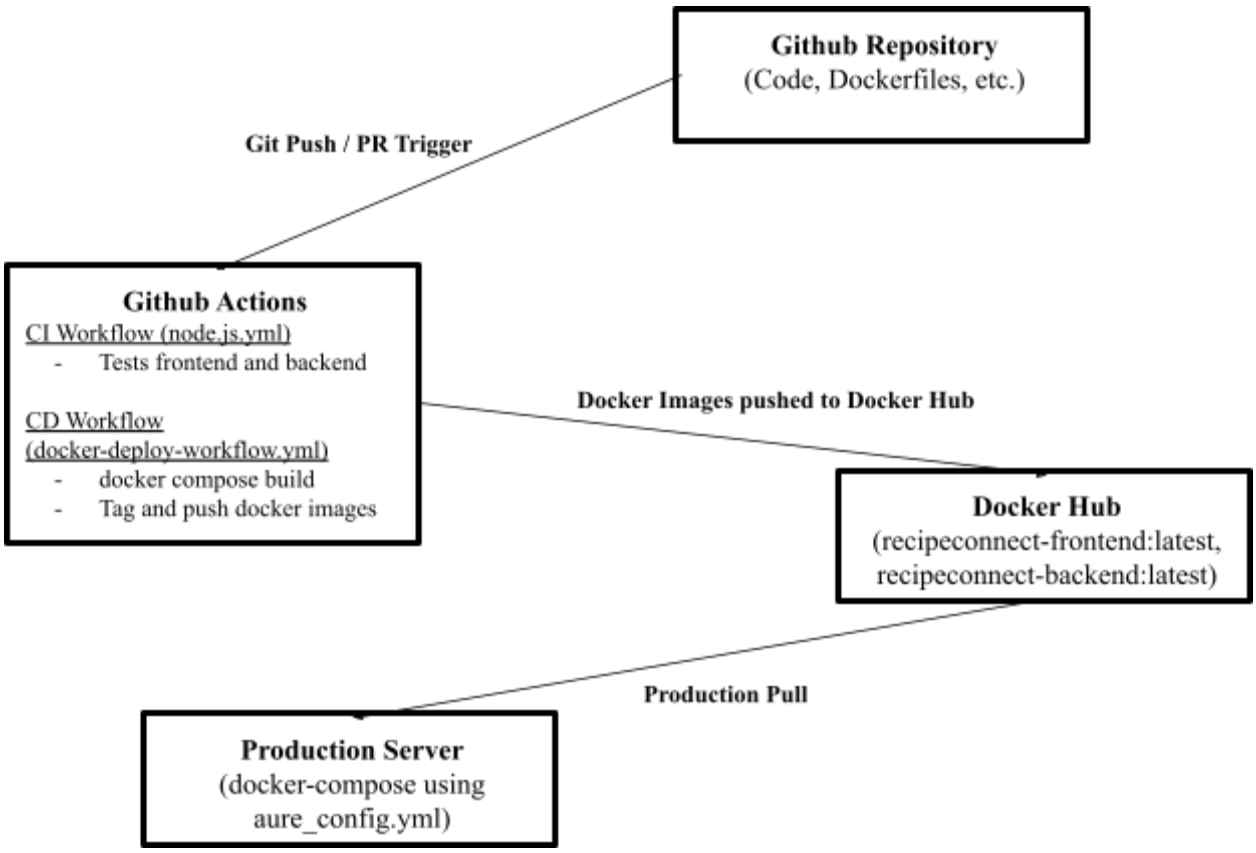
- File: .github/workflows/node.js.yml
- Trigger: Runs on any push or pull request to the main branch.
- Matrix Strategy: Tests run on Node.js versions 18.x, 20.x, and 22.x.
- Steps:
    - Checkout: Uses actions/checkout@v4 to clone the repository.
    - Setup Node.js: Uses actions/setup-node@v4 to install the correct Node.js version and enable npm caching.
    - Testing Frontend: Changes directory to client, installs dependencies, and runs tests.

    - Testing Backend: Installs dependencies at the project root, then moves into the server folder to install dependencies and run backend tests.

4. CD Pipeline Design

- File: .github/workflows/docker-deploy-workflow.yml
- This workflow is designed to be manually triggered (workflow_dispatch)
- Checkout: Clones the repository.
- Docker Compose Build: Uses Docker Compose (with the command docker compose build) to build both frontend and backend images based on the definitions in docker-compose.yml.
- Check Docker Compose Version: Verifies that Docker Compose V2 is available on the runner.
- Log in to Docker Hub: Uses docker/login-action@v2 to authenticate with Docker Hub using stored secrets.
- Tag and Push: Tags the built images with your Docker Hub repository names and pushes them.
- File: docker-compose.yml
    - This file defines how to build the Docker images locally (or in CI) using two services:
    - backend: Uses the Dockerfile in ./server and tags the image as recipeconnect-backend:latest.
    - frontend: Uses the Dockerfile in ./client and tags the image as recipeconnect-frontend:latest.
- File: aure_config.yml
    - This file is intended for production deployment.
    - It doesn't rebuild the images; instead, it uses images already pushed to Docker Hub.
    - It substitutes the Docker Hub username (using the environment variable DOCKER_USERNAME) into the image names.
- Dockerfiles (client/Dockerfile and server/Dockerfile)
    - These files build the frontend and backend container images.

5. Deployment Diagram



Key Components:

- CI/CD Server (GitHub Actions Runner):
    - Executes automated tests, builds Docker images, and pushes them to Docker Hub.
- Docker Hub:
    - Serves as the container registry where built images are stored.
- Production Environment:
    - Uses a Docker Compose file (aure_config.yml) to deploy containers for frontend, backend, and supporting services.