CI/CD Pipeline with GitHub Actions & Docker

Done by: **Mohamed Askhaf B.Tech.IT.**

Date: September 08, 2025

Introduction

Continuous Integration and Continuous Deployment (CI/CD) enable teams to deliver software quickly, safely, and consistently. This project implements a complete CI/CD pipeline for a Node.js application using **GitHub Actions** and **Docker**, pushing images to **Docker Hub** and running locally (no public cloud required). Automation reduces manual effort, catches issues early through testing, and standardizes deployments via containers.

Abstract

A small Express.js service was containerized and wired to a GitHub Actions workflow. On every push to *main*, the pipeline installs dependencies, executes automated tests (Jest + Supertest), builds a Docker image, authenticates using repository secrets, and pushes the image to Docker Hub. This setup demonstrates a reliable, reproducible CI/CD flow that can be extended to Kubernetes/Minikube or on prem environments.

Tools Used

- **Git & GitHub** – version control and remote repository.
- **GitHub Actions** – CI/CD workflow automation (build, test, push).
- **Node.js (Express) & npm** – application runtime and package management.
- **Jest & Supertest** – unit/integration testing for APIs and routes.
- **Docker** – containerization for consistent runtime across machines.
- **Docker Hub** – remote image registry for distribution/storage.
- **VS Code / Terminal** – authoring, debugging, and local runs.

Steps Involved in Building the Project

1. **Initialize the App & Repo:** Created an Express.js app (*app.js*), added routes/health endpoint, and committed code to a new GitHub repository.
2. **Define Project Metadata:** Added *package.json* with *start* and *test* scripts; installed dependencies and devDependencies (express, jest, supertest).
3. **Containerize with Docker:** Wrote a *Dockerfile* (FROM node:18 alpine, set WORKDIR, copy package files, *npm install*, copy source, expose port, set CMD). Optional *docker compose.yml* used for local runs.
4. **Create CI/CD Workflow:** In *.github/workflows/ci-cd.yml*, steps include *checkout*, *setup node*, *npm install*, *npxjest* tests, Docker login using secrets, image build, and push to Docker Hub.
5. **Configure Secrets:** Added *DOCKERHUB_USERNAME* and *DOCKERHUB_TOKEN* (access token) under GitHub → Settings → Secrets and variables → Actions.
6. **Trigger & Verify:** Pushed to *main* to trigger the workflow; validated green checks in the Actions tab and confirmed the new image tag in Docker Hub.
7. **Run Locally:** Pulled the image and ran the container (e.g., *docker run -p 3000:3000 <user>/devops-ci-cd:latest*) to verify the app response at http://localhost:3000.

Key Outcomes

- Automated pipeline ensures repeatable builds and faster feedback on code changes.
- Tests block faulty changes from progressing to image build/publish.
- Containerized app runs consistently across developer machines and VMs.
- No external cloud required—suitable for local/minikube demo setups.

Conclusion

The pipeline successfully demonstrates end to end CI/CD for a Node.js service using open source tooling. By combining automated tests with containerized builds and a remote registry, the project delivers a robust foundation for modern DevOps workflows. These practices are directly transferable to larger services and to orchestrators like Kubernetes/Minikube for staged or production deployments.