CI/CD Pipeline with Docker and GitHub Actions

Introduction

In modern software development, automation plays a vital role in ensuring consistency, speed, and reliability of application delivery. Continuous Integration and Continuous Deployment (CI/CD) pipelines help developers build, test, and deploy applications seamlessly. This project demonstrates a simple yet effective CI/CD workflow using **Node.js, Docker, and GitHub Actions**.

Abstract

The project involves developing a Node.js Express application, containerizing it with Docker, and automating the build and deployment process using GitHub Actions. Each commit or push to the repository triggers a pipeline that runs tests, builds a Docker image, and pushes it to Docker Hub. The deployed app runs locally on localhost:3000 serving a professional front-end page.

Tools Used

- **Node.js & Express.js** – Backend server and application framework
- **Docker** – Containerization for portability and consistency
- **GitHub Actions** – CI/CD automation platform
- **Docker Hub** – Container registry for hosting application images
- **Git & GitHub** – Version control and project collaboration

Steps Involved in Building the Project

1. **Application Development** – Created a Node.js Express server with routes and a static index.html page for UI.
2. **Dockerization** – Wrote a multi-stage Dockerfile to build, test, and produce a lightweight production image.
3. **CI/CD Workflow Setup** – Configured GitHub Actions (ci.yml) to:
    - Install dependencies
    - Run tests (app.test.js)
    - Build Docker image
    - Push the image to Docker Hub automatically
4. **Testing and Verification** – Validated the pipeline by pushing commits and confirming successful builds in GitHub Actions.

5. **Deployment** – Pulled the Docker image from Docker Hub and ran it locally using docker run, exposing the app on http://localhost:3000.

Conclusion

This project successfully demonstrates how CI/CD pipelines streamline software delivery. By integrating Docker and GitHub Actions, the process of building, testing, and deploying a Node.js application becomes automated and reliable. Such workflows reduce manual errors, speed up development cycles, and establish a strong foundation for future scalable cloud deployments.