CI/CD Pipeline with GitHub Actions & Docker (Local Deployment)

Introduction

In the modern software development lifecycle, Continuous Integration and Continuous Deployment (CI/CD) are essential for maintaining high code quality, speeding up delivery, and automating workflows. This project demonstrates the setup of a full CI/CD pipeline using GitHub Actions and Docker, with deployment performed locally using Docker.A basic Node.js application is developed, containerized using Docker, and integrated with GitHub Actions to automate building, testing, and deployment processes. The Docker image is pushed to Docker Hub and can be deployed through Docker on any local machine.

Abstract

This project serves as a complete demonstration of implementing a CI/CD pipeline for a containerized application, enabling seamless automation and deployment using only opensource tools. A minimal web server is developed using Node.js. Upon every code push or pull request, GitHub Actions is triggered to build the Docker image and push it to Docker Hub. The application is then deployed using Docker Compose for local testing.This project is particularly useful for learning DevOps practices, especially in environments where cloud resources are not available or for developers seeking to experiment with automation pipelines in a cost-free, local environment.

Tools Used

- Node.js: Used to create the sample web application.
- Docker: To containerize the Node.js app.
- Docker Compose: Simplifies local development and testing.
- GitHub Actions: Automates CI/CD steps like testing, building, and deployment.
- Docker Hub: Stores and distributes Docker images.

Steps Involved in Building the Project:

Application Development

- A simple web server is created using Node.js.
- The server listens on port 3000 and responds with “Hello from GitHub Actions Pipeline!” to any request.

Dockerization

- A Dockerfile is written to define how the Node.js app should be built and run inside a container.
- docker-compose.yml allows for easy execution in local development environments.

GitHub Repository Setup

- The project is pushed to a GitHub repository.
- A workflow file is created under .github/workflows/main.yaml to define the CI/CD steps.

Configuring CI/CD with GitHub Actions • GitHub Actions is set to trigger on every push or pull request to the main branch.

- Steps in the workflow include:
  - Checking out the repository.
  - Logging in to Docker Hub using GitHub Secrets. ○ Building the Docker image. ○ Pushing the image to Docker Hub.

Creating and Configuring Secrets

- DOCKER_USERNAME and DOCKER_PASSWORD are added to GitHub as secrets to authenticate Docker Hub access.

Local Deployment Options:

Using Docker Compose

- Run docker-compose up to build and run the container.
- The app becomes accessible at http://localhost:3000.

Verifying the Results

- Successful GitHub Actions build logs.
- Docker Hub repository shows the new image.
- Browser displays “Hello from GitHub Actions Pipeline!”.

Conclusion

This project successfully demonstrates the implementation of a full CI/CD pipeline using opensource tools with no cloud dependency. GitHub Actions effectively automates building and pushing Docker images to Docker Hub. Local deployment using Docker ensures that the application can be tested in a production-like environment without incurring cloud costs.This project forms a strong foundation for more complex DevOps implementations in the future, including multi-environment pipelines, advanced testing strategies, and cloud-based deployment workflows.