CI/CD Pipeline with GitHub Actions & Docker

Introduction

In the modern software development lifecycle, Continuous Integration and Continuous Deployment (CI/CD) play a vital role in ensuring fast and reliable delivery of applications. This project demonstrates the implementation of a CI/CD pipeline using **GitHub Actions** for automation and **Docker** for containerization. The pipeline automates code build, testing, container image creation, and deployment, ensuring a smooth workflow for developers and operations teams.

Abstract

The project focuses on building a fully automated CI/CD pipeline that:

1. Monitors changes in a GitHub repository.
2. Automatically builds and tests the application.
3. Creates a Docker image of the application.
4. Pushes the image to **Docker Hub** for versioned storage and reuse.
5. Enables easy deployment to production or staging environments.

This solution minimizes manual intervention, reduces human error, and accelerates the release cycle of applications.

Tools Used

- **GitHub Actions** – For building and running CI/CD workflows.
- **Docker** – For packaging the application into portable containers.
- **Docker Hub** – For storing and distributing container images.
- **GitHub Repository** – For source code management and version control.

Steps Involved in Building the Project

1. **Repository Setup** – Created a GitHub repository and pushed application source code.
2. **Dockerization** – Wrote a `Dockerfile` to define the application image.
3. **GitHub Actions Workflow** – Configured a `.github/workflows/main.yml` file with build, test, and deploy jobs.
4. **CI/CD Pipeline** –
    - On every push or pull request, the pipeline triggers automatically.
    - The workflow builds the Docker image, runs tests, and pushes the image to Docker Hub.
5. **Deployment** – The Docker image can be pulled from Docker Hub and run in any environment, ensuring consistent application behavior.

Conclusion

This project successfully demonstrates a CI/CD pipeline integrating **GitHub Actions** and **Docker**. It highlights the importance of automation in modern DevOps practices and showcases how containerization simplifies deployment across environments. The pipeline is flexible, scalable, and can be extended to include security scans, notifications, and Kubernetes/ECS deployments in the future.