Project title : CI/CD Pipeline with

GitHub Actions & Docker

**Your name : prashanth panjaala**

**Date : 21/04/2025**

**Company name : Elevate labs**

**✅ Objective**

Build and deploy a Dockerized app using GitHub Actions, Docker Hub, and Minikube/local VM.

**⚙️ Tools Used**

- GitHub Actions

- Docker + Docker Hub

- Minikube (or local VM)

- Python / Node.js

**🔧 What I Did**

1. Created Dockerfile and docker-compose.

2. Wrote GitHub Actions workflow to build, test, and push Docker image.

3. Pulled and ran the image locally using Docker/Minikube.

**📸 Results**

- CI/CD pipeline runs on push to `main`.

- Docker image auto-published: `https://hub.docker.com/r/youruser/yourapp`

- App runs successfully on localhost via Docker or Minikube.

**🧠 Conclusion**

I now have a working CI/CD pipeline that builds, tests, and deploys my app locally without needing any cloud services.

**📘 Introduction**

In modern software development, Continuous Integration and Continuous Deployment (CI/CD) pipelines are essential for delivering reliable applications quickly and efficiently. This project demonstrates how to set up a complete CI/CD pipeline using GitHub Actions and Docker, with deployment done entirely on a local environment — eliminating the need for cloud infrastructure.

The goal is to automate the process of building, testing, and deploying a containerized application whenever changes are pushed to the main branch of a GitHub repository. The pipeline uses GitHub Actions to run tests, build a Docker image, and push it to Docker Hub. The image is then pulled and deployed locally using Minikube or a virtual machine, simulating a production-like environment.

This approach is ideal for developers and teams who want to practice or implement CI/CD workflows without relying on cloud services, keeping everything fully local and cost-free. It also lays a strong foundation for extending the pipeline to more advanced deployment strategies like blue-green or canary deployments in the future.

**🎯 Objectives**

**The main objectives of this project are:**

1. Set up a local CI/CD pipeline using GitHub Actions, Docker, and Docker Hub — with no cloud dependency.
2. Automate testing, building, and publishing of a Dockerized application when code is pushed to the main branch.
3. Deploy the Docker image locally using Minikube or a virtual machine to simulate a real-world deployment environment.
4. Ensure code quality and reliability through automated testing before deployment.
5. Track and verify CI/CD pipeline results through workflow logs and local runtime behavior.

**⚙️ Implementation**

**The project implementation is divided into the following key steps:**

**1. Dockerization of the Application**

* A Dockerfile was created to package the app and its dependencies.
* Optionally, a docker-compose.yml file was added to simplify local development**.**

**2. GitHub Actions Workflow**

* A CI/CD workflow was defined in .github/workflows/ci-cd.yml.
* The pipeline includes the following stages:
  + Checkout: Fetch the latest code from the repository.
  + Testing: Run automated tests using pytest (Python) or a test runner like jest (Node.js).
  + Build & Push: Build the Docker image and push it to Docker Hub using secrets for authentication.

**3. Local Deployment**

* **After the image is pushed to Docker Hub, it is pulled and deployed locally:**
  + Using Minikube with kubectl to create a deployment and expose it as a service.
  + Or using Docker CLI directly to run the container locally.

**📈 Monitoring**

**Monitoring in this project is focused on CI/CD workflow feedback and local runtime validation.**

**1. CI/CD Pipeline Monitoring**

* GitHub Actions provides:
  + Real-time logs for each job step (e.g., test pass/fail, build success/failure)
  + Status indicators on commits (green check for success, red X for failures)
* Screenshot evidence and logs were captured to verify successful CI/CD runs.

**2. Local Deployment Monitoring**

* **Logs are inspected using:**
  + docker logs <container-id> for Docker
  + kubectl logs <pod-name> for Minikube
* Port mapping is tested by accessing the app in a browser (localhost:<port>)
* Additional checks like curl and response validation are done to confirm the app is live and responding**.**

**✅ Results**

The CI/CD pipeline was successfully implemented and validated with the following outcomes:

* ✅ CI/CD Workflow Execution
  + On every push to the main branch, GitHub Actions:
    - Ran the test suite automatically.
    - Built the Docker image with no errors.
    - Pushed the image to Docker Hub (devopsprashanth/repo:Zomato).
    - Workflow runs completed successfully, as shown in the GitHub Actions dashboard**.**
* ✅ Automated Testing
  + Tests executed as part of the pipeline ensured code reliability.
  + Only valid builds were pushed to Docker Hub.
* ✅ Local Deployment
  + The Docker image was pulled and deployed locally using:
    - Minikube with kubectl (Kubernetes-style deployment), or
    - Docker CLI with docker run
  + The app was accessible through localhost, confirming successful local deployment.
* ✅ Logs and Screenshots
  + CI/CD logs from GitHub Actions confirm successful steps.
  + Screenshots and runtime output from Docker/Minikube show the app running as expected.

**🧠 Conclusion**

This project successfully demonstrated a full CI/CD pipeline using GitHub Actions and Docker, with deployment in a completely local environment — without relying on any cloud provider.

Key takeaways include:

* CI/CD pipelines improve development speed and reliability by automating testing and deployment.
* GitHub Actions is a powerful tool for integrating CI/CD without extra cost or complexity.
* Local deployment using Docker or Minikube is a great way to simulate production behavior in a controlled environment.
* This setup provides a solid foundation for future enhancements like:
  + Blue/green or canary deployments
  + Container orchestration with full Kubernetes
  + Integration with monitoring and alerting tools (e.g., Prometheus, Grafana)

Overall, the project achieved its objectives and validated a production-like CI/CD pipeline using only free and local tools.

**📎 Appendices**

**📁 A. Project Files**

**1. Dockerfile**

Located at the root of the project; used to containerize the app.

**2. docker-compose.yml**

Optional; simplifies local development with multi-container setups.

**3. GitHub Actions Workflow**

.github/workflows/ci-cd.yml – Defines the CI/CD steps:

* Code checkout
* Automated tests
* Docker build & push to Docker Hub

**4. Application Source Code**

Located under the /app directory (e.g., main.py or app.js).

**5. Test Files**

Located in the /tests folder (e.g., test\_basic.py).

**🔐 B. GitHub Secrets Used**

| **Secret Name** | **Purpose** |
| --- | --- |
| DOCKER\_USERNAME | Docker Hub username |
| DOCKER\_PASSWORD | Docker Hub access token/password |

**🐳 C. Docker Hub Image**

* Public Image:  
  <https://hub.docker.com/r/devopsprashanth/repo:zomato>

**📸 D. Screenshots & Logs**

**1. CI/CD Workflow Screenshot**

* Shows successful GitHub Actions run with green checks

**2. Docker CLI Output**

docker pull devopsprashanth/repo:zomato

docker run -d -p 8000:8000 devopsprashanth/repo:zomato

**3. Minikube Deployment Logs**

kubectl get pods

kubectl logs <zomato.service>

minikube service Zomato.service

**📚 E. Helpful Commands**

**Docker**

docker build -t yourapp .

docker run -itd –name -p 8000:8000 zomato

**Minikube**

minikube start

kubectl create deployment zomato --image= devopsprashanth/repo:zomato

kubectl expose deployment zomato --type=8080 --port=8000

minikube service zomato