CI/CD Pipeline with GitHub Actions & Docker

Introduction

Continuous Integration (CI) and **Continuous Deployment (CD)** are fundamental to modern DevOps practices.
They automate the software delivery process, ensuring faster and more reliable releases.
CI focuses on merging code changes frequently and running automated tests.
CD extends this by deploying the tested code into production or staging environments.

Automation reduces manual errors and increases developer productivity.
It ensures that every code change is validated before deployment.
In many organizations, CI/CD pipelines run on **cloud services**.
However, learning these concepts should not be limited by cloud costs.

This project shows how CI/CD can be built entirely on a **local environment**.
It uses **GitHub Actions** for automation, **Docker** for containerization, and **Docker Hub** for image storage.
For deployment, **Minikube** or a local VM simulates a cluster environment.
This allows learners to experience real DevOps workflows without needing cloud infrastructure.

By writing a **Dockerfile** and **docker-compose.yml**, applications are packaged consistently.
GitHub Actions runs tests, builds images, and pushes them to Docker Hub.
Minikube or a VM then pulls and runs the images to complete the pipeline.
This hands-on approach makes abstract CI/CD concepts practical and understandable.

In summary, the project provides a cost-free, repeatable, and effective way to practice CI/CD.
It bridges the gap between theory and real-world implementation, preparing learners for professional DevOps practices.

Through this approach, the project bridges the gap between **theory and practice**.
It gives learners hands-on experience in building pipelines, handling images, and managing deployments.
More importantly, it provides a practical understanding of DevOps concepts without incurring costs.
In conclusion, this project demonstrates how GitHub Actions and Docker can be combined to create a local CI/CD pipeline.
It offers a safe, cost-free, and efficient way to strengthen CI/CD and containerization skills.

Abstract

This project demonstrates the creation of a **local CI/CD pipeline** using GitHub Actions and Docker.
The pipeline enables automation of testing, building, and deployment stages for applications.

A simple application is written and containerized using a **Docker file**.
A **docker-compose.yml** file is used to define services for local execution.

The workflow begins when developers push code to **GitHub**.
GitHub Actions automatically triggers pipelines defined in workflow files.
The pipeline installs dependencies and executes automated tests.
After testing, the pipeline builds a **Docker image** for the application.

The entire process mimics a real-world DevOps pipeline.
It covers the stages of **build, test, publish, and deploy**.
Automation reduces human errors and improves consistency.
Every update is tested and deployed without manual steps.

The solution also demonstrates **repeatability**.
Any developer can clone the repo and reproduce the workflow.
The pipeline is portable and does not depend on cloud services.
This makes it ideal for students, beginners, and professionals.

The built image is then pushed to **Docker Hub**, serving as a registry.
From there, deployment is handled using **Minikube** or a local virtual machine.
Mini kube pulls the Docker image and runs the container locally.
This ensures the application is deployed consistently across environments.

The pipeline mimics **real-world DevOps practices** without needing cloud services.
It emphasizes **automation**, reducing manual steps in development workflows.
It ensures **repeatability**, as each code push results in the same deployment process.
It provides **portability**, since images can be reused on any Docker-supported platform.
The solution is cost-free, making it ideal for learning and experimentation.

Tools Used

- **GitHub Actions** – For CI/CD automation workflows.
- **Docker** – To build, package, and run applications in containers.
- **Docker Hub (Free)** – For storing and sharing built Docker images.
- **Minikube / Local VM** – For local deployment and testing.
- **docker-compose** – To define and run multi-container applications.
- **YAML** – For workflow and deployment configuration.

Steps Involved in Building the Project

1. Designed and implemented a simple sample application (e.g., Node.js, Python, or Java).
2. Wrote a **Dockerfile** to containerize the application.
3. Created a **docker-compose.yml** to define services for local development.
4. Initialized a **GitHub repository** and pushed the source code.
5. Configured **GitHub Actions workflow** (`ci-cd.yml`) to trigger on code pushes.
6. Added steps to install dependencies and run automated tests.

7. Configured steps to build the Docker image using the project's Docker file.
8. Logged into **Docker Hub** using GitHub Secrets for authentication.
9. Pushed the built image to Docker Hub repository.
10. Installed **Minikube** (or set up a local VM) for local deployment.
11. Pulled the Docker image from Docker Hub into Minikube/VM.
12. Ran the container locally to deploy the application.
13. Verified successful deployment by accessing the application endpoint.
14. Documented all steps in a **README** for reproducibility.

Conclusion

This project demonstrates the successful setup of a **local CI/CD pipeline**.
The pipeline integrates **GitHub Actions** for automation and **Docker** for containerization.
It provides a complete workflow covering build, test, and deployment stages.

Each code change pushed to GitHub triggers the workflow automatically.
Tests are executed to ensure the code is reliable and error-free.
Once validated, the Docker image is built using the project's Dockerfile.

The built image is tagged and securely pushed to **Docker Hub**.
This central registry allows sharing and versioning of application images.
It ensures portability across different systems and environments.

For deployment, the image is pulled into **Minikube** or a local VM.
The container runs locally, simulating a production-like environment.
This approach enables developers to test applications without cloud costs.

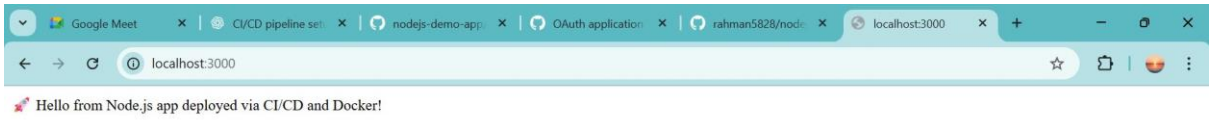
The project highlights the power of **automation in DevOps pipelines**.
It reduces manual work and guarantees consistent deployments.
It also enhances collaboration, since every developer shares the same pipeline.

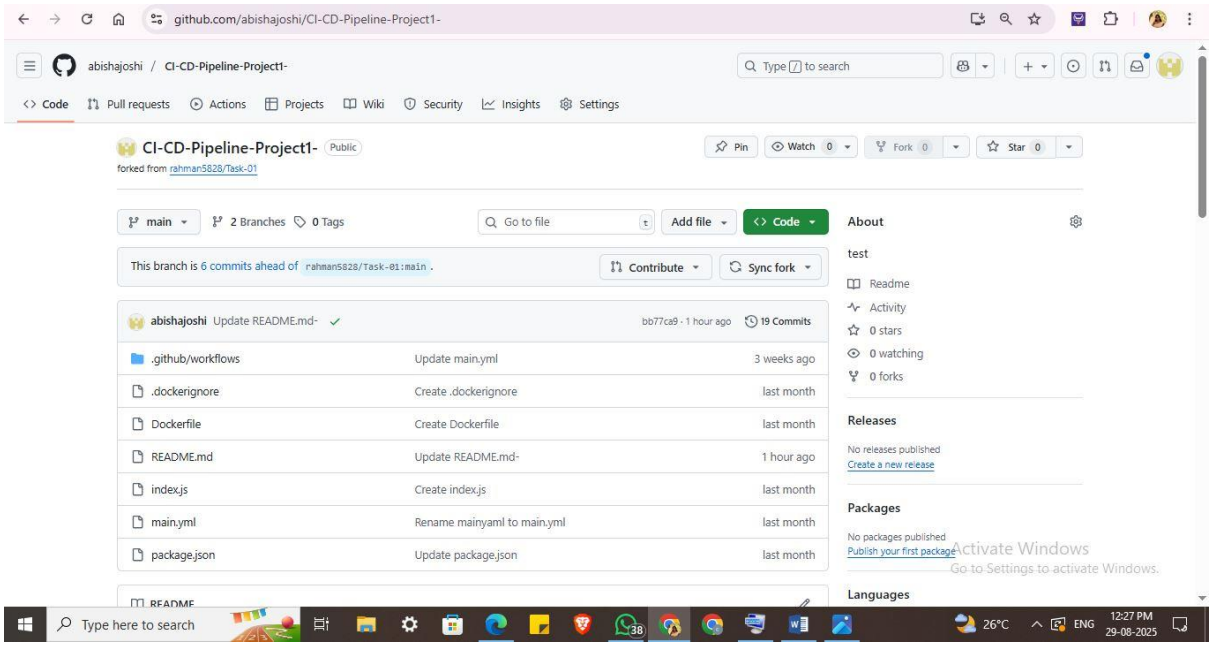
By using Docker Hub, the solution achieves **portability and scalability**.
By using Minikube, it provides a **safe and isolated testing setup**.
Together, these tools create a complete local DevOps workflow.

The project also emphasizes **reliability and repeatability**.
Any user can replicate the same pipeline by cloning the repository.
This makes it highly useful for students, learners, and professionals.

It bridges the gap between theoretical knowledge and real-world practice.
It proves that CI/CD does not always require expensive cloud platforms.
Instead, local setups can offer the same learning benefits at zero cost.

In summary, the pipeline demonstrates best practices in CI/CD automation.
It provides a practical, cost-effective way to master DevOps workflows.
The project is ideal for experimentation, learning, and professional training.
In conclusion, this solution empowers developers to practice **CI/CD concepts** effectively in a local environment.


Google Meet
CI/CD pipeline set
nodejs-demo-app
OAuth application
rahman5828/node
localhost:3000
localhost:3000
Hello from Node.js app deployed via CI/CD and Docker!


github.com/abishajoshi/CI-CD-Pipeline-Project1-
abishajoshi / CI-CD-Pipeline-Project1-
Type / to search
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
CI-CD-Pipeline-Project1- Public
forked from rahman5828/Task-01
Pin
Watch 0
Fork 0
Star 0
main
2 Branches
0 Tags
Go to file
Add file
Code
About
test
This branch is 6 commits ahead of rahman5828/Task-01:main.
Contribute
Sync fork
Readme
Activity
0 stars
0 watching
0 forks
abishajoshi Update README.md-
bb77ca9 · 1 hour ago
19 Commits
.github/workflows Update main.yml 3 weeks ago
.dockerignore Create .dockerignore last month
Dockerfile Create Dockerfile last month
README.md Update README.md- 1 hour ago
index.js Create index.js last month
main.yml Rename mainyaml to main.yml last month
package.json Update package.json last month
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Activate Windows
Go to Settings to activate Windows.
Languages
README
Type here to search
26°C
ENG
12:27 PM
29-08-2025