CI/CD pipeline using GitHub Actions and Docker

BY

VIGNESH M

ABSTRACT

In modern software development, Continuous Integration (CI) and Continuous Deployment (CD) play a crucial role in ensuring faster delivery, improved quality, and reduced manual intervention. This project focuses on implementing a CI/CD pipeline using GitHub Actions and Docker to automate the build, test, and deployment processes of a Node.js application.

The pipeline is triggered on every code push or pull request, where it executes automated steps such as dependency installation, testing, containerization of the application using Docker, and publishing the Docker image to Docker Hub. Deployment is then managed through Docker Compose, ensuring consistent environments across development and production.

By integrating GitHub Actions and Docker, this project eliminates manual errors, accelerates release cycles, and enhances application reliability. The implementation demonstrates how automation can streamline software delivery while providing a scalable foundation for future enhancements such as multi-environment deployment and Kubernetes orchestration.

INTRODUCTION

Continuous Integration (CI) and Continuous Deployment (CD) are essential practices in modern software development. CI/CD pipelines automate the process of building, testing, and deploying applications, ensuring faster delivery and higher quality software.

This project demonstrates how to set up a CI/CD pipeline using GitHub Actions and Docker for containerized application deployment.

Tools & Technologies Used

Programming Language – JavaScript (Node.js):

Used for developing the backend logic of the application, enabling asynchronous operations and lightweight server-side execution.

Version Control – Git, GitHub:

Git managed source code changes, while GitHub hosted the repository and enabled collaborative development and integration with CI/CD workflows.

CI/CD – GitHub Actions:

Automated the build, test, and deployment process using workflows triggered by code commits, ensuring continuous integration and delivery.

Containerization – Docker:

Packaged the application into portable containers, ensuring consistency across different environments.

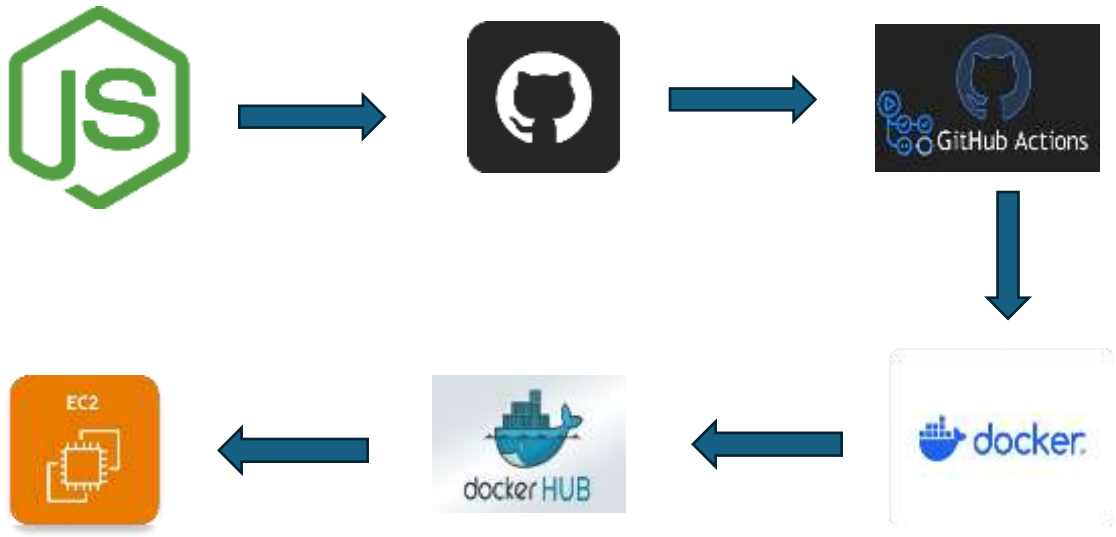
Registry – Docker Hub:

Served as the image repository where Docker images were pushed and stored for easy access during deployment.

Cloud/Deployment – AWS EC2:

The final deployment could be performed on a cloud virtual machine AWS EC2 making the system making the system flexible for different environments.

WORKFLOW OF DEPLOYMENT



Step-by-Step Process of the Project

1. Project Setup

1. Create a new project folder.
2. Initialize Git:
3. git init
4. Create the following files:

- app.js → Simple Node.js Express app
- package.json → Dependencies & scripts
- Dockerfile → Containerization instructions
- docker-compose.yml → Deployment configuration

2. Application Development

- Write a simple Node.js app (app.js) that returns:
- 🚀 Hello from CI/CD Pipeline with GitHub Actions & Docker!
- Define dependencies (express) in package.json.
- Verify locally by running:
- npm install
- npm start

3. Dockerization

1. Write a Dockerfile with Node.js base image.
2. Build the Docker image:
3. docker build -t cicd-docker-app .
4. Run container locally:
5. docker run -p 3000:3000 cicd-docker-app
6. Check app at → http://localhost:3000.

4. GitHub Repository Setup

1. Create a new repository on GitHub.
2. Link local repo with remote:
3. git remote add origin https://github.com/vickiee07/CI-CD-Pipeline-with-GitHub-Actions-Docker.git
4. git branch -M main
5. git push -u origin main

5. GitHub Actions Workflow

1. Create workflow file:
2. .github/workflows/ci-cd.yml
3. Add pipeline stages:
    - Checkout code
    - Install dependencies
    - Run tests
    - Build Docker image
    - Push Docker image to Docker Hub

6. GitHub Secrets Configuration

In **GitHub → Repo → Settings → Secrets → Actions**, add:

- DOCKERHUB_USERNAME → Your Docker Hub username
- DOCKERHUB_TOKEN → Access token from Docker Hub

7. CI/CD Execution

1. Push code to main branch:
2. git add .
3. git commit -m "Initial commit"
4. git push origin main
5. GitHub Actions automatically runs pipeline:
    - ✅ Checkout
    - ✅ Install dependencies
    - ✅ Run tests
    - ✅ Build Docker image
    - ✅ Push to Docker Hub

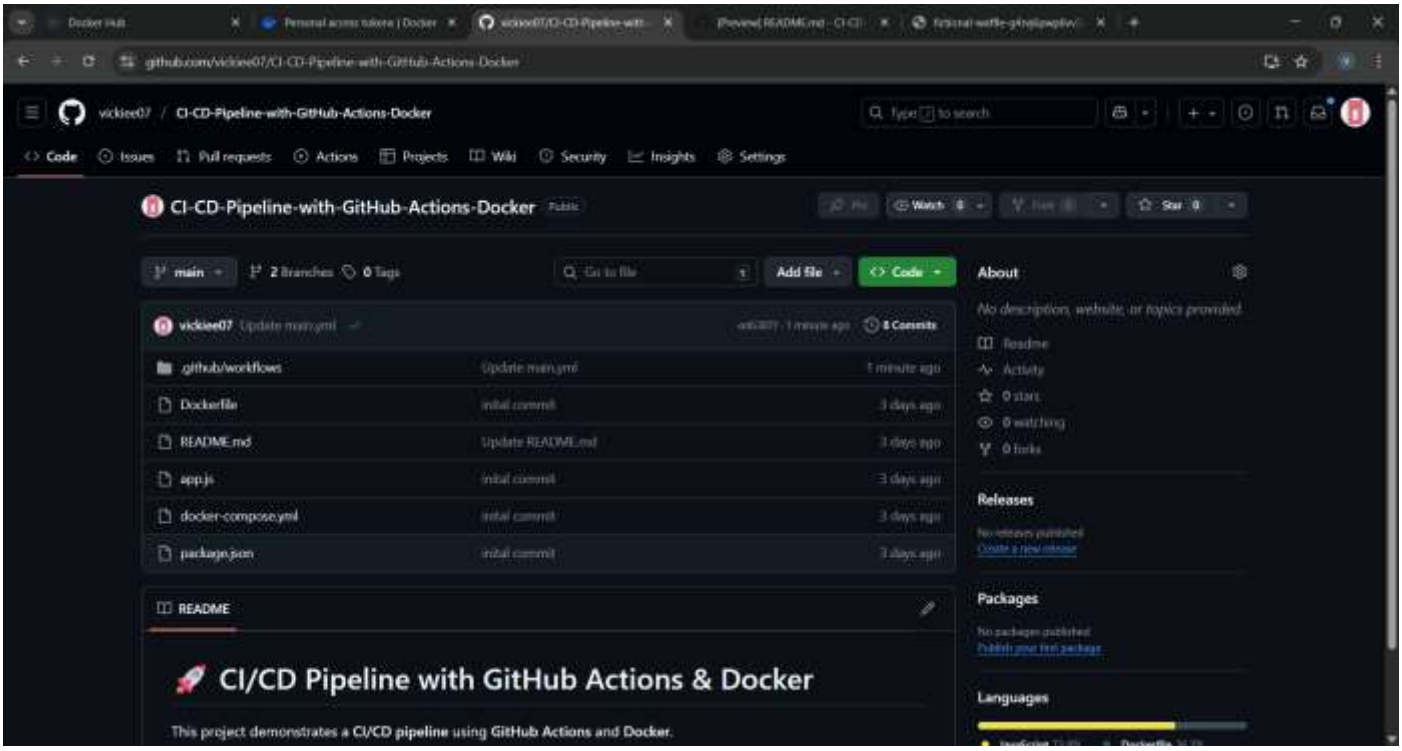
8. Deployment

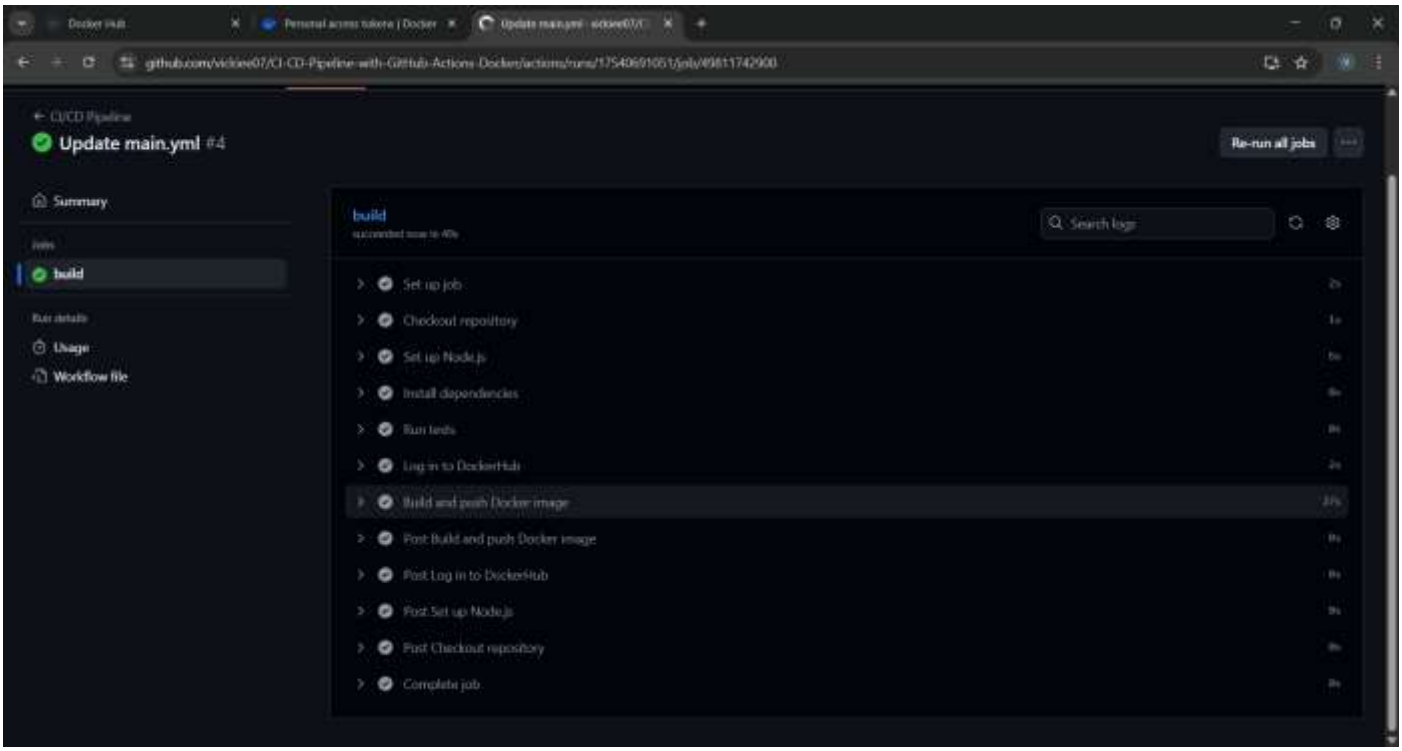
1. Pull image on deployment server:
2. docker pull your-dockerhub-username/cicd-docker-app:latest
3. Run with Docker Compose:
4. docker-compose up -d
5. Access deployed app at:
6. http://localhost:3000

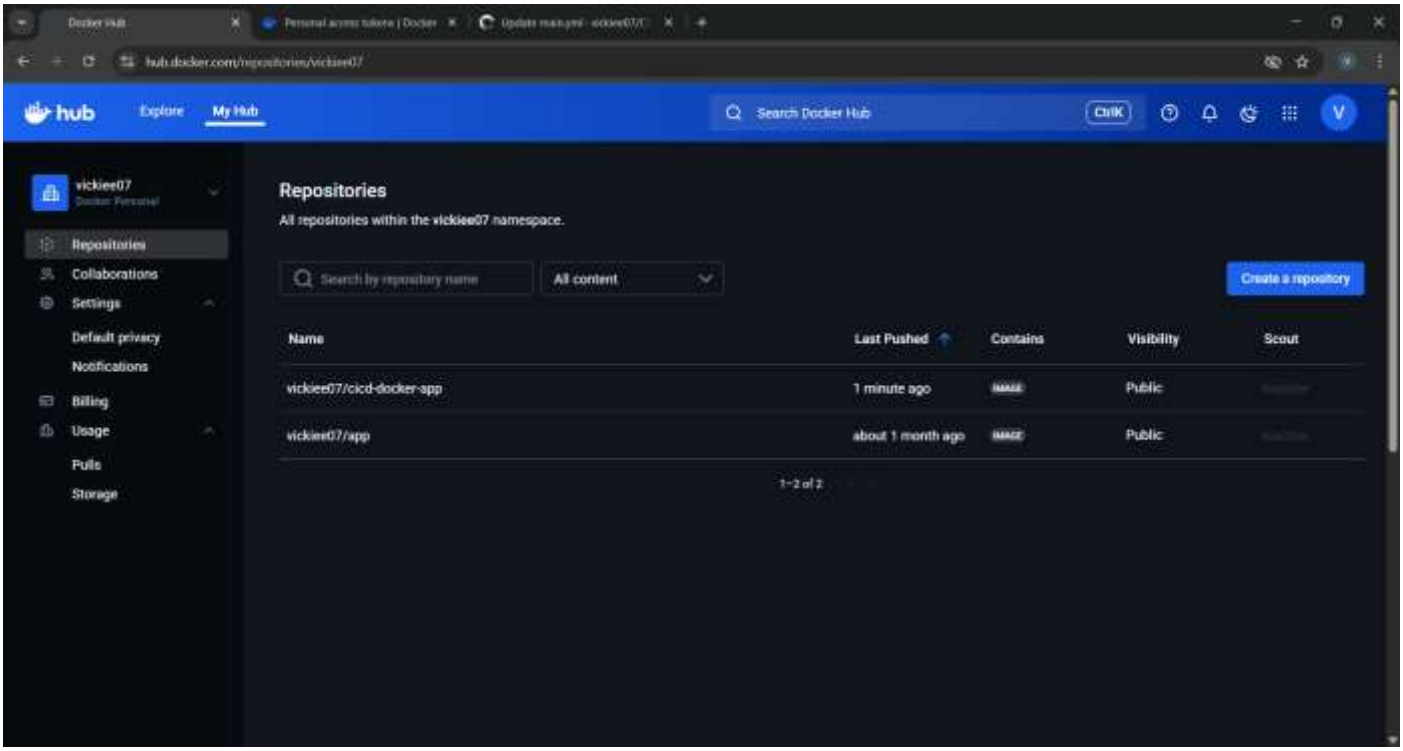
9. Verification

- Confirm that GitHub Actions workflow passes successfully.
- Verify app output in browser → **Hello from CI/CD Pipeline with GitHub Actions & Docker!**

OUTPUT


vickiee07 / CI-CD-Pipeline-with-GitHub-Actions-Docker
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
CI-CD-Pipeline-with-GitHub-Actions-Docker
main
2 Branches
0 Tags
Add file
Code
About
No description, website, or topics provided.
Readme
Activity
0 stars
0 watching
0 forks
Releases
Packages
Languages
.github/workflows
Dockerfile
README.md
app.js
docker-compose.yml
package.json
README
CI/CD Pipeline with GitHub Actions & Docker
This project demonstrates a CI/CD pipeline using GitHub Actions and Docker.


CI/CD Pipeline
Update main.yml #4
Re-run all jobs
Summary
build
Usage
Workflow file
build
Search logs
Set up job
Checkout repository
Set up Node.js
Install dependencies
Run tests
Log in to DockerHub
Build and push Docker image
Post Build and push Docker image
Post Log in to DockerHub
Post Set up Node.js
Post Checkout repository
Complete job


hub
Explore
My Hub
Search Docker Hub
Ctrl K
vickiee07
Repositories
Collaborations
Settings
Default privacy
Notifications
Billing
Usage
Pulls
Storage
Repositories
All repositories within the vickiee07 namespace.
Search by repository name
All content
Create a repository
Name
Last Pushed
Contains
Visibility
Scout
vickiee07/cicd-docker-app
1 minute ago
IMAGE
Public
vickiee07/app
about 1 month ago
IMAGE
Public
1–2 of 2


Hello from CI/CD Pipeline with GitHub Actions & Docker!

CONCLUSION

This project successfully demonstrates the implementation of a CI/CD pipeline using GitHub Actions and Docker. By automating the build, test, and deployment processes, the pipeline ensures faster delivery, improved consistency, and reduced manual effort.

The integration of Docker guarantees environment uniformity across development and production, while GitHub Actions provides seamless automation triggered by code changes. The project highlights the importance of DevOps practices in modern software engineering, enabling continuous integration, continuous deployment, and reliable application delivery.

Overall, this work establishes a strong foundation for extending the pipeline with advanced features such as automated testing, multi-environment deployment, Kubernetes orchestration, and monitoring solutions.