CI/CD Pipeline for Vue.js Application

Course Name: DevOps FOUNDATION

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1.	ANIKET JAT	EN22CS304010
2.	HIMANSHU SAHRAWAT	EN22CS304029
3.	SHRIYANSH GUHA	EN22CS303054
4.	YASHASVI BAKORE	EN22CS303059
5.	ROUNAK PATHEKAR	EN22CS303046

Group Name: 03D11 Project

Number: DO-29

Industry Mentor Name:

University Mentor Name: Prof. Shyam Patel

Academic Year: 2022-2026

Problem Statement

In many development workflows, building, testing, and deploying a Vue.js application manually is slow, error-prone, and unsustainable. Without automation, teams face delays, merge conflicts, and unstable releases due to manual processes. The lack of a continuous integration/continuous delivery (CI/CD) pipeline can lead to inconsistent builds and deployment failures, making it difficult to meet fast-moving business needs. This project addresses those issues by implementing an automated CI/CD workflow that ensures every code change is quickly built, tested, and prepared for deployment. Reliable CI/CD is now recognized as “a key component in producing reliable software,” and neglecting it “leads to problems that slow down work and lower code quality” .

Project Objectives

- **Automate the Development Workflow:** Set up a GitHub Actions pipeline to automatically install dependencies, run unit tests, and build the Vue.js application on each code commit.

- **Containerize the Application:** Use Docker to package the built Vue.js app into a container for consistent deployments across environments.

- **Ensure Fast, Reliable Releases:** Enable rapid, repeatable releases by integrating testing and deployment steps into the pipeline, reducing manual effort and human error.

- **Improve Code Quality and Collaboration:** Enforce automated testing (using Vitest) and continuous integration to catch defects early, thereby boosting code quality and team productivity.

- **Maintain Platform Independence:** Use Docker containers so that the application can run reliably on any platform or cloud environment, ensuring scalability and maintainability.

Scope of the Project

The scope of this project is to design and implement a CI/CD pipeline **specifically for the front-end Vue.js application**. The project covers: (1) setting up a GitHub repository with CI workflows, (2) creating automated build and test processes using Node.js and Vitest, (3) defining Docker configurations to containerize the application, and (4) documenting the architecture and workflow. External systems are not in scope; no third-party APIs or backend services are integrated. In particular: - **Included:** Vue.js application source code, GitHub Actions CI/CD workflows, Dockerfile for containerization, unit tests with Vitest, and related build/test artifacts. - **Excluded:** Backend services, external APIs, or infrastructure (beyond using Docker on any target host).
- **Constraints:** The solution uses GitHub Actions (free tier) and Docker Hub (or equivalent) for images, and assumes the project runs on standard Node.js tooling and Docker Engine.

Key Features

- **Continuous Integration Pipeline:** Automated GitHub Actions workflow named *Vue CI Pipeline* that runs on every commit. It installs packages, runs linting/tests, and compiles the Vue app without manual intervention.
- **Automated Testing:** Integration of Vitest for unit tests, ensuring code changes are vetted immediately. Passing all tests is a gate for successful builds.
- **Containerized Builds:** A multi-stage Dockerfile that first builds the app and then packages it into a lean production image. This makes releases portable and environment-agnostic.
- **Version Control Integration:** Uses Git and GitHub to manage source code, with branch protections requiring the CI pipeline to pass before merges. This enforces code quality and collaboration.
- **Clear Workflow and Logging:** Each step of the pipeline outputs logs (as seen in the screenshots) so developers can trace failures. Issues like build errors or test failures are caught early.
- **Platform Independence:** Since the app is in a Docker image, it can be deployed to any host running Docker (e.g., any cloud or server), guaranteeing consistent behavior.

Overall Architecture / Workflow

The system architecture is straightforward and linear, optimized for clarity and reliability. The logical flow is:

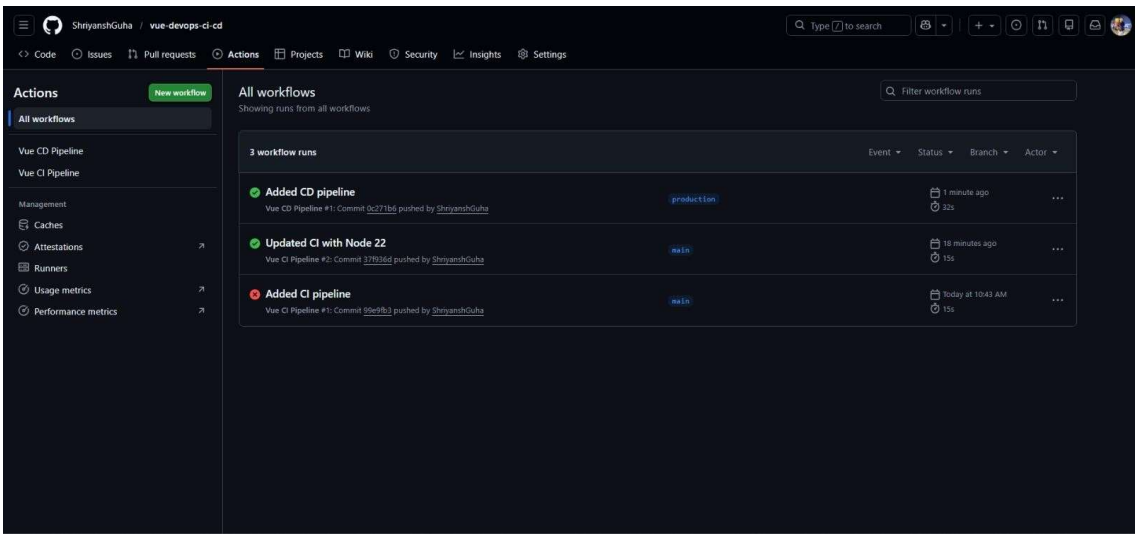
1. **Developer → GitHub Repository:** A developer pushes code changes to the repo (or creates a pull request).
2. **GitHub Actions CI Pipeline:** The *ci.yml* workflow kicks in, running automatically. It performs:
3. Dependency installation (npm install)
4. Code linting (eslint) and formatting checks
5. Unit testing with Vitest (runs files like *sample.test.js*)
6. Building the Vue.js application for production.
7. **Artifact Preparation:** Once built, the compiled app files (HTML/CSS/JS) are ready. The pipeline may also archive any test coverage reports or build logs for review.
8. **Docker/CD Pipeline:** A second workflow (*cd.yml*) is triggered (either on merge to main or as part of the same run). It executes the Dockerfile steps:
9. **Stage 1 (Build):** Uses a Node.js image to build the app.
10. **Stage 2 (Production):** Packages the built files into a lightweight container (e.g., using Nginx or node for static serving).
11. The resulting image is tagged (often with commit SHA or version) and pushed to a registry.
12. **Deployment:** The final Docker image can be deployed to target environments. For example, a server could pull the image and run it, instantly launching the new version of the Vue.js app. Docker ensures the application runs the same on the developer's machine, CI server, or production.

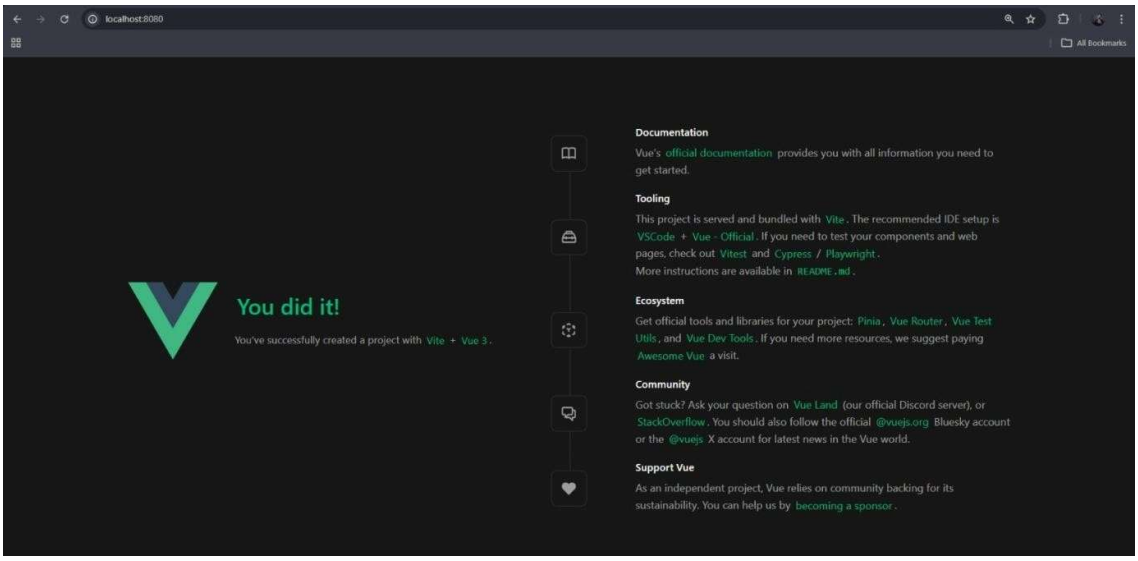
Developer → Git Repo → CI Pipeline → Test Suite → Build System → Docker Image.

Tools & Technologies Used

- **Vue.js:** Front-end JavaScript framework (v3) for building the user interface. The project was scaffolded using Vue's latest tools (Vite).
- **Git & GitHub:** Version control and hosting. The repository on GitHub stores all source code, along with workflow configuration files (.github/workflows).
- **GitHub Actions:** CI/CD platform integrated directly with the GitHub repo. Actions run YAML-defined workflows (as seen in the provided *ci.yml* and *cd.yml*). This automates the pipeline without needing external servers.
- **Node.js / npm:** Provides the runtime for building the Vue.js app. npm scripts (or pnpm) handle dependency management and build commands.
- **Vitest:** Testing framework for unit tests. Vitest is used to run *sample.test.js* and other specs during CI, ensuring functionality.
- **Docker:** Containerization tool. Docker is used to define a multi-stage *Dockerfile* that builds and then packages the app. This ensures the app runs consistently in any environment[4].

Results & Output





Conclusion

In this project, we successfully built and implemented a CI/CD pipeline for a Vue.js application. The main goal was to remove manual work in building, testing, and deploying the application. Before CI/CD, these tasks were done manually, which could cause errors and delays. After implementing automation, the process became faster, more reliable, and more organized.

Using GitHub Actions, the system now automatically installs dependencies, runs tests, and builds the project whenever new code is pushed. This helps in finding mistakes early and ensures that only correct code moves forward. Docker was used to package the application into a container, which makes sure the app runs the same way on every system.

Through this project, we learned:

- How CI/CD works in real-world software development
- How to create automated workflows using GitHub Actions
- How to run automated tests using Vitest
- How Docker helps in making applications portable and consistent
- The importance of automation in improving code quality and team productivity

Overall, this project shows how CI/CD makes development faster, reduces errors, improves collaboration, and ensures stable software releases.

Future Scope & Enhancements

1. Cloud Deployment

In the future, the application can be directly deployed to cloud platforms like AWS or Azure. This will make it more scalable and suitable for real-world production use.

2. More Testing

We can add advanced testing methods like:

- Integration testing
- End-to-end testing
- Code coverage reports

This will improve reliability and reduce bugs even more.

3. Security Improvements

Security features can be added such as:

- Automatic security scanning
- Checking for vulnerable dependencies
- Better management of secret keys

4. Monitoring and Logging

Monitoring tools can be added to track application performance after deployment. This will help detect issues quickly.

5. Multiple Environments

The pipeline can be improved to support different environments like Development, Testing, and Production. This will make the system more professional.

6. Advanced Deployment Methods

Techniques like Blue-Green deployment or Canary deployment can be used to reduce downtime during updates.