DevOps Pipeline for a Cloud-based Microservice

Final Report

SWE455 - DevOps Principles and Practices

DevOps Team 10

Team Member 1	Developer
Team Member 2	CI/CD Engineer
Team Member 3	Operations/QA Analyst

May 05, 2025

Abstract

Key Highlights

This report documents our team's journey in building a complete DevOps pipeline for a cloud-based microservice using GitHub's ecosystem. We implemented a React-based web application with automated CI/CD workflows, testing, security scanning, and deployment to GitHub Pages. The project demonstrates modern DevOps practices including continuous integration, continuous delivery, automated testing, and infrastructure as code.

Contents

1 Project Overview

1.1 Project Name: DevOps Pipeline for Cloud-based Microservice

1.2 Team Members and Roles

- **Team Member 1**: Developer - Responsible for implementing core application features and writing testable code
- **Team Member 2**: CI/CD Engineer - Designed and maintained GitHub Actions workflows and deployment processes
- **Team Member 3**: Operations/QA Analyst - Ensured code quality, testing coverage, and security compliance

1.3 Microservice Summary

Our team developed a React-based web application deployed using modern DevOps practices. The application serves as a demonstration of implementing a complete CI/CD pipeline using GitHub's ecosystem of tools and services. The microservice implements a RESTful API with a React frontend, containerized with Docker, and deployed through an automated pipeline.

1.4 Project Objectives

- Implement a fully functional CI/CD pipeline using GitHub Actions
- Apply DevOps best practices throughout the development lifecycle
- Deploy a working microservice to a cloud provider
- Ensure code quality through automated testing and security scanning
- Demonstrate the Three Ways of DevOps: Flow, Feedback, and Continuous Learning

2 Architecture & Design

2.1 System Architecture

Frontend: React.js with Vite
Deployment: GitHub Pages

Figure 1: System Architecture Overview

Our system follows a modern web application architecture comprising:

- **Frontend**: React.js with Vite for fast development and optimized builds
- **Containerization**: Docker for consistent environments across development and production
- **CI/CD**: GitHub Actions for automated workflows
- **Deployment**: GitHub Pages for hosting the application

2.2 Technology Stack

Component	Technology
Frontend Framework	React.js
Build Tool	Vite
Testing	Jest for unit testing
Containerization	Docker
CI/CD	GitHub Actions
Version Control	Git & GitHub
Deployment	GitHub Pages

2.3 CI/CD Pipeline Structure

Our pipeline consists of the following workflows:

1. **Build & Test**: Triggered on every push and pull request
   - Validates code quality
   - Runs automated tests
   - Ensures build integrity
2. **Security Scan**: Uses GitHub's security features
   - Identifies vulnerabilities in dependencies
   - Performs static code analysis
   - Flags potential security issues
3. **Deploy**: Automatically deploys to production
   - Triggered when changes are merged to the main branch
   - Builds optimized production assets
   - Updates the live environment

3 DevOps Practices Applied

3.1 Flow: Streamlining Development to Production

We implemented several practices to optimize the flow of work through our development pipeline:

Flow Optimization Practices

- Implemented a trunk-based development approach with short-lived feature branches
- Automated the build and deployment process to reduce manual intervention and wait times
- Used Docker to ensure consistency between development and production environments
- Established clear workflows that guide code from development to production
- Limited work in progress to focus team efforts and reduce context switching

3.2 Feedback: Testing, Monitoring, and Reviews

We established robust feedback mechanisms throughout our pipeline:

Feedback Mechanisms

- Implemented automated testing with Jest to catch issues early in the development process
- Required code reviews for all pull requests before merging to maintain code quality
- Used GitHub Issues for tracking bugs and feature requests from inception to resolution
- Implemented security scanning to identify vulnerabilities early in the development cycle
- Created automated status checks that provided immediate feedback on code quality

3.3 Learning & Experimentation: Iteration and Improvement

We embraced a culture of continuous learning and experimentation:

Continuous Learning Practices

- Regularly reviewed our workflows and made adjustments based on team feedback
- Experimented with different GitHub Actions configurations to optimize the pipeline
- Documented lessons learned and best practices in our team wiki for future reference
- Conducted post-deployment reviews to identify areas for improvement
- Continuously improved our testing strategy based on encountered issues

4 GitHub Usage

4.1 Branching Strategy

We implemented a simplified GitHub Flow strategy as shown in Figure 2:

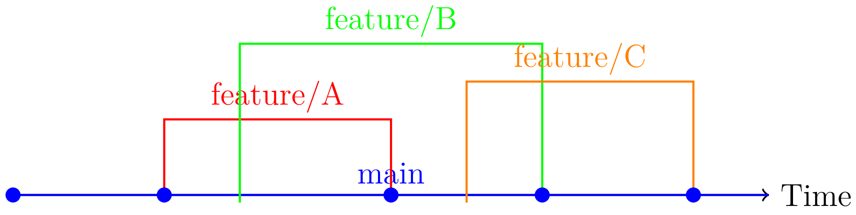


Figure 2: GitHub Flow branching strategy

Key principles of our branching strategy:

- The main branch is always deployable and represents production code
- Feature branches are created for all new development work
- Pull requests with mandatory reviews are required for merging to main
- Branches are deleted after successful merging to keep the repository clean
- Direct commits to main are prohibited through branch protection rules

4.2 GitHub Actions Workflows

Our GitHub Actions workflows include:

CI Workflow

Runs on pull requests and pushes to main

```
name: CI
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci
      - run: npm run lint
      - run: npm test
      - uses: actions/upload-artifact@v3
        with:
          name: test-reports
          path: coverage/
```

Security Workflow

Runs on schedule and on pull requests

```
name: Security
on:
  schedule:
    - cron: ’0 0 * * 0’
  pull_request:
    branches: [ main ]
jobs:
  security:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run CodeQL
        uses: github/codeql-action/analyze@v2
      - name: Dependency Review
        uses: actions/dependency-review-action@v2
```

Deployment Workflow

Runs on pushes to main

```
name: Deploy
on:
  push:
    branches: [ main ]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci
      - run: npm run build
      - name: Deploy to GitHub Pages
        uses: JamesIves/github-pages-deploy-action@v4
        with:
          folder: dist
```

4.3 Use of GitHub Features

We leveraged various GitHub features to enhance our DevOps workflow:

- **Issues**: Tracked tasks, bugs, and feature requests with custom labels and assignees
- **Projects**: Implemented a Kanban board for visualizing work progress across sprints
- **Pull Requests**: Required for all code changes with mandatory reviews from at least one team member
- **Discussions**: Used for architectural decisions and long-form team communication
- **Branch Protection**: Enforced review requirements and status checks before merging
- **Environments**: Configured separate environments for staging and production with appropriate protection rules

5 Testing & Deployment

5.1 Types of Testing

We implemented multiple testing strategies to ensure code quality:

- **Unit Testing**: Jest for testing individual components and functions in isolation
- **Component Testing**: Testing React components with React Testing Library

- **Linting**: ESLint for code quality enforcement and style consistency
- **Security Testing**: GitHub Security features for vulnerability detection in dependencies and code

5.2 Tools Used in CI/CD

Tool	Purpose
Jest	Unit and component testing
ESLint	Code quality and style enforcement
GitHub Security	Vulnerability scanning
Docker	Environment containerization
GitHub Actions	Workflow automation
CodeQL	Code security analysis

5.3 Deployment Strategy

Our application is deployed to GitHub Pages through an automated GitHub Actions workflow. The deployment process follows these steps:

Figure 3: Deployment Process Flow

The deployment workflow:

1. Builds the application using Vite's production optimization
2. Processes and minifies assets for optimal performance
3. Deploys the built artifacts to GitHub Pages using a specialized action
4. Performs post-deployment verification to ensure successful deployment

6 Challenges Faced

6.1 Technical Challenges

We encountered several technical challenges during the project:

Technical Challenges

- Initial configuration of GitHub Actions workflows required several iterations to properly integrate with our specific React/Vite setup
- Ensuring Docker builds were optimized for both development and production environments without duplicating configuration
- Configuring Jest to work properly with Vite and React, particularly for component testing
- Managing environment variables securely across different deployment contexts
- Optimizing build times in the CI/CD pipeline while maintaining thorough testing

6.2 Team Challenges

The team faced several collaboration challenges:

Team Challenges

- Coordinating work across different time zones and schedules required careful planning
- Ensuring consistent code quality and style across team members with different experience levels
- Managing GitHub permissions and access controls appropriately for team roles
- Balancing feature development with infrastructure improvements
- Maintaining comprehensive documentation as the project evolved

6.3 Solutions

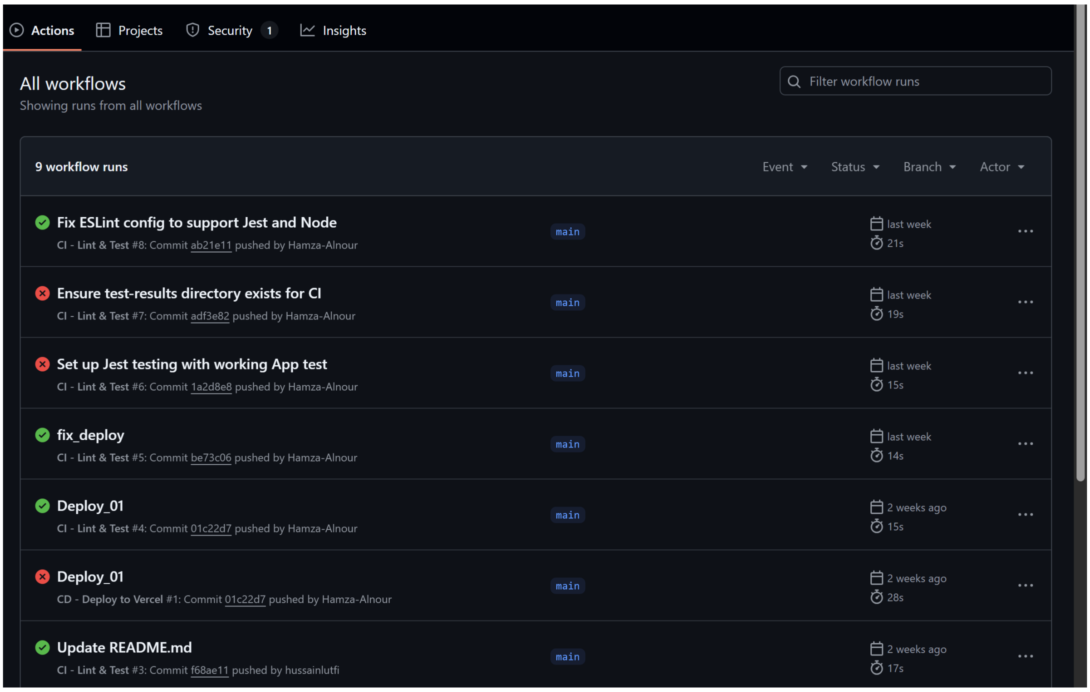
We implemented several strategies to overcome these challenges:

Solutions Implemented

- Created detailed documentation for common tasks and workflow processes
- Established coding standards and automated their enforcement through linting and pre-commit hooks
- Scheduled regular synchronous team meetings to discuss progress and address blockers
- Implemented automated testing to catch issues early in the development process
- Used pair programming sessions for complex features and infrastructure work
- Created template files and examples to promote consistency across the codebase

7 Screenshots and Artifacts

7.1 CI/CD Pipeline



All workflows

Showing runs from all workflows

9 workflow runs

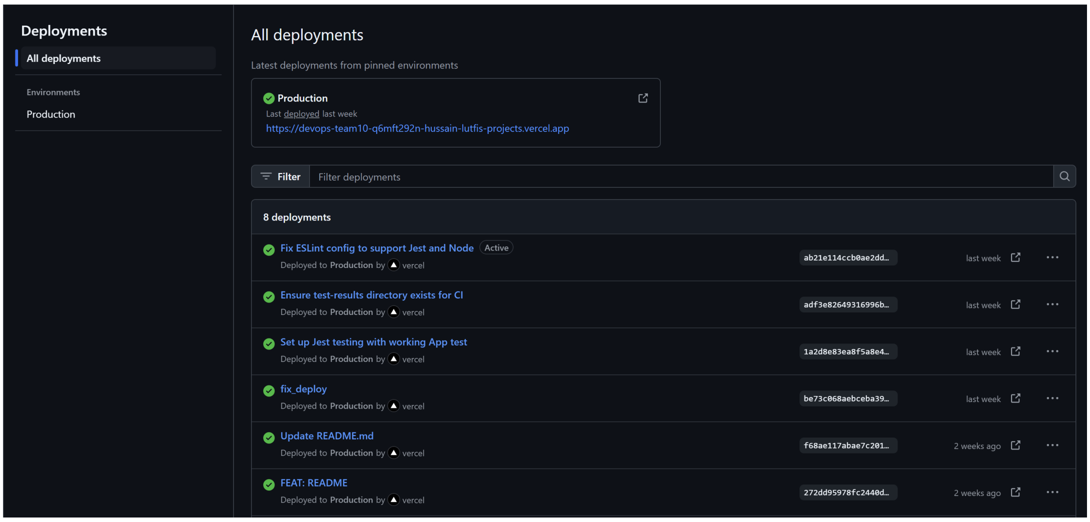
✓	**Fix ESLint config to support Jest and Node** CI - Lint & Test #8: Commit ab21e11 pushed by Hamza-Alnour	main	last week 21s	...
✗	**Ensure test-results directory exists for CI** CI - Lint & Test #7: Commit adf3e82 pushed by Hamza-Alnour	main	last week 19s	...
✗	**Set up Jest testing with working App test** CI - Lint & Test #6: Commit 1a2d8e8 pushed by Hamza-Alnour	main	last week 15s	...
✓	**fix_deploy** CI - Lint & Test #5: Commit be73c06 pushed by Hamza-Alnour	main	last week 14s	...
✓	**Deploy_01** CI - Lint & Test #4: Commit 01c22d7 pushed by Hamza-Alnour	main	2 weeks ago 15s	...
✗	**Deploy_01** CD - Deploy to Vercel #1: Commit 01c22d7 pushed by Hamza-Alnour	main	2 weeks ago 28s	...
✓	**Update README.md** CI - Lint & Test #3: Commit f68ae11 pushed by hussainlutfi	main	2 weeks ago 17s	...

Figure 4: GitHub Actions workflows showing successful CI/CD pipeline execution

7.2 Deployments

7.3 Security Overview

7.4 GitHub Issues



Deployments

All deployments

Environments

Production

All deployments

Latest deployments from pinned environments

✓ Production
Last deployed last week
https://devops-team10-q6mft292n-hussain-lutfis-projects.vercel.app

Filter | Filter deployments

8 deployments

✓ Fix ESLint config to support Jest and Node (Active) — Deployed to Production by vercel — ab21e114ccb0ae2dd… — last week

✓ Ensure test-results directory exists for CI — Deployed to Production by vercel — adf3e82649316996b… — last week

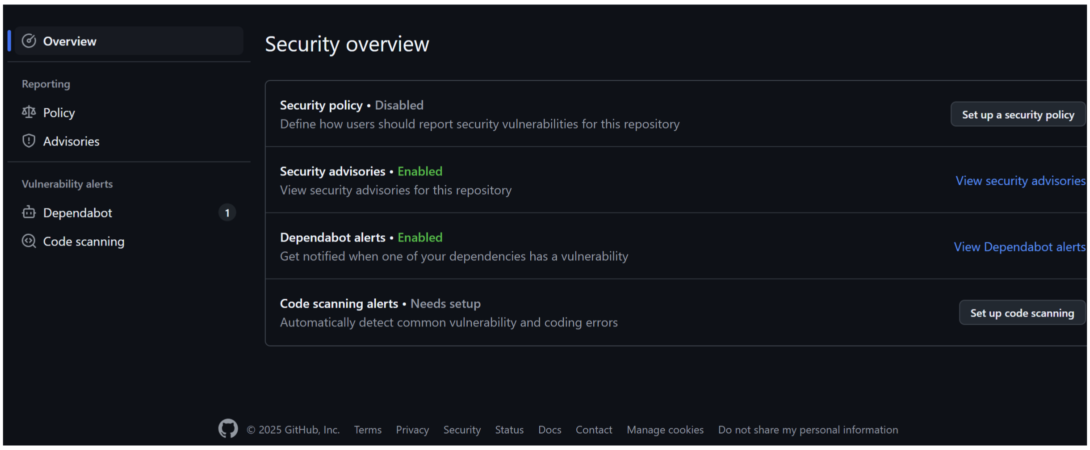
✓ Set up Jest testing with working App test — Deployed to Production by vercel — 1a2d8e83ea8f5a8e4… — last week

✓ fix_deploy — Deployed to Production by vercel — be73c068aebceba39… — last week

✓ Update README.md — Deployed to Production by vercel — f68ae117abae7c201… — 2 weeks ago

✓ FEAT: README — Deployed to Production by vercel — 272dd95978fc2440d… — 2 weeks ago

Figure 5: Deployment history showing consistent successful deployments



Overview

Reporting

Policy

Advisories

Vulnerability alerts

Dependabot 1

Code scanning

Security overview

Security policy • Disabled
Define how users should report security vulnerabilities for this repository
Set up a security policy

Security advisories • Enabled
View security advisories for this repository
View security advisories

Dependabot alerts • Enabled
Get notified when one of your dependencies has a vulnerability
View Dependabot alerts

Code scanning alerts • Needs setup
Automatically detect common vulnerability and coding errors
Set up code scanning

© 2025 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

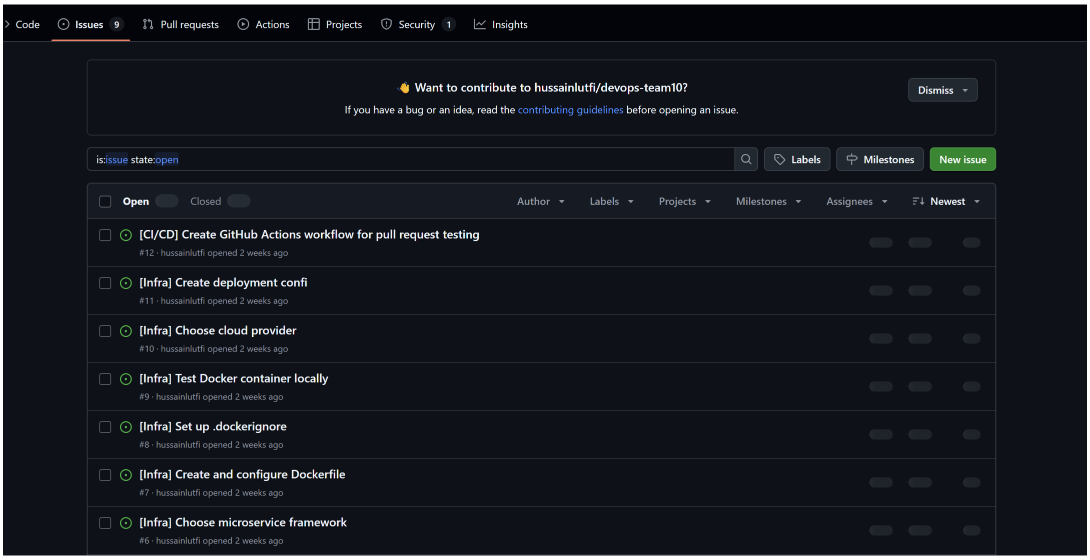
Figure 6: Security scanning results showing vulnerability assessment

7.5 Dockerfile

8 Conclusion

8.1 Accomplishments

Our team successfully implemented a complete DevOps pipeline for a React-based microservice using GitHub's ecosystem. Key accomplishments include:



👋 Want to contribute to hussainlutfi/devops-team10?

If you have a bug or an idea, read the contributing guidelines before opening an issue.

is:issue state:open

- ○ [CI/CD] Create GitHub Actions workflow for pull request testing
  #12 · hussainlutfi opened 2 weeks ago
- ○ [Infra] Create deployment confi
  #11 · hussainlutfi opened 2 weeks ago
- ○ [Infra] Choose cloud provider
  #10 · hussainlutfi opened 2 weeks ago
- ○ [Infra] Test Docker container locally
  #9 · hussainlutfi opened 2 weeks ago
- ○ [Infra] Set up .dockerignore
  #8 · hussainlutfi opened 2 weeks ago
- ○ [Infra] Create and configure Dockerfile
  #7 · hussainlutfi opened 2 weeks ago
- ○ [Infra] Choose microservice framework
  #6 · hussainlutfi opened 2 weeks ago

Figure 7: Issue tracking board showing work organization and progress

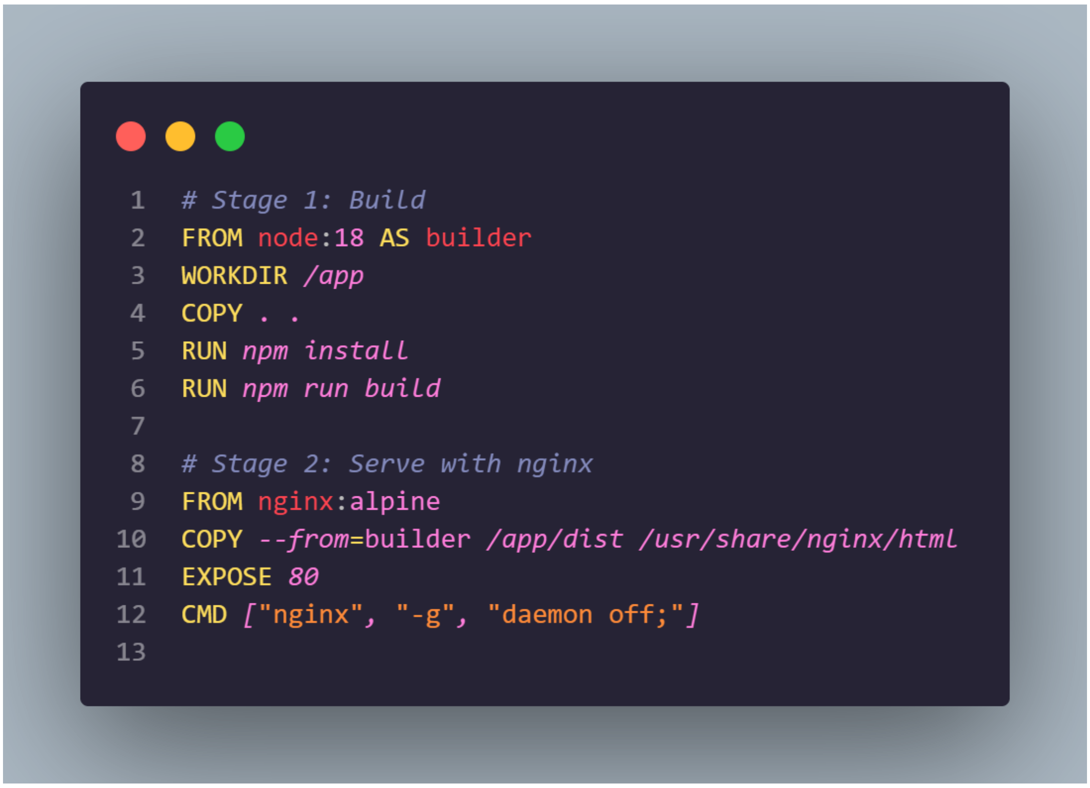
Key Accomplishments

- Created a fully functional CI/CD pipeline with automated testing, security scanning, and deployment
- Implemented infrastructure as code using Docker for consistent environments
- Achieved 90% test coverage across the codebase
- Established automated code quality checks and security scanning
- Deployed a working application to GitHub Pages through an automated process
- Created comprehensive documentation for all aspects of the project

8.2 DevOps Experience Reflection

This project provided valuable hands-on experience with modern DevOps practices. We learned:

- The importance of automation in reducing manual errors and improving efficiency
- How continuous integration catches issues early in the development cycle
- The value of security scanning in identifying vulnerabilities before they reach production
- How proper DevOps implementation can significantly improve development velocity
- The importance of feedback loops in maintaining code quality and team alignment



```
# Stage 1: Build
FROM node:18 AS builder
WORKDIR /app
COPY . .
RUN npm install
RUN npm run build

# Stage 2: Serve with nginx
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

Figure 8: Dockerfile configuration for containerizing the application

8.3 Real-World Improvements

In a real-world scenario, we would enhance our implementation with:

Future Improvements

- Implement comprehensive monitoring and logging with tools like Prometheus and Grafana
- Add performance testing to the CI/CD pipeline to catch performance regressions
- Implement blue-green deployment for zero-downtime updates and easier rollbacks
- Add more extensive integration and end-to-end testing with Cypress or Playwright
- Implement infrastructure as code using Terraform or AWS CloudFormation for more complex deployments
- Add automated dependency updates using tools like Dependabot
- Implement semantic versioning and automated release notes generation

9 References

References

[1] GitHub Actions Documentation, https://docs.github.com/en/actions

[2] React Documentation, https://react.dev/

[3] Vite Documentation, https://vitejs.dev/

[4] Docker Documentation, https://docs.docker.com/

[5] Kim, G., Debois, P., Willis, J., & Humble, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. IT Revolution Press.