

CI/CD Pipeline with Git, GitHub Actions and EC2

Assignment Report - DevOps CS423

Course Code: CS423

Assignment: #3 **Members:** Hamza Faraz, Muhammad Taimoor

Registration Numbers: 2022661, 2022661

Instructor: Muhammad Sajid Ali

A. Introduction

Project Overview

This project is a full-stack web application built with React (frontend) and Node.js/Express (backend) that demonstrates a shopping cart system. The application allows users to view shopping cart items through a RESTful API. The project serves as a practical demonstration of modern web development practices and DevOps principles.

B. Challenges Faced During Deployment

Challenge 1: Environment Configuration and Secrets Management

Description: Setting up secure access to AWS EC2 instances and configuring GitHub secrets required careful management of SSH keys and credentials.

Solution: Used GitHub Secrets to store sensitive information including SSH keys, EC2 host addresses, and email credentials. This ensures secure access without exposing credentials in the workflow files.

Impact: Enabled secure automated deployments without compromising security.

Challenge 2: Branch Management in Deployment Scripts

Description: The QA deployment workflow needed to handle both pull request branches and the main branch correctly, requiring dynamic branch detection.

Solution: Implemented conditional logic using GitHub context variables (`github.head_ref` for PRs and `github.ref_name` for direct pushes) to determine the correct branch to deploy.

Impact: Allowed flexible deployment of feature branches to QA environment for testing before merging to main.

Challenge 3: Node.js Version Compatibility

Description: Ensuring consistent Node.js versions across development, CI/CD pipeline, and deployment servers to avoid compatibility issues.

Solution: Explicitly specified Node.js version (22.x) in GitHub Actions workflow and ensured NVM is properly configured on deployment servers.

Impact: Eliminated version-related build and runtime errors.

Challenge 4: Repository Cloning with Private Repositories

Description: Initial deployment attempts failed when trying to clone private repositories on the deployment server due to authentication issues.

Solution: Used GitHub Personal Access Token (PAT) stored as a secret and included it in the git clone URL for production deployment workflow.

Impact: Enabled successful automated cloning of private repositories on deployment servers.

Challenge 5: Process Management with PM2

Description: Ensuring the application restarts correctly on deployment, handling both first-time deployment and subsequent updates.

Solution: Used conditional PM2 commands (`pm2 restart app || pm2 start index.js --name "app"`) to handle both scenarios gracefully.

Impact: Ensured zero-downtime deployments and proper application lifecycle management.

Detailed Workflow Breakdown


QA/Testing Deployment Workflow (`qa-deploy.yml`)

Triggers:

- Pull requests targeting the `main` branch
- Manual workflow dispatch

Steps:

1. Build and Test Job

- Checkout repository code
- Setup Node.js 22.x environment
- Install dependencies with  `npm ci`
- Build React application

- Execute test suite
- Run ESLint for code quality checks
- Send failure email notification if any step fails

2. **Deploy to QA Job** (runs only if build-and-test succeeds)

- SSH to AWS QA/Testing server
- Navigate to home directory
- Clone or update repository
- Checkout the PR branch or specified branch
- Install dependencies
- Build React application
- Restart/Start application with PM2
- Send success email notification with access URL

Production/Staging Deployment Workflow (`prod-deploy.yml`)

Triggers:

- Push to `main` branch (after merge)
- Manual workflow dispatch

Steps:

1. **Build and Test Job**

- Checkout repository code
- Setup Node.js 22.x environment
- Install dependencies with `npm ci`
- Build React application for production
- Execute comprehensive test suite
- Run ESLint for code quality checks
- Send failure email notification if any step fails

2. **Deploy to Staging Job** (runs only if build-and-test succeeds)

- Load NVM environment on deployment server
- SSH to AWS Staging server
- Navigate to home directory
- Clone or update repository (using PAT for private repos)
- Checkout main branch
- Install dependencies
- Build React application for production
- Restart/Start application with PM2
- Send success email notification with staging URL

C. Complete Flow Diagram

The following diagram illustrates the complete CI/CD pipeline flow from code commit to deployment:

```
graph TD
    A[Developer Pushes Code/PR] --> B{Trigger Type?}
```

```
B -->|Pull Request| C[QA Workflow]
B -->|Push to Main| D[Production Workflow]

C --> E[Build & Test Job]
D --> E

E --> F[Checkout Code]
F --> G[Setup Node.js 22.x]
G --> H[Install Dependencies]
H --> I[Build React App]
I --> J[Run Tests]
J --> K[Run Linting]

K --> L{Tests Pass?}
L -->|No| M[Send Failure Email]
L -->|Yes| N[Deploy Job]

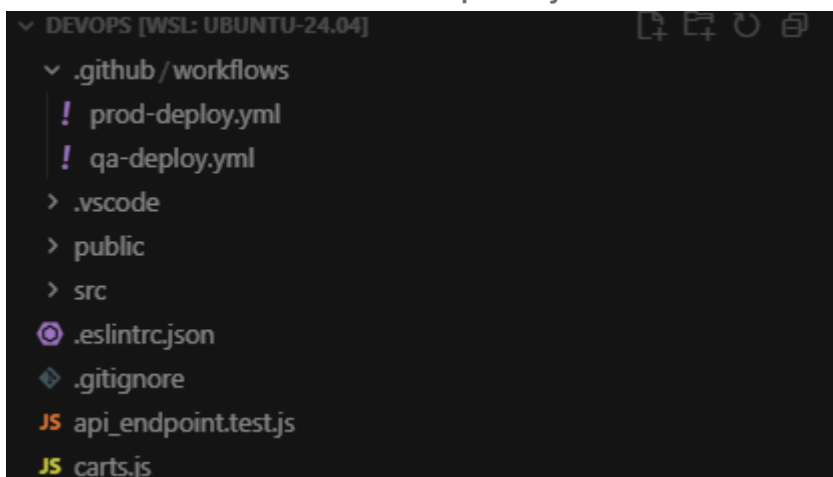
N --> O[SSH to Server]
O --> P[Clone/Update Repo]
P --> Q[Build & Start with PM2]
Q --> R[Send Success Email]
```

Description: The pipeline automatically triggers on pull requests (QA) or pushes to main (Production). After successful build and tests, the code is deployed to the appropriate server and notifications are sent.

D. Screenshots and Documentation

Screenshot 1: GitHub Actions Workflows

Screenshot 1.1: Workflow Files in Repository



Caption: GitHub Actions workflow files showing QA and Production deployment configurations.

Screenshot 2: Successful Workflow Execution

Screenshot 2.1: Complete Workflow Run

← Deploy to QA/Testing

✔ Feature/my feature #28

Summary

Jobs

Run details

✔ build-and-test

✔ deploy-qa

Usage

Workflow file

Triggered via pull request 20 hours ago

muhammadtaimoor9583 opened #3 feature/my-feature

Status

Success

Total duration

58s

Artifacts

-

qa-deploy.yml

on: pull_request

✔ build-and-test26s

→

✔ deploy-qa24s

← Deploy to Staging

✔ Merge pull request #3 from hash-walker/feature/my-feature #31

Summary

Jobs

Run details

✔ build-and-test

✔ deploy-staging

Usage

Workflow file

Triggered via push 20 hours ago

hash-walker pushed → f5cf669 main

Status

Success

Total duration

1m 10s

Artifacts

-

prod-deploy.yml

on: push

✔ build-and-test36s

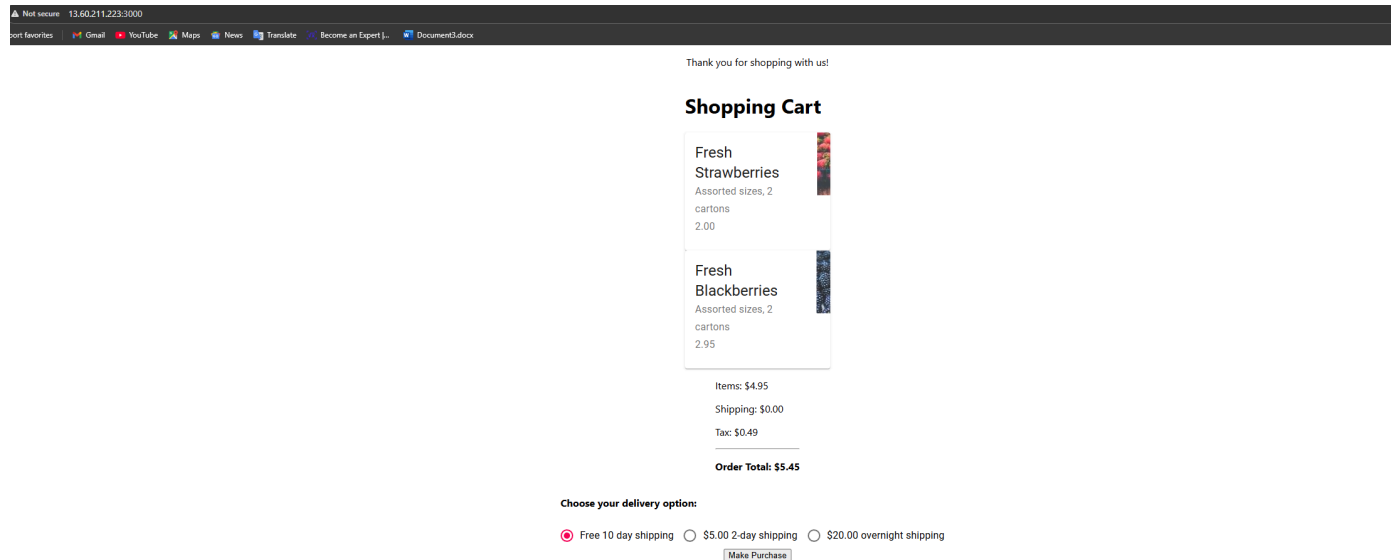
→

✔ deploy-staging27s

Caption: Successful workflow execution showing build, test, and deployment jobs completed successfully.

Screenshot 4: Deployment Verification

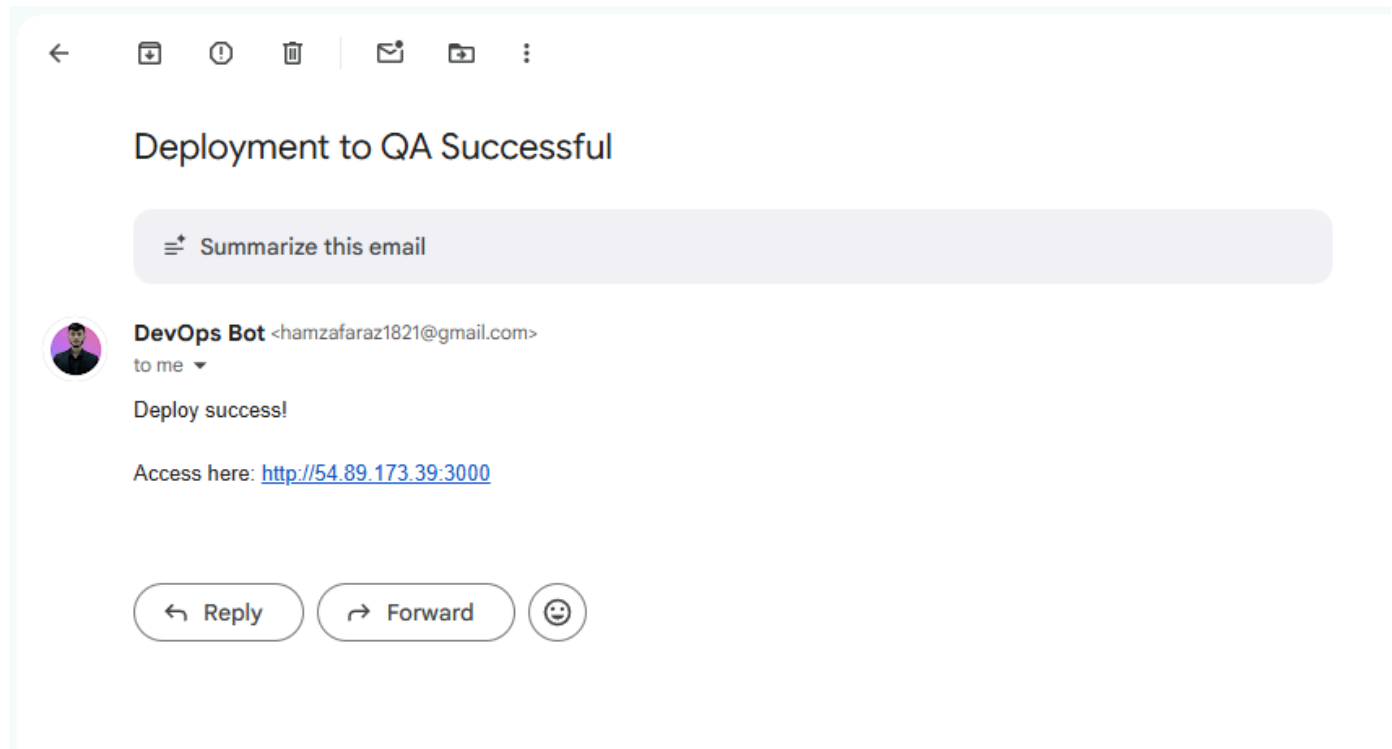
Screenshot 4.1: Application Running on Server

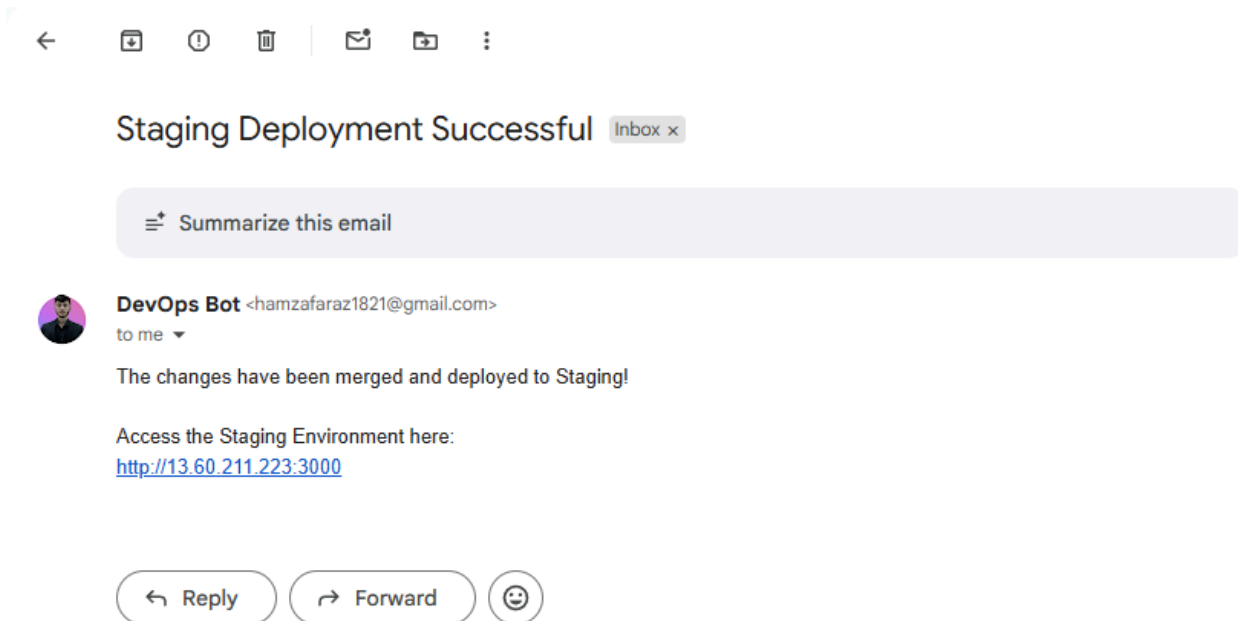


Caption: Application successfully deployed and running on the server, accessible via HTTP.

Screenshot 5: Email Notification

Screenshot 5.1: Success Email Notification





Caption: Automated email notification confirming successful deployment with access URL.

Conclusion

This assignment successfully demonstrates the implementation of a complete CI/CD pipeline using GitHub Actions. The solution includes:

- **Automated testing and code quality checks** ensuring reliability before deployment
- **Multi-environment deployment** supporting both QA and Production workflows
- **Comprehensive error handling** with automated email notifications
- **Infrastructure as Code** using Docker for consistent deployment environments
- **Process management** using PM2 for application lifecycle management
- **Security best practices** through GitHub Secrets management

The workflows are fully automated and trigger on appropriate events (pull requests, pushes to main, or manual dispatch), enabling efficient development and deployment practices. The separation of QA and Production environments ensures proper testing before production releases.

Appendix

Workflow Files Location

- QA Deployment: `.github/workflows/qa-deploy.yml`
- Production Deployment: `.github/workflows/prod-deploy.yml`

Repository Information

- **Repository URL:** <https://github.com/hash-walker/devops>
- **Main Branch:** `main`
- **Workflow Files:** `.github/workflows/`

End of Report