Continuous Integration and Deployment for the Parking System Application for

Master of Science

Information Technology

Bria Wright

University of Denver University College

May 18, 2025

Faculty: Nathan Braun MBA, BBA

Dean: Michael J. McGuire, MLS

Table of Contents

Introduction 3

System Setup 3

Tools and Technologies 3

CI/CD Pipeline Configuration 4

Infrastructure Requirements 6

Developer Interaction 6

Typical Workflow 7

Feedback Loops 8

Outputs and Reporting 8

Ensuring Quality and Stability 9

Monitoring and Logging 10

Conclusion 10

References 12

Introduction

Continuous Integration and Continuous Deployment (CI/CD) are essential to effective software development today. They equip teams to build, test, and deploy applications with impressive frequency, reliability, and efficiency. This paper presents a definitive CI/CD strategy for the Parking System Application, harnessing the capabilities of powerful tools like GitHub Actions and Jenkins. It demonstrates how these technologies integrate seamlessly to optimize development workflows and elevate software quality, ensuring a competitive advantage in today's dynamic digital landscape.

System Setup

Tools and Technologies

To implement Continuous Integration and Continuous Deployment (CI/CD) for the Parking System Application, I will utilize the following tools (Taiwo 2024):

- GitHub for source code management and version control.
- GitHub Actions or Jenkins for CI/CD pipelines.
- JUnit 5 for unit testing.
- Maven for build automation and dependency management.
- Docker for packaging the application in containers.
- AWS EC2 or Heroku for cloud-based deployment.
- SonarQube for static code analysis and quality gates.

As noted by Servile (2024, chap. 1), setting up a fast and repeatable deployment pipeline is essential for continuous delivery, as it enables safer releases and fosters development driven by feedback.

CI/CD Pipeline Configuration

A well-structured CI/CD pipeline enables automated integration, testing, and deployment of software, reducing manual effort and increasing system reliability. The Parking System follows a three-stage CI/CD workflow: version control, build automation, and deployment.

The diagram below illustrates the relationship between these stages. CI encompasses automated builds and tests, while CD incorporates staging and acceptance testing. Continuous Deployment takes this a step further by automatically deploying updates to production without requiring manual approval.

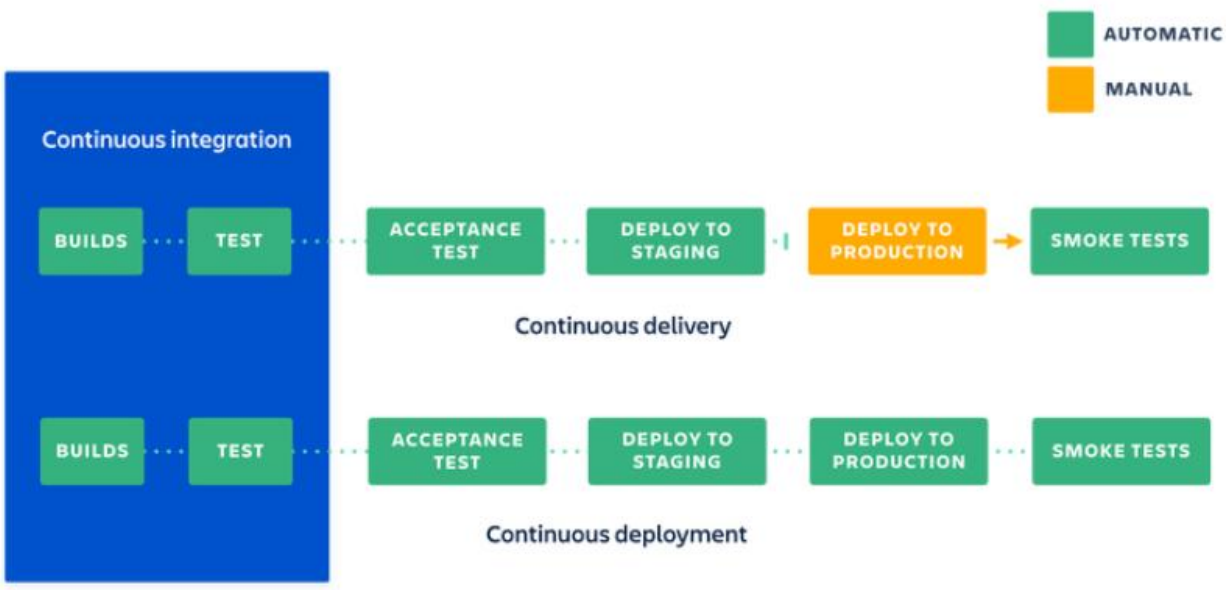

Figure 1. Comparison between Continuous Delivery and Continuous Deployment (Source: GeeksforGeeks, 2024)

As shown in Figure 1, the primary distinction between Continuous Delivery and Continuous Deployment is the method of deployment to production: one is manual, and the other is automatic. This distinction is a critical decision point that depends on project requirements and risk tolerance.

1. Version Control

   All source code is maintained in a GitHub repository. Developers work in feature branches, and GitHub's branch protection rules ensure that no code is merged into the main branch unless it passes all automated checks. This setup supports collaborative development while preserving the stability of the main branch (Servile 2024, chap. 2).

2. Build Automation

   To ensure builds are consistent and testable, a GitHub Actions workflow file (.github/workflows/ci.yml) is configured to run on every code push or pull request. In a typical CI/CD pipeline, each code commit triggers an automated process of build, test, and deployment steps. This ensures consistency and minimizes the risk of human error during deployment (GeeksforGeeks 2021). The build automation process includes the following steps:

   - Checking out the latest code from the repository
   - Installing all necessary dependencies using Maven (mvn install)
   - Executing unit tests with JUnit (mvn test)
   - Performing static code analysis using SonarQube or SpotBugs

   These steps are triggered automatically and provide fast feedback to developers, reducing the likelihood of introducing regressions into the codebase (Servile 2024, chap. 3).

3. Deployment

   A separate deployment workflow (cd.yml) is triggered automatically upon a successful merge into the main branch. This workflow carries out the following actions:

- Builds a Docker image of the application
- Pushes the image to Docker Hub
- Deploys the image to a cloud provider such as AWS EC2 or Heroku
- Sends a deployment notification through Slack or email

This automated deployment ensures that the most recent, tested version of the application is made available in a live environment without manual intervention (Servile 2024, chap. 4).

Infrastructure Requirements

To support the CI/CD pipeline, several infrastructure components must be in place. These include:

- A GitHub repository configured with CI/CD workflow files (ci.yml and cd.yml)
- An instance of SonarQube or a similar tool for static code analysis
- Access to a Docker registry, such as Docker Hub, to store and distribute application images
- A cloud hosting platform (e.g., AWS EC2 or Heroku) for deployment
- Secure credential management using GitHub secrets or Jenkins credentials (API keys, SSH keys, etc.)

These infrastructure elements are essential to ensure that the pipeline is secure, reproducible, and scalable (Servile 2024, chap. 1).

Developer Interaction

A CI/CD system offers developers ongoing feedback throughout the entire software development lifecycle. This section explains how developers interact with the system during code changes and how feedback and issues are managed.

Typical Workflow

1. Code Commit

   When the changes are ready, the developer submits a pull request (PR) to merge the feature branch into the main branch. This action automatically kicks off the CI workflow defined in GitHub Actions (“Building and Testing Java with Maven - GitHub Docs” 2025). During this workflow, several key steps are performed:

   - The entire project is built using Maven.
   - All unit and integration tests are run to ensure functionality.
   - Static code analysis is conducted for security and quality issues with tools like SonarQube.

2. Handling Failures

   If any part of the workflow fails, GitHub signals this by marking the pull request with a red (X) indicator. Developers are promptly alerted through integrated email notifications or Slack messages (“Configuring Notifications - GitHub Docs” 2025). To move forward, they need to address the issues and push the corrected code, which restarts the CI process.

3. Code Review and Merge

   After successful builds and test passes, team members review the code. Once approved,

the PR is merged into main. The entire process embodies Martin's (2006, chap. 4) "collective ownership" and commitment to clean code practices.

4. Continuous Deployment

   A successful main merge triggers the CD pipeline, which deploys the updated application to staging or production automatically. Servile (2024, chap. 4) recommends deploying in small batches to reduce risk and increase release reliability.

Feedback Loops

To foster continuous improvement and quick issue resolution, the CI/CD pipeline includes various feedback mechanisms:

- Build logs and test results are readily available on GitHub, enabling developers to pinpoint problems swiftly ("About Continuous Integration with GitHub Actions - GitHub Docs" 2024).
- SonarQube quality gates are integrated into pull requests to uphold code quality standards (Hicklen 2025).
- Deployment status updates are communicated to the team via Slack or email notifications (GeeksforGeeks 2021).

These feedback loops empower the development team to keep an eye on regressions, monitor code quality metrics, and tackle new issues with enhanced speed and accountability (Servile 2024, chap. 2–4).

Outputs and Reporting

Practical outputs and reporting mechanisms are essential for maintaining the quality, stability, and reliability of the Parking System Application. According to GeeksforGeeks (2021),

the CI stage allows for early bug identification, while CD provides automated quality assurance checks, enabling teams to detect and fix issues before reaching production.

Table 1. Expected Outputs at Each Stage

Stage	**Output**	**Purpose**
Build	.jar/Docker image	Artifact for deployment
Test	Test reports (JUnit XML or HTML)	Proves application correctness
Analysis	SonarQube metrics	Ensures code health and maintainability
Deployment	Live environment updated	Confirms successful release
Notifications	Slack/email alerts	Keeps team informed of system state

Ensuring Quality and Stability

The CI/CD pipeline generates several automated outputs that enhance the overall reliability of the application:

- Automated Testing: Unit and integration tests are executed with every build, allowing teams to catch bugs early in the development cycle—before the code reaches production (Humble and Farley 2010).
- Code Scanning Tools: Tools like SonarQube or SpotBugs analyze code quality, detect security vulnerabilities, enforce style consistency, and ensure long-term maintainability (Hicklen 2025).

- Deployment Automation: CI/CD workflows eliminate the need for manual deployment steps, reducing the risk of human error and enabling consistent, reproducible rollouts across different environments (GeeksforGeeks 2021).

As noted by Servile (2024, chap. 3), automating these outputs increases confidence in every release and prevents teams from delivering unstable or broken code to users.

Monitoring and Logging

To support proactive incident management and continuous performance improvement, the system incorporates various monitoring and logging tools:

- Log Tracking: Services like AWS CloudWatch or Heroku's LogDNA aggregate logs and capture runtime errors, as well as application behaviors ("AWS Services for Logging and Monitoring - AWS Prescriptive Guidance," n.d.).
- Performance Monitoring: Tools such as Prometheus and Grafana provide real-time visualization of system performance metrics, including memory usage, CPU load, and request times ("Monitoring with Prometheus and Grafana: A Complete Guide" 2024).
- Alerting: Custom alerts are configured for critical thresholds, such as high error rates, service downtime, or failed builds. These alerts notify the team through Slack, email, or monitoring dashboards ("Best Practices for Implementing Custom Alerts in API Management | APItoolkit" 2024).

Monitoring enhances operational visibility and provides a data-driven basis for improving application performance and quickly responding to issues, as recommended by Servile (2024, chap. 4).

Conclusion

Using a CI/CD pipeline is essential for modern software development teams that want to work quickly and maintain high quality. CI/CD automates the building, testing, and deployment of code, allowing teams to deliver updates more often and with greater confidence. Tools like GitHub Actions, SonarQube, Docker, and Maven help ensure that code is working properly, as well as secure and easy to maintain. Good feedback systems—like reports on test results and real-time notifications—create a continuous loop of information that helps teams fix issues faster and boosts productivity. Important tools, such as Docker registries, cloud services like AWS or Heroku, and secure management of sensitive information, are necessary for a safe and scalable deployment environment. CI/CD pipelines improve efficiency and promote teamwork, responsibility, and ongoing development. These practices are important for delivering high-quality software that meets users' needs in today's fast-paced world. By adopting these tools and principles, development teams can quickly adapt to changes, lower deployment risks, and maintain excellent software.

References

"About Continuous Integration with GitHub Actions - GitHub Docs." 2024. GitHub Docs. 2024. https://docs.github.com/en/actions/about-github-actions/about-continuous-integration-with-github-actions.

"AWS Services for Logging and Monitoring - AWS Prescriptive Guidance." n.d. Docs.aws.amazon.com. https://docs.aws.amazon.com/prescriptive-guidance/latest/logging-monitoring-for-application-owners/aws-services-logging-monitoring.html.

"Best Practices for Implementing Custom Alerts in API Management | APItoolkit." 2024. Apitoolkit.io. 2024. https://apitoolkit.io/blog/best-practices-for-implementing-custom-alerts/.

"Building and Testing Java with Maven - GitHub Docs." 2025. GitHub Docs. 2025. https://docs.github.com/en/actions/use-cases-and-examples/building-and-testing/building-and-testing-java-with-maven.

"Configuring Notifications - GitHub Docs." 2025. GitHub Docs. 2025. https://docs.github.com/en/account-and-profile/managing-subscriptions-and-notifications-on-github/setting-up-notifications/configuring-notifications.

GeeksforGeeks. 2021. "What Is CI/CD?" GeeksforGeeks. May 11, 2021. https://www.geeksforgeeks.org/what-is-ci-cd/#.

Hicklen, Hannah. 2025. “10 Best Static Code Analysis Tools.” @Clutch_co. Clutch. April 9, 2025. https://clutch.co/resources/10-best-static-code-analysis-tools#best-static-code-analysis-tools.

Martin, Micah, and Robert C. Martin. 2006. “Section I. Agile Development 4. Testing.” In *O’Reilly Online Learning*. Pearson. https://learning.oreilly.com/library/view/agile-principles-patterns/0131857258.

“Monitoring with Prometheus and Grafana: A Complete Guide.” 2024. Phantompixel.dev. Phantom Pixel - Programming, Software Development, and Networking. October 16, 2024. https://phantompixel.dev/posts/monitoring-with-prometheus-and-grafana-complete-guide-2024/.

Taiwo, Abatan. 2024. “Continuous Integration and Continuous Deployment (CI/CD) Pipelines Are Critical in Modern Software Development, Automating the Build, Test, and Deployment Processes to Ensure Efficiency, Reliability, and Consistency in Application Delivery. Let’s Dive Deeper into the Intricacies of a Jenkins-Based C.” Linkedin.com. July 7, 2024. https://www.linkedin.com/pulse/copy-comprehensive-cicd-pipeline-java-application-deployment-taiwo-jrkdf/.

Valentina Servile. 2024. *Continuous Deployment*. *O’Reilly Online Learning*. O’Reilly Media, Inc. https://learning.oreilly.com/library/view/continuous-deployment/9781098146719.

“What Is CI/CD?” 2021. GeeksforGeeks. May 11, 2021. https://www.geeksforgeeks.org/what-is-ci-cd/.