CI/CD Documentation

Introduction to CI/CD

Continuous Integration (CI) and Continuous Deployment/Delivery (CD) are practices aimed at improving software delivery speed and reliability. CI/CD automates building, testing, and deploying code, ensuring high-quality software while enabling frequent and consistent updates.

Modules of Testing

1. Unit Testing

Unit testing involves testing individual components or functions of the software in isolation to verify that they work as expected. Tools commonly used: JUnit, NUnit, PyTest.

2. Functional Testing

Functional testing focuses on testing the application's functionality against the requirements. It ensures that features perform as intended. Tools commonly used: Selenium, Cypress.

3. Regression Testing

Regression testing ensures that new changes or updates do not break existing functionality. Automated regression tests are typically part of the CI/CD pipeline.

Test Automation Concepts

1. Unit Testing

Automated unit tests validate individual components during the CI phase. They detect issues early in the development cycle.

2. Static Code Analysis

Static code analysis evaluates the source code without executing it, ensuring compliance with coding standards and detecting potential vulnerabilities. Tools: SonarQube, Checkstyle.

3. Code Quality

Code quality checks ensure the maintainability, readability, and efficiency of the codebase. Metrics such as code coverage and duplication are often monitored.

4. Automation Testing

Automated testing frameworks execute tests without manual intervention, increasing speed and consistency. Examples include Selenium for UI tests and JUnit for unit tests.

5. Reports

Automated reports provide insights into build results, code quality, and test outcomes. Tools like Jenkins provide visual dashboards for better analysis.

6. Deployment

Deployment automates the process of releasing code to staging or production environments. Strategies include rolling updates, blue-green deployments, and canary releases.

CI/CD Pipeline Flowchart

CI/CD Pipeline Flow

1. **Code Commit**: Developers push code to a version control system (e.g., Git).
2. **Build Trigger**: The CI server detects changes and initiates a build.
3. **Build**: Code is compiled, dependencies are resolved, and the application is packaged.
4. **Unit Testing**: Automated tests are executed to validate individual components.
5. **Code Analysis**: Static code analysis and code quality checks are performed.
6. **Integration Testing**: Ensures components interact correctly.
7. **Functional Testing**: Validates the application's functionality.
8. **Deployment**: Code is deployed to staging and/or production environments.
9. **Monitoring**: Application performance and errors are monitored post-deployment.

Start -> Code Commit -> Build Trigger -> Build -> Unit Testing -> Code Analysis -> Integration Testing -> Functional Testing -> Deployment -> Monitoring -> End

Key Features of a CI/CD Pipeline

- **Automation**: Streamlines repetitive tasks like building, testing, and deployment.
- **Scalability**: Supports large and distributed teams.
- **Feedback Loops**: Provides immediate feedback on code quality and test results.
- **Integration**: Supports integration with multiple tools and platforms.
- **Version Control**: Tracks changes and supports rollbacks.
- **Monitoring and Reporting**: Ensures visibility into pipeline performance and errors.

Build Triggers

Types of Build Triggers

1. **Poll SCM**:
    - Jenkins periodically checks the source control system (e.g., Git) for changes based on a predefined schedule.
    - If changes are detected, a build is triggered automatically.
    - Useful when webhook integration is unavailable or impractical.
2. **Build Periodically**:
    - Schedules builds to run at specific intervals, such as daily or weekly.
    - Defined using a cron-like syntax to specify exact timings.
    - Ideal for generating nightly builds or regular checks.
3. **Webhooks**:
    - An event-driven mechanism where the source control system notifies Jenkins about changes (e.g., a new commit or pull request).
    - Immediately triggers a build upon detecting an event, ensuring minimal delay.
    - Highly efficient as it eliminates the need for frequent polling.

Jenkins Plugins for CI/CD

Jenkins is a popular tool for implementing CI/CD pipelines. Below are essential plugins:

1. Git Plugin

Integrates Git repositories into Jenkins pipelines for source code management.

2. Pipeline Plugin

Defines complex CI/CD workflows using a script-based approach.

3. SonarQube Plugin

Facilitates static code analysis and code quality checks.

4. JUnit Plugin

Processes and visualizes unit test results within Jenkins.

5. Docker Plugin

Enables integration with Docker for containerized builds and deployments.

6. Email Extension Plugin

Sends email notifications based on pipeline events or build status.

7. Slack Plugin

Integrates Slack for build and pipeline notifications.

8. Blue Ocean Plugin

Provides an enhanced user interface for viewing and managing Jenkins pipelines.