CI/CD Automation with Jenkins

Nguyễn Thanh Quân - ntquan@fit.hcmus.edu.vn

Agenda

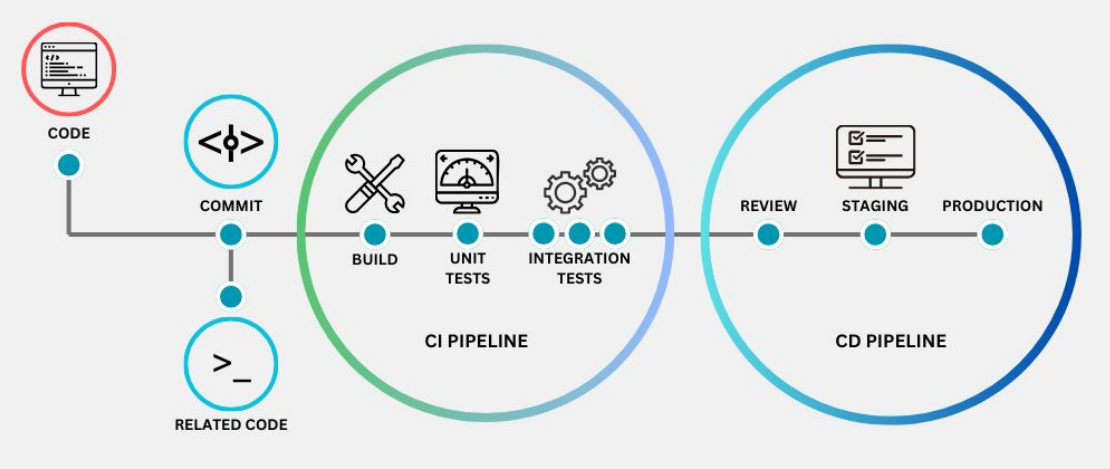
---

1. Overview
2. Introduction to Jenkins
3. Jenkins Architecture
4. Components of Jenkins
5. Jenkins Pipeline
6. Jenkins Shared Library

Overview

What is CI/CD?

- Continuous Integration (CI) - Automating code integration
- Continuous Deployment/Delivery (CD) - Automating software deployment
- Benefits: Faster releases, fewer bugs, higher efficiency



Why Use CI/CD Tools?

Advantages of CI/CD Tools

- Automate build, test, and deployment
- Improve collaboration and code quality
- Reduce human errors
- Faster time to market

Jenkins

Jenkins - The Open-Source Automation Server

- Highly customizable with plugins
- Supports distributed builds
- Works with any language or platform
- Requires manual setup and maintenance

GitLab CI/CD

GitLab CI/CD - Built-in CI/CD for GitLab Repos

- Easy integration with GitLab repositories
- YAML-based configuration (`**.gitlab-ci.yml**`)
- Supports auto-scaling runners
- Requires GitLab account

GitHub Actions

GitHub Actions - Native CI/CD for GitHub

- Deep integration with GitHub
- YAML-based workflows (`**.github/workflows/**`)
- Supports self-hosted and cloud runners
- Free for public repositories

Bitbucket Pipelines

Built-in CI/CD for Bitbucket repositories

- YAML-based configuration (`**bitbucket-pipelines.yml**`)
- Easy integration with Bitbucket
- Requires a Bitbucket account

CircleCI

CircleCI - Cloud & Self-hosted CI/CD

- Supports Docker, Kubernetes, and multi-cloud
- Fast caching and parallel execution
- Optimized for performance and scalability
- Free plan available with limitations

Comparison Table

Feature	Jenkins	GitLab CI/CD	GitHub Actions	Bitbucket Pipelines	CircleCI
Hosting	Self-hosted	Integrated	Integrated	Integrated	Cloud & Self-hosted
Configuration	UI + Script	YAML	YAML	YAML	YAML
Ease of Use	Medium	Easy	Easy	Easy	Easy
Best For	Large, complex projects	GitLab users	GitHub users	Bitbucket users	Performance-driven teams

Introduction to Jenkins

Introduction to Jenkins

- Jenkins is an open-source tool that automates tasks such as building, testing, and deploying software, aiming to accelerate the development and release process. It is one of the most popular CI/CD tools today.
- The first version of Jenkins was released in 2011.

Introduction to Jenkins

Key Features of Jenkins

- **Open-source**: Jenkins is an open-source tool with a strong community, providing a wide range of features and plugins.
- **Easy Integration**: Jenkins seamlessly integrates with various DevOps tools such as Git, Docker, Kubernetes, and testing frameworks like Selenium and JUnit.
- **Cross-platform Support**: Jenkins runs efficiently on multiple platforms, including Windows, macOS, Linux, and even within Docker containers.

Introduction to Jenkins

Benefits of Jenkins

- Automated and Seamless CI/CD: Jenkins helps build an automated and streamlined CI/CD pipeline, reducing development time, improving software quality, and enhancing deployment stability.
- Efficient Job and Pipeline Management: Jenkins allows easy creation of jobs or pipelines that trigger automatically upon new Git commits, enabling DevOps teams to manage complex CI/CD workflows effortlessly.

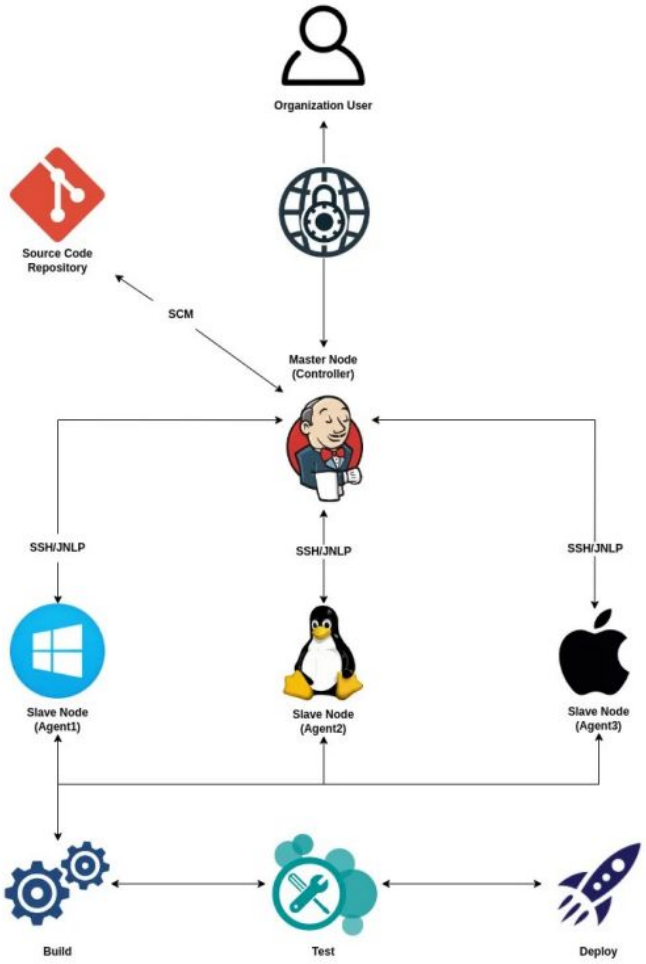
Jenkins Architecture

Jenkins Architecture

Jenkins is designed with a distributed architecture, consisting of a **Jenkins Master** that manages multiple **Jenkins Agents**, enabling workload distribution and parallel execution.

Benefits of the Distributed Architecture:

- Scalability: Allows Jenkins to handle large workloads by running jobs in parallel across multiple agents.
- Improved Performance: Reduces the load on the master, enhancing efficiency and speeding up CI/CD job execution.



Basic Jenkins Architecture

Jenkins Architecture

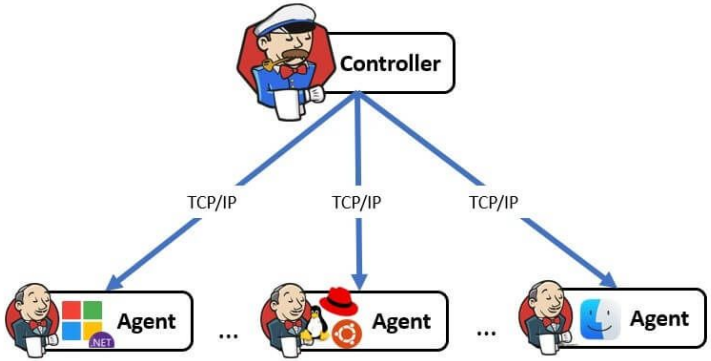
Main Components of Jenkins Architecture

- **Jenkins Master**: The central control unit of Jenkins, responsible for managing the entire system, including job coordination, agent management, and configuration storage.
- **Jenkins Agent (Node)**: Secondary machines that execute jobs assigned by the Jenkins Master.

Agents help distribute workloads and enable Jenkins to run CI/CD jobs in parallel across different environments.

- **Executor**: A component in Jenkins that represents an independent workflow capable of executing a job.

Each Node (including both the Master and Agents) has one or more Executors, allowing Jenkins to process multiple jobs in parallel.



Jenkins Architecture

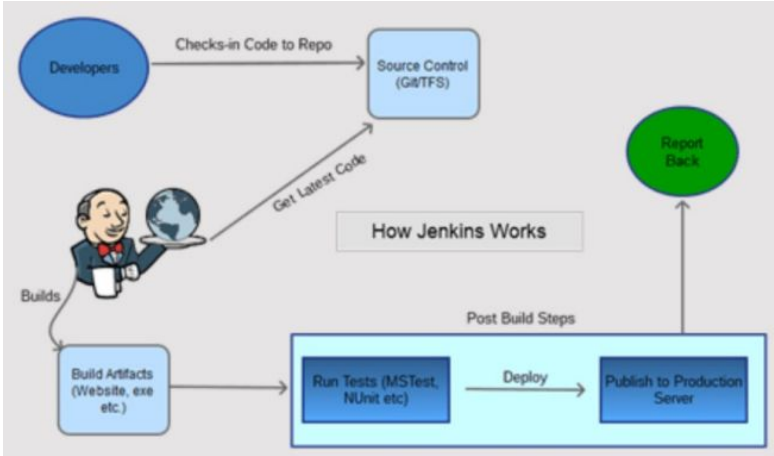
Basic Concepts in Jenkins

- **Job**: The main unit of work in Jenkins, which can be a Freestyle Project, Pipeline, or Multibranch Pipeline.
  - **Freestyle Project**: A simple job type that can execute basic build and test steps.
  - **Pipeline**: A sequence of tasks, usually defined in a Jenkinsfile, enabling more flexible and automated CI/CD workflows.
  - **Multibranch Pipeline**: A pipeline that automatically detects and creates jobs for different branches in a Git repository.
- **Node**: Any machine managed by the Jenkins Master, including both the Master itself and the Agents.
- **Plugin**: Extensions that allow Jenkins to integrate with other tools, add new features, and enhance performance.

Jenkins Architecture

How Jenkins Works

1. Triggering the Job: The Jenkins Master receives a request from the user or an automated trigger (e.g., a Git commit or a scheduled job).
    a. Webhook Notification: When a new commit is pushed to GitHub, GitHub sends a webhook notification to the Jenkins Master.
    b. Job Execution: Jenkins detects the change and automatically triggers the corresponding job or pipeline to build, test, and deploy the updated code.
2. Job Execution: The Master assigns the job to an Agent via an Executor, distributing the workload.
3. Result Collection: The Agent executes the job, and the results (logs, status, artifacts) are sent back to the Master for storage and reporting (email, Slack, ...).



Components of Jenkins

Components of Jenkins

Basic Configuration Components in Jenkins

1. **Plugins**: Extend Jenkins functionality and integrate with other tools like Git, Docker, Kubernetes, and testing frameworks.
2. **Security**: Manages user authentication, authorization, and access control to protect Jenkins from unauthorized access.
3. **Jenkins Agent**: Configures agents to distribute workloads and execute jobs on different machines.
4. **Notifications**: Sends job status updates via email, Slack, or other messaging services.
5. **Information Management**: Stores build logs, artifacts, environment variables, and credentials.
6. **Backup**: Regularly backs up Jenkins configurations, job data, and plugins to prevent data loss.

Jenkins Pipeline

Jenkins Pipeline

A Pipeline in Jenkins is a sequence of automated steps that handle the build, test, and deployment processes.

Benefits of Jenkins Pipeline

- Structured Workflow: Organizes and manages complex CI/CD processes in a clear and maintainable structure.
- Automation & Reliability: Reduces manual work, minimizes errors, and ensures consistent software quality.

Jenkins Pipeline

Main Components of a Pipeline

1. Agent: Specifies the environment or machine where the pipeline will run.
2. Stages: Define the different phases of the pipeline, such as Build, Test, Deploy.
3. Steps: Individual commands executed within each stage.

```
agent any  // Runs on any available agent
agent { label 'docker-node' }  // Runs on a specific agent
```

```
stages {
        stage('Build') {
                steps {
                        echo 'Building the project...'
                }
        }
        stage('Test') {
                steps {
                        echo 'Running tests...'
                }
        }
}
```

Jenkins Pipeline

Types of Pipelines in Jenkins

1. Declarative Pipeline:

- Uses a simplified, structured syntax.
- Easier to read and manage, making it ideal for simple pipelines.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing...'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying...'
            }
        }
    }
}
```

Jenkins Pipeline

Types of Pipelines in Jenkins

2. Scripted Pipeline:

- Written in **Groovy**, providing greater flexibility for complex workflows.
- Allows advanced scripting and dynamic logic.
- Defined inside a **node** block.

```groovy
node {
    // Define environment variables
    def branchName = 'main'

    stage('Checkout') {
        // Checkout code from a Git repository
        git branch: branchName, url: 'https://github.com/yo
    }

    stage('Build') {
        // Run build commands
        echo "Building the project..."
        sh 'make build'
    }

    stage('Test') {
        // Run test commands
        echo "Running tests..."
        sh 'make test'
    }

    stage('Deploy') {
        // Deployment step
        echo "Deploying application..."
        sh 'make deploy'
    }
}
```

Jenkins Pipeline

Jenkinsfile

- A **Jenkinsfile** is a script that defines a Jenkins Pipeline, usually placed in the root directory of a project so Jenkins can automatically detect it.
- It allows version control of the pipeline, making it easier to track and manage changes using Git.

Jenkins Pipeline

Pipeline Triggers in Jenkins

- Poll SCM
  - Jenkins periodically checks the source control system (e.g., Git) for updates.
  - Configured using a cron-like schedule (e.g., every 5 minutes). (H/5 * * * * // Polls every 5 minutes)
- GitHub Hook Trigger
  - GitHub webhook notifies Jenkins immediately when a new commit is pushed.
  - More efficient than Poll SCM because it triggers the pipeline instantly.

Jenkins Pipeline

Setting Up Notifications for a Jenkins Pipeline

- To notify your team about build results, you can configure Jenkins to send Slack messages, emails, or other notifications.

```
post {
    always {
        slackSend(channel: '#devops', message: "Build completed: ${currentBuild.currentResult}")
    }
}
```

- Monitoring and Debugging Jenkins Pipeline via Console Output

Jenkins Shared Library

Jenkins Shared Library

Jenkins Shared Library is a powerful feature that allows you to share code and build steps across multiple pipelines.

Why Use Shared Libraries?

- Avoid Code Duplication - No need to rewrite the same Jenkinsfile logic for every pipeline.
- Modular and Reusable - Store commonly used functions in one place.
- Easier Maintenance - Update once, apply to all pipelines using the library.

Jenkins Shared Library

Structure of a Jenkins Shared Library

A Shared Library is typically stored in a Git repository and follows this structure:

```
└── vars/
    ├── example.groovy   # Shared function
    ├── deploy.groovy    # Another shared function
└── resources/
    ├── config.yaml      # Optional config files
└── src/
    ├── org/company/     # Groovy classes
        ├── Utils.groovy
```

Jenkins Shared Library

Create a Groovy file in the vars/ directory.

vars/deploy.groovy

```
def call(String environment) {
     echo "Deploying to ${environment}"
     sh "deploy-script.sh ${environment}"
}
```

In your Jenkinsfile, load and call the function:

```
@Library('my-shared-library') _  // Load the
library

pipeline {
     agent any
     stages {
        stage('Deploy') {
           steps {
                deploy('staging')  // Call
the shared function
           }
        }
     }
}
```

Thank you