Jenkins

P by Praveen Kumar

Introduction to CI/CD

- Modern software requires fast, reliable, automated delivery
- CI/CD is a DevOps practice to automate build, test, and deployment
- Helps teams release features faster with fewer errors
- Ensures code quality and consistency across environments
- CI = Development automation

What is Software Delivery Lifecycle? (SDLC)

- SDLC is the process of developing and delivering software
- Key stages include:
  - Requirements
  - Design
  - Development
  - Build
  - Testing
  - Deployment
  - Maintenance
- Challenges: Manual steps, delays, human errors, inconsistent builds
- DevOps + CI/CD improves SDLC speed and accuracy

What is Continuous Integration (CI)?

- Practice of merging code frequently into a shared repository
- Every code change triggers an automated build and test
- Ensures code is always in a working state
- Early detection of bugs & integration issues
- Key goals:
  - Reduce merge conflicts
  - Improve code quality
  - Automate build and testing

Benefits of Continuous Integration

- Faster detection of issues
- Automated unit testing
- Consistent builds across all developers
- Less integration pain before releases
- Improves developer productivity
- Releases become predictable and safe

What is Continuous Delivery (CD)?

- Automatic packaging and preparation of software for deployment
- Every successful build is always ready for release
- Deployment is automated up to staging
- Human approval often used before production deployment
- Ensures “deploy anytime” capability

Continuous Delivery vs Continuous Deployment

Continuous Delivery	Continuous Deployment
Code is always release-ready	Every successful build automatically goes to production
Manual approval required	No manual approval
Focuses on automation till staging	Fully automated to production

Real-Time Problems Without CI/CD

- Manual builds lead to inconsistent results
- Delayed testing → bugs found late
- Frequent merge conflicts
- Releases take days or weeks
- Hard to track which version is running
- Human errors during deployment
- No feedback loop for developers
- Production breakdowns during releases

How Jenkins Solves These Problems

- Automates build, test, and deployment
- Tracks every code change
- Integrates with Git, Maven, Docker, Kubernetes
- Reduces manual effort
- Ensures consistent builds
- Provides dashboards for build status
- Easy integration with 2000+ plugins
- Enables end-to-end CI/CD pipelines
- Fast feedback for developers

DevOps + Jenkins Integration

- DevOps focuses on automation, collaboration, and fast delivery
- Jenkins acts as the automation engine for DevOps
- Integrates across the DevOps pipeline:
  - Source Code → Build → Test → Release → Deploy → Monitor
- Automates CI/CD
- Works with all DevOps tools: GitHub, Docker, Ansible, Terraform, AWS, Kubernetes
- Helps teams move from traditional SDLC to DevOps culture

10: Where Jenkins Fits in DevOps

- **Plan** → Using Agile tools (Jira, Azure Boards)
- **Code** → Using GitHub/GitLab
- **Build** → Jenkins automates builds
- **Test** → Jenkins triggers test automation
- **Release** → Jenkins packages artifacts
- **Deploy** → Jenkins deploys using SSH, Docker, Kubernetes
- **Monitor** → Integrates with Prometheus, Grafana

Introduction to Jenkins

- Jenkins is the world’s most widely used **open-source automation server**
- Helps automate **build, test, and deployment** phases
- Core of DevOps and CI/CD workflows
- Highly extensible through plugins
- Works on Windows, Linux, macOS, Kubernetes, and Cloud

What is Jenkins?

- Jenkins is an automation tool for **Continuous Integration and Continuous Delivery (CI/CD)**
- Allows developers to integrate code frequently
- Automates repetitive tasks like:
  - Code compilation
  - Unit testing
  - Packaging
  - Deployment
- Can integrate with nearly every DevOps tool
- Built in Java, supports distributed builds

History of Jenkins (Hudson → Jenkins)

- Originally developed as **Hudson** at Sun Microsystems (2004–2010)
- Conflict occurred between Oracle (owner) and open-source community
- Community forked Hudson into **Jenkins** in 2011
- Jenkins became the primary open-source CI tool maintained by the community
- Oracle continued Hudson, but it eventually died
- Jenkins is now managed by Jenkins Community + Linux Foundation

Why Jenkins Became Popular

- Fully **open-source** and **free**
- Simple to install and configure
- Works on almost all platforms
- Large library of **2000+ plugins**
- Supports automation of any technology: Java, Python, Node, Docker, Kubernetes, etc.
- Strong community support
- Easy integration with GitHub, GitLab, Bitbucket, Docker, AWS, Azure, GCP
- Allows **pipeline-as-code** with Jenkinsfile

Key Features of Jenkins

- **Easy Installation**: Single WAR file, runs on Java
- **Plugin Ecosystem**: Extend functionality as needed
- **Distributed Builds**: Master-agent architecture
- **Easy Integration**: Compatible with all DevOps tools
- **Pipeline as Code** (Declarative & Scripted pipelines)
- **Scalability**: Add multiple agents for load distribution
- **Monitoring & Reporting** tools built-in
- **Security features**: Role-based access, credential management
- **Extensible via REST API and Groovy scripting**

Jenkins Architecture – Overview

- Jenkins follows a **Master–Agent (Controller–Worker)** architecture
- Master (Controller):
  - Orchestrates the entire build pipeline
  - Stores configurations, plugins, jobs
- Agents (Workers):
  - Execute build/test workloads
  - Can be physical machines, VMs, containers, or cloud instances
- Communication via:
  - SSH
  - JNLP
  - WebSocket
- Supports distributed CI/CD pipelines

Jenkins Master (Controller)

- Main Jenkins server responsible for:
  - Managing jobs and pipelines
  - Scheduling builds
  - Coordinating agents
  - Monitoring build progress
  - Storing build history, logs, and workspace metadata
  - Managing plugins, credentials, users
- Ideally **not used for building** heavy applications
- Should focus only on **orchestration**, not workload execution

Jenkins Agent (Slave / Worker Node)

- Executes the actual build tasks
- Can run on separate machines for load balancing
- Types of agents:
  - SSH agents
  - JNLP agents
  - Docker agents
  - Kubernetes ephemeral agents
- Benefits:
  - Parallel builds
  - Platform-specific builds (Windows/Linux)
  - Reduced load on master
- Agents connect securely with the controller

Controller vs Worker Nodes

Controller (Master)	Worker Node (Agent)
Schedules jobs	Executes jobs
Manages UI, plugins, credentials	Performs build, test, deploy tasks
Stores job configuration	Has required tools (JDK, Maven, Docker, Node, etc.)
Should NOT run heavy builds	Can handle heavy workloads
Central orchestration	Distributed execution

Plugin Ecosystem

- Jenkins supports **2000+ plugins**
- Plugins add support for:
  - Source control: Git, GitHub, GitLab, Bitbucket
  - Build tools: Maven, Gradle, Ant
  - Piperine tools: Blue Ocean, Pipeline utility
  - Containers: Docker, Kubernetes
  - Notifications: Slack, Email, Teams
  - Test reporting: JUnit, Selenium, Allure
  - Cloud integrations: AWS, Azure, GCP
- Plugins help Jenkins integrate with **any** DevOps technology
- Plugin Manager allows easy install/update