Continuous Integration using Jenkins

By,

Harshil Kanakia

Outline

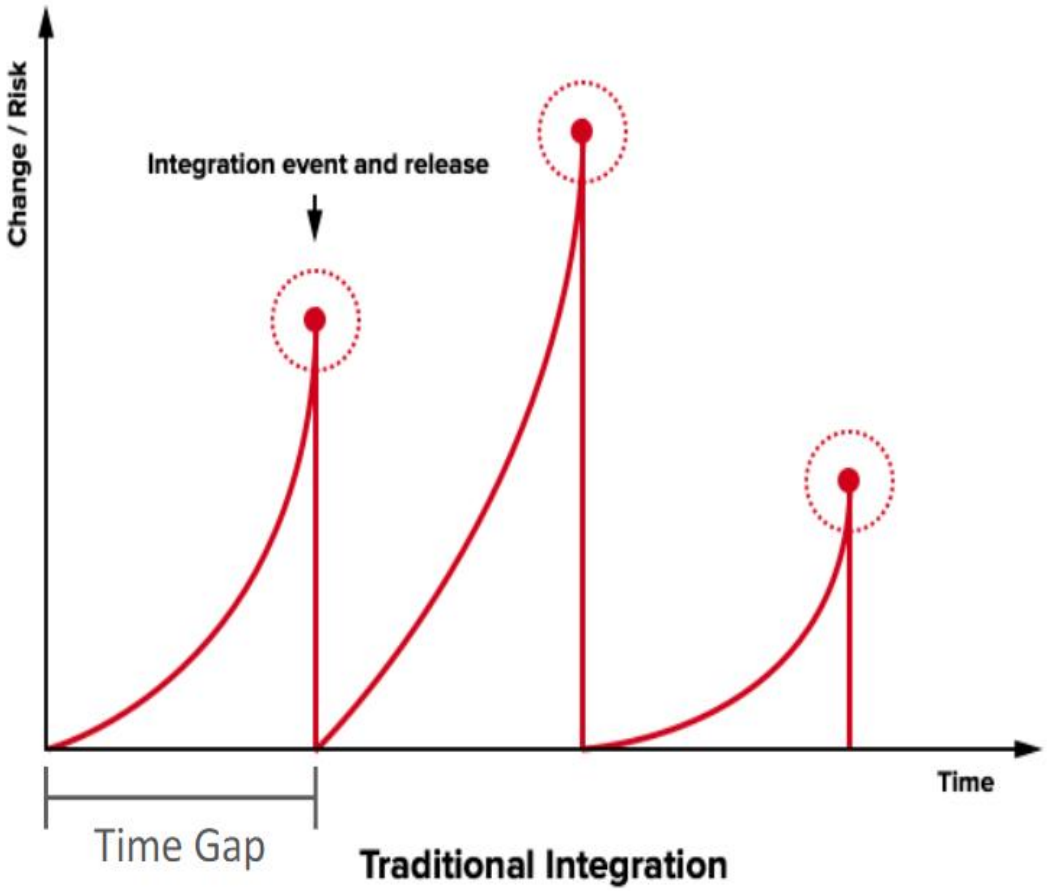
- Introduction to Continuous Integration
- Introduction to Maven and its architecture
- Introduction to Jenkins and its architecture
- Jenkins workflow
- Jenkins Pipeline
- Jenkins Master Slave Architecture

Traditional Integration

This approach requires custom coding, manual configuration, and dedicated infrastructure to enable communication between systems.

Traditional Integration

- Greater time gap in case of Traditional Integration
- Relatively greater risk or change in conflicts

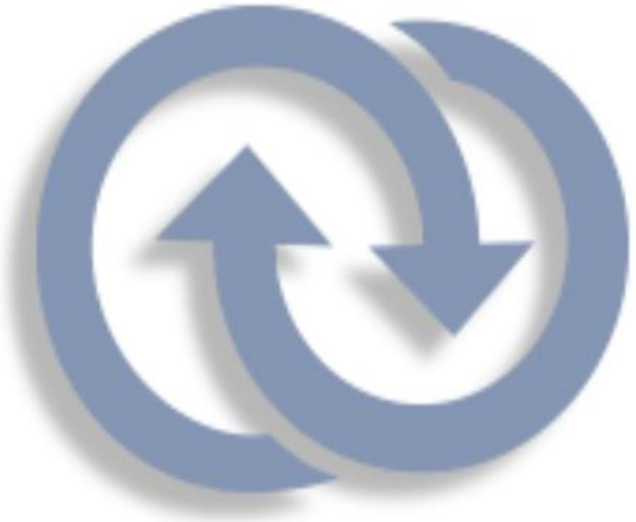


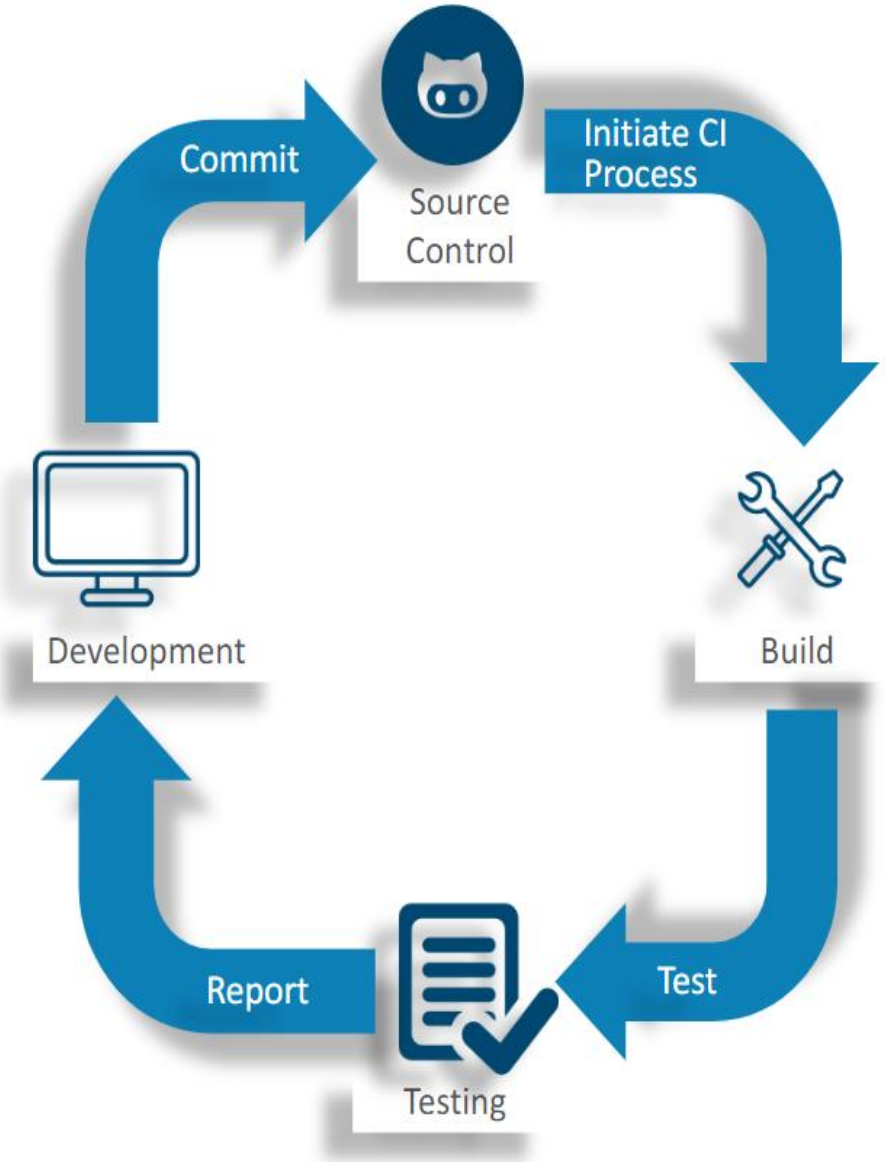
Problems With Traditional Integration

- Difficult to Implement all the details that have changed
- Difficult to get the latest version of the code early
- Difficult to manage all the changes with the changing API versions.
- Difficult to manage the entire subsystem of the application that behave differently
- Difficult to rewrite the whole code

What Is Continuous Integration?

"It is the process of automating the building and testing of code, each time one of the team member commits changes to version control."





Importance Of Continuous Integration

01

Improves Quality

Improves the quality by running multiple unit tests and analysing various static code

02

Increases Productivity

Automating build of code saves a lot of time, thereby increasing productivity

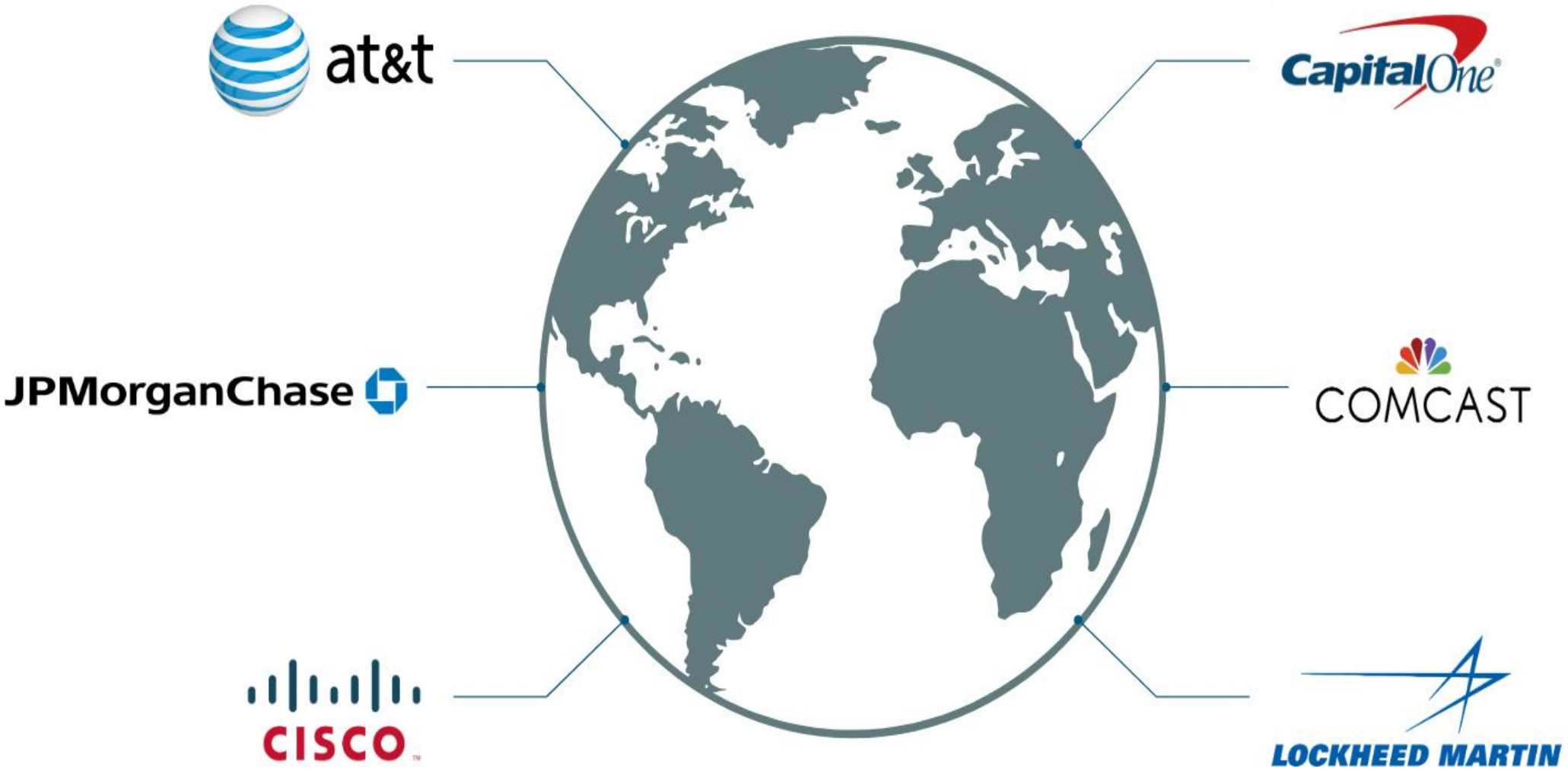
03

Reduces Risk

Eliminate the risk of potential human errors by automating test

Popular Continuous Integration Tools

Companies Using Jenkins



Maven

- Maven is like a smart assistant for Java projects.

It helps you:

1. Manage Dependencies – If your project needs external libraries (e.g., database drivers, logging tools), Maven automatically downloads and manages them.

2. Automate Builds – Instead of manually compiling code, running tests, and packaging files, Maven does it for you in a standard way.

3. Standardize Project Structure – Every Maven project follows a common structure, making it easier to understand and collaborate.

Core Components of Maven

- POM.xml (Project Object Model) – The configuration file defining dependencies, plugins, and build settings.
- Lifecycle Phases – Standard stages like clean, compile, test, package, install, and deploy.
- Repositories –

1. Local Repository (stored on the developer's machine)
2. Central Repository (default online source for dependencies)
3. Remote Repository (custom repositories for organization-specific dependencies)

Basic Maven Commands

- mvn clean # Removes target directory (clean build)
- mvn compile # Compiles the source code
- mvn test # Runs unit tests
- mvn package # Packages the project into a JAR/WAR
- mvn install # Installs the package into the local repository
- mvn deploy # Deploys the package to a remote repository

How Maven Works?

- You have a special file called pom.xml where you define:

1. Project details
2. Required dependencies (libraries)
3. Build instructions

- When you run Maven commands, it reads pom.xml and takes care of everything—compiling, testing, and packaging your project.

Example

1. Developer runs a Maven command like mvn package.
2. Maven reads pom.xml to find project settings and dependencies.
3. If required dependencies are missing, Maven downloads them from the Central Repository and stores them in the Local Repository.
4. Maven executes the Build Lifecycle step-by-step.
5. The final output (e.g., a .jar or .war file) is created and can be deployed.

Analogy

Imagine you are making a pizza:

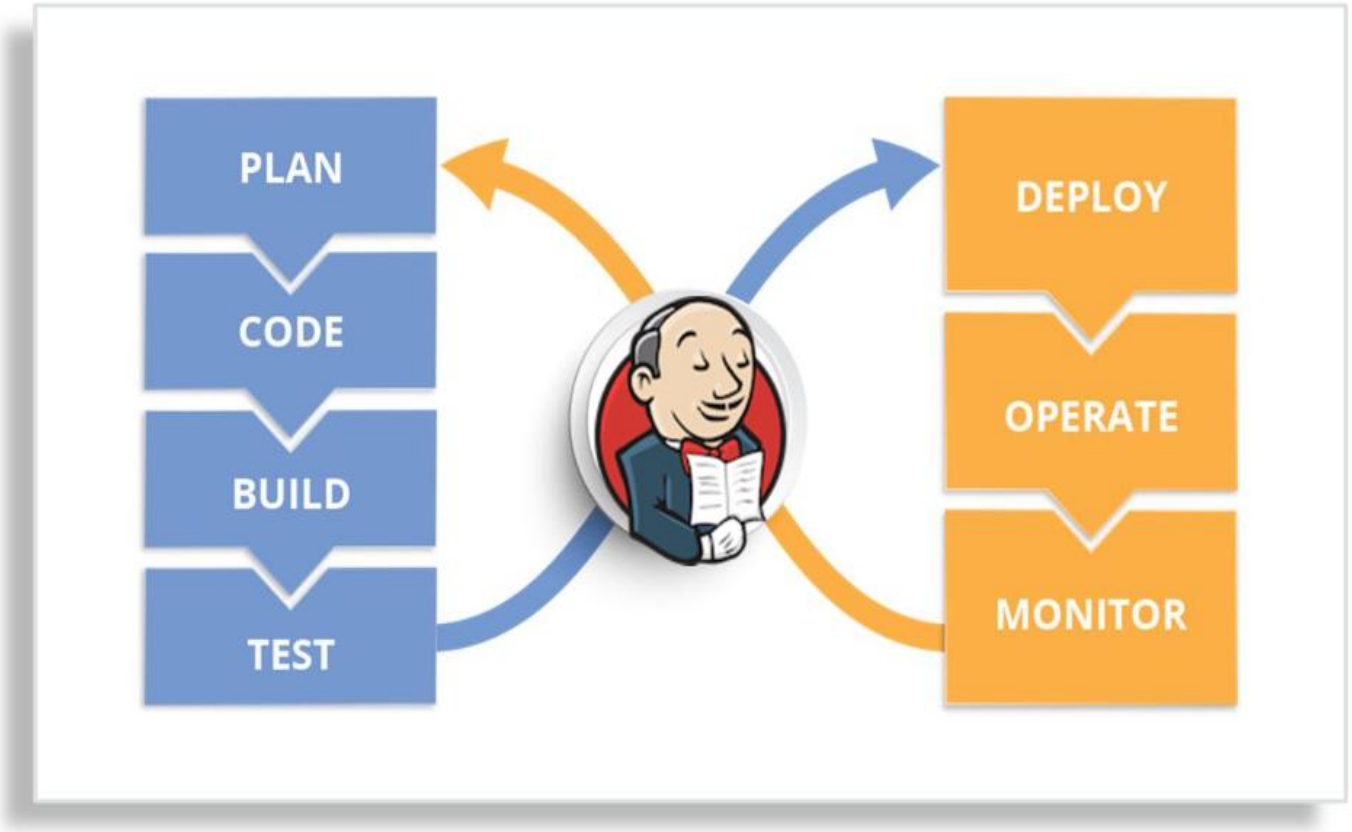
- POM file = Recipe with ingredients and instructions.
- Repositories = Grocery stores where you get ingredients.
- Build Lifecycle = Steps like preparing dough, adding toppings, baking.
- Plugins = Extra tools like a pizza cutter or delivery service.

Maven ensures everything runs smoothly so you always get a perfect pizza (or Java project)!

Introduction To Jenkins

What Is Jenkins?

"A Continuous Integration server which manages and control processes such as plan, code, build, test, deploy, operate and monitor in DevOps Environment."



Jenkins in Simple Terms

Jenkins is like a robotic assistant for developers that helps automate the process of building, testing, and deploying software. Instead of doing everything manually, Jenkins does it for you, making software development faster, easier, and error-free.

Why use Jenkins?

- Imagine a team of developers working on a project. Every time they make changes, they need to:

✓ Compile the code

✓ Run tests to check for errors

✓ Package the software

✓ Deploy it to a server

Doing this manually for every update is time-consuming and prone to mistakes. Jenkins automates this process!

Components of Jenkins

- Jenkins Master (Control Room)
- Jenkins Agent (Workers)
- Job / Build (Production Line)
- Repositories (Code Storage)
- Plugins (Extra Tools)
- Build Artifacts (Final Product)
- Notification (Alerts and Reports)

Jenkins Master (Control Room)

- The brain of Jenkins.
- It manages jobs, schedules builds, and assigns tasks to agents.
- Provides a web dashboard to control everything.

Jenkins Agent (Workers)

- These are workers that actually build, test, and deploy the software.
- The Master assigns tasks to Agents to distribute the workload.
- Can run on different machines (Windows, Linux, etc.) for better performance.

Job / Build (Production Line)

- A job is a set of instructions Jenkins follows (e.g., fetch code, compile, test, deploy).
- Jobs are defined in Pipelines (step-by-step automation).

Repositories (Code Storage)

- Jenkins pulls the latest code from GitHub, GitLab, or Bitbucket.
- It checks for changes automatically (Continuous Integration).

Plugins (Extra Tools)

- Jenkins has thousands of plugins to integrate with tools like Docker, Kubernetes, AWS, Slack, SonarQube.
- Plugins extend Jenkins' functionality.

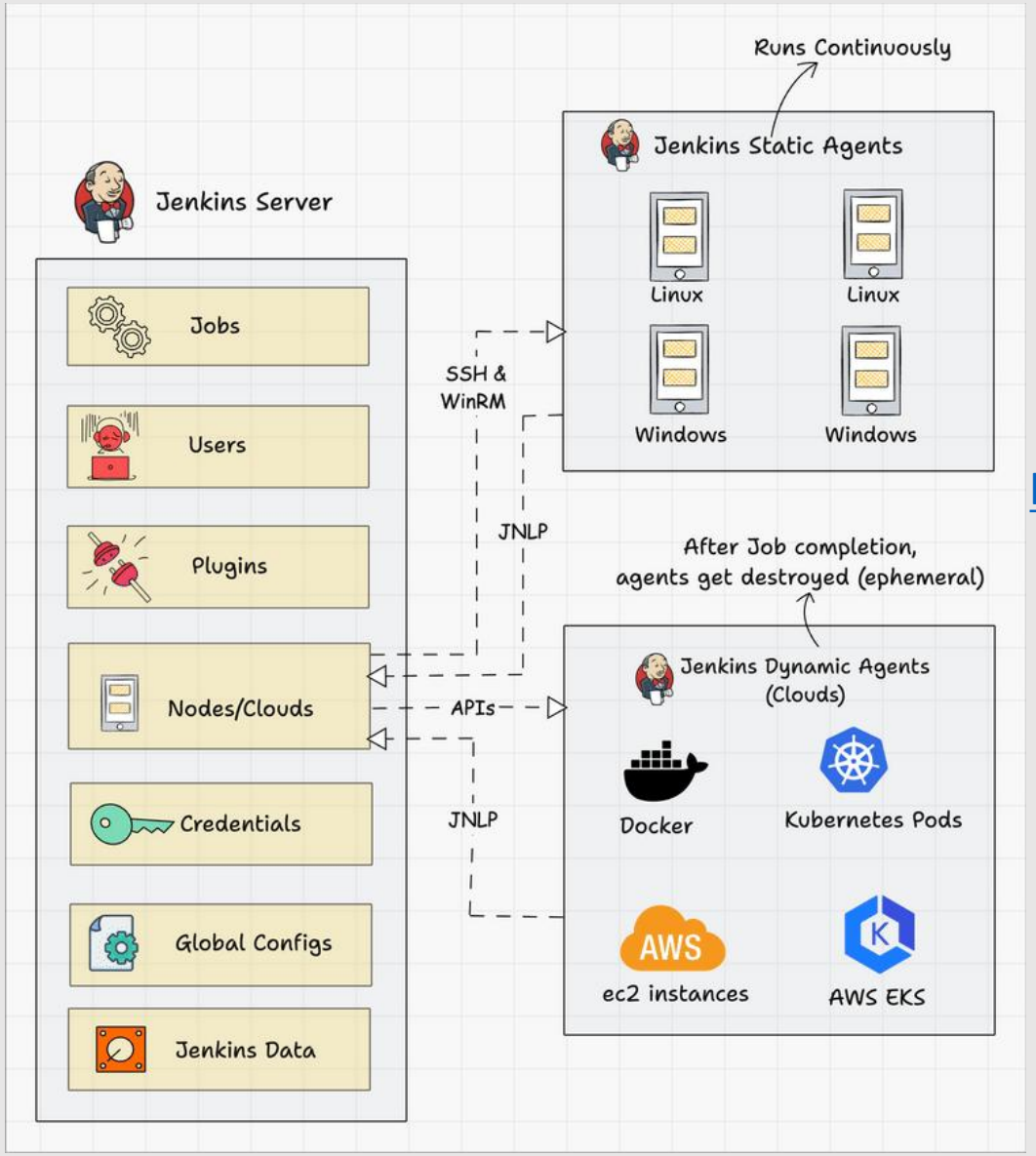
Build Artifacts (Final Product)

- After a successful build, Jenkins packages the software (e.g., .jar, .war, .zip).
- Stores it or deploys it to a server/cloud.

Notification (Alerts and Reports)

- Jenkins sends emails, Slack messages, or alerts to inform developers about build status (Success/Failure).

Jenkins Architecture



https://devopscube.com/jenkins-architecture-explained/

How Jenkins Works (Step-by-Step Example)

Imagine a pizza-making factory where Jenkins is the manager:

1 Order Received (Code Commit) – Developer pushes code to GitHub.

2 Factory Starts Work (Jenkins Master Detects Change) – Triggers the build automatically.

3 Assigns Work (Master to Agents) – Agents compile the code and run tests.

4 Quality Check (Testing Stage) – Agents check if the code is working correctly.

5 Packaging (Build Artifacts) – Creates a .jar or .war file.

6 Delivery (Deployment) – Sends the software to a server or cloud.

7 Sends Notification – Jenkins emails the team with the results.

Diagram Representation

[Developer] → Pushes Code → [GitHub]

↓

[Jenkins Master] → Schedules Job → [Jenkins Agent]

↓

[Compile] → [Test] → [Package] → [Deploy]

↓

[Send Notification] (Success or Failure)

Jenkins Handson

- **Jenkins Installation Link:**

https://www.jenkins.io/doc/book/installing/linux/#debianubuntu

Problem Statement

Problem Statement

You are working as a DevOps Engineer in a company named Sanders &Fresco Pvt ltd. You have been asked by your manager to create a Maven Project using Jenkins and build a war file of that project. As a proof of concept, you have been given a webapplication to build.

Steps to solve:

- Open Jenkins and create a Maven project using it.
- You will have to create thef ollowing jobs,which are as follow:

    1 Compile

    2 Code Review

    3 Unit test

    4 Package

    5 Deploy