**Jenkins**

Jenkins is an open-source automation server used for **Continuous Integration (CI) and Continuous Deployment/Delivery (CD)**. It helps automate the software development process by enabling developers to build, test, and deploy their applications efficiently

**Introduction to CI & CD**

Continuous Integration (CI) and Continuous Delivery/Deployment (CD) are DevOps practices aimed at automating the software development lifecycle.

* **CI**: Automates code integration from multiple developers into a shared repository frequently, ensuring code correctness and reducing integration issues.
* **CD**: Ensures that code changes are automatically prepared for production release. Continuous Deployment goes a step further by automatically deploying changes without manual intervention.

**Continuous Delivery vs. Continuous Deployment**

* **Continuous Delivery**: Code is automatically tested and ready for deployment but requires manual approval to push to production.
* **Continuous Deployment**: Every change that passes the automated tests is deployed to production without manual intervention.

**Introduction to Jenkins**

Jenkins is an open-source automation server that enables CI/CD by automating the build, test, and deployment processes.

**Advantages of CI/CD**

* Faster development and release cycles
* Reduced integration and deployment issues
* Higher code quality with automated testing
* Improved collaboration and efficiency

**Use Cases of CI/CD**

* Automated testing and deployment
* Code quality assurance
* Rapid feedback on code changes
* Infrastructure as code (IaC) deployment.

**Jenkins Architecture**

* **Master-Slave Architecture**: Master handles jobs, and slaves execute them.
* **Pipeline-based Architecture**: Uses Jenkins pipelines for complex workflows.

**Jenkins UI Description**

* Dashboard: Shows all jobs and builds
* Build history, console output, and job details
* Plugin management and system configuration

Jenkins follows a **Master-Agent Architecture**, which helps it handle multiple tasks at the same time. Let's break it down:

**1. Master Node (The Manager)**

* The **Master** is like the **brain of Jenkins**.
* It controls everything: scheduling jobs, distributing tasks, and displaying results.
* It has the **Jenkins Dashboard (UI)** where users can configure jobs.
* It does **lightweight tasks** like checking code changes and sending notifications.

💡 **Think of it as a project manager who assigns work to employees (agents).**

**2. Agent Nodes (The Workers)**

* Agents (also called nodes) are the **workers** that actually perform the tasks.
* They take the instructions from the **Master** and execute builds and tests.
* Multiple agents can work in parallel, making Jenkins fast and efficient.

💡 **Think of agents as employees in a company who do the actual work while the manager (Master) assigns tasks.**

**3. Build Executors (The Work Desks)**

* Each Agent has **Build Executors**, which are like individual **workstations** where jobs run.
* If an agent has **two executors**, it can run two jobs at the same time.
* More executors = More jobs running simultaneously.

💡 **Think of an executor as a worker's desk. If a worker has multiple desks, they can handle more tasks at once.**

**4. Job Execution Flow (How It Works)**

1. A developer pushes code to GitHub/GitLab.
2. Jenkins Master detects the change and assigns a job.
3. The job is sent to an **available Agent**.
4. The Agent runs the build and tests on its **Build Executors**.
5. The result (Success/Failure) is sent back to the Master.
6. If the build fails, Jenkins can notify the team via email or Slack.

**Why Use This Architecture?**

✅ **Scalability** – You can add more agents to handle more builds.  
✅ **Efficiency** – Work is distributed, avoiding slowdowns.  
✅ **Flexibility** – Agents can run on different OS (Linux, Windows, macOS).  
✅ **Faster Builds** – Multiple builds can run in parallel.

**Configuring Jenkins Jobs**

* Job types: Freestyle, Pipeline, Multi-branch, etc.
* Defining build steps and triggers

**Plugins in Jenkins**

* **Introduction to Plugins**: Extend Jenkins functionality
* **Installing Plugins**: Manage via Jenkins Plugin Manager
* **Common Plugins**: Git, Maven, Pipeline, Docker, etc.

**Features and Functionality**

* Scheduling and executing jobs
* Integrating with SCM tools like Git
* Build pipelines and workflows

**Configuring Jenkins with Java, Git, and Maven**

* Installing JDK and setting JAVA\_HOME
* Configuring Git credentials
* Setting up Maven and its repositories

**Jenkins Build and Workspace**

* Workspace: Directory where builds execute
* Build directories and logs

**Jenkins Directory Structure**

* Home directory, plugins, builds, and job configurations

**Webhooks and Auto-detecting Code Changes**

* Using GitHub webhooks to trigger builds
* Poll SCM and push triggers

**Maven Build Jobs with Options**

* Running Maven builds with different goals (clean, package, install, deploy)

**Build Jobs**

* **Creating Freestyle Build Jobs**: Configuring source code management, build triggers, and post-build actions
* **Build Triggers**: SCM polling, webhook triggers
* **Build Steps**: Executing scripts, compiling code
* **Pre and Post Build Actions**: Running tests, publishing reports

**Properties and Properties File Description**

* Using properties for dynamic build configurations

**Executing Build Jobs and Identifying Success/Failures**

* Console logs and build statuses (Success, Failure, Unstable)

**Parameterized Builds**

* Passing parameters to Jenkins jobs for flexible execution

**Distributed Builds**

* Running builds on multiple nodes to improve performance

**Failure and Success Email Notifications**

* Configuring email notifications using Jenkins Email Extension plugin

**Security in Jenkins**

* **Authentication and User Levels**: Admin, Developer, Read-only roles
* **Access Types and Administration**: Role-based access control (RBAC)

**Jenkins Maintenance**

* **Maintenance and Troubleshooting**: Monitoring logs, debugging failed builds
* **Filesystem Management**: Managing Jenkins disk usage
* **Troubleshooting**: Debugging failed jobs and resolving dependency issues

**Jenkins Backup and Recovery**

* Backing up Jenkins configurations and jobs
* Restoring Jenkins from backups
* **Jobs Backup and Recovery**: Using plugins for job backup

**Migrating Jenkins from One Server to Another**

* Exporting configurations and importing on a new server
* Handling plugin and environment dependencies

These topics cover the fundamentals and advanced concepts of Jenkins for CI/CD automation.

**1. What is a Freestyle Project?**

A **Freestyle Project** is a basic, GUI-based Jenkins job. It provides a simple way to configure builds using dropdowns and checkboxes.

✅ **Best for**: Small, straightforward tasks like compiling code, running tests, or deploying applications.  
✅ **No coding required** – everything is configured through the Jenkins UI.  
✅ **Limited flexibility** – Complex workflows are harder to manage

**2. What is a Pipeline?**

A **Pipeline** is an advanced, script-based Jenkins job. It allows defining the entire CI/CD workflow using **Jenkinsfile** (written in Groovy).

✅ **Best for**: Complex automation workflows, multi-step builds, and deployments.  
✅ **More flexible** – Supports parallel execution, conditional logic, and external integrations.  
✅ **Code as Configuration** – The pipeline script is stored in **Jenkinsfile**, making it easy to version control.

**1️⃣ What is an Upstream Project?**

An **upstream project** is a job that **triggers another job**.

💡 **Example:**

* **Job A (Upstream)** → Triggers **Job B (Downstream)** when it completes.
* If **Job A fails**, **Job B will not run** (unless configured otherwise

**What is a Downstream Project?**

A **downstream project** is a job that **gets triggered by another job (upstream job)**.

💡 **Example:**

* **Job B (Downstream)** starts **after Job A (Upstream) completes**.
* This is useful when you have multiple **dependent tasks** (e.g., build → test → deploy).

**Plugins** – Extensions that add new features or integrate Jenkins with external applications.

**Tools**- Software or utilities required for building, testing, and deploying applications.

** Declarative Pipeline –** Easier to read and write, structured with predefined syntax.

** Scripted Pipeline –** More flexible, uses Groovy scripting, allows complex workflows

|  |
| --- |
|  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |