**CICD in Practice with Jenkins**

**1. Jenkins Fundamentals**

**What is Jenkins?**

* **Jenkins** is the most widely used **open-source automation server** for building, testing, and deploying applications.
* It provides **plugins** to integrate with almost any DevOps tool (Git, Docker, Kubernetes, Terraform, etc.).

**Key Features**

* Supports **Continuous Integration (CI)** and **Continuous Delivery (CD)**.
* Extensible via **1,800+ plugins**.
* Provides both **UI-based jobs** and **Pipeline-as-Code**.
* Scales via master–agent architecture.

**Common Jenkins Terminology**

* **Job/Project** → Unit of work (build/test/deploy).
* **Node/Agent** → Machine where Jenkins executes jobs.
* **Executor** → Thread that runs a job on a node.
* **Plugin** → Adds new functionality (Git, Docker, etc.).
* **Pipeline** → Workflow definition for CI/CD.

**2. Jenkins Pipeline-as-Code**

Jenkins supports **declarative** and **scripted** pipelines, defined in a Jenkinsfile stored in the project repository.

**Sample Jenkinsfile (Declarative Pipeline)**

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git branch: 'main', url: 'https://github.com/example/myapp.git'

}

}

stage('Build') {

steps {

sh 'mvn clean package'

}

}

stage('Test') {

steps {

sh 'mvn test'

}

}

stage('Docker Build & Push') {

steps {

sh 'docker build -t myrepo/myapp:latest .'

sh 'docker push myrepo/myapp:latest'

}

}

stage('Deploy') {

steps {

sh 'docker run -d -p 8080:8080 myrepo/myapp:latest'

}

}

}

}

**Key Benefits of Pipeline-as-Code**

* Version-controlled workflows.
* Repeatable and auditable.
* Easier collaboration between Dev and Ops.

**3. Git + Jenkins + Docker Integration Pattern**

This integration forms a **modern CI/CD pipeline**.

**Workflow**

1. **Code Commit (GitHub/GitLab/Bitbucket)**
   * Developer pushes code to a Git repository.
   * Webhook triggers Jenkins job.
2. **Jenkins Build**
   * Jenkins checks out the code.
   * Builds the application (e.g., Maven/Gradle for Java, npm for Node.js).
3. **Testing**
   * Runs automated unit and integration tests.
   * Uses frameworks like JUnit, Selenium, PyTest.
4. **Docker Build & Push**
   * Jenkins builds a Docker image with application code.
   * Pushes the image to Docker Hub or private registry (ECR, ACR, Harbor).
5. **Deployment**
   * Jenkins deploys container to test/prod environments (Docker Compose, Kubernetes, or cloud).

**Diagram (textual)**:

Git Commit → Jenkins Build → Test → Docker Build → Push Image → Deploy

**4. Build and Deployment Automation**

Automation ensures **speed, consistency, and reliability**.

**a) Build Automation**

* Compile source code.
* Run static analysis (SonarQube, Checkstyle).
* Package application artifacts (JAR, WAR, Docker image).
* Store in artifact repository (Nexus, Artifactory).

**Example: Java Build in Jenkins**

stage('Build') {

steps {

sh 'mvn clean package'

}

}

**b) Deployment Automation**

* Deploy artifacts/containers automatically to target environments.
* Use **IaC (Terraform, Ansible)** for infra setup.
* Integrate with Kubernetes for container orchestration.

**Example: Deploy with Docker**

stage('Deploy') {

steps {

sh 'docker run -d -p 8080:8080 myrepo/myapp:latest'

}

}

**Example: Deploy with Kubernetes (kubectl)**

stage('Deploy to K8s') {

steps {

sh 'kubectl apply -f k8s/deployment.yaml'

}

}

**5. Example Use Case: Online Retail Application**

* **Scenario:** A retail company wants automated deployments for its Spring Boot app.
* **Setup:**
  + GitHub repo with application + Jenkinsfile.
  + Jenkins pipeline builds JAR, runs unit tests.
  + Docker image built and pushed to AWS ECR.
  + Kubernetes deployment updates pods using rolling updates.
* **Outcome:** Weekly releases turned into **daily automated deployments**, reducing downtime and integration issues.

**6. Summary**

* Jenkins is a **flexible automation server** for CI/CD pipelines.
* **Pipeline-as-Code** (Jenkinsfile) enables version-controlled, repeatable workflows.
* Integration of **Git + Jenkins + Docker** ensures seamless code-to-deployment pipelines.
* **Automation** in build and deployment ensures faster, reliable, and scalable software delivery.