**Jenkins CI/CD: Practical Implementation**

**Table of Contents**

**1. Introduction**

* Overview of Jenkins & Course Agenda
* Why Use Docker as Jenkins Agent?
* GitHub Repository Reference

**2. Setting Up Jenkins on AWS EC2**

* Launching an EC2 Instance (Ubuntu)
* Installing Java (Prerequisite)
* Installing Jenkins
* Opening Jenkins Port (8080) in AWS Security Group
* Accessing Jenkins UI & Initial Setup

**3. Jenkins Architecture & Docker Agents**

* Traditional Jenkins Master-Worker Model (Challenges)
* Modern Approach: Docker as Dynamic Agents
* Benefits of Docker Agents (Cost, Efficiency, Isolation)

**4. Configuring Docker for Jenkins**

* Installing Docker on EC2
* Granting Jenkins User Docker Permissions
* Installing Docker Pipeline Plugin in Jenkins
* Restarting Jenkins for Changes

**5. Creating Jenkins Pipelines**

* Freestyle vs. Pipeline Projects (Declarative vs. Scripted)
* Writing Your First Pipeline (Docker Agent Example)
* Multi-Stage & Multi-Agent Pipelines (Frontend, Backend, DB)
* Using Jenkins Pipeline Syntax Generator

**6. CI/CD with Kubernetes & Argo CD**

* Deploying a Python Todo App via Jenkins
* Updating Kubernetes Manifests in Git
* Argo CD for GitOps (Auto-Deployment to Kubernetes)

**7. Jenkins Interview Questions**

* Explaining CI/CD Workflow in Your Org
* Handling Worker Node Failures
* Jenkins Administration (Installation, Security, Plugins)

**Detailed Notes**

**1. Introduction**

* **Goal**: Learn Jenkins from scratch (installation, Docker agents, Kubernetes deployment).
* **Why Docker Agents?**
  + Lightweight compared to VMs.
  + No dependency conflicts (each job runs in an isolated container).
  + Cost-effective (containers spin up/down dynamically).
* **GitHub Repo**: Contains all commands, Jenkinsfiles, and documentation.

**2. Setting Up Jenkins on AWS EC2**

**Steps:**

1. **Launch an EC2 Instance (Ubuntu)**
   * No special requirements (t2.micro is sufficient).
   * Ensure SSH access is configured.
2. **Install Java (Jenkins Dependency)**

bash

sudo apt update

sudo apt install openjdk-11-jdk -y

java -version *# Verify installation*

1. **Install Jenkins**

bash

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt update

sudo apt install jenkins -y

sudo systemctl start jenkins

1. **Open Port 8080 in AWS Security Group**
   * Edit inbound rules → Allow TCP 8080 from 0.0.0.0/0 (or your IP).
2. **Access Jenkins UI**
   * Open http://<EC2\_Public\_IP>:8080.
   * Get initial admin password:

bash

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

* + Install suggested plugins & create admin user.

**3. Jenkins Architecture & Docker Agents**

**Traditional Approach (Static Worker Nodes)**

* **Problem**:
  + Wasted resources (idle VMs).
  + Dependency conflicts (different teams need different tools).
  + Maintenance overhead (upgrading VMs, packages).

**Modern Approach (Docker Agents)**

* **How It Works**:
  + Jenkins Master schedules jobs.
  + Docker dynamically creates containers per job (e.g., node:16 for Node.js jobs).
  + Containers terminate after job completion (no idle resources).
* **Benefits**:
  + No VM maintenance.
  + Easy to switch versions (just change Docker image in Jenkinsfile).

**4. Configuring Docker for Jenkins**

**Steps:**

1. **Install Docker on EC2**

bash

sudo apt install docker.io -y

sudo systemctl start docker

1. **Grant Jenkins User Docker Access**

bash

Copy

Download

sudo usermod -aG docker jenkins

sudo usermod -aG docker ubuntu *# Optional*

sudo systemctl restart docker

1. **Install Docker Pipeline Plugin**
   * Go to **Manage Jenkins → Manage Plugins → Available Plugins**.
   * Search for **"Docker Pipeline"** and install.
   * Restart Jenkins (http://<IP>:8080/restart).

**5. Creating Jenkins Pipelines**

**Pipeline vs. Freestyle Projects**

* **Freestyle**: GUI-based (limited scalability, hard to version control).
* **Pipeline**: Code-based (Jenkinsfile stored in Git, supports CI/CD workflows).

**Example 1: Simple Docker Agent Pipeline**

groovy

pipeline {

agent {

docker { image 'node:16-alpine' }

}

stages {

stage('Test') {

steps {

sh 'node --version'

}

}

}

}

* **What Happens**:
  + Jenkins pulls node:16-alpine image.
  + Runs node --version inside the container.
  + Container terminates after job.

**Example 2: Multi-Stage Pipeline (Frontend + Backend)**

groovy

pipeline {

agent none

stages {

stage('Backend') {

agent { docker { image 'maven:3.8' } }

steps { sh 'mvn --version' }

}

stage('Frontend') {

agent { docker { image 'node:16' } }

steps { sh 'npm --version' }

}

}

}

**Pipeline Syntax Generator**

* Use **"Pipeline Syntax"** in Jenkins to generate code snippets (e.g., Git checkout, shell commands).

**6. CI/CD with Kubernetes & Argo CD**

**Workflow:**

1. **Jenkins Pipeline**:
   * Checks out Python app code from Git.
   * Builds Docker image & pushes to Docker Hub.
   * Updates Kubernetes manifest (deployment.yaml) in Git with new image tag.
2. **Argo CD (GitOps)**:
   * Watches Git repo for changes.
   * Automatically deploys new version to Kubernetes.

**Jenkinsfile Snippet:**

groovy

stage('Deploy to Kubernetes') {

steps {

sh 'sed -i "s/image:.\*/image: abhishek/todo-app:${BUILD\_NUMBER}/" deployment.yaml'

sh 'git commit -am "Update image tag to ${BUILD\_NUMBER}"'

sh 'git push origin main'

}

}

**7. Jenkins Interview Questions**

**Common Questions:**

1. **Explain your CI/CD workflow.**
   * Mention stages (build, test, deploy) and tools (Jenkins, Docker, Argo CD).
2. **How do you handle worker node failures?**
   * Traditional: Monitor VMs, auto-scaling.
   * Modern: Use Docker agents (no permanent workers).
3. **How do you secure Jenkins?**
   * Use RBAC, limit plugin permissions, enable HTTPS.

**Final Notes**

* **GitHub Repo**: Clone/Fork for ready-to-use Jenkinsfiles.
* **Assignment**: Extend the pipeline (e.g., add a DB stage with MySQL).
* **Subscribe**: For more DevOps tutorials!