**JENKINS PIPELINE**

**1. Simple Pipeline with Stages**

A Jenkins Pipeline consists of stages, each representing a specific part of the CI/CD process. Here's a basic pipeline that runs three stages: build, test, and deploy.

groovy

Copy code

pipeline {

agent any // Run on any available agent

stages {

stage('Build') {

steps {

echo 'Building the application...'

// Add build steps like Maven, Gradle, npm, etc.

}

}

stage('Test') {

steps {

echo 'Running tests...'

// Add test steps like unit tests, integration tests

}

}

stage('Deploy') {

steps {

echo 'Deploying application...'

// Add deployment steps like pushing to production

}

}

}

}

**2. Defining Variables**

You can define variables in a Jenkinsfile to use across the pipeline. These variables can be assigned within the pipeline for flexibility.

groovy

Copy code

pipeline {

agent any

environment {

APP\_VERSION = '1.0.0' // Assign a basic version variable

DEPLOY\_ENV = 'production'

}

stages {

stage('Build') {

steps {

echo "Building version ${APP\_VERSION}..."

}

}

stage('Deploy') {

steps {

echo "Deploying to environment: ${DEPLOY\_ENV}..."

}

}

}

}

**3. Conditional Execution (Using when)**

You can make the pipeline smarter by running stages conditionally. For example, only deploy if the branch is main.

groovy

Copy code

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building the application...'

}

}

stage('Test') {

steps {

echo 'Running tests...'

}

}

stage('Deploy') {

when {

branch 'main' // Only deploy on the main branch

}

steps {

echo 'Deploying to production...'

}

}

}

}

**4. Parallel Execution**

Jenkins supports running multiple stages in parallel, which can reduce the time for tasks that can be executed independently.

groovy

Copy code

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building the application...'

}

}

stage('Test') {

parallel {

stage('Unit Tests') {

steps {

echo 'Running unit tests...'

}

}

stage('Integration Tests') {

steps {

echo 'Running integration tests...'

}

}

}

}

}

}

**5. Post Actions (Success, Failure, Always)**

You can define actions that will run after the stages, depending on the result of the pipeline (success, failure, or always).

groovy

Copy code

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building the application...'

// Simulate a failure

// error('Build failed!')

}

}

}

post {

success {

echo 'The build was successful!'

}

failure {

echo 'The build failed!'

}

always {

echo 'This will always run, no matter what.'

}

}

}

These examples give you a starting point for basic assignments with Jenkins Pipeline scripts. You can expand on these by integrating them with tools like Maven, Gradle, Docker, or Kubernetes as needed.