Groovy-based Jenkins Pipeline DSL (Domain-Specific Language) is powerful for defining continuous integration and delivery workflows.

1. Pipeline Block

Defines the entire Jenkins Pipeline script.

pipeline {

agent any

stages {

// Define stages here

}

// Other pipeline configurations

}

2. Agent

Specifies where the Pipeline will execute.

agent {

label 'docker' // Execute on a node with the label 'docker'

}

3. Stages

Divides the Pipeline into stages.

stages {

stage('Build') {

// Define steps for this stage

}

stage('Test') {

// Define steps for this stage

}

// Additional stages

}

4. Steps

Actions or tasks performed within a stage.

steps {

sh 'mvn clean package' // Execute Maven build

junit 'target/surefire-reports/\*\*/\*.xml' // Publish JUnit test results

}

5. Environment Variables

Define environment variables accessible throughout the Pipeline.

environment {

PATH = '/usr/local/bin'

VERSION = '1.0'

}

6. Parameters

Accept input from users or external systems.

parameters {

string(name: 'DEPLOY\_ENV', defaultValue: 'dev', description: 'Environment to deploy')

}

7. Post Actions

Actions to execute after the Pipeline or specific stages.

post {

success {

// Actions to perform on successful completion

}

failure {

// Actions to perform on failure

}

}

8. Timeouts

Define timeout for a stage or entire Pipeline.

options {

timeout(time: 1, unit: 'HOURS') // Timeout entire Pipeline after 1 hour

}

9. Input

Pause Pipeline and wait for user input.

stage('Deploy') {

input {

message 'Deploy to production?'

ok 'Deploy'

}

steps {

// Deployment steps

}

}

10. Conditions

Execute steps based on conditions.

when {

branch 'master'

environment name: 'DEPLOY\_ENV', value: 'prod'

}

11. Checkout SCM

Checkout source code from version control system.

stage('Checkout') {

steps {

checkout scm

}

}

12. Build

Trigger and wait for another Jenkins job.

build job: 'my-job', parameters: [string(name: 'VERSION', value: '1.0')]

13. Environment Blocks

Define environment variables within a stage.

stage('Build') {

environment {

BUILD\_DIR = "${WORKSPACE}/build"

}

steps {

sh 'mkdir -p $BUILD\_DIR'

}

}

14. Docker

Execute steps within a Docker container.

agent {

docker {

image 'maven:3.8.1-jdk-11'

args '-v /tmp:/tmp'

}

}

15. Tools

Define tools installations for the Pipeline.

tools {

maven 'Maven-3.8.1' // Use specific Maven tool installation

jdk 'JDK-11' // Use specific JDK tool installation

}

16. Retry

Retry a stage or steps upon failure.

stage('Test') {

options {

retry(3) // Retry stage up to 3 times on failure

}

steps {

// Test steps

}

}

17. Matrix

Execute stages in a matrix of configurations.

matrix {

axes {

axis {

name 'OS'

values 'Linux', 'Windows'

}

}

stages {

stage('Build') {

steps {

// Build steps

}

}

}

}

18. Parameters Blocks

Define parameters within specific stages.

stage('Deploy') {

parameters {

string(name: 'TARGET\_ENV', defaultValue: 'prod', description: 'Environment to deploy')

}

steps {

// Deployment steps

}

}

19. Branches

Execute Pipeline based on specific branches.

when {

branch 'master'

}

20. Triggers

Configure triggers to start the Pipeline execution.

triggers {

cron('H \*/4 \* \* 1-5') // Execute every 4 hours on weekdays

}

21. Wrapper Blocks

Apply wrappers to the Pipeline or specific stages.

wrapper {

timeout(time: 3, unit: 'HOURS') // Timeout wrapper for specific stage

}

22. Environment Blocks

Define environment variables within a stage.

stage('Build') {

environment {

BUILD\_DIR = "${WORKSPACE}/build"

}

steps {

sh 'mkdir -p $BUILD\_DIR'

}

}

23. Locks

Control concurrency with lock resources.

stage('Deploy') {

locks {

resource 'production'

}

steps {

// Deployment steps

}

}

24. Tools Blocks

Define tools installations for the Pipeline.

tools {

maven 'Maven-3.8.1' // Use specific Maven tool installation

jdk 'JDK-11' // Use specific JDK tool installation

}

25. Milestone

Mark a stage as a milestone.

stage('Build') {

milestone 1

steps {

// Build steps

}

}

26. Library

Load shared libraries or functions into the Pipeline.

@Library('my-shared-library') \_

27. Post Blocks

Execute actions after a stage or the entire Pipeline.

post {

always {

echo 'Pipeline finished'

}

success {

echo 'Pipeline succeeded'

}

failure {

echo 'Pipeline failed'

}

}

28. Docker DSL

Interact with Docker containers using DSL methods.

docker.image('maven:3.8.1-jdk-11').inside {

sh 'mvn clean install'

}

29. Git

Use Git operations within the Pipeline.

stage('Checkout') {

steps {

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

}

}

30. JUnit

Publish JUnit test results within the Pipeline.

stage('Test') {

steps {

sh 'mvn test'

junit 'target/surefire-reports/\*\*/\*.xml'

}

}

These concepts and DSL statements cover a wide range of functionalities and configurations available in Jenkins Pipeline using Groovy syntax. They enable you to define sophisticated CI/CD workflows with Jenkins, allowing for flexibility, automation, and scalability in your software delivery processes.