Jenkins Declarative Pipelines - Detailed Notes

# Overview

Declarative Pipeline is a newer and more structured syntax for Jenkins pipelines. It provides a predefined and opinionated syntax designed to simplify pipeline creation while still offering powerful features.

# Key Characteristics

- Language: Declarative DSL (Domain Specific Language)

- Structure: pipeline { ... }

- Flexibility: Moderate (less than Scripted)

- Readability: High

- Error-prone: Lower due to syntax checks

- IDE Support: Better due to stricter format

# Basic Structure

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building...'

}

}

stage('Test') {

steps {

echo 'Testing...'

}

}

stage('Deploy') {

steps {

echo 'Deploying...'

}

}

}

}

# Agent Directive

- `agent any`: Runs the pipeline on any available agent.

- Can also specify specific labels or docker containers.

Example:

pipeline {

agent {

label 'linux'

}

}

# Stages and Steps

- `stages`: Contains one or more stage blocks.

- `stage`: Defines a logical segment of the pipeline.

- `steps`: Contains the actual tasks to run.

Each stage must have a steps block or parallel block.

# Environment Variables

pipeline {

environment {

ENV\_VAR = 'value'

}

...

}

# Post Block

Used for defining actions that run after the pipeline or a stage completes.

Example:

post {

success {

echo 'Pipeline succeeded.'

}

failure {

echo 'Pipeline failed.'

}

always {

echo 'This always runs.'

}

}

# Parallel Execution

pipeline {

agent any

stages {

stage('Parallel Stage') {

parallel {

stage('Branch A') {

steps {

echo 'Running A'

}

}

stage('Branch B') {

steps {

echo 'Running B'

}

}

}

}

}

}

# Input Step

Used to pause pipeline and wait for user input.

Example:

stage('Approval') {

steps {

input message: 'Approve deployment?'

}

}

# When Conditions

Controls whether a stage should run.

Example:

stage('Deploy') {

when {

branch 'main'

}

steps {

echo 'Deploying on main branch'

}

}

# Best Practices

- Use Declarative for standard CI/CD workflows

- Combine with Shared Libraries for reusability

- Use `post` blocks for cleanup and notifications

- Use `when` for conditional stage execution

- Use environment blocks for consistent configuration

# Scripted vs Declarative

Declarative Pipeline:

- Syntax: DSL (opinionated)

- Flexibility: Moderate

- Readability: Higher

- Error Prone: Lower

- Suitability: Standard CI/CD workflows

Scripted Pipeline:

- Syntax: Groovy

- Flexibility: High

- Readability: Lower

- Error Prone: Higher

- Suitability: Complex pipelines

# When to Use Declarative Pipelines

- Simpler and more readable CI/CD workflows

- Teams new to Jenkins

- Consistent, manageable structure

- Built-in validation and error handling