Jenkins supports several types of pipelines to automate the process of building, testing, and deploying applications. The two main types are:

**1. Declarative Pipeline**

This is the most modern and recommended way to write Jenkins pipelines. It uses a structured and predefined syntax.

**Example: Declarative Pipeline**

pipeline {

    agent any

    environment {

        APP\_NAME = 'MyApp'

    }

    stages {

        stage('Build') {

            steps {

                echo "Building ${APP\_NAME}"

                sh 'mvn clean install'

            }

        }

        stage('Test') {

            steps {

                echo 'Running tests...'

                sh 'mvn test'

            }

        }

        stage('Deploy') {

            steps {

                echo 'Deploying application...'

                sh './deploy.sh'

            }

        }

    }

    post {

        success {

            echo 'Pipeline completed successfully!'

        }

        failure {

            echo 'Pipeline failed.'

        }

    }

}

**Key Features:**

* Easy to read and write.
* Built-in error handling (post block).
* Better validation and editor support.

**2. Scripted Pipeline**

This is a more flexible and powerful option, written in pure Groovy. It’s useful for complex workflows but harder to maintain.

**Example: Scripted Pipeline**

node {

    def appName = 'MyApp'

env.APP\_NAME = 'MyApp'

    stage('Build') {

        echo "Building ${appName}"

        sh 'mvn clean install'

    }

    stage('Test') {

        echo 'Running tests...'

        sh 'mvn test'

    }

    stage('Deploy') {

        echo 'Deploying application...'

        sh './deploy.sh'

    }

    echo 'Pipeline completed.'

}

**To run a Jenkins pipeline on a specific node labeled sai, you can use the agent { label 'sai' } directive. Here's a complete example of a Declarative Pipeline that targets the node with the label sai:**

**✅ Jenkins Pipeline for Node Label sai**

pipeline {

    agent { label 'sai' }

    environment {

        APP\_NAME = 'MyApp'

    }

    stages {

        stage('Build') {

            steps {

                echo "Building ${APP\_NAME} on node: ${env.NODE\_NAME}"

                sh 'echo Simulating build...'

            }

        }

        stage('Test') {

            steps {

                echo "Running tests on node: ${env.NODE\_NAME}"

                sh 'echo Simulating tests...'

            }

        }

        stage('Deploy') {

            steps {

                echo "Deploying ${APP\_NAME} from node: ${env.NODE\_NAME}"

                sh 'echo Simulating deployment...'

            }

        }

    }

    post {

        success {

            echo 'Pipeline completed successfully!'

        }

        failure {

            echo 'Pipeline failed.'

        }

    }

}

**✅ Scripted Pipeline Using Node Label sai**

node('sai') {

    def appName = 'MyApp'

    stage('Build') {

        echo "Building ${appName} on node: ${env.NODE\_NAME}"

        sh 'echo Simulating build...'

    }

    stage('Test') {

        echo "Running tests on node: ${env.NODE\_NAME}"

        sh 'echo Simulating tests...'

    }

    stage('Deploy') {

        echo "Deploying ${appName} from node: ${env.NODE\_NAME}"

        sh 'echo Simulating deployment...'

    }

    echo 'Pipeline completed.'

}

**Script{}**

Absolutely! Let's dive into a **detailed explanation of script {}** in a Jenkins **Declarative Pipeline**, with **multiple examples** to show how and when to use it.

**🧠 What is script {} in Jenkins?**

In a **Declarative Pipeline**, most of the syntax is structured and limited to predefined steps. But sometimes, you need more flexibility—like using if, for, or custom logic. That’s where script {} comes in.

It allows you to write **Scripted Pipeline (Groovy)** code inside a Declarative Pipeline.

**✅ Basic Syntax**

script {

    // Groovy code goes here

}

**📘 Example 1: Conditional Execution**

pipeline {

    agent any

    stages {

        stage('Check Condition') {

            steps {

                script {

                    def shouldRun = true

                    if (shouldRun) {

                        echo 'Condition met. Running step.'

                    } else {

                        echo 'Condition not met. Skipping step.'

                    }

                }

            }

        }

    }

}

**🔍 What’s happening?**

* shouldRun is a Groovy variable.
* The if statement decides whether to run the step.

**📘 Example 2: Looping Over a List**

pipeline {

    agent any

    stages {

        stage('Loop Example') {

            steps {

                script {

                    def items = ['apple', 'banana', 'cherry']

                    for (item in items) {

                        echo "Processing item: ${item}"

                    }

                }

            }

        }

    }

}

**🔍 Why use this?**

* Declarative syntax doesn’t support loops directly.
* script {} lets you iterate over data dynamically.

**📘 Example 3: Try-Catch for Error Handling**

pipeline {

    agent any

    stages {

        stage('Try Catch Example') {

            steps {

                script {

                    try {

                        sh 'exit 1' // Simulate failure

                    } catch (err) {

                        echo "Caught an error: ${err}"

                        currentBuild.result = 'UNSTABLE'

                    }

                }

            }

        }

    }

}

**🔍 Why use this?**

* To **gracefully handle errors** and avoid failing the entire pipeline.

**📘 Example 4: Dynamic Stage Skipping**

pipeline {

    agent any

    environment {

        RUN\_TESTS = 'false'

    }

    stages {

        stage('Test') {

            when {

                expression {

                    return env.RUN\_TESTS == 'true'

                }

            }

            steps {

                echo 'Running tests...'

            }

        }

        stage('Always Run') {

            steps {

                echo 'This always runs.'

            }

        }

    }

}

You can also use script {} to dynamically set env.RUN\_TESTS earlier in the pipeline.

**⚠️ Best Practices**

* ✅ Use script {} **only when necessary**.
* ❌ Avoid putting entire stages inside script {}—it reduces readability.
* ✅ Keep logic **modular and clean**.
* **❌ Not Needed in Scripted Pipelines**
* In a **Scripted Pipeline**, everything is already Groovy, so you **don’t need script {}**. For example:

node {

    def shouldRun = true

    if (shouldRun) {

        echo 'Running step'

    }

}

**✅ Example 5: Try-Catch-Finally**

node {

    try {

        stage('Build') {

            echo 'Building...'

            sh 'exit 1'  // Simulate failure

        }

    } catch (err) {

        echo "Caught error: ${err}"

    } finally {

        echo 'Cleaning up...'

    }

}