**when {}**

In a **Jenkins Pipeline**, the when {} block is used inside a stage to control whether or not that stage should be executed. It allows you to define **conditions** under which the stage will run. This is especially useful in **declarative pipelines**.

Here’s a **detailed breakdown** of how when {} works and some common use cases:

**🔧 Basic Syntax**

pipeline {

    agent any

    stages {

        stage('Build') {

            when {

                branch 'main'

            }

            steps {

                echo 'Building on main branch...'

            }

        }

    }

}

In this example, the Build stage will only run if the branch is main.

**🧠 Common when Conditions**

1. **branch**
2. when {
3. branch 'develop'
4. }

Runs the stage only if the branch is develop.

1. **Expression**

when {

    expression {

        return env.BUILD\_NUMBER.toInteger() % 2 == 0

    }

}

Runs the stage only if the build number is even.

1. **Environment**

when {

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

}

Runs the stage only if the environment variable DEPLOY\_ENV is set to production.

1. **Equals**

when {

    equals expected: 'true', actual: env.IS\_RELEASE

}

Runs the stage if IS\_RELEASE is true.

1. **Not**

when {

    not {

        branch 'feature'

    }

}

Runs the stage if the branch is **not** feature.

1. **anyOf / allOf**

when {

    anyOf {

        branch 'main'

        branch 'develop'

    }

}

Runs the stage if the branch is either main or develop.

**Example: Combining Conditions**

stage('Deploy') {

    when {

        allOf {

            branch 'main'

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

        }

    }

    steps {

        echo 'Deploying to production...'

    }

}

This stage runs **only** if:

* The branch is main
* The environment variable DEPLOY\_ENV is production

**input**

In a **Jenkins declarative pipeline**, the input block is used to **pause the pipeline execution** and wait for **human approval or input** before continuing. This is especially useful for manual approval steps, such as before deploying to production.

**🧾 Basic Syntax**

stage('Approval') {

    steps {

        input "Do you want to proceed with deployment?"

    }

}

This will pause the pipeline and show a prompt in the Jenkins UI. A user must click **"Proceed"** to continue.

**🛠️ Advanced Usage**

You can customize the input block with parameters and submitter restrictions:

stage('Manual Approval') {

    steps {

        input message: 'Approve deployment?',

              ok: 'Deploy Now',

              submitter: 'admin,devops',

              parameters: [

                  string(name: 'VERSION', defaultValue: '1.0.0', description: 'Version to deploy'),

                  booleanParam(name: 'SKIP\_TESTS', defaultValue: false, description: 'Skip tests?')

              ]

    }

}

**Explanation:**

* **message**: The prompt shown to the user.
* **ok**: The label on the confirmation button.
* **submitter**: Only users in this list can approve.
* **parameters**: Collects input values from the user.

**📦 Using Input Parameters Later**

You can use the input values in later stages like this:

environment {

    DEPLOY\_VERSION = ''

}

stage('Get Approval') {

    steps {

        script {

            def userInput = input message: 'Approve deployment?', parameters: [

                string(name: 'VERSION', defaultValue: '1.0.0', description: 'Version to deploy')

            ]

            env.DEPLOY\_VERSION = userInput

        }

    }

}

stage('Deploy') {

    steps {

        echo "Deploying version ${env.DEPLOY\_VERSION}"

    }

}

**timeout(time: 5, unit: 'MINUTES')**

In a **Jenkins declarative pipeline**, timeout(time: 5, unit: 'MINUTES') is used to **limit the execution time** of a block of code—typically a stage, steps, or script block. If the block takes longer than the specified time, Jenkins will **abort** it automatically.

**✅ Example: Timeout in a Stage**

pipeline {

    agent any

    stages {

        stage('Wait for Approval') {

            options {

                timeout(time: 5, unit: 'MINUTES')

            }

            steps {

                input "Do you want to proceed?"

            }

        }

    }

}

In this example:

* Jenkins will wait **up to 5 minutes** for user input.
* If no one responds within that time, the pipeline will **fail with a timeout error**.

**🧠 Where You Can Use timeout**

You can use timeout in:

* options block of a stage
* script block
* steps block (with timeout as a wrapper)

**🧪 Example: Timeout Around a Script Block**

stage('Long Running Task') {

    steps {

        timeout(time: 5, unit: 'MINUTES') {

            script {

                // Simulate a long-running task

                sh 'sleep 600'

            }

        }

    }

}

Here, the shell command sleep 600 (10 minutes) will be **interrupted after 5 minutes**.

**retry(n)**

In a **Jenkins declarative pipeline**, retry(n) is used to **automatically retry a block of code** up to n times if it fails. This is helpful for handling **intermittent failures**, such as flaky tests, unstable network calls, or temporary infrastructure issues.

**✅ Basic Syntax**

stage('Flaky Step') {

    steps {

        retry(3) {

            sh 'curl http://unstable-service/api'

        }

    }

}

In this example:

* Jenkins will try to run the curl command.
* If it fails, it will **retry up to 3 times** before marking the stage as failed.

**🧠 How It Works**

* If the block inside retry(n) **succeeds**, the pipeline moves on.
* If it **fails**, Jenkins will **retry** it up to n times.
* If it still fails after n attempts, the pipeline **fails** that stage.

**🧪 Example with timeout and retry**

stage('Download Artifact') {

    steps {

        retry(2) {

            timeout(time: 1, unit: 'MINUTES') {

                sh 'wget http://example.com/artifact.zip'

            }

        }

    }

}

This will:

* Try to download the artifact.
* If it fails or times out, it will retry **once more** (total 2 attempts).

**Parallel**

In a **Jenkins declarative pipeline**, the parallel block is used to run **multiple branches (sets of steps or stages) at the same time**. This is useful for speeding up pipelines by executing independent tasks concurrently—like running tests on different platforms or environments.

**✅ Basic Syntax**

pipeline {

    agent any

    stages {

        stage('Parallel Tasks') {

            parallel {

                stage('Task A') {

                    steps {

                        echo 'Running Task A...'

                        sh 'sleep 5'

                    }

                }

                stage('Task B') {

                    steps {

                        echo 'Running Task B...'

                        sh 'sleep 3'

                    }

                }

            }

        }

    }

}

**🔍 What Happens Here:**

* Task A and Task B run **simultaneously**.
* The pipeline waits for **both** to finish before moving to the next stage.

**🧠 Key Notes**

* Each branch inside parallel must be a **stage**.
* All branches **must succeed** for the pipeline to continue.
* If one branch fails, the others are **aborted** unless failFast: false is set.

**🛠️ Advanced Example with failFast**

stage('Parallel Testing') {

    failFast true

    parallel {

        stage('Unit Tests') {

            steps {

                sh './run-unit-tests.sh'

            }

        }

        stage('Integration Tests') {

            steps {

                sh './run-integration-tests.sh'

            }

        }

    }

}

* failFast true: If one test fails, the other is **stopped immediately**.
* You can also use failFast false to let all branches finish regardless of failures.