Jenkins Declarative Pipeline Tutorial

Table of Contents

- Introduction
- Structure of Declarative Pipeline.
  - Pipeline
  - Agent
  - Stages
  - Stage
  - Steps
  - Environment (Defined at stage or pipeline level)
  - Input (Defined at stage level)
  - options (Defined at stage or pipeline level)
  - When
  - Parallel
  - Parameters
  - Post
  - Tools
  - Triggers

Introduction

Jenkins Pipeline is the workflow that implements the Continuous Delivery pipeline with the Jenkins features, tools, and plugins. There are two different ways to create a Jenkins pipeline. One is a Declarative Pipeline, and another is a Scripted Pipeline. In this article, we will see how to create a Jenkins Declarative pipeline. This tutorial will give you insight into the structure and directives of declarative pipeline and syntax. Let's Get into the Jenkins Declarative Pipeline Tutorial.

Structure of Declarative Pipeline.

As part of The Jenkins Declarative Pipeline Tutorial, We will discuss the structure of the Declarative Pipeline first. Declarative Pipeline starts with the “Pipeline” label. This pipeline label opens the block where the following directives and blocks are present.

- pipeline
- agent
- stages
- stage
- steps
- environment (Defined at stage or pipeline level)
- input (Defined at stage level)
- options (Defined at stage or pipeline level)
- parallel
- parameters
- post
- script
- tools
- triggers
- when

Let us see all these directives and blocks one by one in this Jenkins Declarative Pipeline Tutorial.

Pipeline

The Pipeline is a block where all other blocks and directives are declared. This is the start of the Declarative pipeline. So, Pipeline label will be declared as follows.

```
pipeline {
        agent…
        stages…
        step..
}
```

Agent

Jenkins is popular for one of its best features called distributed build process to the agent nodes. Jenkins build pipeline plugin ensures the same feature present in the pipeline that is created in the Declarative method. Normally, the agent will be declared at the top level as a global agent declaration. So, the Agent label is ideally mentioned with the following parameters.

- **any** – This means the pipeline will run in any available node. (`agent any`)
- **none** – Ideally None means, the pipeline will run in not run in any agent at the top-level, but each stage should be defined with its own agent (`agent none`).
- **label** – This means the pipeline will be mentioned as label name and pipeline will look for the available node with the label name mentioned (`agent {label ‘my label name for node’}`)
- **node** – mentioning node in the agent is same as mentioning label but this will give us more option like custom Workspace (`agent {node{label ‘my label name’}}`).
- **docker** – By mentioning docker, the pipeline will run in a docker environment. (`agent{ docker {image ‘ruby3:develop’ label ‘my label name’ arg ‘–port 3000:80’} } ) )`

Stages

Stages is the section which contain one or more stage tag. So, In this section we can write the part of continuous delivery process.

Example:

```
pipeline {
    agent any
    stages {
        stage('Example') {
        ...
        ...
        }
    }
}
```

Stage

This is the section that contains steps section or agent section which is isolated for the particular stage and other directives. So, A Stage section will have a command that runs processes like Build, Deploy, Test, or any other Continuous Delivery process.

Example:

```
pipeline {
    agent any
    stages {
        stage('Example') {
            agent { docker 'maven:3-alpine' }
                steps {
                    sh 'mvn compile'
            }
        }
    }
}
```

Steps

A stage can have multiple commands to run for completing the continuous delivery process. basically, Step is the set of commands that performs an action and It will be executed one by one.

Example:

```
pipeline {
    agent any
    stages {
        stage('Example') {
            steps {
                sh 'mvn compile'
            }
        }
    }
}
```

Environment (Defined at stage or pipeline level)

The environment is the directive that contains the Key-value pairs of the environment variable that should be available for the steps that are going to be executed in the stages.

Ideally, Environment directive supports a special helper method called credentials type which is very helpful for dynamically passing the credentials into the steps. So, Following are the types of credentials that can be passed into the environments

- **Secret Text** – String-based
- **Secret file** – Location of the file
- **Username and Password** – As in the name, It is specified as "username:password"
- **SSH with Private Key** – Location of the SSH key file and optionally "username:password"

Example:

```
pipeline {
    agent any
    environment {
        example key = 'example value'
    }
    stages {
        stage('Example') {
            environment {
                Secret_key = credentials('secret-text')
            }
        }
    }
}
```

Input (Defined at stage level)

Input Directive is declared inside stage and it will help us getting input by prompting users. Basically, The stage section will wait until any Input is submitted. So, the following are the configuration options.

- **Message** – This is to show some message to the user
- **ID** – This is the Default value
- **Ok** – This is the "OK" Button
- **Submitter** – is the comma-separated value list of users who are allowed to enter the input.
- **submitterParameter –** This will add submitter's name into the environment variable
- **parameters –** This is the list of parameters that need to be provided by the submitter.Line 1

Example:

```
pipeline {
    agent any
    stages {
        stage('Example') {
            input {
                message "Can we Proceed?"
                ok "Yes"
                submitter "Digital Varys"
                parameters {
                    string(name: 'PERSON', defaultValue: 'DigiralVarys', description:
'Member')
                }
            }
            steps {
                echo "${PERSON}, is proceeding..."
            }
        }
    }
}
```

options (Defined at stage or pipeline level)

Options derivatives will provide options for configuring the pipeline with in the pipeline. So, Let us discuss the options one by one.

- **buildDiscarder** – To discard this build after a particular time (Example: `options {buildDiscarder(logRotator(daysToKeepStr: '7'))} )`
- **checkoutToSubdirectory** – Checkout the particular subdirectory from the SCM (For Example: `options { checkoutToSubdirectory('foldername')}`)
- **disableConcurrentBuilds** – This will restrict the parallel access od shared resources and stop the concurrent Build process. (For example: `options { disableConcurrentBuilds() }`)
- **newContainerPerStage** – This is used in top-level agent declaration and this will allow the process to run in the different Container each time. (For example: `options {newContainerPerStage () } )`
- **preserveStashes** – To preserve the stash from the completed build process (For example: `options { preserveStashes(buildCount: 5) }`)
- **quietPeriod** – This is the pause period or sleep time (For example: `options { quietPeriod(30) }`)
- **retry** – This is will enable retrying the failed build for the specified times (For Example: `options { retry(3) }`)
- **skipDefaultCheckout** – This will skip the default SCM checkout process in the agent (For Example: `options { skipDefaultCheckout() }`)
- **skipStagesAfterUnstable** – This will skip the mentioned stage after it comes to UNSTABLE state (For Example: `options { skipStagesAfterUnstable() }`)
- **timeout** – This is the timeout period setting at pipeline level (For Example: `options { timeout(time: 30, unit: 'MINUTES') }`)

Example:

```
pipeline {
    agent any
    options {
        timeout(time: 30, unit: 'MINUTES')
    }
    stages {
        stage('Example') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

When

"When" is the conditional derivate of the Jenkins declarative pipeline. Just like "IF" conditions, "when" will have the one or more condition statement. This can be either for from the user input or from the Built-in conditions.

"When" condition can be placed before "Agent" in stage or before input directive. If "beforeAgent" is set true, "when" condition will be executed before the agent declaration. If "beforeInput" is set true, "when" condition will be executed before the Input derivative.

So, Let us see some built-in conditions.

- **Branch** – This will execute the stage only when the branch in SCM is matching. (`when { branch 'master' }`)
- **buildingTag** – This executes only when the Build is building the tag (`when { buildingTag() }`)
- **changelog** – Stage will run only when the changelog is having a specified pattern in it. (`when { changelog '.*^\\[SERVICE2\\] .+$' }`)
- **changeset** – This will run only when the SCM changeset is having the specified pattern in file (`when { changeset "**/*.js" }`)
- **changeRequest** – ChangeRequest is like the pull request in GitHub and the stage will run only after the change request is raised (`when { changeRequest target: 'master' }`)
- **environment** – Stage will execute only when the specified environment variable is matching (`when { environment name: 'DEPLOY_TO', value: 'production' }`)
- **equals** – Executes only when the expected value and the actual value is matching in the condition (`when { equals expected: 2, actual: currentBuild.number }`)
- **expression** – Condition is created with groovy expression and expects the Boolean true to pass. (`when { expression { return params.DEBUG_BUILD } }`)
- **tag** – Stage will be executed only after the specified tag name pattern matches the stage tag (`when { tag "release-*" }`)
- **not** – This is like "NOT" in the "IF" condition. (`when { not { branch 'master' } }`)
- **allOf** – This is for nested condition and this will expect all should be true. This is like "AND" in "IF" Condition (`when { allOf { branch 'master'; environment name: 'DEPLOY_TO', value: 'production' } }`)
- **anyOf** – anyOf is like "OR" in the "IF" condition. (`when { anyOf { branch 'master'; branch 'staging' } }`)
- **triggeredBy** – the stage will execute only when it is triggered by the specified build. (`when { triggeredBy 'SCMTrigger' }`)

Example:

```
pipeline {
    agent any
    stages {
        stage('Example Build') {
            when {
                anyOf {
            branch 'master'; branch 'staging'
        }
            }
            steps {
                echo 'Hello World'
            }
        }
        stage('Example Deploy') {
            when {
                branch 'production'
            }
            steps {
                echo 'Deploying'
            }
        }
    }
}
```

Parallel

Parallel block will allow us to run the stages concurrently. Idealy, Stages that are declared inside the Parallel derivative will run concurrently at the same time. If you want to stop the build

```
pipeline {
    agent any
    stages {
        stage('NORMAL Stage') {
            steps {
                echo 'I am one'
            }
        }
        stage('Parallel Stage') {
            when {
                branch 'master'
            }
            failFast true
            parallel {
                stage('stage one') {
                    agent {
                        label "stageonebranch"
                    }
                    steps {
                        echo "Me in stage one"
                    }
                }
                stage('Stage two') {
                    agent {
                        label "stage two"
                    }
                    steps {
                        echo "Me in stage two"
                    }
                }
                stage('Stage three') {
                    agent {
                        label "Stage Three"
                    }
                }
            }
        }
    }
}
```

Parameters

Parameter Directive will provide the trigger parameters to the pipeline. Following are the available Parameters

- **string** – simple parameter in string datatype (`parameters { string(name: 'yyyyy', defaultValue: 'XXX', description: 'Hello world') })`
- **text** – multiline text datatype (`parameters { text(name: 'Demo', defaultValue: '', description: 'Demo parameter')})`
- **booleanparam** – Boolean type parameter (`booleanParam(name: 'Boolean', defaultValue: true, description: 'Boolean value'))`
- **choice** – it is like a multiple-choice parameter (`parameters { choice(name: 'CHOICE', choices: ['A', 'B', 'C'], description: 'Choose one'))`
- **password** – Password type parameter (`password(name: 'PASSWORD', defaultValue: 'Key', description: 'Enter a password'))`

Example:

```
pipeline {
    agent any
    parameters {
        string(name: 'yyyyy', defaultValue: 'XXX', description: 'Hello world')
        text(name: 'Demo', defaultValue: '', description: 'Demo parameter')
        booleanParam(name: 'Boolean', defaultValue: true, description: 'Boolean value')
        choice(name: 'CHOICE', choices: ['A', 'B', 'C'], description: 'Choose one')
        password(name: 'PASSWORD', defaultValue: 'Key', description: 'Enter a password')
        file(name: "FILE", description: "file to upload")
    }
    stages {
        stage('Example') {
            steps {
                echo "Hello ${params.yyyyy}"
                echo "Biography: ${params.Demo}"
                echo "Toggle: ${params.Boolean }"
                echo "Choice: ${params.CHOICE}"
                echo "Password: ${params.PASSWORD}"
            }
        }
    }
}
```

Post

Post is another directive which is like “When”. This will execute some commands after the stage or steps as per the condition. the following are some examples of the conditions.

- Always
- Changed
- Fixed
- Regression
- Aborted
- Failure
- Success
- Unstable
- Unsuccessful
- cleanup

Example:

```
pipeline {
    agent any
    stages {
        stage('Example') {
            steps {
                echo 'Hello World'
            }
        }
    }
    post {
        always {
            echo 'Running after the stages'
        }
    }
}
```

Tools

This will install and configure the tools in pipeline. This block will support following tools

- **maven**
- **jdk**
- **gradle**

Example:

```
pipeline {
    agent any
    tools {
        maven 'apache-maven-1.0.1'
    }
    stages {
        stage('Example') {
            steps {
                sh 'mvn compile'
            }
        }
    }
}
```

Triggers

Triggers are the typical Build triggers. This can be happening in any form like cron, pollscm, Upstream.

- **Cron** – Build job will run in a regular interval (`triggers { cron('H */4 * * 1-5') }`)
- **pollSCM** – Check for the changes in SCM in regular interval to run the Build job (`triggers { pollSCM('H */4 * * 1-5') }`)
- **upstream** – Upstreaming another build job (`triggers { upstream(upstreamProjects: 'job1,job2', threshold: hudson.model.Result.SUCCESS) }`)

Example:

```
pipeline {
    agent any
    triggers {
        cron('H */4 * * 1-5')
    }
    stages {
        stage('Example') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```