Jenkins is an open source Continuous Integration(CI) and Continuous Delivery(CD) Build tool for software applications.

This tool developed on java version, hence java installation required.

Jenkins is a tool used for both CI and CD

**Why Jenkins used?**

Jenkins is used in application or software development by developers. It helps automatize the process of below tasks for

* Integration module process
* Development test process that includes unit and integration test
* End to end Automation test
* Building applications
* Testing
* Deployment of apps to different environment such as Dev, QA, Stage and Production

**Jenkin Features**

* It triggers a build for every code commit.
* Easy to integrate with Different source git repositories such as GitLab, GitHub and Bitbucket etc.
* Automates Build and test phase
* Send notifications about integration status, failure
* Support Java, Python and Javascript build tools etc.
* Deployed artifacts

Jenkins provides two ways of developing a pipeline — scripted and declarative.

Traditionally, Jenkins jobs were created using Jenkins UI called freestyle jobs. In Jenkins 2.0, Jenkins introduced a new way to create jobs using the technique called pipeline as code. In the **pipeline as code** technique, jobs are created using a script file that contains the steps to be executed by the job. In Jenkins, that scripted file is called a **Jenkinsfile**.

A Jenkinsfile can be written in two aspects — **scripted pipeline syntax**and **declarative pipeline syntax.**

## Jenkins Scripted Pipeline

Jenkins pipelines are traditionally written as scripted pipelines.

The end-to-end scripted pipeline script is written in Groovy.

 It requires knowledge of Groovy programming as a prerequisite.

 The Jenkinsfile starts with the word **node**.

### **Sample Scripted Pipeline**

A black text on a white background

Description automatically generated

**Node:** The node is a machine on which Jenkins runs is called a node. A node block is used in scripted pipeline syntax.

node{

}

**parameters**

The parameters directive provides a way for Jenkins jobs to interact with Jenkins CI/CD users during the running of the build job.

A parameter can be of the following types: string, text, booleanParam, choice, and password.

**string**– Accepts a value of String type from Jenkins user.

E.g. string(name: ‘NAME’, description: ‘Please tell me your name?’)

**text** – Accepts multi line value from Jenkins user E.g. text(name: ‘DESC’, description: ‘Describe about the job details’)

**booleanParam** – Accepts a true/false value from Jenkins user E.g. booleanParam(name: ‘SKIP\_TEST’, description: ‘Want to skip running Test cases?’)

**choice**– Jenkins user can choose one among the choices of value provided E.g. choice(name: ‘BRANCH’, choices: [‘Master’, ‘Dev’], description: ‘Choose the branch’)

**password**– Accepts a secret like password from Jenkins user E.g. password(name: ‘SONAR\_SERVER\_PWD’, description: ‘Enter SONAR password’)

pipeline {

agent any

parameters

string(name: 'NAME', description: 'Please tell me your name?')

text(name: 'DESC', description: 'Describe about the job details')

booleanParam(name: 'SKIP\_TEST', description: 'Want to skip running Test cases?')

choice(name: 'BRANCH', choices: ['Master', 'Dev'], description: 'Choose branch')

password(name: 'SONAR\_SERVER\_PWD', description: 'Enter SONAR password')

stages

stage('Printing Parameters')

steps

echo "Hello $params.NAME"

echo "Job Details: $params.DESC"

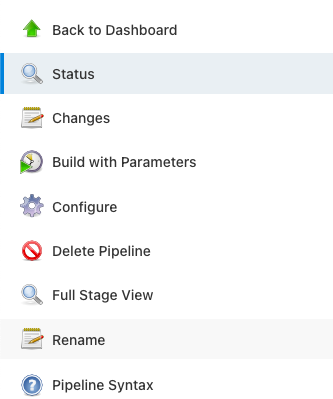
echo "Skip Running Test case ?: $params.SKIP\_TEST"

echo "Branch Choice: $params.BRANCH"

echo "SONAR Password: $params.SONAR\_SERVER\_PWD"

}

If a parameters directive is used in the pipeline, Jenkins CI/CD can sense that it needs to accept the user’s input while running the job. Hence, Jenkins will change the **Build now** link in the left side menu to a **Build with parameters** link.



When we click on the **Build with Parameters** link, Jenkins CI/CD will let us pass values for the parameters we configured in the declarative pipeline.

A screenshot of a web page

Description automatically generated

Once we enter each parameter’s values, we can hit the **Build** button to run the job.

**script**

The script block helps us run **Groovy** code inside the Jenkins declarative pipeline.

## Declarative Pipeline

The Declarative Pipeline subsystem in Jenkins Pipeline is relatively new, and provides a simplified, opinionated syntax on top of the Pipeline subsystems.

* The latest addition in Jenkins pipeline job creation technique.
* Jenkins declarative pipeline needs to use the predefined constructs to create pipelines. Hence, it is not flexible as a scripted pipeline.
* A Jenkinsfile starts with the word **pipeline**.

Jenkins declarative pipelines should be the preferred way to create a Jenkins job, as they offer a rich set of features, come with less learning curve, and there are no prerequisites to learn a programming language like Groovy just for the sake of writing pipeline code.

We can also validate the syntax of the declarative pipeline code before running the job. It helps to avoid a lot of runtime issues with the build script.

A white background with black text

Description automatically generated

## Declarative Pipeline Syntax

Declarative pipelines contain one or more declarative steps or directives, as explained below.

**pipeline**

Entire Declarative pipeline script should be written inside the pipeline block. It’s a **mandatory** block.

This is the user-defined block, which contains all the processes such as build, test, deploy, etc. it is a group of all the stages in a Jenkinsfile. All the stages and steps are defined in this block. It is used in declarative pipeline syntax.

pipeline{

}

**agent**

Specify where the Jenkins build job should run. agent can be at pipeline level or stage level. It’s mandatory to define an agent.

#### **Possible Values**

1. any– Run Job or Stage on any available agent.
2. none– Don’t allocate any agent globally for the pipeline. Every stage should specify their own agent to run.
3. label – Run the job in agent which matches the label given here. Remember Jenkins CI/CD can work on Master/Agent architecture. Master nodes can delegate the jobs to run in Agent nodes. Nodes on creation given a name and label to identify them later, e.g., all Linux nodes can be labeled as linux-machine.

A black text on a white background

Description automatically generated

1. docker **–** Run the job in a given Docker container.

**stages**

The**stages**block constitutes different executable stage blocks. At least one stage block is mandatory inside the stages block.

A black text on a white background

Description automatically generated

**Stage:** This block contains a series of steps in a pipeline. i.e., build, test, and deploy processes all come together in a stage. Generally, a stage block visualizes the Jenkins pipeline process.

Let's see an example for multiple stages, where each stage performs a specific task:

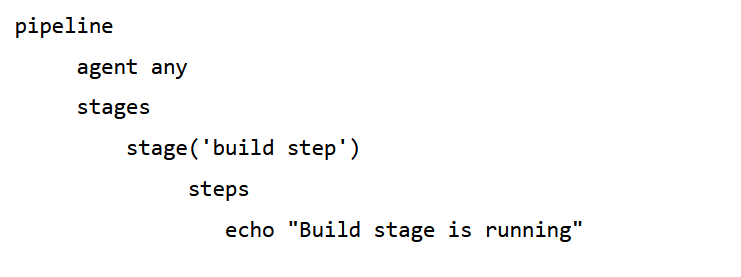
A screenshot of a computer code

Description automatically generated

### **steps**

The steps block contains the actual build step. It’s mandatory to have at least one step block inside a stage block.

Depending on the Agent’s operating system (where the Jenkins job runs), we can use shell, bat, etc., inside the steps command.



**Step:** A step is a single task that executes a specific process at a defined time. A pipeline involves a series of steps defined within a stage block.

A computer code with text

Description automatically generated with medium confidence

**script**

The script block helps us run **Groovy** code inside the Jenkins declarative pipeline.

pipeline {

agent any

parameters

string(name: 'NAME', description: 'Please tell me your name')

choice(name: 'GENDER', choices: ['Male', 'Female'], description: 'Choose Gender')

stages {

stage('Printing name')

steps

script

def name = "$params.NAME"

def gender = "$params.GENDER"

if(gender == "Male")

echo "Mr. $name"

else

echo "Mrs. $name"

}

}

The script block is wrapped inside the steps block. In the above example, we are printing the name passed as parameter prefixed with Mr. or Mrs. based on the gender chosen.

A screenshot of a computer

Description automatically generated

We used **build with parameters** to pass params at the time of running the job.