**Jenkins Tutorial**

**TABLE OF CONTENT**

* [What Is CI (Continuous Integration)?](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#CI)
* [What Is Jenkins?](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#Jenkins)
* [What Is Jenkins Pipeline?](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#JenkinsPipeline)
* [Why Use Jenkins Pipeline?](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#Why)
* [What Is Jenkinsfile?](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#Jenkinsfile)
* [How To Install & Run Jenkins On Windows?](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#Install)
* [Jenkins Pipeline Tutorial Concepts](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#Tutorial)
* [Integrating Jenkins Pipeline With Cloud Selenium Grid](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#Integrating)
* [Creating A Jenkins Pipeline & Running Our First Test](https://www.lambdatest.com/blog/jenkins-pipeline-tutorial/#FirstTest)

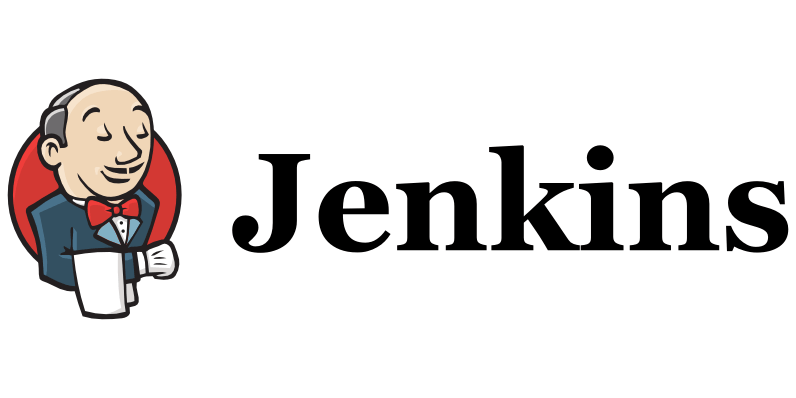
**What Is CI (Continuous Integration)?**

Continuous Integration is a development approach in which members work on the same project coordinate to integrate their work more frequently. The code is integrated into a shared repository, and any integration is tested by automated test cases or sequences to look for an error. Each team member is expected to integrate their code at least once a day or more as and when required.

The CI process consists of four key elements which help in the execution:

* Hosted CI Tool Solution
* Virtual Machine
* Version Control System
* Tools

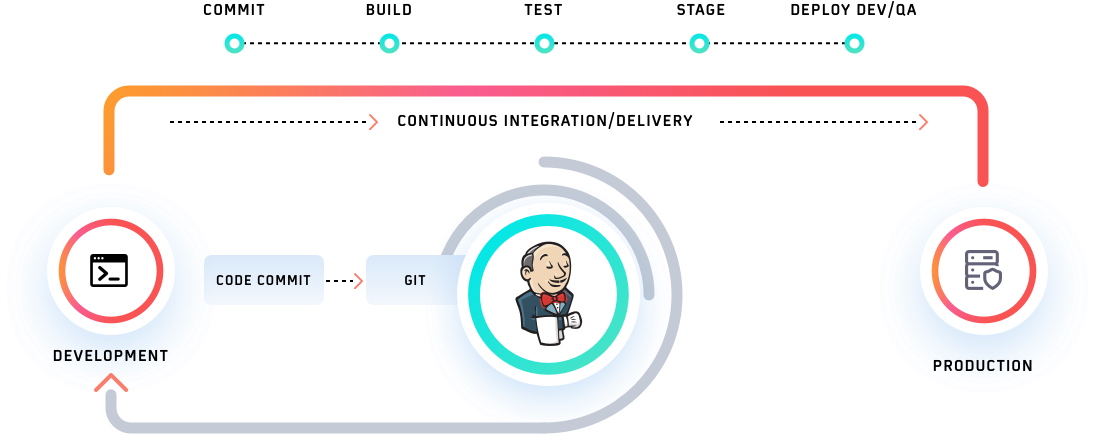
## What Is Jenkins?



Jenkins CI/CD is a continuous integration open-source tool. It has been developed using Java. This allows real-time monitoring and recording of discrete improvements to a more comprehensive codebase. It lets developers easily identify, fix bugs in their codebase, and simplify their builds’ validation. Jenkins is the most popular CI/CD tool because it closely monitors repetitive jobs and assists in automated execution during a project’s production.

Jenkins CI/CD is a cloud-based system that can work as a standalone application running on a server of its own. Alternatively, it can also utilize a web server like Glassfish, JBoss, or WebSphere. Using Jenkins, developers can speed their product creation cycle because Jenkins can simplify the build and test instantly. Jenkins CI/CD promotes the entire software development cycle (SDLC) by designing, testing, monitoring, deploying, and other phases of the cycle.

**What Is Jenkins Pipeline?**



Inside Jenkins CI/CD, a pipeline is defined as a series of events or tasks which are interconnected in a particular order. In simple terms, Jenkins pipeline is a set of modules or plugins which enable the implementation and integration of Continuous Delivery pipelines within Jenkins.

The Jenkins pipeline has an expandable automation system for building basic or complicated ‘template’ distribution pipelines via the Domain-specific language (DSL) used in the pipeline. There are four states of Continuous Delivery in Jenkins pipeline-

* Build
* Deploy
* Test
* Release

## Why Use Jenkins Pipeline?

As explained above in this Jenkins pipeline tutorial, Jenkins CI/CD plays a significant role in delivering high-quality applications or products. We are now aware that Jenkins has proved to be a specialist in Continuous Integration, Continuous Testing, and Continuous Delivery. It uses a feature called Jenkins pipeline for Continuous Delivery, which is basically the ability to release apps regularly at an interval. This process ensures that the software is always ready for production.

### **Advantages Of Jenkins Pipeline**

* By using Groovy DSL (Domain Specific Language), it models easy to complex pipelines as code.
* The code is stored in the form of a text file called ‘Jenkinsfile’ that can be scanned into Source Code Management.
* It supports complex pipelines by adding conditional loops, forks, or joining operations and allowing parallel execution tasks.
* It improves user experience by integrating user feedback into the pipeline.
* It’s resilient in terms of Jenkins’ master unplanned restart.
* It can resume from checkpoints saved.
* It can incorporate multiple additional plugins and add-ins.

**What Is Jenkinsfile?**

To move ahead with our Jenkins pipeline tutorial, we need to understand the role of Jenkinsfile. It is a text file that stores the whole process as code in our local machine. It can be reviewed in a Source Code Management (SCM) platform such as Git. It is instrumental as it helps the developers to view, edit, and test the code whenever required.

The Jenkinsfile is written using the Groovy Domain-Specific Language and can be generated using a text editor or the Jenkins instance configuration tab.

There are two different types in which the Jenkins pipeline can be constructed. These are the syntaxes-

* Declarative pipeline syntax
* Scripted pipeline syntax

The [Declarative Pipelines](https://www.lambdatest.com/blog/jenkins-declarative-pipeline-examples/) is a relatively new feature that supports the concept of code pipeline. It enables the reading and writing of the pipeline code. This code is written within a Jenkinsfile, which can be tested into a tool such as Git for source control.

The **Scripted**pipeline is a typical method of code writing. The Jenkinsfile is written on the Jenkins user interface instance in this pipeline.

While both of these pipelines are Groovy-based, the scripted pipeline uses more strict Groovy-based syntaxes. This is because it was the first groovy foundation pipeline that was created for use. As this Groovy script was not usually suitable to all users, it introduced the declarative pipeline to provide a simpler and more flexible Groovy syntax. The declarative pipeline is defined within a ‘pipeline’ block, while the scripted pipeline is defined within a ‘node’ block.

**Jenkins Pipeline Tutorial Concepts**

We hope that Jenkins is up and running in your Windows system, following the steps mentioned above in this Jenkins pipeline tutorial. Before moving ahead with Jenkins test automation with Selenium, it is time to get accustomed to some of this Jenkins pipeline tutorial’s crucial concepts.

Here they are-

**Pipeline**

It is a user-defined framework that includes all the processes like create, check, deploy, etc. In a Jenkinsfile, it’s a list of all the levels. All of the stages and steps within this block are described. This is the fundamental block to the syntax of a declarative pipeline.



|  |  |
| --- | --- |
| 1  2  3 | pipeline  {    } |

**Node**

A node is a system running a complete workflow. It’s an integral part of the syntax of the scripted pipeline.



|  |  |
| --- | --- |
| 1  2  3 | node  {    } |

Some standard sections are available to both declarative and scripted pipelines. These are:

**Agent**

An agent is described as a directive that can run multiple builds using just one Jenkins instance. This feature helps spread the workload to various agents and execute multiple projects within Jenkins’s single instance. It instructs Jenkins to assign the builds to an executor.

A single agent may be defined for a whole Jenkins pipeline, or different agents may be assigned to execute each stage within a pipeline. Some of the most commonly used Agent parameters are:

1. Any

Runs the stage pipeline on any available agent.

1. None

This parameter is added to the root of the pipeline. It means that there is no global agent for the entire pipeline, and each stage must define its own agent.

1. Label

Performs on the labeled agent the pipeline/stage.

1. Docker

This parameter uses a docker container as a pipeline execution environment or as a specific level. For example, the docker can be used to pull an image of Ubuntu. This image can now be used to run multiple commands as an execution environment.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | pipeline {      agent {          docker {              image  'ubuntu'                  }              }          } |

**Stages**

This section includes all of the work that needs to be completed. The work is defined in the form of stages. Within this Directive, there may be more than one level. Each stage executes a particular task.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | pipeline {  agent any      stages {          stage ('Build') {            }          stage ('Test') {            }          stage ('QA') {            }          stage ('Deploy') {            }          stage ('Monitor') {            }        }  } |

**Steps**

Within a stage block, the pipeline can be described as a series of steps. Such steps are performed in sequence for the execution of a level. Within a Steps guideline, there must be at least one step.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | pipeline {      agent any          stages {              stage ('Build') {                  steps {                      echo                      'Running build phase. '                  }              }          }      } |

## Jenkins Shared Library

In today’s Microservices world, large monolith applications are broken down into small applications & deployed separately. Interestingly, most of these applications have the same steps to build, test & deploy the application. So instead of repeating those steps for every application, we can actually create a shared library, which can be used across applications.

### **Why Use Jenkins Shared Library?**

Let’s say we have five different Spring Boot applications in our Microservice architecture. All of them need to be built using Maven, run Unit & Integration tests to ensure the integrity of code, packed as a JAR/WAR file & pushed to the artifactory.

Typically, all five applications need their own Jenkinsfile, but the content of the Jenkinsfiles is going to be mostly the same. Jenkins Shared Library avoids this repetition of pipeline code by creating a shared library.

In brief, here are steps to create & use the Jenkins shared library:

* Create a separate git repo for the Jenkins pipeline library & push the shared library code to that repo.
* Integrate the shared library repo in Jenkins under the Manage Jenkins section.
* Create Jenkinsfile in the project. In that Jenkinsfile, Import & use the shared library.