## What is Jenkins?

Jenkins is an open-source server that is written entirely in Java. It lets you execute a series of actions to achieve the continuous integration process, that too in an automated fashion.

This CI server runs in servlet containers such as Apache Tomcat. Jenkins facilitates [continuous integration and continuous delivery](https://www.lambdatest.com/blog/what-is-continuous-integration-and-continuous-delivery/) in software projects by automating parts related to build, test, and deployment. This makes it easy for developers to continuously work on the betterment of the product by integrating changes to the project.

Jenkins automates the software builds in a continuous manner and lets the developers know about the errors at an early stage. A strong Jenkins community is one of the prime reasons for its popularity. Jenkins is not only extensible but also has a thriving plugin ecosystem.

Some of the possible steps that can be performed using Jenkins are:

* Software build using build systems such as Gradle, Maven, and more.
* Automation testing using test frameworks such as Nose2, PyTest, Robot, Selenium, and more.
* Execute test scripts (using Windows terminal, Linux shell, etc.
* Achieve test results and perform post actions such as printing test reports, and more.
* Execute test scenarios against different input combinations for obtaining improved test coverage.
* Continuous Integration (CI) where the artifacts are automatically created and tested. This aids in identification of issues in the product at an early stage of development.

At the time of what is Jenkins blog, it had close to 1500+ plugins contributed by the community. Plugins help in customizing the experience with Jenkins, along with providing support for accelerating activities related to building, deploying, and automating a project.

### History Of Jenkins

Jenkins has an early mover advantage since it has been in development since 2011. Kohsuke Kawaguchi created Jenkins (then called ‘Hudson’) while working at Sun Microsystems. Hudson was created in the summer of 2004 and the first release was in February 2005.

After the acquisition of Sun Microsystems by Oracle, a proposal was approved by the Hudson community for creating the Jenkins project. In February 2011, Oracle intended that the development of Hudson should continue hence, Hudson was forked instead of renaming it to Jenkins.

Though Hudson and Jenkins were being developed independently, Jenkins acquired significantly more projects & contributors than Hudson. Consequently, Hudson is no longer maintained by the community.

### Jenkins Release Cycle

Like other open-source projects, Jenkins also produces two release lines – LTS (Long-Term Support) and Weekly (regular) releases. Jenkins is very good with releases, as stable releases happen every four weeks.

At the time of this ‘, what is Jenkins’ article, the latest version of Jenkins LTS was 2.235.2, and Jenkins Weekly was 2.249.

### Salient Features Of Jenkins

Jenkins is more functionality-driven rather than UI-driven hence, there is a learning curve involved in getting to know what is Jenkins. Here are the powerful developer-centric features offered by Jenkins:

#### ****1. Easy Installation & Configuration****

Jenkins is a self-contained Java program that is agnostic of the platform on which it is installed. It is available for almost all the popular operating systems such as Windows, different flavors of Unix, and Mac OS.

It is available as a normal installer, as well as a .war file. Once installed, it is easy to configure using its web interface.

#### ****2. Open-Source****

As it is open-source, it is free for use. There is a strong involvement of the community which makes it a powerful CI/CD tool. You can take support from the Jenkins community, whether it is for extensibility, support, documentation, or any other feature related to Jenkins.

#### ****3. Thriving Plugin Ecosystem****

The backbone of Jenkins is the community and the community members have been instrumental in the development (and testing) of close to 1500+ plugins available in the Update Center.

#### ****4. Easy Distribution****

Jenkins is designed in such a manner that makes it relatively easy to distribute work across multiple machines and platforms for the accelerated build, testing, and deployment.

This Jenkins Tutorial for beginners and professionals will help you learn how to use Jenkins, one of the most popular CI/CD tools used in DevOps.

## How Does Jenkins Work?

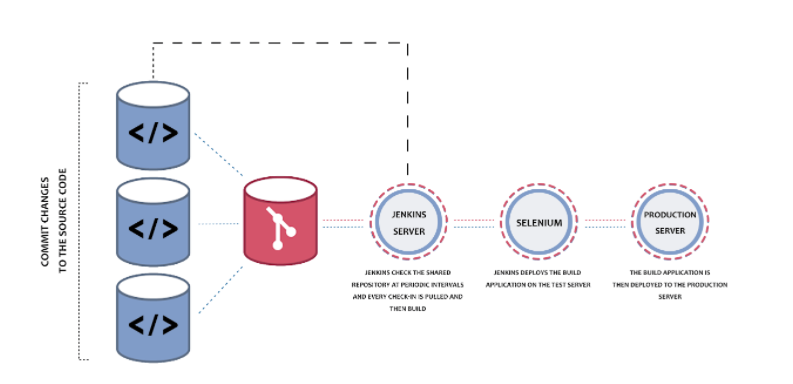
In this section of the What is Jenkins blog, we look at the internal functioning of Jenkins i.e. what happens once the developer commits changes to the repository and how CI/CD is realized in Jenkins. We also look at the Master-Agent architecture in Jenkins.

### Architecture Of Jenkins

Before we dive into how does Jenkins works, we must understand the architecture of Jenkins. These are the series of steps that outlines the interaction between different elements in Jenkins:

* Developers do the necessary modifications in the source code and commit the changes to the repository. A new version of that file will be created in the version control system that is used for maintaining the repository of source code.
* The repository is continuously checked by the Jenkins CI server for any changes (either in the form of code or libraries) and **changes are pulled by the server**.
* In the next step, we ensure that the build with the ‘pulled changes’ is going through or not. The Build server **performs a build with the code and an executable** is generated if the build process is successful. In case of a build failure, an automated email with a link to build logs and other build artifacts is sent to the developer.
* In case of a successful build, the built application (or executable) is deployed to the test server. This step helps in realizing **continuous testing where the newly built executable goes through a series of automated tests**. Developers are alerted in case the changes have caused any breakage in functionality.
* If there are no build, integration, and testing issues with the checked-in code, the changes and tested application are automatically **deployed to the Prod/Production server**.

Here is the diagrammatic representation of the Jenkins architecture:



### Master- Agent Architecture In Jenkins

The master-agent (or distributed) architecture in Jenkins is used for managing distributed builds. The Master and Agent(s) communicate through the TCP/IP protocol.

These are the roles and responsibilities of the Jenkins Master and Agent(s):

#### ****Jenkins Master****

The main server in Jenkins is the Master. Here are the jobs handled by Jenkins Master:

* Schedule build jobs
* Choosing the appropriate agent in the master-agent ecosystem for dispatching the builds.
* Monitor agents and take them online/offline as and when required.
* Presenting the build results (and reports) to the developer.

The Jenkins master can also execute the jobs directly but it is always recommended to select the appropriate agent(s) for build and execution-related tasks.

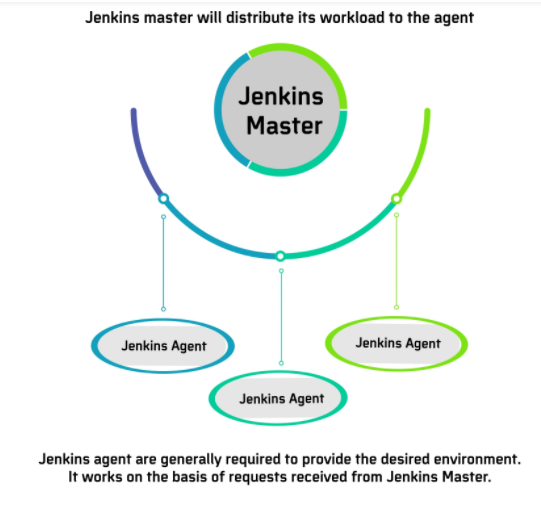
#### ****Jenkins Agent(s)****

A agent is a remote machine that is connected to the Master. Depending on the project and build requirements, you could opt for ‘N’ number of agents. agents can run on different operating systems and depending on the ‘type of build request’, the appropriate Agent is chosen by the Master for build execution and testing.

Here are the jobs handled by the Jenkins Agent(s):

* Listen to commands from the Jenkins Master.
* Execute build jobs that are dispatched by the Master.
* Developers have the flexibility to run the build and execute tests on a particular agent or a particular type of Agent. The default option is Jenkins Master selecting the best-suited Agent for the job.

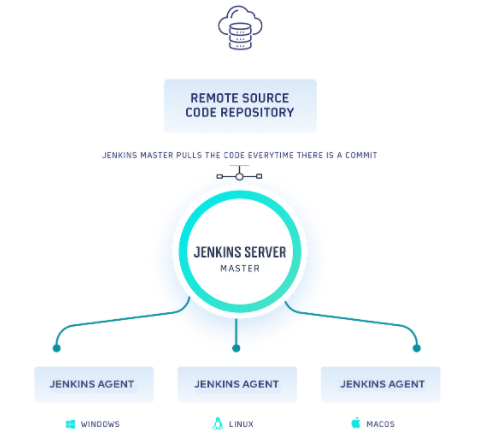
Here is a simple diagrammatic representation of how does Jenkins work, with multiple Jenkins Agents connected to the Jenkins Master:



### How Does Jenkins Work In Master-Agent Architecture?

In the previous section of the What is Jenkins blog, we touched upon the brief responsibilities of Master and Agent(s) in Jenkins. Let’s look at exactly how does Jenkins works in the Master-Agent (or distributed) architecture:

In the Jenkins Master-Agent architecture shown below, there are three Agents, each running on a different operating system (i.e. Windows 10, Linux, and Mac OS).



### Prerequisites For Jenkins Installation

We need JRE (Java Runtime Environment).

**Jenkins-Installation– Ubuntu**

Refer this for ubuntu - <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-20-04>

[root@ip-172-31-5-203 ~]# cat /var/lib/jenkins/secrets/initialAdminPassword🡪 File to get admin password.

a1258dccccd44060ba4ba1d7a65e58af

## What Is Jenkins Pipeline?

Pipeline in Jenkins is a group of jobs (or events) that are interlinked in a particular sequence. Jenkins Pipeline is a set or suite of plugins that provides support for implementation and integration of Continuous Delivery pipelines into Jenkins.

The Pipeline also provides a set of tools that are useful for modeling simple as well as complex delivery pipelines ‘as code’ through ‘Pipeline Domain-Specific Language (DSL)’ syntax.

Every job in the Jenkins pipeline has some dependency on one or more events. Continuous delivery pipeline in Jenkins consists of four states – Build, Deploy, Test, and Release. Each of these states consist of events that execute in a sequence.

### What Is Jenkinsfile?

Now that you understand what is Jenkins pipeline, we can dive deeper into the concept. The entire definition of a Jenkins Pipeline is written into a text file called Jenkinsfile. It contains the steps required for running a Jenkins Pipeline. ‘Pipeline as code’ can be implemented using Jenkinsfile and Domain Specific Language (DSL) is used for defining the same.

Jenkinsfile can also be committed to the source control repository of the project. With Jenkinsfile, the CD Pipeline is also treated as a part of the application that is versioned, committed, and reviewed like any other piece of code.

Some of the major benefits of Jenkinsfile are:

* Single Jenkinsfile can be used for creating a Pipeline build process for all the branches and executing pull requests.
* The implementation in a Pipeline can be reviewed like normal source code.
* Audit trail of the Pipeline.
* Singular source for the Pipeline can be viewed as well as edited by multiple members associated with the project.

Though Pipeline can be defined either in web UI or with a Jenkinsfile, it is recommended to define what is Jenkins Pipeline in a Jenkinsfile and maintain the same in a source control repository.

### Syntax For Defining A Jenkinsfile

To define what is Jenkins pipeline, Jenkinsfile can be written using the following types of Syntax:

#### ****a. Declarative****

[Declarative Pipeline](https://www.lambdatest.com/blog/jenkins-declarative-pipeline-examples/) is a recent feature of Jenkins Pipeline that makes reading and writing the Pipeline code a lot easier. Unlike Scripted Syntax with Pipelines, Declarative Syntax helps in controlling different aspects of the Pipeline in a simplistic manner.

#### ****b. Scripted****

Scripted Pipeline is a more traditional way of writing a Jenkins Pipeline as code. It uses traditional Groovy-based syntax. The Web UI of Jenkins is normally used for writing the Scripted Pipeline in a Jenkinsfile.

### Important Concepts Of Jenkins Pipeline

Moving on with what is Jenkins pipeline, here are some basic concepts that need to be well-understood if you are planning to use the Jenkins Pipeline:

#### ****a. Pipeline****

The Pipeline consists of a set of instructions written as code. It defines the entire build process, which ideally consists of different stages for building, testing, and delivering the application.

#### ****b. Node****

A node is a machine that is a part of the Jenkins environment. The Jenkins Pipeline executes on a node block that is usually a part of the Scripted Pipeline syntax.

#### ****c. Stage****

Stage in a Jenkins Pipeline consists of a unique subset of tasks such as Build, Test, Deploy, etc. The Stage block is used by many plugins for providing the visualization of Jenkins status (and progress).

#### ****d. Step****

Step is a single task that tells Jenkins what exactly needs to be done. For example, setting an environment variable can be done in a step, executing a build command can also be a step. On the whole, a Jenkins Pipeline constitutes a series of steps.

### Advantages Of Using Jenkins Pipeline

Jenkins Pipeline is instrumental in adding a rich set of automation tools onto Jenkins. Hence, Jenkins can be used for simple continuous integration as well as for comprehensive CD pipelines.

These are some of the primary reasons for using the Jenkins Pipeline:

* As the Jenkins Pipeline is implemented as code, it can be checked in the source control repository. Teams can view, edit, as well as iterate upon the delivery pipeline.
* Jenkins Pipelines are robust. A Pipeline is automatically resumed in case the server witnesses an unplanned restart.
* Pipeline process can pause and wait for input for any input from the user.
* Pipelines are versatile as they can be used for realizing complex CD requirements, including performing work in parallel.
* Jenkins Pipelines are extensible by plugin developers and users with Pipeline Shared Libraries.

### Declarative Pipeline Sample

Here is a simple example of a Declarative Pipeline-



The pipeline blocks define the work that has been done through the defined Pipeline. Listed down here are some of the essential details of this Pipeline:

* 1. Agent ‘any’ means the Pipeline stages can be executed on any available Agent.
  2. ‘Build’, ‘Test’, and ‘Deploy’ are the different stages in the pipeline. Each of them will run in a sequential manner.
  3. Instructions inside ‘Steps’ indicates the work that needs to be performed as a part of a particular ‘Stage’. For example, sh ‘build command’ can be used for triggering a build command in the ‘Build’ stage.

We have not covered the Scripted Pipeline, as that is beyond the scope of this blog. In the later part of the What is Jenkins blog, we would demonstrate the usage of the Declarative Pipeline for the Maven project.