Jenkins Tutorial

Jenkins Tutorial is designed for both beginners and professionals. Our Tutorial provides all the basic and advanced concepts of Jenkins, such as Jenkins installation, Jenkins Configuration, Jenkins Pipeline, etc.

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins builds and tests our software projects, which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

What is Jenkins?

Jenkins is an **open source automation tool** written in Java programming language that allows **continuous integration**.

Jenkins **builds** and **tests** our software projects which continuously making it easier for **developers to integrate changes to the project**, and making it easier for users to obtain a fresh build.

It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including **build, document, test, package, stage, deploy static analysis** and much more.

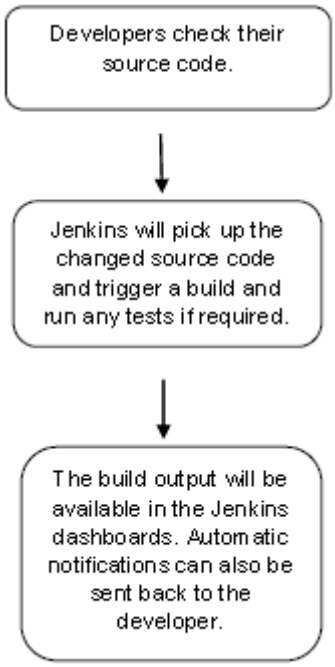
Jenkins achieves CI (Continuous Integration) with the help of plugins. **Plugins is used to allow the integration of various DevOps stages**. If you want to **integrate a particular tool**, you have to **install the plugins for that tool**. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

For example: If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

- Perform a **software build** using a **build system** like Gradle or **Maven Apache**
- **Execute a shell script**
- **Archive a build result**
- **Running software tests**

Work Flow:



History of Jenkins

Kohsuke Kawaguchi, who is a Java developer, working at SUN Microsystems, was tired of building the code and fixing errors repetitively. In 2004, he created an automation server called **Hudson** that automates build and test task.

In 2011, Oracle who owned Sun Microsystems had a dispute with **Hudson** open source community, so they forked Hudson and renamed it as **Jenkins**.

Both Hudson and Jenkins continued to operate independently. But in short span of time, Jenkins acquired a lot of contributors and projects while Hudson remained with only 32 projects. Then with time, Jenkins became more popular, and Hudson is not maintained anymore.

What is Continuous Integration?

Continuous Integration *(CI)* is a development practice in which the **developers are needs to commit changes to the source code** in a shared repository at regular intervals. Every commit made in the repository is then built. This allows the development teams to detect the problems early.

Continuous integration requires the developers to have regular builds. The general practice is that whenever a code commit occurs, a build should be triggered.

Continuous Integration with Jenkins

Let's consider a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to *develop software*, but this process has many problems.

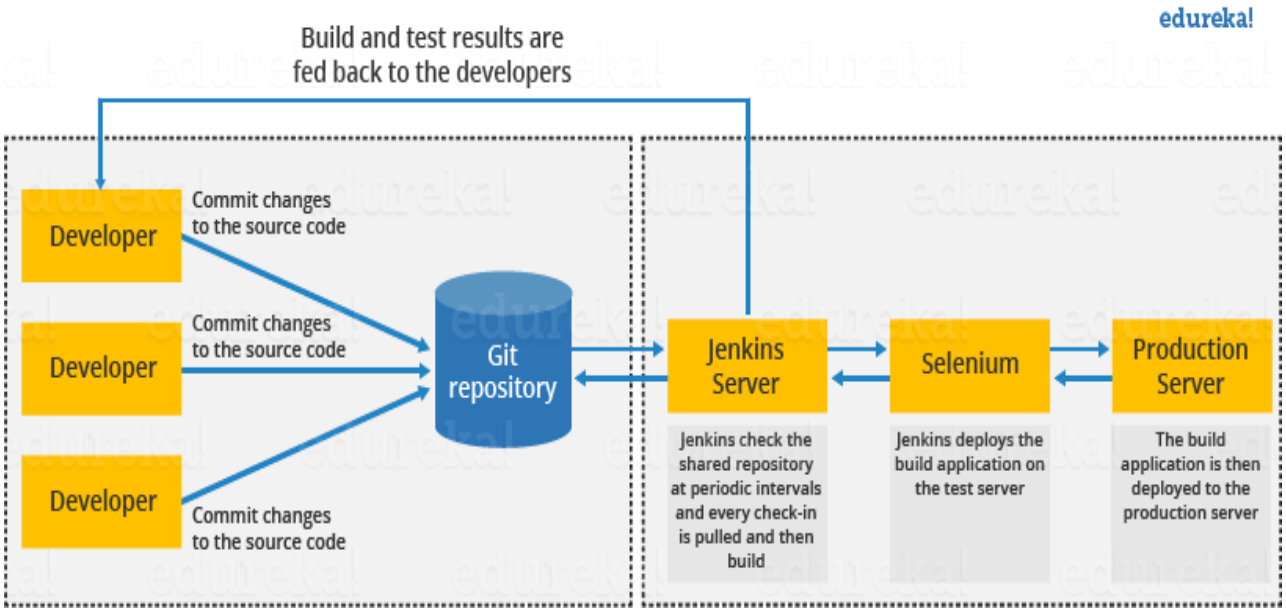
- Developer teams have to **wait** till the complete software is developed for the test results.
- There is a high prospect that the test results might show **multiple bugs**. It was tough for developers to locate those bugs because they have **to check the entire source code** of the application.
- It **slows** the software delivery process.
- Continuous feedback pertaining to things like architectural or **coding issues, build failures, test status and file release uploads** was missing due to which the quality of software can go down.
- The whole **process** was **manual** which increases the threat of **frequent failure**.

It is obvious from the above stated problems that not only the software delivery process became **slow** but **the quality of software also went down**. This leads to customer dissatisfaction.

So to overcome such problem there was a need for a system to exist where **developers can continuously trigger a build and test for every change** made in the source code.

This is what Continuous Integration (CI) is all about. Jenkins is the most **mature Continuous Integration tool** available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.

Let's see a generic flow diagram of Continuous Integration with Jenkins:



Let's see how Jenkins works. The above diagram is representing the following functions:

- First of all, a developer **commits the code** to the source code repository. Meanwhile, the Jenkins **checks the repository at regular intervals** for changes.
- Soon after a commit occurs, the **Jenkins server finds the changes** that have occurred in the source code repository. Jenkins will draw those changes and will **start preparing a new build**.
- If the **build fails**, then the concerned team will be notified.
- If built is **successful**, then **Jenkins server deploys** the built in the test server.
- **After testing**, Jenkins server generates a **feedback** and then **notifies** the developers about the **build and test results**.
- It will **continue** to verify the source code repository for **changes** made in the source code and the **whole process keeps on repeating**.

Advantages and Disadvantages of using Jenkins

Advantages of Jenkins

- It is an **open-source tool**.
- It is **free of cost**.
- It **does not require additional installations or components**. Means it is easy to install.
- Easily configurable.
- It supports **1000 or more plugins** to ease your work. If a **plugin does not exist**, you can **write the script** for it and share with community.
- It is built in java and hence it is portable.
- It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
- Easy support, since it open source and widely used.
- Jenkins also **supports cloud-based architecture** so that we can deploy Jenkins in cloud-based platforms.

Disadvantages of Jenkins

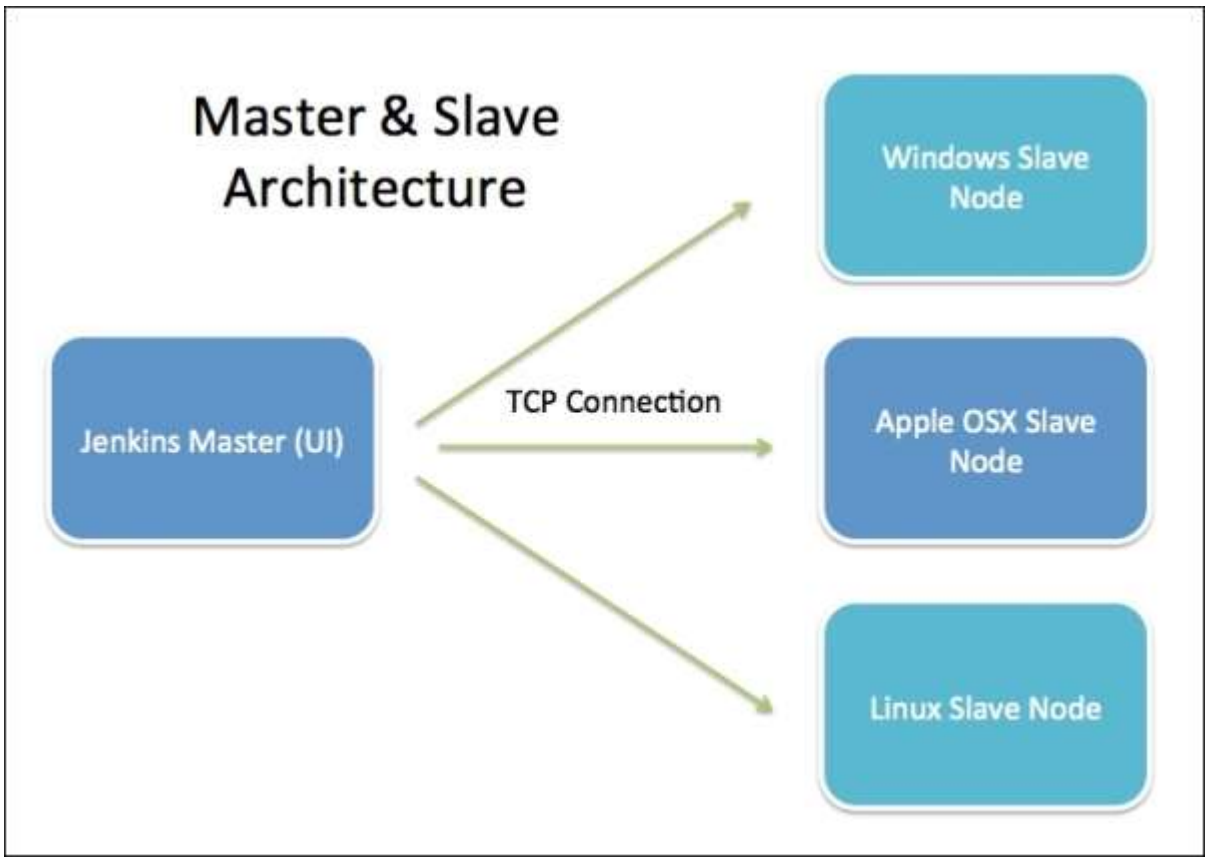
- Its interface is **out dated** and **not user friendly** compared to current user interface trends.
- **Not easy to maintain** it because it runs on a server and requires **some skills as server administrator** to monitor its activity.
- **CI regularly breaks** due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

Jenkins Architecture

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has **two** components:

- Jenkins Master/Server
- Jenkins Slave/Node/Build Server



Jenkins Master

The main server of Jenkins is **the Jenkins Master**. It is a web dashboard which is nothing but powered from a **WAR** (Web Application Resource) file. By default it runs on **8080** port. With the help of Dashboard, we can configure the jobs/projects but the **build** takes place in **Nodes/Slave**. By default **one node (slave)** is configured and running in **Jenkins server**. We can **add more nodes** using IP address, user name and password using the **ssh, jnlp or web start** methods.

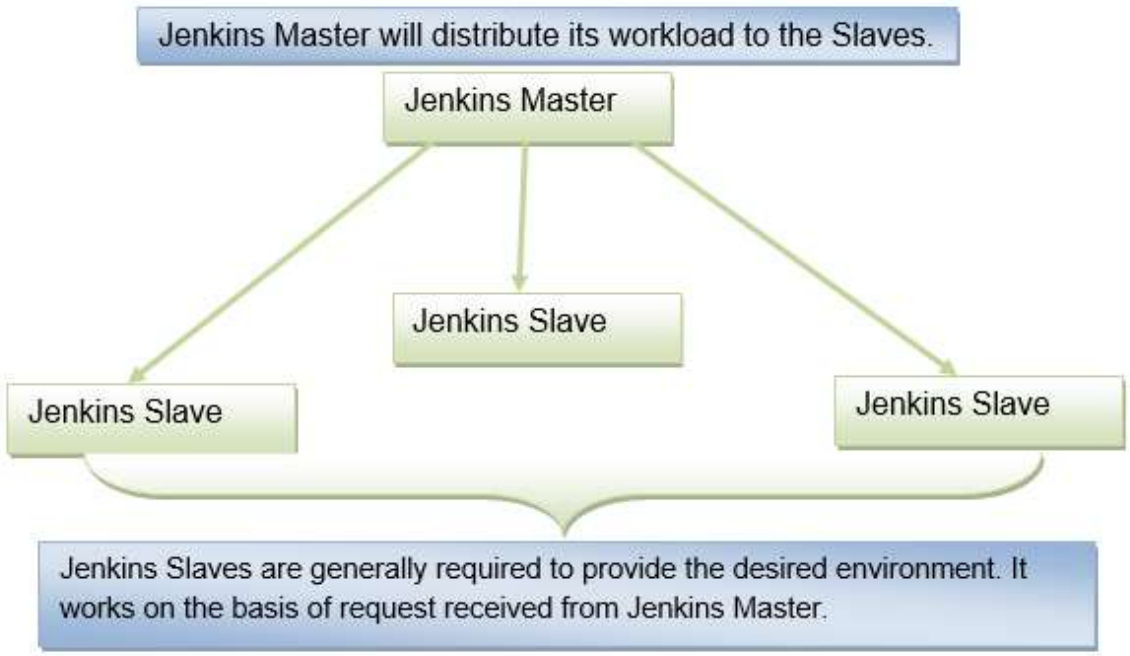
The server's job or master's job is to handle:

- **Scheduling build jobs**.
- **Dispatching builds** to the nodes/slaves for the actual execution.
- **Monitor the nodes/slaves** (possibly taking them online and offline as required).
- **Recording and presenting** the build results.
- A Master/Server instance of Jenkins can also **execute build jobs** directly.

Jenkins Slave

Jenkins slave is used to **execute the build jobs** dispatched by the master. We can configure a project to always **run on a particular slave machine**, or particular type of **slave machine**, or simple let the Jenkins to pick the next available slave/node.

As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nodes can be configured in any servers including Linux, Windows, and Mac.



The above diagram is self-explanatory. It consists of a Jenkins Master which is managing three Jenkins Slaves.

Prerequisite

Before learning Jenkins, you should have a basic understanding of testing and Java.

Audience

Our Jenkins Tutorial is designed to help beginners and professionals.

Problems

We assure that you will not find any problem with this Jenkins Tutorial. But if there is any mistake, please post the problem in the contact form.