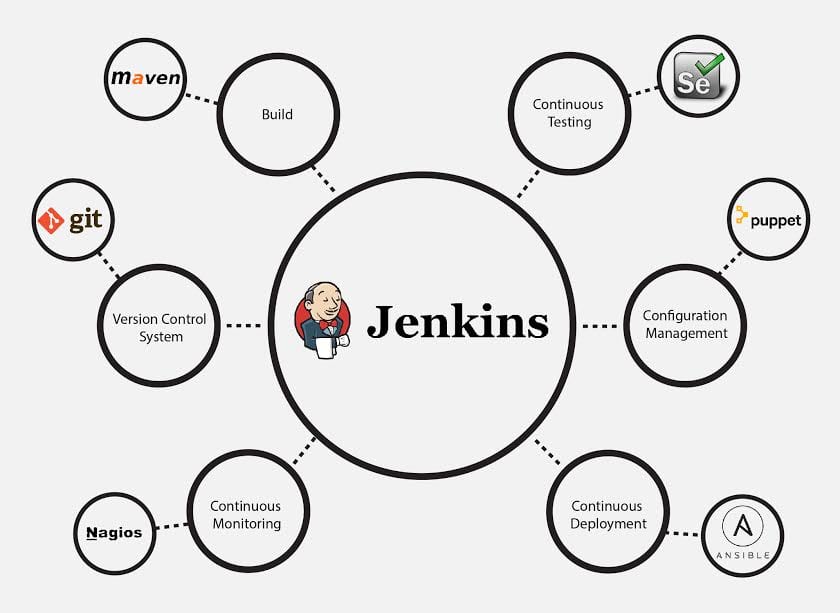
Jenkins

What is Jenkins?

* Continuous Integration is the most important part of DevOps that is used to integrate various [*DevOps stages*](https://www.edureka.co/devops).
* Jenkins is the most famous Continuous Integration tool.
* Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes.
* Jenkins is used to build and test your software projects 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 you to continuously deliver your software by integrating with a large number of testing and deployment technologies.
* With Jenkins, organizations can accelerate the software development process through automation.
* Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis, and much more.
* Jenkins achieves Continuous Integration with the help of plugins. Plugins allows the integration of Various DevOps stages.
* If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven 2 project, Amazon EC2, HTML publisher etc.
* The image below depicts that Jenkins is integrating various DevOps stages:



### **Advantages of Jenkins include:**

* It is an open-source tool with great community support.
* It is easy to install.
* It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share it with the community.
* It is free of cost.
* It is built with Java and hence, it is portable to all the major platforms.

**There are certain things about Jenkins that separates it from other the Continuous Integration tool. Let us take a look on those points.**

## Jenkins Features

Following are some facts about Jenkins that makes it better than other Continuous Integration tools:

* **Adoption:** Jenkins is widespread, with more than 147,000 active installations and over 1 million users around the world.
* **Plugins:** Jenkins is interconnected with well over 1,000 plugins that allow it to integrate with most of the development, testing and deployment tools.

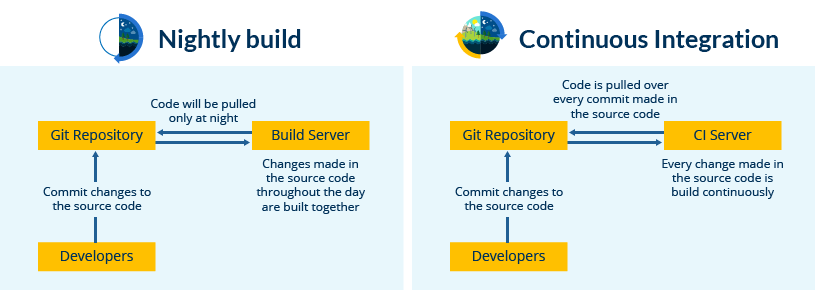
## What is Continuous Integration?

* Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently.
* Every commit made in the repository is then built. This allows the teams to detect the problems early.
* Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results, etc.

**Let us understand its importance with a use-case**

## Continuous Integration Example: Nokia

* I am pretty sure you all have used **Nokia** phones at some point in your life. In a software product development project at Nokia, there was a process called **Nightly builds**.
* Nightly builds can be thought of as a predecessor to Continuous Integration. It means that every night an automated system pulls the code added to the shared repository throughout the day and builds that code.
* The idea is quite similar to Continuous Integration, but since the code that was built at night was quite large, locating and fixing of bugs was a real pain. Due to this, Nokia adopted Continuous Integration (CI).
* As a result, every commit made to the source code in the repository was built. If the build result shows that there is a bug in the code, then the developers only need to check that particular commit. This significantly reduced the time required to release new software.

****

Now is the correct time to understand how Jenkins achieves Continuous Integration.

## Continuous Integration With Jenkins

Let us imagine 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 flaws. I will try to explain them one by one:

* Developers have to wait until the complete software is developed for the test results.
* There is a high possibility 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 coding or architectural 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 risk of frequent failure.

It is evident 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 chaos there was a dire 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 CI is all about. Jenkins is the most mature CI tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.

I will first explain you a generic flow diagram of Continuous Integration with Jenkins so that it becomes self-explanatory, how Jenkins overcomes the above shortcomings:



The above diagram is depicting the following functions:

* First, a developer commits the code to the source code repository. Meanwhile, the Jenkins server checks the repository at regular intervals for changes.
* Soon after a commit occurs, the Jenkins server detects the changes that have occurred in the source code repository. Jenkins will pull 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 deploys the built in the test server.
* After testing, Jenkins generates a feedback and then notifies the developers about the build and test results.
* It will continue to check the source code repository for changes made in the source code and the whole process keeps on repeating.

## Jenkins Architecture

This single Jenkins server was not enough to meet certain requirements like:

* Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
* If larger and heavier projects get built on a regular basis then a single Jenkins server cannot simply handle the entire load.

To address the above stated needs, Jenkins distributed architecture was introduced.

## Jenkins Distributed Architecture

Jenkins uses a **Master-Slave** architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

**Jenkins Master**

Your main Jenkins server is the Master. The Master’s job is to handle:

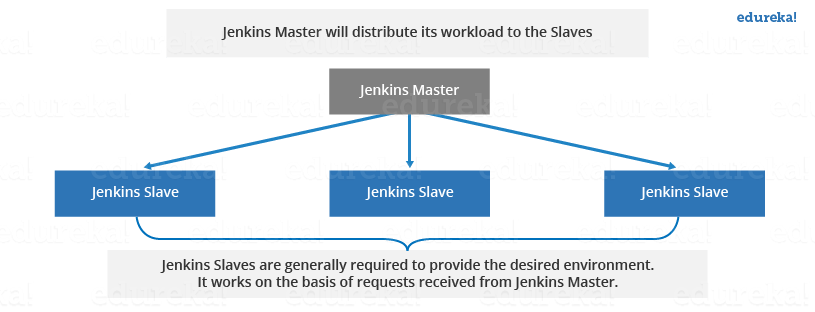
* Scheduling build jobs.
* Dispatching builds to the slaves for the actual execution.
* Monitor the slaves (possibly taking them online and offline as required).
* Recording and presenting the build results.
* A Master instance of Jenkins can also execute build jobs directly.

**Jenkins Slave**

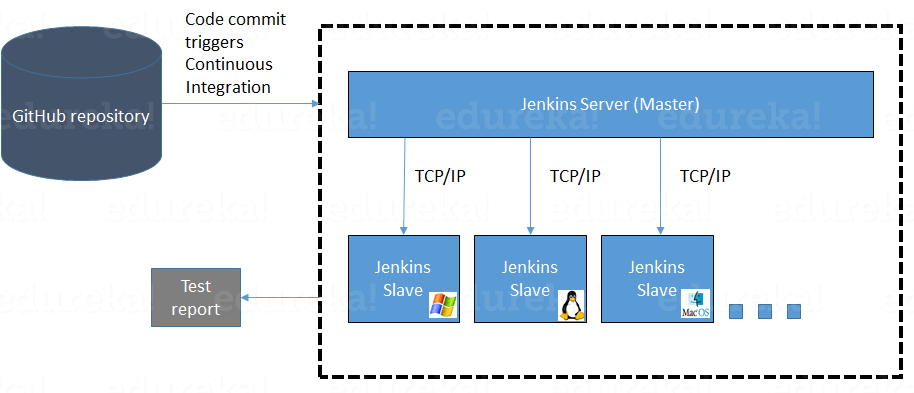
A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

* It hears requests from the Jenkins Master instance.
* Slaves can run on a variety of operating systems.
* The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
* You can configure a project to always run on a particular Slave machine, or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

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



Now let us look at an example in which Jenkins is used for testing in different environments like: Ubuntu, MAC, Windows etc.



The following functions are performed in the above image:

* Jenkins checks the Git repository at periodic intervals for any changes made in the source code.
* Each builds requires a different testing environment which is not possible for a single Jenkins server. In order to perform testing in different environments Jenkins uses various Slaves as shown in the diagram.
* Jenkins Master requests these Slaves to perform testing and to generate test reports.