## 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

1. **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.

AD

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.

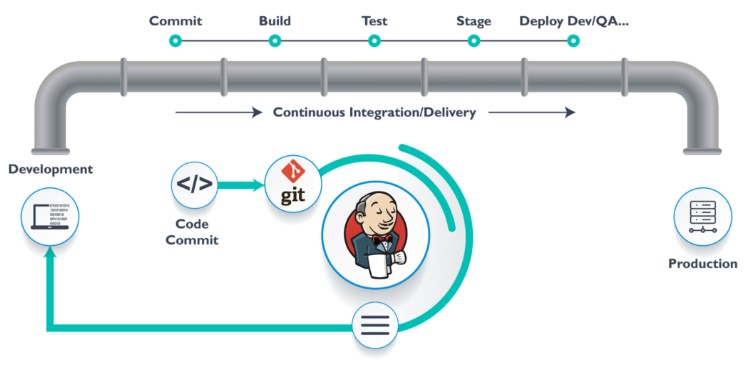
So to overcome such problem there was a need for a system to exist where developers can continuously trigger a build and test for every change made in the source code.

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:

AD





**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**

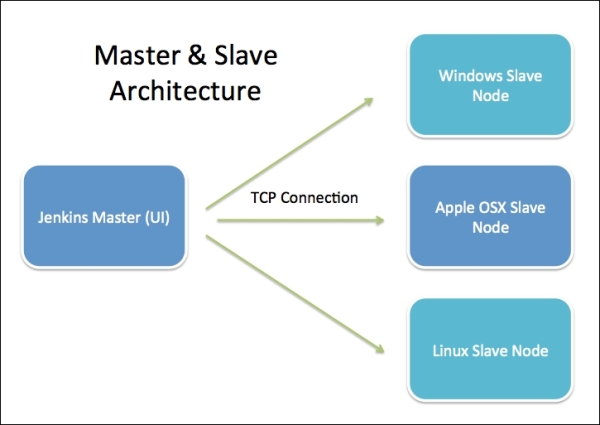
* Its interface is out dated and not user friendly compared to current user interface trends.
* Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.
* CI regularly breaks due to some small setting changes. CI will be paused and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

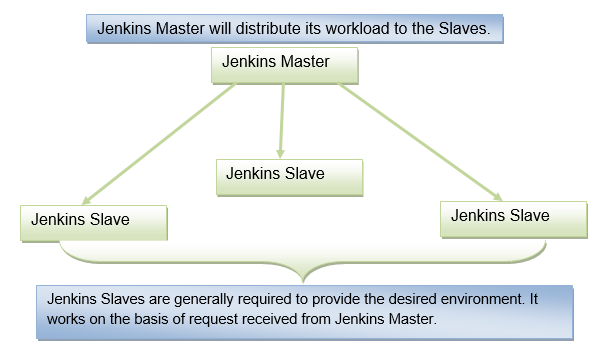
Main Jenkins server is the Master and its job is to handle:

* Scheduling build jobs
* Dispatching builds to the slaves for the actual execution
* Monitoring the slaves (possibly taking them online and offline as required)
* Recording and presenting the build results
* Executing build jobs directly

**Jenkins Slave**

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

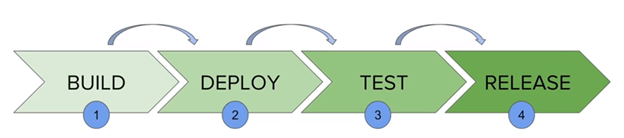
* It hears requests from the Jenkins Master instance.
* A Slave can run on a variety of operating systems.
* The job of a Slave is to “obey” the Master, that is, execute build jobs dispatched by it.
* You can either configure a project to always run on a particular (type of) Slave machine or simply let Jenkins pick the next available Slave.



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

What is Continuous Delivery Pipeline?

In a Jenkins Pipeline, every job has some sort of dependency on at least one or more jobs or events.



The above diagram represents a continuous delivery pipeline in Jenkins. It contains a collection of states such as build, deploy, test and release. These jobs or events are interlinked with each other. Every state has its jobs, which work in a sequence called a continuous delivery pipeline.

A continuous delivery pipeline is an automated expression to show your process for getting software for version control. Thus, every change made in your software goes through a number of complex processes on its manner to being released. It also involves developing the software in a repeatable and reliable manner, and progression of the built software through multiple stages of testing and deployment.

## JenkinsFile

Jenkins Pipeline can be defined by a text file called JenkinsFile. You can implement pipeline as code using JenkinsFile, and this can be defined by using a DSL (Domain Specific Language). With the help of JenkinsFile, you can write the steps required for running a Jenkins Pipeline.

The benefits of using JenkinsFile are:

* You can make pipelines automatically for all branches and can execute pull requests with just one JenkinsFile.
* You can review your code on the pipeline.
* You can review your Jenkins pipeline.
* This is the singular source for your pipeline and can be customized by multiple users.

JenkinsFile can be defined by using either Web UI or with a JenkinsFile.

Pipeline syntax

Two types of syntax are used for defining your JenkinsFile.

* Declarative
* Scripted

AD

**Declarative:**

Declarative pipeline syntax offers a simple way to create pipelines. It consists of a predefined hierarchy to create Jenkins pipelines. It provides you the ability to control all aspects of a pipeline execution in a simple, straightforward manner.

**Scripted:**

Scripted Jenkins pipeline syntax runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to convert the pipeline into atomic commands.

Both scripted and declarative syntax are different from each other and are defined totally differently.

AD

Why Use Jenkins Pipeline?

Jenkins is a continuous integration server which has the ability to support the automation of software development processes. You can create several automation jobs with the help of use cases, and run them as a Jenkins pipeline.

AD

Here are the reasons why you should use Jenkins pipeline:

* Jenkins pipeline is implemented as a code which allows several users to edit and execute the pipeline process.
* Pipelines are robust. So if your server undergoes an unpredicted restart, the pipeline will be automatically resumed.
* You can pause the pipeline process and make it wait to continue until there is an input from the user.
* Jenkins Pipelines support big projects. You can run many jobs, and even use pipelines in a loop.

**Jenkins Pipeline Concepts**

**Pipeline:** 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{

}

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

node{

}

**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:

AD

pipeline {

    agent any

    stages {

            stage ('Build') {

                ...

            }

            stage ('Test') {

                ...

            }

            stage ('QA') {

                ...

            }

            stage ('Deploy') {

                ...

            }

            stage ('Monitor') {

                ...

            }

    }

}

**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.

pipeline {

    agent any

    stages {

            stage ('Build') {

                steps {

                        echo 'Running build phase...'

                }

            }

    }

}

# Jenkins Pipeline

In Jenkins, a pipeline is a collection of events or jobs which are interlinked with one another in a sequence.

It is a combination of plugins that support the integration and implementation of **continuous delivery pipelines** using Jenkins.

In other words, a Jenkins Pipeline is a collection of jobs or events that brings the software from version control into the hands of the end users by using automation tools. It is used to incorporate continuous delivery in our software development workflow.

A pipeline has an extensible automation server for creating simple or even complex delivery pipelines "as code", via DSL (Domain-specific language).

**A sample Scripted pipeline**

node{

stage('Compile'){

steps {

echo "Compiled Successfully!!";

}

}

stage('JUnit') {

steps {

echo "JUnit Passed Succesfully!";

}

}

stage('Quality-Gate') {

steps {

echo "SonarQube Quality Gate passed succesfully!!";

/\*sh exit ("1");\*/

}

}

stage('Deploy') {

steps {

echo "Pass!";

}

}

}

**A sample Groovy script Declarative pipeline**

pipeline {

agent any

stages {

stage('Git-Checkout'){

steps {

echo "Checking out from Git Repo";

}

}

stage('Build') {

steps {

echo "Building the checkedout project!";

}

}

stage('Unit-Test') {

steps {

echo "Running JUnit tests";

}

}

stage('Quality-Gate') {

steps {

echo "Verifying Quality Gates";

}

}

stage('Deploy') {

steps {

echo "Deploying to stage environments for more tests";

}

}

}

post {

always {

echo 'This will always run'

}

success {

echo 'This will run only if successful'

}

failure {

echo 'This will run only if failed'

}

unstable {

echo 'This will run only if the run was marked as unstable'

}

changed {

echo 'This will run only if the state of the pipeline has changed'

echo 'For example, if the pipeline was previously failing but is now successful'

}

}

}

**A MASTER-SLAVE Groovy script Declarative Pipeline**

pipeline {

agent none

stages {

stage('Non-parallel Stage'){

agent {

label "built-in";

}

steps {

echo 'This stage will be executed first';

}

}

stage('Run Tests') {

parallel {

stage('Test on Windows') {

agent {

label "slave\_node1";

}

steps {

echo "Task1 on Agent";

}

}

stage('Test on Master') {

agent {

label "built-in";

}

steps {

echo "Task1 on Master";

}

}

}

}

}

}