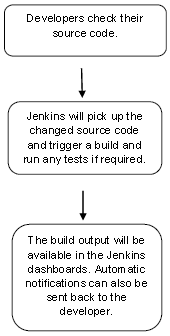
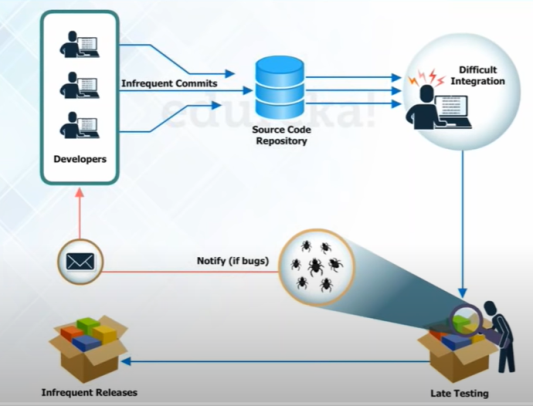


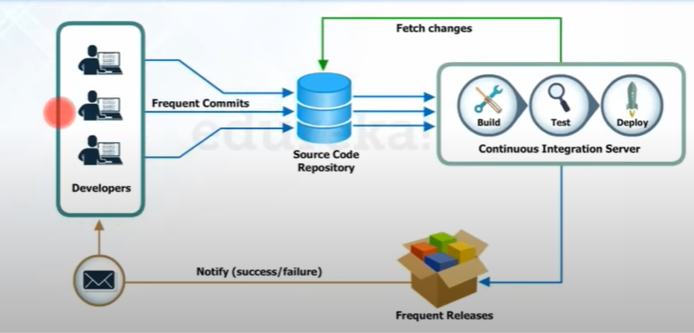
**JENKINS**

Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place. The following flowchart demonstrates a very simple workflow of how Jenkins works.



Along with Jenkins, sometimes, one might also see the association of **Hudson**. Hudson is a very popular open-source Java-based continuous integration tool developed by Sun Microsystems which was later acquired by Oracle. After the acquisition of Sun by Oracle, a fork was created from the Hudson source code, which brought about the introduction of Jenkins.

What is Continuous Integration?



Continuous Integration is a development practice that requires developers to integrate code into a shared repository at regular intervals. This concept was meant to remove the problem of finding later occurrence of issues in the build lifecycle. Continuous integration requires the developers to have frequent builds. The common practice is that whenever a code commit occurs, a build should be triggered.

**Jenkins Architecture**

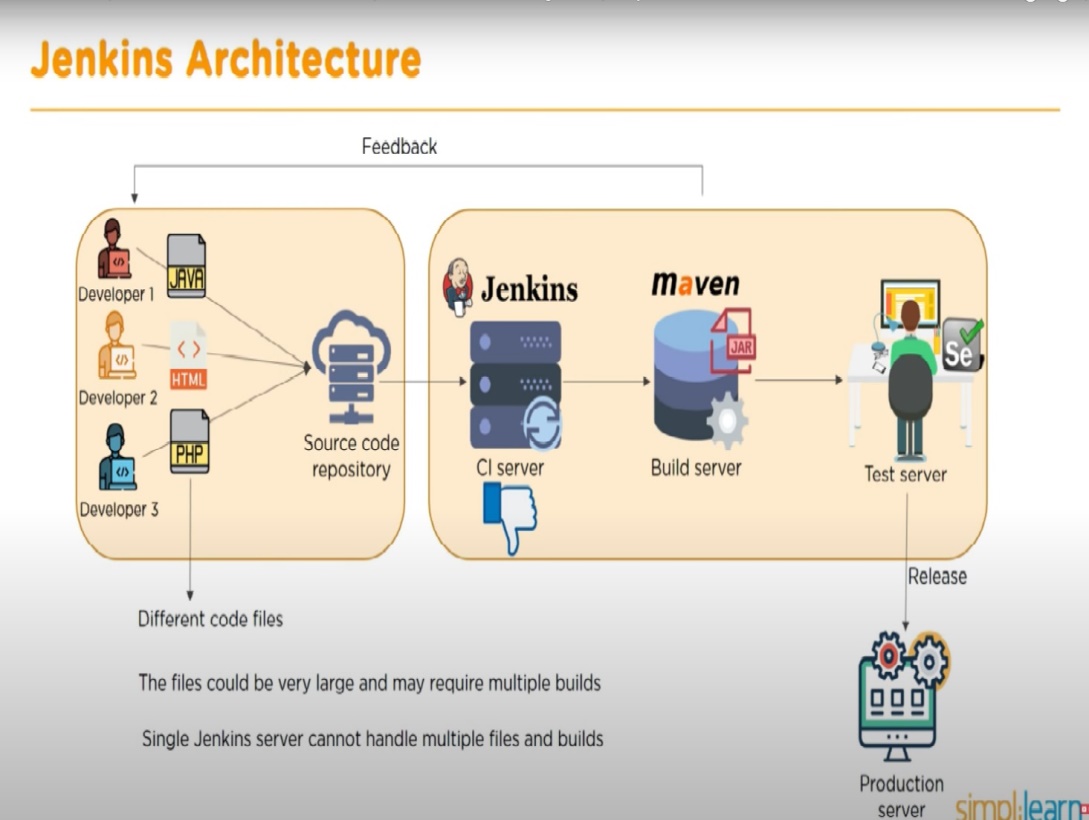
* Developers commit changes to the source code, found in the repository.
* The Jenkins CI server checks the repository at regular intervals and pulls any newly available code.
* The Build Server builds the code into an executable file. In case the build fails, feedback is sent to the developers.
* Jenkins deploys the build application to the test server. If the test fails, the developers are alerted.
* If the code is error-free, the tested application is deployed on the production server.

The files can contain different code and be very large, requiring multiple builds. However, a single Jenkins server cannot handle multiple files and builds simultaneously; for that, a distributed Jenkins architecture is necessary.

Mainly the executable files are very large in size, and it is not possible to handle multiple & large executable files by a single Jenkins server, so Jenkins uses Master-Slave Architecture to perform continuous integration and continuous deployment.

A single Jenkins server can also not perform testing in different platforms like on windows, Linux and Mac operating systems. To fulfil this requirement, we need to use Jenkins slaves. You can use many Jenkins slaves under a single Jenkins server, which is known as Jenkins Master.

Jenkins architecture is given below, but it is the architecture of single Jenkins server.



**Salient features of Jenkins**

1. Easy Installation & Configuration

Jenkins is a self-contained Java program that is platform independent . 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.

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.

* As explained earlier Jenkins is an open source automation server written in java. It is used to automate software development process via continuous integration and facilitates continuous delivery.

**Some Use-cases before and after continuous integration with Jenkins:**

1. **Nightly builds:**

* It means that every night an automated system pulls the code added to the shared repository throughout the day and builds that code.
* In a software product development project at Nokia, this Nightly builds technique was used.
* Nightly builds can be thought of as a predecessor to Continuous Integration.
* But the problem with the Nightly builds is that if the any Bugs or errors were present in the very next day the developer has to go through all the commits he made to source code in the previous day.
* The above process is seems to be difficult as the developer should find the bugs and fixes it.
* To overcome this problem, 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.

1. **Continuous Integration with Jenkins:**

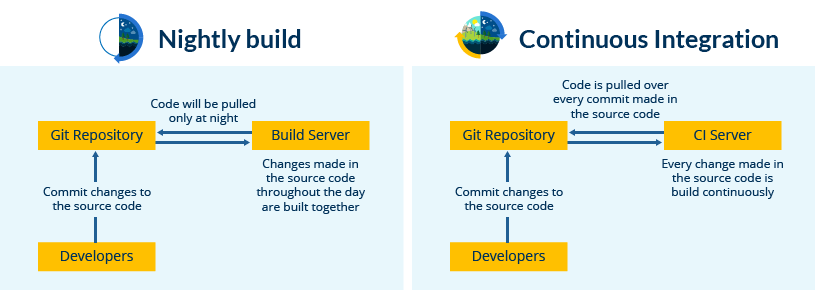


* The above diagram shows the continuous integration with Jenkins

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.

Now we can compare the software development process with night builds and with Jenkins:



**Before Jenkins:**

* The entire source code was built and then tested. Locating and fixing bugs in the event of build and test failure was difficult and time-consuming, which in turn slows the software delivery process.
* Developers have to wait for test results.
* The whole process is manual.

**After Jenkins:**

* Every commit made in the source code is built and tested. So, instead of checking the entire source code developers only need to focus on a particular commit. This leads to frequent new software releases.
* Developers know the test result of every commit made in the source code on the run.
* You only need to commit changes to the source code and Jenkins will automate the rest of the process for you.

**Why use Jenkins**

* It is by far the most widely used tool for managing continuous integration builds and delivery pipelines.
* It helps developers in building and testing software continuously.
* It increases the scale of automation and is quickly gaining popularity in DevOps circles.
* One of the key advantages of Jenkins is that it requires little maintenance and has built-in GUI tool for easy updates.
* Jenkins also provides customized solution as there are over 400 plugins to support building and testing virtually any project.
* Basically, Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis and much more.
* 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.