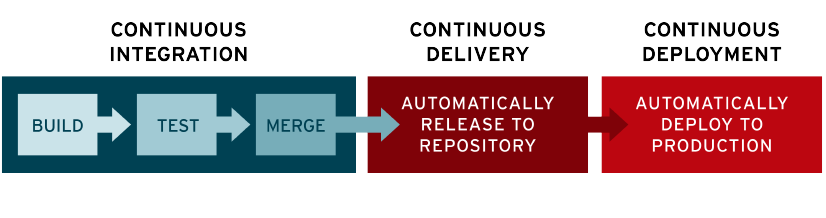
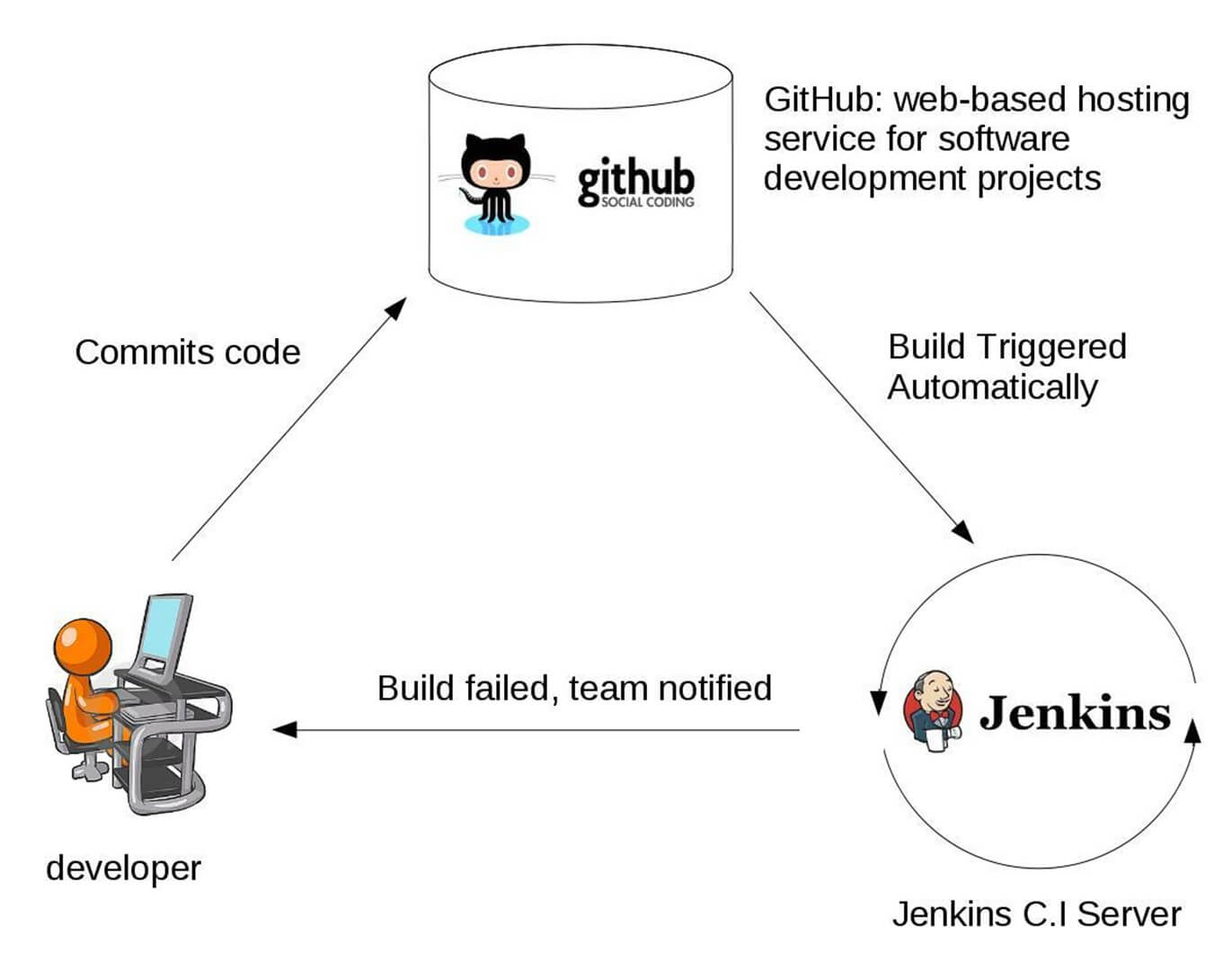
**Jenkins** Jenkins is a continuous integration tool that perform packaging. It takes input from GIT(VCS)





**Jenkin installation steps**

Jenkin is written in java language, so we need to install java as a prework for sudo apt-get install default-jre

wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt-get update

sudo apt-get install jenkins

**Jenkin installation by Docker**

# Install docker by docker shell script

curl -fsSL https://get.docker.com -o get-docker.sh sh get-docker.sh

# Fetch jenkin image from dockerhub docker run --name myjenkins -d -p 8080:8080 -p 50000:50000 -v /var/jenkins\_home jenkins

# Run the URL: URL- <http://host:8080>

**Casestudy**

We want to execute a job relatively how we can do? Answer: cronjob (or) Scheduler. These kinds of scheduling taken into next level by Jenkins.

Click on new project-> freestyle-> post build command as: whomai then build project

**Sample console output**



Continuous integration life cycle:



It takes GIT as input, then it complier project and perform unit testing. After that depending on input it builds project.

Task git clone cd <> git checkout -b <> git push origin

Note: We can integrate Jenkins with Ansible, Docker and other tools like GIT. Even We can integrate with AD for authentication purpose.

CI workflow having issue: 1) software not having permission 2) mess up with working directory

**Other CI tools**: Bamboo.

**what is compilation?** Generating from .java to .jar

**Packaging?** Adding html, and other supporting files into jar file, and create EAR file.

**Compiling java project** mvm compile mvn test mvn package

**What CI Engineer do?** Developer put code into GIT the, 1) check if build is happening or not? 2) execute testing 3) then build then send feedback to development team.

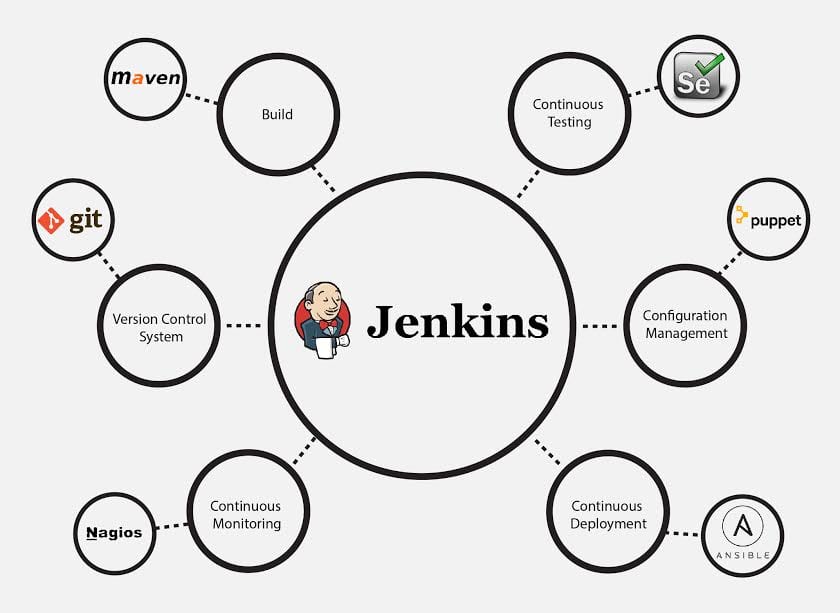
## What is Jenkins?

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. 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 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.

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.

It is evident from the above points that Jenkins has a very high demand globally. Before we dive into Jenkins it is important to know what Continuous Integration is and why it was introduced.

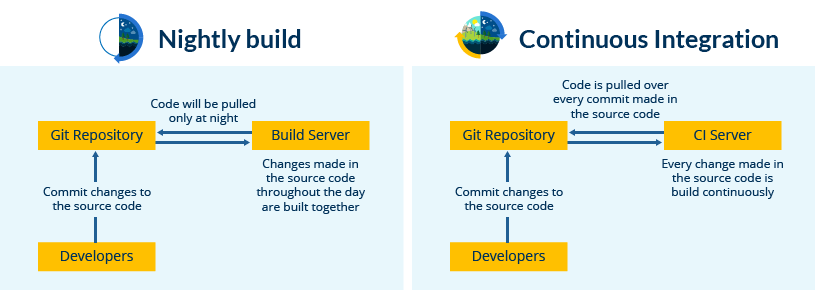
## What is Continuous Integration?

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 in 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.



### DevOps Training

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 a software, but, this process has many flaws. I will try to explain them one by one:

* Developers have to wait till 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 a 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.



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.

You now know how Jenkins overcomes the traditional SDLC shortcomings. The table below shows the comparison between “Before and After Jenkins”.

|  |  |
| --- | --- |
| **Before Jenkins** | **After 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. | 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 have to wait for test results | Developers know the test result of every commit made in the source code on the run. |
| The whole process is manual | You only need to commit changes to the source code and Jenkins will automate the rest of the process for you. |