**Jenkins**

**What is Jenkins:**

**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**

Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration (CI), Continuous delivery (CD) and continue deployment purpose in software development.

**Work Flow:**



**What is Continuous Integration (CI), Continuous delivery (CD) and continue deployment:**

It is a software development practices in which we will automatically build, test and deploy

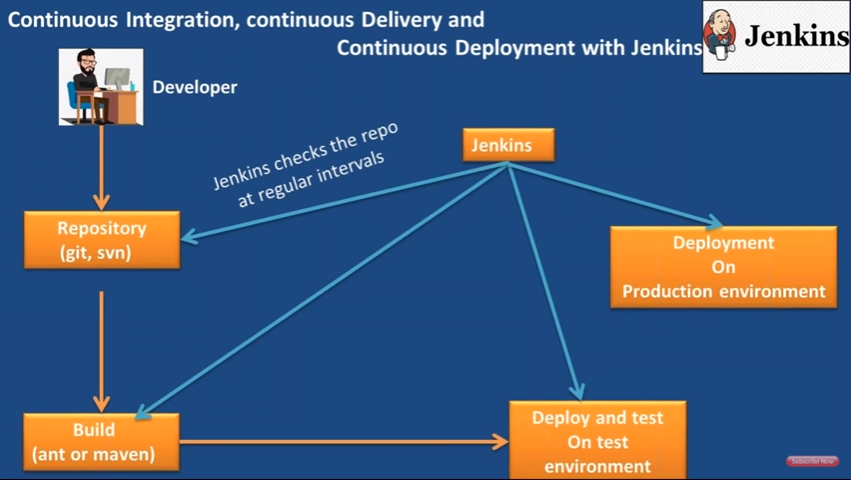
test/production environment.

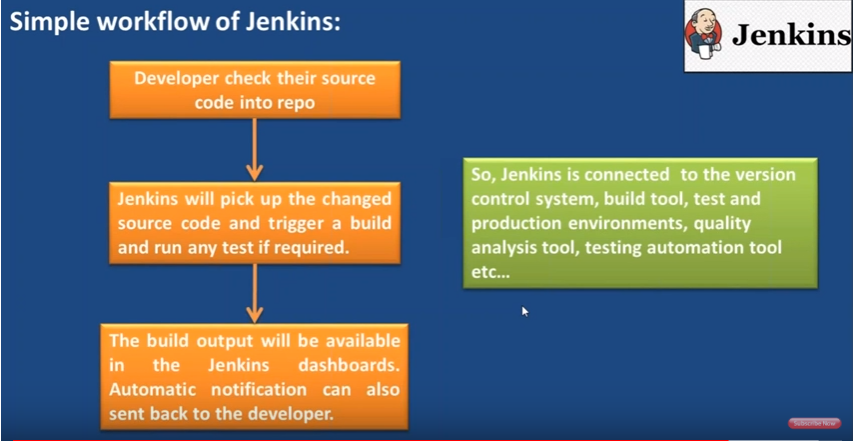
Software projects every time whenever a developer pushes code change to the repository.

In Continuous Integration after a code commit, the software is built and tested immediately. In a large project with many developers, commits are made many times during a day. With each commit code is built and tested. If the test is passed, build is tested for deployment. If deployment is a success, the code is pushed to production. This commit, build, test, and deploy is a continuous process and hence the name continuous integration/deployment.

A Continuous Integration Pipeline is a powerful instrument that consists of a set of tools designed to **host**, **monitor**, **compile** and **test** code, or code changes, like:

* Continuous Integration Server (Jenkins, Bamboo, CruiseControl, TeamCity, and others)
* Source Control Tool (e.g., CVS, SVN, GIT, Mercurial, Perforce, ClearCase and others)
* Build tool (Make, ANT, Maven, Ivy, Gradle, and others)
* Automation testing framework (Selenium, Appium, TestComplete, UFT, and others)





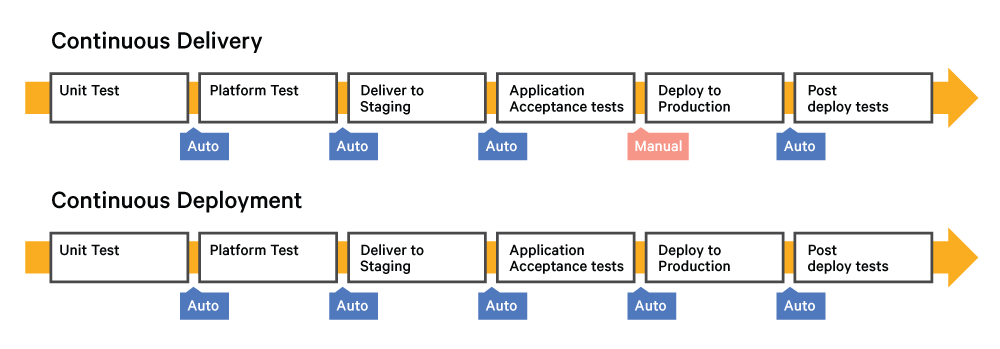
|  |  |
| --- | --- |
| **Before Jenkins** | **After Jenkins** |
| Once all Developers had completed their assigned coding tasks, they used to commit their code all at same time. Later, Build is tested and deployed.  Code commit built, and test cycle was very infrequent, and a single build was done after many days. | The code is built and test as soon as Developer commits code. Jenkin will build and test code many times during the day  If the build is successful, then Jenkins will deploy the source into the test server and notifies the deployment team.  If the build fails, then Jenkins will notify the errors to the developer team. |
| Since the code was built all at once, some developers would need to wait until other developers finish coding to check their build | The code is built immediately after any of the Developer commits. |
| It is not an easy task to isolate, detect, and fix errors for multiple commits. | Since the code is built after each commit of a single developer, it's easy to detect whose code caused the built to fail |
| Code build and test process are entirely manual, so there are a lot of chances for failure. | Automated build and test process saving timing and reducing defects. |
| The code is deployed once all the errors are fixed and tested. | The code is deployed after every successful build and test. |
| Development Cycle is slow | The development cycle is fast. New features are more readily available to users. Increases profits. |

Continuous Deployment: continues deployments is completely automate process write from source code check in till deployment to the production.

Continuous Delivery: continues deployments is a automate process expact deploying the application to the production based on its approval.

**Delivery vs. Deployment**

In this graphic, you can see the point at which the difference between Continuous Delivery and Continuous Deployment exists.



With Continuous Delivery, "Deploy to Production" is a manual process, meaning that it is initiated manually. This differs from Continuous Deployment, which is automated all the way through "Post Deployment Test."

In summary, Continuous Delivery is a state of being ready and able to release any version at any time on any platform, whereas Continuous Deployment is being able to continually deploy

The focus of Continuous Delivery must be on Efficiency of the delivery process and Quality of working Software. And the focus of Continues Deployment must be on the frequency and speed of the deployment of working software on production.

Hence using Continuous Delivery , we can focus on building quality working software and making the delivery process more repeatable with maximum automation and using the Continuous Deployment, we can focus on increasing the frequencies to release on to the production based on the business needs

continuous delivery means that you product is always ready to be deployable to production systems , it is increasing the confidence that your software is production ready. Continous deployment is the same as described above , fully automated pipeline from start to deployment. So one thing you can think of here is the pull vs push model. In Continuous deployment you keep on pushing to production continously , in continuos delivery , it is production ready that can be pulled to production systems any time

What’s the Difference Between Continuous Delivery vs. Continuous Deployment

Every DevOps user today is aware of CI/CD as a concept. CI stands for continuous integration, whereas CD is often used interchangeably to signify “continuous delivery” and “continuous deployment.”

**Continuous integration** involves a series of steps that are automatically performed to integrate code from multiple sources, create a build and test. Each time a build or a set of code passes the tests, it’s automatically deployed out to a staging environment where further testing such as load testing and manual exploratory testing is conducted. This process can be repeated for days depending upon the project delivery requirements.

**Continuous delivery** helps you build a refined version of the software by continuously implementing fixes and feedback until finally, you decide to push it out to production. In other words, continuous delivery involves human decision-making around what to release to the customers, and when. This forms the basis of the difference between the two.

In **continuous deployment**, every change goes through an automated pipeline and a working version of the application is [automatically pushed to production](https://blog.codeship.com/seven-steps-to-continuous-deployment/). It does not involve any release approval cycle and the project teams need to ensure that every time a code is updated, tested and released, it works smoothly at the customer’s end. Naturally, a lot depends on the quality of the test suite. With continuous deployment, teams can have multiple software deployments on any given day and don’t have to sweat about a major release

**Jenkins steps:**

**Declarative Pipeline fundamentals**

In Declarative Pipeline syntax, the pipeline block defines all the work done throughout your entire Pipeline.

Jenkinsfile (Declarative Pipeline)

pipeline {

agent any

stages {

stage('Build') {

steps {

//

}

}

stage('Test') {

steps {

//

}

}

stage('Deploy') {

steps {

//

}

}

}

}

Execute this Pipeline or any of its stages, on any available agent.

Defines the "Build" stage.

Perform some steps related to the "Build" stage.

Defines the "Test" stage.

Perform some steps related to the "Test" stage.

Defines the "Deploy" stage.

Perform some steps related to the "Deploy" stage.

|  |
| --- |
| script type:  pipeline {  agent any  stages {  stage('clone repo and clean it') {  steps {  sh "rm -r SwaggerWithRestApi"  sh "git clone https://github.com/NagendraMekala/SwaggerWithRestApi.git"  sh "mvn clean -f SwaggerWithRestApi"  }  }  stage('Test') {  steps {  sh "mvn test -f SwaggerWithRestApi"  }  }  stage('Deploy') {  steps {  sh "mvn package -f SwaggerWithRestApi"  }  }  }  }  Using git scm:  pipeline {  agent any  stages {  stage('---clean it---') {  steps {  sh "mvn clean"  }  }  stage('---test---') {  steps {  sh "mvn test"  }  }  stage('---deploy---') {  steps {  sh "mvn package"  }  }  }  } |

**Unix-based Jenkins commands**

You installed Jenkins on a Debian-based or a Fedora-based distribution, you can use the following commands:

$ sudo service jenkins restart

$ sudo service jenkins stop

$ sudo service jenkins start

Or in the latest distribution of Linux:

$ sudo systemctl start jenkins.service

$ sudo systemctl stop jenkins.service

$ sudo systemctl restart jenkins.service

**Windows**

You installed Jenkins as a service on Windows, you can either use the UI component Services manager (by running services.msc) or you can use the following command:

$ C:\Program Files (x86)\Jenkins>jenkins.exe start

$ C:\Program Files (x86)\Jenkins>jenkins.exe stop

$ C:\Program Files (x86)\Jenkins>jenkins.exe restart