D — DEVOPS

Understanding Continuous Integration, Delivery and Deployment

by **devopscube** · November 3, 2016

Continuous Integration (CI), ***Continuous Delivery*** and ***Continuous Deployment*** (CD) have become a part of the daily life for quite some time now for the IT personnel who practice DevOps. When it comes to DevOps discussion, everything "continuous" has become a part of it. However, there are still many companies out there who do not use the CI/CD pipeline for the development of their applications and services. They either lack the understanding of these processes or they simply lack the support of their employees to apply the CI/CD structure.

Implementing the **continuous integration** into the services and applications without the proper knowledge of **eXtreme Programming** (XP) was not possible in the early stages of the development of CI but now several other methodologies have been developed out of which the automated testing is the most frequently used one. For

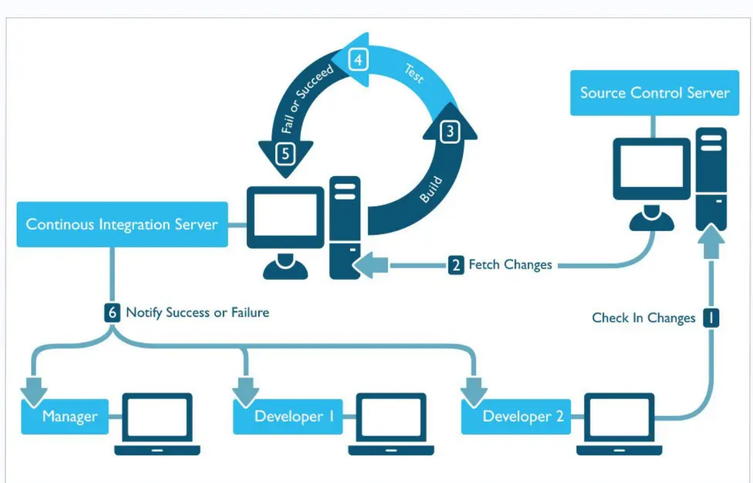
the production was even easier as every build can be deployed down the pipeline automatically without waiting for the confirmation from anyone.

Thus, the best way to utilize this three processes is to implement them in synergy for the better development of your applications and services. But to do so, you need to understand each concept first because without having the knowledge about the processes it will not be possible to implement them together.

What is Continuous Integration (CI)?

In the early stages of the development of the Continuous Integration model, it was not possible to use it without eXtreme Programming (XP). But since then a lot of things have changed and the integration process has become mostly automated.

Continuous Integration is an automated process. It is done in order to integrate various codes from different potential sources in order to build it or test it. The codes are often required to be integrated into the form of a shared repository by the developers.



Application code can be integrated into the repository either by pushing them into it or by merging the code with the repository. No matter which type of integration process you choose, you need to have a pipeline to check and test the codes integrated with the repository.

The result of these tests pipeline can either be ***red*** or ***green,*** where red corresponds to the failure of something and green signal implies that the test was successful.

The tests that are performed on the pipeline are mostly automated and there are several types of automated tests that can be performed on the integrated codes on

These tests are automatically performed in the pipeline whenever the codes are pushed or merged into the repository. That is why the tests should be written in a test driven development fashion. The automation and connection with the pipeline make the continuous integration process possible.

Choosing the perfect CI tool can be a little tricky. There are several things that are to be taken into account before choosing a continuous integration tool. Some CI tools allow you to run the application on your server by testing it in your own data center (eg: Jenkins, Bamboo etc) whereas there are several hosted CI services (Travis CI, Circle CI) also available that provides a multitude of benefits.

For example- most of the hosted continuous integration tool allow you to run the tests in single use **containers** thereby guaranteeing fresh as well as a consistent environment for every test in the pipeline. They also provide relief to the small teams by relieving them from maintenance duties which can become huge burdens.

Each automated testing has their own implications very different from one another. For example- The Unit tests mostly need a database to perform. Thus, the continuous integration environment must have to be able to provide a database for the **unit tests** to work with maximum efficacy.

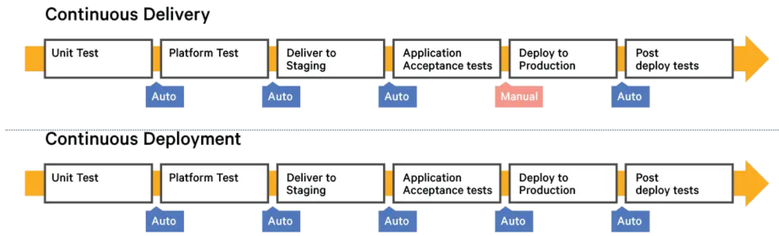
Similarly, for the **service tests** or the API tests, a server is required to return the API calls and the CI service must be efficient enough to be able to provide that. The tests may also need to install some additional service packages so choosing the correct CI service which will allow you to do that is very important. Same goes for the **functional tests** too.

You can also choose to increase the efficiency of your continuous integration pipeline by performing parallel tests. It will help to speed the processes in the pipeline by several times.

What is Continuous Delivery and Continuous Deployment?

Most of the people think that ***continuous delivery*** and ***continuous deployment*** is the same thing. But that is not true, they have some subtle differences. The pipeline of continuous delivery (CD) is almost similar to that of the CI pipeline.

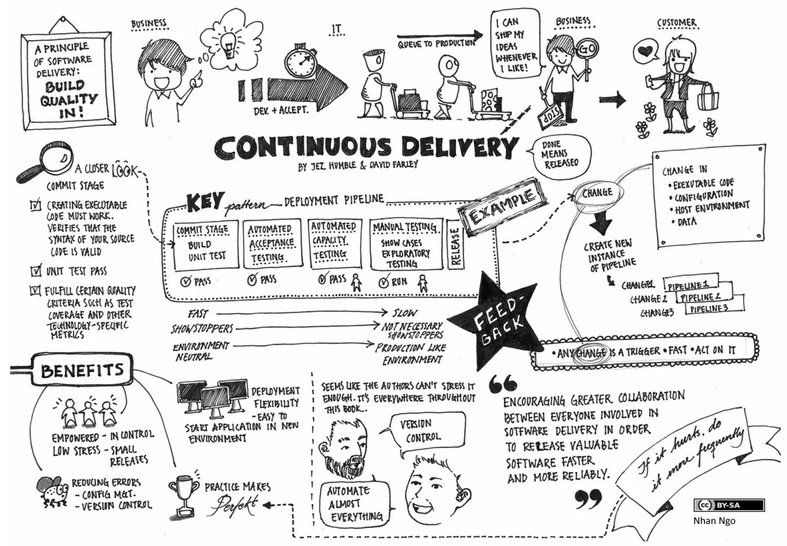
The major difference between the CI and the CD is that the former assumes that there are several actions to be taken post-deployment whereas the CD pipeline always makes the products and artifacts ready for the production. Simply put every



Another difference of continuous integration from continuous delivery is that the CD does not have any manual testing phase, unlike CI after the promotion of the repository to the pipeline. The pipeline is fully automated thus no manual actions are required.

The continuous deployment process goes one step further than the integration and delivery process. The pipeline of continuous deployment automatically deploys each and every build that have been verified. It is a totally automated process which has a commitment towards coding repository thus the services or applications get deployed to production automatically.

There is no need of any human intervention in the pipeline or any related process in the continuous deployment pipeline. Only if the build is being sent to the QA then you can server then you can decide to run the post-deployment tests in different subsets.



The continuous integration does not always require running the tests during production although it has provisions for it but for the continuous delivery and continuous deployment the production tests are absolutely mandatory due to the

features temporarily from functioning in the production code.

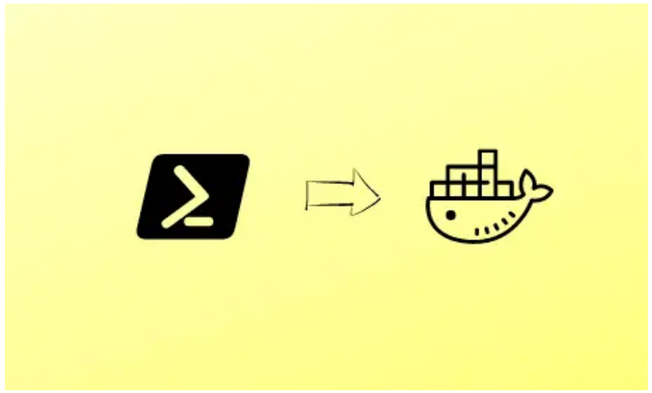
Conclusion

Using all three scenarios will make the application development and deployment very smooth. Most of the companies start working with the continuous integration and in later stages shift to the continuous deployment or delivery, as the CI prepares the base for the CDs to perform efficiently. Using the continuous integration, continuous delivery, and continuous deployment altogether reduces the risks by increasing confidence.

devopscube

Established in 2014, a community for developers and system admins. Our goal is to continue to build a growing DevOps community offering the best in-depth articles, interviews, event listings, whitepapers, infographics and much more on DevOps.

YOU MAY ALSO LIKE



D — DEVOPS

Running Custom Scripts In Docker With Arguments – ENTRYPOINT Vs CMD

by devopscube · April 18, 2021

Use Case: You need to run a custom shell script on your Docker container with arguments passed to...

C — COMMON

Docker Machine Tutorial : Getting Started Guide

by devopscube · August 22, 2015

D — DEVOPS

How to Use Parameters in Jenkins Declarative Pipeline

by **Bibin Wilson** · August 12, 2020

In Jenkins's declarative pipeline, you can add parameters as part of Jenkinsfile. There are many supported parameters type...

D — DEVOPS

Install Jenkins on Ubuntu in 10 Easy Steps

by **devopscube** · March 3, 2021

Jenkins 2.x has lots of great functionalities that will make the CI pipeline smooth using the pipeline as...

D — DEVOPS

List of the best programming languages to learn in 2017

by **devopscube** · December 2, 2016

Programming Languages... AH! There are so many!! And it is really confusing to figure out which language you...

D — DEVOPS

Jenkins Multibranch Pipeline Tutorial For Beginners

Jenkins Continuous Integration & Delivery (CI/CD)...