CONTINUOUS DELIVERY VS DEPLOYMENT VS INTEGRATION: WHAT'S THE DIFFERENCE?

As DevOps solidifies its position in the software world, a new buzzword is capturing our attention. "Continuous" is currently one of the most popular words in the DevOps lexicon, and it is used widely by developers and companies alike.

Although the word has become widespread, what exactly does it mean? When it is used in concepts such as continuous delivery, continuous deployment and continuous integration, how does its meaning change? And what exactly are the differences between the three? We have put together complete and comprehensive definitions of continuous delivery, continuous deployment, and continuous integration, as well as how businesses, developers, and project managers can best utilize them in one integrated environment.

What is continuous?

Before we go into any details about the various DevOps concepts, it is important to begin with the explanation of what continuous means in the software field. Simply put, continuous refers to the software changes that roll down the pipeline in a never-stopping and completely constant fashion.

Of course, this is an exaggeration by DevOps folks as delivery, deployment, and integration are almost never 100 percent continuous. In fact, a continuously integrated application is probably going to be rebuilt and delivered closer to every 24 hours instead of every single time a code change occurs. While this speed may not be as instantaneous as developers would like to admit, it is still

much faster than the delivery pipelines that were common before the days of DevOps.

What is continuous delivery?

In most cases, continuous delivery is a series of practices designed to deliver software updates to users on an almost constant basis. These practices ensure that code can be rapidly deployed to production while ensuring business applications function as expected. This near constant delivery of updates is possible due to optimizations that were made at earlier stages of the development process.

Once a developer feels that a small amount of code is ready, he or she sends it to the quality assurance (QA) team for testing and monitoring. Due to the fact that the app is basically being sent piece by piece in small batches, the QA team can quickly go through the code and find any errors that could emerge. On top of this evaluation by the QA team, the build is also sent to a production-like environment where it is rigorously tested before any changes are released. After all evaluations are completed, the software can be easily deployed.

Continuous delivery makes the process of introducing new functions quick and reliable, and it is extremely beneficial for trying out new features and immediately seeing how customers respond. By launching a new idea piece by piece, you are able to tell if it is clearly communicating the desired action, and you can see if it works properly without having to launch a huge new system.

Some other benefits of continuous delivery include the ability to:

- Install new features at the backend to see how they work with the system
- Respond quickly to market changes
- Make fast modifications based on changes to your business strategy
- Have less errors and a more predictable pace
- Have a strong advantage over competition

What is continuous deployment?

While some use the terms continuous delivery and continuous deployment interchangeably, there are distinct differences between the two that need to be understood and recognized.

As was previously noted, continuous delivery is the almost constant release of updates to users, however, these updates must be manually released by the DevOps team. In contrast, the continuous deployment pipeline is completely automated. As a result of this, users are able to receive updates as soon as the code is written and tested, without waiting for manual intervention by developers. Any testing that needs to happen is performed in production-like environments and is done prior to merging to the mainline branch.

The differences between continuous delivery and continuous deployment will only continue to become increasingly important as technologies like Docker make it easier to automate application deployment. This ease of use allows DevOps team to drop new container images into production and have them deploy automatically. This automated process is key to continuous delivery as it can be performed by anyone within a matter of minutes. Once a deploy occurs, it is important to inspect logs to determine if key metrics are being affected, whether positively or negatively.

There are instances when the use of continuous deployment may be impractical, such as business cases in which IT must wait for features to go live. While application feature toggles could solve this

in many situations, it won't work all the time. The point of this is that your company must decide if continuous deployment makes sense based on business needs instead of based on IT limitations.

What is continuous integration?

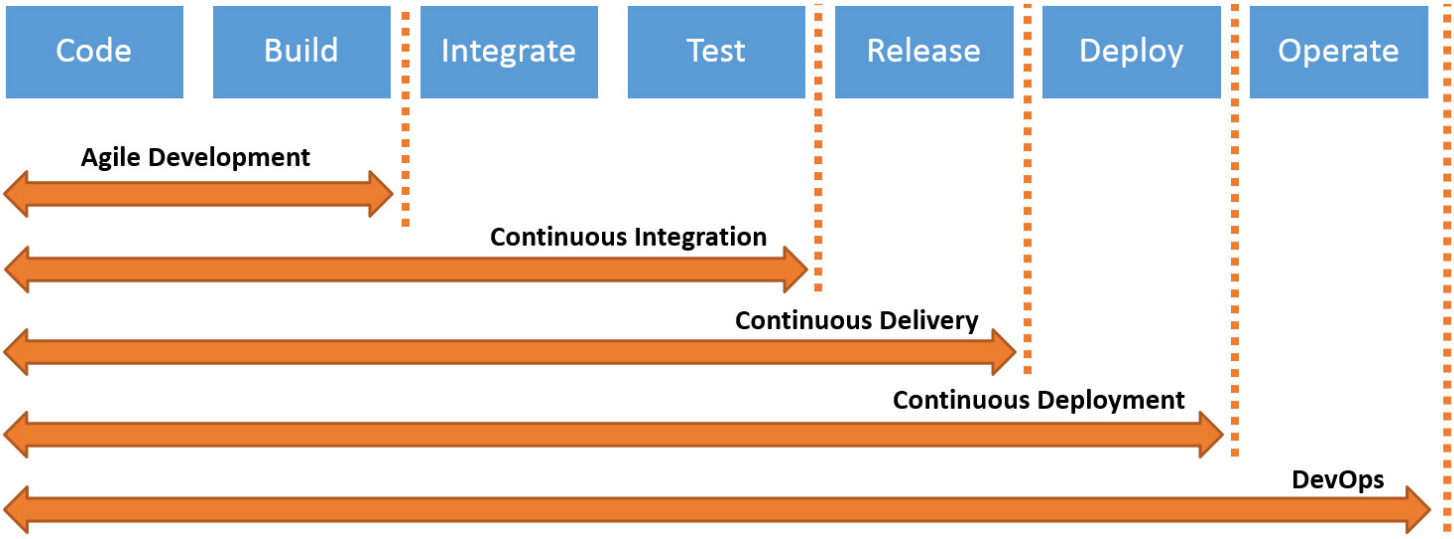
Continuous integration is a key component of Agile Development practices. It is a DevOps software development practice in which code changes are regularly merged into a central repository after tests are run. The main goals of continuous integration are to find and fix potential issues quicker, improve the quality of the software, and decrease the time it takes to release new updates.

Before the practice of continuous integration became widespread, developers would typically work in isolation on codes and then only attempt to merge their changes to the master branch once their individual work was completed. This method of patching together batches of various code made the merging extremely difficult as well as time consuming.

With continuous integration, developers frequently share to the repository, running local unit tests on the code before integrating. A continuous integration server then automatically tests the code that is written by the developers to ensure it can be merged into the main code base without any functional or integration errors. By automating this process, the servers help developers write and test small chunks of code almost constantly, which minimizes the risk of serious problems.

Summary: How they all work together

Once you have moved to a continuous deployment process, you will have multiple components of automation in place. Both the continuous integration build server and continuous delivery must be automated, and you will need to have the ability to automatically deploy to production.



Continuous Integration

The team needs to write automated tests for each new feature or a bug fix.

As regressions are captured early, fewer bugs get shipped to production.

Continuous Delivery

The test suite has to cover enough of your codebase.

Deployment needs to be automated. Updates are supposed to be released manually.

Continuous Deployment

The quality of your test suite can determine the quality of your release.

Deployment pipelines are completely automated and they are triggered automatically for every change.

Developers have to merge their changes as often as possible (at least once a day).	The team may embrace feature flags.	Feature flags are intrinsic part of the process of releasing significant changes.

The ideal process would look like the following:

1. The developer submits the code to the development branch.
2. The continuous integration servers picks up the change, and merges it with the mainline. The server performs unit tests on the code changes and merges them to the staging environment if they are successful.
3. The developer deploys the code to the staging environment where QA tests the environment.
4. The vote is moved to production and the continuous integration server picks it up again and tests it for merging to production.
5. The change is deployed to the production environment.

In the end, all of these "continuous things" contribute to removing development process overhead. Things such as merging code, retesting features after a merge, manually deploying, etc., don't typically contribute to the value of a service, so the practices of continuous delivery, deployment, and integration aim to remove these costs from the process.

If you are a company similar to Flickr, deploying multiple updates and changes every day would make sense. On the other hand, if you are an institution such as a bank, adding different features and functions to your software too often would be confusing for customers and would take away from your business strategy. In the end, it is up to your specific company to decide if utilizing the DevOps practices of continuous delivery, continuous deployment, and continuous integration will be beneficial.

Additional Resources

Flip the Switch On Continuous Delivery from **BMC Software**