IS CONTINUOUS INTEGRATION TO CONTINUOUS DEPLOYMENT A PROGRESSION - OR NOT?

In the world of DevOps, there's no shortage of jargon. You'll hear it in the day-to-day words, like SCRUM, SME, and even the occasional CI/CD. People really struggle with that last one, partially because "CD" in the world of DevOps can mean both "continuous delivery" and "continuous deployment."

DevOps doesn't offer a one-size-fits-all for your organization. This is why it's critical to understand the industry verbiage. Using DevOps in your business, activates efficiency, better communication, and faster product releases. However, if you spend time implementing DevOps aspects that don't make sense for your organization, it can create a negative drain on your company.

In this article, we are going to explore progressions as they relate to CI/CD and CI/CD/CD. That may look like a mess of letters right now but stick with us as we clear it up for you.

(This article is part of our DevOps Guide. Use the right-hand menu to navigate.)

DevOps 101: Definitions

If you're reading this, you likely want an answer to the million dollar question: is CI/CD a progression? But we're going to take it back to basic first by offering some simple definitions that will help beginners understand our discussion.

What is a Progression?

For the purposes of this article, a progression is forward momentum in a chain of DevOps exercises required to reach the end goal of greater efficiency and faster software delivery. A progression is more critical to the end result than a mere recommendation. By nature, the word suggests one thing in the progression has to be completed so the next thing can work effectively and so on and so forth. The order component to progression lends itself to the name, CI/CD.

What is Continuous Integration?

Continuous integration is a term used in software development that encompasses the merging of code where manual coding meets automation. The joint effort between multiple people on a DevOps team work towards the singular goal of streamlining processes while offering consistency. A testing component exists as the coding process typically looks like several units of code being merged together while testing automation occurs with each merger. By having multiple team members work on different parts of what is to be an eventual whole package of code, the testing automation in real-time speeds up the time to production.

Under this coding philosophy, DevOps teams implement small changes, little by little, confirming that any testing is run frequently to check code for problems. Because of the nature of modern coding being dispersed across a number of tools, resources, and platforms, successfully merging the code in one identifiable place serves as a key component to this practice.

In most cases, a version control repository is necessary to implement CI successfully. When one team member finishes working on their part, they submit their code to a repository for it to be merged with the next piece of code. At the cornerstone of CI is consistency. With practices in place to produce, merge and test code packages and applications consistently, teams can rest assured in their ability to collaborate on small code changes, making the overall software package even better.

One implication of CI is continuous testing. This goes beyond simply launching automated tests each time a code is merged. Instead, by integrating the automated testing into the fabric of the CI/CD pipeline, it creates a testing loop, often referred to as continuous testing.

What is Continuous Delivery?

Now that you understand the idea behind continuous integration, here's where it gets a little muddy. In many DevOps organizations, continuous integration isn't an end goal but a means to an end. Some IT leaders come in with the mindset that CI is the first step, with continuous delivery (CD) as the natural second step. Viola! A progression, right?

But let's not get ahead of ourselves. Here's why:

Continuous delivery is a process that usually occurs after ironclad testing in the CI phase. In a nutshell, CD is the automated arm, pushing code packages that have completed the CI cycle into an automated and configured environment for testing and deployment. In DevOps organizations, it's not uncommon for teams to have multiple environments like this within different resources and for different means.

A comprehensive CD environment can consist of several steps with varying degrees of automation and manual execution. These might include:

1. Removing code from version control and executing a new build.
2. Moving code from one computing environment to another.
3. Managing environments and configurations to support code.
4. Executing all required cloud infrastructure steps.
5. Moving application components around to database services, API, or web servers as needed.
6. Creating logs and providing alerts.

As evidenced by the above steps, continuous delivery serves as a means to automating the environment and infrastructure required to support the code produced by the CI loop. For many companies, CI/CD is a means to reach the ultimate goal of continuous deployment, with CD being the necessary intermediary between the two.

However, unlike what some IT leaders may presume, CI/CD is **not** a progression.

In fact, the oft-sought-after continuous deployment isn't necessarily the right fit for all organizations, as we will further discuss below. It's best for DevOps organizations to use anything from CI and CD that makes sense for their enterprise business, in any order that works best for them. Imagine a company that has a great deal of compliance and regulatory constraints in the coding environment. This organization might skip CI completely and focus only on automating delivery. For that reason, CI/CD is not a true progression as we defined it above, because these functions can occur out of order or without the support of the other

What is Continuous Deployment

So that leaves us with continuous deployment (also known as CD, confusing, right?). In many cases when businesses implement CI/CD, what they really want is CI/CD/CD. The philosophy of continuous deployment allows every step of the CI/CD/CD process to be automated without any manual involvement. For example, continuous delivery allows for the automating of merged code from the CI loop into further testing environments. But if you stop there, the act of deployment is still a manual one.

Enter continuous deployment.

With continuous deployment practices implemented successfully throughout a DevOps organization, the entire CI/CD/CD chain is automated, removing the manual step between final testing while moving the code to deployment environments. During this phase, each change is subject to automated testing. When it passes the testing phase, it moves directly to the production phase, allowing businesses to have multiple deployments in a single day.

That said, this final act isn't right for every business. Simply striving for continuous deployment without considering its benefit to your organization is a misguided act. One practice common in continuous deployment which encourages teams to hide work in progress behind features could cause the development process to become more cumbersome while making refactoring even more difficult. As such, business leaders should understand what they are seeking to get out of a CI/CD/CD philosophy without blindly seeking some random gold standard.

Summing It Up

In the end, it's plain to see that neither CI/CD or CI/CD/CD constitutes a true progression. Because they don't rely on one another to move development forward, the best practices from each can be

cherry-picked, creating a coding philosophy that's tailored to the needs of each coding business. The good news is that all enterprise companies incorporating DevOps can benefit from learning about more about the functions and benefits, while also learning how to implement CI/CD or CI/CD/CD or any variation to produce the most consistently efficient results.