CONTINUOUSDELIVERY 101

VOLUME 1

SUZIE PRINCE
ARAVIND S.V.
DAVID RICE

GoCD is an on premises, open source, continuous delivery tool with comprehensive pipeline modeling and value stream map to help you get better visibility into and control of your teams' deployments.

gocd.org

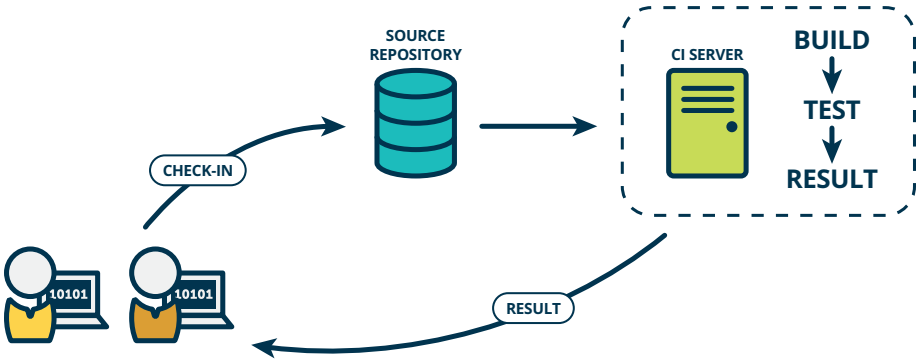
Contents

PART 1: THE BASICS

In this section, we'll look at what is 'DevOps' and how it has improved our way of working. We'll introduce key practices that aid a DevOps culture - Continuous Integration, Continuous Delivery and Continuous Deployment, and highlight the differences between them.

Continuous Integration (CI)

Continuous integration is a practice that puts the integration phase earlier in the development cycle so that building, testing and integrating code happens on a more regular basis.

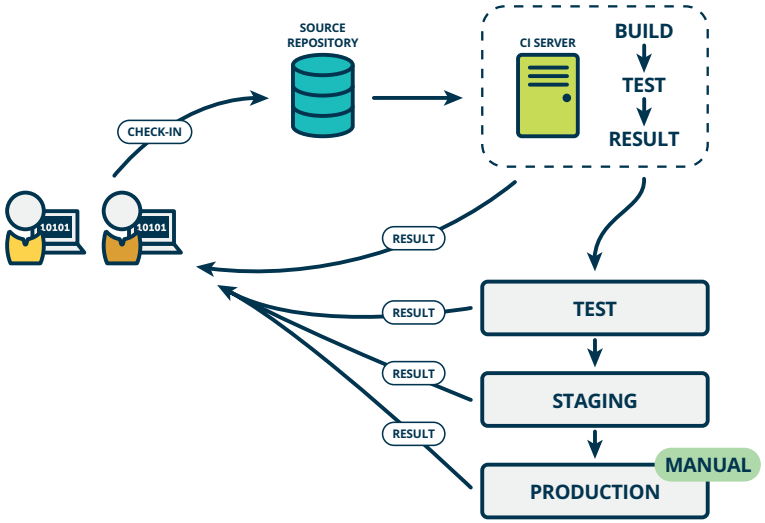


CI means that a developer who writes code on his laptop at home, and another dev who codes on her desktop in the office, can both write software for the same product separately, and integrate their changes together in a place called the "source repository". They can then build the combined software from the bits they each wrote and test that it works the way they expect.

Developers generally use a tool called the "CI server" to do the building and the integration for them. CI requires that developers have self-testing code. This is code that tests itself to ensure that it is working as expected, and these tests are often called unit tests. When all the unit tests pass after the code is integrated, developers will get a green build. This indicates that they have verified that their changes are successfully integrated together, and the code is working as expected by the tests. However, while the integrated code is successfully working together, it not yet ready for production because it has not been tested and verified as working in production-like environments.

Continuous Delivery

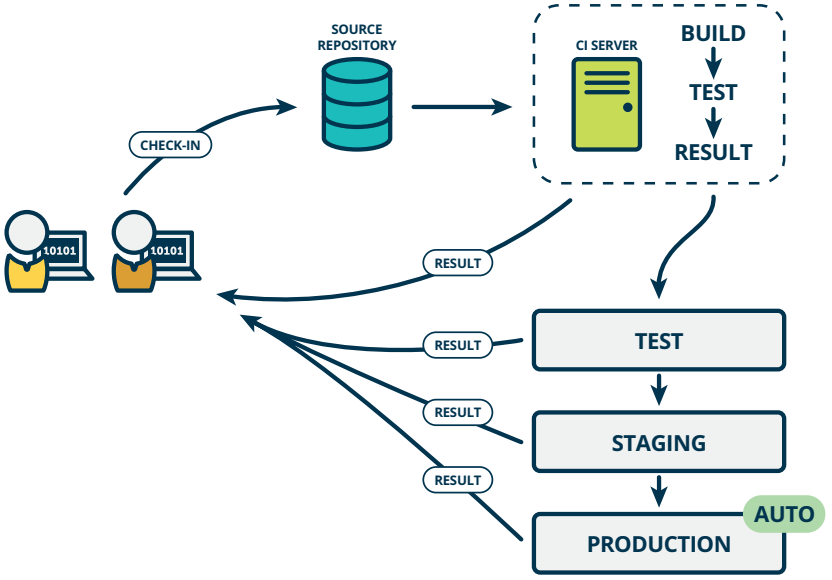
Continuous delivery means that each time the dev team makes changes to the code, integrates and builds the code, they also automatically test this code on environments that are very similar to production. We call this progression of deploying to – and testing on – different environments a "deployment pipeline". Often the deployment pipeline has a development environment, a test environment, and a staging environment, but these stages vary by the team, product and organization.



In each different environment, the code is tested differently. This gives more and more confidence to team that the code will work on the production environment when the code is deployed there. Crucially, the code is only promoted to (tested on) the next environment in the deployment pipeline if it passes the tests of the previous environment. This way the dev team can get new feedback from the tests in each environment and, if there is a failure, they can understand more easily where the issue might be and get it fixed before the code goes to the production environment.

Continuous Deployment

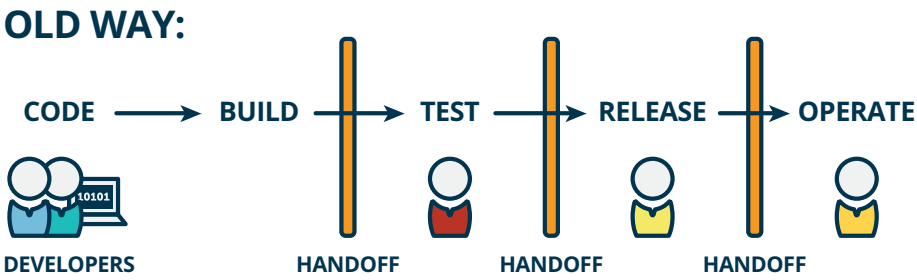
This is a practice where every change that developers make, assuming it passes all the test stages, automatically goes to production. You need to be practicing continuous delivery before you can achieve continuous deployment. Some may want to decide whether to do continuous delivery (versus continuous deployment) from the start, but that decision really can't be made until you're doing CD.

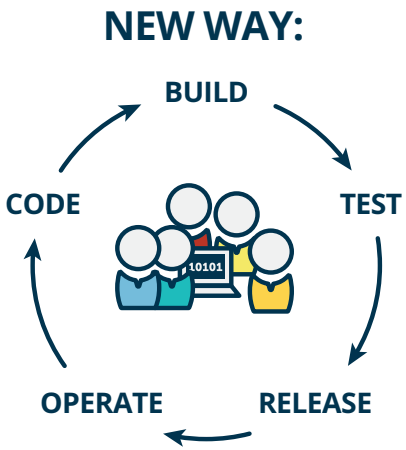


DevOps

The word "DevOps" comes from the combination of the words "development" and "operations". DevOps is a culture that promotes teams developing and operating software working together. Specifically, DevOps refers to cross-functional communication and collaboration of all roles, including things like security and compliance, during the software delivery and deployment process.The goal is the ability to release better quality software more quickly and more reliably.

Common traits of organizations who have a so-called DevOps culture are: autonomous poly-skilled teams, high levels of test and release automation and common goals between the poly-skilled members.





DevOps culture is commonly associated with continuous delivery because they both aim to increase collaboration between development and operations teams, and both use automatic processes to build, test and release software more quickly, frequently and reliably. All things that people like us want.

PART 2: ARE YOU READY FOR CONTINUOUS DELIVERY?

In this section, we will explore some core development practices that are prerequisites for continuous delivery. We'll present questions you need to answer honestly about your own people, teams, and organization to determine your readiness for continuous delivery.

Do you put everything into version control?

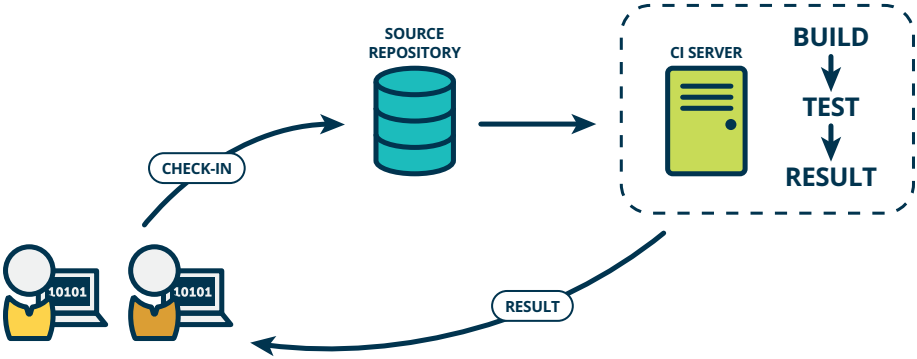
A foundation of CD is the ability to put a specific version of your application into a given environment at any point in time.

- Putting everything needed to make your application into a version control system
- Any time you change anything, push the changes to version control.
- Write an automated script that, given a version, checks out everything from version control and assembles your application

CD is impossible when software teams (or the people on a single team) work in isolation from each other. When development work happens in isolation, our preferred mechanism for doing this is called "trunk-based development":

- Everyone regularly pulls others' changes from version control
- Everyone regularly pushes their changes to version control
- Everyone works in the same place in version control typically called "trunk" or "master"

Do your developers practice Continuous Integration?

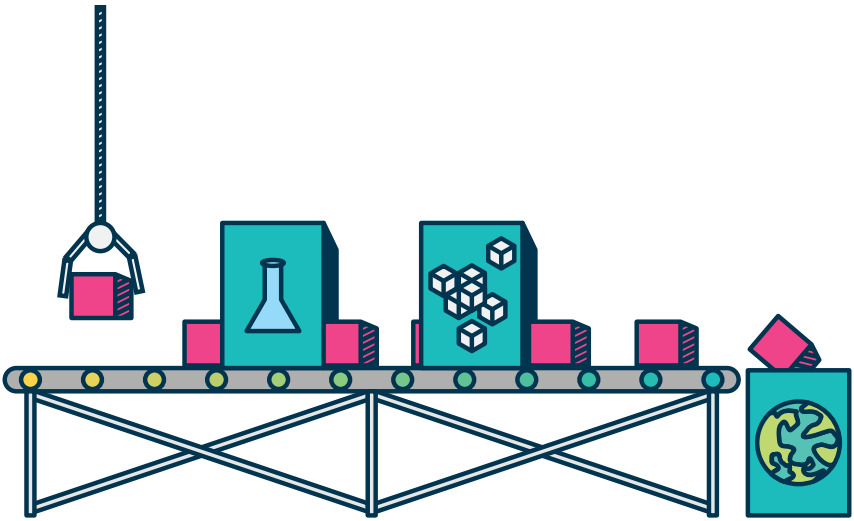


For CD to be successful, the entire organization must trust that your software is high quality and always in a working state. In terms of development team practices, CI is the fundamental building block to achieve this level of trust.

- Developers check code into trunk/master multiple times each day
- Developers maintain a suite of unit tests that verify the code works - before merge, locally, and post merge - on an integration machine or CI server

The end result is a development team that has high trust that the code in trunk/master actually works. This will leave the development team more willing to push code to testers, or even production, more regularly.

Do you automate relentlessly?



To practice CD, the entire team needs to get into the mindset of relentless automation of nearly everything.

Some components and aspects of your process that need automation:

- Tests at different levels, such as unit, integration, UI, regression,security and performance
- Database schema creation, data migration and rollback
- Installer creation and signing (if you have them)
- Generation of documentation for every release
- Last-mile deployment of your application to any environment
- Provisioning of infrastructure all the way from test environments to production
- Provisioning of developer workspaces

Relentless automation might seem daunting. The best approach is to figure out the manual processes you are already using, and then make a plan to gradually automate them. As you begin to achieve small successes, you will want to automate more and more.

There are more small steps you can take early in your continuous delivery journey that will have immediate, positive impacts.

Read more about our insights into continuous delivery at www.gocd.org/blog