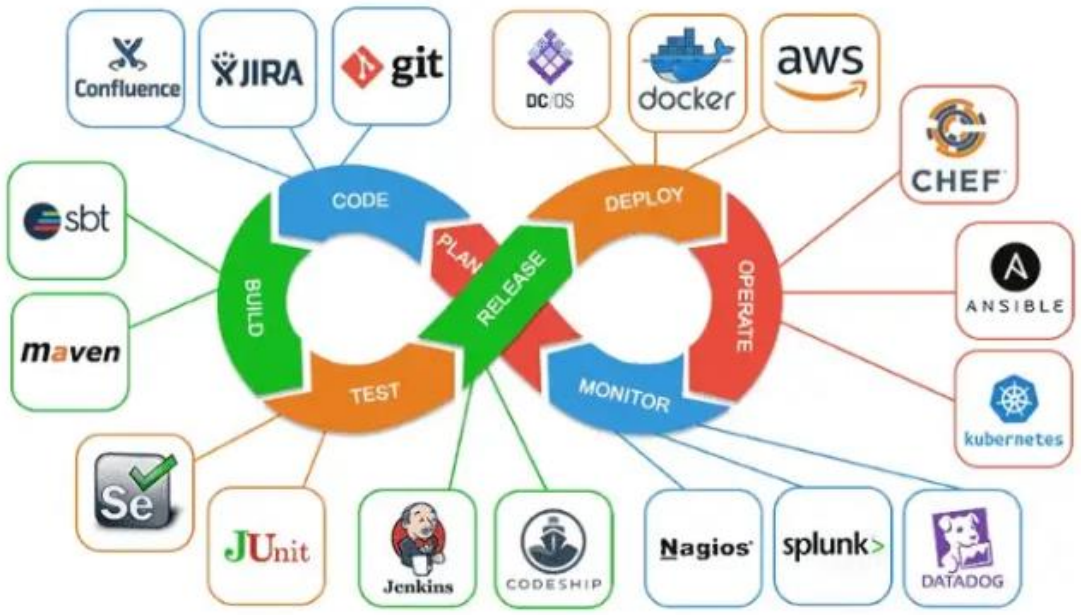
Introduction to CI/CD



Agenda

1- What is ci/cd
2- Why cd/cd
3- difference between CI and CD
4- Continuous integration
5- Continuous delivery
6- What are some common CI/CD tools?
7- CI benefits
8- Cd benefits

1- What is ci/cd

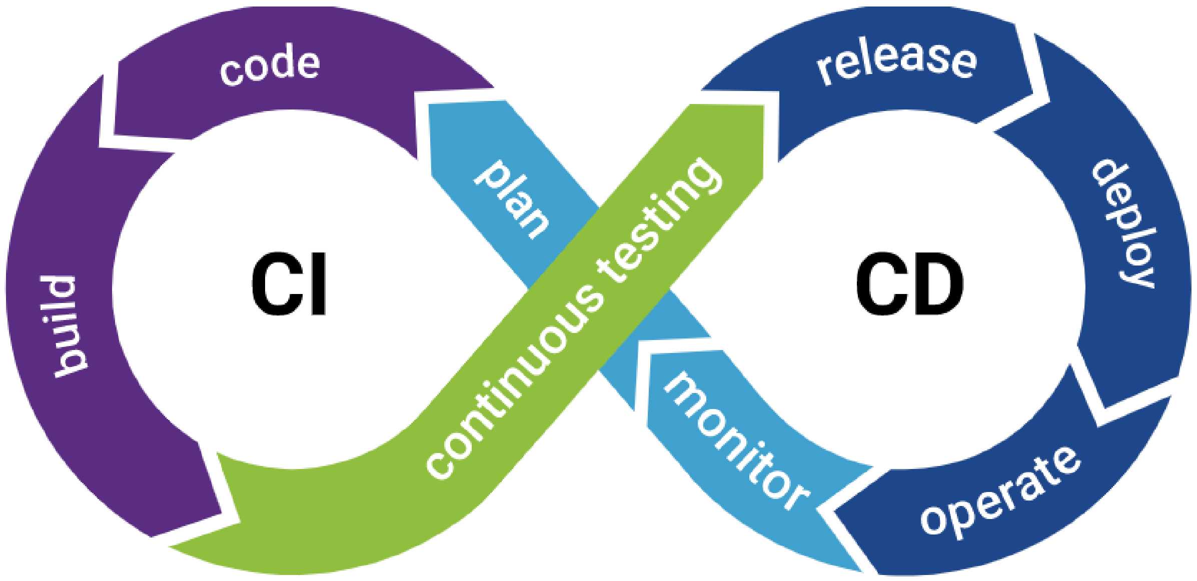
CI and CD stand for continuous integration and continuous delivery/continuous deployment. In very simple terms, CI is a modern software development practice in which incremental code changes are made frequently and reliably. Automated build-and-test steps triggered by CI ensure that code changes being merged into the repository are reliable. The code is then delivered quickly and seamlessly as a part of the CD process. In the software world, the CI/CD pipeline refers to the automation that enables incremental code changes from developers' desktops to be delivered quickly and reliably to production.

2- Why cd/cd

CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code can cause for development and operations teams (AKA "integration hell"). Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment. Taken together, these connected practices are often referred to as a "CI/CD pipeline" and are supported by development and operations teams working together in an agile way with either a DevOps or site reliability engineering (SRE) approach.

I/CD allows organizations to ship software quickly and efficiently. CI/CD facilitates an effective process for getting products to market faster than ever before, continuously delivering code into production, and ensuring an ongoing flow of new features and bug fixes via the most efficient delivery method.

3- difference between CI and CD



The acronym CI/CD has a few different meanings. The "CI" in CI/CD always refers to continuous integration, which is an automation process for developers. Successful CI means new code changes to an app are regularly built, tested, and merged to a shared repository. It's a solution to the problem of having too many branches of an app in development at once that might conflict with each other.

The "CD" in CI/CD refers to continuous delivery and/or continuous deployment, which are related concepts that sometimes get used interchangeably. Both are about automating further stages of the pipeline, but they're sometimes used separately to illustrate just how much automation is happening.

Continuous delivery usually means a developer's changes to an application are automatically bug tested and uploaded to a repository (like GitHub or a container registry), where they can then be deployed to a live production environment by the operations team. It's an answer to the problem of poor visibility and communication between dev and business teams. To that end, the purpose of continuous delivery is to ensure that it takes minimal effort to deploy new code.

Continuous deployment (the other possible "CD") can refer to automatically releasing a developer's changes from the repository to production, where it is usable by customers. It addresses the problem of overloading operations teams with manual processes that slow down app delivery. It builds on the benefits of continuous delivery by automating the next stage in the pipeline.

It’s possible for CI/CD to specify just the connected practices of continuous integration and continuous delivery, or it can also mean all 3 connected practices of continuous integration, continuous delivery, and continuous deployment. To make it more complicated, sometimes "continuous delivery" is used in a way that encompasses the processes of continuous deployment as well.
In the end, it’s probably not worth your time to get bogged down in these semantics—just remember that CI/CD is really a process, often visualized as a pipeline, that involves adding a high degree of ongoing automation and continuous monitoring to app development.
Case-by-case, what the terms refer to depends on how much automation has been built into the CI/CD pipeline. Many enterprises start by adding CI, and then work their way towards automating delivery and deployment down the road, for instance as part of cloud-native apps.
Our experts can help your organization develop the practices, tools, and culture needed to more efficiently modernize existing applications and to build new ones.

4- Continuous integration ”CI”

Following the automation of builds and unit and integration testing in CI, continuous delivery automates the release of that validated code to a repository. So, in order to have an effective continuous delivery process, it’s important that CI is already built into your development pipeline. The goal of continuous delivery is to have a codebase that is always ready for deployment to a production environment.
In continuous delivery, every stage—from the merger of code changes to the delivery of production-ready builds—involves test automation and code release automation. At the end of that process, the operations team is able to deploy an app to production quickly and easily.

5- Continuous deployment “CD”

The final stage of a mature CI/CD pipeline is continuous deployment. As an extension of continuous delivery, which automates the release of a production-ready build to a code repository, continuous deployment automates releasing an app to production. Because there is no manual gate

at the stage of the pipeline before production, continuous deployment relies heavily on well-designed test automation.
In practice, continuous deployment means that a developer's change to a cloud application could go live within minutes of writing it (assuming it passes automated testing). This makes it much easier to continuously receive and incorporate user feedback. Taken together, all of these connected CI/CD practices make deployment of an application less risky, whereby it's easier to release changes to apps in small pieces, rather than all at once. There's also a lot of upfront investment, though, since automated tests will need to be written to accommodate a variety of testing and release stages in the CI/CD pipeline.

6- What are some common CI/CD tools.

CI/CD tools can help a team automate their development, deployment, and testing. Some tools specifically handle the integration (CI) side, some manage development and deployment (CD), while others specialize in continuous testing or related functions.

One of the best known open source tools for CI/CD is the automation server Jenkins. Jenkins is designed to handle anything from a simple CI server to a complete CD hub.

Tekton Pipelines is a CI/CD framework for Kubernetes platforms that provides a standard cloud-native CI/CD experience with containers.

Beyond Jenkins and Tekton Pipelines, other open source CI/CD tools you may wish to investigate include:

- Spinnaker, a CD platform built for multicloud environments.
- GoCD, a CI/CD server with an emphasis on modeling and visualization.
- Concourse, "an open-source continuous thing-doer."
- Screwdriver, a build platform designed for CD.

Teams may also want to consider managed CI/CD tools, which are available from a variety of vendors. The major public cloud providers all offer CI/CD solutions, along with GitLab, CircleCI, Travis CI, Atlassian Bamboo, and many others.

Additionally, any tool that's foundational to DevOps is likely to be part of a CI/CD process. Tools for configuration automation (such as Ansible, Chef, and Puppet), container runtimes (such as Docker, rkt, and cri-o), and container orchestration (Kubernetes) aren't strictly CI/CD tools, but they'll show up in many CI/CD workflows.

7- CI benefits

What you gain:

● Less bugs get shipped to production as regressions are captured early by the automated
tests. ⇒ faster productivity ⇒ more revenue

● Building the release is easy as all integration issues have been solved early. ⇒ faster
productivity ⇒ more revenue

● Less context switching as developers are alerted as soon as they break the build and can
work on fixing it before they move to another task. ⇒ less time wasted ⇒ faster productivity
and higher revnue

● Testing costs are reduced drastically – your CI server can run hundreds of tests in the matter
of seconds. (CI is triggered by a push) ⇒ less waste of resources ⇒ reduce cost

● Your QA team spends less time testing and can focus on significant improvements to the
quality culture. ⇒ faster productivity and higher revenue

8- CD benefits

What you gain

● The complexity of deploying software has been taken away. Your team doesn't have
to spend days preparing for a release anymore. ⇒ less time spent in release ⇒
faster productivity and so higher revenue

● You can release more often, thus accelerating the feedback loop with your

customers. ⇒ customers are more satisfied ⇒ higher revenue
● There is much less pressure on decisions for small changes, hence encouraging
iterating faster. ⇒ faster releases and so higher productivity and revenue
● You can develop faster as there's no need to pause development for releases.
Deployments pipelines are triggered automatically for every change. ⇒ less time
wasted ⇒ faster productivity and higher revenue