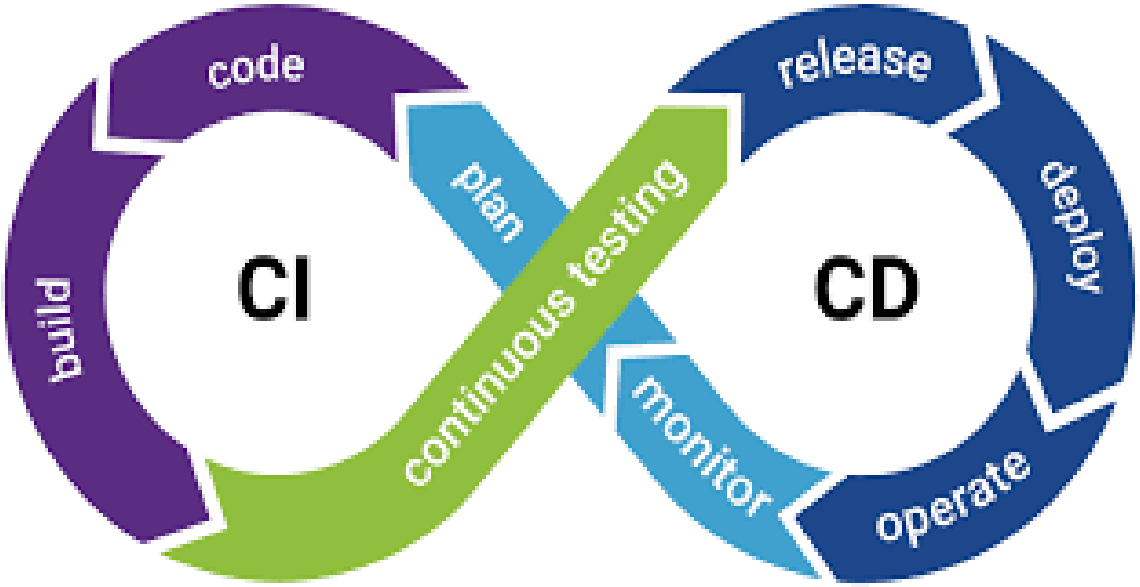
# CI/CD: Continuous integration and continuous delivery



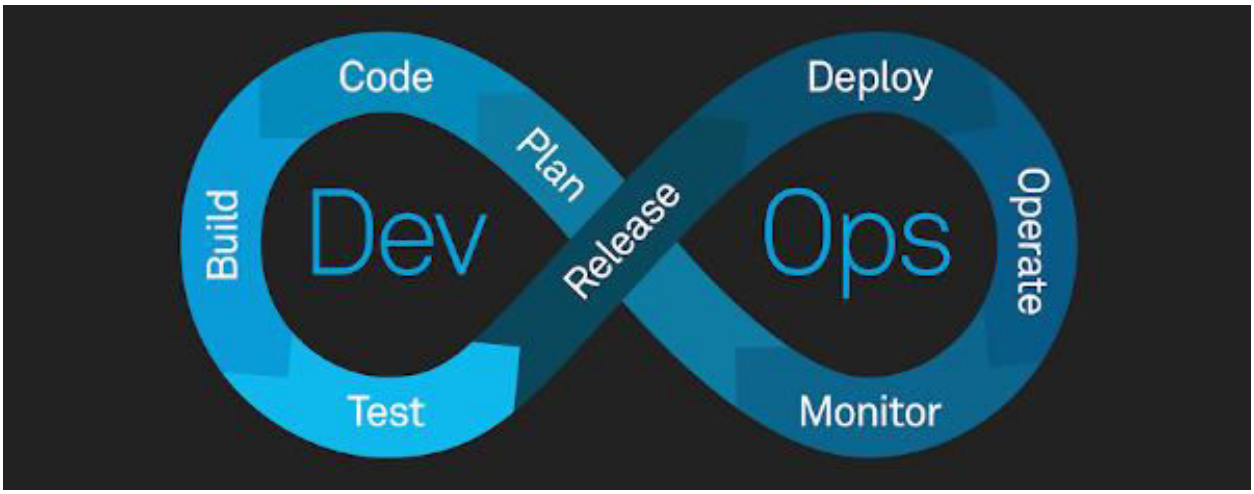

Continuous integration (CI) and continuous delivery (CD), also known as CI/CD, embodies a culture, operating principles, and a set of practices that application development teams use to deliver code changes more frequently and reliably.

CI/CD is a best practice for devops teams. It is also a best practice in agile methodology. By automating integration and delivery, CI/CD lets software development teams focus on meeting business requirements while ensuring code quality and software security.

## Task force

Continuous integration is a coding philosophy and set of practices that drive development teams to frequently implement small code changes and check them in to a version control repository. Most modern applications require developing code using a variety of platforms and tools, so teams need a consistent mechanism to integrate and validate changes. Continuous integration establishes an automated way to build, package, and test their applications. Having a consistent integration process encourages developers to commit code changes more frequently, which leads to better collaboration and code quality.

*Continuous delivery* picks up where continuous integration ends, and automates application delivery to selected environments, including production, development, and testing environments. Continuous delivery is an automated way to push code changes to these environments.




**AUTOMATION**

CI/CD tools help store the environment-specific parameters that must be packaged with each delivery. CI/CD automation then makes any necessary service calls to web servers, databases, and other services that need restarting. It can also execute other procedures following deployment.

Because the objective is to deliver quality code and applications, CI/CD also requires continuous testing. In continuous testing, a set of automated regression, performance, and other tests are executed in the CI/CD pipeline.

A mature devops team with a robust CI/CD pipeline can also implement *continuous deployment*, where application changes run through the CI/CD pipeline and passing builds are deployed directly to the production environment. Some teams practicing continuous deployment elect to deploy daily or even hourly to production, though continuous deployment isn't optimal for every business application.

Organizations that implement a CI/CD pipeline often have several devops best practices in place, including microservices development, serverless architecture, continuous testing, infrastructure as code, and deployment containers. Each of these practices improves process automation and increases the robustness of cloud computing environments. Together, these practices provide a strong foundation to support continuous deployment.

## Builds

In an automated build process, all the software, database, and other components are packaged together. For example, if you were developing a Java application, continuous integration would

package all the static web server files such as HTML, CSS, and JavaScript along with the Java application and any database scripts.

Continuous integration not only packages all the software and database components, but the automation will also execute unit tests and other types of tests. Testing provides vital feedback to developers that their code changes didn't break anything.

Most CI/CD tools let developers kick off builds on demand, triggered by code commits in the version control repository, or on a defined schedule. Teams need to determine the build schedule that works best for the size of the team, the number of daily commits expected, and other application considerations. A best practice is to ensure that commits and builds are fast; otherwise, these processes may impede teams trying to code quickly and commit frequently.

## Goals

CI/CD tools typically support a marketplace of plugins. For example, Jenkins lists more than 1,800 plugins that support integration with third-party platforms, user interface, administration, source code management, and build management.

Once the development team has selected a CI/CD tool, it must ensure that all environment variables are configured outside the application. CI/CD tools allow development teams to set these variables, mask variables such as passwords and account keys, and configure them at the time of deployment for the target environment.

Continuous delivery tools also provide dashboard and reporting functions, which are enhanced when devops teams implement observable CI/CD pipelines. Developers are alerted if a build or delivery fails. The dashboard and reporting functions integrate with version control and agile tools to help developers determine what code changes and user stories made up the build.