What is CI/CD?

CI/CD A continuous integration and continuous deployment (CI/CD) is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation.

By automating CI/CD throughout development, testing, production, and monitoring phases of the software development lifecycle, organizations are able to develop higher quality code, faster. Although it's possible to manually execute each of the steps of a CI/CD pipeline, the true value of CI/CD pipelines is realized through automation

.

Continuous Integration (CI)

CI is a software development practice where development teams frequently commit application code changes into a shared repository. Typically, this would happen at least once or then several times a day (depending on the number of code commits) and this practice encourages committing small changes more often over committing large changes infrequently. These commit/changes automatically trigger new builds which are then validated by automated testing to ensure that they do not break any functionality.

Continuous Delivery (CD)

CD is an extension of that process. CD is all about the ability to continuously deliver integrated code, be it bug fixes or new features, to production. It's the automation of the release process so that new code is deployed to target environments - typically to test or staging environments - in a repeatable and automated fashion.

It means that your "green builds" are ready to go in one click, should you wish to release them.

Continuous Deployment focuses on the automation process to release what is now a fully functional build into production. That means Continuous Deployment goes one step further as it allows you to automatically deploy live every main branch change that passes the CI. However, you might have business reasons for not doing so automatically or you might need to do testing that can't be automated.

What is a CI/CD pipeline?

A pipeline is a process that drives software development through a path of building, testing, and deploying code, also known as CI/CD. By automating the process, the objective is to minimize human error and maintain a consistent process for how software is released. Tools that are included in the pipeline could include compiling code, unit tests, code analysis, security, and binaries creation. For containerized environments, this pipeline would also include packaging the code into a container image to be deployed across a hybrid cloud.

CI/CD is the backbone of a DevOps methodology, bringing developers and IT operations teams together to deploy software. As custom applications become key to how companies differentiate, the rate at which code can be released has become a competitive differentiator.

Examples of CI/CD in Real Projects

ReactJS - The React JavaScript framework developed and maintained by Facebook has a great example of a robust and high visible CI/CD pipeline. It uses CircleCI.

TensorFlow - Google's TensorFlow is an open-source Python framework for building ML and AI applications. Since TensorFlow is a cross-platform tool that can run on CPU and GPU hardware, the maintainers have set up multiple pipelines to build and test the tool on different operating system platforms and CPU/GPU configurations. Their pipelines are built using Jenkins.

Homebrew - If you do any sort of development on macOS, then you've probably at least heard of the popular package manager Homebrew. It uses TravisCI with coverage reporting on codeov.io.

Benefits of continuous integration and delivery (CI/CD)

Let's dig into what implementing CI/CD to your everyday software development process can bring to the table. Realizing these benefits means reducing risks for each build and clearing the way to get your valuable features out to customers faster.

Fast feedback loop - In software development, what you don't know really can hurt you. If there's anything that really slows you down when developing software, it's the lack of feedback on the quality and impact of the changes you made. You might think you move fast if you quickly commit code and move on to another task without running any tests or sanity checks. The reality will hit you later when you're trying to figure out which change and from when (and by whom…) broke something.

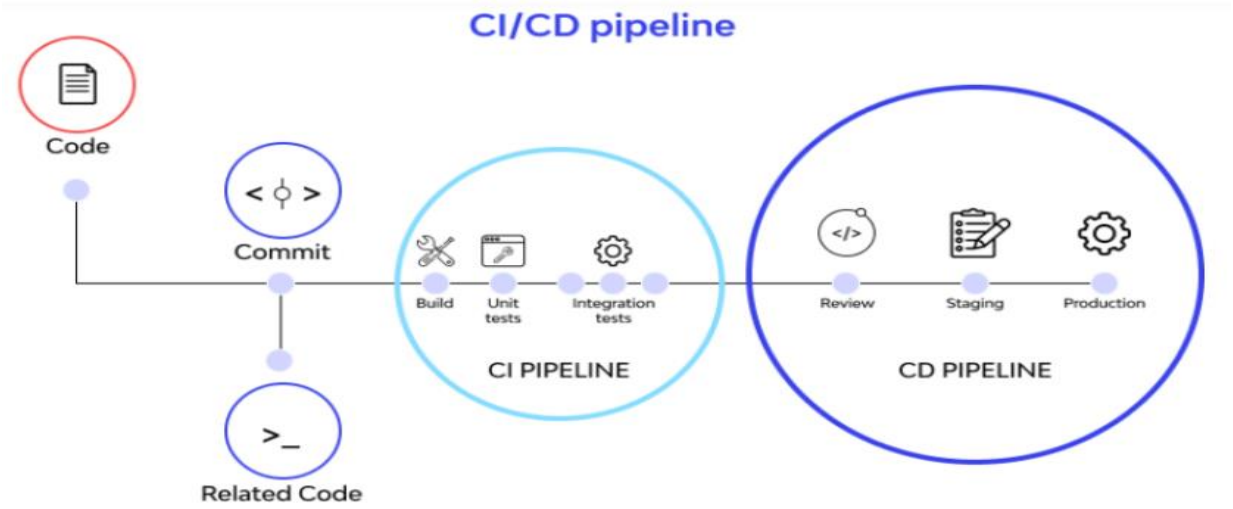
Continuous integration tools - when properly used - will remove much of this headache by providing you with quick answers to the question "did I break something?" for each commit.

Increase transparency and visibility - When your CI/CD pipeline is set up, your entire team will know what's going on with the builds as well as get the latest results of tests, which means they can raise issues and plan their work in context. You can see which changes tend to break builds more often.

Avoid "Integration Hell" (Better Time Mgmt. and Release Mgmt.) - If you think of software as a set of Lego pieces, each of which is crafted by an individual developer separately, it becomes clear that making different pieces go "click!" without any effort and friction is what helps the team make progress every day. If a specific piece is fine on its own in terms of implemented code, you will still need to make sure it plays nice with everything else. Continuous integration supports connecting the pieces of your software every day.

Detect and fix issues early - The thing with bugs in software is that they hide other bugs that hide other bugs that… yes, you guessed it, hide more bugs. The more bugs pile up, the harder it is to test and find them, resulting in nasty last-minute surprises (especially on a Friday night). If various kinds of useful automated tests are run in your continuous integration pipeline, you'll be able to know what to fix as soon as a test fails. Not all of the testing can be automated and it takes time to automate what can be automated, but doing so helps you develop software in a sustainable way and keep the technical debt at a minimum.

Improve quality and testability - The easier it is to test something, the easier it is to determine its quality. Testability has multiple dimensions. On the inside, testability can be characterized by how controllable, observable, unbuggy, and decomposable your product is. Simply put, if your code is written in a way that doesn't accommodate writing tests for it, you will have a hard time making it unbuggy. On the outside, testability is affected by how easily new builds are available, what kind of tools you have, and how much control you have over your test environments. Continuous integration and delivery drive both since you will need to write tests and run them, and also deliver builds frequently and reliably.



In a nutshell, by adopting the CI/CD approach, your business stands to gain the following:

- Detect issues in software builds faster and, therefore, facilitating quicker resolutions
- End-user involvement and feedback during CD can lead to usability improvements
- Deliver software on time with faster time to market
- Showcase your capability in the market, as you adopt (CI/CD) for producing quality mobile apps
- Forgo the stale periods in the lives of products
- Back-to-back releases using CD are easy and less time-consuming
- Share visibility of the development process in real-time

Conclusion

CI/CD are two DevOps best practices as they tackle the misalignment between developers and the operational team. With the presence of automation, developers can release changes and new features more frequently, while operation teams have better overall stability.

Therefore, CI/CD is integral in software building and deployment: smaller teams, constant changes, fast and real-time feedback, and app deployment. Not only do they provide clear benefits to businesses but also to stakeholders such as product owners, development teams, and end-users, just to name a few.

Thanks ,
Ali Ahmed