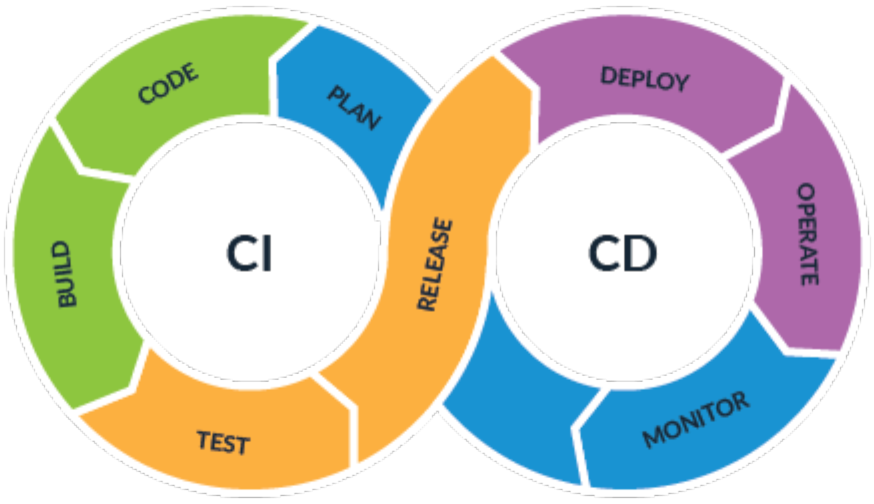
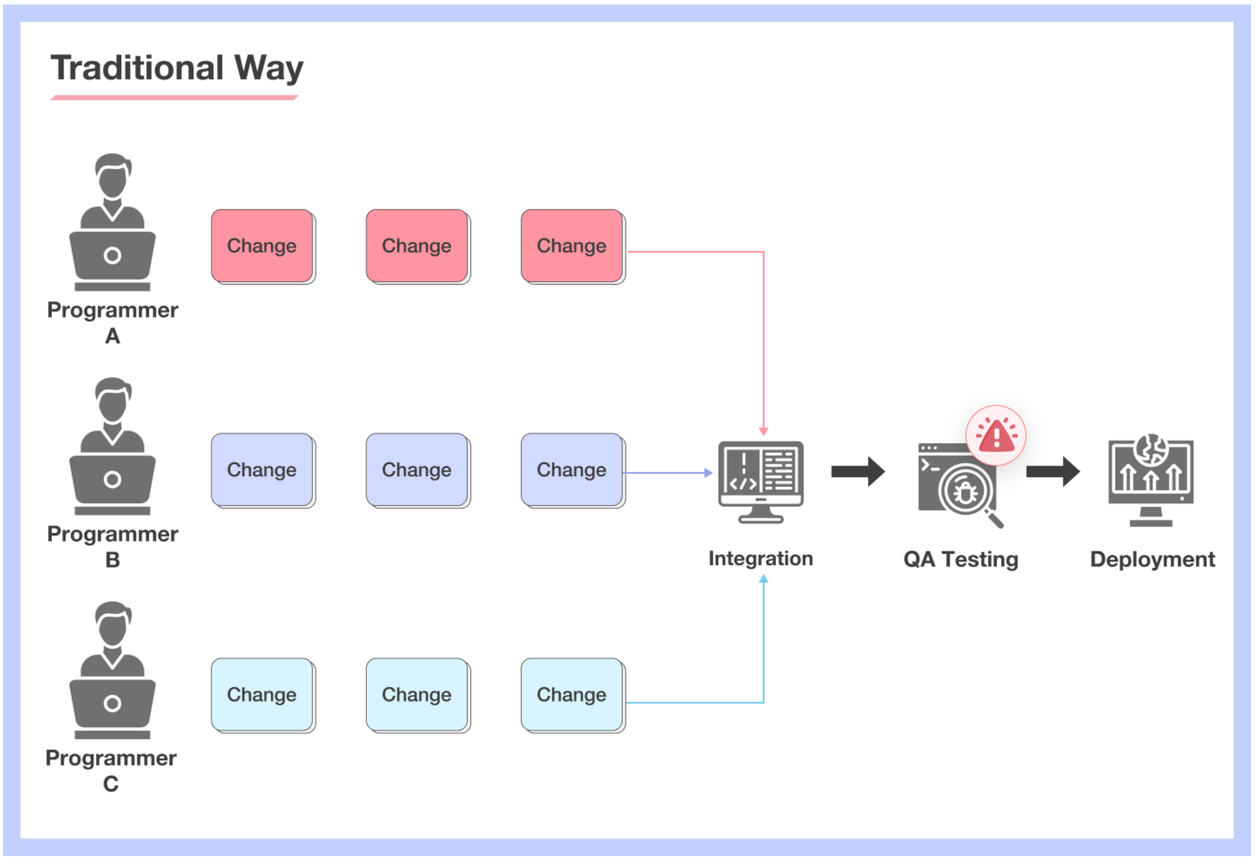
Understanding the key difference between CI & CD

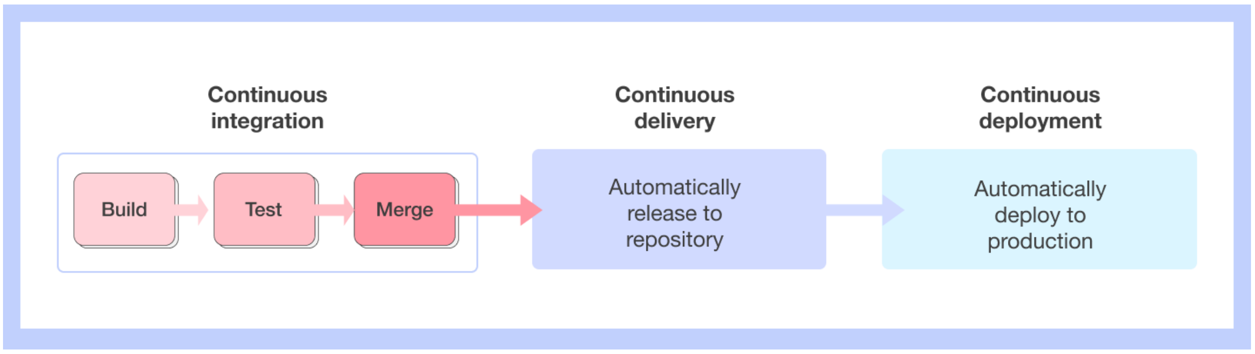


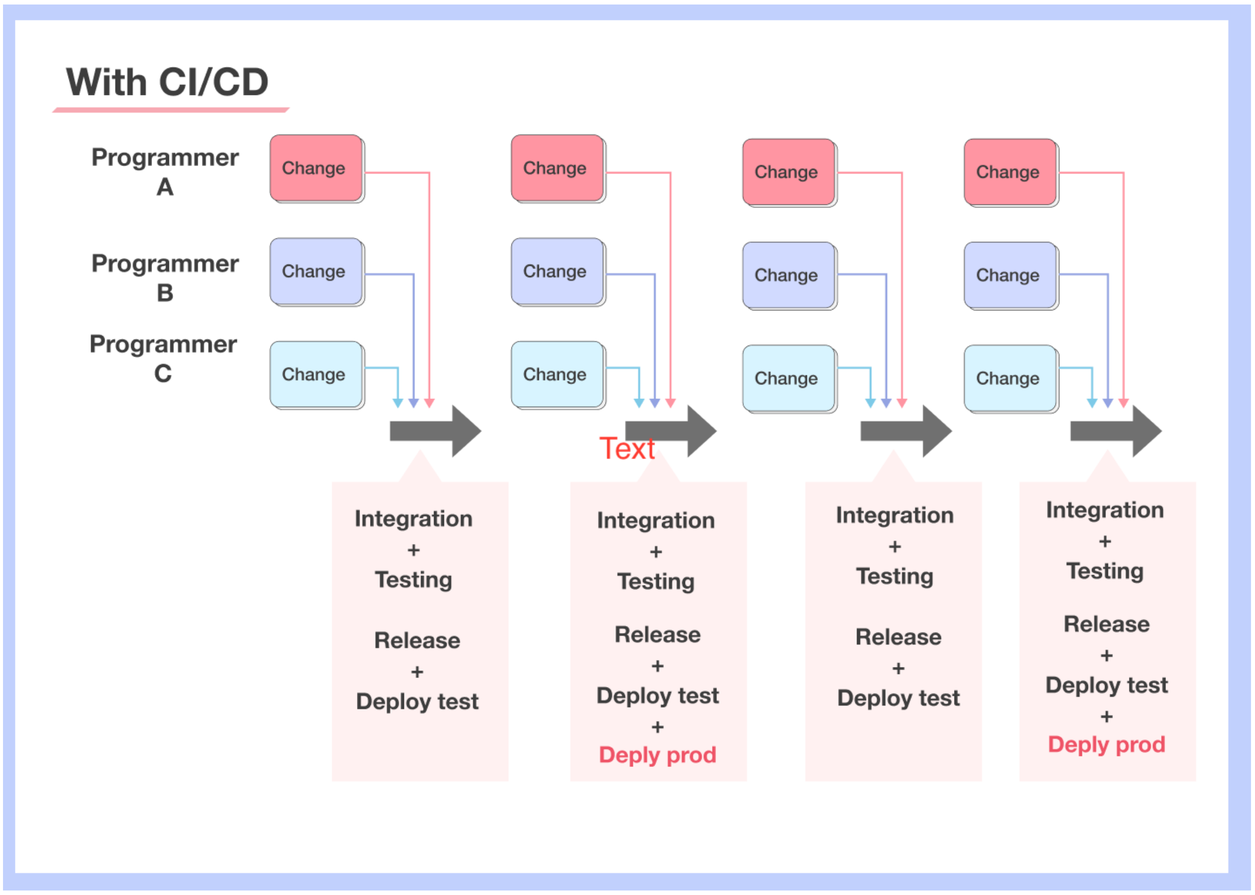
Starting with a little flashback– with the traditional way of development, challenges arise when developers set up to merge all the code changes in a single integration.

Each developer working on different features writes and tests code individually, as shown in the image below. But they use long-running feature branches existing for several weeks or even months to integrate the code changes once into production. As a result, the changes in the application often conflict with similar changes made by other developers. Things become worse when deployment is done only once or infrequently.



However, modern software development demands a system where multiple developers can simultaneously work on different features of the same application and deploy changes seamlessly. Enter CI/CD.





Continuous integration

enables merging code changes back to a shared branch, commonly referred to as the trunk, multiple times daily. After the changes are merged, they are validated by automatic application building and testing, including unit and integration tests, that ensure the changes do not break the application. Even if the automated tests find conflicts between the two codes, CI simplifies fixing them.

Continuous delivery

After automating the builds and testing in CI, the role of continuous delivery is to automate the release of the validated code to a repository. Therefore, an effective continuous delivery process requires a built-in CI in the development pipeline because the entire purpose of continuous delivery is to maintain a codebase that is always ready to be deployed in a production environment.

Continuous deployment

The concluding stage of a CI/CD pipeline is continuous deployment, which acts as an extension of continuous delivery. It is responsible for automating the release of a build ready for production. Moreover, due to the lack of manual intervention, well-structured test automation is the foundation of continuous deployment.

It facilitates a change to go live in an application within a few minutes of writing it. The entire practice of deploying applications becomes less risky, and changes are released in small pieces instead of as a whole.

What are the key CI/CD benefits you need to know about?

1. Reduced time-to-market

The ultimate goal of a CI/CD pipeline is to build and deliver software to users at a rapid pace. Moreover, software development has gone beyond introducing new features, writing robust code, and understanding users' needs. A CI/CD pipeline enables you to ship changes not just weekly, but daily, and even hourly.

Therefore, it is possible to launch new features faster and implement deployment strategies to help collect feedback that can be incorporated promptly into the upcoming update. The ability to push changes as and when they come gives you the confidence to respond to changing trends and work on the pain points more effectively.

HackerOne, as a security platform, was distributed across the globe and required a CI/CD pipeline that could eliminate the dependencies between teams. GitLab allowed the company to leverage a centralized location for its tools and API to create a bot that effectively scan repos and then create the merge requests for the ones that require updation.

Consequently, HackerOne achieved 5x faster deployment time, which meant 4-5 deploys per day instead of earlier single deployment daily. It also saved 4 hours of development time per engineer weekly, significantly improving the team performance.

2. Effective automation testing

Understanding your code's performance is vital in a software release; performing it efficiently requires effort and hours. Manual testing is a repetitive process that demands a high concentration level and demands developers to perform the same process with minute variations for the umpteenth time. Therefore, a major aspect of any CI/CD pipeline is to have a set of automated tests that every build must complete.

Automated tests ensure that they perform more consistently and run tests more frequently and faster than manual testing. Also, your QA and test engineers can concentrate on discovering new failure modes, and these results can be used to extend the overall test coverage.

3. Improved Mean Time to Resolution (MTTR)

With the help of MTTR, development teams can measure the sustainability of repairable features. Also, it establishes the standard time to fix a faulty feature and enables you to determine the time required for failure recovery. CI/CD significantly helps reduce the MTTR, as there are only minor changes in the code and detecting fault isolations is less complicated.

4. Efficient infrastructure

A CI/CD pipeline is largely built on automation that aims to make the release process repeatable and reliable. While building an infrastructure with CI/CD, your initial goal will be to write and run automated tests. After building a strong foundation, your next goal involves automating deployments of your builds to test and staging environments.

The ideal practice is to take the infrastructure-as-code approach, which automates the development process of these environments. Instead of manually managing independent servers, their configuration is documented and stored in version control, making it easier to create new environments. These environments can be integrated online without any risks of unexpected changes and inconsistencies, making continuous delivery faster.

5. Accurate progress monitoring

A majority of the CI/CD tools can be used for instrumenting the process and provide a set of metrics that include building, test coverage, failure rates, test fix times, and more. This data can help you recognize areas causing your pipeline to perform slowly and make improvements as and when required.

More importantly, when you consistently extend your test coverage, witness a reduction in failure rates, and increase release frequency, combined team effort reflects your overall metrics. By effectively measuring your CI/CD pipeline's performance, you can determine if and to what extent it supports your organization's goals.

6. Maximum consumer demand satisfaction

CI/CD helps you adopt and incorporate a customer-first approach. When you release a product, it carefully monitors the initial actions of the customers and maintains a record of the results. As a result, you can study the kind of impression your product creates on the customers.

Implementing CI/CD can facilitate end-user involvement and feedback in the continuous development stage that helps usability modifications. Moreover, it lets you keep your product up-to-date with constant checks for new updates or minor changes. These factors contribute to a high level of user satisfaction.

7. Rapid feedback cycle

Timely feedback is an essential part of the DevOps approach, and it begins with automated build and test stages to track the immediate faults. It enables you to work more efficiently over a lengthy feedback cycle. Also, shipping updates provide rapid feedback on the build than waiting for a big release every few months.

Combining the feedback, studying the user behavior, and tracking the key performance indicators can give insights into what works for your organization and how to design changes or improvements.

Moreover, frequent releases allow you to try out different designs and compare their performance with A/B testing or compare over time for better insights.

8. Cloud-based development

A major share of CI/CD's popularity can be attributed to the rise in cloud development since traditional development techniques often prove inefficient with cloud-based applications. It has become the central pillar of streamlining cloud-native development and making the processes faster, consistent, and more efficient.

A CI/CD pipeline leverages cloud-native tools like Jenkins to automate the DevOps workflow, which initiates the testing process early on. Therefore, changes and new code blocks are automatically deployed in the production environment. This process maintains the continuous flow of new features and updates – an essential aspect of modern software development.

9. Enhance communication and collaboration.

The central point of the DevOps approach is building a collaborative culture and making CI/CD successful. You must bring down the walls between teams and encourage more communication to achieve this. Breaking down the silos between the development and operations teams is a good start. At the same time, CI/CD works as a common framework for developers, QAs, and product managers to coordinate on a project.

For every pipeline running on a CI/CD environment, the stakeholders are notified about the changes and failures. The tools used for CI/CD pipeline management allow even non-developers to know what is trained and give them access to staging environments to give feedback on product development. This practice fosters innovation among developers and confidence among product owners.

10. Shorter review cycles and faster debugging

Continuous integration encourages developers to commit their code changes more frequently, at least once daily. Regularly sharing code with the team ensures that every member is building on the same principles, code reviews are quicker, and integrating changes is less time-consuming. Uploading smaller increments also reduces the code reviewer's responsibilities and encourages the entire team to work in collaboration rather than fixing bugs in isolation. This practice also allows remote developers to participate in the process and removes the communication barrier.

Moreover, CI can be integrated with Version Control Systems, which means when developers push a new change to the merge, it triggers a CI run. This step automatically checks the code coverage and tests and eliminates the time spent reviewing codes and shipping them to production.