Deployment Made Easy through CICD

Anytime, Anywhere and Automated.

Definition

CI and CD stands 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.

Why CICD ?

CI/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.

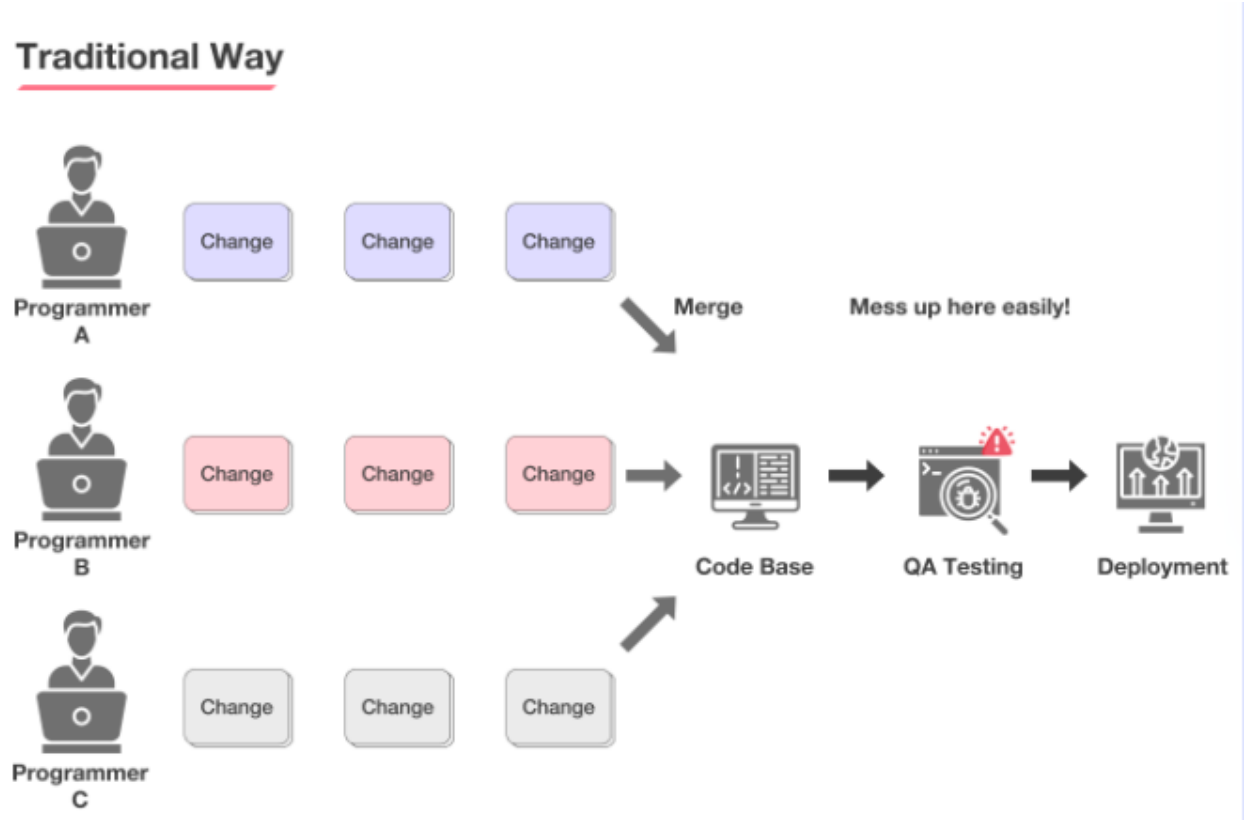
CI vs CDelivery vs CDeployment

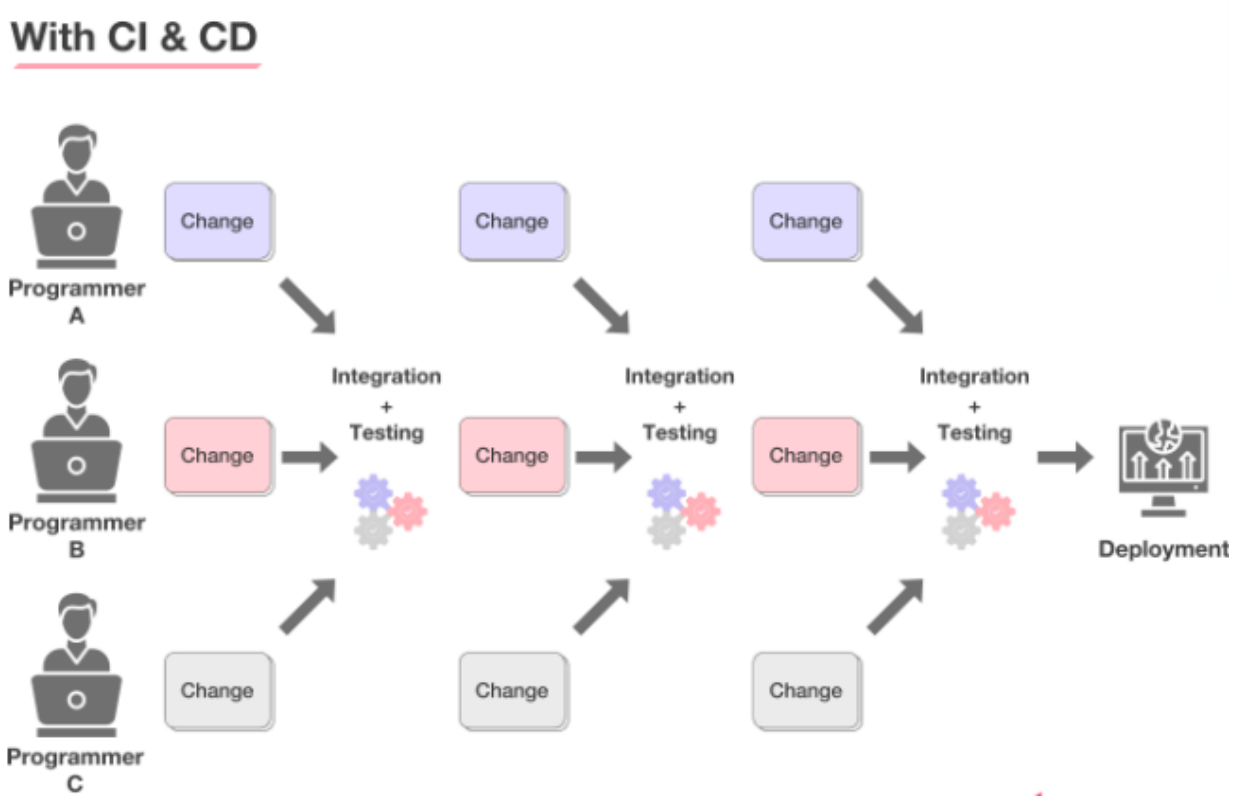
Continuous integration (CI) is practice that involves developers making small changes and checks to their code. Due to the scale of requirements and the number of steps involved, this process is automated to ensure that teams can build, test, and package their applications in a reliable and repeatable way.

Continuous Delivery (CD) is the automated delivery of completed code to environments like testing and development. CD provides an automated and consistent way for code to be delivered to these environments.

Continuous Deployment is the next step of continuous delivery. Every change that passes the automated tests is automatically placed in production, resulting in many production deployments.

Ways of Working: Traditional vs CICD





Key Takeaways

1. Reduce time-to Market

   The 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, daily, and even hourly.

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.

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.

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.

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.

7. 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.

8. 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.