What is CI/CD?

CI or **Continuous Integration** is the practice of automating the integration of code changes from multiple developers into a single codebase.

It is a software development practice where the developers commit their work frequently into the central code repository (Github or Stash).
Then there are automated tools that build the newly committed code and do a code review, etc as required upon integration.

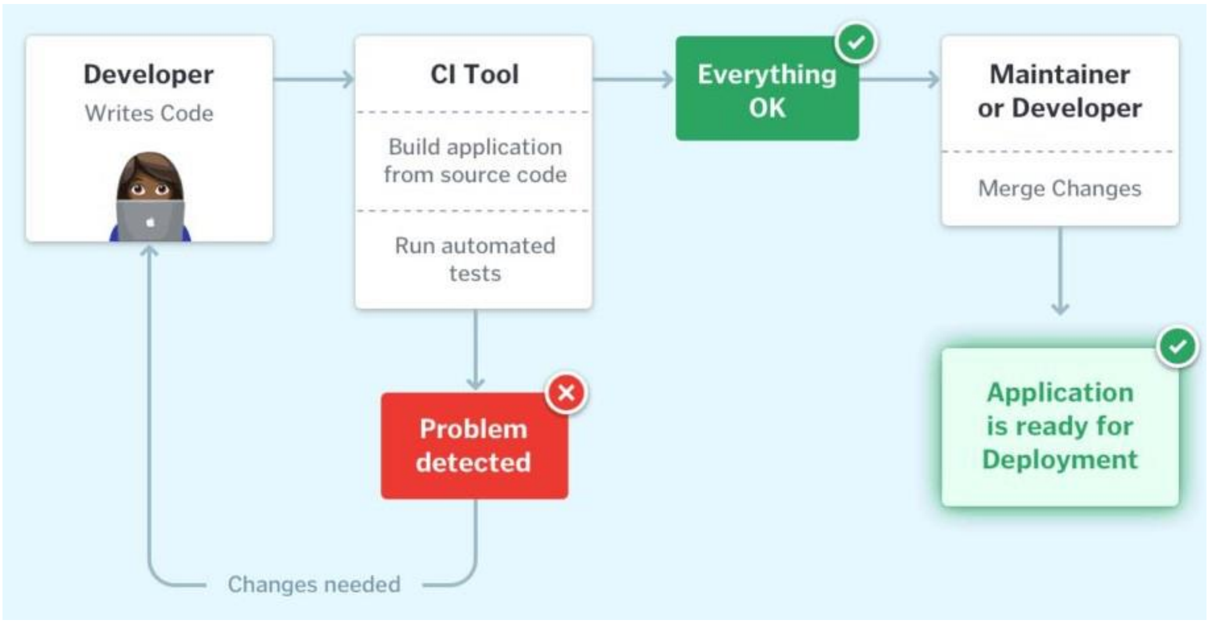
Why CI is important?

There could be scenarios when developers in a team, work in isolation for an extended period of time and only merge their changes to the master branch once their work was completed.
This not only makes the merging of code very difficult, prone to conflicts, and time-consuming but also results in bugs accumulating for a long time which are only identified in later stages of development. These factors make it harder to deliver updates to customers quickly.

With Continuous Integration, developers frequently commit to a shared common repository using a version control system such as Git. A continuous integration pipeline can automatically run builds, store the artifacts, run unit tests and even conduct code reviews using tools like Sonar. We can configure the CI pipeline to be triggered every time there is a commit/merge in the codebase.

CI helps to find and address bugs quicker, make the process of integrating code across a team of developers easier, improve software quality and reduce the time it takes to release new feature updates. Some popular CI tools are Jenkins, TeamCity, etc.

How CI Works?



Once the developer commits their code to a version control system like Git, it triggers the CI pipeline which fetches the changes and runs automated build and unit tests. Based on the status of the step, the server then notifies the concerned developer whether the integration of the new code to the existing code base was a success or a failure.

This helps in finding and addressing the bugs much quickly, makes the team more productive by freeing the developers from manual tasks, and helps teams deliver updates to their customers more frequently. It has been found that integrating the entire development cycle can reduce the developer's time involved by ~25 – 30%.

CD or Continuous Delivery

CD or **Continuous Delivery** is carried out after Continuous Integration to make sure that we can release new changes to our customers quickly in an error-free way. This includes running integration and regression tests in the staging area (similar to the production environment) so that the final release is not broken in production. It ensures to automate the release process so that we have a release-ready product at all times and we can deploy our application at any point in time.
Continuous Delivery automates the entire software release process. The final decision to deploy to a live production environment can be triggered by the developer/project lead as required. Some popular CD tools are AWS CodeDeploy, Jenkins, and GitLab.

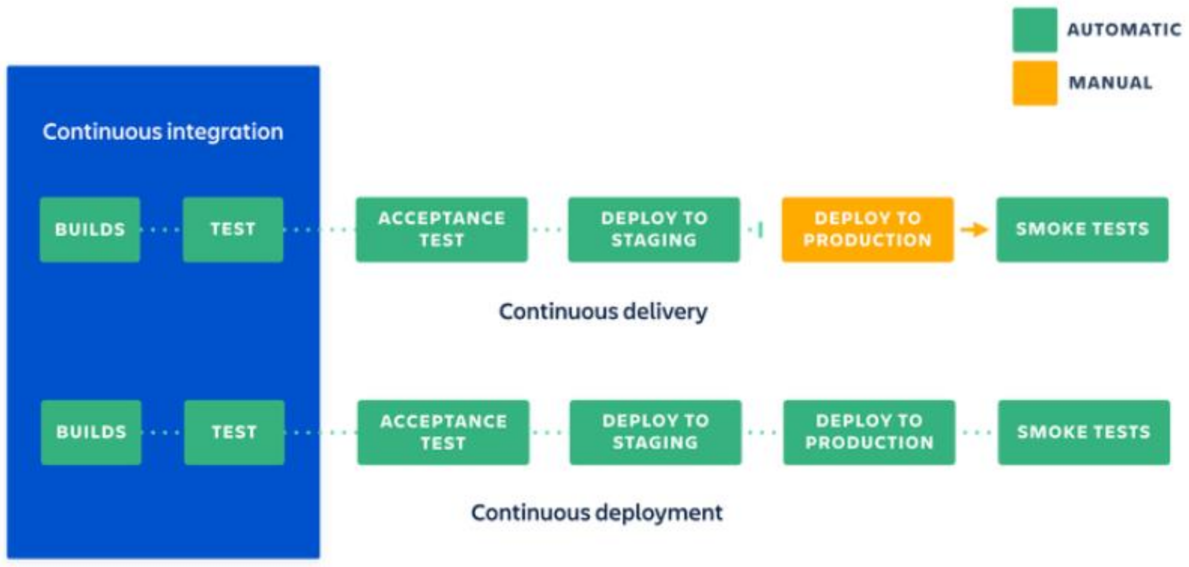
Why CD is Important?

Continuous delivery helps developers test their code in a production-similar environment, hence preventing any last moment or post-production surprises. These tests may include UI testing, load testing, integration testing, etc. It helps developers discover and resolve bugs pre-emptively.

By automating the software release process, CD contributes to low-risk releases, lower costs, better software quality, improved productivity levels, and most importantly, it helps us deliver updates to customers faster and more frequently. If Continuous Delivery is implemented properly, we will always have a deployment-ready code that has passed through a standardized test process.

How CI and CD work together?

The below image describes how Continuous Integration combined with Continuous Delivery helps quicken the software delivery process with lower risks and improved quality.



CI / CD workflow

We have seen how Continuous Integration automates the process of building, testing, and packaging the source code as soon as it is committed to the code repository by the developers.

Once the CI step is completed, the code is deployed to the staging environment where it undergoes further automated testing (like Acceptance testing, Regression testing, etc.). Finally, it is deployed to the production environment for the final release of the product.

If the deployment to production is a manual step, the process is called Continuous Delivery whereas if the process of deployment to the production environment is automated, it is referred to as Continuous Deployment.

Why is CI/CD important?

CI/ CD enables organizations to develop software quickly and efficiently. CI/CD enables an effective process for getting products and software to market faster than ever before, continuously moving code into production, and ensuring a steady flow of new features

What are the benefits of CI/CD?

Automated testing enables continuous delivery that ensures software quality and safety and increases code profitability in production. CI/ CD pipelines enable a much shorter time-to-market for new product features.

The significant increase in overall delivery speed enabled by CI/CD pipelines improves a company’s competitive advantage.
Automation allows team members to focus on what they do best, resulting in the best end products. Companies with a successful CI/CD pipeline can attract outstanding talent.

By moving away from traditional waterfall methods, engineers and developers are no longer engaged in repetitive activities that are often highly dependent on completing other tasks.

DevOps

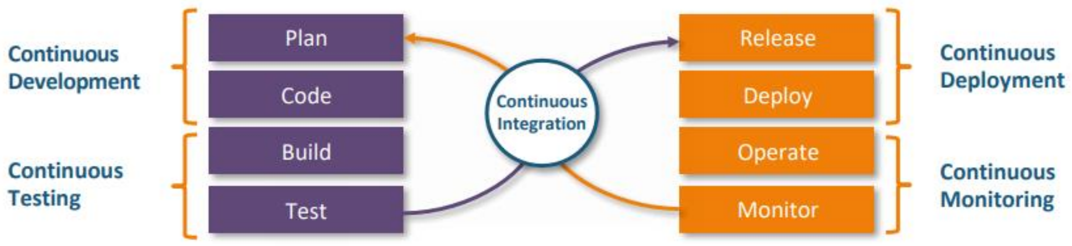
- Software development methodology that improves the collaboration between developers and operations teams using various automation tools. These automation tools are implemented using various stages which are a part of the DevOps Lifecycle.

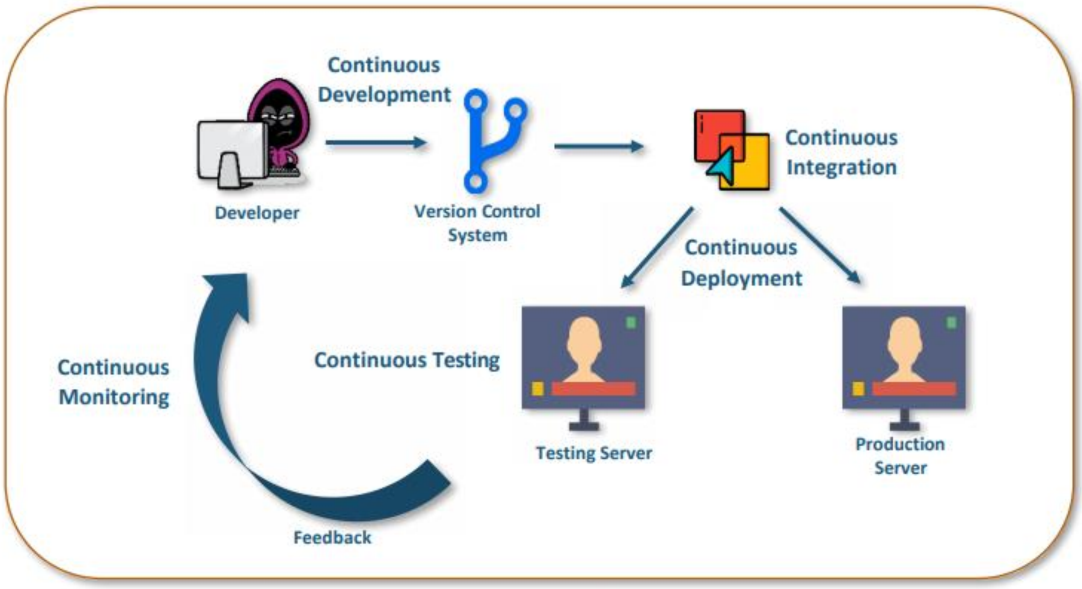
Goal: The goal of DevOps is to increase an organization speed when it comes to delivering applications and services. Many companies have successfully implemented devOps to enhance their user experience like amazon, netflix etc.

Industries have started to gear up for the digital transformation by shifting their means to weeks and months instead of years while maintaining high quality as a result. The solution to all this is- DevOps.

How DevOps Works?

The DevOps Lifecycle divides the SDLC lifecycle into the following stages:





Automated CI/CD Pipeline

1. Continuous Development:
This stage involves committing code to version control tools such as Git or SVN for maintaining the different versions of the code, and tools like Ant, Maven, Gradle for building/packaging the code into an executable file that can be forwarded to the QAs for testing.

2. Continuous Integration:
The stage is a critical point in the whole DevOps Lifecycle. It deals with integrating the different stages of the DevOps lifecycle and is, therefore, the key in automating the whole DevOps Process.

3. Continuous Deployment:
In this stage the code is built, the environment or the application is containerized and is pushed onto the desired server. The key processes in this stage are Configuration Management, Virtualization, and Containerization.

4. Continuous Testing:
The stage deals with automated testing of the application pushed by the developer. If there is an error, the message is sent back to the integration tool, this tool, in turn, notifies the developer of the error, If the test was a success, the message is sent to Integration-tool which pushes the build on the production server.

5. Continuous Monitoring:
The stage continuously monitors the deployed application for bugs or crashes. It can also be set up to collect user feedback. The collected data is then sent to the developers to improve the application.

What is DevOps Security or DevSecOps?

- DevOps security is a philosophy that combines three words:

  development, operations, and security.

  The goal is to remove any barriers that may exist between software development and IT operations. As code is written and applications are developed, the value of constant communication and collaboration between teams cannot be emphasized enough.

Benefits of DevSecOps

The two main benefits of DevSecOps are speed and security.

Development teams deliver better, more-secure code faster, and, therefore, cheaper.

“The purpose and intent of DevSecOps is to build on the mindset that everyone is responsible for security with the goal of safely distributing security decisions at speed and scale to those who hold the highest level of context without sacrificing the safety required,”

Rapid, cost-effective software delivery

When software is developed in a non-DevSecOps environment, security problems can lead to huge time delays. Fixing the code and security issues can be time-consuming and expensive. The rapid, secure delivery of DevSecOps saves time and reduces costs by minimizing the need to repeat a process to address security issues after the fact.

This becomes more efficient and cost-effective since integrated security cuts out duplicative reviews and unnecessary rebuilds, resulting in more secure code.

Improved, proactive security

DevSecOps introduces cybersecurity processes from the beginning of the development cycle. Throughout the development cycle, the code is reviewed, audited, scanned, and tested for security issues. These issues are addressed as soon as they are identified. Security problems are fixed before additional dependencies are introduced. Security issues become less expensive to fix when protective technology is identified and implemented early in the cycle.

Additionally, better collaboration between development, security, and operations teams improves an organization’s response to incidences and problems when they occur. DevSecOps practices reduce the time to patch vulnerabilities and free up security teams to focus on higher value work. These practices also ensure and simplify compliance, saving application development projects from having to be retrofitted for security.

DevSecOps Tools

1. SonarQube
2. Checkmarx
3. GitLab