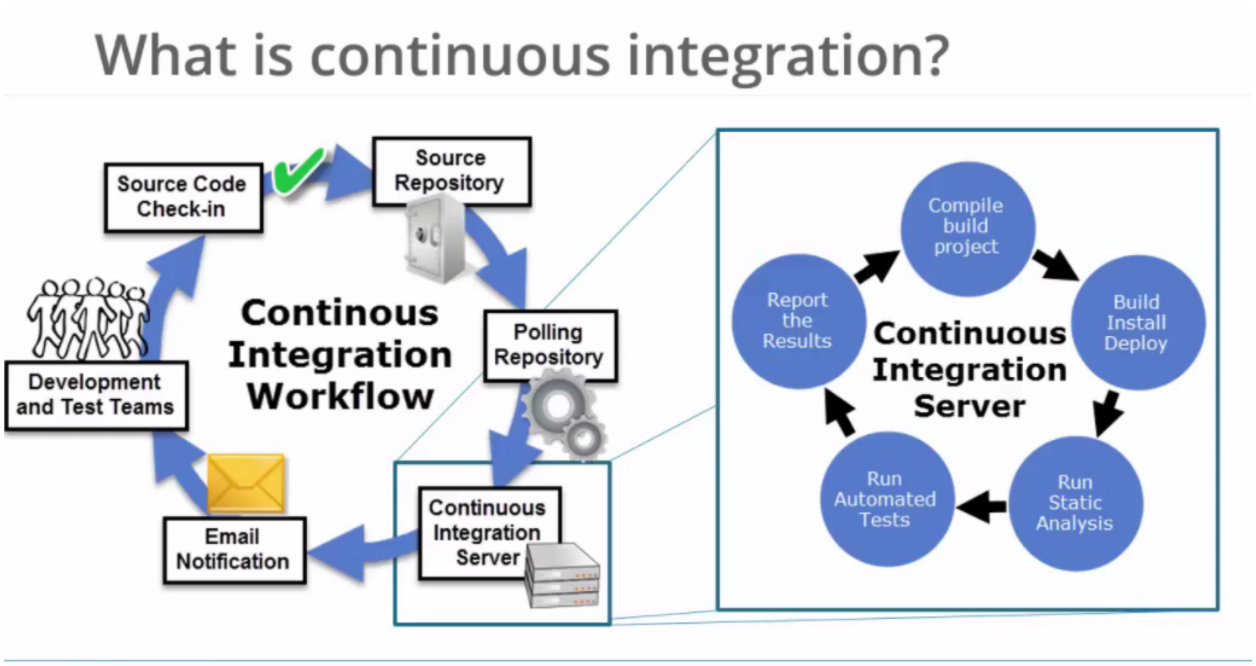
CI Vs CD

"Continuous Delivery is a small build cycle with short sprints…"



Where the aim is to *keep the code in a deployable state at any given time*. This does not mean the code or project is 100% complete, but the feature sets that are available are vetted, tested, debugged and ready to deploy, although you may not deploy at that moment.

The focus and key here is to keep the code base at a deployable state. This is the standard everyday project that goes out to the public or is consumer facing. In today's world, you cannot afford to release a bug-riddled project, so smaller sprints allow for quicker turn times to identify bugs and therefore quicker time in fixing those bugs, creating a much more stable code base early on. *This is our preferred method of working.*

Implementing continuous integration provides several benefits:

• Merge hell, that is, endless merge conflicts while merging code into the mainline, is avoided because the code should be integrated to the mainline as often as possible (at least once a day).

CI Vs CD

• Allows detecting errors quickly as automated tests run on every single commit that checks-in the remote repository. Hence, fewer bugs get shipped to production.

• When an error is detected, it’s much easier to fix because the amount of code involved in each commit is much smaller.

• Promotes human communication earlier when merge conflicts occur. People can sit together and understand how their codes are actually influencing the other’s code.

• Building the release is easy as all integration issues have been solved early.

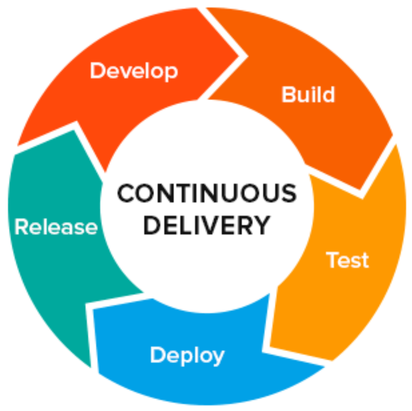
However, everything comes at a price. To implement continuous integration, the development team must understand that:

• Tasks should be split as small as possible in the planning stage because the code should be developed (including automated tests) and integrated into the mainline as often as possible, at least once a day.

• Automated tests should be present for each new feature, improvement or bug fix. They will ensure that a change (even a small one) is not breaking any other part of the system.

CI Vs CD

With **Continuous Deployment**, every change that is made is automatically deployed to production. This approach *works well in enterprise environments where you plan to use the user as the actual tester* and it can be quicker to release.

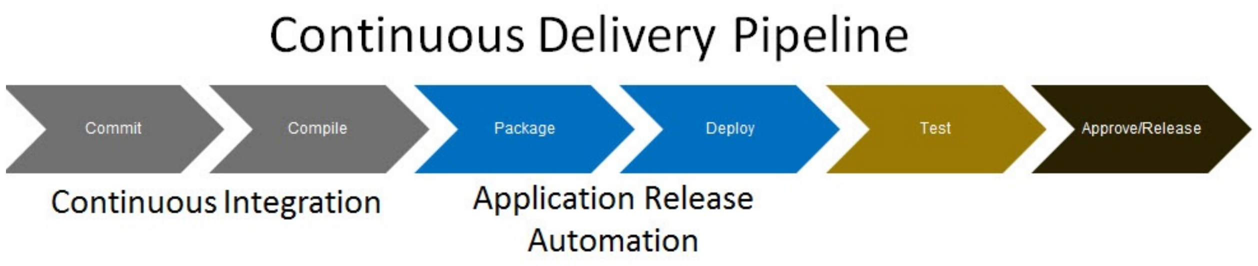


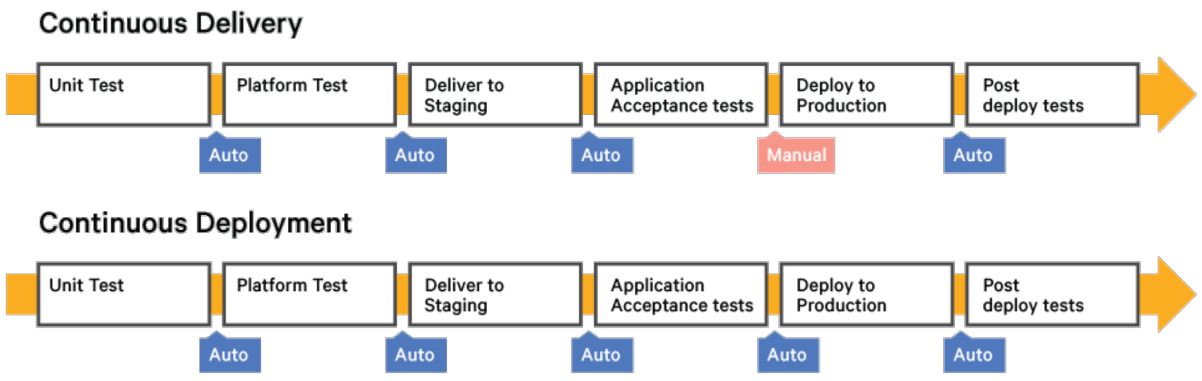
Continuous Integration is merging all code from all developers to one central branch of the repo many times a day trying to avoid conflicts in the code in the future. The concept here is to have *multiple devs on a project to keep the main branch of the repo to the most current form of the source code, so each dev can check out or pull from the latest code to avoid conflicts.*

This is critical in multiple person dev teams on a project. For example, we have a project going on now where the backend Dev team is in California and they make progress on their work, while we are completing the client side of the mobile app (in Virginia). We were stumped for two days because we could not connect to the simulator because of integration issues, which is critical for not only the back end devs to be working in parallel to our work, but all of us to be on the same exact version of the code, otherwise we could build out items that work for some, but not for the rest of the team.

Overall, each option is a concept in developing, and you need to approach your project to determine which works best for you and your project. As noted above, large companies developing internal apps would focus on the Continuous Deployment so they can get user feedback from the actual users faster, plan the next iterations, and get it out faster to the end user. Also they do not stand to lose a customer base if there are bugs.

The key is to approach each project and understand the end user to determine which approach you want to take.


Continuous Delivery Pipeline
Commit
Compile
Package
Deploy
Test
Approve/Release
Continuous Integration
Application Release Automation


Continuous Delivery
Unit Test
Platform Test
Deliver to Staging
Application Acceptance tests
Deploy to Production
Post deploy tests
Auto
Auto
Auto
Manual
Auto
Continuous Deployment
Unit Test
Platform Test
Deliver to Staging
Application Acceptance tests
Deploy to Production
Post deploy tests
Auto
Auto
Auto
Auto
Auto