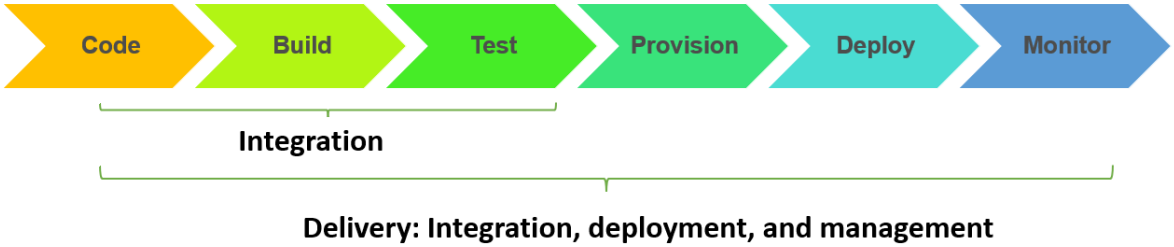
What is CI-CD with respect to the DevOps landscape?

In simple terms, CI-CD stands for Continuous Integration (CI) and Continuous Delivery (CD) in the DevOps landscape. Let's have a look at the diagram below to understand more.



In **continuous integration**: Developers commit code frequently to a code repository. Code is built frequently. Each build is tested using automated unit tests and integration tests.

In **continuous delivery**: Code changes are committed to the repository and built frequently. Builds are **deployed to test environments** and tested using automated and possibly manual tests. Builds that pass rigorous tests are **rapidly deployed to staging or production** environments.

This process of continuous integration and delivery is also called as a **CI-CD pipeline**. A robust CI-CD pipeline also automates the provisioning of infrastructure for testing and production environments. It enables monitoring and management of test and production environments.

How does CI-CD work?

In this section, let's look into more detail about how exactly CI-CD works in a software development team.

Continuous Integration:

In a typical software development team, developers follow the practice of **continuous development in an agile manner**. In continuous development, several developers work on the code. So, it is critical that they all use the latest working build for their efforts.

Code repositories maintain different versions of the code and make the code accessible to the team. You check out code from the repository, make your changes or write new code in your local copy, compile and test your code to ensure that it works, and then frequently commit your code back to the repository.

To achieve continuous integration, it is important to maintain the **integrity of the code** in the repository. This ensures that everyone who checks out the code from the repository is using the latest build and is starting with a working copy and not a broken build.

How do you maintain the integrity of the code?

You must **run the build phase every time you commit code** to the repository. Several tools in the market already automate this process by automatically **triggering a build phase after a code commit**.

All the build-phase steps (build, package, and register to package repository) are then run automatically. Every revision that is committed triggers an automated build and test.

Continuous Delivery:

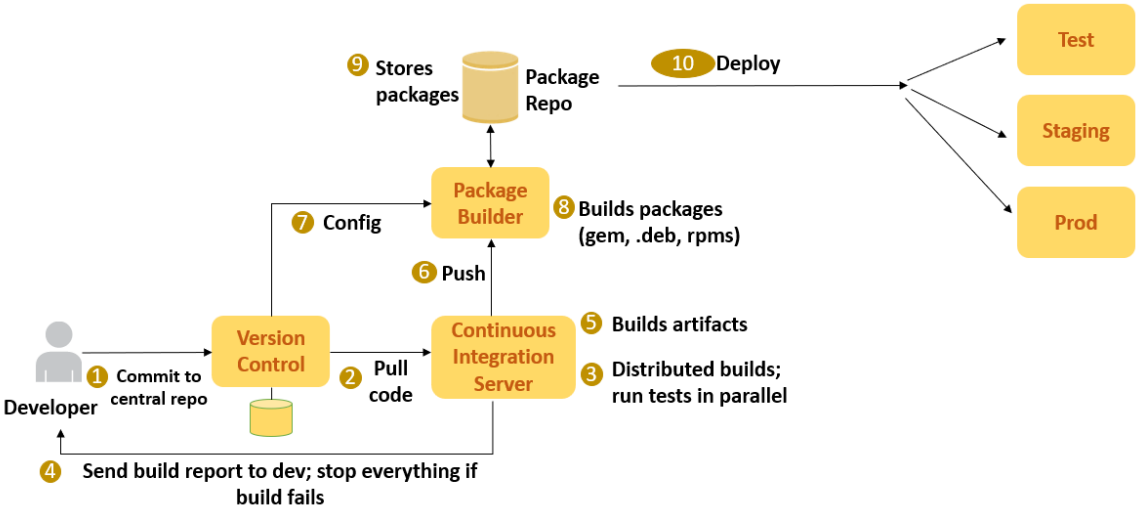
In continuous delivery:

- code changes are committed to the repository and built frequently.
- builds are **deployed to test environments** and tested using automated and possibly manual tests.
- builds that pass rigorous tests are rapidly deployed to **staging or production environments**.

The CI-CD process has **several advantages** such as:

- Improve developer productivity
- Find and address bugs quicker
- Deliver updates faster
- Get faster end-user feedback for the feature

Bringing it all together:



The workflow above shows the start-to-end process of a CI-CD pipeline. Let's go over the steps shown in the workflow:

1. Developer commits code changes to the central repo. Pre-commit hooks run sanity checks and verify that the code meets specified requirements.
2. A CI server pulls the changes from the central repo and builds the code.

3. The CI server runs all required tests against the new branch or mainline change.
4. The CI server generates a build report and sends it to the developer. The CI server also stops the build job if a failure occurs. **Note:** It is recommended to store the reports in a log somewhere that is accessible to all stakeholders.
5. If the changes pass the required tests, the CI server builds the artifacts.
6. The CI server pushes artifacts to the package builder.
7. The package builder gets configuration information from the version control system.
8. The package builder uses the configuration information and the build artifacts to build the deployment packages (gem, .deb, rpms etc.).
9. The deployment packages are stored in a repository.
10. The repo uses a post-receive hook to deploy specific packages to different environments such as test, staging, and production.