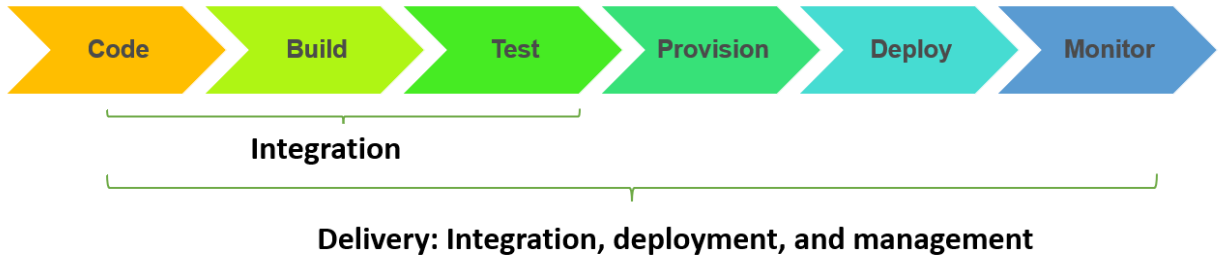
Overview

CI-CD is short for Continuous Integration (CI) and Continuous Delivery (CD) in DevOps. The diagram below explains CI-CD at a high level.



Continuous Integration

- Developers commit code frequently to a code repository.
- An automated process builds and tests the code frequently.

Continuous Delivery

- Builds are deployed to test environments and are 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:

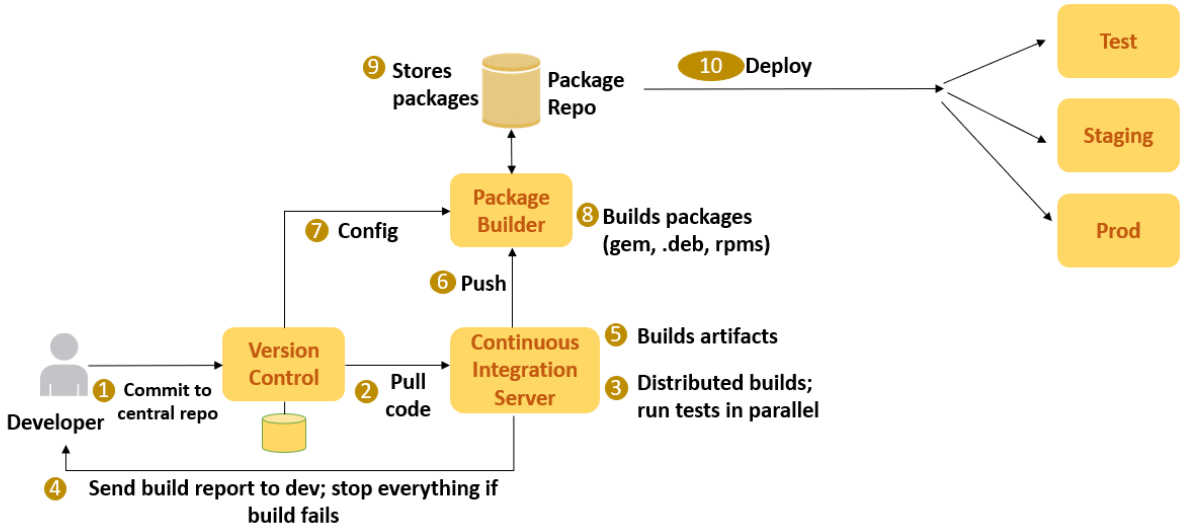
- Automates provisioning of infrastructure for testing and production environments
- Enables monitoring and management of test and production environments

Effective CI-CD

In a software development team, developers follow the practice of rapid development in an agile manner. Several developers work on the code collaboratively. So, it is critical that they all use the latest working build for their efforts. Following points ensure that you are following the CI-CD method effectively:

- Use code repositories to maintain different versions of the code and making the code accessible to the team.
- 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 the code back to the repository.
- 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. To maintain the integrity of the code:
    - Run the build phase every time you commit code to the repository. Tools such as Jenkins automate this process by triggering a build phase after a code commit.
    - All the build-phase steps (build, package, and register to a package repository) are then run automatically.
    - Every revision that is committed triggers an automated build and test.
- Deploy the builds to test environments and test using automated and manual tests
- Deploy the builds that pass the tests to staging or production environments

CI-CD workflow



The workflow shows the start-to-end process of a CI-CD pipeline. Follow the steps below to understand the workflow:

1. Developer commits code changes to the central repository. Pre-commit hooks run sanity checks and verify that the code meets the specified requirements.
2. A CI server pulls the changes from the central repository and builds the code.
3. The CI server runs all tests against the new branch or mainline change.
4. The CI server generates a build report and sends it to the developer. Alternatively, the CI server stops the build job if a failure occurs.
    - **Note**: It is recommended to save the build reports in a log folder.
5. If the code changes pass the 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 to build the deployment packages such as gem, .deb, rpms, etc.
9. The deployment packages are stored in a repository.
10. The repository uses a post-receive hook to deploy specific packages to different environments such as test, staging, and production.

CI-CD benefits

With CI-CD you can:

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