CI/CD and its Workflow

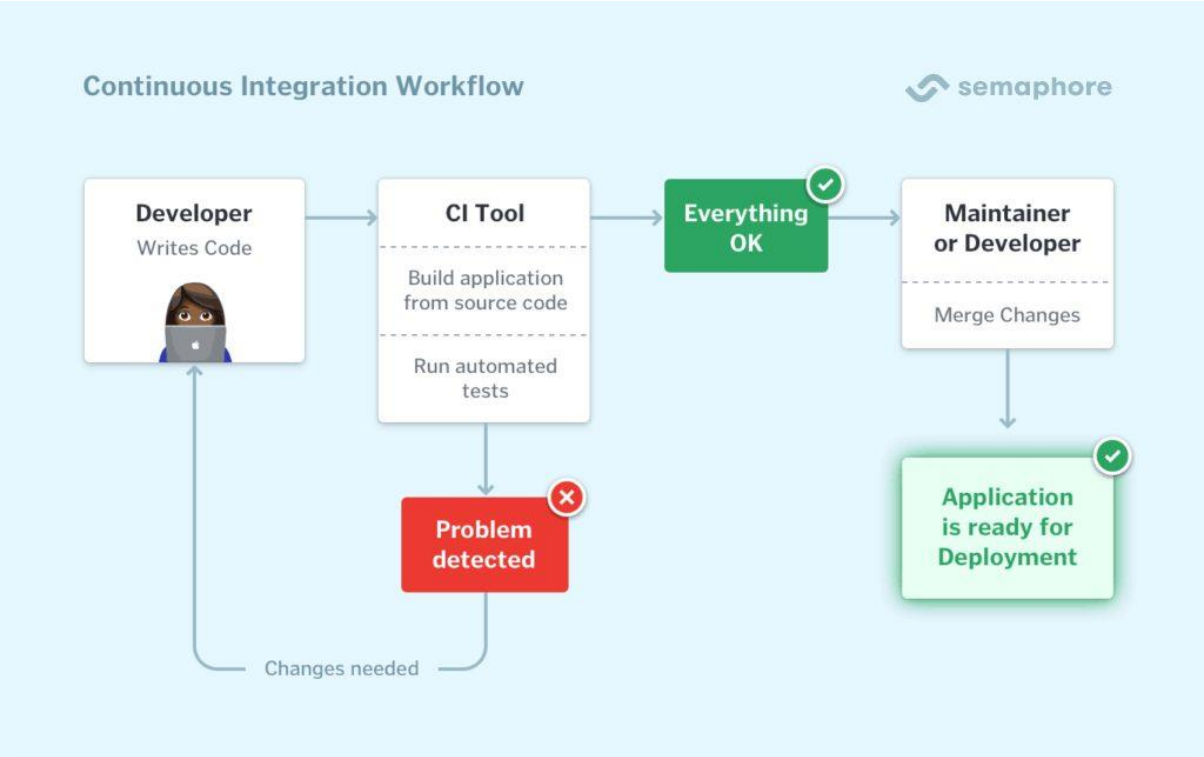
- Ashwin Harish P

CI/CD stands for

- CI (Continuous Integration) – Automates the process of code integration and testing.
- CD (Continuous Delivery/Deployment) – Automates the release of the code to production or staging environments.

A CI/CD pipeline is a series of automated steps that allow software changes to be delivered reliably and quickly from development to production.

Continues Integration



Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g. a CI or build service) and a cultural component (e.g. learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Why is Continuous Integration Needed?

In the past, developers on a team might work in isolation for an extended period of time and only merge their changes to the master branch once their work was completed. This made merging code changes difficult and time-consuming, and also resulted in bugs accumulating for a long time without correction. These factors made it harder to deliver updates to customers quickly.

How does Continuous Integration Work?

With continuous integration, developers frequently commit to a shared repository using a version control system such as Git. Prior to each commit, developers may choose to run local unit tests on their code as an extra verification layer before integrating. A continuous integration service automatically builds and runs unit tests on the new code changes to immediately surface any errors.

Continuous integration refers to the build and unit testing stages of the software release process. Every revision that is committed triggers an automated build and test.

With Continuous Delivery, code changes are automatically built, tested, and prepared for a release to production. Continuous delivery expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage.

Work Flow

1. Code Commit
   - Developers push code to a shared Git repository.

2. Trigger CI Pipeline
   - The CI tool (e.g., Jenkins, GitHub Actions) automatically triggers on every commit or pull request.

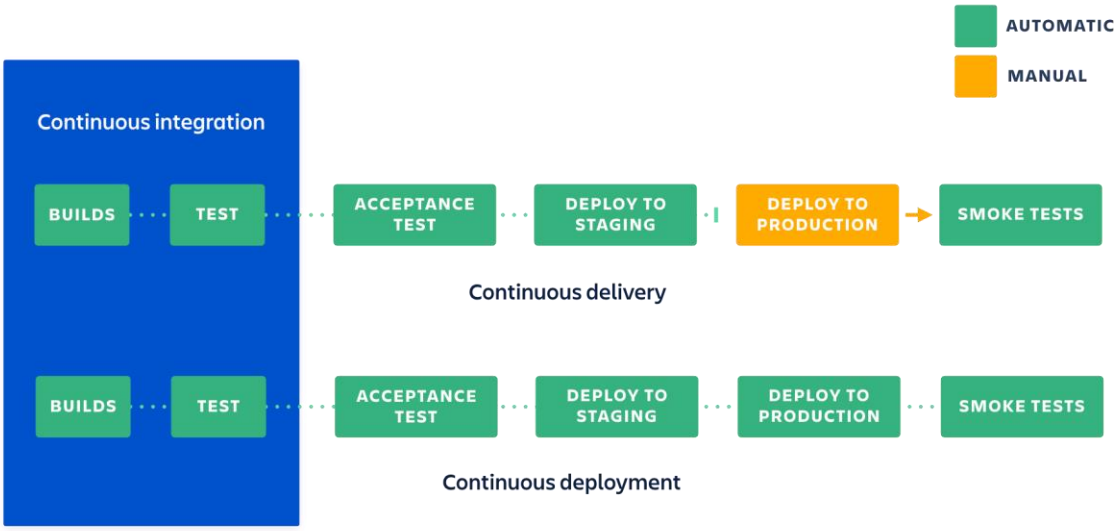
3. Checkout Source Code
   - The pipeline fetches the latest code from the repository.

4. Build the Code
   - Source code is compiled or built using tools like Maven, Gradle, or npm.

5. Run Unit Tests
   - Automated tests are executed to validate the logic and prevent regressions.

6. Generate Reports & Notifications
   - Test results and build status are recorded, and the team is notified of success or failure.

Continues Delivery



Continuous delivery is a software development practice where code changes are automatically prepared for a release to production. A pillar of modern application development, continuous delivery expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When properly implemented, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

Continuous delivery lets developers automate testing beyond just unit tests so they can verify application updates across multiple dimensions before deploying to customers. These tests may include UI testing, load testing, integration testing, API reliability testing, etc. This helps developers more thoroughly validate updates and pre-emptively discover issues. With the cloud, it is easy and cost-effective to automate the creation and replication of multiple environments for testing, which was previously difficult to do on-premises.

Why is Continuous Delivery Needed?

In the past, releasing software was a slow and risky process. After code was integrated and built, it often had to go through multiple manual testing and deployment steps before reaching production. This process introduced delays and increased the chances of human error. Testing environments were not always consistent with production, making it difficult to catch environment-specific issues early. As a result, deployments were infrequent, unpredictable, and stressful for development and operations teams.

With Continuous Delivery, the entire process from code integration to deployment preparation is automated. Code changes are automatically tested and deployed to staging environments that closely mirror production. This ensures that software is always in a release-

ready state, allowing teams to deliver updates to customers quickly, safely, and more frequently.

How Does Continues Delivery work?

With Continuous Delivery (CD), code changes are automatically built, tested, and prepared for a release to production. Continuous delivery expands upon continuous integration by deploying all code changes to a testing or staging environment after the build stage. These environments mirror production, allowing teams to perform further automated tests (such as UI tests, integration tests, and performance tests). A manual approval step is typically included before final production deployment, ensuring that the code is always in a deployable state but giving teams control over the release timing.

Work Flow

1. Code Commit
   - Developer pushes code to the repository.

2. Build & Test (CI)
   - Code is automatically built and unit tests are executed.

3. Package Artifact
   - Successful builds are packaged (e.g., .jar, Docker image) and stored.

4. Deploy to Staging
   - Artifact is deployed to a staging/test environment.

5. Acceptance Testing
   - Automated tests (UI, API, performance) run in staging.

6. Ready for Production
   - Code is in a deployable state anytime.