
CI/CD

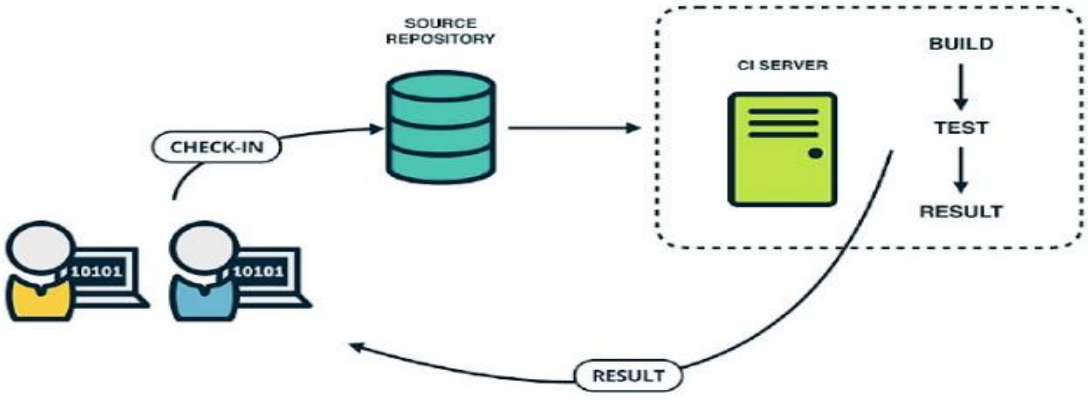
❖ CONTINUOUS INTEGRATION, CONTINUOUS DELIVERY (CI / CD):

- **CI/CD** is a method to frequently deliver apps to customers by automation into the stages of app development
- Main concepts of CI/CD are **continuous integration**, **continuous delivery**, and **continuous deployment.**
- **CI/CD** is a solution to the problems integrating new code can cause for development and operations teams.

➢ DIFFERENCE BETWEEN CI AND CD (AND OTHER CD)

CONTINUOUS INTEGRATION (CI):

- Developers regularly merge their code changes into a central repository, after which automated builds and tests are run.
- CI refers to the build or integration stage of the software release process and entails both an automation component and a cultural component.
- The key goals of CI are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.



- Developers check out code into their own workspaces. When done, commit the changes to the repository.
- CI server monitors the repository and checks out changes when they occur.
- CI server builds the system and runs unit and integration tests.
- CI server releases deployable artefacts for testing.
- CI server assigns a build label to the version of the code it just built.
- CI server informs the team of the successful build.
- If the build or tests fail, the CI server alerts the team.
- The team fixes the issue at the earliest opportunity.

CONTINUOUS DELIVERY (CD):

- CD is the automated delivery of completed code to environments like testing and development.
- CD provides an automated and consistent way for code to be delivered to these environments.

CONTINUOUS DEPLOYMENT (CD):

- CD is the next step of continuous delivery.
- Every change that passes the automated tests is automatically placed in production, resulting in many production deployments.
- In practice, continuous deployment means that a developer's change to a cloud application could go live within minutes of writing it (assuming it passes automated testing).

➢ CI/CD TOOLS:

- The popular CI / CD tools are:
  - Jenkins
  - Drone CI (CD Platform written in GO)
  - TeamCity (JetBrains)
  - Wercker
  - CircleCI
  - CodeShip
  - SemaPhoreCI


Jenkins

❖ JENKINS:

- Jenkins is an **Open-Source CI/CD tool** written in **Java.**
- It is an Automation Tool, used to **Build** and **Deliver** the Software Product.
- Jenkins was forked from Another Project called **Hudson,** after dispute with Oracle.
- It is a server-based application and requires a web server like **Apache Tomcat.**
- Jenkins monitoring of repeated tasks which arise during the development of a project.

  E.g.: Your team is developing a project; Jenkins will continuously test your project builds and show you the errors in early stages of your development.

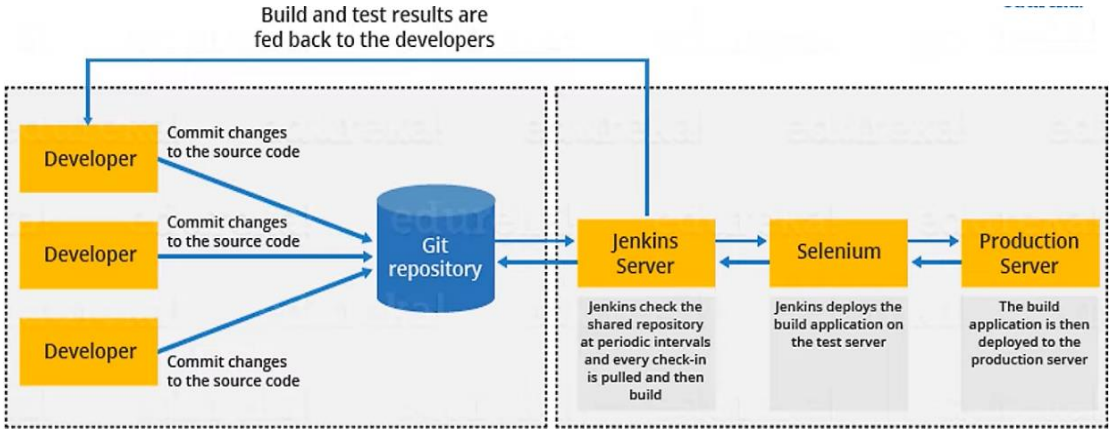
WHY USE CONTINUOUS INTEGRATION WITH JENKINS:

- Code is built and test as soon as Developer commits code.
- Jenkins will build and test code many times during the day.
- On Successful build, Jenkins will deploy the source into test server and notifies the deployment team.
- On Build Failures, Jenkins will notify the errors to the developer team.
- Code is built immediately after any of the Developer commits.
- Automated build and test process saving timing and reducing defects.

➢ ADVANTAGE OF USING JENKINS:

- Jenkins is being managed by the community which is very open.
- Jenkins has around 320 plugins published in its plugins database.
- It supports cloud-based architecture so that you can deploy Jenkins in cloud-based platforms.
- It Support Docker Containers, you can containerize Jenkins Service.

➢ JENKINS ARCHITECTURE:



- This single Jenkins server was not enough to meet certain requirements like:
  - Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
  - If larger and heavier projects get built on a regular basis then a single Jenkins server cannot simply handle the entire load.
  - A standalone Jenkins instance can grow fairly quickly into a disk-munching, CPU-eating monster.
- To prevent this from happening, we can scale Jenkins by implementing a slave node architecture, which can help us offload some of the responsibilities of the master Jenkins instance.

JENKINS SERVER:

- Jenkins checks the shared repository at periodic intervals and every check-in is pilled and then build.

SELENIUM:

- Jenkins deploys the build application on the Test Server.

PRODUCTION SERVER:

- The Build application is the deployed to the Production server.