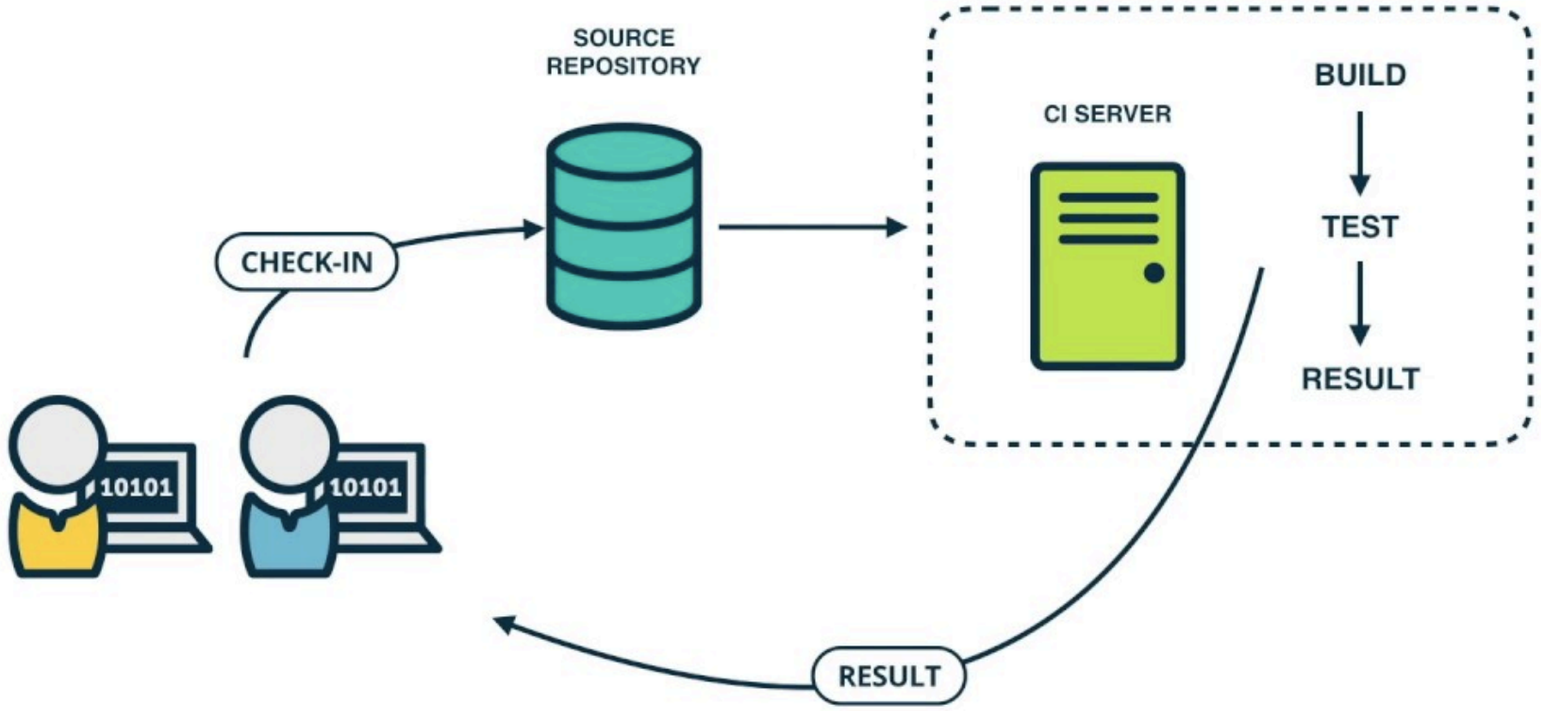
Jenkins

Jenkins: Continuous Integration

- What is Continuous Integration?
- Why we need Continuous Integration?
- Phases of Adopting Continuous Integration.

JENKINS : Introduction



- **Continuous Integration** (CI) is a development practice that requires developers to integrate code into a **shared repository** several times a day.
- Each check-in is then verified by an **automated build**, allowing teams to detect problems early.
- If Build is not Green, system **notify Developer** immediately. By this, developer can detect errors quickly, and locate them more easily.

Why do we need Continuous Integration

- Significantly **less back-tracking** to discover where things went wrong.
- Continuous Integration is cheap. If you don't follow a continuous approach, you'll have **longer periods between integrations**. This makes it exponentially more difficult to **find and fix problems**.
- Say goodbye to long and tense integrations.
- Increase visibility enabling greater communication.
- Catch issues early and nip them in the bud.

Why do we need Continuous Integration

- Spend less time **debugging** and more time **adding features.**
- Stop waiting to find out if your code's going to work.
- **Reduce integration problems** allowing you to deliver software more rapidly.
- **Continuous Integration** doesn't get rid of bugs, but it does make them dramatically easier to find and remove.

Stage of Adopting Continuous Integration

- Continuous Integration is backed by several important principles and practices:

The Practice

- Maintain a single source repository.
- Automate the build.
- Make your build self-testing.
- Make it easy for anyone to get the latest executable version.
- Everyone can see what's happening.
- Automate deployment.

How to Do it

- 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.

Teams Responsibility

- Check in frequently.
- Don't check in **broken code**.
- Don't check in **untested code**.
- **Don't check** in when the **build is broken**.

Continuous Integration

- The Practice of **merging stable Develop work** branch with the **main branch** constantly.

Continuous Delivery

- Continual Delivery of Code to an environment once the code is ready to ship.
- Environment could be staging or production. First product is deliver to QAs and Review before shipping to Customer/ Production.

Continuous Deployment

- Essentially, it is the **practice of releasing every good build** to users.
- The deployment of Product in Production as soon as it's ready.

- By adopting both Continuous Integration and Continuous Deployment, you not only reduce risks and catch bugs quickly, but also move rapidly to working software.

Will see you in Next Lecture…

See you in next lecture …