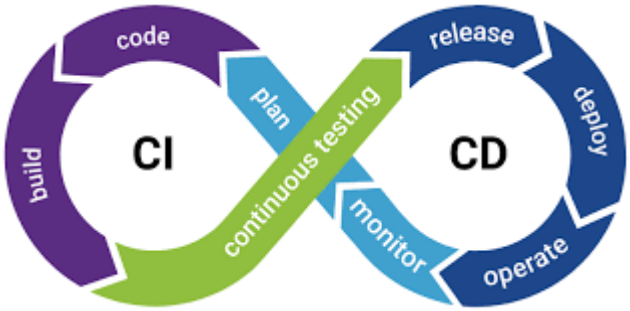
Continous integration / Continous delivery/deployment



- **CI/CD** is one of the best practices for devops teams to implement. It is also an agile methodology best practice, as it enables software development teams to focus on meeting business requirements, code quality, and security because deployment steps are automated.

- *Continuous integration* is a coding philosophy and set of practices that drive development teams to implement small changes and check in code to version control repositories frequently. Because most modern applications require developing code in different platforms and tools, the team needs a mechanism to integrate and validate its changes.

- The technical goal of CI is to establish a consistent and automated way to build, package, and test applications. With consistency in the integration process in place, teams are more likely to commit code changes more frequently, which leads to better collaboration and software quality.

- Continuous integration and continuous delivery require *continuous testing* because the objective is to deliver quality applications and code to users. Continuous testing is often implemented as a set of automated regression, performance, and other tests that are executed in the CI/CD pipeline.

How continuous integration improves collaboration and quality

- Continuous integration is a development philosophy backed by process mechanics and some automation. When practicing CI, developers commit their code into the version control repository frequently and most teams have a minimal standard of committing code at least daily. The rationale behind this is that it's easier to identify defects and other software quality issues on smaller code differentials rather than larger ones developed over extensive period of times. In addition, when developers work on shorter commit cycles, it is less likely for multiple developers to be editing the same code and requiring a merge when committing

Tools for CI

CI is mainly a cultural shift, but some tools could help you to get the job done quickly. And you can even start on a dollar a day, according to James Shore. All you need is an old computer, a rubber chicken (really!), and a desk bell. Shore's post is hilarious, and I recommend you read it. He makes a valid point clear: lack of tools isn't an excuse. You can do CI without them.

Nonetheless, tools can help. Here's a list of common tools that you can start using today.

- Jenkins—a free, open-source, Java-based tool that gives you a lot of flexibility.
- Azure Pipelines—a Microsoft product free for up to five users and open-source projects.
- Cloud Build—the managed service offering from Google Cloud Platform.
- Travis CI—a popular tool for GitHub open-source projects that offers a hosted or self-hosted solution.
- GitLab CI—a free tool from GitLab that can also integrate with other tools via the API.
- CircleCI—a tool that's popular for GitHub projects and has a hosted and self-hosted solution. You can start for free.
- CodeShip—a self-hosted-only solution. You can start with the free version, but it's a paid tool.

- Continuous deployment is a strategy for software releases wherein any code commit that passes the automated testing phase is automatically released into the production environment, making changes that are visible to the software's users.

What is continuous testing?

Testing is one process that can and should be automated to a great extent within the CI/CD pipeline.

Why? Because automated testing, which enables so-called continuous testing, ensures that bugs get found early and are fixed before the end-users experience any breaks. The idea is to 'fail fast' because it's much easier to fix a bug when it's found early, as the bug typically hasn't traveled so far or changed hands, and it's also a lot less costly.

Some types of tests that can be automated include regression tests, functional tests, integration tests, and performance tests.

At the core, continuous testing is about three things:

- testing at earlier stages of the release pipeline
- testing more often before release
- testing everywhere, that is, across environments and devices

Additional benefits of pipelines

Having a CI/CD pipeline has more positive effects than only making an existing process a little more efficient:

- Developers can stay focused on writing code and monitoring the behavior of the system in production.
- QA and product stakeholders have easy access to the latest, or any, version of the system.
- Product updates are not stressful.
- Logs of all code changes, tests and deployments are available for inspection at any time.
- Rolling back to a previous version in the event of a problem is a routine push-button action.
- A fast feedback loop helps build an organizational culture of learning and responsibility.
- Can save a lot of money for business.

Thank you.