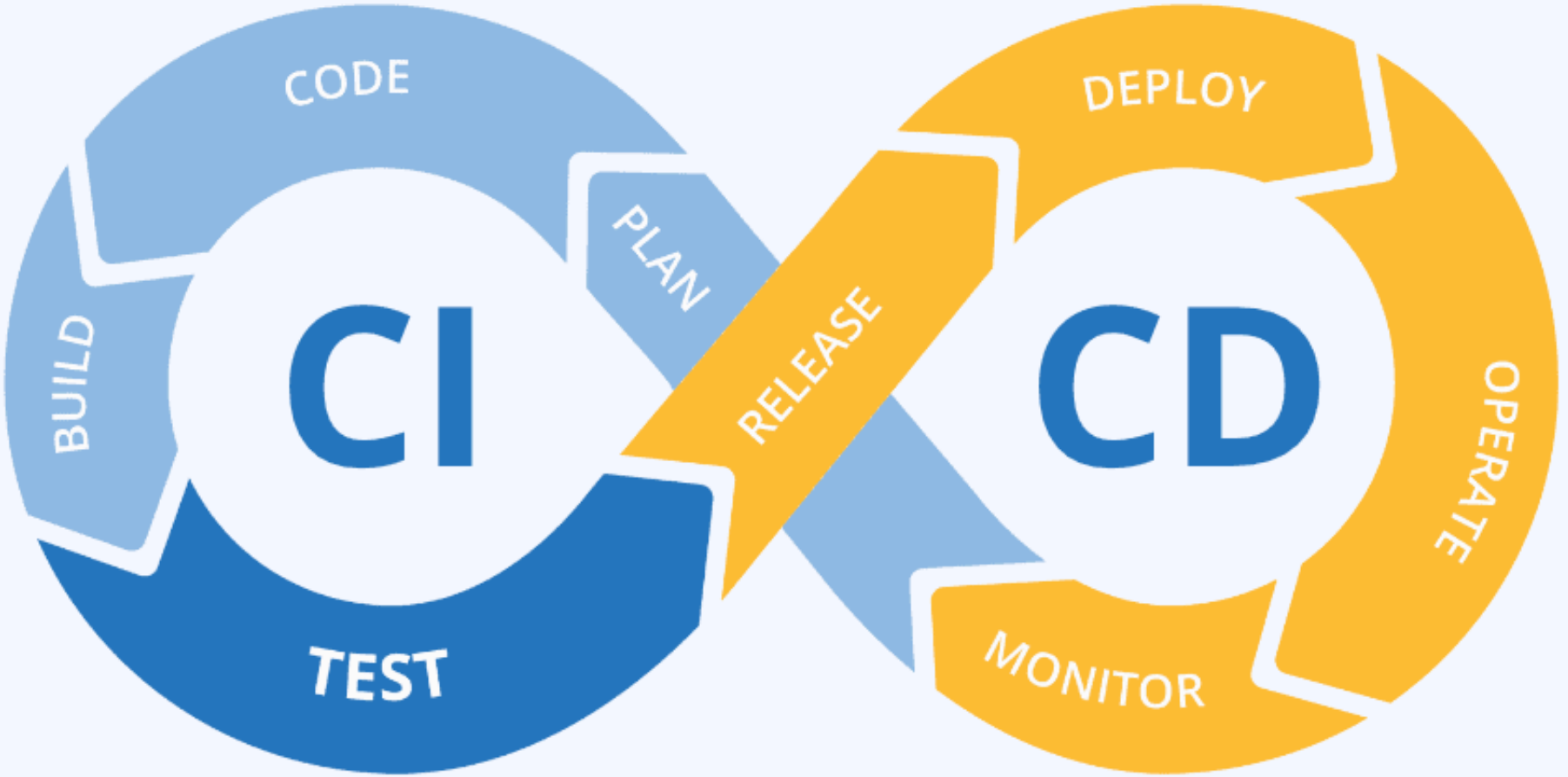
Esoon Ko/ Erik Bremstedt
IT Högskolan

CI/CD


CODE
PLAN
BUILD
CI
TEST
DEPLOY
RELEASE
CD
OPERATE
MONITOR

CI : Continuous Integration

CI is the practice of frequently integrating code changes into a shared repository, where each integration is automatically tested to detect errors early.

The primary goal of CI is to detect and address issues quickly by making integration a regular part of the development process. This allows teams to identify and resolve conflicts between different developers' code early in the development cycle.

- **Regular code commits**
- **Automated testing of each integration**
- **Building the code after commits to verify that the codebase remains functional**

CD : Continuous Delivery

Continuous Delivery extends CI by ensuring that code changes can be automatically deployed to a production-like environment after passing all tests.

The main goal of Continuous Delivery is to ensure that the software can be released to production at any time. The process should be so streamlined that deploying to production is a non-event.

- **Automated deployment of code to staging environments**
- **Extensive automated tests**
- **Manual approval before deploying to production**

Continuous Deployment

Continuous Deployment is a step further than Continuous Delivery. Every code change that passes the automated testing phase is automatically deployed to production without any manual intervention.

The goal is to deploy new features or updates as soon as they are ready, ensuring that the product continuously improves.

- **Fully automated deployment pipeline that goes from code commit to production**
- **Emphasis on robust testing and monitoring to catch and resolve issues post-deployment**

Why CI/CD?

Faster Time to Market: CI/CD allows for quicker release cycles, enabling organizations to bring features and updates to users faster.

Higher Quality and Reliability: Frequent testing and integration reduce the likelihood of bugs and integration issues, resulting in more stable and reliable software.

Improved Collaboration: By automating the integration and testing processes, CI/CD fosters a culture of collaboration where developers are encouraged to share their work more frequently.

Reduction of Manual Errors: Automation reduces the potential for human error in testing and deployment processes, increasing consistency and reliability.

Challenges

Data Dependencies: Data pipelines often depend on specific data sources, making it challenging to test in isolation. CI/CD processes need to account for these dependencies.

Tooling and Infrastructure: Implementing CI/CD for data engineering requires robust tooling and infrastructure, which may not be in place in all organizations.

Cultural Shift: Moving to CI/CD requires a shift in mindset from infrequent, manual deployments to continuous, automated processes. Teams need to embrace this cultural change for CI/CD to be effective.

How does it look in Data Engineering?

Code Repositories:
- Just like in software development, all the scripts, configuration files, and data transformations in data engineering should be stored in a version-controlled repository like Git.

Automated Testing:
- Testing data transformations (e.g., ensuring a transformation script correctly normalizes data).
- Testing data pipelines (e.g., ensuring data flows from source to destination without errors).

Data Pipeline Deployment:
- Automating the deployment of ETL (Extract, Transform, Load) jobs.
- Managing the deployment of data processing tasks on platforms like Apache Spark or Hadoop.

Monitoring and Logging:
- Setting up monitoring and alerting to detect issues in data pipelines.
- Logging pipeline activities to audit and troubleshoot issues.