CI/CD

Continuous delivery continuous integration

Summary

- Overview of CI/CD
- Why CI/CD important?
- CI/CD for microservice
- CD and Facebook/OANDA case studies
- CI and Uber Submit Queue pipeline
- Limitation and Future Work

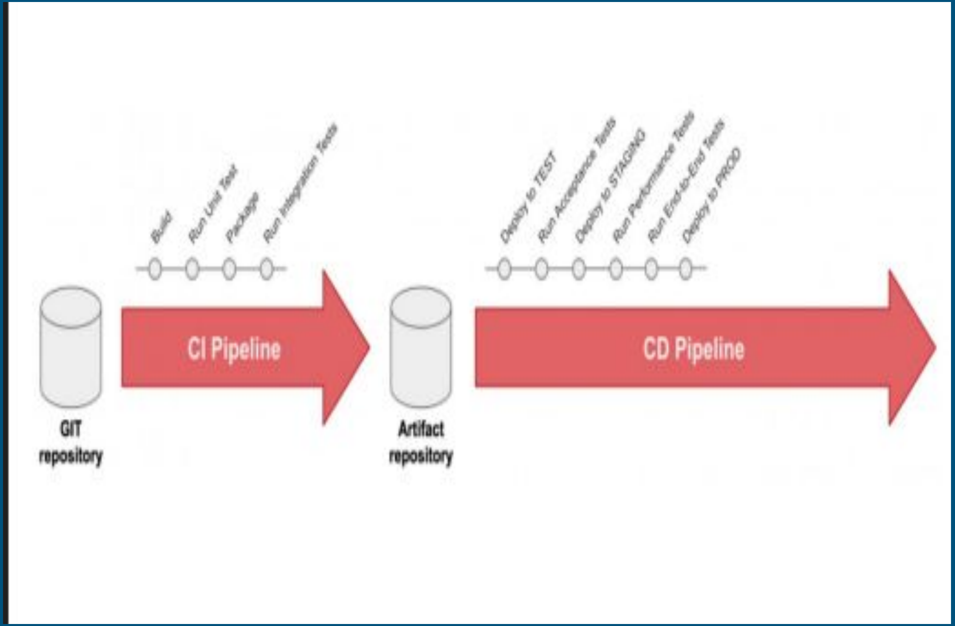
Overview of CI/CD

Continuous Integration - regularly build, test, and merge code changes into main branch.

Continuous Deployment - automatically test and release changes from the repo to production.

CI is generally a standard across all software projects.

CD on the other hand is not. Example: Aerospace Industries, Healthcare, etc.



Why is CI/CD useful?

- Release software with less risks
  - CI/CD takes care of the automated testing, deployment and rollbacks.
- Can improve developer productivity.
  - More automation, less context switching, etc
- Ship features and fix bugs faster.
- Ability to push out small changes and iterate on them continually.

CI/CD for microservices

- Requirements
  - Highly cohesive and loosely coupled services and teams ○ Teams and engineers that can make key-decisions independently
- Challenges:
  - Tools to support deployment and experience in managing them.
  - Senior Management buy-in.
  - Investment on the tools

Despite the challenges, a lot of companies use CI/CD to manage their release.

CI/CD in practice

Steps involved:

- Code Review
- Testing
    - Unit tests, integration tests, performance tests, etc
- Release engineering
    - Assess the risk and manage the deployment
- Deployment

Continuous Deployment

- Blue-green deployment
    - Deploy to a small fraction of people and dial it up

- Dark launches
    - Launch during non
    -peak hours

- Staging
    - Test the builds in multiple staging environments
    - Simulate some traffic

Transition to CI/CD

- Automated Testing infrastructure
    - Unit tests, Integration tests, Shadow tests, performance tests, etc

- Deployment Management System
    - Code reviews, VCS, deployment scheduling, staging pipelines, Rollbacks

CD Case Studies At Facebook and OANDA

- OANDA
  - Currency trading system that manages trades worth many billions every day
  - Small team with about 100 engineers
  - Code check-in -> Deployment Pipelines -> ad-hoc alert system using emails
- Facebook
  - Billions of queries per second
  - 1000s of engineers
  - Code check-in -> Release engineering -> Error reporting (SEV)

Goal: Using CI to Keep the Master Branch Green

- Instantly release new features from any commit point in the mainline
- Can roll back to any previously committed change
- Improve engineer’s productivity

Idea behind:

- Speculatively build possible outcomes for pending changes.
- Try to minimize the number of builds and parallelly build independent changes.

Limitation and Future Work

- No order for non-independent changes: Possible starvation for small changes
- Possible to implement a better machine learning method to train the prediction model
- Should abort a build which is near its completion but likelihood of success drops?
- Batch independent changes