CI/CD PIPELINE

CONTINUOUS INTEGRATION/CONTINUOUS DELIVERY (CI/CD)

INTRODUCTION

Traditional software development has evolved from a waterfall model to a more agile process that follows DevOps principles. In the traditional approach, writing code, compiling, testing packaging, QA, and release into production were done in silos. However, there is no adequate automation to test and identify for bugs quickly,which impacts software quality. This leads to higher technical debt and longer lead times for application deployment due to infrastructure provisioning. Disjointed and prolonged testing is inconsistent and delays the software-release process.

Continuous Integration and Continuous Delivery (CI/CD) is one of the most important and desired processes for modern code development and delivery for new and transforming software development environments. CI/CD provides shorter compilation or build times and faster release cycles with more automation and testing. Adopting CI/CD processes can drive innovation and faster time to market forcommercial and enterprise applications.

CONTINUOUS EVERYTHING

The continuous workflow in the DevOps process is an automated and iterative

process from development to delivery in the software development life cycle (SDLC). Various development, delivery, and platform tools and processes are used during workflow automation. The continuous process shortens the lead time for application development and deployment while accelerating the release cadence.

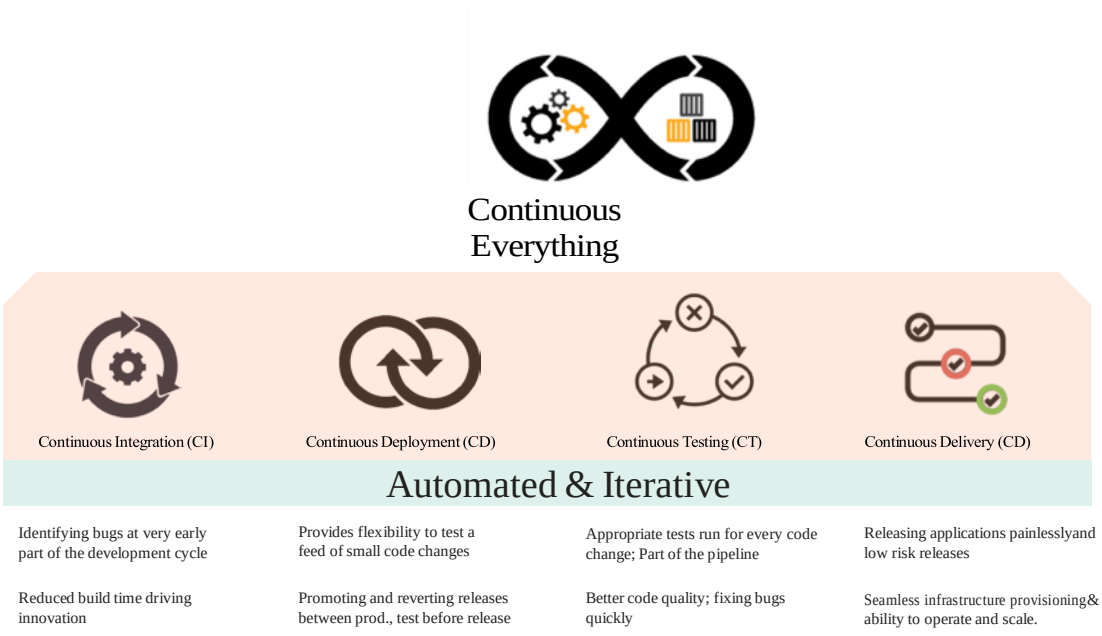


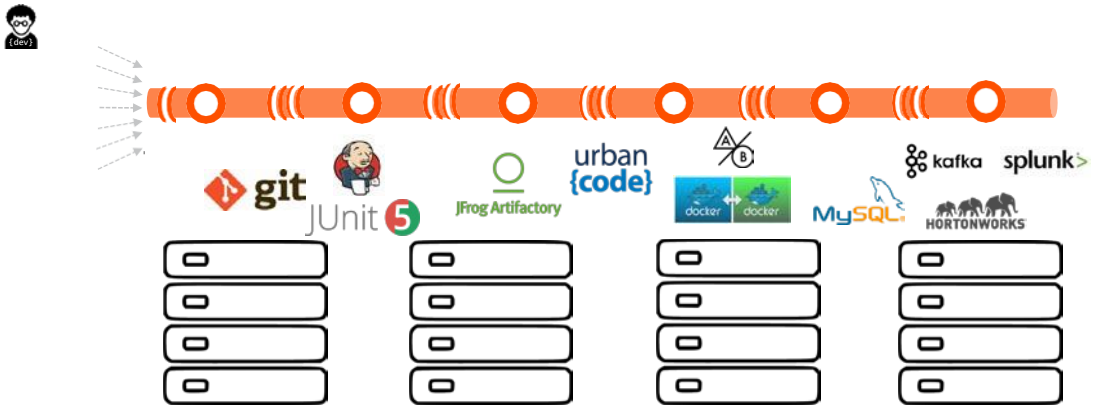
FIGURE 1. Continuous process from development to deliveey

- **Continuous Integration (CI)** – During this process, developers identify bugs at early stages of the development cycle, fix them, and test in an iterative manner. Every time there is a new code change or a bug fix, the build or code compilation process takes place in the developer's private workspace. The developer then integrates the changes into the main code base. Depending on the size of the development team, these multiple builds could be running in parallel. Shorter build times lead to developer creativity that can breed innovation.

- **Continuous Deployment (CD)** – After the build process and packaging in the CI phase, the final build package is automatically deployed for user acceptance testing before it is released to production. More and more modern applications are running as microservices and containers as the unit of deployment on a platform. The platforms could be any public-cloud environment or containers for platform-as-a-service like Red Hat OpenShift or Pivotal Cloud Foundry. As containers are portable, *platform* is synonymous with
  standard operating system like Red Hat, Ubuntu, and others, abstracting the cores and memory required atapplication run time.
- **Continuous Testing (CT)** – Testing has recently started moving closer to the developers, in a move also known as "Shift Left." Using more automation to run various different tests appropriate to code changes is now part of the pipeline. This means that one simple change does not have to run the entire test suite.
  Instead, it focuses only on the relevant tests required for the change. Iterative testing and higher degrees of test success warrant much better code quality.
- **Continuous Delivery (CD)** – Continuous delivery differs slightly from continuous deployment. Both are continuous and automated, but the application is not released into production immediately with CD. There isa gate-keeping process where the latest build package goes through a test-automation process to validate compatibility and interoperability before it's released to production. Automating the process of promoting a new build into production and reverting back to an old build version is less painful and lowers risk.

CHALLENGES WITH THE TRADITIONAL CI/CD PROCESS

Data is generated during the entire CI/CD process. This data growth depends on the size of the development team(s), code base, number of builds/day, and deployments. The platform abstracts the infrastructure to be consumed as code. Data has to be stored, protected, managed, and reused during the entire SDLC. But data has gravity. Most of the phases in the CI/CD process are designed to function in silos.



A typical SDLC pipeline consists of six stages – plan/code, build/test, packaging, deploy, stage, and release into production.

Step 1: Code – Developers perform "git clone and git push" operations to and from their workspaces in their laptops to write and update code. There is a performance impact on the Git server, where there can be hundreds and thousands of developers checking out code simultaneously. There is also a security concern about having the proprietary code moving around on individual user laptops outside the business premises.

Step 2: Build/Test – While making code changes and updates in the workspaces, developers are supposed to run private builds and tests before merging the code to main or development branches. Once the respective branch is updated with the code changes, integrated builds are triggered along a prescribed set of CI tests to check the validity of the code. These builds and tests run in parallel, which requires higher throughput and concurrency with lower job failure objectives.

- Scalability and concurrency are a limitation with local storage.
- Service-level objective: bandwidth, IOPs, and latency are not properly met.

Step 3: Binary Packet Manager – Successful builds are then packaged with the right dependencies and promoted to the next CD phase. You must replicate or move the data from the repository that has all the binary packages to stage it, then release it into production. As the size of the business grows, many versions and new package types are generated and stored. Additional servers to support the large dataset size results in server sprawl with high manageability overhead.

Step 4: Deploy and Stage – During the CD process, applications are tested with corresponding databases to perform functional tests for new features. It's not trivial to get a copy of the production database or set up a new one for testing purposes. It is time-consuming and involves more risk to get a copy of the production database. Obfuscation of certain parts of the data in the database limits developer innovation.

Step 5: Production – The following questions arise when the application is ready to be released into production:

- How do I provision the underlying infrastructure on-premises or move to a public cloud to release it?
- How challenging will it be to operate, manage, and monitor the application and the correspondingdata ingestion?