WHAT IS CI/CD? INTRO TO CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY

Continuous Integration (CI) and Continuous Delivery (CD) are processes within the DevOps SDLC methodology designed to yield fast and robust software development. Both processes follow the same direction within the SDLC pipeline but end at different intervals.

Let's take a look.

What is Continuous Integration?

Continuous Integration refers to the process of merging all coding works of a software development project on a continuous basis. For instance, committing all code changes to a centralized repository can be considered as a simplified version of CI. The concept is further enhanced using automation tools, processes, and culture that drive rapid integration of iterative code developments.

- The build is therefore available at a single accessible machine location for further testing.
- The cultural component is an integral part of a CI strategy. Collaborating, communicating, and learning how to perform and merge small code changes faster requires a cultural shift at an individual and collective level within the organization.

CI strategies encourage small and frequent code commits that can be integrated faster, without breaking the resulting software functionality. The build servers run automated tests on the merged code to identify bugs early during the SDLC pipeline, validate and deliver new application changes to end users.

What is Continuous Delivery?

Continuous Delivery extends CI to incorporate automated software release within the SDLC pipeline. The builds with continuously integrated code changes are automatically released to production after initial testing such as automated unit tests. At the production stage, the software build is available for in-depth testing and therefore ready for production, although a release may require further manual approval for business or technical reasons.

If the release process is also automated, the process is called Continuous Deployment.

Why CI/CD?

In a world where every company is a technology company, improvements and changes in software reaching end-users deliver that vital competitive edge through innovation.

In order to increase the rate of innovation, organizations must ensure that the business and technical challenges in releasing software improvements are mitigated. A close integration between the development, testing, operations roles as well as key business decision makers is critical to meet these goals. While in the past IT has traditionally been used merely to keep the lights on, progressive organizations are driving unprecedented business opportunities by delivering custom application features, services, and innovations that end-users demand. At the same time, organizations recognize their inability to deliver software-enabled business services at rapid pace and low risks using traditional SDLC methodologies.

For organizations following the DevOps approach, corporate culture is shifting their IT philosophy from teams that follow stringent orders into collaborators that contribute to business development through the delivery of innovative software improvements. More business leaders and decision makers intend to invest resources in rapid, incremental innovation, and require IT shops to respond to market disruptions accordingly.

The CI/CD strategy automates the process of innovation through fast and efficient software release process. Secure and functional software updates are ensured through automated build and testing. The development, testing, and operations teams work collectively to implement productive workflows within the SDLC pipeline. IT shops are freed from manual tasks on solving complex bug fixes and resolving code dependencies that appear only too late into the software delivery process. Any code change that introduces a bug is identified immediately and developers can improve on the small iterative code changes accordingly. As a result, the correct, functional, secure and improved software updates are delivered to end-users at a higher velocity.

Organizations can respond to market changes, cybersecurity issues or business circumstances effectively. Unlike traditional SDLC methodologies that focus on delivering software updates to end-users in the matter of weeks or months, CI/CD strategies aim to deliver working updates in a matter of hours or days.

CI/CD Best Practices

In order to achieve these goals, an effective CI/CD strategy can include the following best practices:

Operate Infrastructure as Code

An effective CI/CD requires the infrastructure to be adaptable and consistent with the production environment while preserving the integrity of configurations as resources are provisioned dynamically and automatically. This is known as infrastructure as code (IaC). Any configuration drift will impact the repeatability of the testing and deployment process, and therefore prevent true continuity within the SDLC pipeline.

Maximize version control

In order to ensure that every change to the software build is meaningful and successful, a Version Control System (VCS) can be used to track the changes and revert to earlier deployments as necessary.

An automated CI process can be achieved by triggering software integration and testing processes as the VCS is updated with a new code commit. The changes can be documented accordingly to maintain a single version of truth as the build progresses through the development phase. Additionally, it is beneficial to limit the branching in VCS to reduce the possibility of a branch not being tracked for code updates and testing.

Maintain a consistent deployment process

A cultural as well as tooling change may be necessary to ensure that developers adhere to a standardized process for code commits. The build process should also be consistent throughout the pipeline.

For instance, build unique binary artifacts and reuse the result throughout the SDLC pipeline. When the software is not packaged multiple times in multiple different versions simultaneously between disparate teams, no inconsistency will be injected into the final software product delivered to end-users.

Testing

Perform small and faster testing early during the SDLC pipeline to identify problematic changes before its too late. This means that the SDLC teams must prioritize testing, usually in sequence:

1. Unit tests
2. Integration tests
3. System tests
4. Acceptance tests

Developers may run some tests locally before applying code changes and therefore detected issues before the code is integrated within the centralized repository. Run the tests in containers to standardize the test environment and enhance portability of the testing infrastructure.

Security

Take the necessary measures to ensure optimum security of the CI/CD infrastructure, especially since the pipeline contains valuable data and access to deploy code changes to a centralized repository. Depending on your risk, considering using:

- Advanced identity and access management capabilities
- VPNs for access
- Multiple layers of security

The changes to succeed at CI/CD

As a key component of the DevOps strategy, CI/CD comprises of a tooling, cultural, and process change from traditional software development and delivery methodologies.

The definition and scope of these changes may vary between organizations, their DevOps maturity level and how they choose to practice CI/CD. In most cases, organizations will need to optimize their CI/CD approach to drive business value in response to the evolving technical and business circumstances.

Additional resources

For more on this topic, explore these resources:

- BMC DevOps Blog
- DevOps Guide, with 25+ articles (use the right-hand navigation menu)
- Continuous Delivery vs Deployment vs Integration: What's the Difference?