DIFFERENCES BETWEEN CONTINUOUS INTEGRATION (CI), DELIVERY (CD), AND DEPLOYMENT

Continuous Integration (CI), Continuous Delivery (CD) and Continuous Deployment are important components of the DevOps SDLC methodology. The three SDLC practices are used at various stages of the development cycle with the overall purpose to streamline product release to the end-users while reducing waste processes. The practices require a new perspective on software development, an enhanced focus on automation and a culture of collaboration that would help achieve these goals.

Before describing the new development paradigm, consider a typical case scenario of the traditional approach to software development. Several developers work independently to build different features of a software product. These development branches take several weeks or months to complete. The builds are not merged or tested to work together until the end of the development cycle. On day however, the project manager asks the team to expedite the release process, forcing the developers to merge their feature branches in production. The integration of multiple branches produces new bugs and issues that were previously never identified or tested for. Regardless, the merged branch is passed on to the QA team that must conduct manual and automated testing with extra efforts to resolve the new issues.

Product development is hit with delays across all phases of the SDLC and leads to unhappy developers who must scramble to integrate incomplete feature branches, unhappy testers who must fix new and previously overlooked software issues faster, and unhappy project managers facing unplanned delays over the feature release.

In a previous BMC post, we described in detail the description and the purpose of CI/CD. In this post,

we will further explore the differences between Continuous Integration, Continuous Delivery and Continuous Deployment.

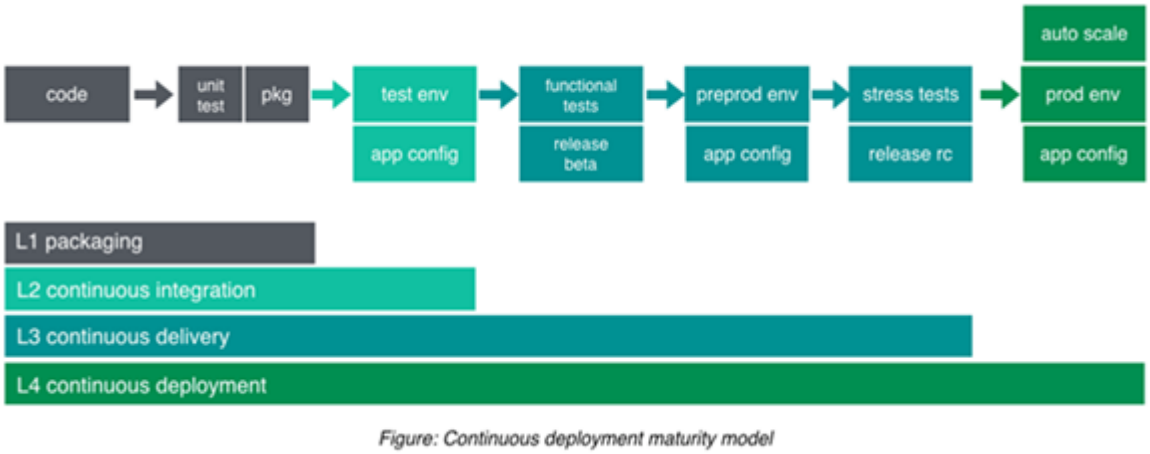

Figure: Continuous deployment maturity model

Source: Testcollab

Continuous Integration is the practice of frequently integrating code changes in a centralized shared repository and validating the build through automated tests. Any issue arising from the integration is fixed at this stage to ensure a running build comprising of all packaged changes on a continuous basis. With this practice, developers avoid "integration hell". Any issue arising from the code integration is identified early during the SDLC. The developers are both individually and collectively responsible to making integrated branches work together from the very beginning.

Differentiating attributes of Continuous Integration include:

- Developers write the code, perform unit tests and commit the package to a centralized server repository.
- The focus of CI is not so much to complete a feature branch at this stage, but to ensure that all software changes integrated and working without issues as a packaged build.
- The code changes are merged frequently, possibly multiple times every day.
- Automation tools and server resources are reserved to ensure consistent build and test environment. This approach eliminates the possibility of "it worked on my machine" excuse and facilitates collaboration between developers working toward different feature branches.
- Feedback on the integrated build performance is identified early and developers gain early context into potential integration issues. The testing takes place at both the feature and the mainline branch level.

Continuous Delivery (CD) extends from the CI stage with automated release capabilities. The continuously integrated code commits are further tested for performance and functionality until approved for immediate release. The latter is ensured by packaging and deploying the build into test environment that replicates a production environment. Every new feature update is ready and can potentially reach end-users following the release approval. The release process is automated, repeatable, as well as deployable with a single click of a button.

Differentiating attributes of Continuous Delivery (CD) include:

- Strong Continuous Integration is a fundamental requirement to effectively execute Continuous Delivery.
- The release process is short, repeatable and automated.
- Strong emphasis on automated testing and builds.

- Each feature candidate should be deployable immediately following manual approval.
- The decision to release a feature candidate to end-users is more of a business decision than a technical decision.

Continuous Deployment is the next evolution stage of Continuous Delivery where the deployment to production is triggered automatically once a functional and validated release cycle is completed. Only a failed test will prevent the build update from going live. The release is usually aimed at a small subset of end-users to gain immediate feedback, which is channeled back to developers who take corrective measures for future development iterations. In many ways, Continuous Deployment requires a change in mindset among developers, testers and operations teams. The DevOps team is required to share a common vision toward Continuous Deployment and follow far-reaching responsibilities across the SDLC pipeline to ensure that multiple small code commits are deployable at a rapid pace. The problems must be identified early and rectified early during the SDLC pipeline with the purpose of reaching end-users. Every feature update has to be focused on end-users and aligned with the business goals, since the feature releases are automatically deployed into production.

Differentiating attributes of Continuous Deployment include:

- No human intervention preventing deployment into production.
- The release process is automated.
- The mainline branch passes all tests through to production stage.
- The release is well-documented at the pace of release process.
- Automation testing is adopted with maximum code coverage.
- Deployment is conducted in small batches of feature updates.
- End-users receive continuous changes and updates to the software.
- Shortened feedback loop between the DevOps teams and end-users translates into fewer Ops processes and limitations.

Continuous Deployment dramatically reduces the time between developer writing a code change and end-users experiencing the updated software functionality. It may take only a few minutes between committing a code change and deployment into production, as long as all automated tests are passed.

While the software engineering practices of Continuous Integration, Continuous Delivery and Continuous Deployment follow pre-defined fundamental principles, each organization may follow their own approach to DevOps flows, team structures and tooling. The continuity and frequency of the development and release cycles also depend on the organization, their business requirements and technical limitations. Every DevOps team must evaluate their own requirements and identify an optimum release cycle strategy using the basic differentiating principles of interdependent CI/CD processes.