The 8 Rules of Uncompromising Continuous Delivery

DBmaestro

Defining the Best Practices and Tools for Every Deployment Pipeline

Executive Summary

More and more, continuous delivery is becoming the mainstream method of software development. The reasons why are clear: increased productivity, speed, reliability and quality. Add to that the reduced risk of production problems due to rapid and iterative testing, and the case for continuous delivery is compelling.

Like any other process, however, continuous delivery can be mishandled. And it must be carefully designed to meet specific organizational and environmental challenges. In this paper, therefore, we have distilled eight all-purpose continuous delivery best practices and identified the leading tools for each of them.

The "unbreakable rules" below touch on building, testing and deployment, with an emphasis on version control and automation throughout – from coding to corrective measures to the database.

Continuous Delivery is More Than Meets the Eye

Continuous delivery refers to the development of software in short and ongoing build-test-release cycles that form a series of validations along the deployment pipeline. This repeated testing of processes and scripts before deployment to production means most errors are discovered early on and have minimal impact. Moreover, finding and fixing such errors is much easier with fewer changes per release. Software is thus released faster and more frequently, yet with greatly reduced risk of deployment problems due to a heavy focus on visibility, instant feedback and incremental changes.

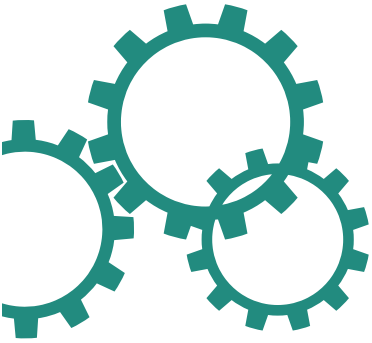
This end-to-end process, along with open communication and enhanced collaboration internally, optimizes the development to production cycle and reduces its cost. The result is an accelerated time to market and greater customer satisfaction, as the end product is more stable, of better overall quality, and often more quickly reflects user feedback on previous releases.

Continuous delivery fits in organically with agile software development and lean thinking, in that it is focused on quickly making small changes that add value for the end user. However, it is actually much more than that. Continuous delivery ensures an iteration of the software under development is always ready for release.

To meet that goal, a straightforward and repeatable deployment process must be in place. Most reliable in that regard, of course, is an automated deployment pipeline that can feed into any environment. As Jez Humble and Dave Farley suggested in their book, *Continuous Delivery*, "Automate everything…. less effort is needed to run it and monitor its progress – and it will ensure you get consistent results." In their view, perfecting the automation process is key to generating a substantial return on investment from continuous delivery.

While such automation is surely an important element, it is not what defines continuous development. Designing repeatable processes, feedback loops, setting the scope of desired changes, and the definition of software readiness are all elements of this process of continuously improvement.

For the most effective results from the entire continuous delivery process, it is useful to look at several specific best practices for an efficient deployment pipeline.



The 8 Unbreakable Rules of Flawless Continuous Delivery

Effective deployment is a pipeline process, from code check-in or configuration to compilation and testing, to production. For the most efficient and effective continuous delivery along this deployment pipeline, we have identified eight unbreakable rules drawn from industry best practices. Each of these best practices is supported by a set of complimentary task-specific tools, in what's known as the "DevOps toolchain", in order to automate the process from end to end and minimize human error.

1. Version Control Everything

The native code deployment process generally includes a built-in safety net, preventing out-of-process locally-generated artifacts from entering the production environment. Every change a developer makes must be documented in the source control repository, or it is not included in the build process. Furthermore, if a developer copies an artifact generated locally to a test environment, the next deployment will override this out-of-process change.

This insistence on having a single source of truth – and a reliable one – creates a stable foundation for all development processes. However, the processes are only as strong as their weakest link. That's why the primary golden rule for effective continuous delivery is: version control everything. What works well for code, will also preserve the integrity of configuration, scripts, databases, website html, and even documentation.

Thus, source control tools (e.g., Perforce, Git, Subversion) become the building blocks for the entire process, ranging across all key assets.

Two underlying cultural shifts are needed to support the "version control everything" approach. One is drilling home the idea that everyone is responsible for playing by the rules, from coding to production. Part of this effort is making sure that all teams are using the same issue tracking tools (e.g., Jira and ZenDesk), for both internal and customer-generated issues, which increases responsiveness and visibility.

The other cultural change is using clear and consistent definitions of process concepts (such as "release" or "done"), for better communication and maximum value from each version. This can be greatly facilitated with application lifecycle management (ALM) tools (e.g., Clarive, IBM RTC, Ms-TFS, TeamForge) that provide consistent and coherent end-to-end development management, from planning to final release.

2. Build Binaries Once

Not every organization's build process is the same or uses the same tools, but every build process in a single organization must be completely consistent. Whether the build is a single file deployed to an automated test environment or a complex build with several different possible deployment versions, each build version should happen exactly the same way and result in unique binary artifacts.

This one-time-only compiling eliminates the risk of untracked differences due to various deployment environments, third-party libraries, or different compilation contexts or configurations that will result in an unstable or unpredictable release. Save the compilation phase output (the binaries) to a binary repository, from which the deployment process can retrieve the relevant artifacts.

Binary repositories (e.g., Artifactory, Nexus) are to binary artifacts what source repositories are to sources. On the simplest level, they become a single source of truth for large, unchanging files like nightly builds or releases; at the same time, the repository allows for managing binary dependencies, avoiding the need to rebuild from scratch, and preserves an original build.

3. Deploy the Same Way Every Time

An inconsistent deployment process can become a source of configuration drift across environments, especially with the rapid releases common in a continuous delivery system. The result is valuable time and effort wasted in identifying and addressing difficulties arising from the application working in one environment and not another.

For a reliable process in all environments, then, the same set of steps must be repeated from start to finish. This ensures the same results in lower environments, with more frequent deployments (such as integration, QA, etc.), as in higher environments (pre-production and production), with fewer deployments.

The surest guarantee of such uncompromising consistency is, as expected, automating the deployment mechanism using configuration management (e.g., Ansible, Puppet, Chef, Salt) and deployment tools (e.g., IBM uDeploy, CA Release Automation, XebiaLabs, Automic).

4. Smoke Test

Smoke testing deployments is a very rapid way to make sure that the most crucial functions of a program work and will pass basic diagnostics. This non-exhaustive software testing (of all elements, such as services, database, messaging bus, external services, etc.) does not provide the same fine-grain comprehensiveness as full test suites; however, it can be run frequently and quickly, often in a

matter minutes (rather than hours or days). This allows for a much quicker turnaround on what can become very time-consuming and basic issues.

The faster the feedback loop works (especially with automated test tools such as Telerik, QTP, TestComplete), the higher the final product's quality will be and the quicker it will reach its release-ready state.

5. Deploy into a Production-like Environment

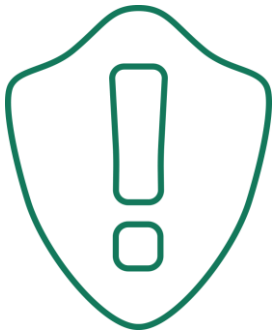
Each new deployment (using, for example, IBM uDeploy, CA Release Automation, XebiaLabs, Automic) should be made into an environment that mimics as closely as possible the actual final production environment. This includes infrastructure, operating system, databases, patches, network topology, firewalls, and configuration. By validating software changes in this type of detailed pre-production environment, mismatches and last minute surprises can be effectively eliminated and applications can be safely released to production at any time.

6. Instant Propagation

Continuous delivery by definition must be an ongoing process of building, testing and releasing. Every check-in to a source control repository should trigger compiling (if needed) and packaging by a build server. This, in turn, should automatically initiate certain predefined testing until the software can either be marked as releasable or returned to development.

Continuous integration tools (e.g., Jenkins / CloudBees, Bamboo, TeamCity) ensure that cascade of actions along the delivery pipeline is consistent, automatic and instant. Thus, development teams merge developed code and get feedback from automated test tools many times a day, for a more efficient deployment and update process.

7. Stop the Line



Short feedback loops are key to dealing with application or feature bugs quickly and efficiently. When a post-build failure is detected, continuous integration tools (e.g. Jenkins / CloudBees, Bamboo, TeamCity) or release automation tools (e.g. IBM uDeploy, CA Release Automation, XebiaLabs, Automic) can be used to automatically halt the deployment process and loop the issue back to developers for immediate attention. While this requires essentially starting from scratch before moving ahead, it also produces higher quality code much faster than other possible repair methods. In addition, continuous improvement all along the automated pipeline process leaves fewer issues to deal with later, when doing so becomes much more expensive.

8. Include the Database

No list of continuous delivery best practices would be complete without advising that the database be managed using the same basic protocols that ensure secure and reliable source code, task, configuration, build and deployment management. However, many times this is neglected because the database is unlike other software components or compiled code that are easily copied from development to testing to production. Yet, the database is actually a repository of the most valued and irreplaceable asset - our data – and preserving it accurately is imperative to continuous delivery.

Although the database poses several unique challenges, specialized database tools such as enforced database source control (for all environments), a database build automation tool, and database release and verification processes can ensure a stable resource in your DevOps chain.

Confident Continuous Delivery

The eight best practices we have looked at for optimizing continuous delivery create a deployment pipeline with increased productivity, faster time to market, reduced risk, and increased quality. Of course, every company's needs are different, but the "golden rules" we've identified are all easily adaptable for enterprises of any scale.

The key to ensuring a quick, robust and accurately repeatable continuous delivery is, as we have seen, automation, deployment consistency, comprehensive testing and database version controls. With these failsafes in place, pushing the "release to production" button can be done much more often and with much more confidence.

About DBmaestro

DBmaestro is a pioneer and a leading solution provider for database DevOps. Its flagship product, **DBmaestro DevOps Platform**, enables Agile development and Continuous Integration and Delivery for the database.

The platform supports streamlining of development process management and enforcing change policy practices.

The solution empowers agile team collaboration while fostering regulatory compliance and governance.

With DBmaestro, organizations can facilitate DevOps for database by executing deployment automation, enhancing and reinforcing security as well as mitigating risk.

DBmaestro's solutions are deployed at many Fortune 500 companies.

Connect with DBmaestro

Read our blog: http://www.dbmaestro.com/blog/

Follow us on Twitter: https://twitter.com/dbMaestro

Follow us on LinkedIn: https://www.linkedin.com/company/dbmaestro

Want to learn more? Sign up for a demo!

DBmaestro

Headquarters 300 Baker Avenue, suite 300, Concord, MA 01742, USA +1 978.405.3368

EMEA Headquarters 59-60 Thames St, Windsor, Berkshire, SL4 1TX, UK +44 (0)1753 968 358