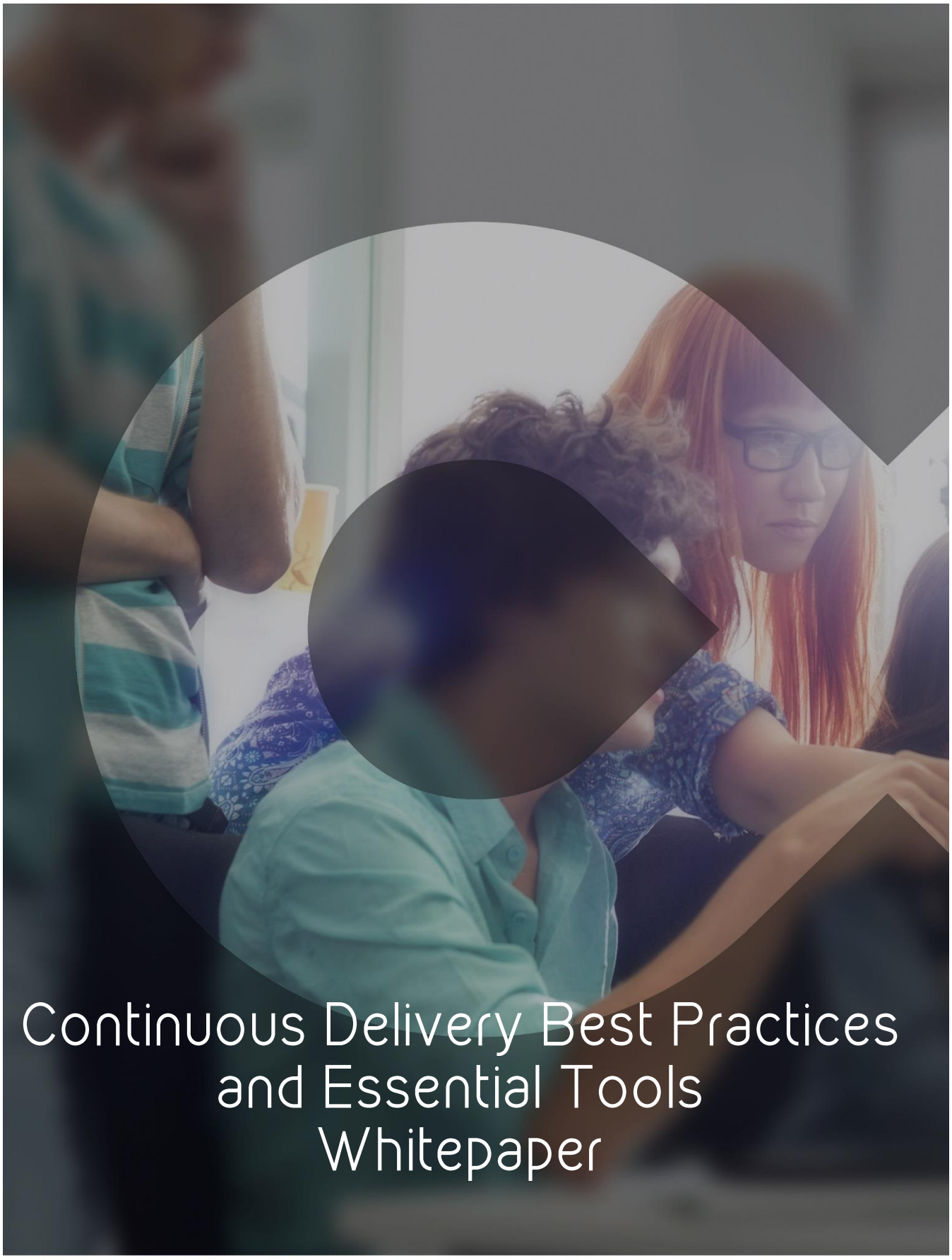


Continuous Delivery Best Practices and Essential Tools Whitepaper

DBmaestro
DevOps for Database

DBmaestro
DevOps for Database

Introduction

Survey after survey prove that DevOps and Continuous Delivery are quickly moving into the mainstream for one reason: they work! Continuous processes done right will increase productivity, speed up time to market, reduce risk, and increase quality.

This whitepaper will answer the following questions:

- What is Continuous Delivery?
- What is DevOps?
- What are the differences between the two?
- How do I build the perfect deployment pipeline?
- Does the database also need DevOps?
- What tools do I need in my DevOps Toolchain?

The Whitepaper is geared to those who are just beginning their DevOps journey as well those with DevOps and CD experience.

What are DevOps and Continuous Delivery?

Continuous delivery is a method that promotes the adoption of an automated deployment pipeline to quickly and reliably release software into production. Its goal is to establish an optimized end-to-end process, enhance the development to production cycles, lower the risk of release problems, and support a quicker time to market.

DevOps and Continuous Delivery possess a shared background in agile methods and Lean Thinking: small and quick changes with focused value to the end customer. They are well communicated and collaborated internally, thus helping achieve quick time to market, with reduced risk.

DevOps is about the culture. About creating better collaboration between development and operations. About building well-defined processes. About being agile.

The combination of 'softer' or flexible DevOps philosophical concepts that go hand in hand with a very practical set of continuous delivery best practices, means that the two are preaching about similar, but not identical things.

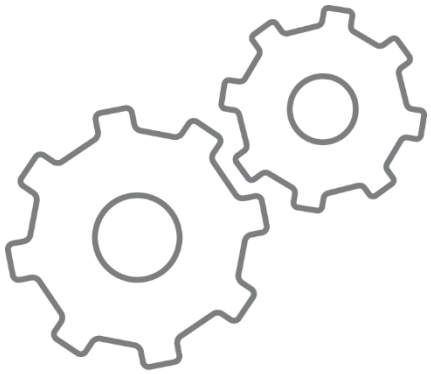
DevOps is not Continuous Delivery - DevOps does not just deal with automation, but works to affect cultural change, create a better collaboration between development and operations, build well-defined processes, and be agile.

Continuous Delivery is not automation; while automation is an important part of Continuous Delivery, there is also a lot more to it. Continuous Delivery also works to build repeatable processes,

create feedback loops, define the scope of changes, and determine what should actually be promoted or defined as “done”.

Continuous Delivery Principles

In order to achieve the Holy Grail of an “automatic, quality based, repeatable, reliable, continuously improving process”, and not having to feel like you are chasing a myth, we must break that process into simpler practices. Building the “pipeline” enables us to deal with the different stages of the process, one by one.



A deployment pipeline makes sure a change is processing in a controlled flow. The process is as follows: a code check-in or configuration change triggers the flow. The change is compiled, and goes through a set of tests – usually unit tests and static code analysis.

A successful unit test triggers automatic application tests and regression tests. After successfully passing these tests, the change can be either ready for production or go through additional manual tests and user-acceptance tests before hitting production.

Jez Humble and Dave Farley defined the following principles in their Book “Continuous Delivery”:

1. **Repeatable reliable process** – use the same release process in all environments. If a feature or enhancement has to work through one process on its way into the integration environment, and another process into QA, issues find a way of popping up.

2. **Automate everything** – automate your builds, your testing, your releases, your configuration changes and everything else. Manual processes are inherently less repeatable, more prone to error and less efficient. Once you automate a process, less effort is needed to run it and monitor its progress – and it will ensure you get consistent results.

3. **Version control everything** – code, configuration, scripts, databases, documentation. Everything! Having one source of truth – and a reliable one – gives you a stable foundation to build your processes upon.

4. **“Bring the pain forward”** – deal with the hard stuff first. Time-consuming or error prone tasks should be dealt with as soon as you can. Once you get the painful issues out of the way, the rest will most likely be easier to prefect.

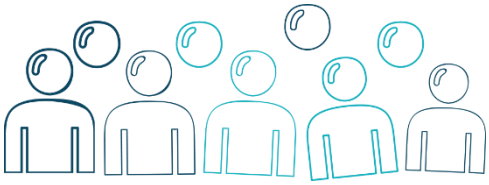
5. **Build in quality** – create short feedback loops to deal with bugs as soon as they are created. By having issues looped back to developers as soon as they fail post-build test, it will enable them to produce higher quality code quicker. In addition, fewer issues will be found later on in the process, when it will be more expensive to fix.

6. **Done means released** – a feature is done only when it is in production. Having a clear definition of “done” right from the start will help everyone communicate better, and realize the value in each feature.

7. **Everyone is responsible** – "It works on my station" was never a valid excuse. Responsibility should extend all the way to production. Cultural change can be the hardest to implement. However, having management support and an enthusiastic champion will certainly help.

8. **Continuous improvement** – of all of the above. This principle is especially important for automation. If "practice makes perfect" than automation is the next level – perfect, repeatable, reliable and efficient – perfecting your automation process is a key ingredient to bringing substantial ROI to CD.

Continuous Delivery Best Practices



Achieving an efficient deployment pipeline is done by following these best practices:

1. **Build your binaries only once**: Compiling code should only happen once, eliminating the risk of introducing differences due to environments, third party libraries, different compilation contexts, and configuration differences.

2. **Deploy the same way to all environments**: Use the same automated release mechanism for each environment, making sure the deployment process itself is not a source of potential issues. While you deploy many times to lower environments (integration, QA, etc.) and fewer times to higher environments (pre-production and production), you can't afford to fail a deployment to production because it was the least tested deployment process.

3. **Smoke-Test your deployments**: A non-exhaustive software test (essentially testing "everything" – services, database, messaging bus, external services, etc.) that doesn't bother with finer details but ascertains that the most crucial functions of a program work, will give you the confidence that your application actually runs and passes basic diagnostics.

4. **Deploy into a copy of production**: Create a production-like or pre-production environment, identical to production, to validate changes before pushing them to production. This will eliminate mismatches and last minute surprises. A copy of production should be as close to production as possible with regards to infrastructure, operating system, databases, patches, network topology, firewalls, and configuration.

5. **Instant propagation**: The first stage should be triggered upon every check-in, and each stage should trigger the next one immediately upon successful completion. If you build code hourly, acceptance tests nightly, and load tests over the weekend, you will prevent the achievement of an efficient process and a reliable feedback loop.

6. **Stop the line**: When a stage in the pipeline fails, you should automatically stop the process. Fix whatever broke, and start again from scratch before doing anything else.

The pipeline process helps establish a release mechanism that will reduce development costs, minimize risk of release failures, and allow you to practice your production releases many times before actually pushing the “release to production” button.

Continuous improvement of the automated pipeline process will ensure that less and less holes remain, guaranteeing quality and making sure that you always retain visibility of your production readiness.

Continuous Delivery for the Database

Making sure your database can participate in the efficient deployment pipeline is obviously critical. However, the database requires dealing with different challenges than application code. Many times, implementing continuous delivery for the database proves to be a challenge. The database, unlike other software components and code or compiled code, is not a collection of files. It is not something you can just copy from your development to testing and to production, obviously, because the database is a container of our most valued asset—our data. It holds all application content, customer transaction, etc. Business data must be preserved.

As a result, DevOps and Continuous Delivery for Database has taken a back seat to source code, as we see the tools and methodologies perfected, leaving database automation to play a game of catch up at a high risk.

To learn how to overcome the challenge of DevOps for the database, **download this free white paper.**

Recommended Toolchain

Continuous processes, can increase productivity, speed up time to market, reduce risk, and increase quality, but they need to be built on top of a robust process. The process must be thought out correctly and able to handle the organizational challenges, but at the same time needs to be very efficient, quick, robust and accurately repeatable. This is exactly the reason why we always use tools – and the reason for the “DevOps Toolchain”.

The tool chain philosophy advocates that a set of complimentary task specific tools are used in combinations to automate an end-to-end process.

The common tools in the chain should be:

1. Collaboration tools: Help teams work together more easily, regardless of time zones or locations. A rapid action oriented communication designed to share knowledge and save time.
(See: Slack, Campfire)

2. Planning tools: Provide transparency to stakeholder and participants. Working together, teams can plan towards common goals, and better understanding of dependencies. Bottlenecks and conflicting priorities are more visible. (See: Clarizen and Asana)

3. Source control tools: The building blocks for the entire process ranging across all key assets. Whether code, configuration, documentation, database, compiled resources and your web site html – you can only gain by managing them in your one true source of truth. From there – everything else can be built when required. (See: Perforce, Git, Subversion)

4. ALM tools: enabling end to end management for development efforts - planning, tracking, managing requirements, tasks, issues, build, releases etc. (See: Clarive, IBM RTC, Ms-TFS, TeamForge)

5. Issue tracking tools: Increase responsiveness and visibility. All teams should use the same issue tracking tool, unifying internal issue tracking as well as customer generated ones.
(See: Jira and ZenDesk)

6. Configuration management tools: Enforcing desired state and consistency at scale. Infrastructure should be treated exactly as code that can be provisioned and configured in a repeatable way. Avoiding configuration drift across environments will save valuable time and prevent difficulties caused by the application working in one environment and not another. (See: Ansible, Puppet, Chef, Salt)

7. Continuous integration tools: Immediate feedback loop by merging code regularly. Teams merge developed code many times a day, getting feedback from automated test tools. (See: Jenkins / CloudBees, Bamboo, TeamCity)

8. Binary Repositories are to binary artifacts what source repositories are to sources. When you need to store large, unchanging files like nightly builds or releases, (and also need to manage their dependencies so you will not need to rebuild everything) or if you want to make sure you can reproduce the original build – using binary repositories will prove to be a best practice. (See: Artifactory, Nexus)

9. Monitoring tools are essential for DevOps, providing crucial information that will help you ensure service uptime and optimal performance. (See: AppDynamics, New Relic, Dynatrace, Sensu, BigPanda, Sumologic and more)

10. Automated test tools: Verify code quality before passing the build. The quicker the feedback loop works – the higher the quality gets, and the quicker you reach the desired “definition of done”. (See: Telerik, QTP,TestComplete)

11. Deployment tools: In an effective DevOps environment, application deployments are frequent, predictable, and reliable. Continuous Delivery means that applications can be released to production at any time you want in order to improve time to market, while keeping risk as low as possible.
(See: IBM uDeploy, CA Release Automation, XebiaLabs)

12. Containers: Using containers may simplify the creation of highly distributed systems by allowing application isolation on physical or virtual machines, and enabling rapid and portable deployments of reusable containers (See: Docker)

13. The database obviously needs to be an honored member of the managed resources family. Managing source code, tasks, configuration, builds and deployments is incomplete if the database is the weak link not following the same best practices and process. Using specialized database tools such as enforced database source control, a database build automation tool, and database release and verification processes will ensure your database is a stable resource in your DevOps chain. (See: DBmaestro)

Summary

While DevOps and Continuous Delivery possess a shared background in agile methods and Lean Thinking they are not the same thing. DevOps is about the culture, primarily creating better collaboration between development and operations. Continuous Delivery on the other hand is about building automated, repeatable processes.

In order to reap the benefits of automation brought on by Continuous Delivery, a "pipeline" must be built. This enables us to deal with the different stages of the process, one by one. A good deployment pipeline is repeatable, completely automated, has blanket version control, and is continuously improving.

An often overlooked part of the deployment pipeline is the database. While the database does present different challenges than application code, it cannot be ignored as it is a container of our most valued asset—our data. The challenges presented by Continuously Delivery for the database can be overcome with the correct tools and processes.

For continuous processes to be efficient, quick, robust and accurately repeatable, you need the right tools called a "DevOps Toolchain". There is a wide range of task specific tools available to build your toolchain. Different tools are more suitable for different needs and business requirements. Selecting the right DevOps tools for your organization requires both strategic vision and must match your organization's business goals.

Headquarters 1302 Waugh Drive, suite 290, Houston, TX. 77019, USA +1 281.668.8799
EMEA Headquarters 21 Yagiya Kapaim st. Petach Tikva 4900101, Israel +972.3.9248558