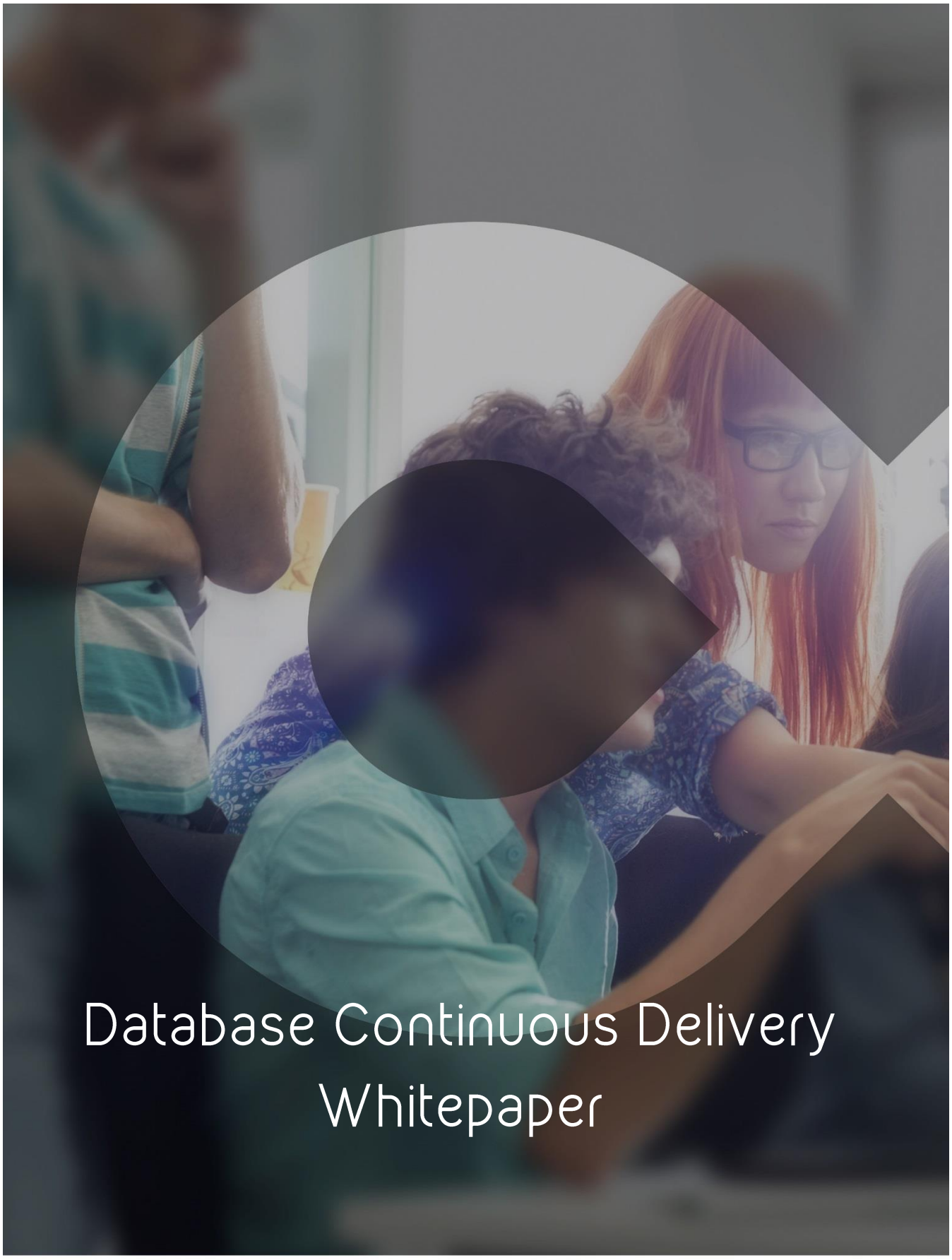


Database Continuous Delivery Whitepaper

DBmaestro
DevOps for Database

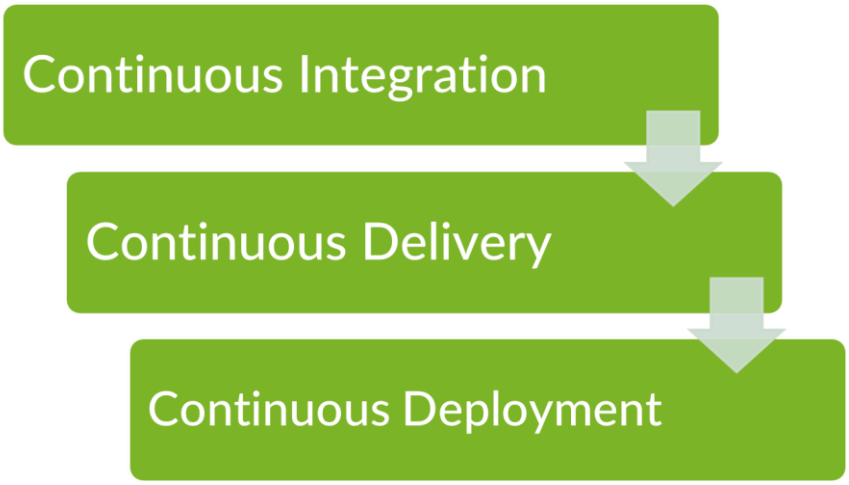
DBmaestro
DevOps for Database

A Fast-Moving World: Agile, DevOps, and Automation

Business needs are the most significant driver of change. The ability to do better with less and deliver it sooner will differentiate leading and successful companies from the rest.

When a competitor rolls out relevant features, faster and with better quality than you, you're eventually going to lose market share. This is exactly why Agile Development was born: the need to move quicker and address ever-changing requirements, while ensuring optimum quality and dealing with limited resources. You just can't wait those six months until the next roll-out or release. The Waterfall methodology's big release concept doesn't cut it anymore. Agility is what is expected from technology companies and IT divisions.

The next natural step from Agile was to find a way to take Agile to production by linking development with operations. This has given rise to DevOps.



To effectively master Agile sprint deployments and practice DevOps, one needs to be able to implement deployment and process automation. Otherwise, deployments and releases will require manual steps and processes, which are not always accurately repeatable, are prone to human errors, and cannot be handled with high frequency.

Continuous Integration, Continuous Delivery, and Continuous Deployment are the common principles and practices for structurally handling the process of automation, and setting ground rules for the many participants involved in the development, building, testing, and release of software update processes.

These principles are not new, but they are gaining traction and adoption, as they prove their benefits - just like Agile development did some years ago.

According to a recent survey, 65% of software developers, managers and executives report that their organizations have started down the path to Continuous Delivery.

DBmaestro
DevOps for Database

Continuous Integration, Delivery, and Deployment

As a set of principles and practices, Continuous Integration, Continuous Delivery, and Continuous Deployment are not a case of "one size fit all". It is important to understand that every company might have its own unique challenges, and these practices should be tuned to fit organizational structure and culture.

Continuous Integration

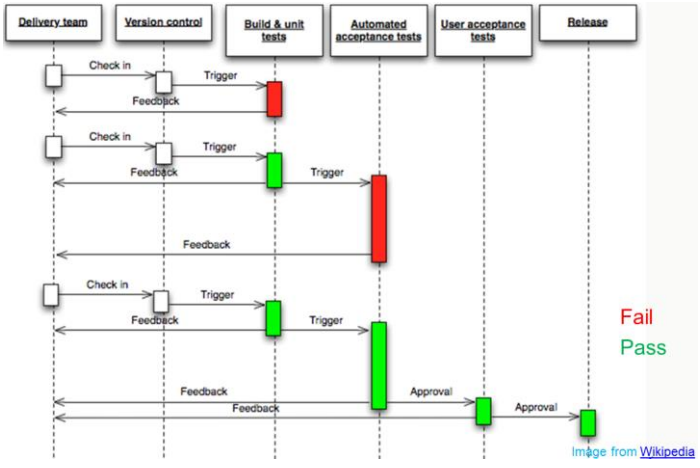
Continuous Integration is designed to streamline development and prevent integration problems. This goal is usually achieved with the help of build servers. These servers receive code changes from the version control repository, automatically build it and run unit tests to verify the changes, and ensure quick feedback to the developers. Unit tests might run from time to time, or even after each change has been committed (checked-in), thus preventing or quickly alerting about code changes that might break other code, or may fail to pass tests.

In addition to running code-centric unit tests to ensure code completeness, running integration tests or application level regression test will help complete the picture, and guarantee quality levels altogether.

Quick feedback on integration problems, and automated tests to ensure quality help to increase visibility into overall development and save time locating problems. This, in turn, contributes to overall savings in development and integration time and higher quality.

Continuous Delivery

Continuous Delivery is the next automation step after Continuous Integration. While striving to become efficient, lean, and even more agile, you can start planning and making sure each change is "releasable", so a tested build is always ready for deployment.



Moving changes between the different lifecycle stages should be done automatically. The overall process is demonstrated in the diagram on the right:

Checking in changes in development ⇨ building the deploy package ⇨ running unit tests ⇨ moving the changes to testing, and later to a staging environment ⇨ running acceptance tests.

In case of a failure, one should get automated alerts about the problem, and then go back to development, and restart the cycle.

Once the process is completed, a fully-tested application is available to be released to production, with a click of a button. The actual deployment to production will be manually actuated, followed by a re-running of the regression test.

As all changes are tested and accounted for, and deployment between previous lifecycle stages had also been tested, the actual deployment to production becomes much easier, and significantly less risky.

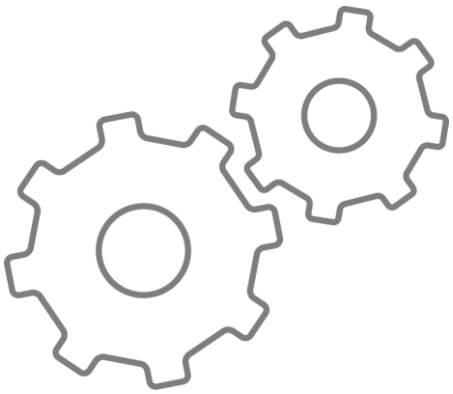
Following Continuous Delivery practices means one always has a releasable version at hand. This enables timely releases based on business decisions, time-to-market, etc.

Continuous Deployment

Continuous Deployment takes the next step, pushing changes automatically to production (unlike Continuous Delivery) and running the concluding set of tests there.

Leveraging Continuous Deployment in a SaaS type of application or product (like Facebook, Amazon, etc.) makes a lot of sense, as a company can stream and throttle traffic to a new feature, do A/B testing to evaluate new changes, run an old release side by side with the new release, and measure and manage changes with confidence.

Continuous Deployment may be risky and doesn't always make sense from a business perspective, unlike Continuous Delivery, which is highly beneficial and plays a central role in the implementation of DevOps.



Measuring Continuous Processes Success

Success from continuous processes is usually clear, and focuses around these areas:
1. More rapid changes – being able to react quicker
2. Less changes backed out – higher code quality, quicker time to market
3. More stable releases – less defects making it out to end customers
4. Better collaboration between Development and Operations (DevOps)

By automating everything and moving tested, focused updates and process "upstream", you'll achieve better service, happier customers, and a stronger bottom line.

According to a recent survey, the top-ranked barrier for adopting Continuous Delivery is integrating automation technologies (version control, automated testing, etc.).

Safe Database Continuous Delivery

Dealing with database deployments is tricky. Unlike other software components, such as code or compiled code, a database is not a collection of files. The database is a container of your most valued asset – the business data, which must be preserved. It holds all application content, customer transactions, etc. In order to promote database changes, a transition code needs to be developed - scripts to handle database schema structure (table structure), database code (procedures, functions, etc.), and content used by the application (metadata, lookup content, or parameters tables).

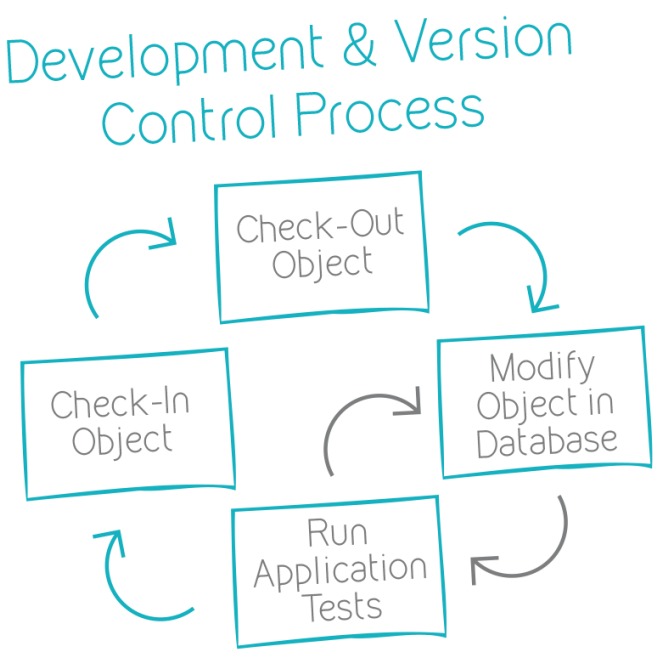
Achieving automation by scripting database object change-scripts into traditional version control is limited, inflexible, disconnected from the database itself, and may be inaccurate and prone to missing updates within the target environment because of conflicting changes. Using "compare & sync" tools is a risky thing to automate. The two concepts do not work well together, as one is unaware of the other.

DBAs, being both well aware of database deployment pitfalls and bearers of the scars of the most inopportune break downs, tend to shy away from automation-based processes, as they are not confident in the accuracy of the automation script generators. Nor are they confident in the ability of pre-prepared, manually-generated scripts to remain true any time after they were developed. In order to avoid conflicts, they often take things into their own hands. The path of carefully examining changes and manually creating change scripts as close to the deployment event as possible seems less frustrating by comparison.
A better solution had to be found.

In order to take a database into proper automation, one must factor in the following:

1. Proper database version control, dealing with databases' unique challenges (structure, code and content), while enforcing a single work process. This prevents any out-of process-changes, code overrides, or incomplete updates .

2. Leverage proven version control best practices (check in & out, changes, etc.) for complete information about who was doing what, when, and why. Making sure changes are perfectly documented is the basis for later deploying them.

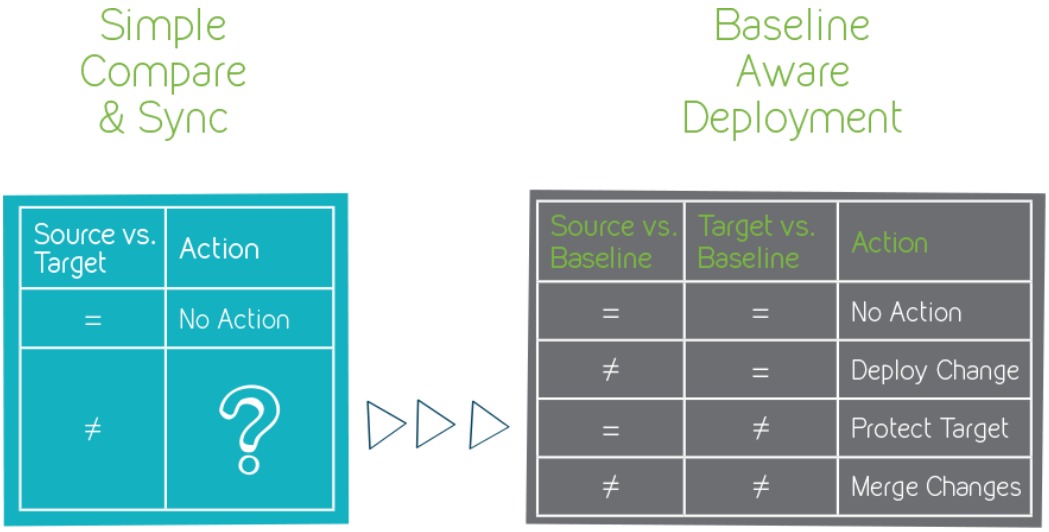


3. Harmony with task-based development allows for correlation of each version control change with a change request or a trouble ticket. This enables task-based deployments, partial deployments, and last minute scope changes to be coordinated between code and database.

4. Ensure configuration management and consistency, so every development environment, branch, trunk, sand-box, and testing or production environment follows the same structure and matching status. This also ensures that any deviation and difference is well accounted for.

5. Scriptable interfaces, to deal with the automation of deployment processes, providing repeatable results every single time. Even the most sophisticated solution becomes cumbersome if you have to use the UI to do the same task over and over again .

6. Utilize reliable deployment script generators, which are capable of dealing with conflicts and merges of database code, as well as cross-updates from other teams, while also ignoring wrong code overrides. These must be fully integrated into the version control repository.

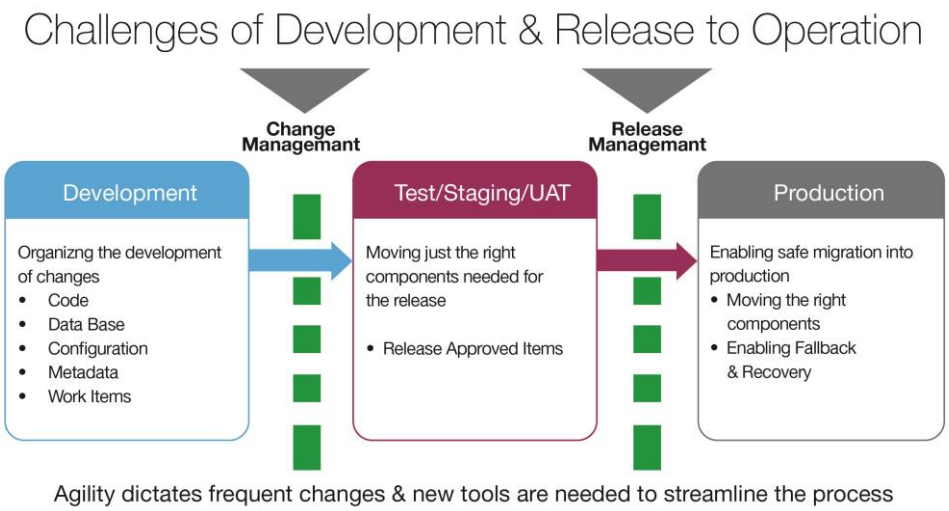


7. Provide automatically-generated development scripts on the fly to efficiently deploy projects of any scope, from multi-schema mega-updates, to a single, task-based change and its dependent objects.

8. Leverage labels before and after deployment of changes, to act as a safety-net/baseline, so quick and easy roll-backs are always close at hand.

9. Seamless integration with other systems (ALM, change management/trouble tickets, build servers, and release managers)

Implementing a solution to deal with these challenges enables easy integration with the rest of the change and release processes, and helps one achieve complete end-to-end Continuous Delivery.



DBmaestro
DevOps for Database

Summary

The database creates a real challenge for automation, which is why organizations participate in continuous processes. Scripting database object change scripts into traditional version control solutions, or using "compare & sync" tools can be plain risky from an automation perspective, as the two concepts are unaware of each other. A better solution needs to be implemented, one that promotes Continuous Delivery and DevOps for the database.

Database Continuous Delivery should follow the proven best practices of change management, enforcing a single change process over the database, and enabling efficient resolution of deployment conflicts to eliminate the risk of code overrides, cross updates and merges of code, while plugging into the rest of the release process.

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

Follow Us: