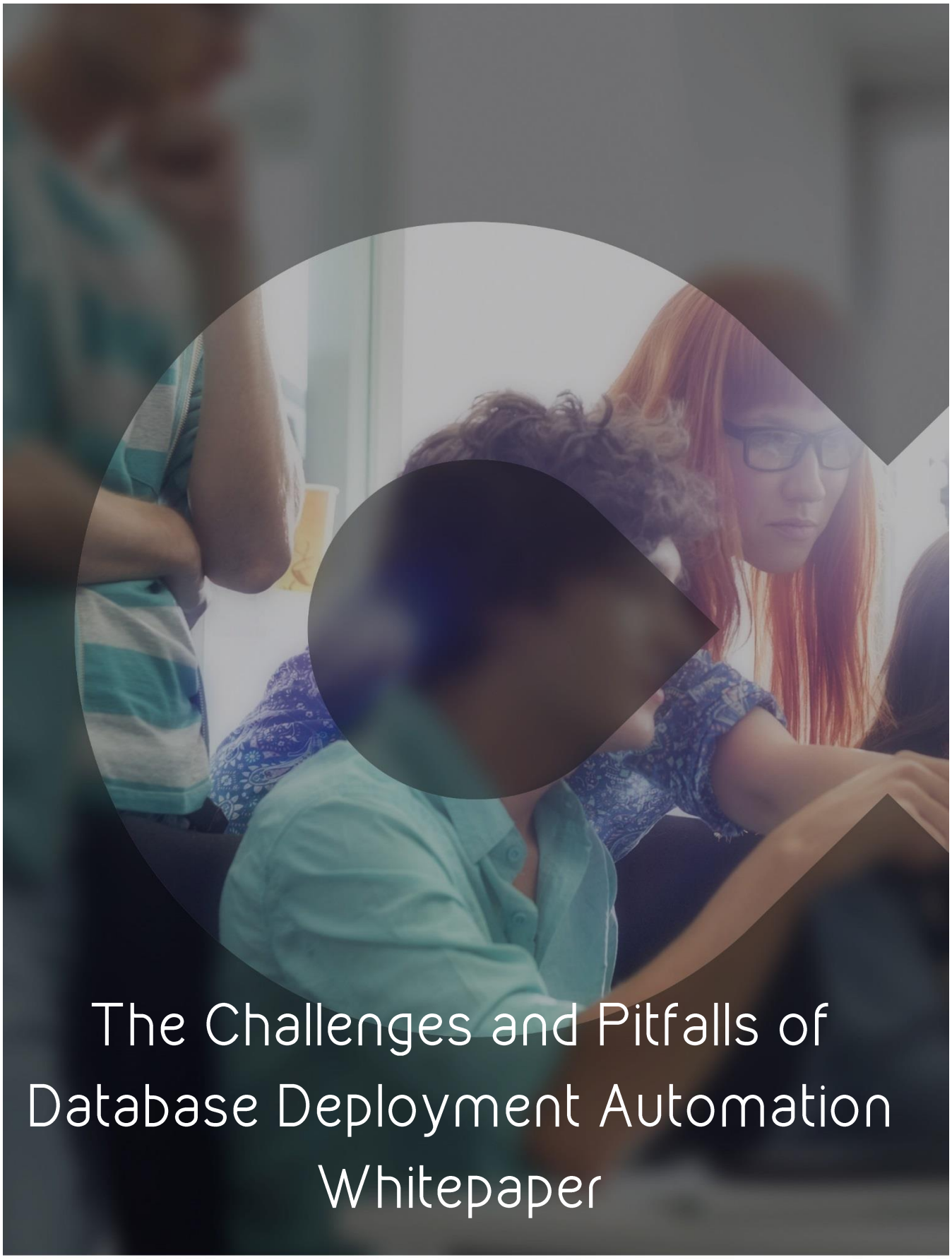


The Challenges and Pitfalls of Database Deployment Automation Whitepaper

DBmaestro
DevOps for Database

DBmaestro
DevOps for Database

A fast moving world; Agile & DevOps

As business needs are the most significant driver of change, doing better with less and delivering it sooner is what differentiates leading and successful companies from the rest.

When a competitor delivers relevant features, faster and with better quality than you, you're eventually going to lose market share. Investing in sales and marketing campaigns to compensate for your product is expensive and not always reliable and you might find out that customers are moving to the superior product.

This is exactly why 'Agile Development' was born: the need to move quicker, deal with ever changing requirements (as our target market and competition are never standing still), with best quality that can be assured, usually with not enough resources. Agile is what is expected from technology companies and IT divisions.

The next natural step from Agile is finding a way to take Agile to production; linking development with operations. This has given rise to 'DevOps.'

Understanding the main goal of operations is to maintain stable and healthy applications, and the main goal of development is to continually innovate and provide applications that meet business and customer needs, is crucial to the development of DevOps. While there is no doubt that change is the greatest enemy of stability, understanding and reconciling this conflict should be the main goal of DevOps.

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

Dealing with database deployments is tricky; unlike other software components and code or compiled code, a database is not a collection of files. It is not something you can just copy from your development to testing and to production because the database is a container of our 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).

DBmaestro
DevOps for Database

The challenges of database change deployment processes

DevOps is a natural evolution of the software industry, it's not a revolution.

One way of dealing with the database challenge is to force the database into the generic process: create scripts out of database objects and store them in the traditional version control .

That creates other challenges, namely:

1. Scripts in the version control system are not connected to the database objects they represent as these are two separated systems. Coding and testing of the database code is done at the database side, disconnected from any of the coding best practices (check-in, check-out, labels etc.), and prone to all illnesses of the 'old days' such as:

   a. Code-overrides in the database are common as there is nothing to prevent it.

   b. Scripts are required to be taken from the version control before starting the code on the database, to prevent working on the wrong version; but there is nothing to enforce that.

   c. Scripts do not always find their way to the version control system, as it depends on the developer to remember to do so.

   d. Out of process updates go unnoticed, etc.

2. Scripts are manually coded, prone to human error, syntax error, etc

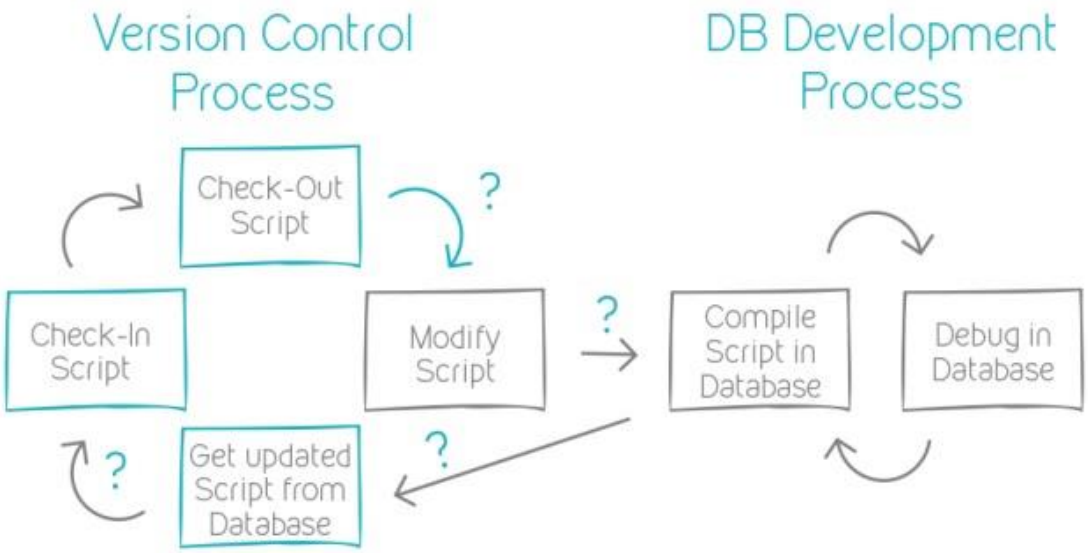
3. To have everything you might later need, you actually have to save two to three scripts for each object; the actual code of the object, the upgrade script, and a roll-back script .

4. Scripts are hard to test in their entirely, as you make changes to a single object, while someone else makes a change to another single object, and running in any order would usually raise errors due to faulty dependencies between these scripts that need to run in a specific order.

5. If a script is developed as a single script to represent the entire update, instead of a single change, it can deal with dependencies, but is much harder to deal with project scope changes. It is a big list of commands.

6. And these scripts, unless super sophisticated, are unaware of changes made in the target environment during the time passed from their coding to the time they are run; potentially overriding production hot-fixes, or work done in parallel by another team.

7. Content changes are very hard to manage. Metadata or lookup content does not practically fit into the version control. In most cases they are simply not managed.



Another concept that emerged in the last decade is using tools for dealing with creation of the transition code between environments. This way of operation is tagged as 'compare & sync', meaning a mechanical comparison examines database objects in a source environment, comparing it to the target environment, and if a difference is discovered, a script to change the target object to mimic the source object is automatically created. For a while it seemed like a good solution, until the holes became more obvious.

The comparison is often performed on a database at selected check-points, usually before deployment, after the development cycle has ended：

1. The compare tool is unaware of any changes that occurred before the time it ran, or any changes that took place at the target environment. We had no information about change, no version control. Just the differences at the given time.

2. Keeping object scripts in a traditional version control solution while using a compare & sync tool to deploy is as un-synergic as you can imagine. One is completely unaware of the other.

3. Manual inspection and detailed knowledge regarding each change must be part of a deployment process. Otherwise, mishaps like overriding good and up-to-date update to production (like a hot fix supplied by one team of developers), with an out-of-date code or structure from a second team that is working on something else entirely.

4. Merging of code between different teams is out of the question. If you need to merge – you need to write the code manually.

Manual processes using a 'compare & sync' tool are possible, but require proficiency and patience. Trying to automate deployment processes based on these tools encompasses a substantial risk to the database.

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 on the above processes, as they are not confident in the accuracy of the automation script generators or the ability for 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.

DBmaestro
DevOps for Database

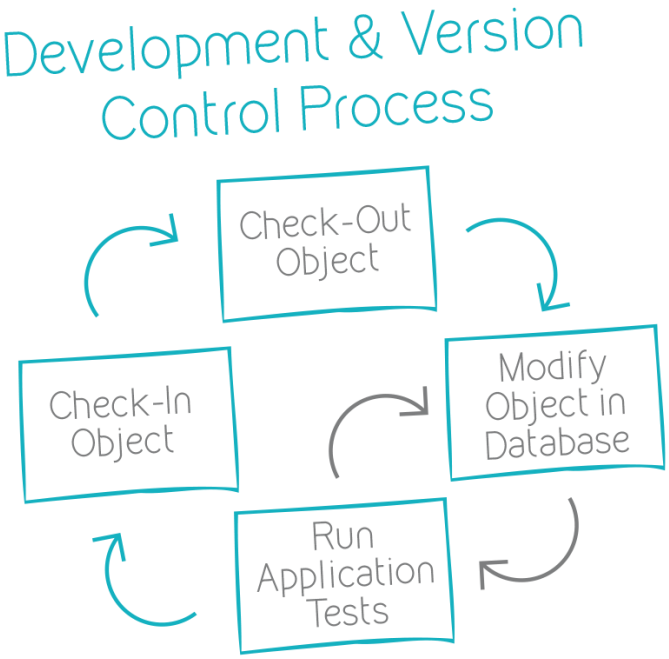
Safe database deployment automation

Achieving automation by scripting database objects change-scripts into traditional version control is limited, inflexible, disconnected from the database itself, and may be untrue and prone to miss updates of target environment because of conflicting scripts. Using 'compare & sync' tools is a risky thing to automate. The two concepts do not play together, as one is unaware of the other. A better solution must be found.

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

1. Proper database version control, dealing with databases' unique challenges (structure, code & 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 base for later deploying them.



3. Harmony with task based development enables correlating 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 & consistency so every development environment, branch, trunk, sand-box, and testing or production environment follows the same structure, and matching status; or any deviation and difference are well accounted for.

5. Scriptable interfaces, to deal with 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. Provide reliable deployment scripts, which are capable of dealing with conflicts and merges of database code, and cross updates from other teams; while also ignoring wrong code overrides, and are fully integrated into the version control repository.

DBmaestro
DevOps for Database

Simple Compare & Sync

Source vs. Target	Action
=	No Action
≠	?

Baseline Aware Deployment

Source vs. Baseline	Target vs. Baseline	Action
=	=	No Action
≠	=	Deploy Change
=	≠	Protect Target
≠	≠	Merge Changes

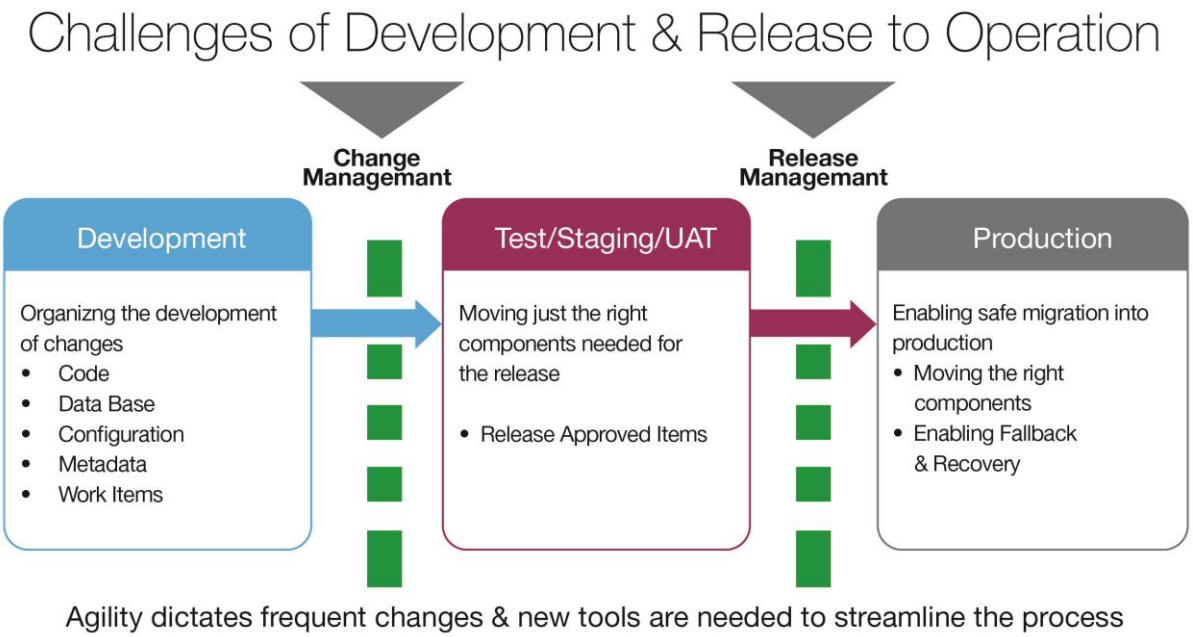
7. Provide automatically generated development scripts on the fly to deal with deploying any combination of project scope, from multi-schema mega-updates, to a single task based change and its dependent objects.

8. Leveraging 'Labels' (Tagging of database structure snapshots and relevant content) before and after deployment of changes, to act as a safety-net, so quick and easy roll-backs are always close at hand.

9. Fully integratable to other systems (ALM, Change management / trouble tickets, build servers, and release managers).

Implementing a solution to deal with these challenges would enable a company to practice proper database automation. Database automation which would be easy to integrate with the rest of change and release processes, to achieve a complete end to end automation.

Once automated continuous integration and continuous deployment the last two steps of automation heaven, are just a decision away from reality.



Summary

The database sets up a real challenge for automation. Scripting database objects change-scripts into traditional version or using 'compare & sync' tools is either an inefficient or plain risky thing to automate, as the two concepts are unaware of the other. A better solution needs to be implemented in the shape of DevOps for database.
DevOps for database should follow the proven best practices of change management, enforcing a single change process over the database, and enable dealing with 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 300 Baker Avenue, suite 300, Concord, MA 01742, USA +1 978.405.3368
EMEA Headquarters 21 Yagiya Kapaim st. Petach Tikva 4900101, Israel +972.3.9248558