Compliant DevOps

WRITTEN BY MATT HILBERT
TECHNOLOGY WRITER AT REDGATE SOFTWARE

CONTENTS

- INTRODUCTION
- STANDARDIZED TEAM-BASED DEVELOPMENT
- AUTOMATED DEPLOYMENTS
- PERFORMANCE AND AVAILABILITY MONITORING
- PROTECTING AND PRESERVING DATA

INTRODUCTION

DevOps has come a long way in a remarkably short time. It first emerged in Flickr's seminal 10+ Deploys Per Day presentation at the 2009 Velocity conference, and it has now moved firmly into the mainstream.

That shouldn't come as a surprise. The 2018 Accelerate State of DevOps Report from DORA shows that the highest performing organizations, which adopt DevOps, release changes 46 times more frequently, have a change failure rate that is 7 times lower, and are able to recover from breaking changes 2,604 times faster.

Crucially, the lead time from committing changes to being able to deploy them is less than one hour in the highest performing organizations — and between one and six months in low performers. Between 46% and 60% of changes deployed by low performers also require some form of hotfix, rollback, or patch.

Database development has also entered the picture because deploying changes to the database is often the bottleneck in software development and slows down releases. To address this, the report investigated which database-related practices help when implementing Continuous Delivery to improve software delivery performance and availability for the first time.

The results revealed that teams that practice Continuous Delivery well use version control for database changes and manage them in the same way as changes to the application. It also showed that integrating database development into software delivery positively contributes to performance, with changes to the database no longer slowing processes down or causing problems during deployments.

The starting block is communication, cross-team collaboration, and visibility, which echoes Redgate's 2018 State of Database DevOps Survey earlier in the year. This showed that 76% of developers are now responsible for both application and database development, and 58% reported their development teams and DBAs work on projects together.

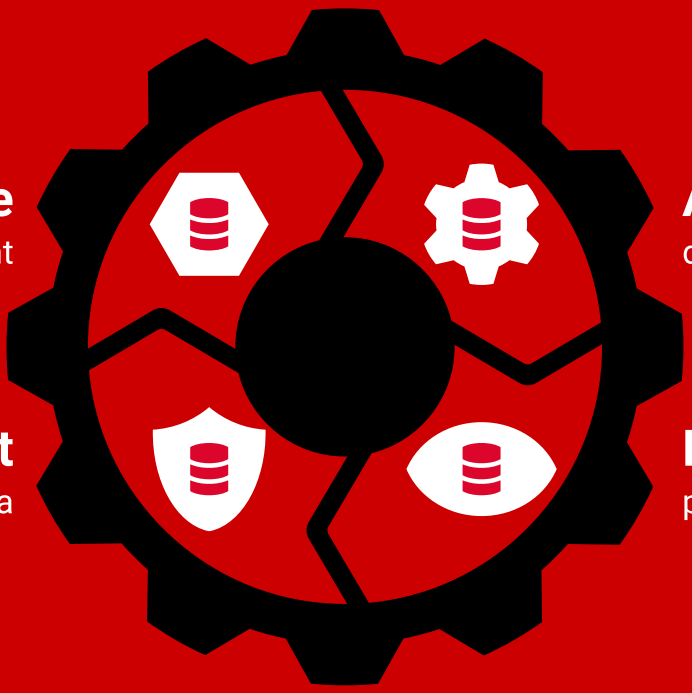
Another element has now emerged, however, driven by the exponential growth of data, the desire by companies to extract more value from it, and the sprawl of data across different databases, various locations, and multiple database copies used in development and testing. This has resulted in data

Compliant Database DevOps

Deliver value quicker while keeping your data safe

Standardize
team based development

Automate
database deployments



Protect
& preserve data

Monitor
performance & availability

Redgate helps IT teams balance the need to deliver software faster with the need to protect and preserve business critical data.

You and your business benefit from a DevOps approach to database development, while staying compliant and minimizing risk.

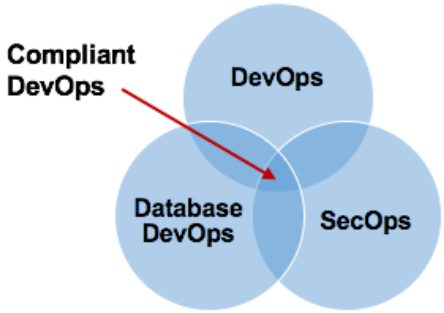
Find out more at
www.red-gate.com/DevOps

redgate

breaches increasing in both frequency and size, along with the misuse of data by companies like Cambridge Analytica.

As a direct consequence, existing data protection regulations such as Sarbanes-Oxley and HIPAA are being joined by new ones that are tougher and demand a lot more from companies in order to demonstrate compliance. The GDPR has already been introduced, affecting any company that handles the data of European citizens, the SHIELD Act is scheduled to come into force in New York in 2019, and it will be joined by the Consumer Privacy Act in California in 2020.

With new data protection laws coming into play, and consumers more aware than ever before of how their privacy is being compromised, there is now a requirement for companies to adopt a compliant DevOps approach. One where protecting data is baked into the software development process from the beginning, by combining the agility of DevOps, the requirement to include the database in DevOps, and the necessity to secure data throughout development:



As can be seen, this approach doesn't replace SecOps. Instead, it makes it easier by automating some of the processes required and making them part of the normal daily routine rather than an extra burden on the team.

Indeed, in its October 2017 report on 10 things to get right for successful DevSecOps, Gartner predicts that by 2021, DevSecOps practices will be embedded in 80% of rapid development teams, up from 15% in 2017.

Note the use of the word "embedded." It was deliberately chosen by Gartner in relation to DevOps because the release-often practice that DevOps encourages has changed the rules of the game. When companies were releasing changes every three or six months, or longer, a review was scheduled at the end of each release cycle for the security team to certify the release and recommend any changes.

With DevOps teams making small changes and releasing them often, this safety window has disappeared, so tools like static code analyzers and open source code vulnerability scanners are now in use to test application code as soon as changes are made.

Include the database in DevOps and this embedded approach also needs to be taken. Fortunately, a number of free and open source initiatives as well as commercial off-the-shelf solutions have emerged over the last 10 years that make database development easier and faster by automating many of the processes involved in four key areas:

- Standardized team-based development
- Automated deployments
- Performance and availability monitoring
- Protecting and preserving data

Importantly, and perhaps conversely, the automation it introduces and the audit trails it provides across the database development cycle ease compliance so that companies can deliver value faster while keeping data safe.

STANDARDIZED TEAM-BASED DEVELOPMENT

Before DevOps arrived on the scene, the wall between Dev and Ops was similar to the division between application developers and database developers. It is someone else's domain, a different coding language is used, and database deployments are problematic at best.

Times have changed. Many application developers are now responsible for database development, too. They switch between coding in C# or Java to writing queries in a database language like T-SQL, used with Microsoft SQL Server. The database DevOps survey mentioned earlier also revealed that 75% of application developers build database deployment scripts, and 47% deploy database changes to production.

In many ways, it had to happen because the faster speed of releases that DevOps encourages means front-end and back-end development are now much more closely connected.

It can also be problematic, however, because developers have different coding styles, writing in a language like T-SQL brings its own challenges, and conflicts can occur with developers working on different branches at the same time.

The key is to introduce collaborative coding, bake in security earlier to prevent issues later down the line and, as the Accelerate State of DevOps Report recommends, put changes to the database into version control.

INTRODUCE COLLABORATIVE CODING

The industry standard language used for relational databases is SQL, of which there are variants like T-SQL for Microsoft SQL Server, PL/SQL for Oracle databases, and the open source PostgreSQL. The examples that follow use T-SQL, but the same principles apply for all versions

A declarative language like T-SQL is relatively easy for software developers to learn because it describes what a program should do, rather than how to accomplish it. That said, it has its own foibles, so Microsoft includes a SQL version of IntelliSense, its context-aware code-completion feature, in SQL Server Management Studio.

There are other free versions available like ApexSQL Complete, as well as paid-for versions like Redgate SQL Prompt, but common to all of them is that they provide hints and suggestions as users type T-SQL, helping to speed up coding and improve its accuracy.

T-SQL is, however, a less strict language than an imperative language like C# or Java, where the sequence and wording of each line of code is critical. This has resulted in developers who code in T-SQL having preferred styles. Take, for example, the following query which is written in three different styles:

COLLAPSED STYLE

```
SELECT CompanyName, AddressType, AddressLine1
FROM Customer
    JOIN CustomerAddress ON(Customer.
    CustomerID=CustomerAddress.CustomerID)
    JOIN Address ON(CustomerAddress.
    AddressID=Address.AddressID)
WHERE CompanyName='ACME Corporation'
```

COMMAS BEFORE STYLE

```
SELECT CompanyName
    , AddressType
    , AddressLine1
FROM Customer
    JOIN CustomerAddress
        ON (Customer.CustomerID = CustomerAddress.
        CustomerID)
    JOIN Address
        ON (CustomerAddress.AddressID = Address.
        AddressID)
WHERE CompanyName = 'ACME Corporation'
```

RIGHT ALIGNED STYLE

```
SELECT CompanyName,
    AddressType,
    AddressLine1
FROM Customer
JOIN CustomerAddress
    ON (Customer.CustomerID = CustomerAddress.\
    CustomerID)
JOIN Address
    ON (CustomerAddress.AddressID = Address.AddressID)
WHERE CompanyName = 'ACME Corporation'
```

CODE CONTINUED ON NEXT COLUMN

These are just three examples, and there are many other styles in use. Some developers prefer plain black type, others hate indents, others like lots of indents, and the argument about commas at the beginning of a line or the end of a line goes on.

All of which can result in confusion, particularly when different developers have worked on the same code base over time. Where teams of developers are updating a database repeatedly, they can collaborate much more easily if all of the code in the database is presented in the same style.

This doesn't have to be as draconian as it sounds. There are tools on the market that let users code in the style they prefer and then change the code to the team's standard style in seconds. That way, the speed at which individual developers code is not affected, but neither is the understanding of the whole body of code confused by lots of different styles in play.

INSIST ON SECURE CODING

The faster speed at which applications are developed using DevOps has seen the end of the security review at the end of long development cycles. This has resulted in security becoming baked into the pipeline with tools like static code analyzers and open source code vulnerability scanners testing code as soon as changes are made.

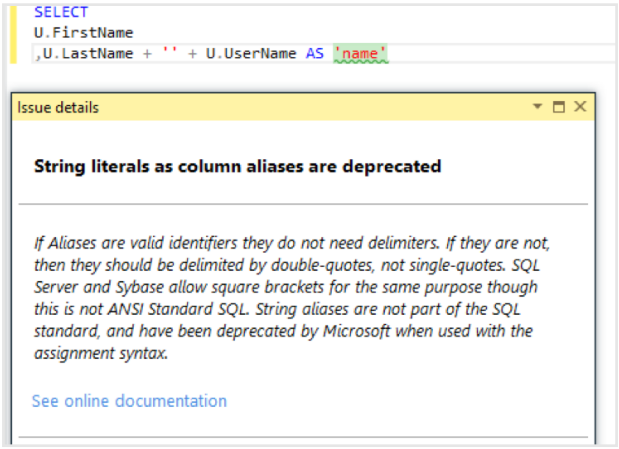
If the speed at which databases are developed is to follow the same route, a similar approach needs to be taken. Security needs to shift left so that errors are caught earlier and the chances of them ever reaching production are minimized.

Just as C# and other languages have "code smells" that, while not necessarily breaking changes, are errors in source code that can have a negative impact on performance and quality, so does T-SQL. 150 common T-SQL code smells can be found in this free eBook.

There are free static code analysis tools available to detect them, the most well-known of which is probably SQL Cop, which highlights potential problems in SQL Server database code that should be investigated.

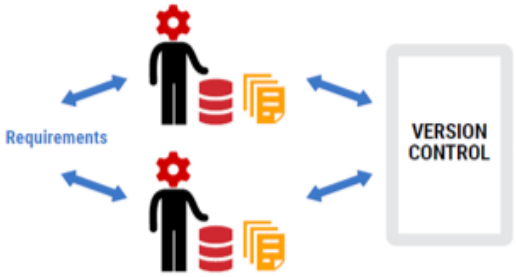
SQL Prompt from Redgate takes a slightly different approach in that it flags up potential errors and pitfalls in code as it is typed,

using a library of over 90 rules behind the scenes. It then explains what the issue is and provides links to online documentation:



This can be particularly useful when first starting to code in T-SQL, or if there are specific rules that have to be followed by everyone on the team. It also provides a quality control gate at the point at which code is written so that before it is even committed, any potential errors have been minimized.

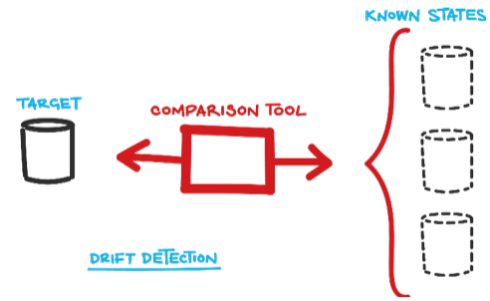
VERSION CONTROL EVERYTHING



Version control is becoming standard in application development and involves developers checking their changes into a common repository during the development process, preferably at the end of each working day. As a direct result, everyone has access to the latest version of the application, one source of truth is maintained, and it's always clear what was changed, when it was changed, and who changed it.

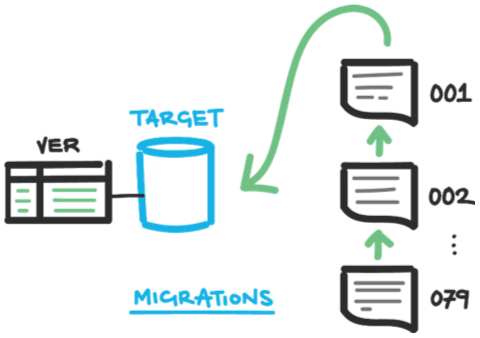
This is just as true for the database code, which can also be version controlled, preferably by integrating with and plugging into the same version control system used for applications. There are two approaches for version controlling databases, however, which appear to be diametrically opposed.

The state-based approach compares the existing database schema with a snapshot (or state) of the target database schema and generates a SQL script that modifies, creates, or deletes objects in the existing database. After running the script, the database will be up-to-date with respect to your latest schema.



This approach doesn't need to know the move from version 3.1 to 3.2 dropped two columns and added one table. It's the job of the comparison tool to discern that in its discovery phase before it generates the script.

The migrations-based approach is at the opposite extreme. Consider that as you develop your application, you create a table, perhaps drop a column, and rename a stored procedure. Each of these changes occurs a step at a time as you develop new code, moving from the old database schema to the new database schema. These steps are called migrations.



The migrations-based approach saves a script of each change and, to update the database to the latest schema, you simply run in sequence all of the migration scripts since your last deployment.

State-based version control allows database code to be held at the component level, with each table, view, or procedure being held separately, making merge conflicts less likely and easier to resolve if they do occur. Migration-based version control gives a much more granular level of control over scripts, which can be viewed and modified as required. The approach that is chosen tends to be based on team size, database complexity, and the amount of refactoring involved.

With large teams or complex databases, for example, there are more likely to be merge conflicts, so the state-based approach is better. Microsoft SQL Server Data Tools (SSDT) can be used but lacks features like the ability to handle reference data.

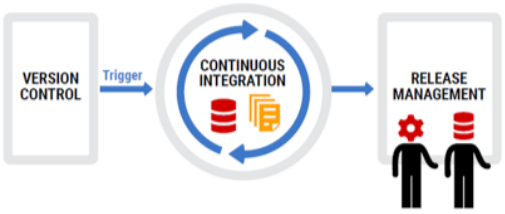
Another option is a tool like Redgate SQL Source Control which, when the correct transition can't be determined, also enables custom change scripts to be substituted for the auto-generated migration script at the object level.

For teams that prefer to evolve databases via a number of refactorings with frequent changes, a pure state-based approach will struggle, so the migrations approach is often preferred. Here, there are many good and proven free open source tools available such as Flyway and Liquibase.

AUTOMATED DEPLOYMENTS

Introducing version control to database development brings many advantages. Ad hoc changes and hot fixes are minimized, reducing the chance the database will drift from its expected state. Every developer works from the single source of truth, so there are fewer errors in the development process. And an audit trail of who made what changes, when, and why is provided, which can be useful in demonstrating compliance.

It also makes the automation that DevOps encourages possible and means the whole development process is more secure.



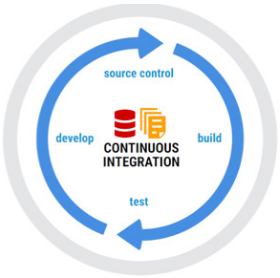
For example, every time a change is committed to version control, a continuous integration process can test the change and flags up any errors in the code. The errors can be fixed immediately and tested again, before the change is then passed along to a release management tool where the change can be reviewed before it is deployed to production.

In this way, the same discipline can be applied to every process that is automated, and all code changes are tested before they are deployed to ensure the production environment is never compromised.

THE FIRST STEP IS CONTINUOUS INTEGRATION

Continuous integration is the process of ensuring that code and related resources are integrated regularly and tested by an automated build system, allowing teams to detect problems early.

Once version control is in place, continuous integration can be used to trigger an automated build as soon as code changes are checked in, with unit tests and early feedback in the form of errors returned. If a build fails, it can be fixed efficiently and re-tested, so a stable current build is always available.



A typical continuous integration server uses a script to execute a series of commands that build an application. These commands can clean directories, run a compiler on source code, and execute unit tests. Where applications rely on a database back-end, those build scripts can be extended to perform the additional tasks of testing and updating the database.

If it sounds like a hard task, it's not. There are already tools out there that plug into existing build servers like Jenkins or TeamCity that, upon each check-in to version control:

- Build and validate the SQL creation script contained in the database package the continuous integration tool needs to deploy the changes.
- Run tests against the database package by generating test data and outputting the results in JUnit XML format.
- Sync the existing database with the latest version in version control.
- Publish the database package to a feed artifact repository ready for deployment.

Any migration scripts that have been checked in for deployment with the database changes are also executed against the target database during this step. The database package, or artifact, that is published will then include the migration scripts alongside a snapshot of a state of the database schema, and any version controlled static data.

This artifact is an important part of the release process because it represents a version validated through testing. It thus becomes a consistent starting point for the release of database changes to subsequent environments.

THE SECOND STEP IS RELEASE MANAGEMENT

Although the continuous integration environment often mirrors the production environment as closely as possible for applications, this is rarely the case for databases.

The artifact published at this stage therefore needs to be deployed against a staging database, which should be an

exact copy of the production database, or as near as possible. This will generate an upgrade script for deployment, and the whole artifact can then be reviewed by the DBA to confirm it is production-ready.

Just as there are many strategies for application deployment, there are a variety of ways to handle database deployment. The three most common are:

FULLY MANUAL

A comparison tool is used to compare the structure of the staging database against the production environment and generate a script for the differences. DBAs then review this script before running the updates against the target environment. This makes change management simple and is often the preferred method when working with smaller databases or less frequent deployments.

AUTOMATED, ONE-CLICK RELEASE USING A RELEASE MANAGEMENT TOOL

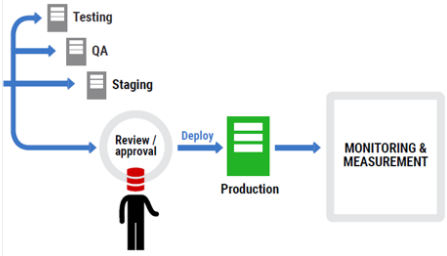
If the release management tool in place uses a NuGet feed as its package repository, there are tools available that can publish the package from the continuous integration server to the release management software. The release management tool can then automate the deployment to production.

MANAGED DEPLOYMENTS USING A STAGING ENVIRONMENT

For DBAs who want more control and have a deeper insight into database deployments, tools have been developed specifically for databases that integrate with release management tools like Octopus Deploy and Bamboo to provide the update scripts, change reports, and review steps needed to make database changes to production efficiently. DBAs can review the changes, check the staging and production environments match, and use the same script to deploy to production.

PERFORMANCE AND AVAILABILITY MONITORING

The speed of release that DevOps encourages puts features into the hands of users faster. By using secure coding, version control, and continuous integration, errors are caught earlier in the development process and the chance of breaking changes reaching production are minimized.



It can happen, however, particularly if changes to the database start reaching production multiple times a day rather than one or two times a month. This is where monitoring becomes an important part of the process.

MONITOR FOR PERFORMANCE

Even after changes have been through testing, QA, and staging, there is still a chance they will cause problems, particularly when databases are under heavy load. So beyond monitoring for memory utilization, file sizes, and growth trends, any monitoring solution should be able to spot queries having an unusual impact, deadlocks, and blocking processes — and be able to drill down in seconds to the cause.

Many companies that have adopted DevOps for the database have also found it useful to share performance monitoring screens with development teams on a permanent basis. That way, the effect that deployments have on performance can be seen as soon as changes hit production.

MONITOR FOR COMPLIANCE

New data protection regulations have moved monitoring up to the next level because organizations are now required to monitor and manage access, ensure data is available and identifiable, and report when any breaches occur.

They need to know and have a record of which servers and what data is being managed and be able to discover the reason for any performance issues quickly and accurately.

Should a data breach occur, it becomes even more crucial because organizations are obliged to describe the nature of the breach, the categories and number of individuals concerned, the likely consequences, and the measures to address it.

This makes an advanced monitoring solution a necessity in most cases in order to monitor the availability of servers and databases containing personal data and be alerted to issues that could lead to a data breach before it happens.

Given the added complexity it brings to monitoring, organizations should look for a solution that offers the extra capability but makes taking advantage of it easier — for example, by allowing all SQL Server instances, availability groups, clusters, and virtual machines to be viewed on one central web-based interface. And by having customizable alerts that be configured to suit SQL Server estates of any size and complexity.

PROTECTING AND PRESERVING DATA

Including database development in DevOps enables the full

advantage of DevOps to be realized without the database acting as a bottleneck. The new requirement for Compliant DevOps, however, which requires data to be protected all the way through the development process, adds another factor.

The 2018 Database DevOps Survey showed that 67% of developers want a copy of the production database in their development, test, or QA environments to ensure changes will work once deployed. Yet those same production databases invariably contain sensitive data that needs to be protected.

DON'T RELY ON ANONYMOUS OR LIMITED DATA SETS

One solution is to have a version of the production database with a limited dataset of anonymous data that is always used to develop and test against. This does, though, mean testing changes against a database that is neither realistic, nor of a size where the impact on performance can be assessed.

Another solution is to take a copy of the production database and mask the data manually by replacing columns with similar but generic data. This copy can then be used in development and testing, but will age very quickly as ongoing changes are deployed to the production database.

This is where data masking tools, which pseudonymize and anonymize data, are now being adopted to provide database copies that are truly representative of the original and retain the referential integrity and distribution characteristics. Perhaps unsurprisingly, Gartner's 2018 Market Guide for Data Masking predicts that the percentage of companies using data masking or practices like it will increase from 15% in 2017 to 40% in 2021.

TAKE ADVANTAGE OF VIRTUALIZATION TECHNOLOGIES

Gartner's guide also recommends that data masking solutions should provide tools to manage the full lifecycle of masking data. It points in particular to data virtualization, which, when used in tandem with static data masking, can speed up the provisioning of database copies while also reducing the storage space required.

Written by **Matt Hilbert**, Technology Writer at Redgate Software

Matt Hilbert has 20 years' experience working for lots of the world's biggest tech companies – and many of the smallest. He has a particular fascination for emerging technologies and is on a continuing mission to decompile, untangle, and explain techspeak to a wider audience, and excite people about the endless possibilities technology offers.

DZone, Inc.
150 Preston Executive Dr. Cary, NC 27513
888.678.0399 919.678.0300