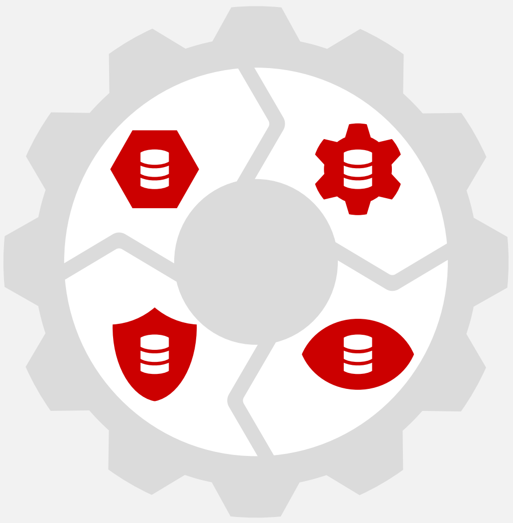
Whitepaper

Compliant Database DevOps

How teams can balance the need to deliver software faster with the requirement to protect and preserve data



Introduction

Database DevOps is now a thing, as demonstrated by its inclusion for the first time in DORA's long-running **Accelerate State of DevOps Report**. Now in its fifth year, the 2018 report calls out database development as a key technical practice which can drive high performance in DevOps

With new data protection laws coming into play, however, and consumers more aware than ever before of how their privacy can be compromised, there is also a requirement for companies to go one step further and 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 desire to include the database in DevOps, and the necessity to secure data throughout development.

This whitepaper outlines how organizations can introduce compliant database DevOps by transforming some 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 and audit trails this provides across the database development cycle eases compliance so that organizations can deliver value faster while keeping data safe.

Contents

Standardized team-based development

The way databases are developed has changed over the last few years. Once the sole responsibility of Database Administrators and database developers in siloed teams, application developers are increasingly expected to write the code for the database as well as the application

This was highlighted in Redgate's **2018 State of Database DevOps Survey**, which revealed that 76% of respondents had developers in their team who work across both applications and databases, and 75% also build the database deployment scripts.

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

The survey also showed, however, that the greatest challenge to integrating database changes into the DevOps process was synchronizing them with application changes, and overcoming the different approaches to development within multi-function teams.

The solution is to introduce collaborative coding, bake in security earlier to prevent issues later in the development pipeline and, as DORA's DevOps Report recommends, put changes to the database into version control.

Collaborative coding

Application developers typically use an imperative language like C# or Java, where the sequence and wording of each line of code is critical. Relational databases, on the other hand, use a declarative language like T-SQL, which describes what a program should do, rather than how to accomplish it.

Because the syntax is not as strict, developers who work with T-SQL often have preferred styles. Some, for example, prefer plain black type rather than seeing type in different colors, others hate indents, and arguments about whether commas should be at the beginning or the end of a line go on.

All of which can result in the code behind a database being muddled and difficult to understand, 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.

SQL Prompt from Redgate can help here because it includes a useful feature where users can code in the style they prefer and then change the code to the team's standard style with a couple of keystrokes. 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.

Secure coding

The introduction of DevOps to application development saw security shift left in the development process. Rather than relying on a security review at the end of the development cycle, tools like static code analyzers now test code as soon as changes are made, catching errors earlier and minimizing problems at deployment.

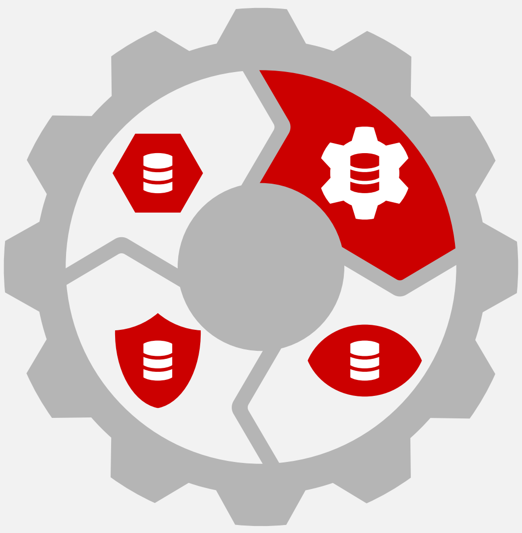
Those code analyzers look for 'code smells' – errors in code which can affect performance, as well as those which could be breaking changes. T-SQL has its own share of code smells and SQL Prompt flags up potential errors and pitfalls in code as it is typed, using a library of over 90 rules behind the scenes, and 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 issues have been minimized.

Version control

Version control is standard in application development and involves developers checking their changes into a common repository frequently during the development process. As a direct result, everyone has access to the latest version of the application 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 should also be version controlled, preferably by integrating with and plugging into the same version control system used for applications. Developers might add a new table or stored procedure, amend a column name, or update a foreign key constraint, for example. They can also change the static data the database needs to function, like lookup values, constants or settings. By capturing the changes in version control, one source of truth can be maintained, reducing the chance of problems arising later in the development process, and also providing an audit trail of every change that has been made.



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 same source of truth, so there are fewer errors when different development branches are merged. And an audit trail of who made what changes, when, and why is provided, which can be useful in demonstrating compliance.

It also opens the doors to the automation that DevOps encourages, while making the development process more secure.

Continuous integration

Once version control is in place, continuous integration (CI) can be used to trigger an automated build as soon as code changes are checked in, which tests the changes and flags up any errors in the code. If a build fails, it can be fixed efficiently and re-tested, so a stable current build is always available.

A typical CI server in application development executes a series of commands that build the application. These commands clean directories, compile the source code, and execute unit tests. Where applications rely on a database back-end, the CI server can also perform the additional tasks of testing and updating the database.

Redgate's SQL Change Automation, for example, works with any CI server that can run PowerShell. There are also extensions for CI build servers like Azure DevOps, Jenkins or TeamCity and, on each check-in to version control (or however often it is set to run), they can take care of the whole database CI process.

The artifact that is published at the end of the process will include the deployment script with updates to the database schema and any version controlled static data. It also includes additional information to make sure the release is safe, including any warnings that might result in data loss, a detailed diff report so you can see exactly what objects are being changed and how, and information to check that the schema hasn't drifted since the deployment script was created.

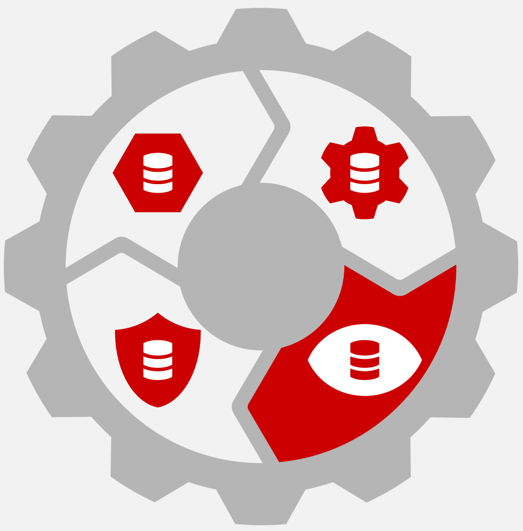
This artifact is an important part of the release process because it represents a version that has been validated through testing as part of the CI process, and is a consistent starting point for the release of database changes to subsequent environments.

Release management

Although the CI environment often mirrors the production environment as closely as possible for applications, this is rarely the case for databases. The artifact published at the CI 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 the deployment script for that environment, and the whole artifact can then be reviewed to confirm it is production-ready. A lot of teams still have a manual approval step in place for a Database Administrator to review and give the final sign-off before going to production. It's easy to do this in a release automation system, which provides another audit trail of the changes and who approved them.

SQL Change Automation also integrates with any release automation system that can run PowerShell, and there are extensions for Octopus Deploy and Azure DevOps to make the setup easier.

This gives organizations the option to include the release of database changes with the workflow already in place for the application, rather than have to introduce a new, unfamiliar one.



Performance and availability monitoring

It is normal practice to monitor databases to keep an eye on factors like memory utilization, I/O bottlenecks and growth trends. The increase in the size and complexity of SQL Server estates has already prompted many organizations to introduce third party tools to give them a wider and deeper picture. Adding DevOps to the equation makes it even more important.

While the automation which DevOps introduces to many parts of database development minimizes errors and gives much better visibility across the whole process, there is a flipside. Instead of releasing changes to the database once or twice a quarter, changes can now be released at any time. It is therefore important to monitor your system to understand if any releases are causing a problem so that they can be fixed quickly.

Performance monitoring

If databases are updated more often, performance monitoring becomes crucial. Even though the changes will probably be the small, iterative ones which DevOps encourages, there is still a chance they will cause problems when they are deployed, particularly if databases are under heavy load or there are differences between environments.

Given the importance the database has to many business operations, organizations 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 which have adopted DevOps for the database have also found it usual 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. Redgate's SQL Monitor, for example, shows the database releases on the same timeline with the performance information so it's very easy to see if a release had a negative impact and recover from this quickly.

Compliance monitoring

New data protection regulations now require organizations to monitor and manage access to personal data, ensure data is identifiable, and report when any breaches occur. This makes an advanced monitoring solution a necessity in most cases, in order to monitor the availability of servers and databases containing personal data.

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. By, for example, 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 the database in DevOps enables the full advantages of DevOps to be realized without the database causing 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.

Redgate's 2018 State of 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. This helps find problems sooner before they get to production, yet those same production databases invariably contain the sensitive data that needs to be protected.

Masking data

Some organizations get around the problem by having a version of the production database with a limited dataset of anonymous data to develop and test against. Testing changes against a database that is neither realistic, nor of a size where the impact on performance can be assessed, can lead to problems during deployments, however.

Other organizations mask the data in a copy of the production database by replacing columns with similar but generic data, but this will age quickly as ongoing changes are deployed to the live 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. Indeed, 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.

Provisioning masked database copies

While data masking can help when provisioning copies of production databases for use in development and testing, it can also lead to resource issues. It is not unusual for databases to be 1TB in size or more, and provisioning copies to multiple developers can take up a lot of time as well as space.

This is where the tried and tested virtualization technologies built into the Windows operating system come into place. SQL Clone from Redgate uses it to create copies, or clones, of databases in seconds which, while only around 40MB in size for a 1TB database, work just like normal databases and can be connected to and edited using any program.

Redgate integrates this with data masking capabilities to provision masked database copies with a process that is simple, fast repeatable, transparent, and auditable.

Summary

These are interesting – and challenging – times for database development.

DevOps has entered the conversation and provided a route to removing the database as the bottleneck in delivering value. By encouraging collaboration and integration, and moving to releasing small changes, often, database deployments can move from worrying, infrequent problems to a standard part of the development and release workflows.

Data privacy and protection concerns, however, have interrupted the conversation with the GDPR now being joined by the upcoming Stop Hacks and Improve Electronic Data Security (SHIELD) Act in New York, the Consumer Privacy Act in California, and India's Personal Data Protection Bill, among many others.

This whitepaper demonstrates, however, that DevOps and data privacy do not need to oppose each other. Rather, they can complement one another. The automation and audit trails that DevOps processes introduce to database development can ease compliance with data protection regulations and enable organizations to balance the need to deliver software faster with the requirement to protect and preserve personal data.

Compliant
Database DevOps

To find out more about the benefits of Compliant Database DevOps go to **www.red-gate.com/DevOps**