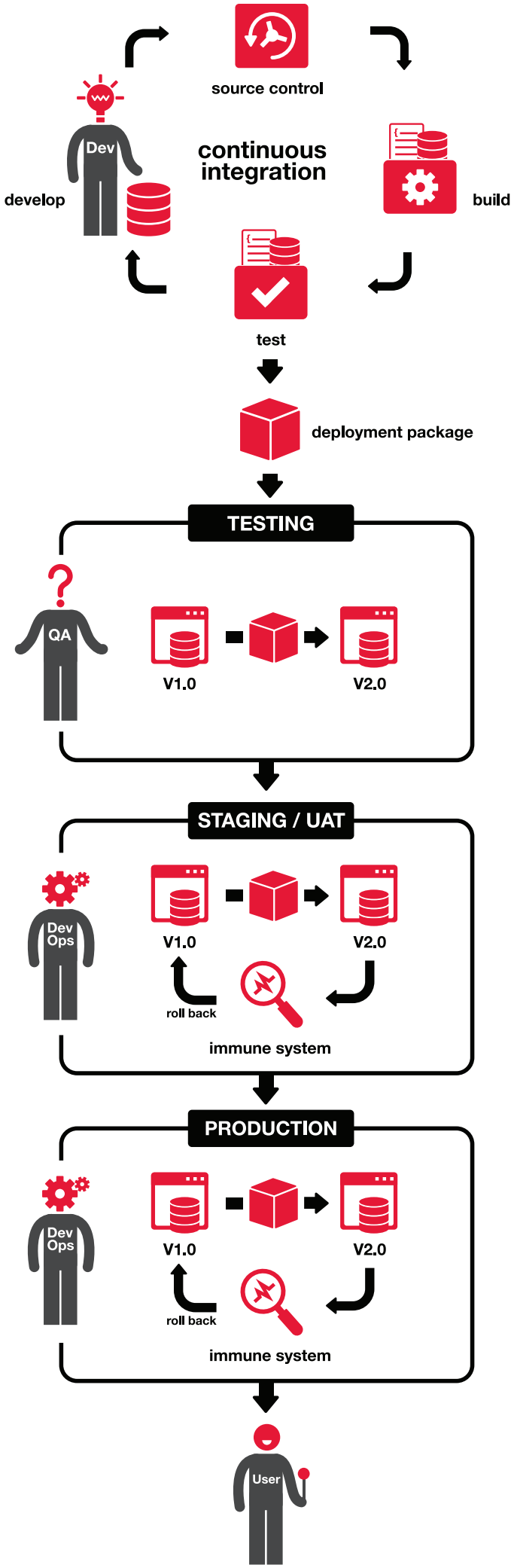
Whitepaper

Continuous integration for databases using Redgate tools

An overview

redgate

ingeniously simple


source control
continuous integration
develop
Dev
build
test
deployment package
TESTING
QA
V1.0
V2.0
STAGING / UAT
Dev Ops
V1.0
V2.0
roll back
immune system
PRODUCTION
Dev Ops
V1.0
V2.0
roll back
immune system
User

Continuous integration for databases using Redgate tools
An overview

Contents

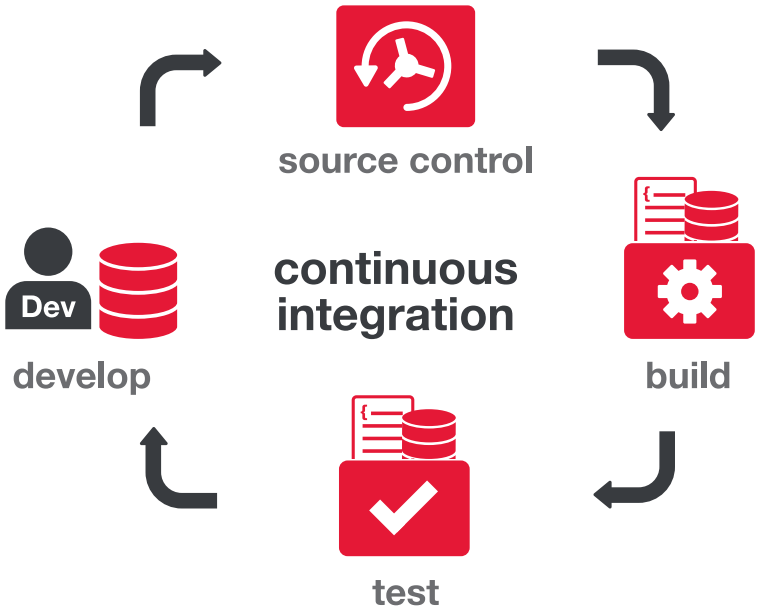
Why continuous integration?

> *"The key word to associate with a continuous integration server is 'automation'. You need to set up automatic deployment and automatic testing so that you can validate the build and deployment without involving a single human being. This provides a solid early warning system."*
>
> **Grant Fritchey, SQL Server MVP and Product Evangelist at Redgate**

Continuous integration (CI) is the process of ensuring that all code and related resources in a development project are integrated regularly and tested by an automated build system. Code changes are checked into source control, triggering an automated build with unit tests and early feedback in the form of errors returned. A stable current build should be consistently available, and if a build fails, it can be fixed efficiently and re-tested.

A CI server uses a build script to execute a series of commands that build an application. Generally, these commands clean directories, run a compiler on source code, and execute unit tests. However, for applications that rely on a database back-end, build scripts can be extended to perform additional tasks such as testing and updating a database. It's this process of generating, testing, and synchronizing the database build scripts that forms continuous integration for databases – the subject of this paper.

The following diagram illustrates the fundamentals of a typical integration process – both for applications and their database back-end. The automated continuous integration process begins each time the server detects a change that has been committed to source control by the development team. Continuous integration ensures that if at any stage a process fails, the 'build' is deemed broken and developers are alerted immediately.



CI originated from the Extreme Programming (XP) movement and is now an established development practice.

> *"Continuous Integration is a practice designed to ensure that your software is always working, and that you get comprehensive feedback in a few minutes as to whether any given change to your system has broken it."*
>
> **Jez Humble, ThoughtWorks, co-author of Continuous Delivery**

For many software projects, this will include a database. Author and thought leader Martin Fowler recommends that "getting the database schema out of the repository and firing it up in the execution environment" should be part of the automatic build process. However, this is not always simple, which is why this paper seeks to clarify the process of bringing continuous integration to database changes.

The database delivery lifecycle

Database CI is an important part of a wider database delivery lifecycle. Continuous integration for databases encompasses three main areas of activity:

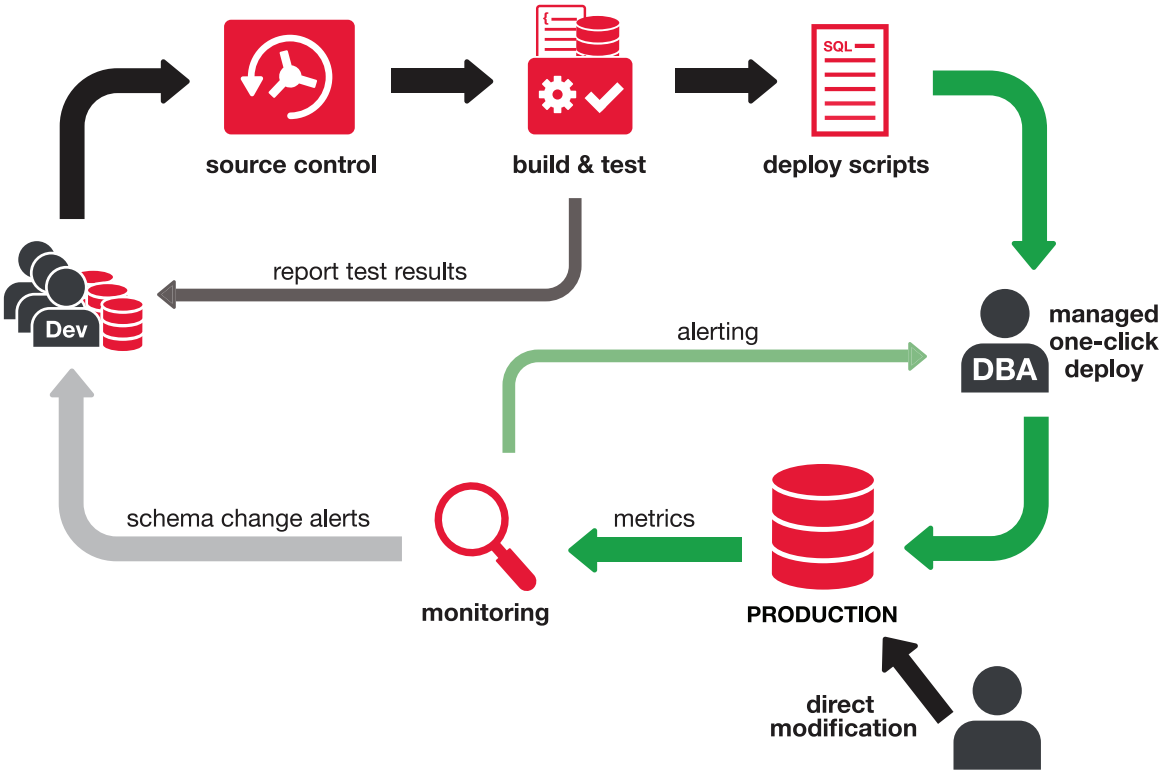
1. Ensuring the database is in a valid state after changes have been made
2. Running tests against changes
3. Synchronizing a target database so that it is updated with changes

These activities together mean that continuous integration can catch problems that might otherwise lie undetected as they pass through to the DBA for deployment to production.

Continuous integration is the next natural step once database changes have been committed to source control. It's the early testing gateway for development teams and proves that all database changes committed can be built successfully. This is an important milestone - a test in its own right - in the development of database changes. If changes fail to build successfully, you know that there's something you need to fix proactively before deploying to pre-production environments. You can also add automated database unit tests as another testing stage for your database changes to pass, as part of your continuous integration process. The CI build server will show you any tests that fail so you can take steps to resolve any issues.

Once changes have passed continuous integration and pre-production testing, and are deemed fit by the DBA to go to the production server, it's good practice to use monitoring techniques to ensure that not only the performance of the database remains acceptable, but also that the end user is benefiting from the changes.

Another example of monitoring to consider is monitoring for any unexpected schema changes being made directly to the production database, in the form of database drift. These unexpected changes have likely bypassed rigorous development and test processes and require an action. Monitoring for these gives the DBA and development manager the information to review whether any of these changes need to be undone or, once approved, copied into the development environment.



The challenge of bringing continuous integration to database changes

In contrast to applications, databases contain state and this needs to be preserved after an upgrade. There is no source code to compile. Where a production database already exists, DML and DDL queries modify the existing state of a database. Migration and deployment therefore rely on creating upgrade scripts specifically for that purpose.

The lack of database source code makes it more complicated to maintain a current stable version in source control. Creation and migration scripts can be checked into the source control repository, but despite its importance, the disciplined creation and on-going maintenance of these scripts is not always considered to be a core part of the database development cycle.

Where changes are deployed to an existing database, all differences and dependencies must be accounted for. In some production deployments, this involves multiple targets with different schemas and data. In either case, this manual process is time-consuming, prone to errors, and should not be left unresolved at the end of the project cycle.

Object creation scripts can be generated relatively simply (for example using Microsoft SQL Server Management Studio), but referential integrity is difficult to maintain. Objects and data must be created and populated in the correct order, and as dependency chains can be complex, third-party tools are often required to save time and reduce potential for human error.

These are the challenges to consider when planning continuous integration for your databases. Overcoming them means you give yourself extra safety nets as you develop your databases. If database changes are deployed directly to production environments, there's a real risk of downtime and data loss. If, thanks to the ongoing building and testing of changes in continuous integration, bugs and errors are discovered earlier, the risk that they could bring the production server down or cause data loss further down the line is substantially lower.

Database continuous integration

Continuous integration for databases can involve multiple processes – generating a build artifact, running unit tests against database code, updating an existing database with the latest version in source control, and packaging the database changes for deployment.

In the case of using Redgate tools for continuous integration, a NuGet package is generated as the build artifact. It is this package that is used as the input for further CI tasks – testing, synchronization, and publishing the changes. A database package contains a snapshot of the state of the database schema, including the structure of the database and any source-controlled static data. This package can be used to update a target database to match the state of this package. Alternatively, a database can be compared to a package to ensure that the database is in the desired state.

Databases may feature in your CI process simply because the application code requires there to be a database present to function correctly. The database schema version corresponds to an analogous application code version. Any changes to the application code or the database structure could in theory break the system and should consequently trigger the CI process.

If you already have internal test databases that need to match the development databases, you can keep them up-to-date with the latest version using continuous integration. Once a database is maintained in source control, Redgate tools are able to build a clean database from its source files to accompany the application build.

Although it is best practice to test application code as part of a CI process, database code and its accompanying business logic is often overlooked. Database code comes in the form of stored procedures, functions, views, triggers, and CLR objects. If it is deemed important to test application code as part of CI, the same must apply to the database code.

Fortunately, there are many open source frameworks that can be used for these purposes, some implemented in .NET (e.g. NUnit) and others in SQL (e.g. the popular open source SQL Server unit testing framework tSQLt). Unit tests can be easily created, run, and managed with Redgate SQL Test, a SQL Server Management Studio add-in.

The advantage of database continuous integration even without unit tests

Even if you don't have a set of database unit tests to run against your changes as part of the continuous integration process, your team can still benefit from the fundamentals of continuous integration – ensuring that your database changes are automatically tested to see if they build successfully as soon as changes are committed to version control. This simple step of integrating your version-controlled changes with a CI build server is the equivalent of testing whether the source code compiles in the application world.

Take the change of renaming a table as an example for a database. If a view refers to that table and the name of that table is not updated in the view, the view becomes an invalid object when you try to deploy the database as a whole. If your application uses this view in the database back-end, your application will break. Without the fundamentals of database continuous integration in place, some time-consuming detection would be required to assess whether the issue is related to your application or database. However, if you have a CI build server automating the database build on every check-in to source control, you can detect invalid objects, fix the issue, commit your new changes, and ensure the build is successful the next time.

Redgate SQL Prompt can help you find invalid objects in your code from SQL Server Management Studio. You can review the results of the search, the SQL creation script for each object, and open the script as an ALTER statement.

Using custom migration scripts in database continuous integration

Automated deployment script generation with schema comparison tools such as Redgate SQL Compare is powerful and time-saving. However, the comparison engine that generates the deployment scripts has no way of interpreting developer intent. The most common cause of a broken deployment script is as a result of development changes that include data migrations and object renames. This is an important consideration as you continuously integrate database changes.

If, for example, a column is renamed, the comparison engine only sees the before and after state, and will interpret this as a DROP and CREATE, leading to disastrous consequences should this script be inadvertently applied to the production database.

There are a number of circumstances where developer intent is required to supplement the comparison engine knowledge, some of which are listed below:

- Renaming tables and columns
- Splitting tables and columns
- Merging tables and columns
- Adding a new NOT NULL column to a table without supplying a default value
- Refactorings that include data migrations

Fortunately, Redgate SQL Source Control provides the capability to save custom migration scripts, which are then re-used by the comparison engine to generate reliable and repeatable deployment scripts. This capability makes the continuous validation of the upgrade process possible in an automated CI environment.

Deploying database changes to pre-production environments

Once validated in a CI environment, the database (and application) changes need to go through further testing before releasing to production.

In many ways continuous integration can be considered a dry run for deployment to production. But although the CI environment often mirrors the production environment as closely as possible for the application, this is rarely the case for the database. Production databases can be huge, and it is therefore impractical to restore a production backup to the CI environment for testing purposes. Lengthy restore times make recreating the CI database impractical. Moreover, test environments rarely benefit from the same storage capacity as for production and pre-production environments. It is therefore prudent to push these changes from the CI process through some final test phases using these pre-production environments, for example staging environments and UAT.

Redgate command line tools in the DLM Automation Suite help transition code and database changes through the pre-production testing stages of the release process. You can either synchronize the tested output of the CI process, encapsulated as packages, with the databases in your testing environments, or you can publish the package to a package feed, ready for a release management tool to deploy.

How Redgate tools help

Redgate offers the following tools to support the CI and delivery process. Many of these tools feature in the worked examples in the next section. See www.red-gate.com/products/dlm for more information.

SQL Developer Suite

The SQL Developer Suite contains the end-user tools you need for database continuous integration in a single installer.

SQL Source Control

- Commit schema and data changes to any version control system from within SQL Server Management Studio
- Inspect database version history and access specific revisions
- Store custom migration scripts in source control

SQL Compare Pro and SQL Data Compare Pro

- Create a database from source files in version control
- Generate schema and data deployment scripts
- Validate that two databases are identical
- Generate pre-/post-deployment reports for troubleshooting

SQL Test

- Create, run, and manage tSQLt unit tests on databases
- Lay foundations for automated testing as part of your CI process*

SQL Data Generator

- Generate realistic test data based on your existing schema
- Use alongside SQL Test as part of your CI process*

SQL Doc

- Generate documentation for your database
- Enable automation of documentation as part of your CI process*

* Requires DLM Automation Suite

DLM Automation Suite

The DLM Automation Suite contains SQL CI, which powers the command line versions of the tools needed for database continuous delivery. These are installed on your build agents and work with any CI server.

SQL CI

SQL CI is part of the DLM Automation Suite and uses the command line versions of SQL Compare and SQL Data Generator to perform the four continuous integration tasks:

- **Build** – generate a package and check that this database contains valid objects
- **Test** – generate test data using SQL Data Generator and run tSQLt tests against the packaged database
- **Sync** – update an existing database to match the version in the package
- **Publish** – publish the package to a NuGet feed, ready for a release management tool to deploy to other environments

The **Test**, **Sync**, and **Publish** steps are optional, but require the **Build** step to be performed first in order for a package to be generated.

You can run these stages multiple times, for example to test on different versions of SQL Server, or to synchronize multiple databases.

You can integrate SQL CI with TeamCity and TFS Build using the plugins and scripts included, or any other CI tool through the command line interfaces.

Licensing

Please contact dlm@red-gate.com or visit www.red-gate.com/products/dlm for information about licensing options for these tools. A free trial is available for all Redgate tools.

Worked examples

This section illustrates worked examples for making databases part of a continuous integration process using the Redgate command line tool, SQL CI, included in the DLM Automation Suite. You can integrate SQL CI's four default steps into your chosen CI build server to build, test, synchronize, and publish a NuGet package quickly and easily:

Build – generate a package and check that this database contains valid objects

Test – generate test data using SQL Data Generator and run tSQLt tests against the packaged database

Sync – update an existing database to match the version in the package

Publish – publish the package to a NuGet feed, ready for a release management tool to deploy to other environments

The NuGet package generated at the **Build** step is our build artifact that is then used as the input for the remaining three steps in SQL CI.

With the integration of SQL CI with build servers such as TeamCity and TFS Build, these steps can be readily automated to run on every check-in.

Continuous integration typically starts with a check-in to the version control system. The CI server copies the latest version of the repository to the build agent, which contains both the application code and database scripts.

1. **Use the SQL CI Build step on check-in of changes to source control to trigger the creation and validation of the database state and generate the NuGet package:**

```
sqlCI.exe build /scriptsFolder=<DatabaseScriptsFolder>
/packageId=<PackageId> /packageVersion=<buildNumber>
```

This step verifies that the committed database state is valid and deployable. Here 'deployable' means that there are no invalid objects and that it can be deployed by SQL CI to a temporary database on the specified server.

If this deployment is successful, a NuGet package is created and named <PackageId>.<buildNumber>.nupkg.

By default the **Build** step uses a temporary LocalDB database, accessed via Windows authentication. If you prefer to use a SQL Server instance instead, add the `/temporaryDatabaseServer=` parameter. If you need to use a SQL Server account, add the `/temporaryDatabaseUserName=` and `/temporaryDatabasePassword=` parameters.

2. **Use the SQL CI Test step to run tSQLt tests as an additional step in the continuous integration process:**

```
sqlCI.exe test /package=./examplePackage.2.3.nupkg
```

This step runs any existing tSQLt database unit tests that have been checked into source control. SQL Test makes it easy to develop unit tests for SQL Server databases, using the open source tSQLt framework (http://tsqlt.org).

By default, the **Test** step uses a temporary LocalDB database, accessed via Windows authentication. If you prefer to use a SQL Server instance instead, add the `/temporaryDatabaseServer=` parameter. If you need to use a SQL Server account, add the `/temporaryDatabaseUserName=` and `/temporaryDatabasePassword=` parameters.

Once the tests have been run as part of the SQL CI **Test** step, results will be generated in a JUnit format of report.

3. **Use the SQL CI Sync step to update an existing database to match the version in the package:**

```
sqlCI.exe sync /databaseServer=<server>
/databaseName=<databaseName> /package=./examplePackage.2.3.nupkg
```

The **Sync** command updates an existing database with the version in the package and may be used to update your integration testing environment with your database changes.

By default the **Sync** step uses Windows authentication. If you need to use a SQL Server account, add the `/username=` and `/password=` parameters.

4. **Use the SQL CI Publish step to publish to a NuGet feed for your release management tool:**

```
sqlCI.exe publish /package=examplePackage.2.3.nupg
/nugetFeedUrl=http://your-nuget-server/nuget
/nugetFeedApiKey=password
```

If your release management tool uses a NuGet feed as its package repository, you can publish your package from your CI server to the release management software using SQL CI **Publish**. You can then use your release management software to automate deployments to subsequent environments.

You can configure your release management tool to run further tests on your database changes as it deploys them to Staging or Production. You can also build in a manual approval step in your deployment process to ensure your DBA or operations team can review any database changes before they are deployed to Production.

Conclusion

This whitepaper has outlined important considerations for continuous integration as part of your database change management processes. It has also included worked examples for bringing databases into your CI development process, using Redgate tools.

Version controlling your database code is the very first step on the path to continuous delivery of your database changes. This vital step ensures that there is one source of truth for your CI build server to work from, enabling every committed change to be built and tested.

Once you have your database under version control, Redgate SQL CI works in tandem with your CI build server to continuously integrate and test each change committed. A database package can be created as part of the CI process and, with minimal effort, deployed through multiple environments using your chosen release management tool.

Further reading and resources

Visit **www.red-gate.com/products/dlm** for further resources on database delivery and to download free trials of the Redgate tools mentioned in this whitepaper.

Educational resources are available in the **DLM Patterns and Practices Library** (**www.red-gate.com/simple-talk/learning-program**), a bank of free online content covering patterns and practices, plus tutorials for Redgate tools across four key learning stages to help you implement continuous integration and delivery for your databases:

Manual
No source control, or manual, ad hoc source control

Source control
Automated source control solution

Continuous integration
Source control and CI for the database

Continuous delivery
Source control, CI, and automated deployment

Monitoring
Drift checking and performance monitoring

You can also watch online videos about database continuous delivery on the Redgate YouTube channel.

If you've got any questions, or if you'd like a demo of how you can set up database lifecycle management for your team, just get in touch: dlm@red-gate.com.