WHITEPAPER

Continuous Integration for databases using Redgate tools

Contents

“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.”

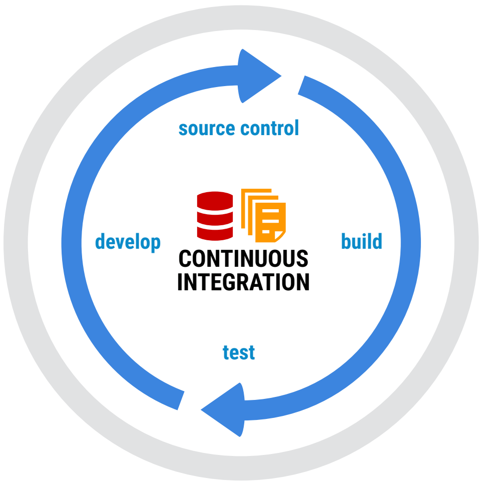
Grant Fritchey, SQL Server MVP

Why continuous integration?

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.



"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

All software applications consist of components that are subject to change. Each time any component changes, we must perform a complete build of all parts of the application that rely on that component, so that we can prove, continuously, that nothing has been broken. While the majority of applications depend on one or more databases, those databases also have their own hierarchy of components on which they depend. Just as for an application, any change to the database code or components should be verified by an automated database build and subsequent testing to allow teams to detect problems early.

Continuous integration for databases

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

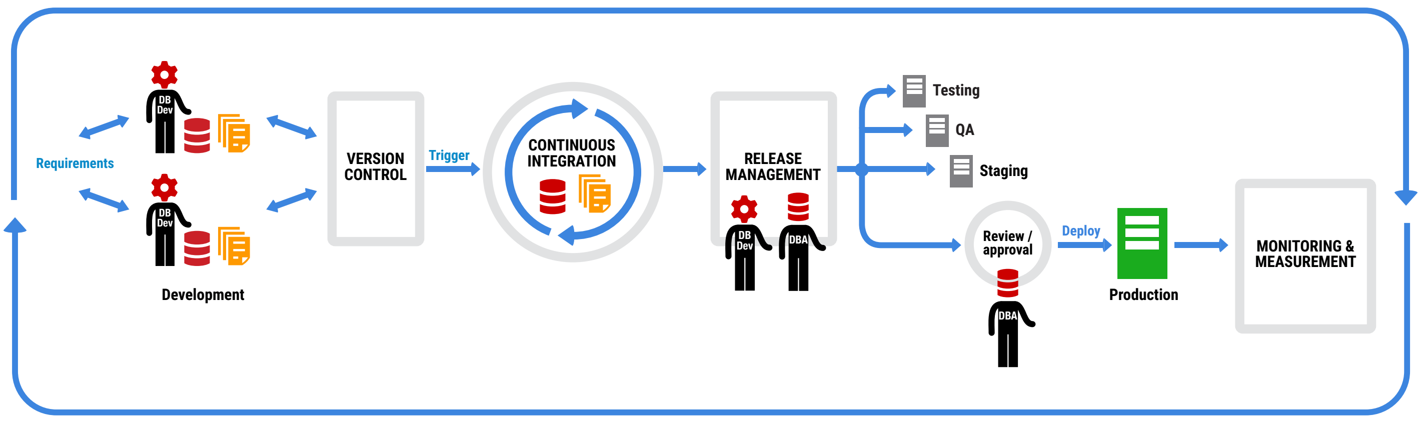
These activities together mean that continuous integration can catch problems that might otherwise lie undetected as they pass through the different environments before being deployed to production.

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 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 provides an opportunity 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 DLM Automation helps 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.

Conclusion

This whitepaper has outlined important considerations for continuous integration as part of your database change management processes.

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 DLM Automation 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

These articles on Simple Talk dive deeper into the process of applying continuous integration to databases:

Database Continuous Integration by Grant Fritchey
Implementing Continuous Integration for Databases by Sjors Takes

Further articles on continuous delivery, and related topics, are also available in our free Simple Talk library

If you have any questions about setting up database continuous integration for your team, please email databasedevops@red-gate.com

To learn more about the benefits of extending DevOps practices like CI to SQL Server databases, please visit www.red-gate.com/solutions