WHITEPAPER

Compliant Database DevOps for Oracle with Redgate

Compliant Database DevOps

redgate

Abstract

Building great software is never just about the code. It's also about managing multiple teams, timelines, and frequently changing business or customer requirements. As organizations face increasing pressure to create and release software more rapidly, DevOps has emerged as an effective way to help teams collaborate and speed up the delivery cycle.

At the heart of the DevOps approach is the concept of continuous delivery. Coupled with the cultural change that DevOps requires, continuous delivery enables teams to work together to produce software in short cycles so they no longer need to rely on 'big bang' releases to provide value to customers. Instead, they have reliable and repeatable processes in place to provide a steadier stream of more frequent deployments, both increasing efficiency and reducing risk in the software release process.

One of the biggest advantages is in automating the repetitive development and testing processes that database development teams use to deliver, manage, and maintain the database. From version controlling changes to deploying them to different environments, and, when ready, choosing to deploy to production, continuous delivery helps teams reduce risk and increase both efficiency and reliability in the software release process.

This whitepaper explains how you can apply DevOps practices like continuous delivery to databases, and how Redgate tools for Oracle can support you along the way.

Contents

There's more to DevOps than tooling but it can be a good place to start

Sharing common processes and tooling can be one of the best ways to open up communication channels between development and operations teams, paving the way for a culture of collaboration.

Redgate offers a suite of Oracle tools that work alongside best of breed third party software, including CI build servers and release management tooling. These help to automate processes and make it possible to safely and quickly deliver database changes alongside application code. Support is available for the full database lifecycle from connected development, versioning, to automating the build, and deployment of database changes.

"DevOps is not all about automation but it enables fast feedback loops and encourages a culture of collaboration."

James Betteley, Head of Education, DevOpsGuys

Source control is the first step to continuous delivery and brings immediate, lasting advantages

Version controlling, or source controlling, database changes is the first – and vital – step to better database development practices. It ensures database development teams communicate their changes with others in the team, always have a version to roll back to if required, and maintain a solid audit trail. With the ability to share code, multiple people and teams can access pieces of code, or a database, at the same time.

Introducing source control for databases, however, can be problematic if new tools enforce strict, unfamiliar procedures and compel database developers to work in a different way from application developers. This can make the divide between them wider rather than bridging it.

The key is to integrate with existing application version control systems like Git, Azure DevOps or Subversion, thereby making the most of the knowledge that already exists inside organizations.

Source Control for Oracle, for example, allows developers to check in changes made directly in their development environments using their preferred IDE. Behind the scenes it scripts out files that represent the new state of each object and saves them to whichever version control system is in use.

It also enables developers to check in static data, so teams can track any changes to, and migrate, any static data required for an application to function. (This might typically be non-transactional data that is updated infrequently, like a table of US states.)

“Every proposed change to your systems, whether to an application, your infrastructure, your database schema, or your build, test and deployment process itself, should be made via source control.”

Jez Humble, Continuous Delivery and ITIL: Change Management

Continuous Integration speeds up releases and reduces deployment problems

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

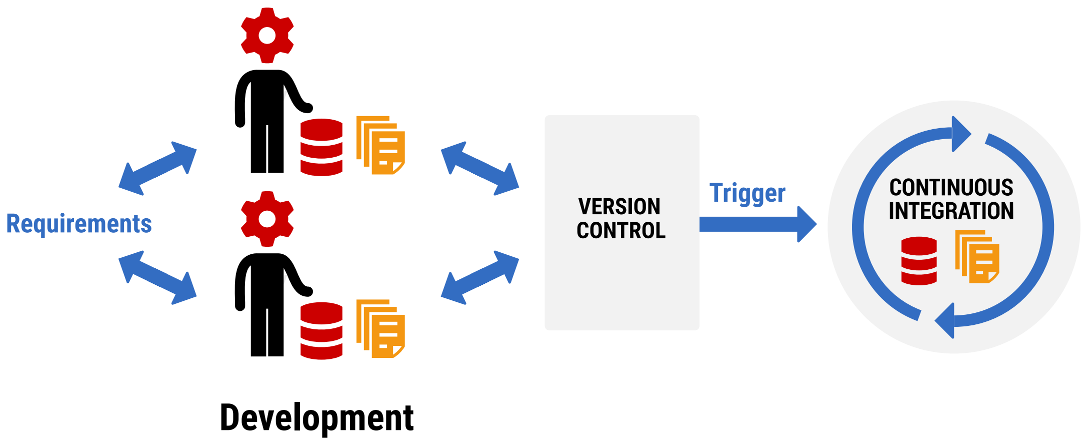
A CI server uses a build script to execute a series of commands that build an application. Generally, these commands clean directories, run a compiler on source code, and execute automated tests. For applications that rely on a database backend, build scripts can be extended to perform additional tasks such as testing and updating a database.

It's this process of generating, testing, and deploying the database build scripts that makes continuous integration for databases possible. With the **Deployment Suite for Oracle**, the Schema Compare command line can be used with existing CI build servers like Jenkins, Azure DevOps, Bamboo or TeamCity. On each check-in to source control (or however often you set it to run), it takes care of the whole database CI process:

- It builds and validates the SQL creation scripts that represent the Desired State, which is necessary for the CI tool to deploy the changes.
- It runs automated tests to make sure any problems are found quickly so they can be fixed more easily.

"Whenever we have a successful build, by packaging the database artifacts along with the application artifacts, we have a complete and synchronized version history of both application and database."

Martin Fowler and Pramod Sadalage, Evolutionary Database Design



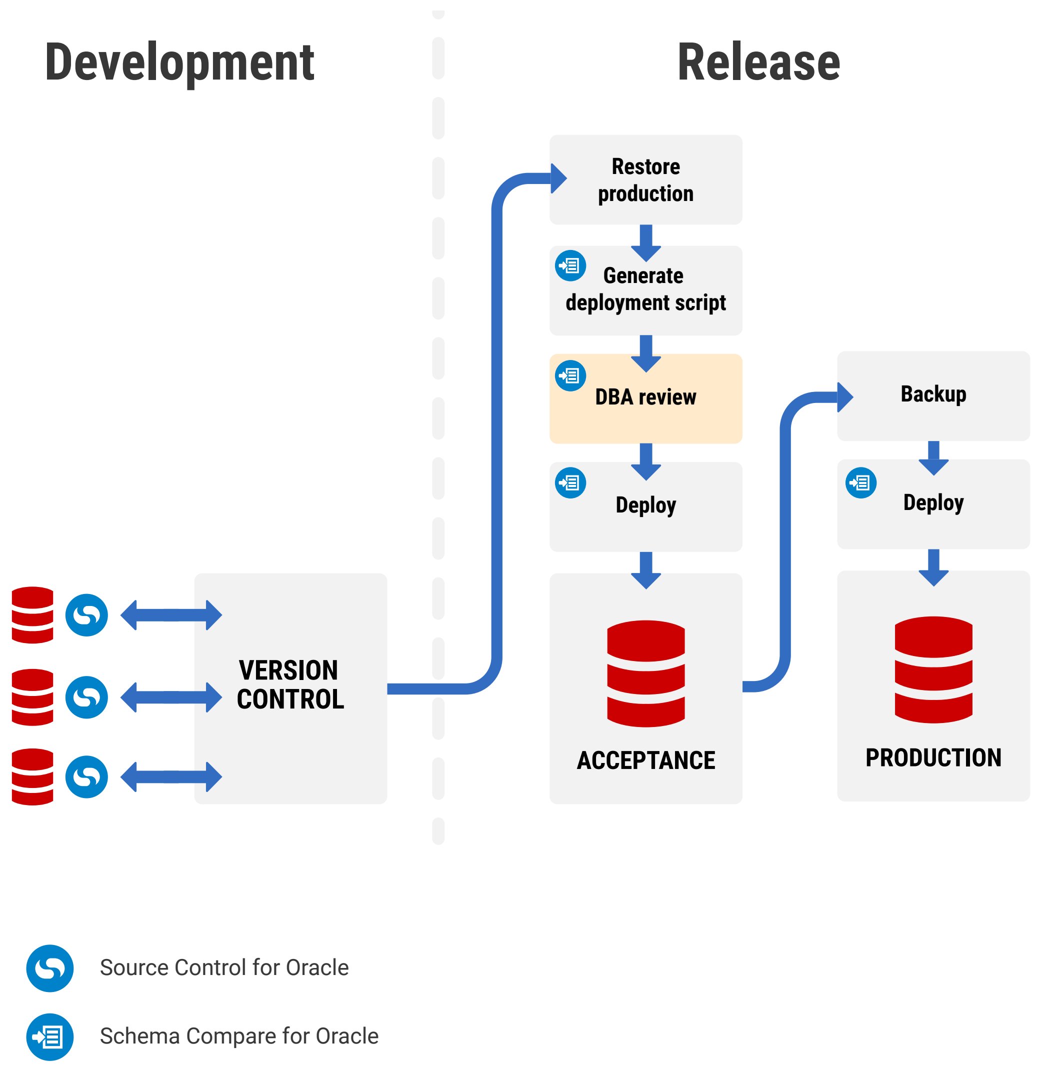
Release management is safer, easier, and matches the way you already work

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 an upgrade script for deployment, and the whole artifact can then be reviewed by the DBA to confirm it is production-ready.

It is also a wise – some would say essential – move to check if there have been changes to the production database. Someone may have made a hotfix in an emergency, for example. To guard against deployment issues resulting from unexpected changes like these, which will have caused the database to drift from its expected (and therefore untested) state, a pre-deployment drift check must be employed.

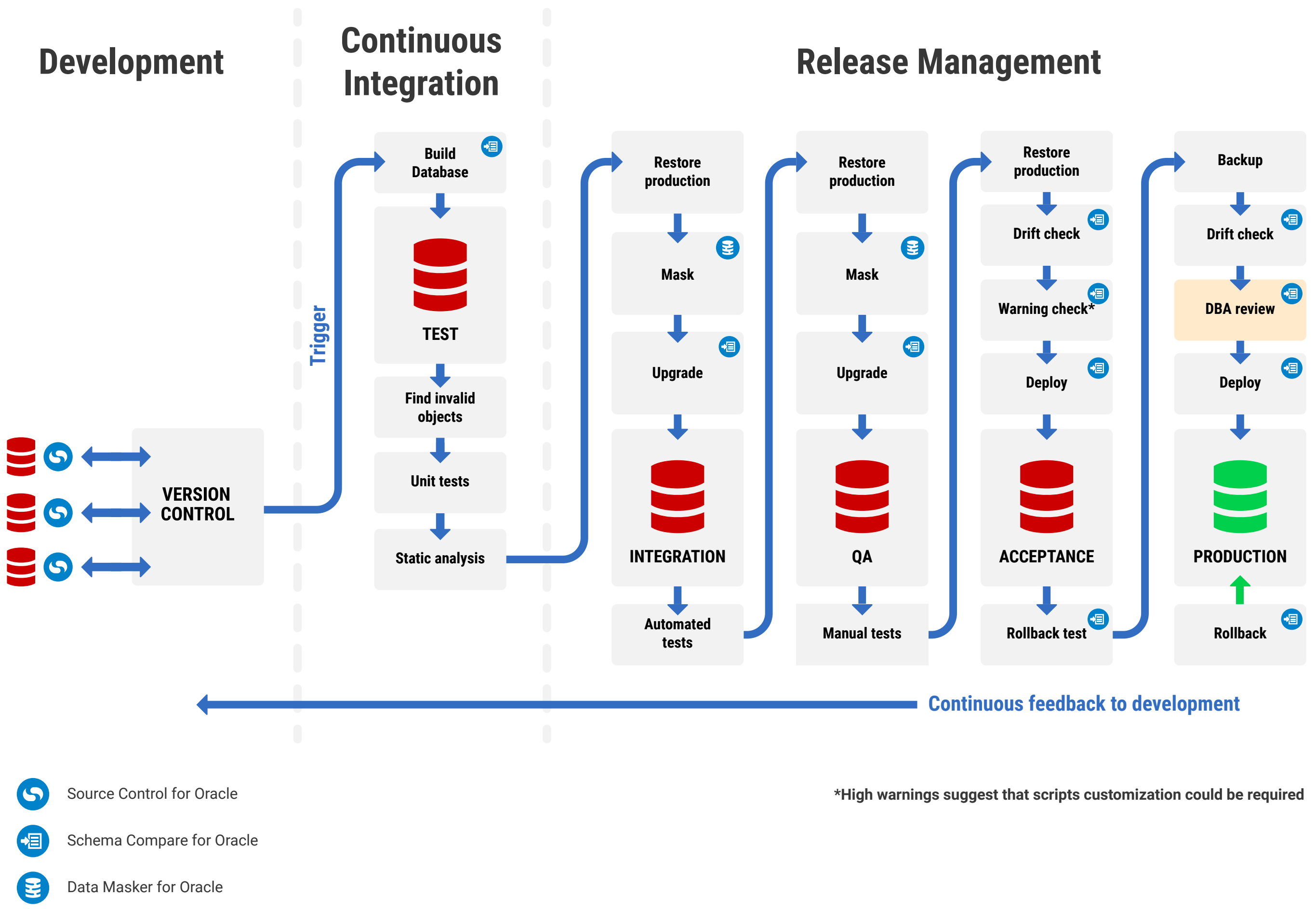
There are two deployment methodologies available when considering production deployments, manual and automated. These methodologies can be combined to form a hybrid process, or used alongside one other. This works so long as the source of truth for both manual and automated deployments is version control.

For manual deployment Schema Compare for Oracle is used to generate a script to run against staging or acceptance. This “practice” deployment allows for a review step and is effectively a rehearsal for deployment. Should everything deploy as expected, the now validated deployment script should be used once more to update the production database.



A simple manual deployment pipeline

Automated continuous delivery leverages continuous integration and release management tools. A deployment script is generated at release-time by comparing the versioned scripts folder containing the desired state against acceptance or another copy of production. DBAs can review the deployment SQL script alongside the change report and warnings report, and on approval, use the same script against production. Because automation is being used, many more test processes and validation checks can be included, affording increased confidence in the ultimate success of the release.



An automated deployment pipeline

As setting up an automated DevOps pipeline can appear daunting, the remainder of this document will run through the process, describing the rationale of each step along the way.

Redgate Oracle tools as the building blocks for Database DevOps

For each topic in this section there are worked examples and code samples on Redgate's Oracle DevOps **documentation pages**.

Version Control

There are few organizations that don't have a version control system in place today. Code versioning has long since been regarded as an indispensable development practice and has well-understood benefits:

- Object level history
- A mechanism to easily share changes between developers
- An audit trail of who made changes, what changed and when
- The ability to recover to a previous historic version
- Branching allows different workstreams to exist in parallel
- Version control stores the source of truth and is therefore a prerequisite for continuous integration and automated releases

Unlike application code, the database has two key version control artifacts: the object code (tables, views, stored procedures), and the deployment scripts (the code that upgrades the existing database state to the next state).

Source Control for Oracle unlocks all of the benefits of version control listed above, managing the schema state and its history in a folder structure in your version control system. Each object type has its own subfolder, which in turn contains one .sql creation script for each individual object. This scripts folder represents the desired state source of truth for subsequent deployments.

Static data (lookup data, reference data, seed data) can also be source controlled alongside the schema objects, and deployed using ***Data Compare for Oracle***.

Upon deployment, the ***Schema Compare for Oracle*** engine compares the state represented in the scripts folder with the target database, calculating a dependency-aware deployment script to upgrade the target to the newer version. This script becomes the key deployment artifact that will be validated and re-used throughout the release pipeline.

Continuous Integration (CI)

Whereas application code is compiled and unit tested as part of a CI process, database code can be used to build a database, can be checked for invalid objects, and as with application code, can be unit tested.

Database build

By triggering an automated build when you check database changes into source control, you get early feedback on whether changes can be successfully built and have a stable current build consistently available. This is analogous to checking whether application code compiles. ***Schema Compare for Oracle*** can be tasked to build a database from the desired state scripts, along with a full database creation script.

Invalid objects

The SQL that ***Schema Compare for Oracle*** generates in its deployment script is dependency-aware and are therefore changes are ordered accordingly. This reduces the chance of ending a deployment with uncompiled objects, and their associated performance-impacting runtime recompilation. However, an extreme class of uncompiled objects, invalid objects, exist simply because they can't compile. These come about if an object references a second object that has since been renamed or removed. Oracle itself is forgiving about the existence of invalid objects, and will not prevent a database from building successfully if they exist. This means that a separate build check is necessary to track them down so they can be fixed. Although invalid objects can often be spotted during the course of development, nothing prevents invalid objects from being inadvertently checked into version control, or as the by-product of a mismanaged branch merge.

Unit testing

The established pattern for unit testing application code is to isolate the code under test using mocks and stubs. This means that the database layer is abstracted out of the tests and often gets excluded from the testing process, which introduces risk. To ensure that coverage includes the database code, a database unit testing framework such as utPLSQL can be employed. This exercises database objects, such as packages, functions, and procedures, and also benefits from code coverage reporting.

Static code analysis

Static code analysis inspects the code for "smells", best practices, deprecated syntax and naming conventions. Static analysis tools offer a selection of rules that
can be customized according to an organization's own chosen standards, conventions and policies.

Release management

Whereas the build and CI process is more concerned with testing, release management is focused on orchestration and supplying confidence that the deployment will occur predictably and safely.

Confidence comes from having a review and approval step, where the deployment script can be scrutinized by an expert, and putting in place a reliable and repeatable process to orchestrate the deployments.

Deployment artifacts

The key deployment artifact is the SQL deployment script generated by ***Schema Compare for Oracle*** as this is what will eventually be run against production. ***Schema Compare for Oracle*** can also output a report detailing the full set of object changes with their respective "before and after" differences. The schema comparison engine can raise deployment warnings for changes that could lose data, or have potential to adversely impact the deployment and therefore warrant further inspection.

Orchestrating the deployment

While CI tools are great at building chains of dependent tasks to assemble, compile and test an application, release tools specialize in simplifying the process of provisioning environments and the orchestration of deployments, promoting changes through the pipeline all the way to production. The philosophy behind such tools is to "build once, deploy many", which minimizes the number of "moving parts", thereby strengthening the predictability of the deployment process.

However, the one moving part that can be hard to control is the database itself. This is because the database suffers from a unique tendency to change out of process. This is known as database drift.

Database drift

Database drift is the name given for unexpected structural change to a database. Databases can drift for a number of reasons - either because a deployment has been made outside the mandated process without informing the necessary stakeholders, or because a necessary emergency fix has been applied directly to production.

The unpredictability of drift introduces high risk to the deployment process as it invalidates the integrity of the validation stages. If the state of the intermediate test databases differs from that of production, how can there be confidence that the production deployment will be successful?

Schema Compare for Oracle can be used to check for drift throughout the release process, ensuring that if drift is detected, the process can be halted.

Masking

With increasing regulatory need to safeguard the privacy of organizational data, particularly Personally Identifiable Information, data masking is emerging as a practical solution to both allow realistic data to be used in dev and test environments, while also obfuscating the sensitive data. Redgate ***Data Masker for Oracle*** can be leveraged to assist in the provisioning of development database instances, and environments further down the release pipeline.

Recovery

Although backups provide the most comprehensive recovery solution, ***Schema Compare for Oracle*** can be used to generate roll-back scripts and schema snapshots, allowing a rapid recovery mechanism for those circumstances where recovery wouldn't lose any data.

Conclusion

This whitepaper has demonstrated how database deployments can be faster, easier, and error-free by extending DevOps practices to databases. It has shown how development and operations teams can orchestrate database processes in order to better safeguard data. It has also highlighted the importance of continuous delivery for databases in encouraging the organization, and development and operations teams to work together to build and deliver great software.

The very first step on this path to the continuous delivery of database changes starts at version controlling your database code. 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 teams have the database under version control, they can use further Redgate tools, together with their chosen CI build server, to continuously integrate and test each change committed. This speeds up releases to customers, reduces the risk of deployment problems, and frees developers from time-consuming, manual change management tasks.

As part of this CI process, a database deployment artifact can be created and, with minimal effort, promoted through multiple environments using a release management tool.

Importantly, your teams can continue using the software they're already familiar with, making the transition simple and easy.

Resources and further reading

To find out more about the benefits of Database DevOps or learn how you can set up these processes for your team, please consult the following resources.

Deployment Suite for Oracle

Data Masker for Oracle