DevOps Compliance Essentials

CONTENTS

ALIREZA CHEGINI
SENIOR DEVOPS ENGINEER, S-RM

INTRODUCTION

DevOps has become a must for organizations. Considering DevOps as a culture is key to ensuring not only that existing processes are continuously improved, but that any new project follows the same mindset to keep all projects and solutions consistent by following the software development lifecycle (SDLC) and DevOps best practices.

What makes this integration so challenging is determining how an organization can apply DevOps practices while maintaining security regulations and compliance. Organizations struggle to keep their infrastructure and applications safe — the primary goal is to manage and protect information.

In recent years, new regulations have been introduced to ensure DevOps compliance, particularly regulations such as ISO 27001 and GDPR. These regulations help organizations implement an effective approach for information security. In this Refcard, we are going to review DevOps compliance essentials and regulatory policies for securing all stages of the SDLC.

WHAT IS DEVOPS COMPLIANCE?

DevOps aims to create a process in which any organization can release software smoothly — from ideation to running in your production environment. While DevOps ensures a high-quality release, DevOps compliance means the software release process is meeting regulations applicable to a system. These regulations can be organization-, industry-, or government-related. By being DevOps compliant, organizations are adopting and integrating regulations into their DevOps culture. In the end, one process will reflect DevOps best practices and meet regulations and compliance in full.

In recent years, database development has also entered the picture since deploying changes to the database is often the bottleneck in software development that slows down releases. Most of the software teams practicing continuous delivery effectively use version control for database changes and manage them the same way as changes to the application. By exponential growth of data, companies are interested in extracting more value from database development and the sprawl of data across different databases, various locations, and multiple database copies used in development and testing. This has resulted in data 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 stricter and demand a lot more from companies in order to achieve compliance. GDPR has already been introduced, affecting any company that manages data from European citizens. However, many organizations still did not implement the GDPR properly, which may result in legal issues or fines.

One

interface

DevOps Platform

data store

permissions model

value stream

set of reports

compliance framework

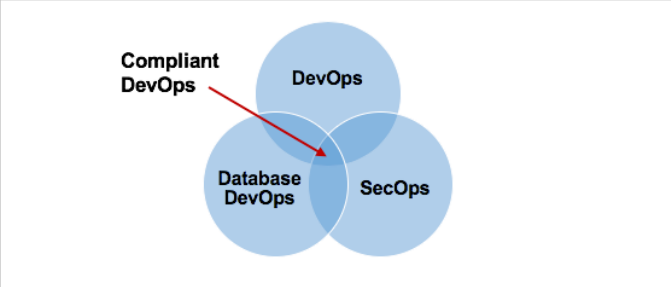
place to contribute

GitLab

Ship secure code faster, deploy to any cloud, and drive business results.

With new data protection laws coming into play and consumers more aware than ever of how their privacy is being compromised, there is now a requirement for companies to adopt a DevOps compliance approach. This approach is 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.

Figure 1



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

In the past, companies were releasing changes every three to six months or longer, and a review was scheduled at the end of each release cycle for the security team to certify and recommend any changes.

With DevOps teams making small changes and releasing them more frequently, this safety window has nearly 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.

Including the database in DevOps processes is critical. 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 introduced and the audit trails it provides across the database development cycle ease compliance so that companies can deliver value faster while keeping data safe.

COMPLIANCE PROCESSES

Compliance processes are simply administrative tasks in which we define the standards and acceptance criteria and then evaluate the development stages against the constraints that we defined earlier. A big part of this administration is to make sure organizations are documenting all processes.

A compliance process has several key elements:

- Definition – an objective of compliance property or control
- Validation – a test which measures a control
- Acceptance criteria – a value or ranges that you expect to see as a result of validation
- Evidence – a document that proves the validation results

DEVOPS COMPLIANCE ESSENTIALS

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.

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

This 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.

INSIST ON SECURE CODING

Thanks to DevOps, applications are developed faster and eliminate the need for security reviews at the end of development. Instead, security is baked into the pipeline with tools like static code analyzers and open-source code vulnerability scanners that test 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, it is common to integrate code scanning tools (for example, SonarQube and Sonar Cloud) which identify code smells and bugs earlier in the development pipeline. Additionally, there are tools to detect application licenses, which are quite useful for enterprise organizations to ensure any dependent

software is licensed properly — and that the license is in line with their organization's requirements. As an example, a common open-source tool for license scanning is the ScanCode toolkit. There are other tools that scan against vulnerabilities, containers, and infrastructure as code, such as the popular tool Snyk, which helps to ensure a secure software development.

VERSION CONTROL EVERYTHING

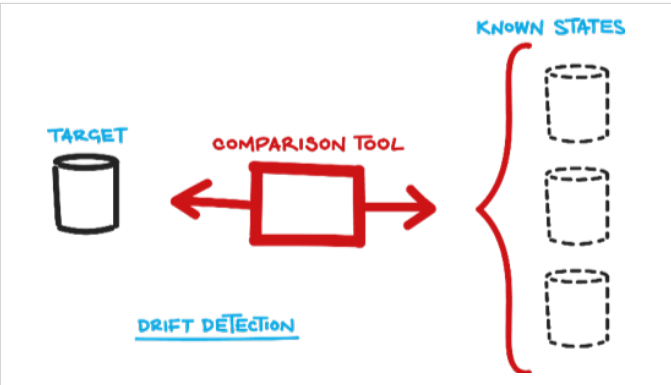
Version control is becoming standard in application development and requires developers to check their changes into a common repository during the development process, preferably at the end of each working day. As a direct result, developers have access to the latest version of the application, one source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

Figure 2

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. However, there are two approaches for version controlling databases that appear to be diametrically opposed.

A 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.

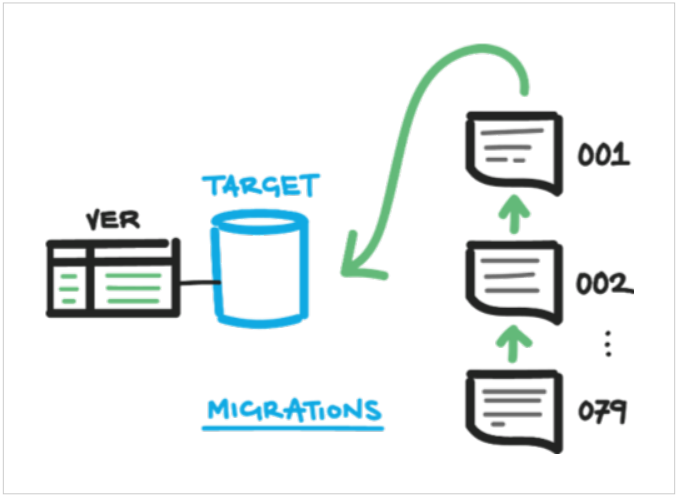
Figure 3



This approach doesn't need to know that 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.

A 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.

Figure 4



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

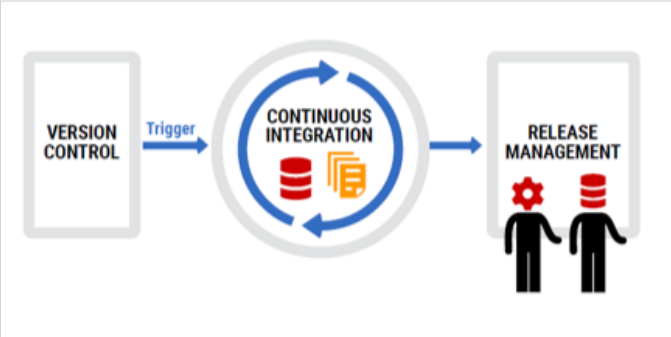
State-based version control allows database code to be held at the component level, with each table, view, or procedure being held separately, making merge conflicts less likely and easier to resolve if they do occur. Migration-based version control gives a much more granular level of control over scripts, 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.

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.

Figure 5



For example, every time a change is committed to version control, a continuous integration process can test the change and flag 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.

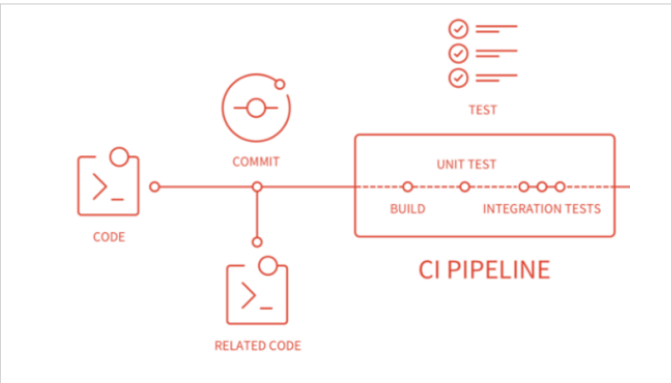
THE FIRST STEP IS CONTINUOUS INTEGRATION

Continuous integration, or CI, is the first part of the DevOps pipeline in which code is compiled to get ready for the continuous deployment portion. It 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.

The common practice is to put the source code in the code repository. The main source remains on a master branch, but for each feature development, a feature branch should be created. Therefore, any new changes should be committed as a pull request.

Continuous integration can be used to trigger an automated build once a pull request is created. Then all code scanning, security checks, and code review is complete within the pull request process. If a build fails or there is feedback from the code reviewer, the code will be kicked back to the developer to be fixed. The point is for the master branch to always contain the stable version. That's why this process helps fix issues efficiently and involves retesting, so a stable current build is always available.

Figure 6



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 that 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-control 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. This means the database on both production and the staging environment should be the same in terms of structure.

It is good to mention that client data, which resides in the production environment, cannot be used on a non-production environment. This is one of most important controls within information security to reduce the risk of data breaches. This will generate an upgraded 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 found on the next page.

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. However, this is not a good practice since it is often error-prone. It is common to have a production issue because of a small mistake in a manual deployment.

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.

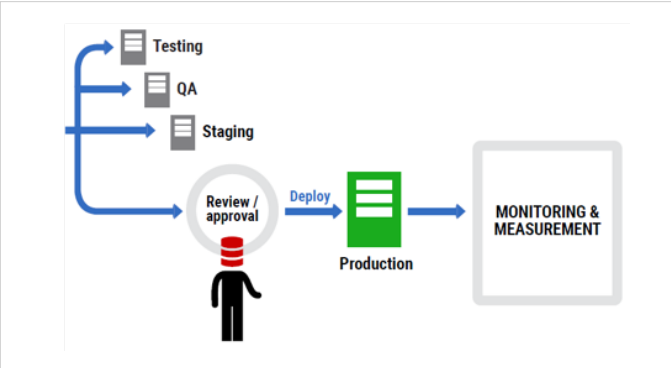
MANAGED DEPLOYMENTS USING A STAGING ENVIRONMENT

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 that 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.

Figure 7



However, it can happen, 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.

SECURE CODING

Nowadays, there are various practices that developers can easily follow to secure coding. However, when it comes to building a compliant environment, awareness is not enough, and there should be a process to verify new code continuously. Continuous integration (CI) is designed to tackle these challenges by scanning any new code for vulnerabilities, security, penetration, risk assessment, etc.

MONITORING AND 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 obligated 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.

Today, most organizations are putting their infrastructure into the cloud, which brings dozens of new functionalities and services with less development and a better price model than on-premise. However, cloud infrastructure has its own complexity — especially when it comes to monitoring, information security, and policies.

Most cloud providers offer solutions to monitor compliance, security, and auditing. For example, Azure provides different layers of security for SQL Server such as:

- Encryption in transit
- Encryption at rest
- Encryption in use

The other service is Azure policy, which helps organizations to enforce their standards and assess compliance at scale.

CONCLUSION

Implementing DevOps practices is a great achievement for organizations, but the most important part is to stay compliant as processes are improving. That's why DevOps compliance is the only way to ensure both DevOps practices and meeting regulations can be achieved at the same time and keep processes consistent.

Regulations and policies help organizations manage their business and activities to avoid any operational or legal issues. It is important to incorporate regulations in DevOps practices to keep this transformation efficient and cost-effective.

Currently, every organization is defining its own DevOps compliance, which is quite costly and complicated in most cases. Most organizations are suffering due to a parallel process in compliance, regulations, and security controls. However, they are still trying to optimize their processes and stay compliant.

Efficiency is one of the key areas in all projects. Soon, compliance by design will be the solution to reduce inefficiency and the best opportunity to adapt to meet your compliance demands from an architectural standpoint.

WRITTEN BY ALIREZA CHEGINI,
SENIOR DEVOPS ENGINEER
AZURE SPECIALIST AT S-RM

Alireza is a software engineer with more than 20 years of experience in software development. He started his career as a software developer, and in recent years, he transitioned into DevOps practices. Currently, he is helping companies and organizations move away from traditional development workflows and embrace a DevOps culture. Additionally, Alireza is coaching organizations as Azure Specialists in their migration journey to the public cloud.

DZone
600 Park Offices Drive, Suite 300
Research Triangle Park, NC 27709
888.678.0399 | 919.678.0300

At DZone, we foster a collaborative environment that empowers developers and tech professionals to share knowledge, build skills, and solve problems through content, code, and community. We thoughtfully — and with intention — challenge the status quo and value diverse perspectives so that, as one, we can inspire positive change through technology.