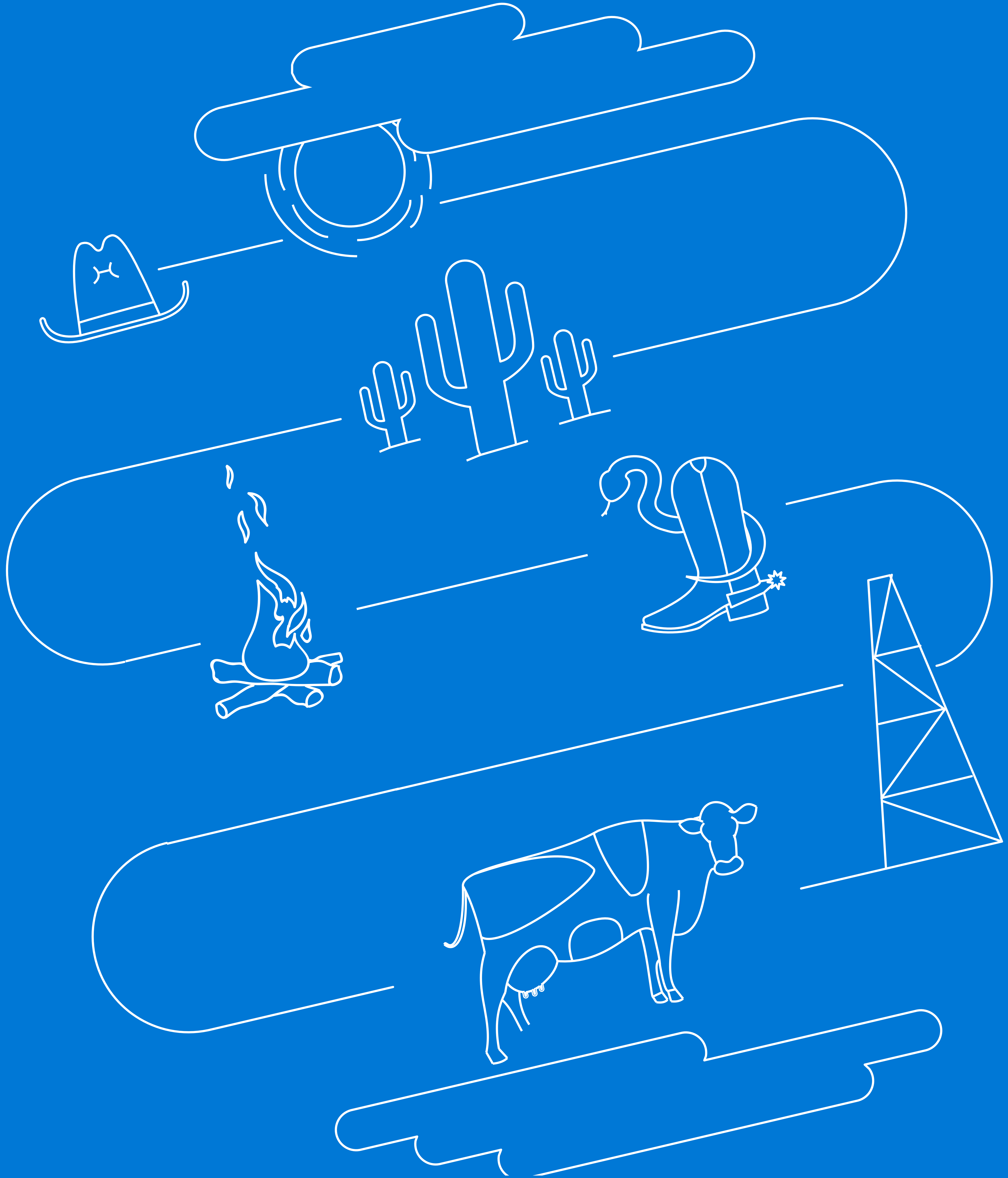
Microsoft

Welcome to the new frontier of DevOps:

Improve application development and deployment using Microsoft SQL Server 2017



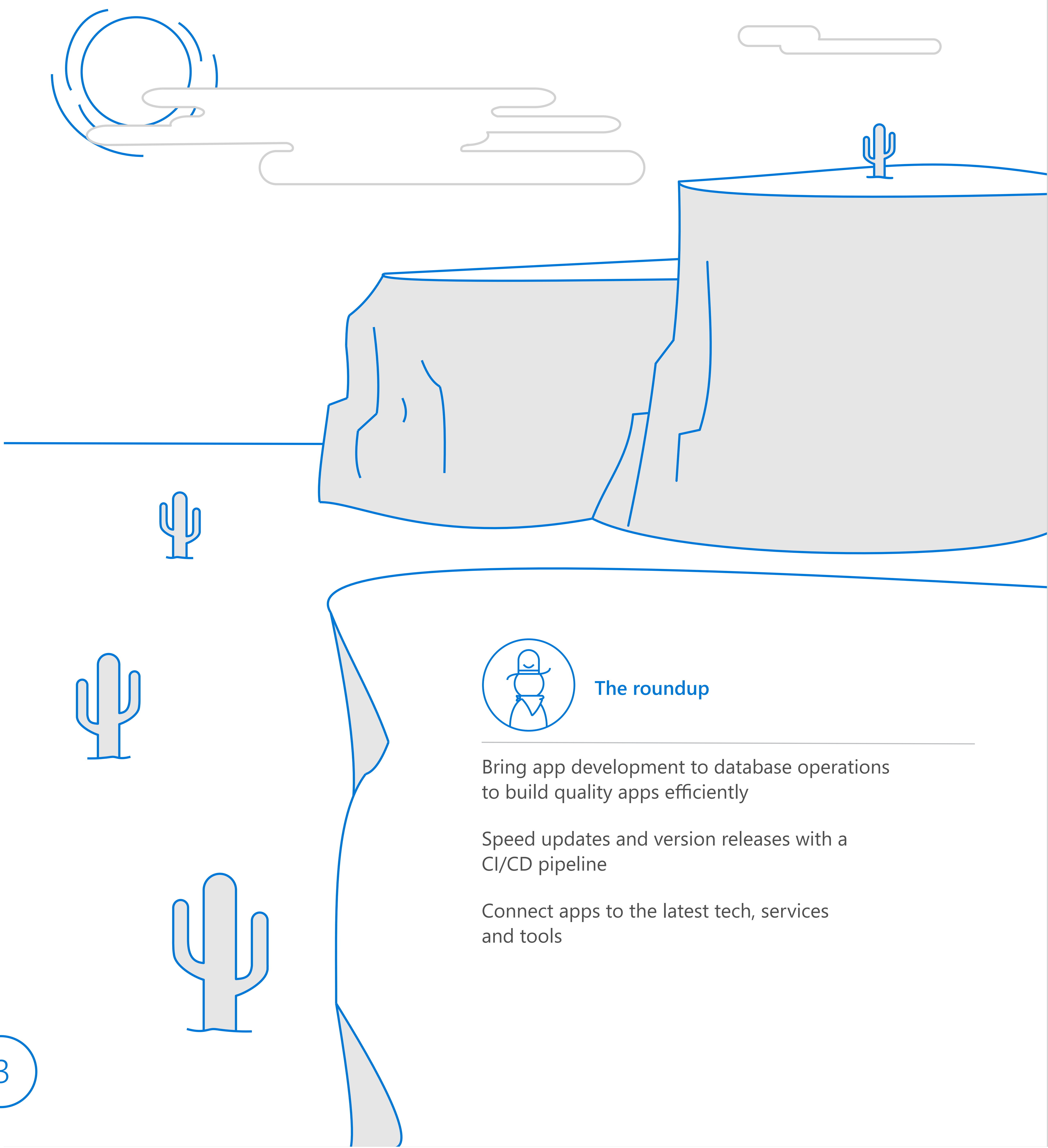
The draw of the DevOps frontier

It’s no wonder so many teams are switching to the DevOps approach for application development and deployment.

Teams that successfully employ the DevOps approach use a systematic method for development and deployment. They understand that fast doesn’t mean hasty, efficient doesn’t mean inflexible, and bringing two functions closer together doesn’t mean eliminating one. In order to create their continuous integration environment, they don’t just improvise database development, they plan for it.

We want to help you successfully embrace a DevOps approach in which you can realize the benefits of a CI/CD environment without compromising the integrity of your application or your database.

Throughout this document, we’ll look at a new deployment process that supports the DevOps approach and explain three methods you can use to facilitate it.



The roundup

Bring app development to database operations to build quality apps efficiently

Speed updates and version releases with a CI/CD pipeline

Connect apps to the latest tech, services and tools

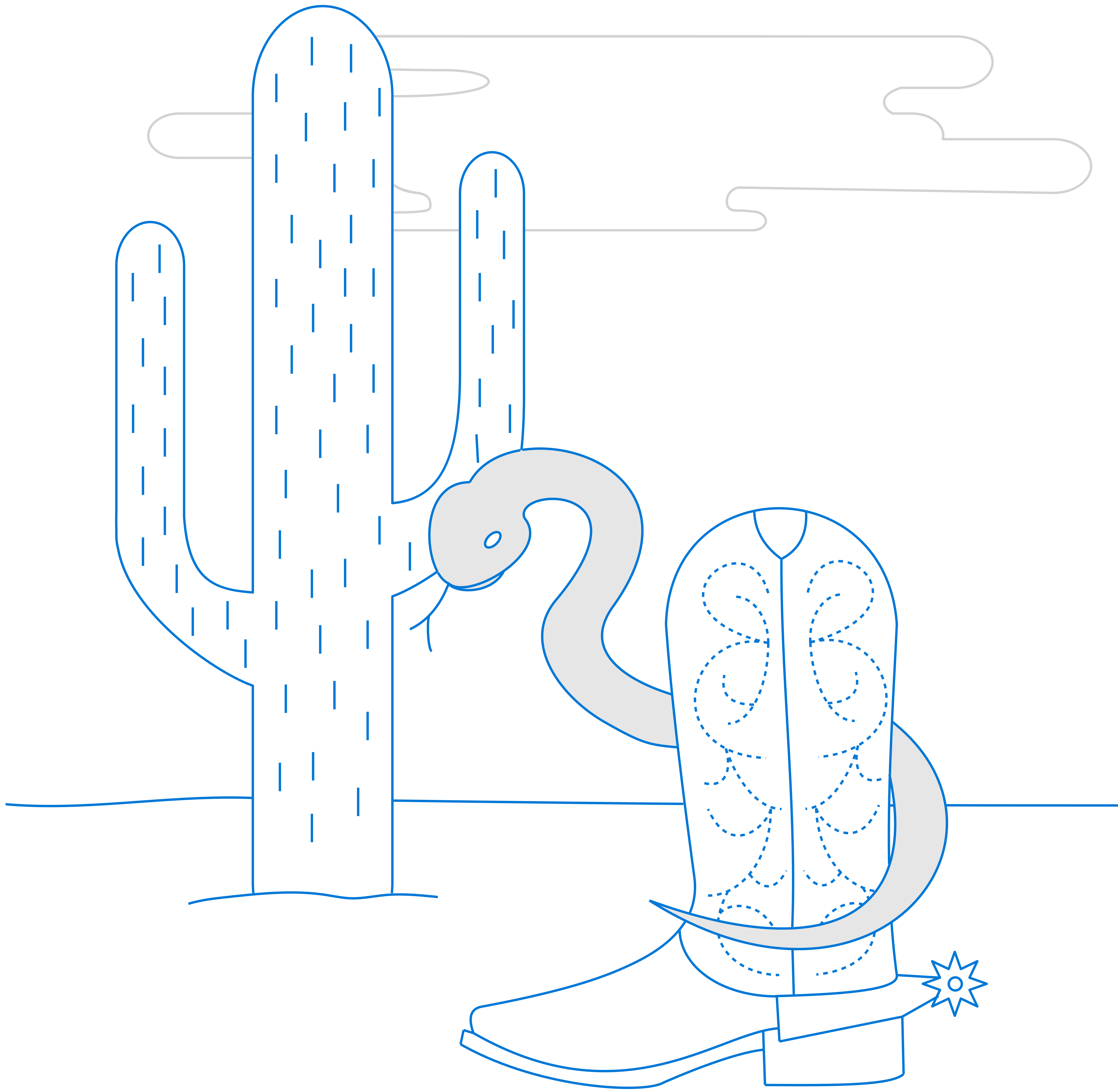
The cowboy method makes for a wild rodeo ride

Many developers understand the benefits of checking their app into source control, applying continuous integration, and performing automatic testing before deployment. However, not all developers realize these steps are just as important when it comes to connecting the app to its supporting database.

When developers use requisite SQL code changes instead of a structured, methodical process, we refer to it as the cowboy method.

Shooting from the hip and making up rules along the way, the cowboy method is wild, unruly and downright discourteous to your future self. Sure, it works in the short term, if your only goal is to deploy new applications fast. But in the long term, or as your app evolves, the cowboy method causes certain calamity.

- Cowboys may attempt to track changes and save versions by writing comments directly in the code.
- They may use source control for certain scripts in case they need them later, but they don't do it frequently or systematically.
- When it comes to testing, cowboys may test in production or perhaps not at all.
- And, cowboys almost always deploy using a manual process.



Tracking cattle cowboy style

Stake claim to the app

Imagine you take a job at a dude ranch and inherit a cattle-tracking app that helps ranchers locate cows.

TThe app's been on the fritz, showing cows where they aren't. The staff started to notice inconsistencies around the time of the last app update which was supposed to include new territory but didn't. Before then, the app worked just fine. You decide to roll back to the previous version from before the buggy updates, add the new territory, and include business intelligence that predicts cow movement under different weather events.

Corral the problem

When you open the app's database, you quickly realize the original author used the cowboy method of deployment, meaning there's no source control or version record. Instead you have to dig around in the code to decipher the pieces you need.

A quick scan of the database code reveals:

- Stored procedures with a lot of green comments. These are supposed to explain the who, what, when and why of the app changes, but the comments are so out of date and out of order that they are no help.
- Confusing names for the stored procedures, tables, and other objects. These don't make sense and confuse the story of the app even further.
- No record of testing, no tracking of active code, and no record of detected bugs. Testing must have been performed at the user level by the developer or in production, if at all.
- Signs of manual deployment, meaning you'll have to manually deploy any changes you make.

Drive home the cows

The update that should have taken a day or two is about to cost you weeks. You contemplate trading your computer and keyboard for a cowboy hat and a horse to track these cows instead.

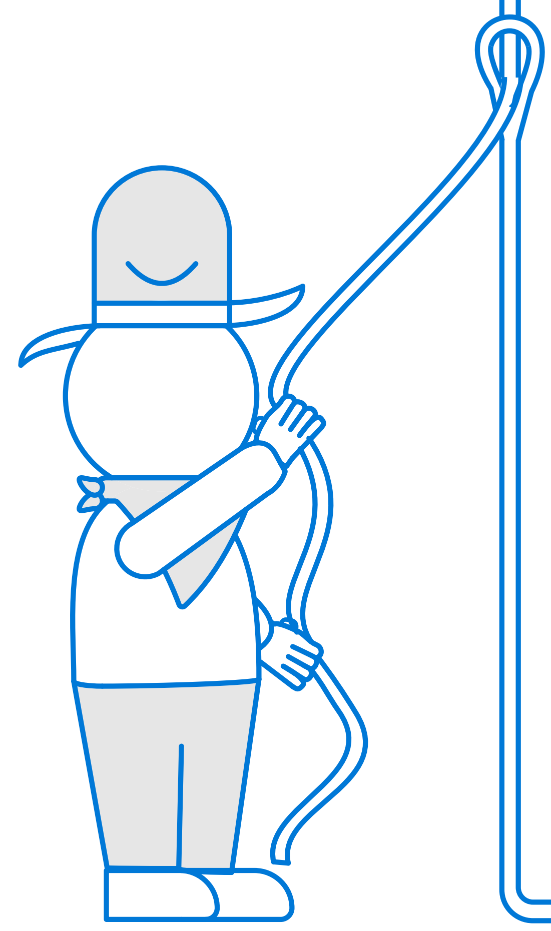
Before you roll up your sleeves and dive in, you plan to meet up with three developer buddies who work at competing dude ranches.

Finding the holy grail of database operations

The best practice for DevOps deployment involves parallel paths of source control, continuous integration, automatic testing and deployment for your app as well as its supporting database.

Applying this same four-step process to your database not only makes versions and updates easier to manage, but it also makes your overall deployments repeatable and less prone to errors.

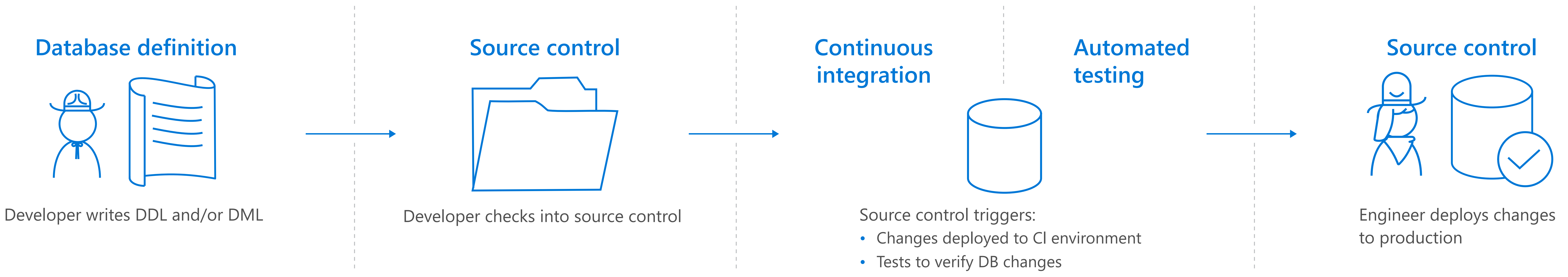
Furthermore, once you have established this process, you have the ability to more easily instate a CI/CD environment that deploys updates automatically.



SQL Server 2017: Designed with DevOps in mind

SQL Server 2017 can help you along the DevOps deployment pathway. The latest release now runs in Docker or Linux containers, allowing you to attach and ship a database as a component of your app. Containers make deployment faster, easier and more consistent. Whereas it used to take hours to stand up and connect a database to your app, with a container it takes about 20 seconds – just pull, run and compose!

The DevOps deployment process


Database definition
Developer writes DDL and/or DML
Source control
Developer checks into source control
Continuous integration
Automated testing
Source control triggers:
• Changes deployed to CI environment
• Tests to verify DB changes
Source control
Engineer deploys changes to production

Bringing civility to your DevOps approach

There are three methods you can use in lieu of the rough-and-tumble cowboy method, all with systematic processes for database deployment. They rely on establishing and preserving a single source of truth (SSoT) which can then be referenced from one primary location and propagated throughout the system. Using a SSoT helps reduce errors, version inconsistencies and overall frustration.

Object Relational Mapping (ORM) method

The ORM method stores the SSoT in source control. It uses drivers to generate a physical database based on a selected, pre-defined model.

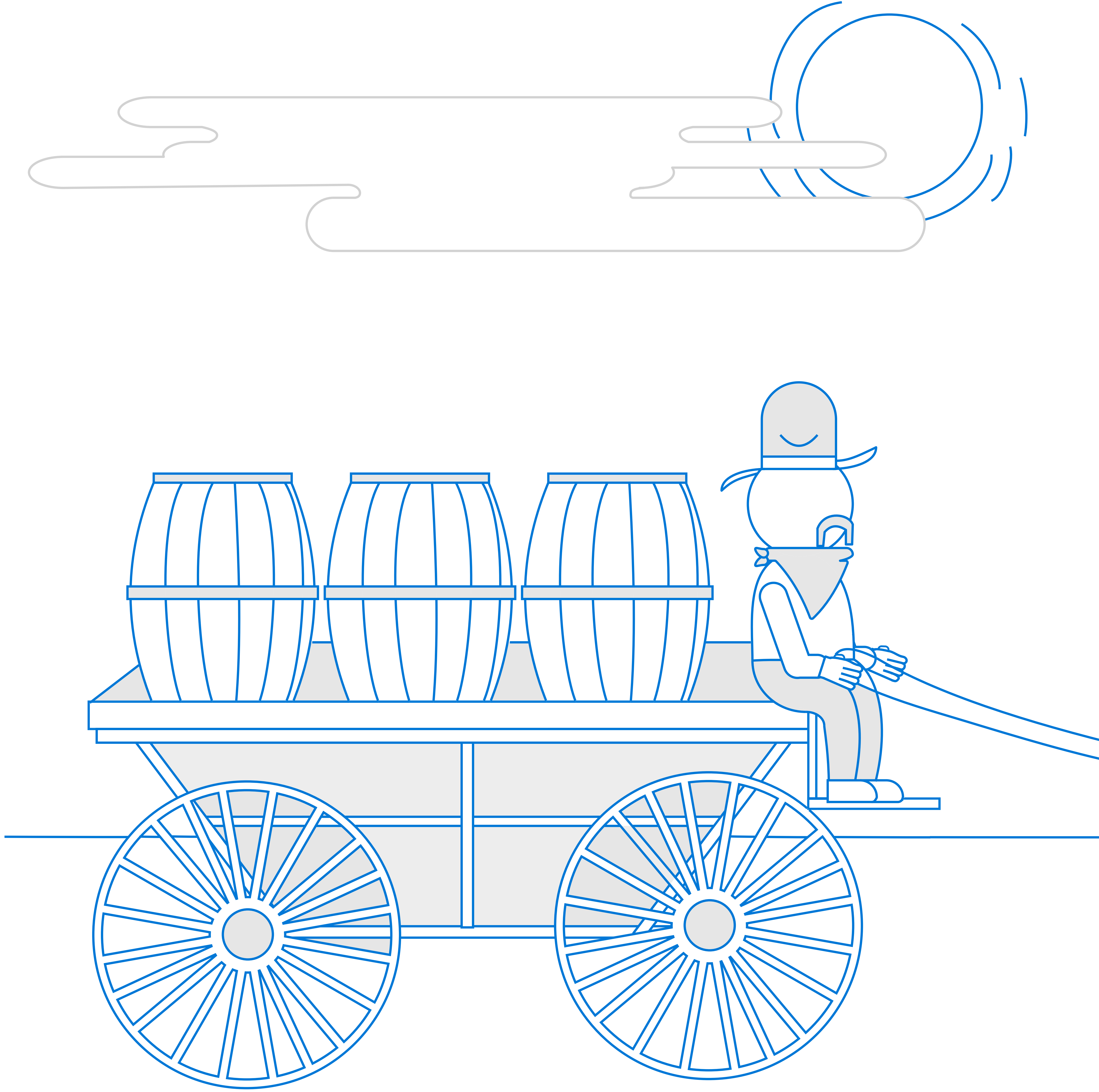
Imperative or Migrations method

The imperative method relies on the live database to act as the SSoT. It requires you to write a series of detailed steps that specify how your database is created, altered and deployed.

Declarative or State-based method

The declarative method stores SSoT in source control and lets you design your own database model. It then uses a database operations product to compare the model against the current version and implement upgrades.

To consistently deploy high-quality apps quickly, you'll want to choose the method that complements your existing database environment as well as the complexity and requirements of your app.



Exploring the ORM method in greater detail

Using the Object Relational Mapping (ORM) method, you write any application code to describe tables and relationships in your application. The pre-configured driver that you select from Entity Framework translates the object model from your app to the relational database using PowerShell Command Enable-Migrations.

With ORM you don't need to write (or even know) T-SQL, and there's no need to muck around in the database. This also means that you cannot change the database schema – it must be used exactly the way it is built.

ORM works especially well when the application draws from a single database.

Supporting tools: Entity Framework is a free, open source collection of drivers created by Microsoft that connect to SQL Server to build pre-configured database models. It uses PowerShell to translate application code into T-SQL queries.

The roundup

Write your app to match a predefined model.

Use a driver to translate code into T-SQL.

No need to touch the data base.

Tracking Cattle at Ranch ORM

Background

That night, your buddy from Ranch ORM mentions he also has a cattle-tracking app in need of territory updates and BI. The only real difference between your apps is that the original developer at Ranch ORM connected the app to its database using the ORM method.

Approach

Using the ORM method, he can update the app by:

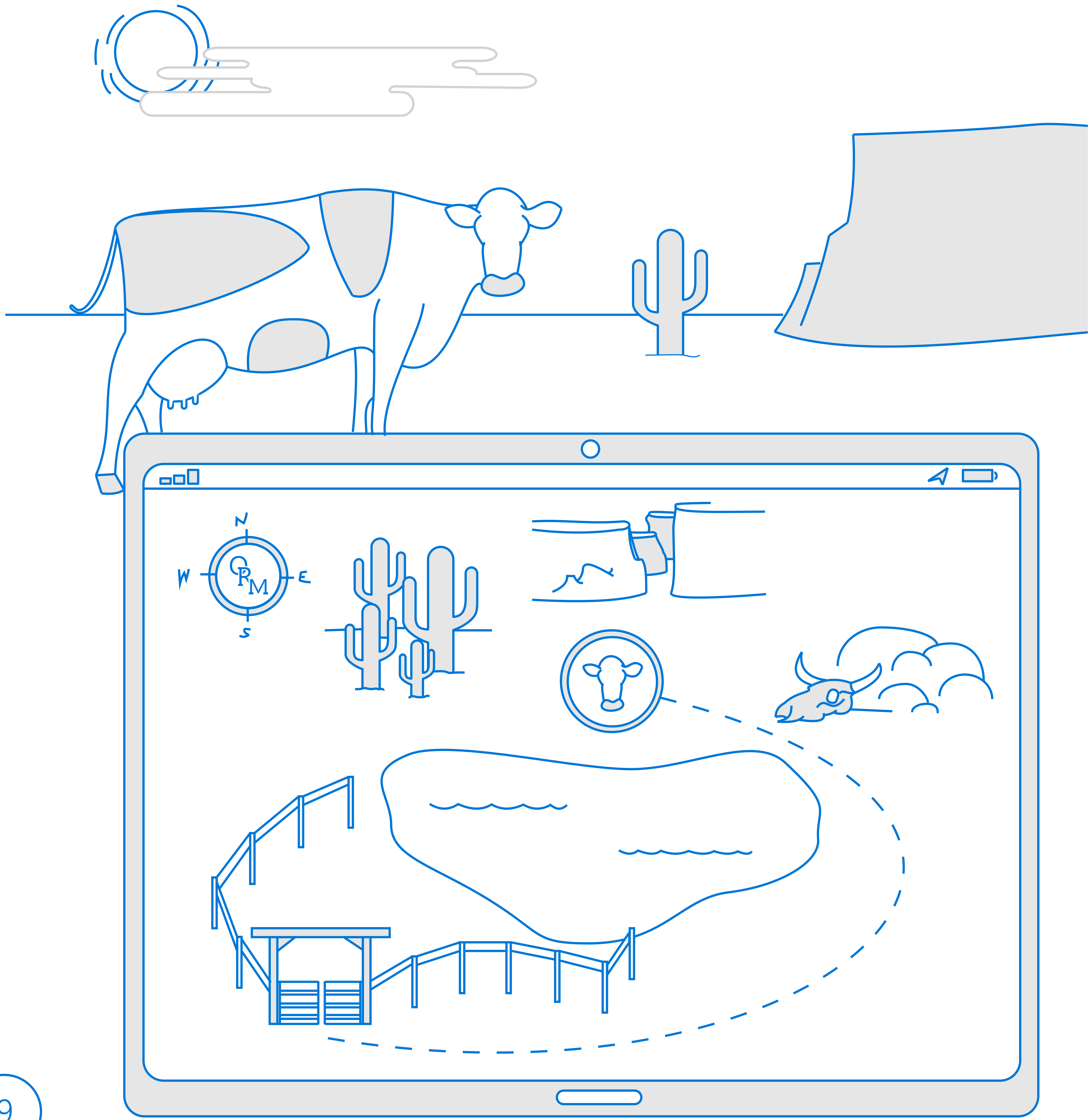
1. Finding a new driver in Entity Frameworks that can build a database model to complement the app's expanded functionality.
2. Making the required changes to the app's source code.
3. Deploying the app, which automatically triggers the driver to translate the app source code and deploy the database.

Outcome

Once he finds the driver for the database model he needs (provided it exists), he won't have to write a single line of T-SQL or perform any testing. You toast to his good fortune.

He points out that using the ORM method means he can't modify the database or roll back to a previous version. Instead, he must use new drivers each time. He's also worried if they need multiple databases feeding into the app, it could require some serious tomfoolery.

But you're not feeling much sympathy. This guy's smart and he'll find a way.



The roundup

Change and check app code into source control.

Follow a disciplined process for updating scripts directly in the live database.

Implement C/I and testing in the live database.

Exploring the imperative method in greater detail

Under the imperative method, changes are recorded by handwriting scripts (.SQL) directly in the data environment. The database itself acts as the SSoT by recording all these updates.

Each script is housed in a shared folder that's eventually compiled into a single script and checked into source control for repeatable deployments. This can be a time-consuming and technical process, but it provides the greatest level of control. It works best for skilled, disciplined, and experienced developers who are comfortable with T-SQL and can accurately anticipate what they are building.

The method allows devs to connect the app to multiple databases as needed – behind a firewall, buried on-premises, or from a public database source.

Database operations products, such as ReadyRoll (available through Redgate and shipped with Visual Studio Enterprise), streamline the process with modification, upgrade, and creation scripts integrated into your source control.

Supporting tools

- ReadyRoll is a third-party product that integrates seamlessly with Visual Studio to create and modify scripts as well as check them into source control.
- MS SQL-CLI tool helps with querying the database when there is a firewall or the user wants to automate development and deployment processes.
- SQL Server Script, which is a series of Transact-SQL statements stored in a file, can be used as input to SQL Server Management Studio Code editor or the sqlcmd and osql utilities.
- SQL Server Management Studio is the primary database administration tool used for SQL Server. It has several integrations with RedGate products, including ReadyRoll.
- Command line utilities (ie: sqlcmd, mssql-scripter, etc) can be used when there's no UI or direct access to the database).

Tracking cattle at Ranch Imperative

Background

Your second buddy, who landed at Ranch Imperative, explains the cattle-tracking app she's working on now. She has to roll back to a previous version, update the territory parameters and add new BI functionality. In her case, though, she's wrangling multiple databases – a couple behind the firewall and a couple public sources. She thinks that's probably why her predecessor used the Imperative method to connect them to the app in the first place.

Approach

Under the Imperative method, she can update the app by:

1. Writing the expanded functionality into the app source code and checking it into source control.
2. Handwriting scripts directly in the database to reflect the changes made to the app.
3. Storing and compiling update scripts as the latest database source control.
4. Deploying the live database.

Outcome

She's got a complete version history living in the database that she can fall back on at any time, and she can change the database however she wants by writing T-SQL scripts. She points out they're using Readyroll to create and compile all the scripts, which she says makes it surprisingly fast, and they're working on writing the scripts to enable CI/CD. You toast to her good fortune.

The developer from Ranch ORM tells everyone how glad he is to not be working at Ranch Imperative. He points out that since the SSoT is the live database itself, the database has to go live before it can be tested which means your friend at Ranch Imperative has to alter and deploy the same database over and over in order to maintain the source control system.

But you wave away his concerns. Your friend's whip-smart – it suits her just fine to have to be a little more diligent.

Exploring the declarative method in greater detail

The declarative method uses a database operations product, such as Visual Studio Team Foundation Server (TFS) and SQL Server Data Tools (SSDT), to compare new create scripts against the current database model. SSDT generates upgraded scripts based on that comparison (it also works in reverse by pushing committed changes back to developer).

Under the declarative method, the source control contains the SSoT which means you never alter the database, only create it new. Each time there is a version update of the app, the database operations product tears down the current database and builds it new to preserve version history and data integrity.

The declarative method is easy to implement and allows you to determine the process and complexity for database development and deployment. Best of all, it enables integration with other service components further down the pipeline.

Supporting tools

Visual Studio including:

- SQL Server Data Tools (SSDT) for creating and comparing scripts.
- Team Foundation Server for sharing and tracking processes between team members.
- MS build for publishing changes and running automated unit tests on a scheduled basis.
- NuGet for helping to automate deployment.
- VS Test, an open and extensible test platform, for running tests, collecting diagnostics data and reporting results.
- tSQLt for unit testing for databases.

SQL Server 2017 for container deployment.

The roundup

Database ops automatically push changes to the database.

Check the app and database into source control at the same time.

Implement C/I and automatic testing in a controlled environment.

Tracking cattle at Ranch Declarative

Background

The last person in your group of buddies works at Ranch Declarative, where they too have a cattle-tracking app which needs the same updates.

Approach

Under the declarative method, the groups on her team can update the app by:

1. Accessing the last (or any previous) version of the app and database stored in source control.
2. Writing new functionality and updates into source code and checking each portion into source control – this can be done simultaneously by different groups using TFS.
3. Relying on SSDT in VS to update the database model(s) by writing new scripts based on the new app source code.
4. Using MSBuild (integrated in VS) to tear down and build the database(s) according to the updated model(s).
5. Performing scheduled, automated tests before going live.
6. Automating deployment with NuGet.

Outcome

In addition to accessing the complete version history available in source control, she can change the database however she wants merely by changing the app and letting VS to do the heavy lifting. You toast to her good fortune.

She's excited about all the time-saving capabilities available like:

- CI/CD as a natural extension of the declarative method that automates ongoing updates
- Container deployment with SQL Server 2017 in Dockers that saves hours of hassle
- Code Analysis that scans for classic mistakes before even reach testing.

Furthermore, she's planning to leverage VS's integration with components further down the pipeline and make some feature enhancements to the app.

The Ranch Imperative dev points out that the declarative method could limit the database schema, but since the model they have works it isn't a problem for now.

Comparing DevOps deployment methods

Method	Highlights	Source control	Continuous network	Automated testing	Production development
ORM	• Low learning curve. • Limited by the availability of drivers for pre configured database models.	Not applicable (the driver predicates the database schema)	Not applicable (the driver predicates the database schema)	Not applicable (the driver predicates the database schema)	Happens automatically with app deployment.
Imperative	• Steepest learning curve. • Highly controllable schema built from handwritten scripts.	App and database have different systems. The live version of database functions as its source control.	Manual or available through the Readyroll workflow.	Can perform manual testing. Automatic testing is available through the Readyroll workflow, however there is limited ability to test without putting it in production first.	Can use various deployment tools and container of choice.
Declarative	• Low learning curve that complements many developers' skillsets. • Offers flexibility for model customization. • Uses source control as the single source of truth.	Both app source code and database source code are checked in to source control at the same time.	Happens automatically with changes made to the model. Database is torn down and built new with each version in source control.	Application runs on a model that can be put into pre-production for a better understanding of how the app will interact with the data layer.	Push code to production with a click.

Cattle-tracking code comparison

Here's a closer look at how developers from each of the competing ranches might go about increasing the territory constraints of their respective apps.

ORM

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;

namespace ORM_Database_Sample
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new TerritoryDirectory())
            {
                // Create and save a new territory
                Console.Write("Enter a name for a new
territory: ");
                var temp_name = Console.ReadLine();

                var temp_territory = new Territory { Name
= temp_name };
                db.Territories.Add(temp_territory);
                db.SaveChanges();

                // Display all territories from the database
                var query = from b in db.Territories
                            orderby b.Name
                            select b;
    Console.WriteLine("All territories in the database:");
                foreach (var item in query)
                {
                    Console.WriteLine(item.territoryName);
                }

                Console.WriteLine("Press any key to exit...");
                      Console.ReadKey();
                }
            }
        }

    public class Territory
    {
        public int territoryID { get; set; }
        public string territoryName { get; set; }
    }

    public class TerritoryDirectory
    {
        public DbSet<Territory> Territories { get; set; }
    }
}
```

Imperative

```
/* Example Table*/
GO

PRINT N'Creating [dbo].[territory]...';

GO
CREATE TABLE [dbo].[territory] (
    [territoryID] INT NOT NULL,
    [territoryName] NVARCHAR(50) NOT NULL,
    [DATECreated] DATETIME2(7) NOT NULL,
    CONSTRAINT [PK_TERRITORY PRIMARY KEY
CLUSTERED ([ID])]
);


/* Imperative
 *
 * Change name of table and increase number of
characters accepted due to longer names
 *
 */


EXEC sp_rename '[dbo].[territory]', '[dbo.ranchTerritories]'

ALTER TABLE dbo.ranchTerritories
ALTER COLUMN territoryName NVARCHAR(100) NOT
NULL;
```

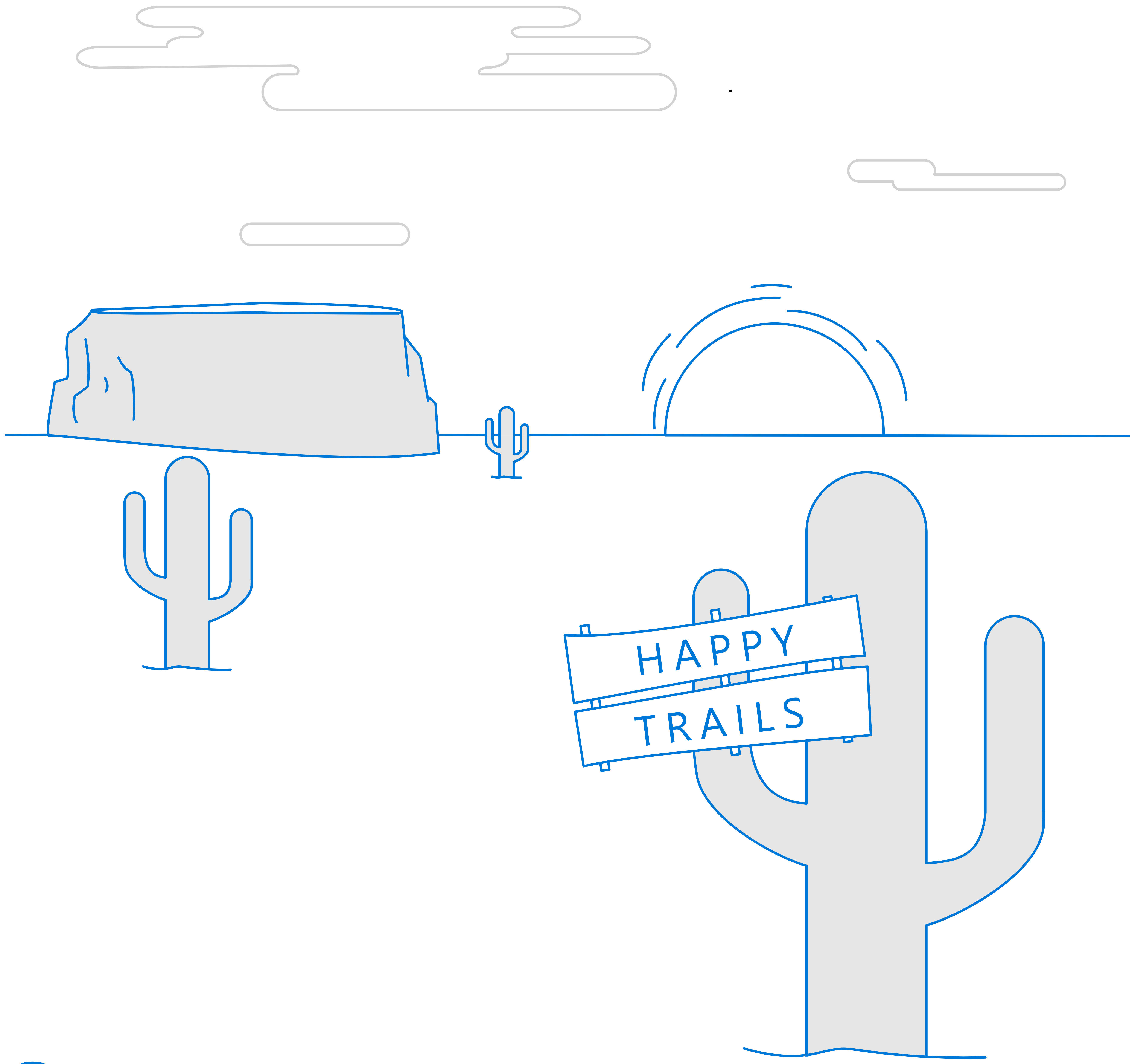
Declarative

```
/* Example Table*/
GO

PRINT N'Creating [dbo].[territory]...';

GO
CREATE TABLE [dbo].[territory] (
    [territoryID] INT NOT NULL,
    [territoryName] NVARCHAR(50) NOT NULL,
    [DATECreated] DATETIME2(7) NOT NULL,
    CONSTRAINT [PK_TERRITORY PRIMARY KEY CLUSTERED
([ID])]
);


/* Declarative
 *
 * Add territoryZip name
 *
 */

CREATE TABLE [dbo].[territory] (
    [territoryID] INT NOT NULL,
    [territoryName] NVARCHAR(50) NOT NULL,
    [DATECreated] DATETIME2(7) NOT NULL,
    [territoryZip] NVARCHAR(50) NOT NULL,
    CONSTRAINT [PK_TERRITORY PRIMARY KEY CLUSTERED
([ID])]
);

/* msbuild to project */
msbuild addColumn.csproj /t:territory
```

Forging your way through the new DevOps frontier

The new DevOps approach for app and database deployment offers vast improvements in efficiency and agility to those who master it. By using database operations to help you connect your database to your app, you can avoid the unruly and tedious update process that haunts many cowboys today.

There are three methods of database operations – ORM, Imperative and Declarative – that all support the systematic process of DevOps database deployment and set you up for success. Microsoft works closely with developers and partners to ensure you have products, tools and drivers you need to smooth your way – whichever path you choose to take.



Put the giddy in your giddy-up with tools from Microsoft

Entity Framework, a collection of drivers, is free from Microsoft.
Explore Entity Framework

ReadyRoll by Redgate integrates seamlessly in Visual Studio
Explore Readyroll

SQL Server Data Tools (SSDT) is built-in at no additional cost with Visual Studio.
Explore SSDT