Professionele Bachelor Toegepaste Informatica

Database DevOps

Christof Nies

Promotoren:

Dries Van Dyck — Accenture NV
Bart Wetzels — Accenture NV
Bart Clijsner — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Database DevOps

Christof Nies

Promotoren:

Dries Van Dyck — Accenture NV
Bart Wetzels — Accenture NV
Bart Clijsner — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Na vier jaar aan Hogeschool PXL kan ik met trots mijn eindwerk voorleggen. In deze jaren heb ik veel bijgeleerd zowel op vlak vlak van kennis als op persoonlijk vlak. De mogelijkheid om deze opleiding te volgen, dit eindwerk te schrijven en mijn stage te volgen heb ik te danken aan vele personen waarvan ik er een paar specifiek wil bedanken.

Eerst en vooral wil ik Accenture bedanken voor de leerrijke stageperiode. Binnen mijn stagebedrijf wil ik mijn stagebegeleiders Dries Van Dyck en Bart Wetzels bedanken voor de goede begeleiding, fijne samenwerking en kennis die ze mij doorgaven. Tot slot wil ik ook Bart Certyn, de applicatieontwikkelaar op de stageplek, bedanken voor zijn begeleiding tijden de stage.

Daarnaast zou ik ook graag mijn hogeschoolpromotor, Bart Clijsner, willen bedanken voor de opvolging van mijn stage en de zeer goede begeleiding van dit eindwerk.

Mijn ouders wil ik ook bedanken voor de steun die ze me gaven doorheen mijn hogeschool carrière en de mogelijkheid om deze opleiding te volgen. Ten slotte wil ik ook nog mijn vriendin, Lori Delli Santi, bedanken om mij te steunen en te motiveren tijdens mijn studies.

Abstract

De stageopdracht is onderverdeeld in twee opdrachten:

- Uitwerking van een *proof of concept* (POC) van de tool Liquibase
- Technische implementatie van verbeteringen aan een *error monitoring tool*

Ten eerste wordt er een *POC* uitgewerkt van de tool 'Liquibase'. Deze tool wordt toegepast tijdens het *deployment process* van de applicatie (*backoffice system* bij de klant) en dient om aanpassingen in een databasestructuur op te volgen, oftewel als versiebeheer voor databasecode.

Liquibase moet SQL-scripts uitvoeren op een Oracle Database en bijhouden welke scripts al uitgevoerd zijn. Hiernaast wordt het ook gebruikt om aanpassingen op een omgeving over te brengen naar andere omgevingen. Ook wordt er een vergelijkende studie gevoerd naar tools met gelijkaardige functionaliteiten. Op basis van een Fit/Gap-analyse wordt gekeken of een andere tool beter voldoet aan de eisen.

De tweede opdracht omvat een technische implementatie van verbeteringen aan een *error monitoring tool*. Deze tool wordt gebruikt voor het opvolgen en melden van functionele en technische fouten op de applicatie bij de klant. Deze tool bevat zowel een frontend, die gemaakt is met Angular, als een backend geschreven in Java.

De onderzoeksopdracht is een uitbreiding op de eerste stageopdracht. Er wordt onderzoek gevoerd worden naar de integratie van DevOps op een database. Hierbij worden eerst alle DevOps processen uitgelegd, gevolgd met een literatuurstudie. In de literatuurstudie is er voornamelijk gekeken naar de verschillende voordelen en nadelen van database DevOps, alsook naar verschillende tools die nodig zijn om dit te realiseren.

Na de literatuurstudie wordt er een POC gemaakt om aan te tonen hoe de database opgenomen kan worden in de geautomatiseerde processen die DevOps met zich mee brengt. Als laatste wordt er een conclusie gevormd over het onderzoek.

Inhoudsopgave

Lijst van gebruikte figuren

Lijst van gebruikte tabellen

Lijst van gebruikte afkortingen

Afkorting	Betekenis
API	Application programming interface
AI	Artificiële intelligentie
aaS	As-a-service
CD	Continuous delivery
CI	Continuous integration
CLI	Command line interface
DevOps	Development & Operations
POC	Proof of concept
NDA	Non-Disclosure Agreement

Inleiding

Deze bachelorproef is een samenstelling van een stageverslag en een onderzoek. De stage is uitgevoerd bij Accenture, daarom wordt er eerst uitgelegd wie, wat Accenture is. Hierna worden de stageopdrachten besproken samen met de opgeleverde resultaten van deze opdrachten. Op het einde van het stageverslag is er ook nog een conclusie over de resultaten en een reflectie op de stage.

Het tweede deel is het onderzoek. Er zal onderzoek gevoerd worden naar de integratie van DevOps bij een database. Hierbij worden eerst alle DevOps processen uitgelegd, gevolgd met een literatuurstudie. In de literatuurstudie gaat er voornamelijk gekeken worden naar de verschillende voordelen en nadelen van database DevOps, alsook naar verschillende tools die nodig zijn om dit te realiseren.

Na de literatuurstudie gaat er een POC gemaakt worden om aan te tonen hoe de database opgenomen kan worden in de geautomatiseerde processen die DevOps met zich mee brengt. Als laatste wordt er een conclusie gevormd over het onderzoek.

I. Stageverslag

1 Bedrijfsvoorstelling

1.1 Accenture [1] [2]

Accenture is een globaal organisatieadviesbureaus dat diensten biedt voor management, technologie en *outstourcing*. Deze diensten worden aangeboden in meer dan 40 industrie- en bedrijfstakken. Accenture richt zich op advies, transformatie & implementatie van allerhande oplossingen voor haar klanten zoals strategie, marketing, IT-infrastructuur, enz. Door de aard van haar activiteiten richt Accenture zich vooral op grote bedrijven waaronder de meerderheid uit de Fortune 500.

Accenture heeft kantoren in meer dan 200 steden in 52 landen. De hoofdvestiging ligt in Dublin, Ierland en Accenture heeft globaal meer dan 469.000 werknemers.

Accenture, maakt geen gebruik van standaard technologieën of programmeertalen maar richt zich eerder op klant specifieke oplossingen. Deze worden vastgelegd door een analyse aan de hand van de eisen van een klant of op basis van wat de klant al in gebruik heeft. De programmeertaal en gebruikte tools en *frameworks* binnen de stage worden later in het eindwerk toegelicht.

Hieronder worden ook nog de verschillende departementen binnen Accenture besproken:

accenturestrategy

Accenture Strategy helpt klanten bepaalde bedrijfsresultaten te bereiken door sectorspecifieke strategieën te definiëren en uit te voeren. Hierbij combineert het haar expertise binnen bedrijfs-, technologie- en operatiestrategieën met een goed inzicht in hoe technologie de industrie- en bedrijfsmodellen kan beïnvloeden.

accentureconsulting

Accenture Consulting biedt haar klanten branchespecifieke consultancydiensten, evenals functionele en technologische consultancydiensten. Hun consultancy ervaring stelt de klanten in staat om transformationele veranderingen te ontwerpen en te implementeren. Zo ondersteunen ze de grootste bedrijven in de wereld baanbrekende beslissingen te maken.

accenturedigital

Accenture Digital combineert haar expertise in digitale marketing, mobiliteit en analyse om klanten te helpen met het optimaliseren van de *user experience*. Het helpt ook met het creeëren van nieuwe producten en bedrijfsmodellen. Verder ondersteunen ze hun klanten ook bij het optimaliseren van de digitale mogelijkheden van hun bedrijf.

Accenture Technology richt zich op technologische trends. Dus ook in dit aspect helpt Accenture hun klanten. Door te anticiperen op de veranderingen in de wereld van technologie, zorgt Accenture Technology ervoor dat hun klanten voor blijven op deze digitale curve. Wanneer er een verandering is, worden klanten aangezet en geholpen om mee te gaan met deze transformatie.

Accenture Operations biedt bedrijfsproces-, infrastructuur-, beveiligings- en clouddiensten aan. Ze beheren IT-infrastructuur en bedrijfsprocessen, op basis van as-a-service (aaS), namens klanten aangedreven door data en artificiële intelligentie (AI) om hun productiviteit, ervaring en prestaties te optimaliseren.

2 Voorstelling stageopdracht

De volledige stage is onderverdeeld in twee opdrachten:

1. Uitwerking van een *proof of concept* (POC) van de tool Liquibase
2. Technische implementatie van verbeteringen aan een *error monitoring tool*

2.1 Opmerking

In dit project is een *Non-Disclosure Agreement* (NDA) van toepassing . Dit wil zeggen dat zowel de identiteit van de klant of het bedrijf alsook de scope van het project niet vermeld mag worden. Wat wel vrijgegeven mag worden is een high-level beschrijving van de verwachtingen van de klant en welke technologieën gebruikt werden tijdens de stageopdrachten.

2.2 Opdracht 1: Uitwerking POC van Liquibase

Momenteel worden databasescripts bij de klant nog altijd manueel aangeleverd en uitgevoerd. Als er iets misloopt, moet dit ook manueel nagekeken worden. Dit zorgt ervoor dat het *deployen* van de database lang kan duren en veel problemen met zich kan meebrengen. Daarom wil Accenture, bij de klant, nu overstappen op een DevOps manier van werken waarbij het *deploymentproces* van de database geautomatiseerd wordt. Dit brengt ook een extra voordeel met zich mee: er kan gecontroleerd worden welke scripts al zijn uitgevoerd op de database.

Om het *deploymentproces* te veranderen wordt er gebruik gemaakt worden van een tool. De tool die aangeraden werd is Liquibase. Dit is een opensourcetool die gebruikt wordt om aanpassingen in een databasestructuur op te volgen en om scripts uit te voeren op een database. Eerst wordt er gekeken naar andere tools die op de markt zijn. Zowel open source als commerciële tools worden hiervoor vergeleken. Dit gebeurt aan de hand van een Fit/Gap-analyse.

De stageopdracht bestaat dus uit het maken van een POC van de gekozen tool. Hierbij wordt er gekeken of het mogelijk is om deze tool te integreren binnen het huidige *deploymentproces* . Daarnaast worden bepaalde functionaliteiten van de tool getest en wordt er gekeken of deze daadwerkelijk gebruikt kunnen worden.

2.3 Opdracht 2: Implementatie error monitoring tool

De tweede opdracht omvat een technische implementatie van verbeteringen aan een *error monitoring tool*. Deze tool wordt gebruikt voor het opvolgen en melden van functionele en technische fouten in de applicatie van de klant. Er wordt een integratie naar andere systemen en platformen binnen het applicatielandschap voorzien. Dit bevat onder andere het opstellen van een technisch ontwerp, een link naar REST API's uittekenen en implementeren, etc. De implementatie van de backend gebeurt in Java en de frontend wordt gemaakt in Angular.

3 Uitwerking stageopdracht 1

Stageopdracht één staat in het teken van DevOps waarbij er een POC uitgewerkt wordt in verband met een tool die het *deployment*-proces van de database automatiseert alsook bijhoudt welke scripts uitgevoerd zijn.

3.1 Onderzoek van de tools

In dit deel wordt er een korte bespreking gegeven over de verschillende tools die gekozen zijn voor de Fit/Gap-Analyse. Er is gekozen voor twee commerciële tools en twee opensourcevarianten.

3.1.1 Liquibase

Deze niet-commerciële tool werd via een extern persoon aangeraden aan de klant en Accenture. Hierdoor was het een logische keuze om deze tool op te nemen in de analyse.

3.1.2 Flyway

Flyway is een grote concurrent van Liquibase op het gebied van opensourcesoftware. Daarom wordt Flyway ook in deze analyse opgenomen.

3.1.3 Datical

Datical is een commerciële tool die gebouwd is op Liquibase. Het bedrijf Datical doet zelf regelmatig updates aan Liquibase om deze tool te optimaliseren. De additionale fucntionaliteiten van Datical lijken interessant om in deze analyse bij te voegen. Ook heeft Accenture een partnership met Datical wat de relevantie van deze tool benadrukt.

3.1.4 Red Gate Development Suite for Oracle

Red Gate software lijkt een relevante speler op vlak van "versie controle voor databases". Zij hebben wereldwijd ongeveer 800.000 gebruikers en 91 van de Fortune 100 gebruiken software van Redgate. Daarbuiten hebben ze ook nog eens negentien jaar ervaring in databaseoplossingen. Verder heeft Redgate meer dan 19 jaar ervaring in databaseoplossingen. Hierdoor lijkt Redgate een geschikt softwarepakket om toe te voegen in deze analyse.

3.2 Fit/Gap-analyse

Een Fit/Gap-analyse wordt gebruikt om elk functioneel gebied in een bedrijfsproject of bedrijfsproces te evalueren om een specifiek doel te bereiken. Het omvat het identificeren van belangrijke gegevens of componenten die passen in een bedrijfssysteem en lacunes waarvoor oplossingen nodig zijn. Deze techniek is gebaseerd op verschillende doelstellingen, gericht om te bepalen welke de belangrijkste componenten zijn die nodig zijn om de best practice binnen een organisatie te bereiken. [3]

De analyse is beperkter opgesteld dan andere Fit/Gap-analyses aangezien enkel een globaal beeld geschetst wordt van de mogelijkheden van elke tool. Er zijn verschillende *requirements* opgesteld aan de hand van de wensen van de klant en door de verschillende functionaliteiten te bestuderen waarover de tools beschikken. Deze *requirements* werden hierna beantwoord voor elke tool. In *tabel 1* is de volledige analyse weergegeven.

SVCT Fit-Gap Analysis	Liquibase	Flyway	Datical	Redgate Deployment Suite for Oracle
Licensing?	Open source	Open source	Commercial - pricing: ?	Commercial - pricing: €1585/user
Cli-tool or Graphical UI	CLI-tool	CLI-tool	Both	Both
Plugin for Jenkins?	No	Yes (Last release: 2y ago)	Yes	No
Integration with TFS	No	No	Yes	Connection from database to TFS possible
Possibility to create your own plugin/extension?	Yes, they have an JAVA API and Extension support		No, since Datical already is an extension of Liquibase	No since it is not an opensourceproject
Can it be used with an existing database?	Yes	Yes	Yes	Yes
Is Oracle version 12.1 supported?	Yes	Free version only 12.2 & 18.3	Yes	No version specified, "an Oracle database is required"
In which language is the tool implemented?	Java	Java	/	/
When was the last release of the tool?	29-01-19	12-02-19		
How is the support?	Community support	Community support	Help desk, Web portal	Support team, support forum
Staging database?	No	No	No (Delphix)	Yes
Feedback on naming conventions etc. In SQL-scripts	No	No	Yes	No
Pre-deployment validation	No	No	Yes	No

Tabel 1: Fit/Gap-analye

Er is gekozen om verder te gaan met de tool Liquibase en hiervoor een POC uit te werken. Flyway lijkt geen optie aangezien er gebruik wordt gemaakt van een Oracle 12.1 database en Flyway pas ondersteuning biedt vanaf versie 12.2. Verder is er een voorkeur om de mogelijkheden van een opensourcetool te bekijken vooraleer er gebruik gemaakt gaat worden van een commerciële tool. Daarnaast is Datical een uitbreiding op het Liquibase-framework en is het in een latere fase mogelijk om over te schakelen.

3.3 Liquibase-terminologie

Binnen Liquibase wordt er gebruik gemaakt van een bepaald jargon om commando's en objecten te benoemen. Hieronder is een oplijsting van verschillende termen binnen het framework.

Commando/Object	Betekenis
Changeset	In een *changeset* worden de veranderingen op een database bijgehouden, liefst één aanpassing per *changeset*. Oftewel bevat de *changeset* één SQL-script en wordt het geïdentificeerd aan de hand van de attributen "Id", "Author" en het pad van de *changelog*.
Changelog	Een *changelog* is een verzameling van verschillende *changesets*. Een nieuwe *changeset* wordt op het einde van een *changelog* toegevoegd.
DatabaseChangelogTable	De *databaseChangelogTable* is een tabel die bijhoudt welke scripts in een *changelog* al zijn uitgevoerd.

Tabel 2: Liquibase terminologie

3.4 Oplossing 1

Omdat er bij de klant gewerkt wordt met SQL-scripts wilden ze graag gebruik maken van de SQL geformatteerde changelog in plaats van de XML-changelog. In dit hoofdstuk wordt in het kort een tool besproken die gemaakt is om een changelog te genereren. Hierbij worden ook nog enkele functionaliteiten van Liquibase uitgelegd die in de changelog zijn opgenomen.

3.4.1 Hulptool voor automatisering

Het gebruik van Liquibase werkt door steeds bij een verandering op de database, het script eerst handmatig aan de changelog toe te voegen samen met de juiste attributen. Doordat er bij de klant scripts worden aangeleverd vanuit verschillende teams/bedrijven, is het moeilijk om een changelog handmatig bij te houden. Ook is het moeilijk deze teams een nieuwe manier van werken aan te leren om één tool correct te kunnen gebruiken, dit vergt immers veel extra tijd. Daarom heb ik een CLI-tool ontwikkeld die van een folder met scripts een changelog genereert.

Doordat het een CLI-tool is kan deze makkelijk opgeroepen worden door bijvoorbeeld Jenkins. Deze kan hierna ook Liquibase oproepen waardoor er een automatisch proces ontstaat dat updates op een database uitvoert.

3.4.1.1 3.4.2.1 Attributen

Een voorbeeld van het gebruik van de tool is duidelijk te zien op Figuur 1. Er worden drie parameters verwacht:

1. Author: een changeset verwacht een attribuut "*author*", deze wordt meegegeven met de hulp tool zodat die dit label kan meegeven.
2. Script folder: het twee attribuut is de locatie van de folder met de scripts waar changesets van gemaakt moeten worden.
3. Changelog locatie: als laatste moet de locatie van de te genereren changelog meegegeven worden.

```
C:\Users\christof.nies>java -jar parser.jar Author C:\Users\christof.nies
\Documents\Scripts C:\Users\christof.nies\Documents\changelog.sql
```

Figuur 1: Gebruik hulp tool

3.4.2.2 Resultaat

In Figuur 2 is een mogelijke uitkomst te zien. Er zaten twee scripts in de scriptfolder, hiervan zijn changesets gegenereerd en in de changelog gezet.

De "—liquibase formatted sql" commentaar is vereist voor Liquibase, evenals de "—changeset" commentaar met de attributen "auteur:id". Deze worden voorzien door de extra tool en tussenin wordt ook de SQL-code van elk script gezet.

```
--liquibase formatted sql

--changeset author:001_Create_Customers
CREATE TABLE customers
( customer_id number(10) NOT NULL PRIMARY KEY,
  firstName varchar2(50) NOT NULL,
  lastName varchar2(50) NOt NULL,
  postalCode number(20)
);

--changeset author:002_Alter_Customers
ALTER TABLE CUSTOMERS RENAME COLUMN customer_id to customerId;
```

Figuur 2: SQL Changelog

3.5 Oplossing 2

Tijdens het bespreken van oplossing één met de klant stelde zich enkele beperkingen van de gebruikte tool. Zo kan Liquibase niet aangeven welke scripts zouden falen. Verder stop Liquibase met het uitvoeren van scripts bij de eerste error. Concreet zullen enkel scripts die voor de error staan, de database hebben geüpdatet. Dit lijkt echter niet altijd nodig aangezien er via *dependencies* tussen scripts wel een mogelijkheid zou bestaan om verder te gaan met het updaten van de database.

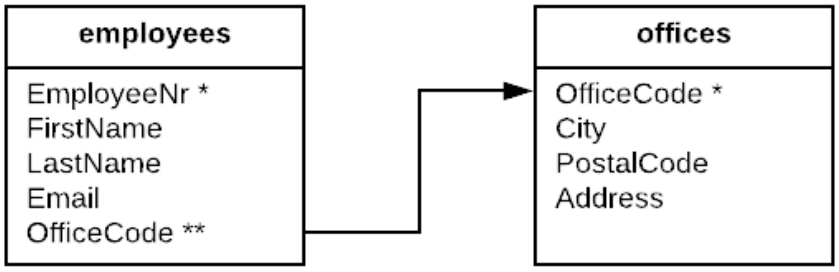
Voor de hiervoor vermelde reden wordt er overgeschakeld worden op XML-changelogs. De changesets gaan ook niet meer achteraan de changelog toegevoegd worden, maar er gaat elke keer een nieuwe changelog gegenereerd worden.

In dit hoofdstuk wordt de *proof of concept* stap voor stap uitgelegd.

3.5.1 Databasestructuur

Er wordt gebruik gemaakt van een fictieve database om er zeker van te zijn dat de *Non-Disclosure Agreement* (NDA) nageleefd wordt. De database is in realiteit groter, maar in theorie zou Liquibase op dezelfde manier moeten functioneren.

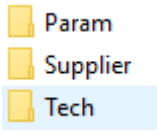
De gebruikte structuur is te zien op Figuur 3. Het is een database met twee tabellen "Employees" en "Offices".



Figuur 3: Databasestructuur

3.5.2 Stap 1: Levering van scripts

Wanneer er een nieuwe release wordt uitgevoerd, worden er door drie bronnen scripts aangeleverd om de database te hervormen. Deze worden dan in een mappenstructuur verwerkt zoals te zien is op Figuur 4.



Figuur 4: mappenstructuur

In de demo wordt gebruik gemaakt van scripts die simpele aanpassingen doen op de database structuur of data hieraan toevoegen. Een voorbeeld van zo een script is gegeven op Figuur 5, deze verandert de naam van een kolom in de tabel “Employees”.

```
ALTER TABLE Employees RENAME COLUMN EmployeeNr to EmployeeId;
```

Figuur 5: Voorbeeld script

3.5.3 Stap 2: Changelog genereren via hulptool

De volgende stap is het inroepen van de hulptool. Op Figuur 6 is te zien dat de hulptool twee attributen vereist. Deze vereisten zijn:

1. De map waar de mappenstructuur uit het vorige hoofdstuk in zit
2. De locatie waar de changelog geplaatst moet worden.

De vorige versie vereiste een auteur maar deze wordt nu bepaald aan de hand van de mappenstructuur. Alle scripts in bijvoorbeeld “Param” krijgen “param” als auteur.

```
C:\Demo>java -jar parser.jar C:\Demo\Scripts C:\Demo\scriptArchive.xml
Creating new scriptArchive file...
Adding metadata to scripts.
Processing supplier scripts
Processing tech scripts
Processing param scripts
ScriptArchive succesfully generated.
```

Figuur 6: werking hulptool

Vervolgens zal er een script gegenereerd worden gelijkaardig aan degene die te zien is op Figuur 7. Er zijn twee *changesets* zichtbaar maar in deze demo wordt er gebruik gemaakt van vijf scripts. Er wordt een extract getoond van het script vanwege de omvang.

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog-ext http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd
    http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.6.xsd"
    logicalFilePath="scriptArchive" >
    <preConditions>
        <dbms type="oracle" />
        <runningAs username="MERCATOR" />
    </preConditions>

<changeSet id="001_Alter_Employees" author="supplier" failOnError="false" >
<comment>
ALTER TABLE Employees RENAME COLUMN EmployeeNr to EmployeeId;
--test
</comment>

<sqlFile path="C:\Demo\Scripts\Supplier\001_Alter_Employees.sql" stripComments="true" />
</changeSet>

<changeSet id="002_Alter_Employees" author="supplier" failOnError="false" >
<preConditions onFail="CONTINUE"  onFailMessage="001_Alter_Employees of supplier could not be executed due too failed dependencies">
<changeSetExecuted id="001_Alter_Employees" author="supplier" changeLogFile="scriptArchive" />
</preConditions>
<comment>
ALTER TABLE Employees
ADD job_title VARCHAR2(100);
</comment>
```

Figuur 7: Extract changelog

Deze *changelog* is opgebouwd met XML-tags. Zo wordt een *changeset* nu aangeduid met “<changeset>”. Deze tag heeft nu ook een extra attribuut gekregen “failOnError”. Dit zal er voor zorgen dat wanneer deze *changeset* faalt, niet de hele executie van scripts faalt. Liquibase zal dus proberen om de scripts die erna komen verder uit te voeren in plaats van een error te gooien.

Ook is er bij de configuratie in de “databaseChangeLog” tag een attribuut logicalFilePath toegevoegd. Deze heeft tot doel dat waar de *changelog* gegenereerd wordt van geen belang is waardoor er geen problemen ontstaan met de md5-hash die Liquibase gebruikt om te controleren of een script aangepast is.

Om het gebruik te faciliteren is er ook de tag “comment” toegevoegd. Hier worden de eerste drie regels van een script geplaatst. Dit resulteert in een snelle controle van opgenomen scripts in de database doordat de kolom “comment” van de Liquibase tabellen ingevuld is.

Verder is er de reden waarom de *SQL-changelogs* niet bruikbaar waren, de mogelijkheid om precondities toe te voegen waarbij er wordt gekeken of een voorafgaand scripts is uitgevoerd. Als er niet aan deze conditie voldaan wordt, gaat Liquibase de bijhorende *changeset* overslaan. Dit laatste wordt gedefinieerd door het “onFail” attribuut.

3.5.4 Stap 3: Liquibase

Nu er een *changelog* is gegenereerd kan Liquibase worden aangeroepen om de scripts uit te voeren op de database. Hiervoor moet er een *properties* bestand aangemaakt worden, dit bestand is te zien op Figuur 8.

```
#Liquibase.properties
driver: oracle.jdbc.OracleDriver
classpath: C:\\Demo\\ojdbc8.jar
url: jdbc:oracle:thin:@localhost:1521:orcl
username: christof
password: pxl
logLevel: INFO
liquibaseSchemaName: LOGSCHEMA
```

Figuur 8: Liquibase properties bestand

Om Liquibase te gebruiken moeten er verschillende attributen meegegeven worden. Door dit in een *properties* bestand te doen moet dit niet aan ieder commando meegegeven worden. Hieronder volgt een korte uitleg over elk attribuut.

Attribuut	Beschrijving
Driver	Een JDBC-driver is nodig om commando's uit te voeren op de database
Classpath	Het pad dat verwijst naar de locatie van de JDBC-driver
URL	URL van de database die lokaal geïnstalleerd is en waar Liquibase commando's op moet uitvoeren
Username	Gebruikersnaam om in te loggen op de database
Password	Wachtwoord van de gebruiker
LogLevel	Liquibase moet alleen essentiële info loggen dus deze is ingesteld op "INFO"
LiquibaseSchemaName	Het schema waar Liquibase tabellen genereert om bij te houden welke script uitgevoerd zijn

Tabel 3: Liquibase properties beschrijving

Om de scripts uit te voeren op de database moet er nu enkel nog gebruik gemaakt worden van het "update" commando dat te zien is op Figuur 9. Dit commando moet uitgevoerd worden op dezelfde locatie als waar het *properties* bestand zich bevindt. Er wordt nog een attribuut meegeven en dat is de locatie van de *changelog.* Achter het commando wordt er ook nog het logbestand geconfigureerd.

```
liquibase --changeLogFile="C:\Demo\scriptArchive.xml" update > log.txt
```

Figuur 9: Liquibase update commando

3.5.5 Resultaat

Ten eerste is er een logbestand aangemaakt, deze is te zien op Figuur 10 Hier kan er gekeken worden of alles succesvol verliep. Als er errors optreden kunnen deze worden teruggevonden met een bijhorende melding.

```
Starting Liquibase at Tue, 04 Jun 2019 13:59:35 CEST (version 3.6.3 built at 2019-01-29 11:34:48)
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: Successfully acquired change log lock
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: Creating database history table with name: LOGSCHEMA.DATABASECHANGELOG
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: Reading from LOGSCHEMA.DATABASECHANGELOG
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: SQL in file C:\Demo\Scripts\Supplier\001_Alter_Employees.sql executed
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: ChangeSet scriptArchive::001_Alter_Employees::supplier ran successfully in 28ms
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: SQL in file C:\Demo\Scripts\Supplier\002_Alter_Employees.sql executed
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: ChangeSet scriptArchive::002_Alter_Employees::supplier ran successfully in 10ms
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: SQL in file C:\Demo\Scripts\Tech\001_Create_Customers.sql executed
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: ChangeSet scriptArchive::001_Create_Customers::tech ran successfully in 10ms
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: SQL in file C:\Demo\Scripts\Tech\002_Alter_Customers.sql executed
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: ChangeSet scriptArchive::002_Alter_Customers::tech ran successfully in 10ms
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: SQL in file C:\Demo\Scripts\Param\001_Insert_Offices.sql executed
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: ChangeSet scriptArchive::001_Insert_Offices::param ran successfully in 7ms
[34mINFO[0;39m 4/6/19 13:59 PM: liquibase: Successfully released change log lock
Liquibase: Update has been successful.
```

Figuur 10: Liquibase log file

Om terug te gaan op het voorbeeld script uit hoofdstuk 3.5.4.1, kunnen we gaan kijken in de “Employees” tabel of de verandering is doorgekomen. Dit is te zien op Figuur 11.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	EMPLOYEEID	NUMBER	No	(null)	1	(null)
2	FIRSTNAME	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	LASTNAME	VARCHAR2(50 BYTE)	No	(null)	3	(null)
4	EMAIL	VARCHAR2(100 BYTE)	Yes	(null)	4	(null)
5	OFFICECODE	VARCHAR2(20 BYTE)	No	(null)	5	(null)
6	JOB_TITLE	VARCHAR2(100 BYTE)	Yes	(null)	6	(null)

Figuur 11: Employees tabel

Als laatste gaat er in de Liquibase tabel gekeken worden. Deze tabel heet, indien deze niet zelf benoemd werd, “databasechangelog”. Hierin worden alle scripts bijgehouden die al uitgevoerd zijn op de database. Elke keer als er een update wordt gedaan, wordt deze dus aangevuld. Een deel van de huidige tabel is te zien op Figuur 12.

ID	AUTHOR	FILENAME	DATEEXECUTED	ORDEREXECUTED	EXECTYPE	MD5SUM
001_Alter_Employees	supplier	scriptArchive	04-JUN-19 01.59.38.857000...	1	EXECUTED	8:6f14abcfc8c7d470f43c3deff864963a
002_Alter_Employees	supplier	scriptArchive	04-JUN-19 01.59.38.887000...	2	EXECUTED	8:a8d875c11ee724b470cc75db05e726a1
001_Create_Customers	tech	scriptArchive	04-JUN-19 01.59.38.907000...	3	EXECUTED	8:3a92c91e322715fa10817180c622dd57
002_Alter_Customers	tech	scriptArchive	04-JUN-19 01.59.38.920000...	4	EXECUTED	8:75d2c5b5cad750f274f6ddadce855c59
001_Insert_Offices	param	scriptArchive	04-JUN-19 01.59.38.927000...	5	EXECUTED	8:5ed15f9dce27c3f3f07291e495c50e4e

Figuur 12: Databasechangelogtable

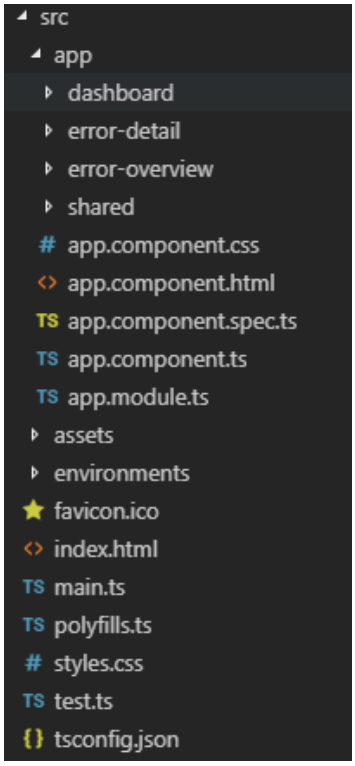
4 Uitwerking stageopdracht 2

Voor het opvangen en bekijken van errors is er een tool gemaakt, de "*error monitoring tool*". Er is reeds aanstalten gemaakt om een nieuwe versie hiervan te maken. In deze nieuwe versie waren er enkel nog een aantal problemen in de code die opgelost moeten worden. Stageopdracht twee houdt dus in om deze problemen op te lossen en enkele verbeteringen te implementeren.

4.1 Structuur frontend

Op Figuur 13 is de structuur van de frontend te zien. De inhoud van de mappen wordt uitgelegd in volgende opsomming:

- Dashboard: : in de tool komen meerdere grafieken voor. Deze grafieken worden zoals eerder vermeld gemaakt met hulp van de chart.js bibliotheek. In de dashboard folder worden deze grafieken allemaal apart geconfigureerd. Tot slot worden hier de html, css en typscript pagina's van de dashboard component gedefinieerd.
- Error-detail: hierin worden de html, css en typscript pagina's van de error-detail component in gedefinieerd.
- Error-overview: hierin worden de html, css en typscript pagina's van de error-overview component in gedefinieerd.
- In de shared folder worden de modellen, services en extra benodigdheden die gebruikt worden binnen de applicatie gedefinieerd.
- De app component bevat de router-outlet die ervoor zorgt dat er kan genavigeerd worden op de *error monitoring tool*. De *header* en het menu staan hier ook in gedefinieerd.
- De assets folder bevat de foto's die gebruikt worden binnen de componenten.
- De environments folder bevat configuraties voor verschillende omgevingen. Zo kan de tool gebouwd worden voor de lokale omgeving of voor de productieomgeving.

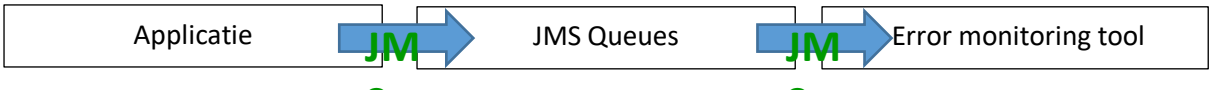


Figuur 13: Mappenstructuur frontend

4.2 Structuur backend

Wegens de NDA kan de structuur van de backend niet getoond worden. Alle aanpassingen die gedaan zijn waren op één controller die alle URL's beschikbaar stelt. Hierin zijn een paar extra methodes in aangemaakt en bestaand methodes in aangepast.

4.3 Lagen



Errors die voorkomen zijn excepties die onderverdeeld worden in een van de volgende gebieden:

1. Aanroepen van triggers
2. Asynchrone taken zoals creëren van een klant, updaten van een klant, etc.
3. OOB Errors

Er zijn drie verschillende queues die opgezet worden in een instantie van ActiveMQ:

1. ErrorMonitoring.Input.Queue.TriggerError
2. errorMonitoring.Input.Queue.AsyncError
3. ErrorMonitoring.Input.Queue.OOBError

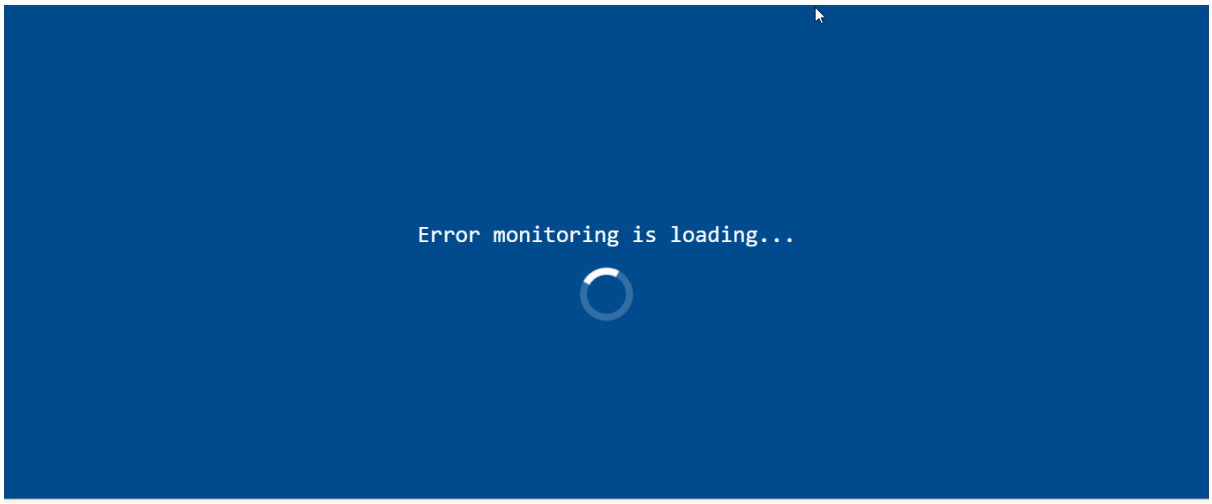
Op de *error monitoring tool* zijn er listeners geconfigureerd die JMS berichten inlezen van ActiveMQ queues. Deze berichten worden vervolgens verwerkt en geconverteerd in domein objecten van het type error en in de database gestopt. Hierna geeft de tool deze errors grafisch weer.

4.4 Error monitoring tool

In dit hoofdstuk gaan de *userinterface* (UI) en functionaliteiten van de *error monitoring tool* kort besproken worden.

4.4.1 Laadscherm

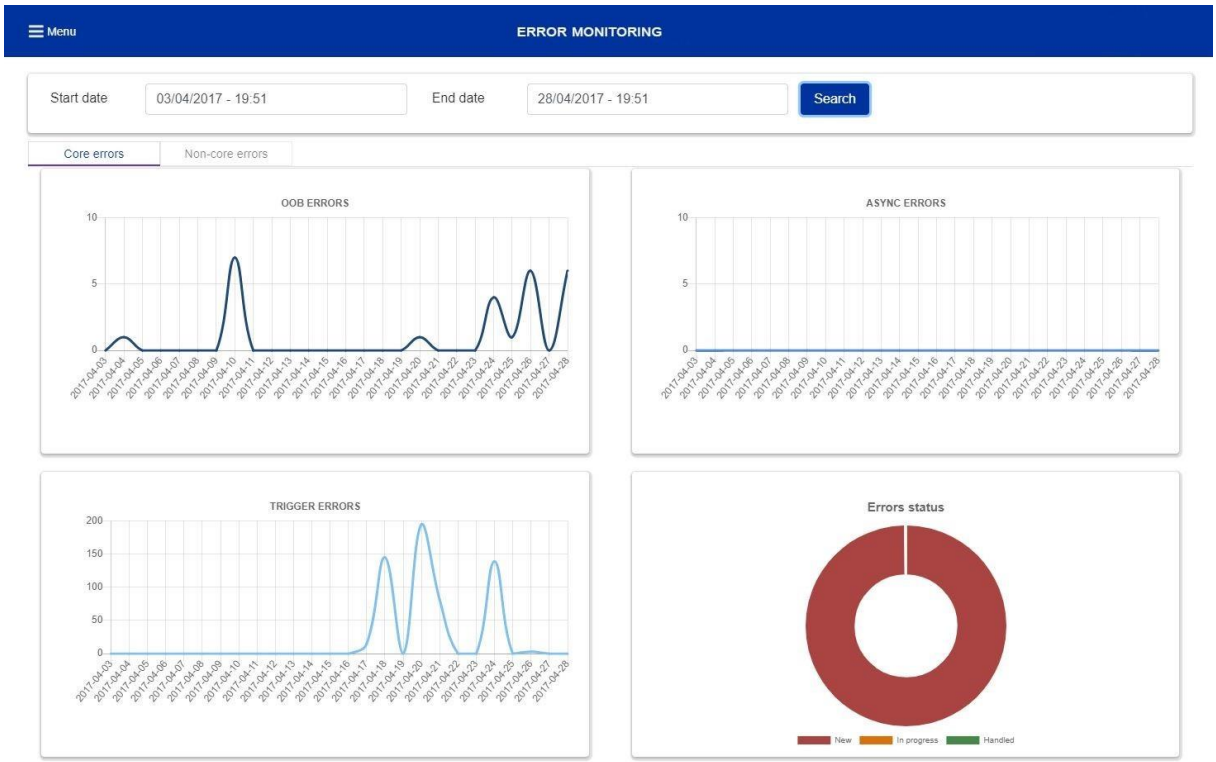
Tijdens het opstarten van de applicatie wordt er data ingeladen om verschillende grafieken mee te maken. Tijdens het laden hiervan wordt het scherm getoond dat te zien is op Figuur 14.



Figuur 14: laadscherm EM

4.4.2 Homepagina

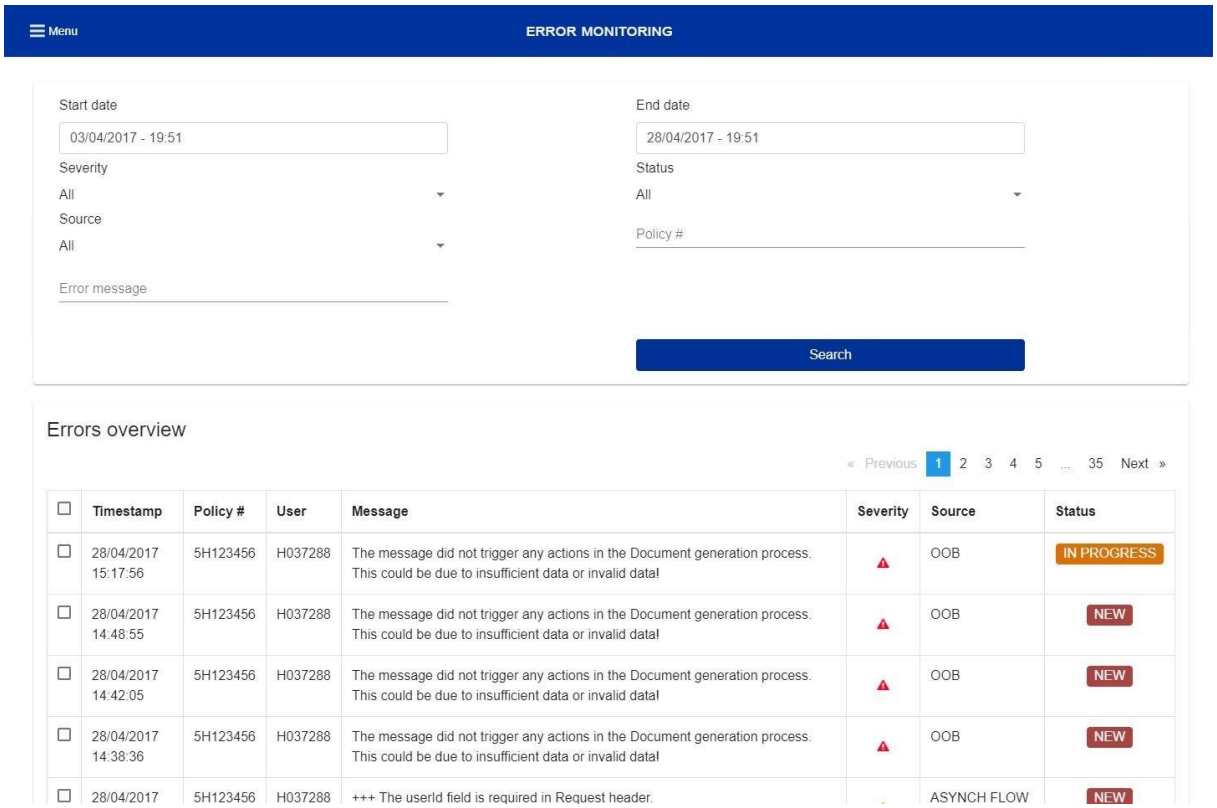
Op Figuur 15 is de homepagina van de applicatie te zien. Op deze pagina wordt alle data van de drie verschillende errors grafisch weergegeven. Men kan hier filteren op tijd, als het tijdsverschil kleiner wordt passen de grafieken zich hierop aan. De grafiek kan op jaren, maanden, dagen, uren en per 10 minuten data weergeven. Daarnaast is er ook nog de donutgrafiek, deze geeft aan hoeveel errors binnen de geselecteerde tijd afgehandeld, nieuw of in verwerking zijn.



Figuur 15: Homepagina EM

4.4.3 Zoekpagina

Op Figuur 16 is de zoekpagina te zien van de applicatie. Hier kunnen gebruikers op verschillende criteria filteren, bijvoorbeeld op tijd, status of bron van een error. Op basis van de ingevuld criteria wordt een lijst weergeven. Op deze pagina is het ook mogelijk om errors op afgehandeld of in verwerking te zetten. Verder is het mogelijk om op een rij te klikken, dit weergeeft een kader met meer details over een specifieke error.

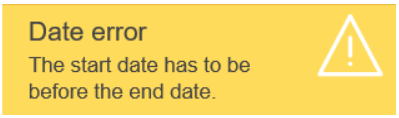


Figuur 16: Zoekpagina EM

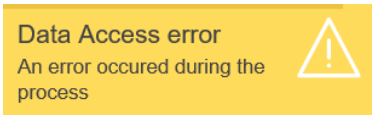
4.4.4 Errors binnen de applicatie

Er zijn ook drie verschillende errors die kunnen optreden tijdens het gebruik van de applicatie.

- Datum error: wordt getriggerd wanneer het tijdsbereik ongeldig is, bijvoorbeeld een einddatum die voor de startdatum valt. De melding die wordt weergeven bij dit soort errors is te zien op Figuur 17.



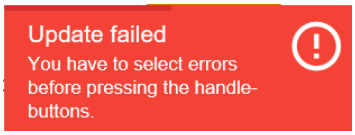
Figuur 17: Datum error EM



Figuur 18: Gegevenstoegangsfout EM

- Gegevenstoegangsfout: wordt getriggerd wanneer er iets misloopt bij het afhandelen van de gevraagde query in de backend. De melding die wordt weergeven bij dit soort errors is te zien op Figuur 18.

- Fouten bij het updaten: wordt getriggerd wanneer er een updateproces wordt uitgevoerd op ongeldige gegevens, bijvoorbeeld de knop voor errors af te handelen gebruiken zonder een of meerdere errors aan te duiden. De melding die wordt weergeven bij dit soort errors is te zien op Figuur 19.



Figuur 19: Updatefout EM

5 Reflectie

Ik leerde veel bijover de infrastructuur en gang van zaken op de IT-afdeling van een groot bedrijf. Aangezien we op school eerder kleinere projecten uitvoerden, was het erg leerrijk om te zien hoe grote projecten worden uitgewerkt in de bedrijfswereld. In het begin van mijn stage ondervond ik een aantal moeilijkheden. Zo vond ik het moeilijk te begrijpen wat alle omgevingen waren en hoe een applicatie en database onderhoud, *gebuild* en *gedeployed* werden. Daarom duurde het even voor ik wist wat en hoe ik tools moest analyseren voor de PoC-stageopdracht uit te werken. Verder wist ik niet dat een database opgenomen kon worden in bron controle. Hier had ik nooit bij stilgestaan, maar nu ik hier mee gewerkt heb zou ik niet meer zonder database versiecontrole werken.

Ik heb tijdens de stageopdracht van de PoC ook geleerd dat het belangrijk is dat er veel gecommuniceerd wordt met de klant. Door regelmatig de status te bezorgen van de PoC, krijgt de klant een beter inzicht van de toepassing ervan. Tijdens de vergaderingen geeft de klant richtlijnen over hun visie van de tool, alsook mogelijke extra functionaliteiten die geanalyseerd dienen te worden. Verder heb ik ook geleerd dat het heel belangrijk is om rekening te houden met het opvangen van fouten.

De afgewerkte PoC duidt aan dat Liquibase gebruikt kan worden binnen de klant zijn infrastructuur/omgevingen zodat er altijd gecontroleerd kan worden welke scripts al uitgevoerd zijn op de database. Toch is Liquibase niet honderd procent betrouwbaar aangezien alles afhankelijk is van een gegenereerd bestand. Daarnaast kunnen er menselijke fouten in voorkomen aangezien sommige scripts metadata vereisen die op een correcte manier geschreven moet zijn. Ook worden de scripts nog altijd manueel aangeleverd. Voor deze levering wordt verwacht dat dit op een constante manier gebeurt waardoor er snel een kleine fout in kan kruipen.

Gedurende de PoC-stageopdracht heb ik ook nog de opdracht gekregen om verder te werken aan een *error monitoring tool*. De Angular applicatie begrijpen verliep vlot aangezien deze niet zo uitgebreid was. De backend daarentegen was groter en het duurde even voordat ik begreep waar elk Javaproject binnen dit geheel voor diende. Tijdens deze opdracht heb ik geleerd over hoe grotere applicaties eruitzien en opgebouwd zijn. Daarnaast werd JBOSS gebruikt om de applicatie te *deployen* en debuggen, hierdoor heb ik de tool leren gebruiken en configureren. Op school werd hier voornamelijkTomcat voor gebruikt wat ook beter is tijdens het ontwikkelen. Ook heb ik tijdens dit project veel bijgeleerd over Maven, voor mij was dit eerder iets wat ik gebruikte om een goede mappenstructuur te hebben en makkelijke afhankelijkheden mee kon in laden. Tijdens de stage heb ik Maven uitgebreider gebruikt door *builds, clean installs, etc.* te doen. Ten slotte kan ik nog zeggen dat deze applicatie in de maand na mijn stage getest gaat worden in productie zodat deze kan gebruikt worden bij de klant.

II. Onderzoekstopic

1 Vraagstelling onderzoek

Al een aantal jaren is DevOps een hot-topic bij applicatieontwikkelaars. Het zorgt voor snellere releases, gestroomlijnde processen, automatisering, etc. Toch moeten ontwikkelaars nog vaak wachten op databasescripts om verder te gaan met hun werk. Het is vaak ook moeilijk om te bepalen welke versie van de database hoort bij de huidige versie van de applicatie. Dit zorgt ervoor dat de *time-to-market* van een product vaak langer is dan gewenst.

Dus, waarom zou er geen gebruik gemaakt kunnen worden van DevOps als er gesproken wordt over een database?

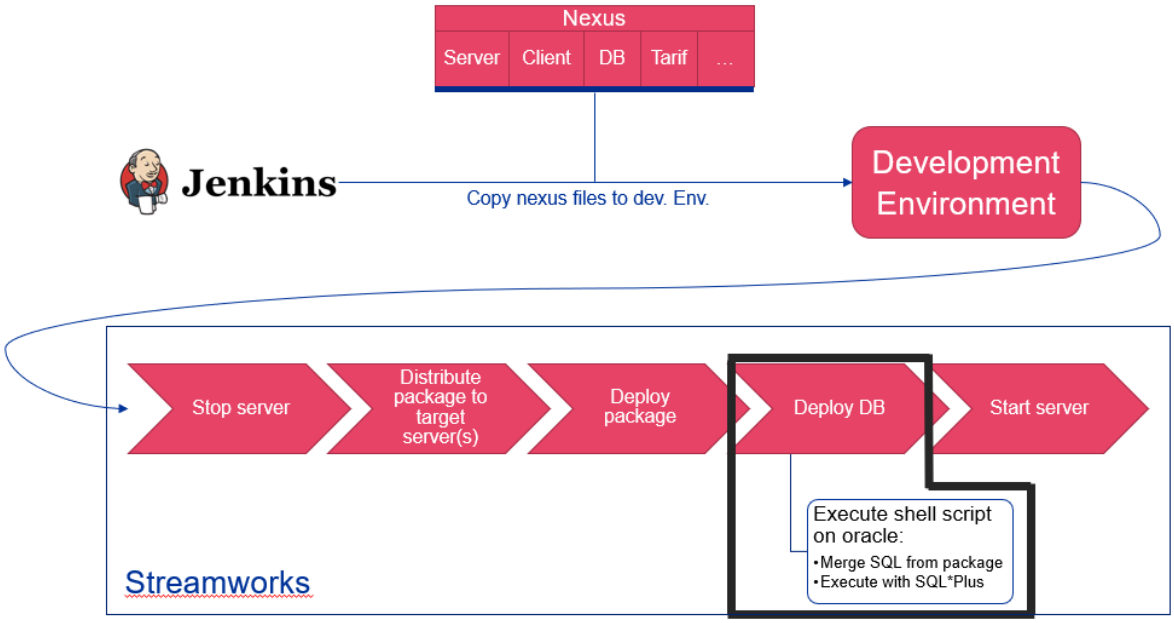
Een paar vragen die zeker aan bod moeten komen om een succesvol onderzoek te realiseren zijn:

- Wat zijn de voordelen van DevOps te gebruiken op een database?
- Welke tools of technologieën kunnen dienen om DevOps op een database te realiseren?
- Kan er gebruik gemaakt worden van dezelfde tools als bij de applicatiecode?
- Hoe ziet een workflow eruit wanneer DevOps gebruikt wordt op een database?

2 Onderzoeksmethode

Voor de stageopdracht moest er onderzocht worden of Liquibase kon geïntegreerd worden binnen het huidige *deploymentproces*, zoals te zien is op Figuur 20. Liquibase moet een upgrade zijn op de huidige manier van de database *deployment*. De belangrijkste eigenschappen waarnaar gekeken worden zijn:

- Bijhouden welke SQL-scripts zijn uitgevoerd;
- Het updaten van de database.



Figuur 20: Structuur stageopdracht

Deze kleine stappen naar versiecontrole op een database zijn het begin van database DevOps. In dit onderzoek gaat er gekeken worden naar de vele voordelen die versiecontrole bij een database

kunnen bieden. Naast de versiecontrole, dient DevOps ook voor het automatiseren van processen, frequentere releases en kwaliteitsvollere code.

Eerst gaat er in de literatuur gezocht worden naar verschillende bronnen die de voordelen van DevOps met een database benoemen, maar ook hoe dit in zijn werk gaat. Aan de hand van deze literatuurstudie gaat er ook gekeken worden waarom overwogen wordt om het database proces op een DevOps manier te laten verlopen. Met andere woorden, wat zijn de voordelen die het met zich meebrengt. Van deze literatuurstudie gaat er een persoonlijke reflectie gemaakt worden die een kritische samenvatting bevat over de artikels.

Hierna gaat er aan de hand van een eigen onderzoek gekeken worden naar verschillende tools die gebruikt kunnen worden om DevOps te werken met een database. Er gaat ook nog een POC worden opgesteld die gebruik gaat maken van verschillende tools. Deze POC gaat een DevOps manier van werken met een database weergeven. Dit gaat het proces zijn van database aanpassingen die ingecheckt worden via bron controle tot de uiteindelijke release.

Het doel van deze onderzoeksopdracht is om mensen te tonen dat DevOps werken met een database een goede keuze is. Het onderzoek is hierbij dus geslaagd wanneer er een duidelijk beeld gegeven kan worden over wat de voordelen zijn en hoe dit geïmplementeerd kan worden. Dit betekent dat het onderzoek geslaagd is wanneer alle vragen correct beantwoord zijn.

3 Uitwerking onderzoek

3.1 Inleiding

In de inleiding wordt er uitgelegd wat DevOps precies is. Vervolgens worden verschillende begrippen uitgelegd die verwant zijn aan het topic DevOps.

3.1.1 Wat is DevOps? [4, 5]

DevOps is de samenstelling van twee woorden “dev” (development; ontwikkeling) en “ops” (operations; activiteiten). DevOps is een cultuur en praktijk die ontwikkelaars en operationele teams integreert. Dit zorgt voor een betere samenwerking en productiviteit tussen voorheen geïsoleerde disciplines. Bij DevOps komt de gehele ontwikkelcyclus aan bod: product aflevering, doorlopende controle, kwaliteitstests, ontwikkeling van nieuwe functionaliteiten en onderhoud releases. Het hoofddoel van DevOps is bijdragen aan automatisering van de levenscyclus van een toepassing, zodat ontwikkelingstijden gereduceerd worden, de markintroductietijd korter wordt en de productkwaliteit beter is.

Volgens Donovan Brown is DevOps veel breder dan enkel het automatiseren van de ontwikkelcyclus. Zijn definitie is als volgt: “DevOps is the union of people, process, and products to enable continuous delivery of value to our end users.” [6]

3.1.2 Continuous Integration (CI) [7]

Continuous Integration is opgekomen als een best practice omdat softwareontwikkelaars hun code moeten integreren met de rest van het team. CI zorgt ervoor dat ontwikkelaars hun code delen op een gedeeld versiebeheersysteem telkens als er wijzingen zijn gedaan. Het is een proces voor het automatiseren van de build-fase en het testen van de code telkens wanneer een teamlid veranderingen in het versiebeheersysteem vastlegt.

3.1.3 Continuous Delivery (CD) [8]

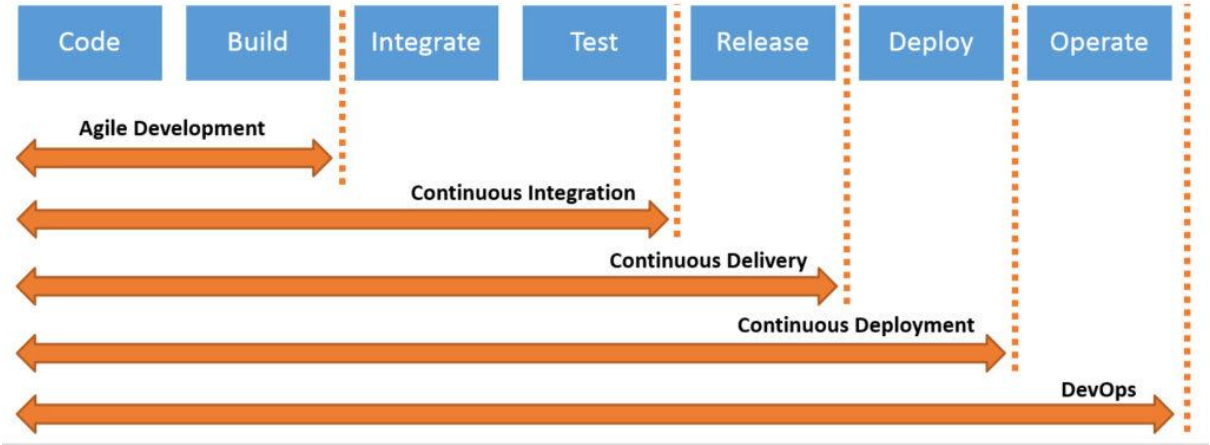
Continuous Delivery is een uitbreiding op het CI-proces waarbij ook het releaseproces geautomatiseerd wordt. Dit zorgt ervoor dat er altijd een release klaar staat die op de server kan uitgevoerd worden. Dit laatste is het *deployproces* van de applicatie en gebeurt manueel.

3.1.4 Continuous Deployment [8]

Continuous Deployment gaat nog een stap verder dan CD. Door deze praktijk toe te passen worden alle processen, van het code inchecken op een versiecontrolesysteem tot de applicatie uit te voeren op een server, geautomatiseerd. Er is geen menselijke ingreep meer, en alleen een gefaalde test zal ervoor zorgen dat veranderingen niet automatisch opgezet worden in productie.

3.1.5 Relatie van deze processen [8]

Zoals te zien is op Figuur 21 is CI een deel van zowel CD als Continuous Deployment. En is Continuous Deployment hetzelfde als CD, met het verschil dat Continuous Deployment een release automatisch op een server uitvoert. Deze praktijken, om de ontwikkelcyclus te automatiseren, zijn op hun beurt allemaal een deel van DevOps.



Figuur 21: Relatie tussen processen [9]

3.2 Broncontrole op de database

3.2.1 Devops naar de database brengen [10]

3.2.1.1 Beschrijving

De eerste bron is geschreven door Matt Hilbert. Matt is een schrijver in het topic technologie bij Red Gate. Hij heeft al 20 jaar werkervaring voor grote tech-bedrijven.

Deze bron bevat twee verschillende blog berichten die mekaar opvolgen. Matt beschrijft hoe DevOps naar de database gebracht kan worden. Het eerste artikel zijn focus ligt op versiecontrole en het tweede zijn ligt op *continuous delivery*.

3.2.1.2 Inhoud

Part 1: versiecontrole

Al enkele jaren winnen DevOps-technieken aan populariteit bij applicatieontwikkelaars. Dit is geen verrassing omdat deze technieken veel automatiseren waardoor ontwikkelaars zich meer kunnen

focussen op wat ze graag doen: coderen. Ook bedrijven zijn hier een grote fan van omdat DevOps zorgt voor een verhoging van productiviteit, winstgevendheid en marktaandeel. Dan zijn er ook nog de databaseontwikkelaars en -administrators. Zij moeten omgaan met live data waar niks mee mag foutlopen, want dan zijn er grote problemen. Hierdoor is er een bepaald geloof dat DevOps-technieken niet samen kunnen gaan met een database.

Maar eigenlijk is het relatief eenvoudig om over te stappen op een DevOps manier van werken met een database. Als eerste stap kan er niet meer apart gedacht worden, maar samen, op een DevOps manier. Het doel is om kleinere en frequentere releases uit te brengen waarbij ontwikkel- en operationteams samenwerken. Als tweede moet er nagedacht worden over welke tools er gebruikt gaan worden. Dit betekent niet dat er allemaal nieuwe tools gebruikt moeten worden, database DevOps is perfect implementeerbaar door gebruik te maken van dezelfde tools als de ontwikkelaars.

Termen zoals *continuous delivery, continuous integration* en *automated release management* klinken ingewikkelder dan dat ze in feite moeten zijn. Dit kan databaseadministrators afschrikken om over te stappen op een DevOps manier van werken, maar de sleutel is om klein te beginnen. Namelijk door de databasecode in *source control* te brengen, op deze manier kan er makkelijk nagegaan worden welke databasecode hoort bij welke applicatiecode.

Het is ook belangrijk om te beseffen dat het manueel uitvoeren van aangeleverde scripts, waarbij één of een selectieve groep personen kennis hebben van de database, niet hetzelfde is als source control. Bij een versiebeheer is het de bedoeling dat de geschiedenis van aanpassingen bijgehouden wordt. Hierdoor kan er makkelijk naar een bepaalde staat van de database gegaan worden.

Source control voor een database brengt veel voordelen met zich mee, zoals het synchroniseren van database structuren over alle omgevingen, vermindering van werk en het minimaliseren van errors. Het is wel belangrijk dat er best gebruik gemaakt wordt van de development tools zodat er geen nieuwe tools moeten aangeleerd worden. Dit zou alleen maar voor extra verdeling zorgen tussen de verschillende teams.

3.2.2 Zes redenen om de database onder versiecontrole te stoppen [11]

3.2.2.1 Beschrijving

De tweede bron is geschreven door Mary Robbins. Deze bron komt, net zoals de eerste bron, van de Redgate blog.

In dit blogbericht legt Mary zes verschillende voordelen uit die versiecontrole op databases met zich meebrengt.

3.2.2.2 Inhoud

Door het verhoogde tempo van database-ontwikkeling en de vele nieuwe regels omtrent data is het noodzakelijk om efficiëntere processen op te zetten rond de database. Het standaardiseren van processen binnen teams, projecten en rond de database en applicatie code zorgt voor veel voordelen. Zo zorgt het voor verhoogde kwaliteit en tijdsbesparingen. Om dit op te zetten gaat eerst de database voorzien moeten worden van een versiebeheer. Hieronder zijn zes verschillende voordelen van waarom dit aan te raden is:

1. **Vergemakkelijkt samenwerking tussen verschillende teams**
   Databasecode in een versiecontrolesysteem brengen maakt het makkelijker om veranderingen te delen en het werk van verschillende teamleden, die verantwoordelijk zijn voor de database, beter te coördineren. Het zorgt er ook voor dat ontwikkelaars over een

lokale database kunnen beschikken zodat ze alle vrijheid hebben om risicoloos nieuwe aanpassingen te testen.

2. **Meer inzicht krijgen in de development pipeline**
   Een versiecontrolesysteem zorgt voor een overzicht op het werk dat verricht wordt, de vooruitgang, wie een aanpassing deed en waarom.

3. **Mogelijkheid om naar vorige versies te *rollbacken***
   Ondanks dat er best een betrouwbare back-up strategie moet zijn, zorgt ook versiecontrole op de database voor een efficiënt mechanisme om back-ups van de SQL-code te hebben. Omdat de geschiedenis incrementeel is, is het makkelijk om een *rollback* uit te voeren.

4. **Makkelijker aantonen van compliance en auditing**
   Complaince auditors verwachten van een organisatie om alle database veranderingen te verantwoorden en kunnen voorleggen wie er toegang tot heeft. Met source control kan er altijd nagegaan worden wie, wanneer en waarom er aanpassingen gedaan zijn.

5. **Funderingen voor automatische database *deployments* leggen**
   Versie controle op de database zet de deur open voor automatische *deployments*. Het synchroniseren van applicatie- en database code haalt ook de bottlenecks uit het proces.

6. **Veranderingen in de database- en applicatiecode synchroniseren**
   Doordat de database en applicatie versies synchroon zijn is er een betere coördinatie tussen teams, werken ze efficiënter en het helpt ook bij het oplossen van problemen.

3.2.3 Reflectie

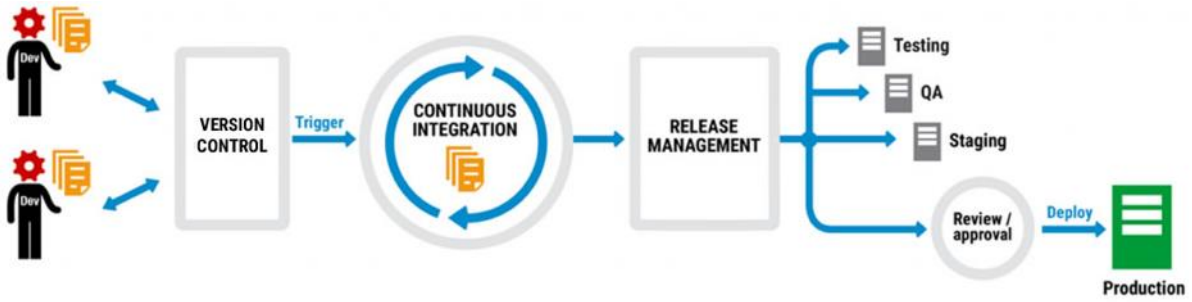
Er wordt hier een korte reflectie gegeven omdat het onderzoek initieel gestart was om enkel broncontrole voor databases te onderzoeken. Tijdens het maken van deze literatuurstudie werden er veel interessante artikels gevonden over het brengen van DevOps naar de database. Broncontrole maakt DevOps mogelijk, daarom lijkt het mij zeer interessant om ook te kijken wat de mogelijkheden zijn na de broncontrole toe te passen. Hierdoor werd er beslist om de scope van het onderzoek te verbreden zodat er ook gekeken wordt naar CI/CD.

3.3 Database DevOps

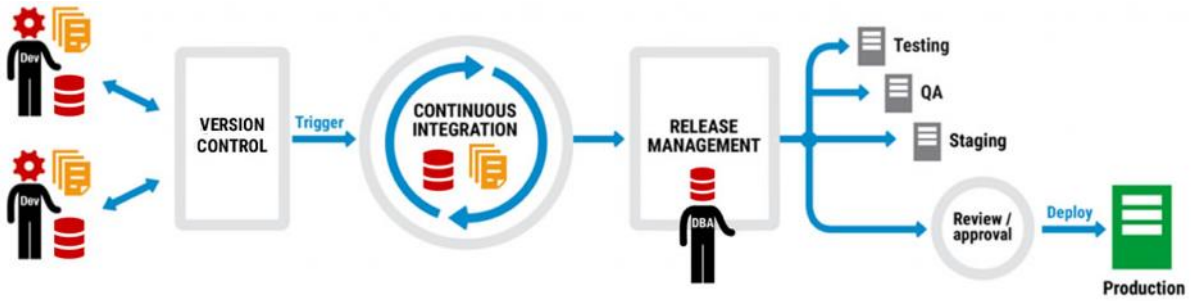
In dit hoofdstuk wordt er besproken wat de database in het DevOps proces opnemen betekent en de toepassing ervan. Er wordt gekeken naar de voor- en nadelen en eventuele moeilijkheden die eraan vasthangen. Daarnaast gaan de verschillende stappen besproken worden die nodig zijn om de database in het DevOps proces op te nemen. Tot slot worden nog enkele tools besproken die gebruikt kunnen worden tijdens de verschillende stappen.

3.3.1 Wat is database DevOps?

Hieronder zijn twee foto's die aantonen wat database DevOps inhoud. Op Figuur 22 staat een DevOps schema afgebeeld waarin enkel rekening wordt gehouden met de ontwikkelcyclus van de applicatie. Figuur 23 weergeeft het DevOps schema waar de database ontwikkeling geïntegreerd is.



Figuur 22: DevOps schema



Figuur 23: Devops database schema [12]

Deze schema's tonen aan dat database DevOps niet meer is dan het betrekken van de database bij reeds bestaande DevOps-processen is. Dit zal ervoor zorgen dat de database geen bottleneck wordt in een ontwikkelcyclus.

3.3.2 Stappen richting database DevOps

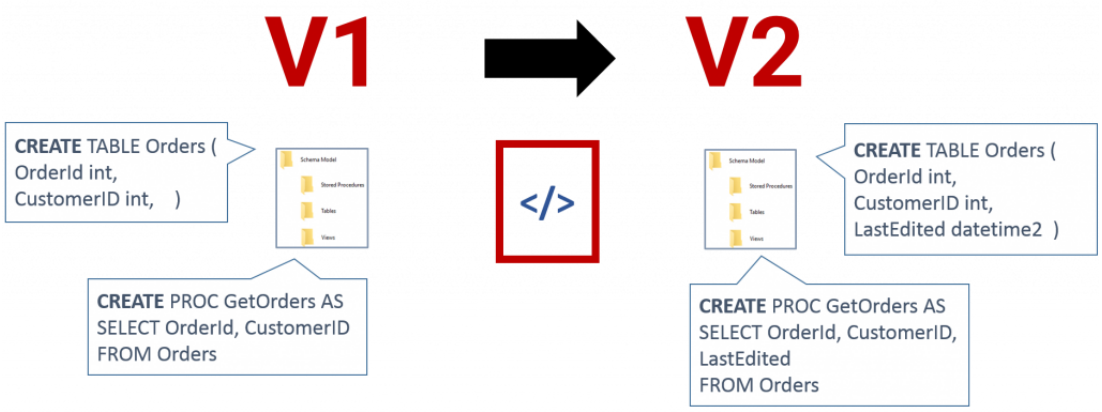
In dit hoofdstuk worden verschillende stappen aangehaald om richting een DevOps manier van werken over te stappen. Dit betekent niet dat elke stap vereist is om DevOps te werken, omdat niet elk bedrijf baat heeft bij een volledige geautomatiseerde pipeline. Maar sommige stappen vereisen dat een voorafgaand proces geïmplementeerd is.

3.3.2.1 Versie- of broncontrole op de database [13]

Versie- of broncontrole op wijzigingen aan de database zijn de eerste stap richting het introduceren van de database aan DevOps-processen. Als wijzigingen aan de database worden bijgehouden kan er op elk moment een werkende *build* gemaakt worden. Daarnaast zorgt versiecontrole ervoor dat er altijd een versie is om naar terug te gaan. Verder is dit de basis om een automatische *deployment pipeline* te integreren.

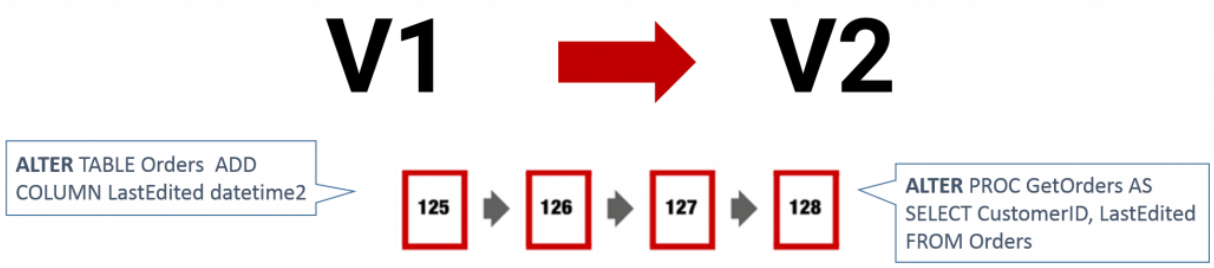
Er zijn twee methodes hoe broncontrole aangepakt kan worden op een database:

- State-based:
  Bij deze methode is het basisconcept dat er op een aparte database ontwikkeld wordt, deze vormt dan de primaire *source of truth.* Eens er beslist wordt dat de database zich in een staat bevindt die geschikt is voor de volgende release wordt er een schemavergelijking uitgevoerd tussen de primaire database en de productiedatabase. Deze schemavergelijking zal automatisch scripts aanmaken die de productiedatabase naar de nieuwe versie brengen.



Figuur 24: state-based broncontrole [14]

- Migration-based:
  Bij deze methode wordt er in plaats van een staat van de database, alle verschillende stappen om tot die staat te komen bijgehouden. Dit betekent dat elke wijziging in een script zit en elk script op een versiecontrolesysteem bijgehouden wordt. Op deze manier kunnen alle scripts uitgevoerd worden op een database.



Figuur 25: migration-based broncontrole [14]

3.3.2.2 Wijzigingen eerder controleren

Database scripts worden vaak gecontroleerd op het einde van de ontwikkelingscyclus, net voor ze uitgevoerd gaan worden. Dit proces moet meer naar links verplaatsen tijdens de ontwikkeling zodat wijzigingen direct worden herzien wanneer ze toegepast worden. [15]

3.3.2.3 Continuous Integration

Eens dat versiecontrole wordt toegepast op databasewijzigingen, kan CI geïntegreerd worden in de ontwikkelcyclus. Wanneer veranderingen in het versiecontrolesysteem worden gezet, gaat CI een automatische *build* triggeren. Hierbij worden de veranderingen geverifieerd. Als de *build* faalt kunnen problemen opgelost en opnieuw getest worden. [15]

3.3.2.4 Release management

Ondanks dat CI-omgevingen vaak de productieomgevingen weerspiegelen bij applicaties, is dit zelden het geval bij databases. Daarom moet er gebruik gemaakt worden van een *stagingomgeving* met een database die bijna een perfecte replica is van de productiedatabase. Hier worden alle wijzingen op uitgevoerd voordat ze naar de productieomgevingen worden gezet. Op deze *stagingomgeving* kunnen databasebeheerders controleren of de wijzigingen klaar zijn om in de productieomgeving te implementeren. [16]

3.3.2.5 Continuous Delivery

Continuous Delivery bouwt verder op *release management.* Release management kan manueel gedaan worden waarbij een databasebeheerder een script met wijzigingen controleert en hierna de release door een druk op een knop uitvoert op de database. Er kan ook gekozen worden om nog meer DevOps te gaan werken door *release management* te automatiseren. Er zijn verschillende tools die het manuele proces kunnen automatiseren. Als derde optie kan er ook gekozen worden voor semi-automatische releases. Hier worden de scripts automatisch uitgevoerd op de *stagingdatabase(s)* en moet een databasebeheerder deze controleren. Hierna voert hij de wijzigen uit op de productieomgeving. [15]

3.3.3 Cons/uitdagingen

DevOps implementeren doorheen een ontwikkelingscyclus heeft veel voordelen. Toch is het niet altijd gemakkelijk om DevOps te implementeren. Ook kan het resultaat van DevOps ervoor zorgen dat *releases* te snel gaan. In dit hoofdstuk worden deze pijnpunten toegelicht.

3.3.3.1 Werken met dicht gekoppelde architecturen

Wanneer er database DevOps geïmplementeerd gaat worden kan het een uitdaging zijn wanneer er gebruik gemaakt wordt van dicht gekoppelde architecturen met een centrale database. Dit kan opgelost worden door het gebruik van microservices. Dit zijn onafhankelijk inzetbare services die zijn afgestemd op een bedrijfsfunctie, waarbij elke microservice een eigen database heeft. Deze services communiceren met elkaar via duidelijk gedefinieerde interfaces in plaats van rechtstreeks met de database om te gaan. [16]

3.3.3.2 Databasebeheerders willen niet meewerken

Veel databasebeheerders staan twijfelachtig tegenover DevOps implementeren op de database ontwikkelcyclus. De automatisatie die DevOps met zich meebrengt, wordt vaak als niet betrouwbaar gezien of beheerders denken dat automatisatie van processen hun werk zal afnemen. In plaats van uren manueel wijzigingen door te voeren op de database kan een beheerder zijn tijd steken in *requirements* die handig zijn voor een bedrijf. DevOps zal het nut van een databasebeheerder niet wegnemen maar ervoor zorgen dat een beheerder de tijd heeft om aan hogere snelheden, veiliger de database te onderhouden. [17]

3.3.3.3 Te veel automatisatie – te snelle levering van een product

Voor sommige bedrijven is het aantrekkelijk om alles te automatiseren, van het inbrengen van wijzigen tot het *deployen.* Aan de andere kant zijn er ook bedrijven zoals banken. Continu nieuwe features en functionaliteiten aanbrengen op een applicaties zou voor verwarring kunnen zorgen bij klanten. Dit is niet interessant in het kader van hun bedrijfsstrategie omdat dit ervoor zorgt dat klanten overgaan naar concurrenten.

Daarom is het belangrijk om te beslissen welke DevOps-processen goed zijn binnen een specifiek bedrijf. Het brengen van DevOps betekent niet dat alles geautomatiseerd moet zijn, het staat eerder voor het brengen van zoveel mogelijk automatisatie en het verenigen van ontwikkelaars en operatieteams zodat er sneller en efficiënter gewerkt kan worden. Als een bedrijf geen baat heeft bij het implementeren van een CD-pipeline maar wel in het implementeren van een CI-pipeline, is dit een persoonlijke keuze wat toegepast kan worden.

3.3.3.4 Aanpassing van cultuur

Om DevOps succesvol te implementeren is er meer nodig dan enkel de correcte tools aan te kopen. DevOps is een cultuur, een manier van werken. Om over te schakelen van een cultuur naar de andere is er tijd nodig en moeten werknemers bereid zijn om hiervoor open te staan. Het belang van snel releases uit te brengen van hoge kwaliteit zet bedrijven onder druk om naar DevOps over te schakelen. [18]

3.3.4 Pro's [19]

1. **Snellere time-to-market:**
   In plaats van een heel groot deel database veranderingen in een keer te moeten implementeren, kan er door DevOps gewerkt worden met kortere ontwikkelingsfases. Daardoor worden veranderingen in kleinere, meer beheerbare stukken gekapt.
2. **Minder errors:**
   Door DevOps worden fouten in scripts veel eerder ontdekt, dit zorgt ervoor dat er snel aanpassingen kunnen gedaan worden aan deze scripts.
3. **Stabielere *builds* en snellere *deployments* en releases:**
   CI en CD proberen zoveel mogelijk processen te automatiseren, ook op de database. Door bijvoorbeeld het invoegen van automatische testen worden fouten veel eerder ontdekt, waardoor er stabielere *builds* gemaakt worden die minder kans hebben op falen in latere fases van het DevOps proces.
4. **Verbeterde release management:**
   DevOps naar de database brengen zorgt ervoor dat de testomgevingen beter de productieomgeving nabootsen. Dit heeft als gevolg dat er betere testen gemaakt kunnen worden om mogelijke errors op te sporen.

3.3.5 Tools

In dit hoofdstuk worden verschillende type tools besproken zoals een versiebeheersysteem, buildtool etc. voor elke type worden een aantal mogelijke tools opgesomd.

3.3.5.1 Versiebeheersysteem

In elk softwareontwikkelingsbedrijf wordt er al gebruik gemaakt van een versiebeheersysteem, het is hier dan ook de beste keuze om verder te gaan met wat er reeds gebruikt wordt. Indien er van niets begonnen wordt, is er in Tabel 4 een kleine vergelijking tussen drie verschillende, bekende versiebeheersystemen te zien.

	Pro's	Cons
Apache Subversion (SVN)	• Goede GUI-tools zoals TortoiseSVN • Ondersteunt lege folders • Betere ondersteuning voor Windows in vergelijk met Git • Makkelijk op te zetten en te beheren • Integreert goed met Windows, toonaangevende IDE en Agile-tools	• Bewaart tijdstip van aanpassing van bestanden niet • Gaat niet goed om met normalisatie van bestandsnamen • Ondersteunt geen ondertekende herzieningen

Git	• Supersnelle en efficiënte prestaties • Platformonafhankelijk • Wijzigingen in de code zijn makkelijk na te gaan • Makkelijk in onderhoud en robuust • Goed CLI-programma, git bash • Kan ook gebruikt worden met een GUI	• Ingewikkelde en grote log geschiedenis soms moeilijk te begrijpen • Geen ondersteuning voor uitbreiding van zoekwoorden en behoud van tijdstempels
Team Foundation Server (TFS)	• Eenvoudig beheer. Vertrouwe interfaces en nauwe integratie met andere Microsoftproducten • Mogelijkheid om continue integratie te implementeren • Goede ondersteuning voor *branching* en *merging* • Aangepast incheckbeleid	• Veel mergeconflicten • Connectie met de centrale repository is altijd vereist • Vrij traag bij het uitvoeren van pull, check-in- en branchoperaties

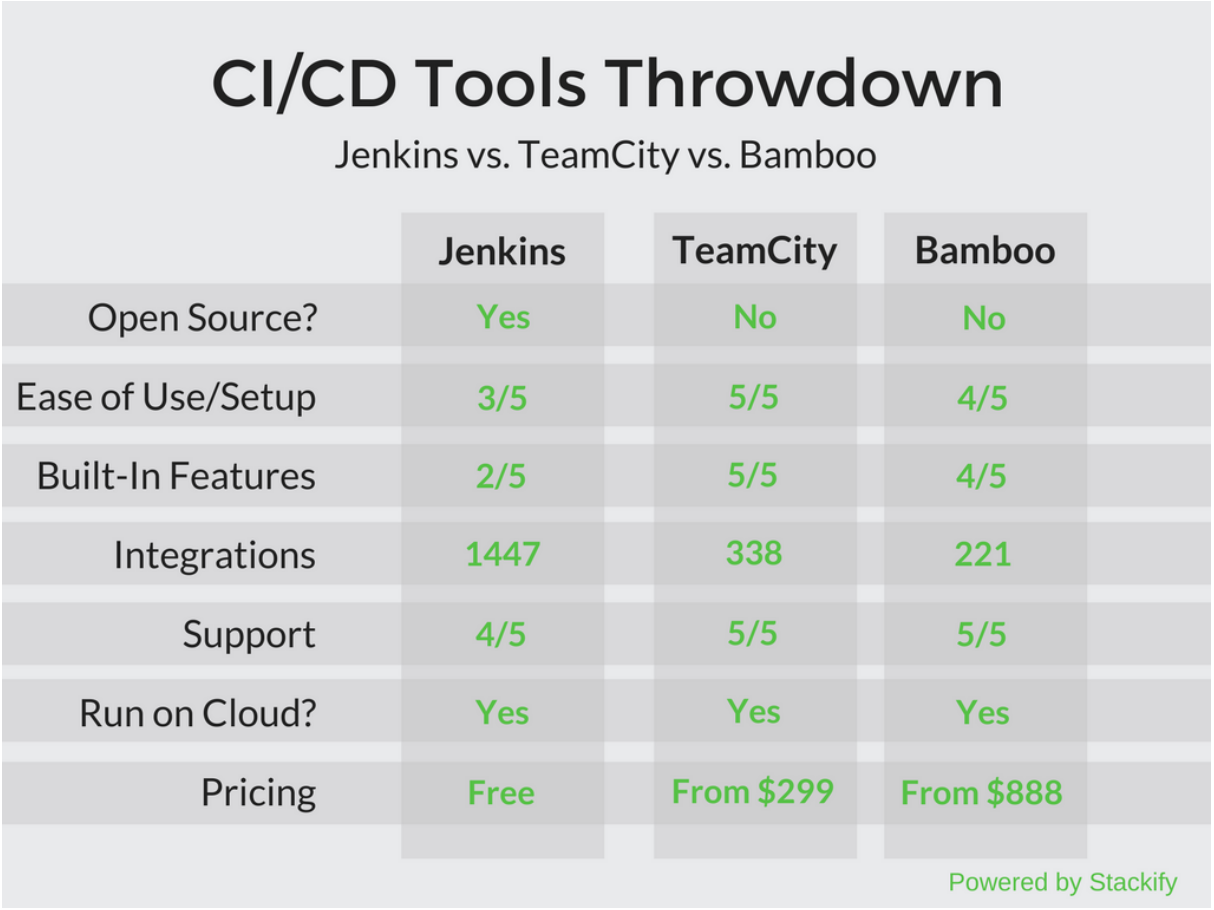
Tabel 4: Vergelijking versiebeheersystemen [20]

3.3.5.2 Database broncontrole tool

Een zeer belangrijk onderdeel om de database te verwerken in DevOps-processen is de database scripts in broncontrole krijgen. Dit wordt gedaan via tools die hier specifiek voor ontworpen zijn. In hoofdstuk 3.2 van deel 1 in dit eindwerk is er al een kleine vergelijking opgesteld voor een aantal tools, twee commerciële tools en twee opensourcetools. Daarom wordt er naar dit hoofdstuk verwezen om meer informatie te krijgen over database broncontrole tools.

3.3.5.3 Buildtool [21]

In dit hoofdstuk gaan er drie CI/CD-tools vergeleken worden. De tools zijn Jenkins, Bamboo en TeamCity. Dit zijn geschikte tools voor CI, geautomatiseerde *builds*, automatisch testen en CD. Er gaat vooral gekeken worden naar de belangrijkste attributen die een CI/CD-tool moet bevatten. Op Figuur 26 is de vergelijkingstabel te zien.



Figuur 26: CI/CD tools

Naast deze drie zijn er nog verschillende *buildtools* waaruit gekozen kan worden. Dit is enkel een vergelijking tussen drie grote tools om een idee te geven van de mogelijkheden.

3.3.5.4 Release management tool

Meestal kunnen de tools die gebruikt worden voor *builds* te maken, ook gebruikt worden voor het release management. Toch zijn er tools op de markt die zich specifiek toeleggen op release management en het *deployen* van *buildpakketten.* Een voorbeeld hiervan is Octopus Deploy.

3.4 Toepassing

In dit hoofdstuk gaan er twee toepassingen stap voor stap uitgelegd worden. De toepassingen zullen aangeven hoe de database onder broncontrole gebracht kan worden en hoe er makkelijk een CI/CD-pipeline kan gemaakt worden eens dit op zijn plaats staat. Bij de eerste toepassing wordt er gebruik gemaakt van software van RedGate: de Deployment Suite for Oracle. [22] De tweede toepassing wordt gemaakt aan de hand van Liquibase. Zo kan er een vergelijking gemaakt worden tussen een opensourcetool en een commerciële tool.

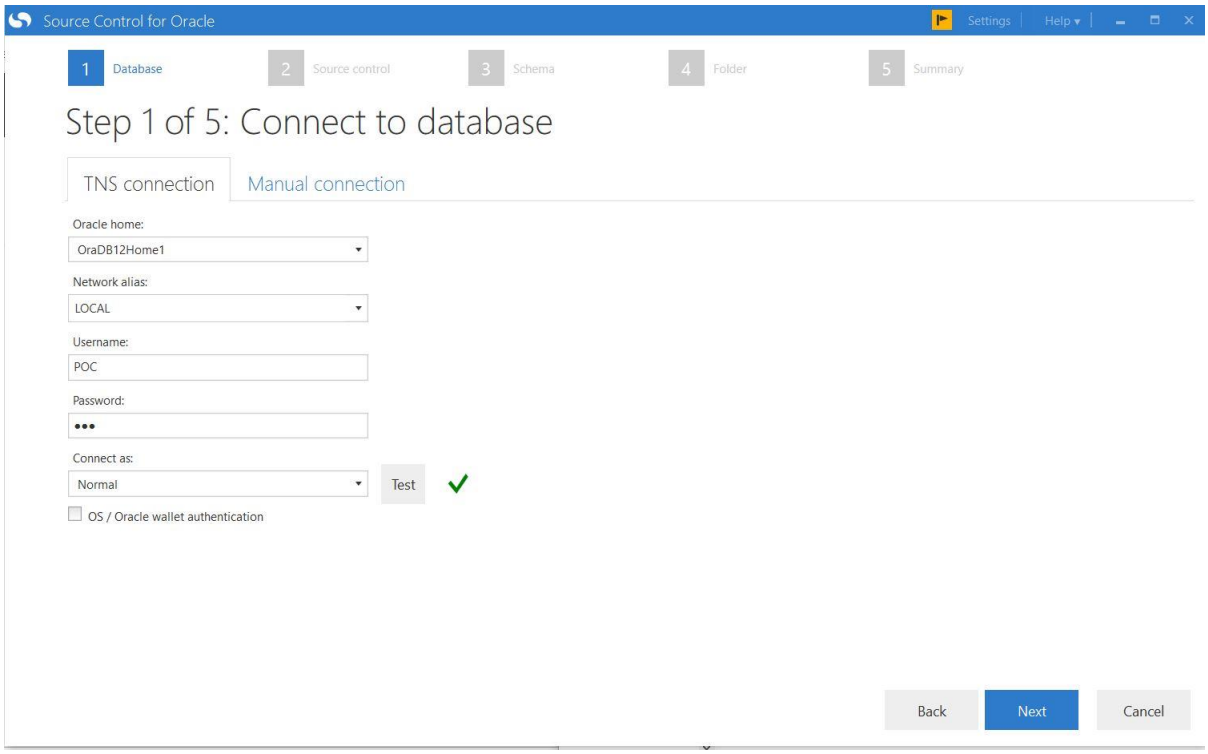
Deze tools zijn ook gekozen vanwege hun verschillen in de aanpak van broncontrole. De Deployment Suite gaat via het *state-based* principe werken door scripts bij te houden van elk object. Liquibase daarentegen werkt via het *migration-based* principe waarbij alle scripts worden bijgehouden. Deze principes zijn eerder aangehaald in hoofdstuk 3.2.2.1.

3.4.1 Redgate

3.4.1.1 Broncontole op de database

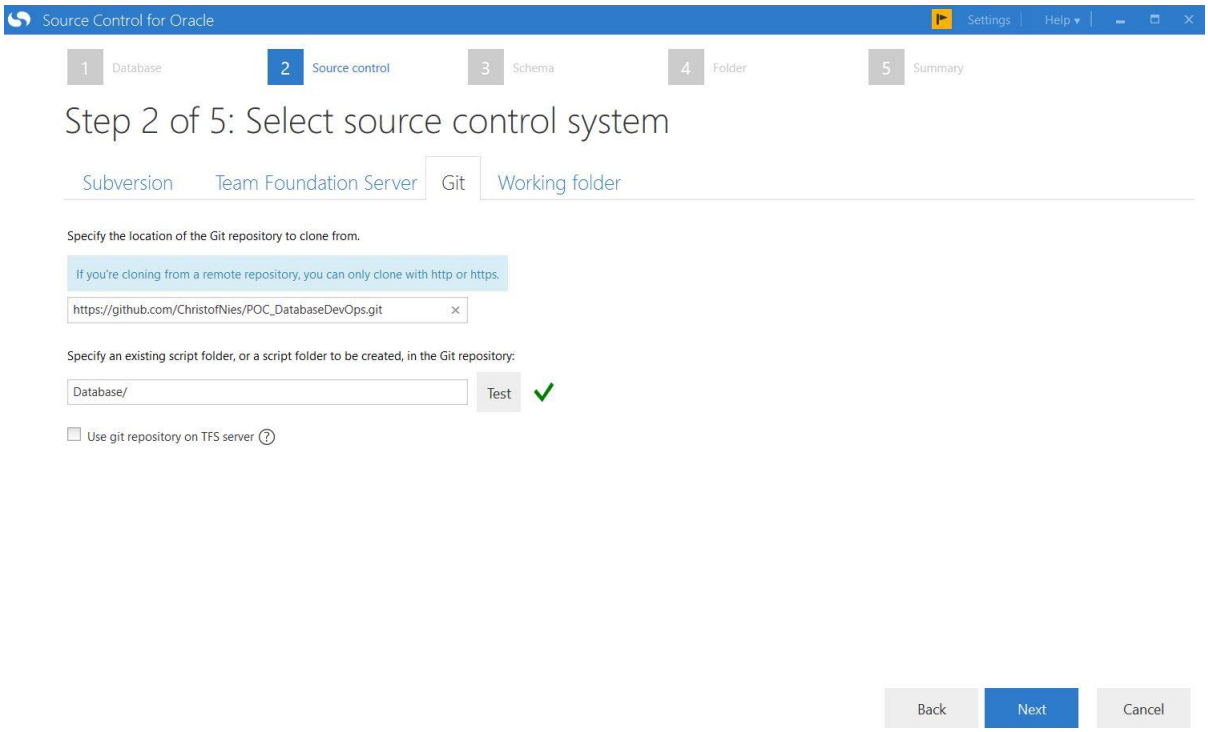
De eerste stap is ervoor zorgen dat de database scripts worden bijgehouden op een versiebeheersysteem. Hiervoor is er een Git *repository* aangemaakt op Github. [23] Daarnaast wordt er ook gebruik gemaakt van Source Control for Oracle van Redgate. Deze tool detecteer aanpassingen aan de lokale database, nadien kan een ontwikkelaar deze wijzigingen *committen* naar de *repository*.

Het verbinden met Source Control van Red Gate is redelijk makkelijk. Als eerste wordt er vastgelegd met welke database er verbonden moet worden, dit is te zien op Figuur 27.



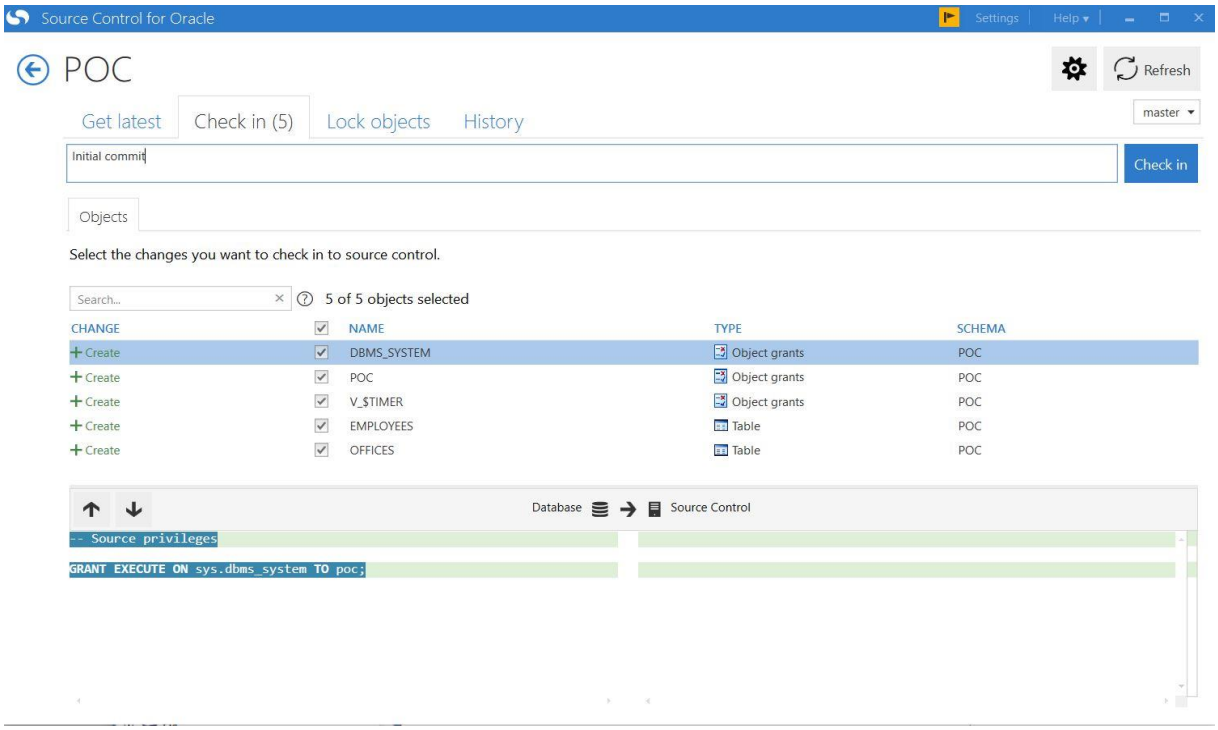
Figuur 27: Redgate connect to database

Daarna wordt de Github *repository* gekoppeld aan de applicatie zoals te zien is op Figuur 28. De rest van de stappen wijzen hun eigen uit en gaan dus niet verder overlopen worden.



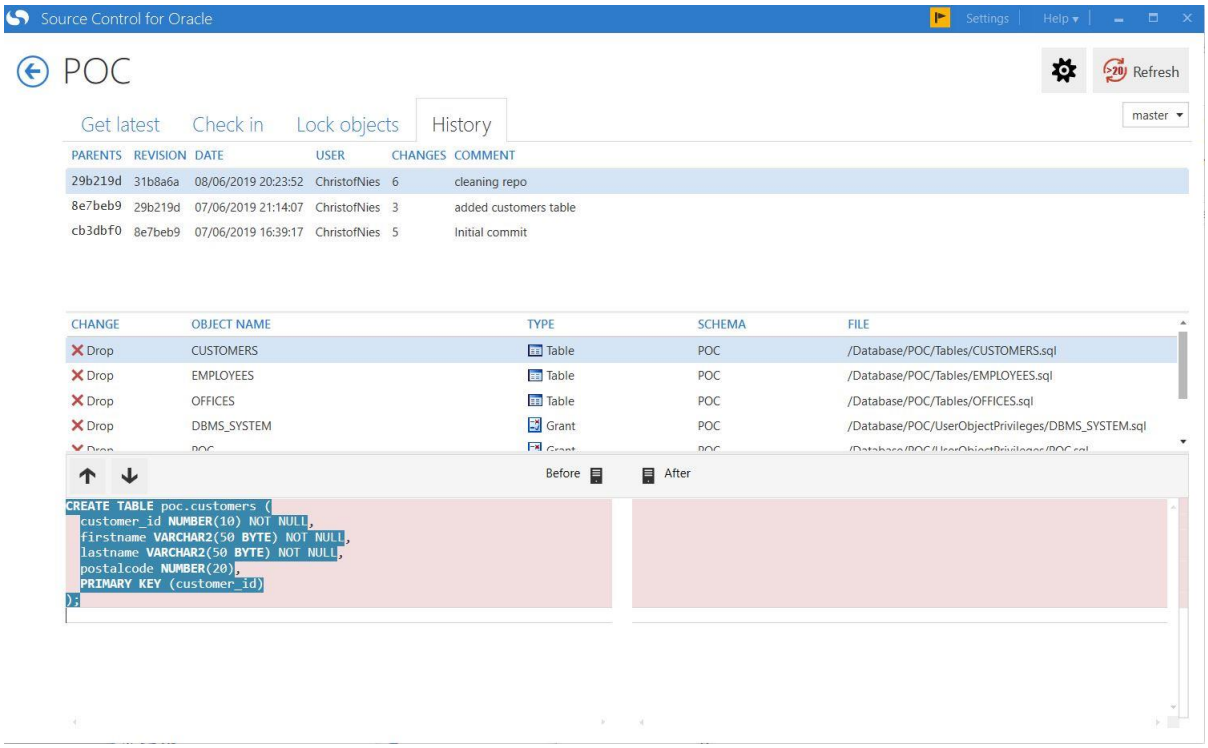
Figuur 28: Redgate Select source control system

Zoals te zien is op Figuur 29 kunnen veranderingen makkelijk ingecheckt worden op Git. Er kunnen via scripts aanpassingen doorgevoerd worden op de lokale database, maar er kan ook gewerkt worden met een IDE zoals SQL Developer. Dit omdat Source Control werkt via *state-based* migraties. Hierbij weergeeft de lokale database één staat van de database en de scriptfolder een andere staat. Deze worden dan met elkaar vergeleken en hieruit kan Source Control zelf detecteren wanneer er aanpassingen zijn gedaan.



Figuur 29: Redgate check in

In het “History”-tabblad kan er gekeken worden naar de commitgeschiedenis, hiervan een voorbeeld op Figuur 30. Hier kan er makkelijk gezien worden wie, wat en wanneer er iets is aangepast.



Figuur 30: Redgate change history

Dit zijn in het algemeen alle stappen die nodig zijn om de database in broncontrole te krijgen via de tool Source Control inclusief een kleine toelichting over belangrijke schermen binnen de applicatie.

3.4.1.2 Buildpakket maken

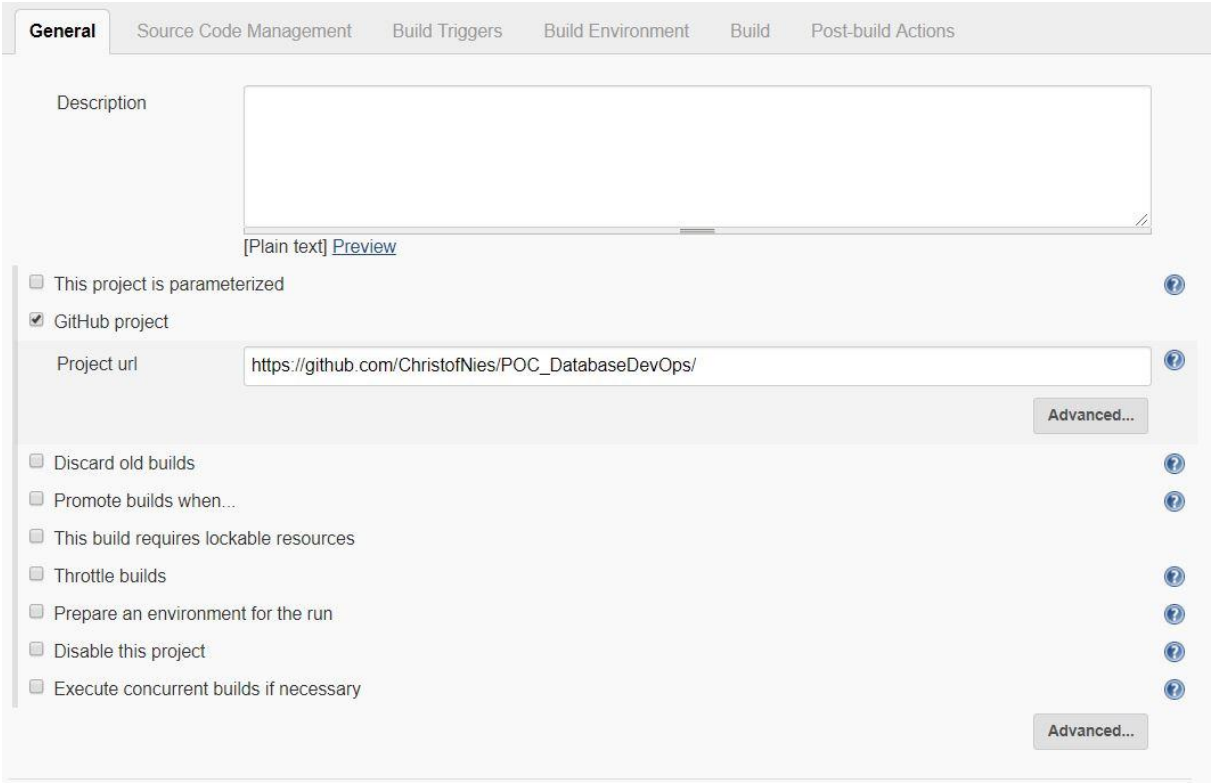
Elke keer als er code wordt ingecheckt op de *repository* moet er een nieuw pakket aangemaakt worden. Voor het buildpakket in te pakken wordt er gebruik gemaakt van NuGet, omdat dit eenvoudig in gebruiken is. Hiervoor moet er een nuspec-bestand toegevoegd worden bij de broncode. Een voorbeeld van een basis nuspec-file is te zien op Figuur 31.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <package xmlns="http://schemas.microsoft.com/packaging/2010/07/nuspec.xsd">
3    <metadata>
4      <id>MySchema</id>
5      <version>0.1</version>
6      <authors>Christof Nies</authors>
7      <owners>Christof Nies</owners>
8      <requireLicenseAcceptance>false</requireLicenseAcceptance>
9      <licenseUrl>https://creativecommons.org/licenses/by/4.0/</licenseUrl>
10     <projectUrl>http://www.dlmconsultants.com/</projectUrl>
11     <description>Include everything in this directory</description>
12     <language>en-US</language>
13   </metadata>
14 </package>
```

Figuur 31: Nuspec-bestand

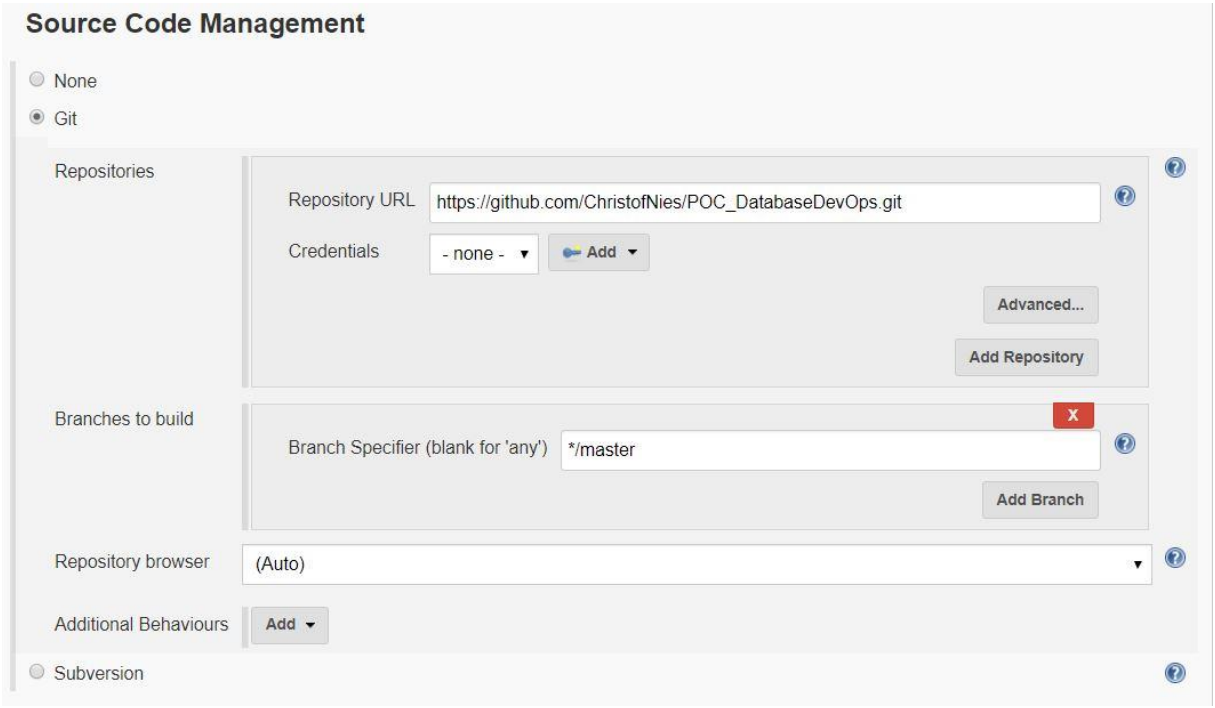
Om een pakket automatisch aan te maken, moet er een nieuw Jenkins project aangemaakt worden. Aangezien we willen dat het project elke keer uitgevoerd wordt wanneer er aanpassingen zijn

gedaan aan de broncode, moet het project aangeduid worden als een GitHub project. Deze configuratie is te zien op Figuur 32.



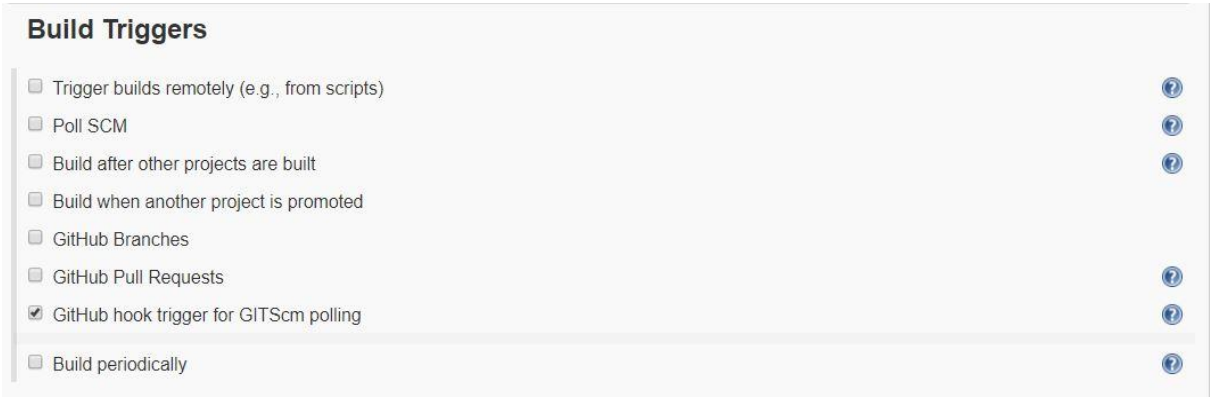
Figuur 32: Redgate algemene instellingen

Zoals te zien op Figuur 33, dient er bij "source code management" Git aangeduid te worden. Hier wordt de URL van de *repository* meegegeven en kan er bepaald worden welke *branch* er gebruikt moet worden. Voor deze POC is het voldoende om met één *masterbranch* te werken.



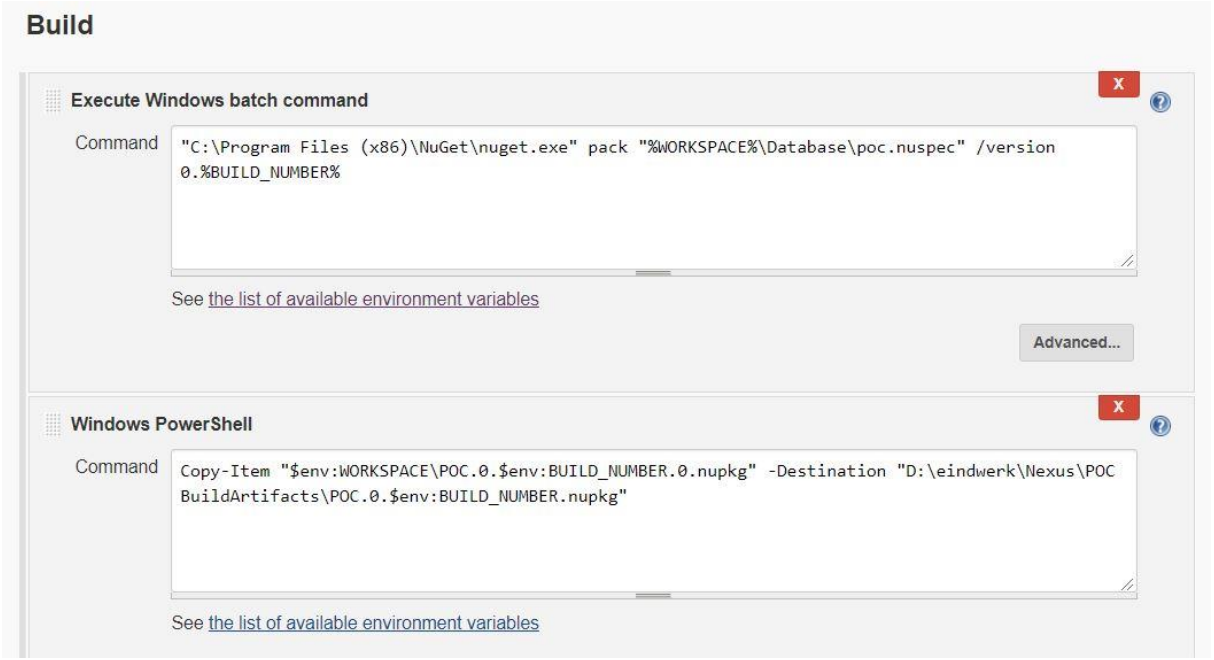
Figuur 33: Redgate buildpakket scm

Zoals eerder vermeld moet dit project elke keer uitgevoerd worden wanneer er nieuwe code wordt ingecheckt via Git. Daarom moet er bij “build triggers”, “Github hook trigger for GITScm polling” aangeduid worden. Deze gaat de requests van GitHub opvangen. De configuratie is te zien op Figuur 34.



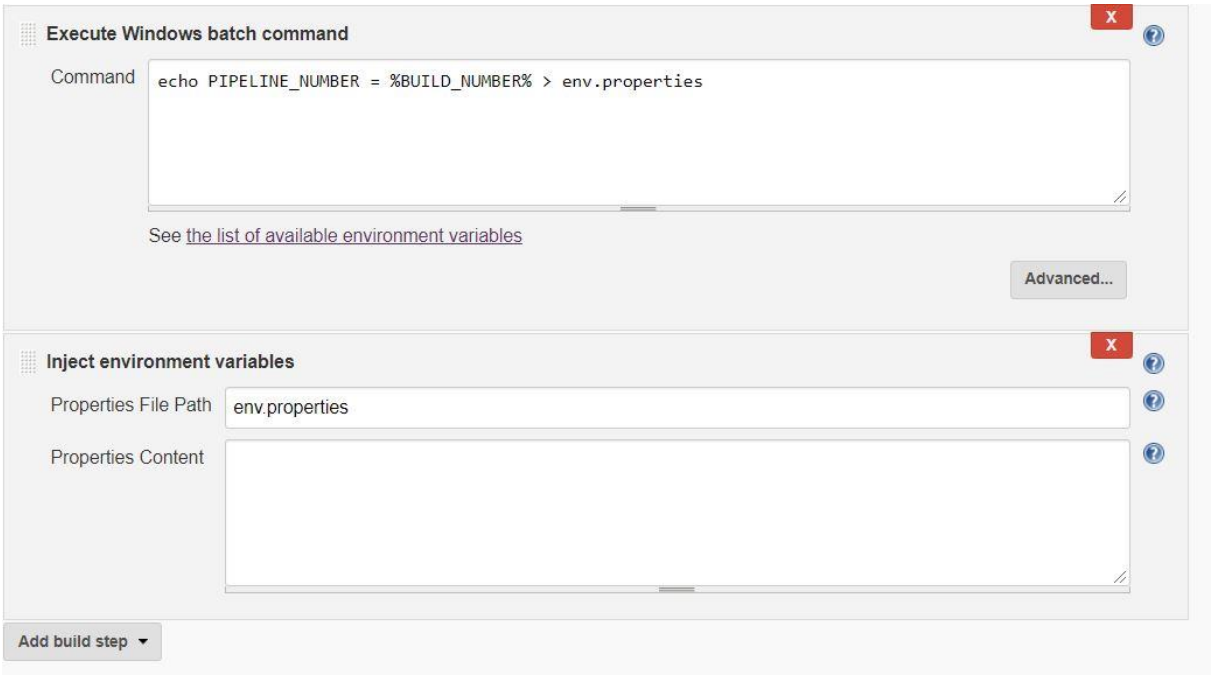
Figuur 34: Redgate buildpakket triggers

Hierna gaan we bepalen wat er precies moet gebeuren bij het uitvoeren van dit project, deze stappen zijn te zien op Figuur 35 en Figuur 36. Er worden vier “build steps” toegevoegd. Eerst wordt er een Windows batch commando uitgevoerd die de broncode omvormt tot een NuGet-pakket. Hierna wordt dit pakket gekopieerd naar een andere locatie. In het voorbeeld zal deze locatie een plek zijn op de lokale server, maar in praktijk zal dit bijgehouden worden op een *artifactrepository*, de Nexus.



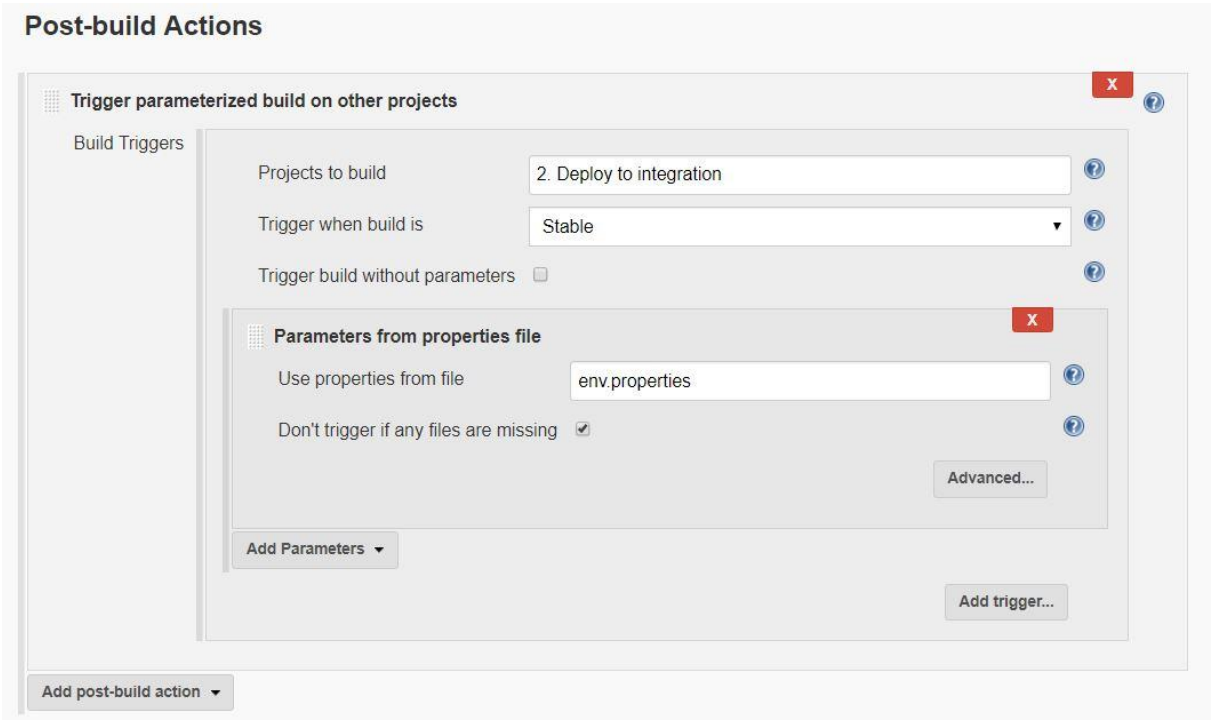
Figuur 35: Redgate buildpakket buildsteps 1

Om ervoor te zorgen dat het volgende Jenkins project het NuGet pakket kan downloaden met het juiste *buildnummer* moeten er nog twee *buildsteps* aangemaakt worden in Jenkins, deze zijn te zien op Figuur 36. Eerst moet er een Windows batch commando uitgevoerd worden die het nummer van de pipeline gelijkzet aan het *buildnummer* en deze daarna in een bestand opslaat. Dit bestand wordt via de InjEnv plugin geconfigureerd.



Figuur 36: Redgate buildpakket buildsteps 2

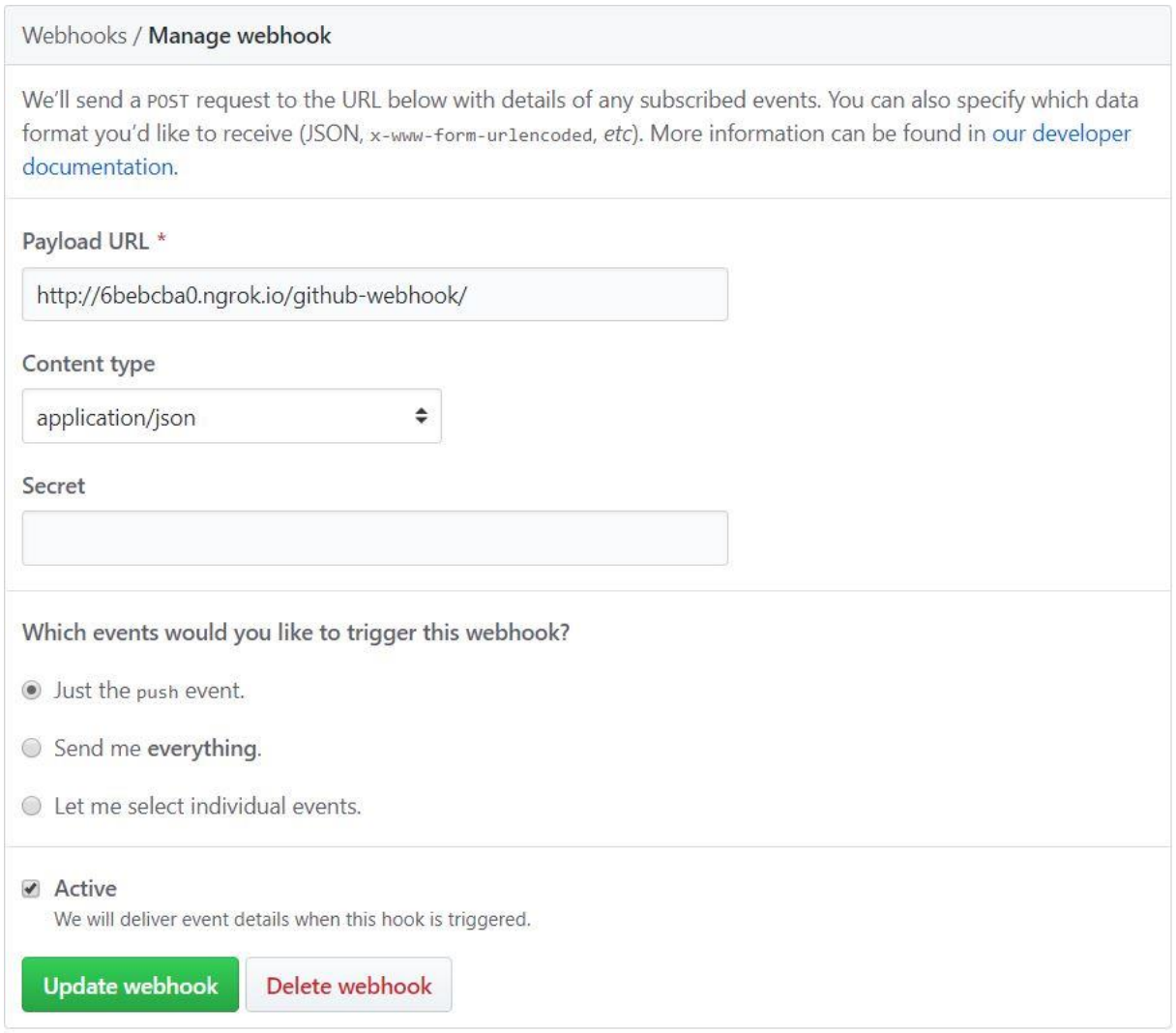
Als laatste moet er nog een post-build actie toegevoegd worden. Dit zien we op Figuur 37, deze actie zal ervoor zorgen dat het volgende Jenkins project uitgevoerd wordt als dit project zonder fouten uitgevoerd wordt.



Figuur 37: Redgate buildpakket post-build acties

Om ervoor te zorgen dat de pipeline in Jenkins iedere keer wordt getriggerd wanneer er aanpassingen gedaan worden in het versiebeheersysteem moet er een webhook ingesteld worden op GitHub. Op Figuur 38 zijn de configuraties hiervoor te zien. De Payload-URL is de URL waar een

post-request naar verstuurd moet worden. Er wordt ook gekozen voor enkel het *pushevent* door te sturen omdat er geen extra data vereist is.



Figuur 38: Github webhook

3.4.1.3 Deploy naar de integratieomgeving

Zodra het *buildpakket* succesvol is aangemaakt, moet dit pakket automatisch worden *gedeployed* naar de integratieomgeving. In deze omgeving worden de unit tests uitgevoerd en worden de aanpassingen getest op een database om te controleren of deze correct opgebouwd wordt. Voor deze stap is er een nieuw Jenkins project aangemaakt, "2. Deploy to integration".

In dit project moeten er twee *"build steps"* toegevoegd worden, beide stappen moeten uitgevoerd worden via Windows Powershell. Het eerste commando gaat het NuGet pakket van de "Nexus" afhalen en deze uitpakken in de werkomgeving van Jenkins, dit wordt gedaan op lijn 17 in Figuur 39.

```
1  $pipeline = ${env:PIPELINE_NUMBER}.trim()
2  $nuget = "C:\RelativePath\JenkinsBuildArtifacts\MySchema.0.${pipeline}.nupkg"
3  $destination = "$env:WORKSPACE\extractedNuGet"
4
5  Write-Output "Using the following variables:"
6  Write-Output "pipeline: $pipeline"
7  Write-Output "nuget: $nuget"
8  Write-Output "destination: $destination"
9
10 # Clean the destination directory.
11 If (Test-Path $destination) {
12     rmdir $destination -Recurse -Force
13 }
14
15 # Extract the NuGet package
16 [Reflection.Assembly]::LoadWithPartialName('System.IO.Compression.FileSystem')
17 [IO.Compression.ZipFile]::ExtractToDirectory($nuget, $destination)
```

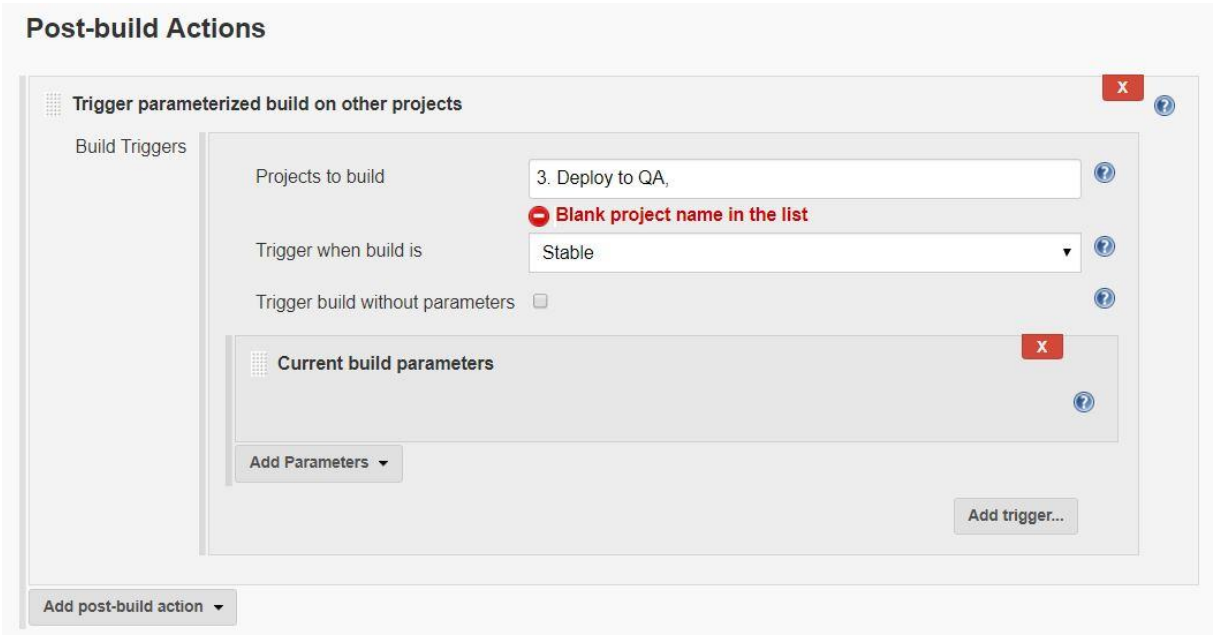
Figuur 39: Redgate pak NuGet pakket uit

De volgende stap voert het script in Figuur 40 uit, alleen moeten de variabelen nog ingevuld worden met persoonlijke configuraties. Eerst worden er verschillende variabelen gedeclareerd zodat deze hergebruikt kunnen worden. Daarna worden deze variabelen afgeprint zodat als er iets fout loopt, de variabelen gecontroleerd kunnen worden in Jenkins. Van lijn 29 tot 31 staat de code die ervoor gaat zorgen dat de targetdatabase geüpdatet gaat worden, evenals een bijhorend scripts en report hiervan genereren

```
1  # Configuration
2  $pipeline = ${env:PIPELINE_NUMBER}.trim()
3  $schemaComparePath = 'C:\\Program Files\\Red Gate\\Schema Compare for Oracle 3\\SCO.exe'
4  $targetSchemaName = "MySchema_INTEGRATION"
5  $schemaRelativePath = ""
6  $tns = "MyTNS"
7  $username = "MyUsername"
8  $password = "YourReallyWantToPerameteriseAndEncryptThis-ButForThePoCIllLetYouOff"
9  $sourceName = "MySchema"
10 $extractedNuGetDirectory = "${env:Workspace}\ExtractedNuGet\${schemaRelativePath}{$sourceName}"
11 $deploymentArtifactDirectory = "${env:Workspace}\artifacts\${pipeline}"
12
13 # Logging config variables for troubleshooting
14 Write-Output "Using the following variables:"
15 Write-Output "pipeline: $pipeline"
16 Write-Output "schemaComparePath: $schemaComparePath"
17 Write-Output "schemaRelativePath: $schemaRelativePath"
18 Write-Output "targetSchemaName: $targetSchemaName"
19 Write-Output "tns: $tns"
20 Write-Output "username: $username"
21 Write-Output "password: $password"
22 Write-Output "sourceName: $sourceName"
23 Write-Output "extractedNuGetDirectory: $extractedNuGetDirectory"
24 Write-Output "deploymentArtifactDirectory: $deploymentArtifactDirectory "
25
26 # Deploy changes using Redgate Schema Compare for Oracle
27 Write-Warning "Upgrading $targetSchemaName"
28
29 & $schemaComparePath -source $extractedNuGetDirectory -target "$username/$password@$tns{$targetSchemaName}"
30  -deploy -scriptfile "${deploymentArtifactDirectory}\upgradeScript.sql" -report
31  "${deploymentArtifactDirectory}\diffReport.html" -reporttype Simple -includeidentical | Out-Host
32
33 # Logging the Schema Compare exit code and path to artifacts
34 Write-Output "Schema Compare for Oracle exited with code $lastExitCode"
35 Write-Output "UpgradeScript and diff report created and saved to ${deploymentArtifactDirectory}"
36
37 # Exit code 61 is simply telling us there are differences that have been deployed.
38 if( $lastExitCode -eq 61)
39 {
40     exit 0
41 }
```

Figuur 40: Redgate deploy pakket script

Het laatste ding dat geconfigureerd moet worden in dit Jenkins project is de post-build actie. Dit gaat het CD-gedeelte van de pipeline in gang zetten. Alles tot en met deze stap behoorde bij CI. Op Figuur is de post-build actie te zien, deze gaat het volgende project triggeren met als voorwaarde dat dit project succesvol gebouwd is.



Figuur 41: Redgate CI post-build actie

3.4.1.4 Buildpakket deployen naar andere omgevingen

Voor dit te configureren moeten dezelfde "*build steps*" toegevoegd worden als in hoofdstuk 3.3.1.3, alleen moeten de variabelen aangepast worden aan de omgevingen. Daarna dient er ook een post-build actie toegevoegd te worden die op zijn beurt het volgende proces in gang zet. Enkel bij de productieomgeving dient dit niet toegevoegd te worden aangezien dit de laatste omgeving is waar het *buildpakket* op geïmplementeerd wordt.

3.4.2 Liquibase

In dit hoofdstuk worden er specifieke termen gebruikt van Liquibase, meer uitleg over deze terminologie is te vinden in deel 1 hoofdstuk 3.3.

3.4.2.1 Broncontrole op de database

Om te beginnen moet de database weer onder broncontrole gebracht worden. Dit doen we door een map op de Git-repository aan te maken. In deze map plaatsen we een *changelog*. Telkens wanneer er een aanpassingen wordt gedaan op de database dient hier manueel een script van geschreven te worden. Dit script wordt in de *changelog* geplaatst zodat Liquibase deze kan gebruiken. Aan elk script moet er ook metadata toegevoegd worden: de auteur en een id. Een voorbeeld van zo een *changelog* is te zien op Figuur 42.

```
1  --liquibase formatted sql
2
3  --changeset christofn:001_alter_employees
4  ALTER TABLE Employees RENAME COLUMN EmployeeNr to EmployeeId;
5
6  --changeset christofn:002_alter_employees
7  ALTER TABLE Employees
8  ADD job_title VARCHAR2(100);
```

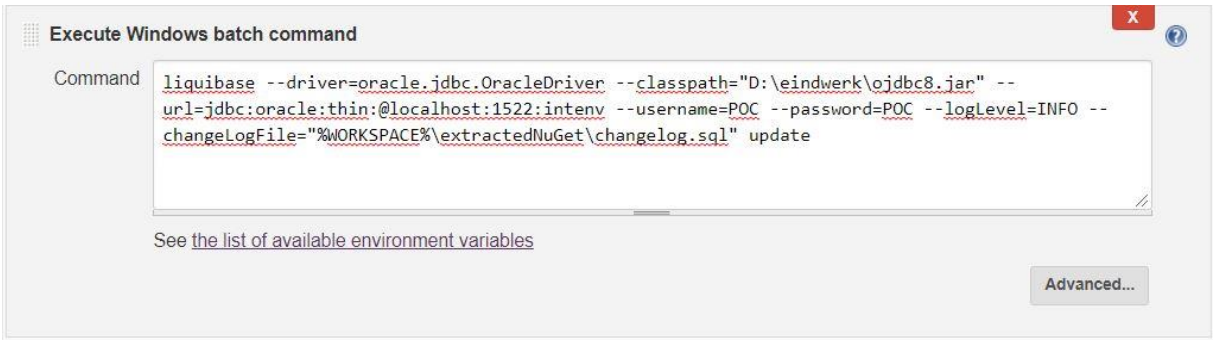
Figuur 42: Liquibase broncontrole changelog

3.4.2.2 Buildpakket maken

Hier zijn de stappen hetzelfde als in hoofdstuk 3.3.1.2. Alle configuratie zijn hetzelfde. Er moet ingesteld worden dat het een Github project is en wacht tot GitHub een push event doorstuurt via een webhook. Er wordt ook eerst een nuspec-bestand toegevoegd aan dezelfde map als de *changelog*. Daarna gaat er via een Windows Powershell "*build step*" het NuGet-pakket gemaakt worden waarna het op een Nexus geplaatst wordt. Als laatste wordt er een post-build actie geconfigureerd die er voor zorgt dat het volgende project wordt opgeroepen indien het *buildpakket* succesvol is aangemaakt.

3.4.2.3 Deploy naar de integratieomgevingen

Opnieuw is dit een nieuw project waar twee "*build steps*" in voorkomen. De eerste stap is hetzelfde als in hoofdstuk 3.3.1.3, dit is het afhalen en uitpakken van de NuGet package. Daarna verandert er een stap, in plaats van een PowerShell commando moet er een Windows batch commando uitgevoerd worden, deze is te zien op Figuur 43. In dit commando gaat Liquibase opgeroepen worden om de gewenste database te updaten. Er moeten verschillende parameters ingesteld worden zodat het update commando uitgevoerd kan worden.



Figuur 43: Liquibase windows batch commando

4 Conclusie

4.1 Bespreking literatuurstudie

Uit de literatuurstudie kan geconcludeerd worden dat op DevOps manier te werk gaan met databases altijd voordelig is. De nadelen wegen niet op tegen de voordelen. Deze zijn namelijk meer gebonden aan het aanpassingsvermogen van de werknemers. Het is van belang dat zij vertrouwen op de voordelen van geautomatiseerde processen. De voordelen omvatten meer dan enkel het versnellen van de ontwikkeling. Zo zorgt het ook voor stabielere code en betere *deployments* naar andere omgevingen, omdat fouten sneller gedetecteerd worden.

4.2 Bespreking proof of concepts

Er zijn twee toepassingen gemaakt tijdens dit onderzoek. Enerzijds, de POC waarin Redgate Source Controle for Oracle gebruikt wordt als broncontrolesysteem en anderzijds, een POC waarin Liquibase gebruikt wordt. De reden waarom deze ten opzichte van mekaar zijn vergeleken is om het verschil aan te tonen tussen een commercieel en een open source broncontrolesysteem voor databases.

Uit proefondervindelijk onderzoek kan er vastgesteld worden dat Liquibase geschikt is om te gebruiken binnen kleine projecten. Voor grote projecten is een commerciële tool zoals de Source Control van Redgate aan te raden.

Binnen Liquibase dienen de scripts manueel gemaakt te worden en in een bestand gezet te worden. Deze handelingen vergen veel tijd wanneer er met een grote database gewerkt wordt. Redgate detecteert automatisch de veranderingen in de database waarna deze in een grafische applicatie naar het versiebeheersysteem gestuurd worden.

4.3 Mogelijke uitbreidingen

Nu er aangetoond is dat de database makkelijk in de DevOps processen kan opgenomen worden, kan er verder gekeken worden naar mogelijke uitbreidingen. Bij DevOps wordt er vaak te weinig aandacht gegeven aan beveiliging en gegevensbescherming. In de opgenomen bronnen werd er regelmatig gebotst op tools die hier aandacht aan besteden en in de DevOps-processen geïntegreerd kunnen worden. Deze tools houden onder andere rekening met de GDPR-wetgeving waarbij productiedata gemanipuleerd wordt zodat deze data bruikbaar is in testomgevingen. Een uitbreiding op dit onderzoek zou zijn: het introduceren van dit soort tools binnen DevOps-processen.

4.4 Relevantie voor het stagebedrijf

Bij de klant waar de stage uitgevoerd werd, moesten databasescripts nog manueel uitgevoerd worden. Dit leidt vaak tot onduidelijkheden over welke scripts al zijn uitgevoerd. Om deze reden is er onderzoek gedaan naar de mogelijkheid om tools in te schakelen die kunnen dienen als versiecontrole voor de database. Binnen mijn onderzoek is er dieper ingegaan op dit soort tools en wat de voordelen zijn van versiecontrole. De voordelen zijn onder andere dat er een overzicht verkregen wordt van de scripts die al uitgevoerd zijn. Daarnaast is versiecontrole de eerste stap richting DevOps. Dit zorgt ervoor dat de database opgenomen kan worden in CI/CD-processen, waardoor er een snellere *time-to-market* kan plaatsvinden.

4.5 Reflectie

Oorspronkelijk ging mijn onderzoek enkel over broncontrole op de database en tools die dit mogelijk maken. Naarmate ik meer onderzoek deed naar dit topic werd mijn interesse gewekt door database DevOps. Dit leidde ertoe dat mijn de scope van mijn onderzoek zich uitbreidde van broncontrole op databases naar database DevOps. Ik heb binnen dit onderzoek heel veel bijgeleerd over wat DevOps precies inhoudt en hoe een database geïntegreerd kan worden binnen deze processen.

Voor ik aan dit onderzoek begon had ik nooit stilgestaan bij het feit dat de database meegenomen kan worden in broncontrole. Nu ik hier onderzoek naar heb gedaan besef ik pas wat ik gemist heb sinds dat ik zelf in groepswerken applicaties maak met een bijhorende database.

5 Bibliografie

[1] „Accenture,” Wikipedia, [Online]. Available: https://nl.wikipedia.org/wiki/Accenture. [Geopend 24 mei 2019].

[2] „Accenture,” Accenture, [Online]. Available: https://www.accenture.com/be-en. [Geopend 24 mei 2019].

[3] G. Wilton, „What is a fit gap analysis,” Quora, 4 Mei 2017. [Online]. Available: https://www.quora.com/What-is-a-fit-gap-analysis. [Geopend 24 mei 2019].

[4] „Wat is DevOps?,” Senet, 14 januari 2016. [Online]. Available: https://www.senet.nl/blog/wat-is-devops/. [Geopend 24 mei 2019].

[5] „Wat is DevOps?,” Microsoft, [Online]. Available: https://azure.microsoft.com/nl-nl/overview/what-is-devops/. [Geopend 24 mei 2019].

[6] D. Brown, „What is DevOps?,” 1 september 2015. [Online]. Available: http://donovanbrown.com/post/what-is-devops?WT.mc_id=devops-channel9-cephilli. [Geopend 1 juni 2019].

[7] „What is Continuous Integration?,” Amazon, [Online]. Available: https://aws.amazon.com/devops/continuous-integration/. [Geopend 28 mei 2019].

[8] „Continuous integration vs. continuous delivery vs. continuous deployment,” Atlassian, [Online]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment. [Geopend 28 mei 2019].

[9] „Continuous Delivery vs Deployment vs Integration: What's the Difference?,” BMC, [Online]. Available: https://www.bmc.com/blogs/continuous-delivery-continuous-deployment-continuous-integration-whats-difference/. [Geopend 28 mei 2019].

[10] M. Hilbert, „Bringing DevOps to the database. Part 1: Version control,” Redgate, 15 Januari 2018. [Online]. Available: https://www.red-gate.com/blog/database-devops/bringing-devops-database-part-1-version-control. [Geopend 28 Mei 2019].

[11] M. Robbins, „Six reasons to version control your database,” Redgate, 10 Mei 2019. [Online]. Available: https://www.red-gate.com/blog/database-devops/database-version-control-3. [Geopend 28 Mei 2019].

[12] M. Hilbert, „Devops doesn't stop at the database,” 5 juni 2017. [Online]. Available: https://devops.com/devops-doesnt-stop-database/. [Geopend 1 juni 2019].

[13] S. Nitsche, „One does not simply update a database - migration based database development,” 15 januari 2018. [Online]. Available: https://dev.to/pesse/one-does-not-simply-update-a-database--migration-based-database-development-527d. [Geopend 2 juni 2019].

[14] T. Austin, „Moving from application automation to true DevOps by including the database,” Redgate, 20 juni 2018. [Online]. Available: https://www.red-gate.com/hub/product-

learning/sql-change-automation/moving-from-application-automation-to-true-devops-by-including-the-database. [Geopend 1 juni 2019].

[15] M. Hilbert, „Bringing DevOps to the database. Part 2: Continuous delivery," Redgate, 22 januari 2018. [Online]. Available: https://www.red-gate.com/blog/database-devops/bringing-devops-to-the-database-part-2-continuous-delivery. [Geopend 3 juni 2019].

[16] J. Kanjilal, „Database DevOps: Why you need it - and how to do it," Techbeacon, 3 december 2018. [Online]. Available: https://techbeacon.com/devops/database-devops-why-you-need-it-how-do-it. [Geopend 31 mei 2019].

[17] S. M. Consulting, „What is database DevOps, and why is it important for DBAs?," Redgate, 14 December 2017. [Online]. Available: https://www.red-gate.com/blog/database-devops/what-is-database-devops-and-why-is-it-important-for-dbas. [Geopend 3 Juni 2019].

[18] S. Horrocks, „The pros and cons of adopting a DevOps model," AppDynamics, 24 April 2017. [Online]. Available: https://www.computerworld.com.au/article/618100/pros-cons-adopting-devops-model/. [Geopend 3 Juni 2019].

[19] R. Pruess, „Five benefits of DevOps for Database and how to achieve them," Devops Zone, 21 Februari 2019. [Online]. Available: https://dzone.com/articles/five-benefits-of-devops-for-database-and-how-to-ac. [Geopend 30 May 2019].

[20] „15 Version control software tools," 23 April 2019. [Online]. Available: https://www.softwaretestinghelp.com/version-control-software/. [Geopend 7 Juni 2019].

[21] B. Putano, „CI/CD Tools Throwdown: Jenkins vs. TeamCity vs. Bamboo," Stackify, 3 Februari 2018. [Online]. Available: https://stackify.com/jenkins-teamcity-bamboo/. [Geopend 8 Juni 2019].

[22] „Deployment Suite for Oracle," Red Gate, 29 April 2015. [Online]. Available: https://documentation.red-gate.com/oracletools. [Geopend 10 Juni 2019].

[23] C. Nies, „POC DatabaseDevops," [Online]. Available: https://github.com/ChristofNies/POC_DatabaseDevOps.