SQL Konferenz 2023
powered by PASS Deutschland e.V.

PASS Microsoft Data Platform Community
DEUTSCHLAND e.V.

Azure DevOps Build, Deploy, Pull Requests and Testing for your Data Warehouse

13. September 2023
SQL Konferenz Hanau
Felix Möller, noventum consulting GmbH

Über mich

Felix Möller

- Service Lead Microsoft Intelligent Data Platform bei noventum
- Erfahrungen seit gut 6 Jahren mit der Azure Data Platform
- Seit 11 Jahren Microsoft BI
- Konferenzsprecher
  - 2021: Understanding Costs of your Data Estate with Power BI
  - 2021: Chocolatey for the Data Platform Guy
  - 2019: Advanced Modelling – Power Platform World Tour, München (Slides)
  - 2019: CI/CD for your Cloud Data Platform, München (Slides)
  - 2018: TFS / VSTS for SSIS projects, St. Augustin (Slides)
  - 2018: Power BI Dashboard in a Day, Köln
  - 2015: Einführung in BIML metadatengetriebene DWH-Entwicklung, Hannover (Slides)
- Fokusthemen
  - Data Platform DevOps
  - Metadata-driven DWH-Entwicklung
  - SAP-Datenextraktion
- E-Mail: felix.moeller@noventum.de

data & analytics @noventum consulting

AZURE DATA GUY

https://azdataguy.com/

Publikumsfragen

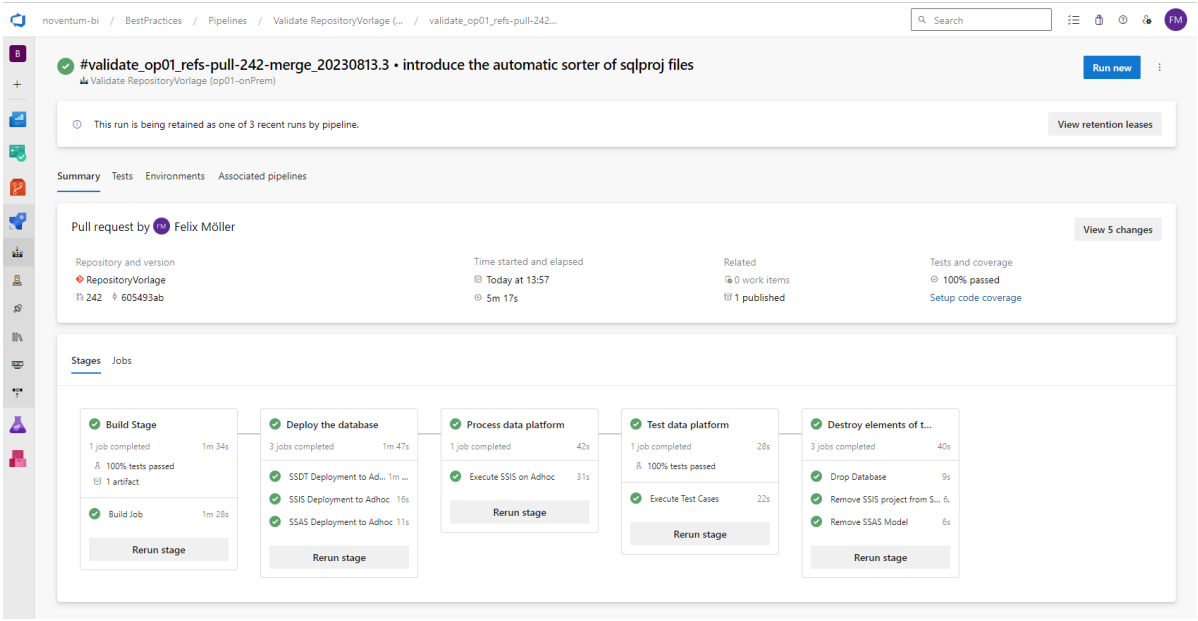
Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

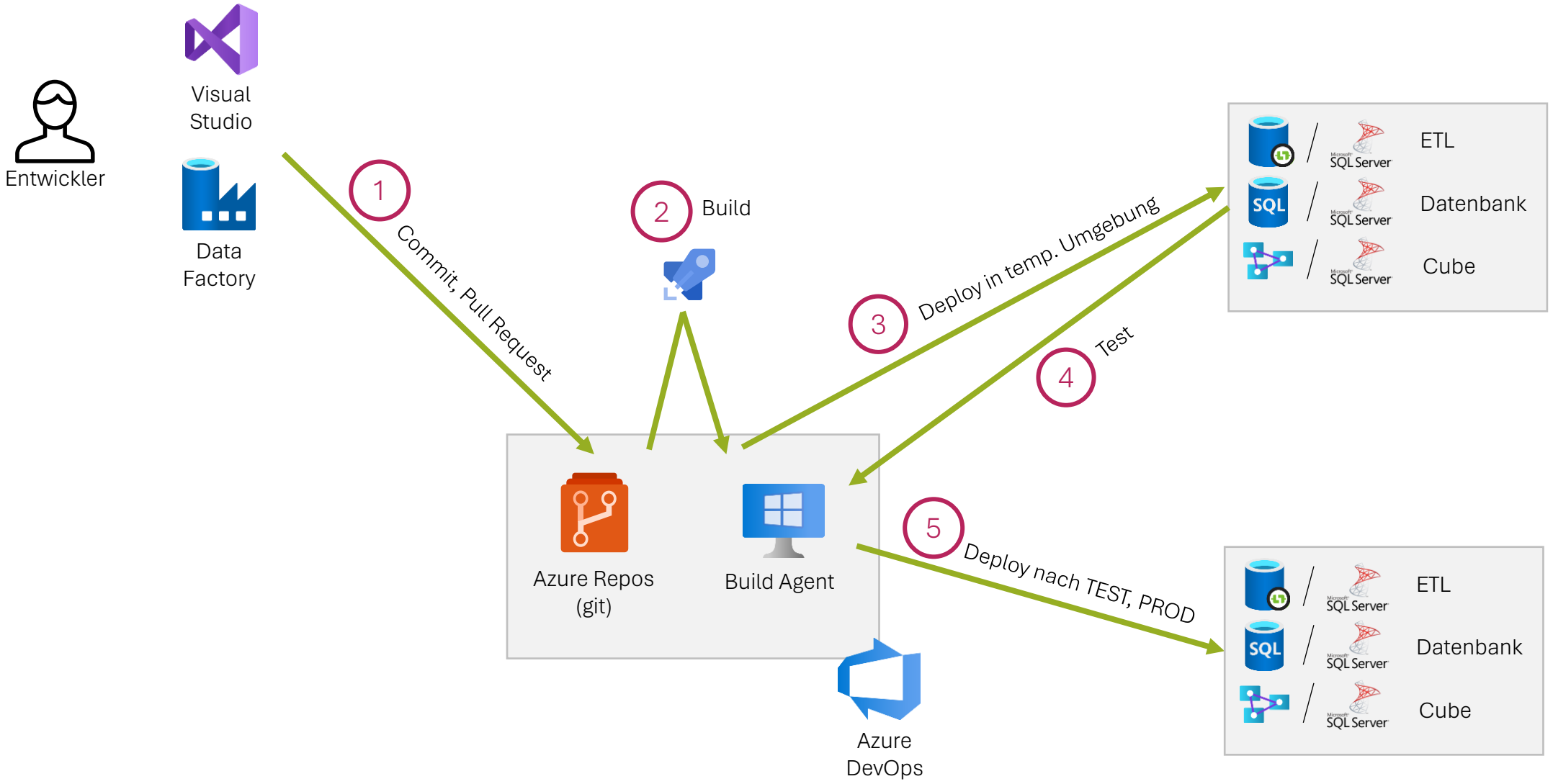
Qualität erhöhen, effizient zusammenarbeiten

Zielsetzung

- Qualität erhöhen, schneller bereitstellen
  - Build – eigener Agent und mit MSFT-hosted Agent
  - Deployment – onPrem und Azure PaaS
  - Integrationstest
  - Datentests
- Vollständige Dokumentation
  - Infrastruktur
  - Build
  - Deployment
- Skalierung für Zusammenarbeit mehrerer Entwickler
  - Parallele Entwicklungen mit Pull Requests
- Vereinfachung des Deployments
  - Deployen in jede Umgebung ohne manuelle Schritte



Die Schritte des DevOps-Prozesses

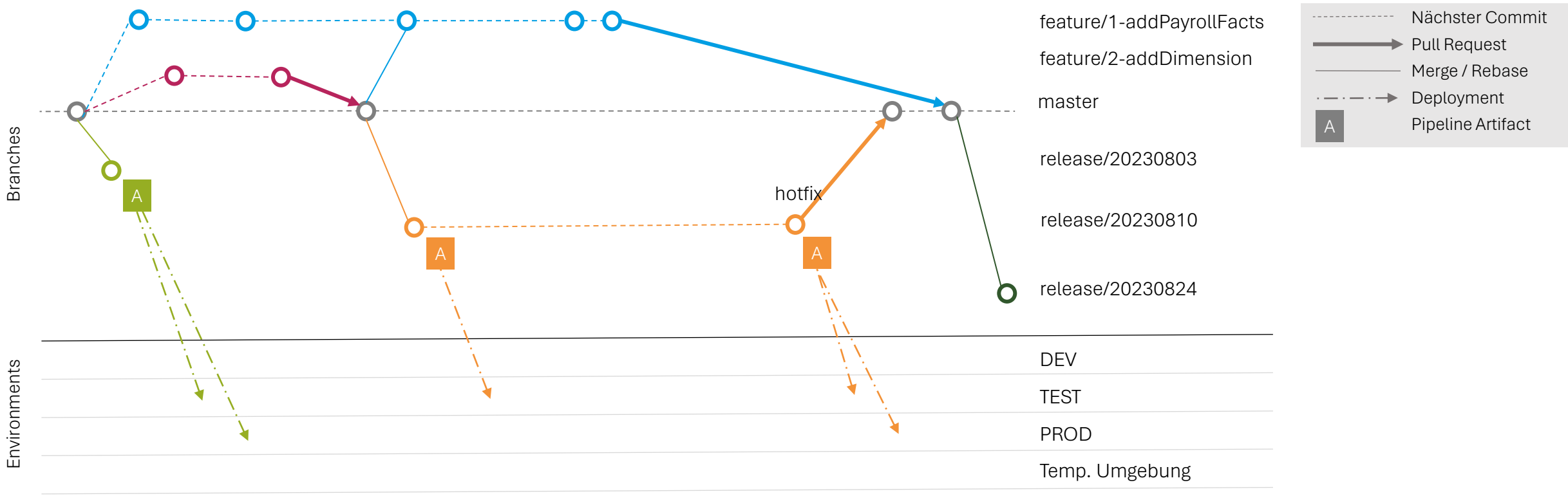


Azure Icons: https://github.com/David-Summers/Azure-Design

Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

Branching: basierend auf MS Release Flow



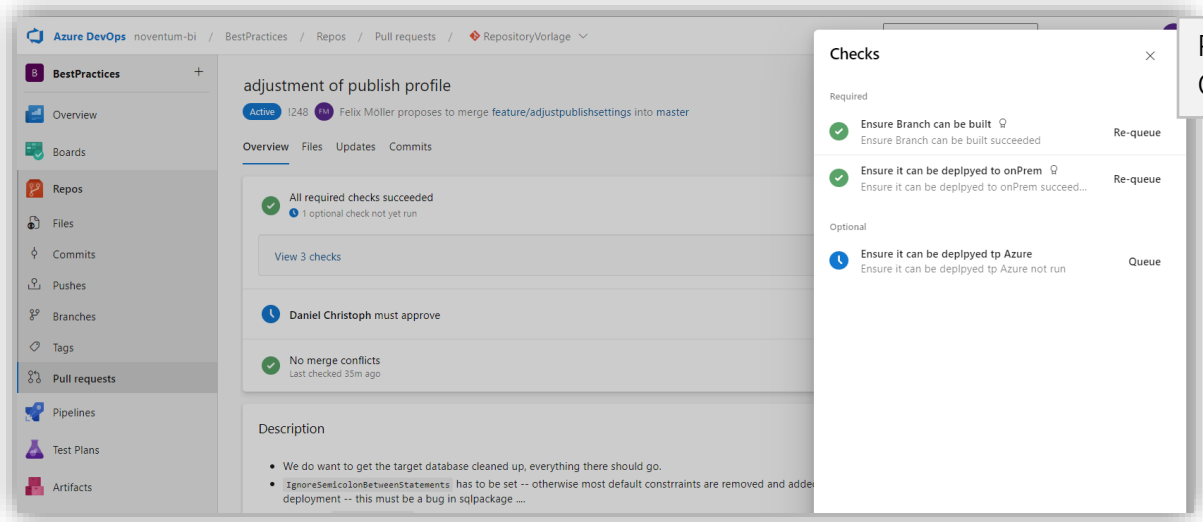
Beschreibung

- Es werden Releases erzeugt, aus diesen entstehen Artefakte, die in Umgebungen deployt werden
- Details unter How Microsoft develops with DevOps - Azure DevOps | Microsoft Learn, Release Flow: How We Do Branching on the VSTS Team - Azure DevOps Blog (microsoft.com)

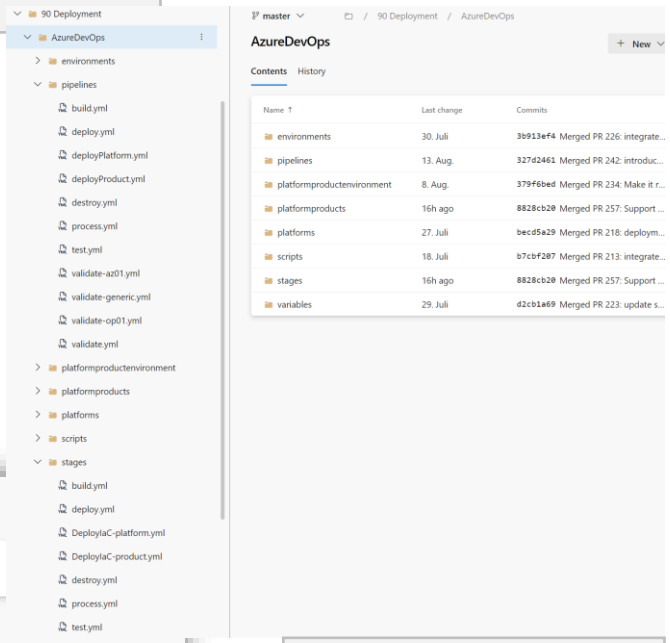
Bewertung

- Vorteil: Eine führende Branch (master), keine Drift zwischen Branches möglich
- Vorteil: Gleicher Code in alle Umgebungen released
- Vorteil: Entkoppelung von Deployment und Pull Request Merge

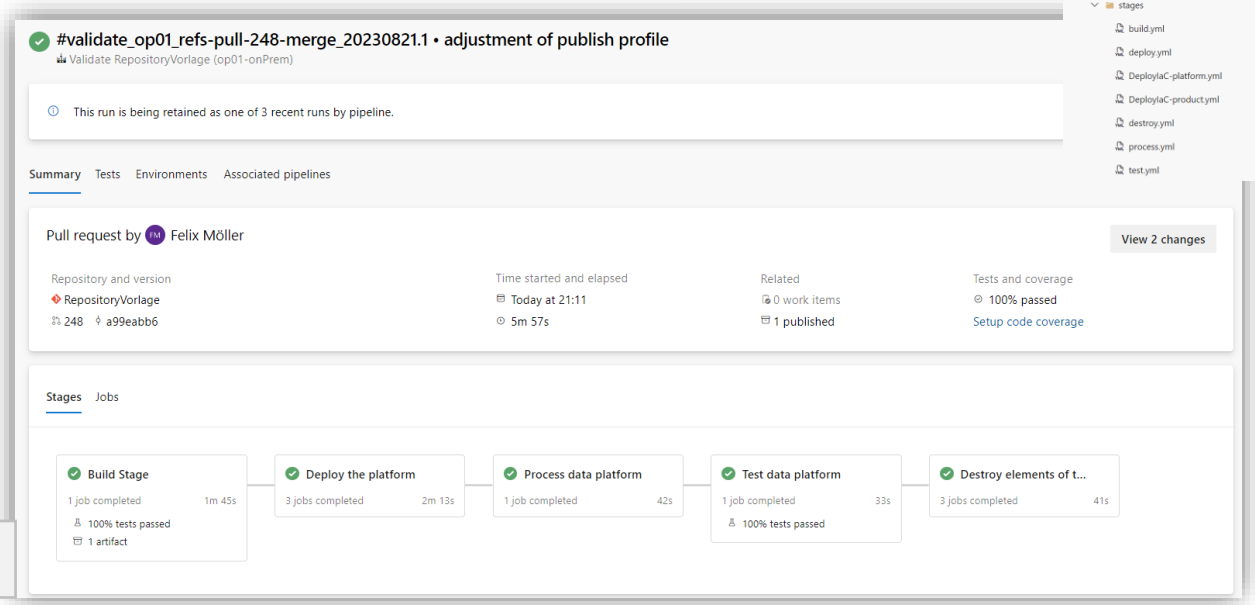
Pull Request validiert sich automatisch



Pull Request führt automatisch Checks aus



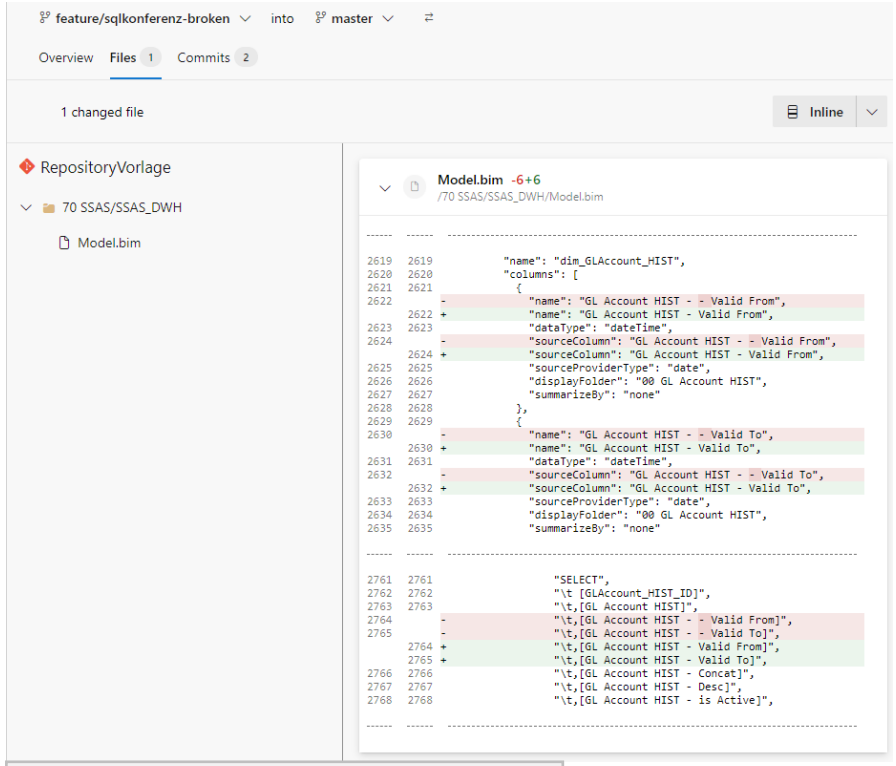
YAML Pipelines



Ein Check testet das Deployment

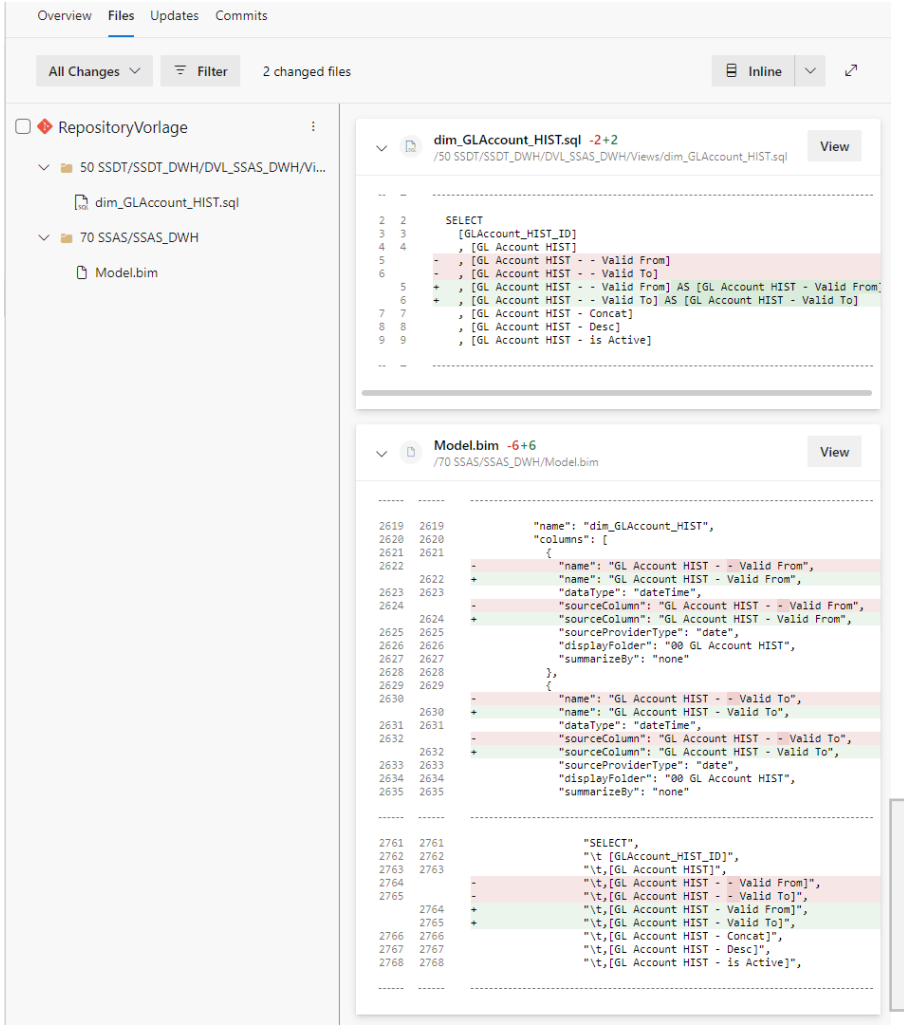
Live Demo

Demo



Anpassung am View vergessen

feature/sqlkonferenz-broken



Ganzheitliche Anpassung

feature/sqlkonferenz-good

Gleiche Formatierung I/III

- Bei paralleler Entwicklung entstehen Konflikte, diese sollten vermieden werden
- Pull Request lassen sich nur effizient reviewen, wenn der diff möglichst klein ist
- Git unterstützt „pre-commit“ hooks, um dies zu automatisieren

- Pre-commit https://pre-commit.com/
  - Framework für Tools, die vor einem Commit ausgeführt werden
  - Bspw. Leerzeichen statt Tabs, einheitliche Windows Zeilenumbrüche, einheitlich UTF-8, XML-Formatierung, YAML-Validierung
  - Wird mit `pre-commit install` in einem git Repository aktiviert

Gleiche Formatierung II/III

- Sqlfluff https://github.com/sqlfluff/sqlfluff
  - Formatiert SQL Code für fast jede Datenbank
  - Sehr feingranular konfigurierbar
    - Einheitliches Casting (**CAST** vs **CONVERT**)
    - einheitliches Aliasing (**AS** vs „“)
    - einheitliche Reihenfolge in **JOIN**s
    - Einheitliche Kommas
    - Perfekte Einrückung
- Leider nicht sehr schnell

Demo

query.sql | .sqlfluff | 1-demo.ps1

202309-SQLKonferenz > .sqlfluff

```
# otherwise {{ gets interpreted as Jinja template which for us certainly is wrong
templater = raw
# otherwise initial formatting of medium complex SQL statements does not work
runaway_limit = 50
large_file_skip_byte_limit = 0
max_line_length = 200

[sqlfluff:layout:type:comma]
line_position = leading

[sqlfluff:indentation]
tab_space_size = 2

[sqlfluff:rules:references.quoting]
prefer_quoted_identifiers = True
```

query.sql | .sqlfluff | 1-demo.ps1

202309-SQLKonferenz > 1-demo.ps1

```
& sqlfluff.exe fix --config ".sqlfluff"  "query.sql" --fixed-suffix fixed --force
```

```
SELECT
   c.Customer_KEY
  , m.Material_KEY,
 CONVERT(int, Quantity) AS Quantity,
    Price * Quantity Amount
  , case when MaterialType_KEY = 'ZC' THEN 'A' ELSE CASE WHEN
MaterialType_KEY = 'ZD' THEN 'B' ELSE 'C' END END AS
MaterialCategory_KEY
FROM DPL.fact_sales s
left join DPL.dim_customer c ON (c.Customer_ID = f.Customer_ID)
LEFT JOIN DPL.dim_material as m on (f.Material_ID = m.Material_ID)
WHere s.Fiscalyear BETWeen 2012 and 2023
```

vorher

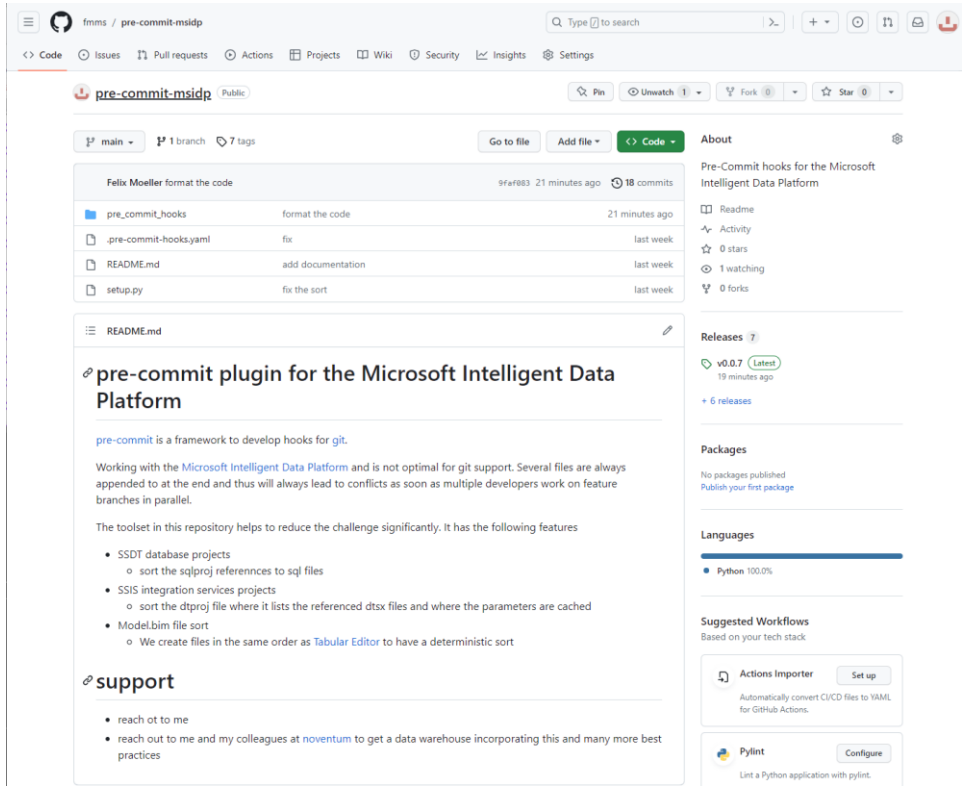
```
SELECT
  c.Customer_KEY
  , m.Material_KEY
  , CAST (Quantity AS int) AS Quantity
  , Price * Quantity AS Amount
  , CASE
    WHEN MaterialType_KEY = 'ZC' THEN 'A'
    WHEN MaterialType_KEY = 'ZD' THEN 'B'
    ELSE 'C'
  END AS MaterialCategory_KEY
FROM DPL.fact_sales f
LEFT JOIN DPL.dim_customer c ON (c.Customer_ID = f.Customer_ID)
LEFT JOIN DPL.dim_material m ON (m.Material_ID = f.Material_ID)
WHERE s.Fiscalyear BETWEEN 2012 AND 2023
```

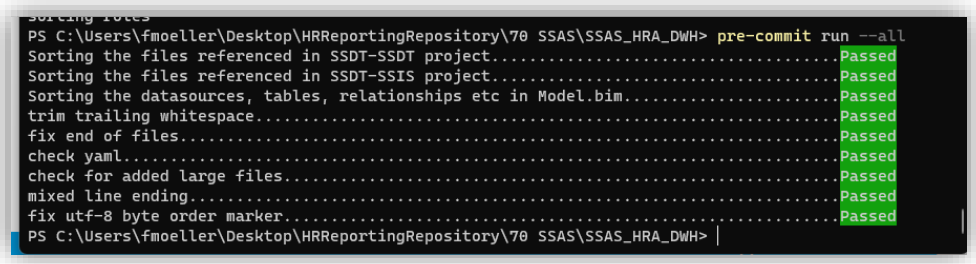
nachher

Gleiche Formatierung III/III

- Größtes Konfliktpotential in Projektdateien für SSIS, SSDT
  - Neue Artefakte immer unten angehängt
  - Immer ein Merge-Konflikt

- Intelligent Data Platform hooks https://github.com/fmms/pre-commit-msidp
  - sortiert alle Einträge im **`Build`** und **`Folder`** der Projektdatei (**`.sqlproj`**)
  - Sortiert alle Referenzen auf SSIS-Pakete in Projektdatei (**`.dtproj`**)
  - Sortiert alle Objekte in **`Model.bim`** entsprechend der Tabular Editor Serialisierung – effizientere Reviews





Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

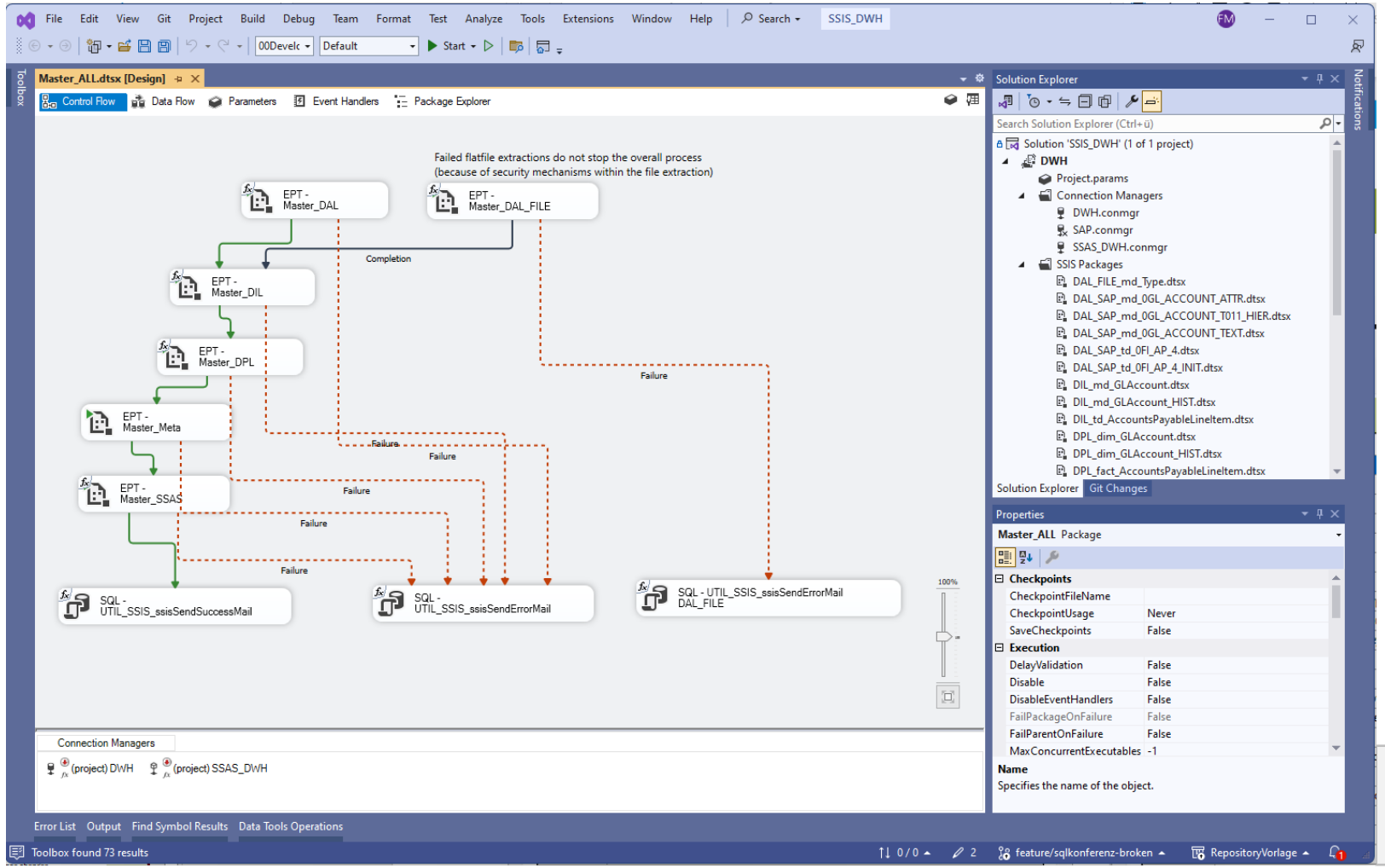
Was passiert im Build-Vorgang

Technologie	Source Code	Deployment Artefakt	
 Integration Services	*.dtsx	*.ispac	1
 SQL Datenbank	*.sql	*.dacpac	2
 Analysis Services	Model.bim	Model.asdatabase	3
 Data Factory	*.json	ARM-Template *.json mit azure.datafactory.tools	4
 Power BI	*.pbip, *.pbix	*.pbix	

Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

Integration Services



Integration Services 1.1
In Visual Studio 2022

1 Integration Services

Build

- Git Integration seit Visual Studio 2017
- SSIS Pakete sind XML-Dateien
  - Durch enthaltene Metadaten sehr umfangreich
  - Layout Informationen sorgen für Konflikte
- Pre-commit kann verwendet werden um Layoutinformationen zu entfernen, entfernt aber auch Kommentare …

Deploy

- SSIS normalerweise deployt mit `ISDeploymentWizard.exe`
- Deployment in Azure-SSIS Runtime mit AccessToken nur möglich über SMO (PowerShell) (Deploy an SSIS project with PowerShell - SQL Server Integration Services (SSIS) | Microsoft Learn)
- PowerShell Deployment funktioniert nicht auf MSFT-hosted build agents (https://github.com/actions/runner-images/issues/8174)
- Konfiguration über Stored Procedures in der SSISDB (e.g. `[catalog].[set_object_parameter_value]`)

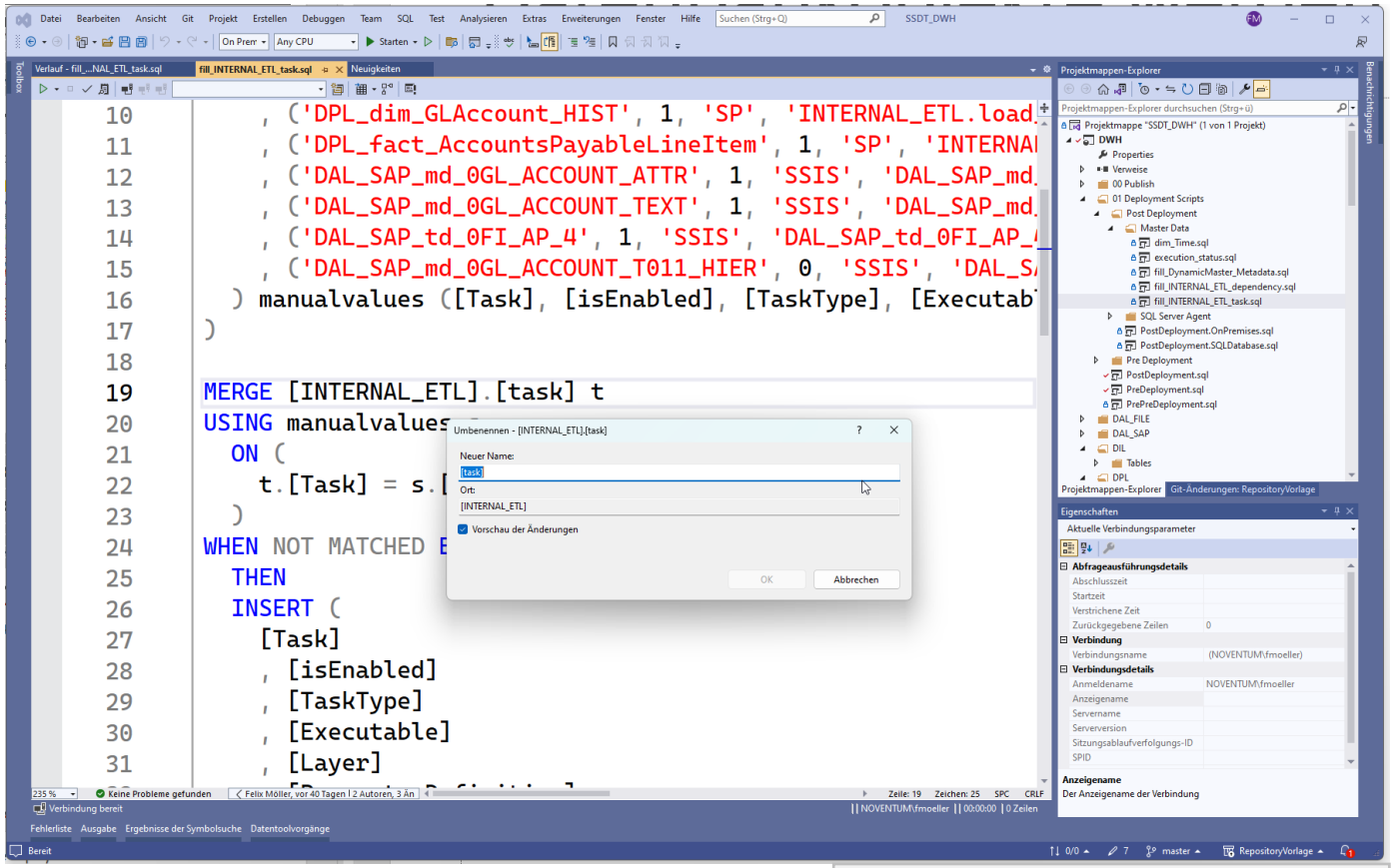
Learnings

- Visual Studio auf Microsoft Hosted Agents ist SEHR langsam – etwa 4 – 9-fache Geschwindigkeit mit Self-Hosted Agent
- `ISDeploymentWizard.exe` unterstützt keine AccessToken-Authentifizierung
- Deployment mit PowerShell über SMO auf onPrem Server ist sehr langsam.

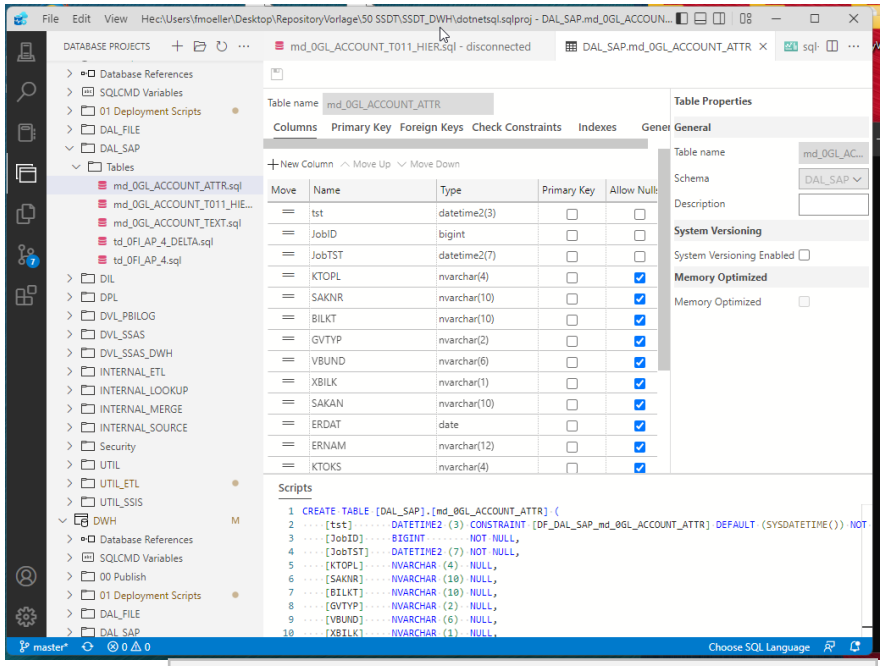
Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

2 Datenbankobjekte werden im Datenbankprojekt verwaltet



SSDT in Visual Studio 2022/2019
Unterstützt Umbenennungen, Refactorings, etc.



Datenbankprojekte in Azure Data Tools basieren auf neuem Projekttyp, aber haben noch sehr viele fehlende Funktionen. (vgl. [SQL Database Projects - add refactoring (which is available in SSDT) · Issue #13920 · microsoft/azuredatastudio (github.com)](#)).

2 Datenbankobjekte

Build

- Git Integration direkt seit Visual Studio 2017
- Datenbankobjekte werden als SQL-Datei abgespeichert
  - `.refactorlog` für Umbenennungen
- Postdeployment-Skripte
- Ergebnis ist eine `dacpac`-Datei

Deploy

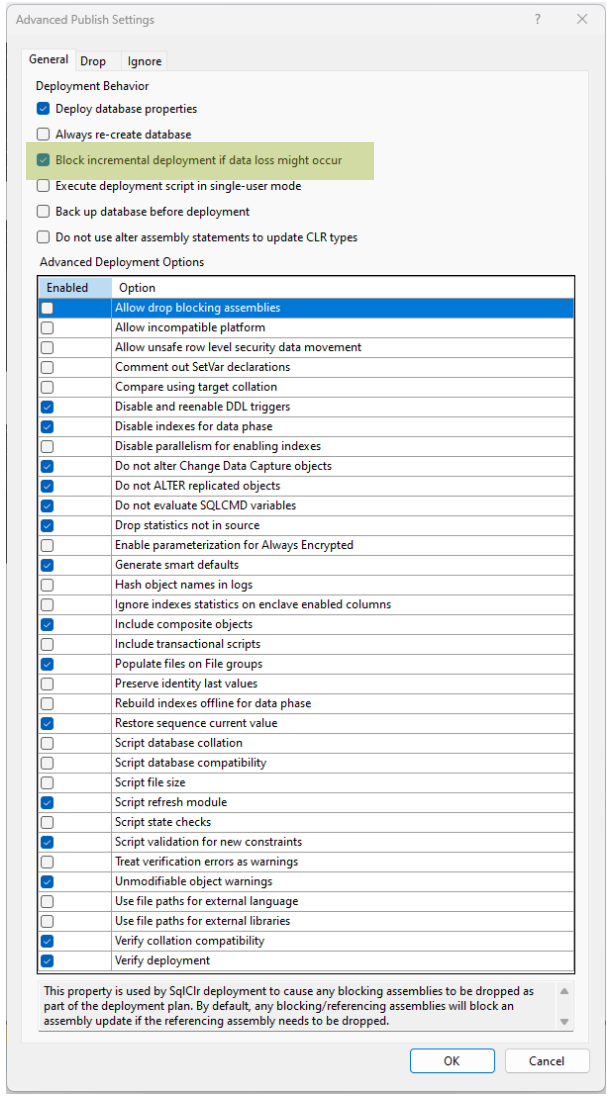
- Dacpac gibt den Sollzustand vor, Datenbank wird per `sqlpackage.exe` entsprechend angepasst
- Allgemeine Deployment Optionen
  - State-basierend: SSDT, dbForge Source Control for SQL Server
  - Migration-basierend: DbUp

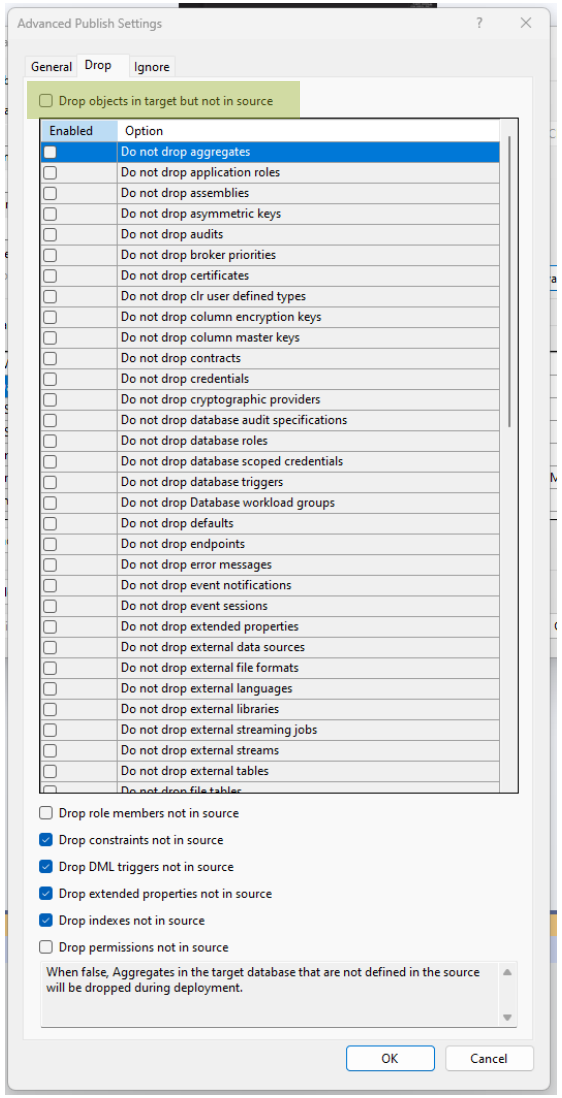
Learnings

- Ausführungsreihenfolge von `sqlpackage.exe`
  1. Vergleich DacPac zu Datenbank
  2. Erzeugen eines Deploymentscripts
  3. Ausführen des Pre-Deploymentscripts
  4. Ausführen des Deploymentscripts (vgl. Schritt 2)
  5. Ausführen des Post-Deployment
- Pre-Pre-Deployment-Skript notwendig

Vgl. https://blogs.msdn.microsoft.com/gertd/2009/09/14/pre-deployment-scripts/
und https://the.agilesql.club/2015/07/pre-compare-pre-deployment-scripts-to-ssdt/

2 Hunderte Optionen im Publish-Profil





Block on Possible Data Loss

- Sicherstellen, dass es zu keinem Verlust kommt

Generate Smart Defaults

- Hilfreich für non-nullable columns mit Default Constraints

Drop objects not in source

- Notwendig um im Ziel nicht immer weitere Objekte anzusammeln

Ignore Whitespace

- Notwendig um nicht mit sqlfluff Formatierungen in Endlosschleife zu landen.

Script database collation

- Project auf Case Sensitive, Datenbank kann auf Case insensitive bleiben

Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

3 Tabular Models

Build

- Entwicklung in Tabular Editor 2 (https://github.com/TabularEditor/TabularEditor)
- Microsoft AS Designer vernachlässigt (Microsoft Analysis Services Projects - Visual Studio Marketplace)
  - Visual Studio 2022 letztes Update April 2022
  - Visual Studio 2019 letztes Update Juli 2021
- Provider- statt Structured-DataSources zu bevorzugen (vgl. [1] und [2])

Learnings

- Authentifizierung gegen Azure Analysis Services ist möglich mit der Service Connection aus Azure DevOps: `User ID=;Password=AccessToken`

Deploy

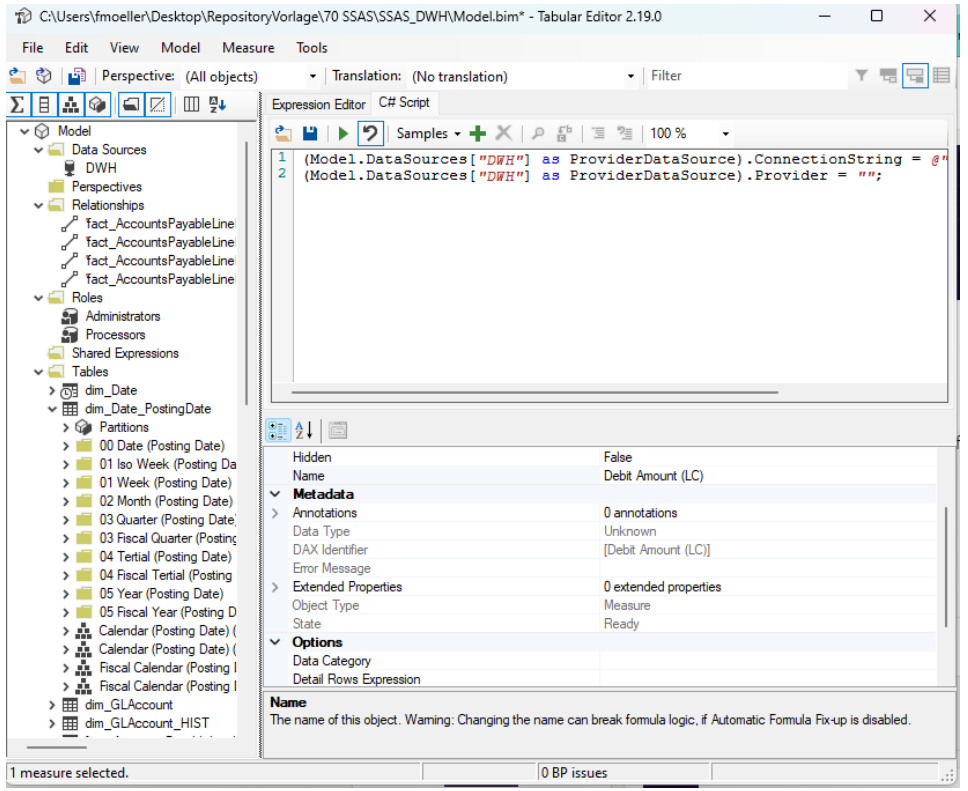
- `Microsoft.AnalysisServices.Deployment.exe` Teil von SSMS nicht vorhanden auf MSFT-hosted Agents
- Tabular Editor erlaubt sehr bequemes Deployment, Datasources können im Deployment über ein C#-Skript aktualisiert werden.
- XMLA-Endpunkte (Premium) ermöglicht Deployment von Power BI Datasets

```
$aasato = $(Get-AzAccessToken -Resource "https://westeurope.asazure.windows.net").Token
$modelbim = "$(Agent.BuildDirectory)\mybuild\arti1\cube\Model.asdatabase"
$asModel = "RepositoryVorlage"
$connstr = "Provider=MSOLAP;Data Source=$(asServerFQDN);User ID=;Password=$aasato"

$cmd = "C:\Program Files (x86)\Tabular Editor\TabularEditor.exe"
$params = @(
"""$modelbim""",
"-DEPLOY", """$connstr""", """$asModel""",
"-ROLES",
"-MEMBERS",
"-OVERWRITE",
"-CONNECTIONS",
"-SCRIPT", "model.cs",
"-VSTS"
)
$p = Start-Process -Wait -NoNewWindow -PassThru -FilePath $cmd -ArgumentList $params
exit($p.ExitCode)
```

3 Tabular Editor

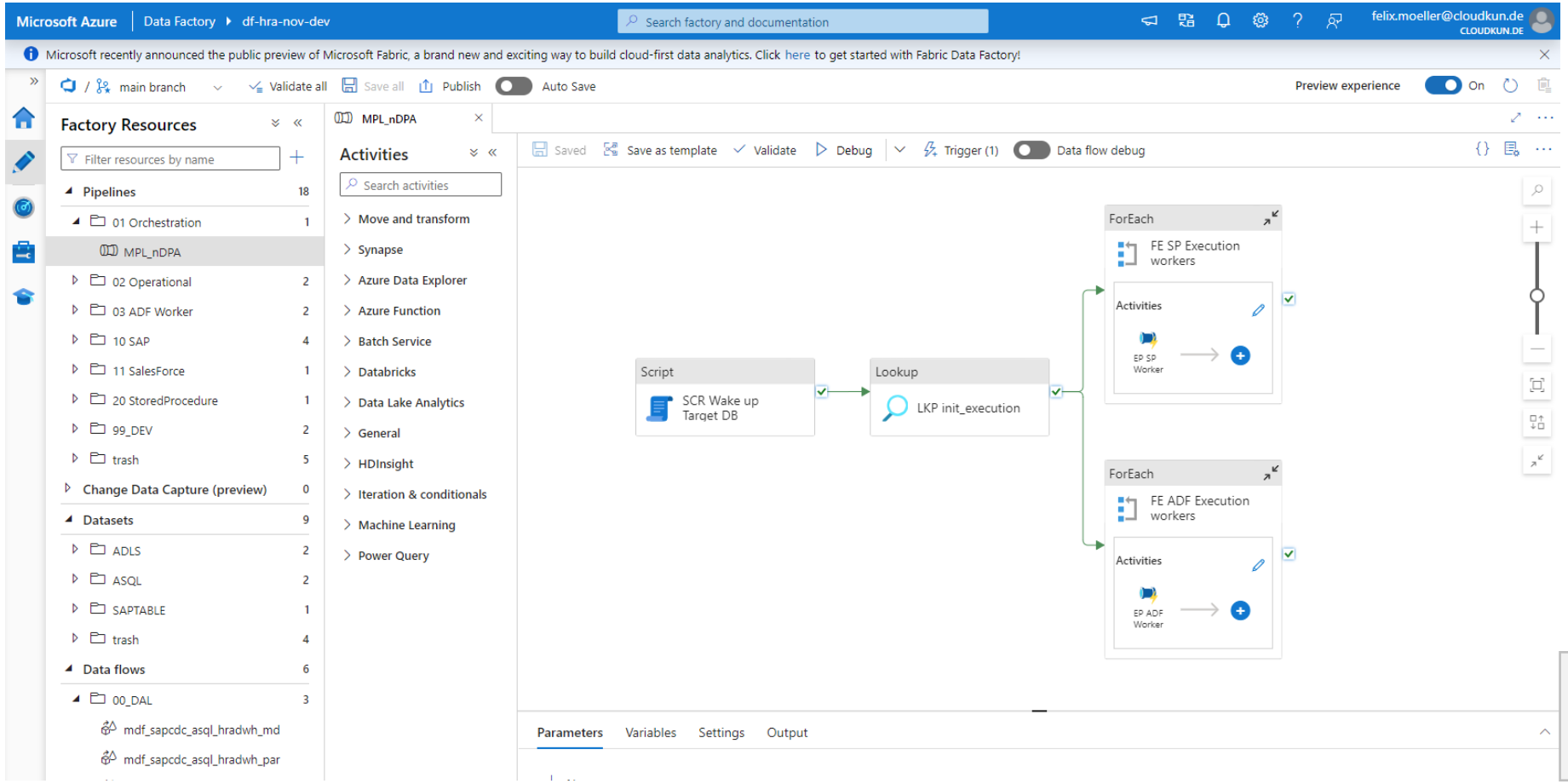
- Entwicklung von Daniel Otykier
- Viel schneller und stabiler als SSAS in VS
- Erlaubt es Best Practices zu validieren (vgl. GitHub - TabularEditor/BestPracticeRules: An official collection of standard Rules for use with Tabular Editor's Best Practice Analyzer.)
- Integration mit DaxFormatter (vgl. DAX Formatter by SQLBI)
- C# Skripte möglich zur Automatisierung
- Kommandozeile bietet viele Optionen zur Automatisierung
- Kann als Ersatz für Visual Studio verwendet werden, wenn man auf PowerQuery/Structured-Datasources verzichtet
- Tabular Editor 3 – Entwicklungsumgebung mit u.a. DAX Intellisense – 90 EUR/Benutzer/Monat



Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

Azure Data Factory



ADF
Branching und git Integration direkt im Browser.

4 Data Factory

Build

- Git Integration direkt im ADF Studio
- Azure Data Factory Pipelines werden als JSON abgespeichert
- Zwei Wege zum ARM-Template
  - ARM-Templates in adf_publish Branch (vgl. Continuous integration and delivery - Azure Data Factory | Microsoft Learn)
  - NPM basiertes Build (vgl. Automated publishing for continuous integration and delivery - Azure Data Factory | Microsoft Learn)

Deploy

- Zwei Optionen des Deployments
  - ARM JSON mit Standardtools
  - ADF JSON mit azure.datafatory.tools (vgl. GitHub - Azure-Player/azure.datafactory.tools: Tools for deploying Data Factory (v2) in Microsoft Azure)

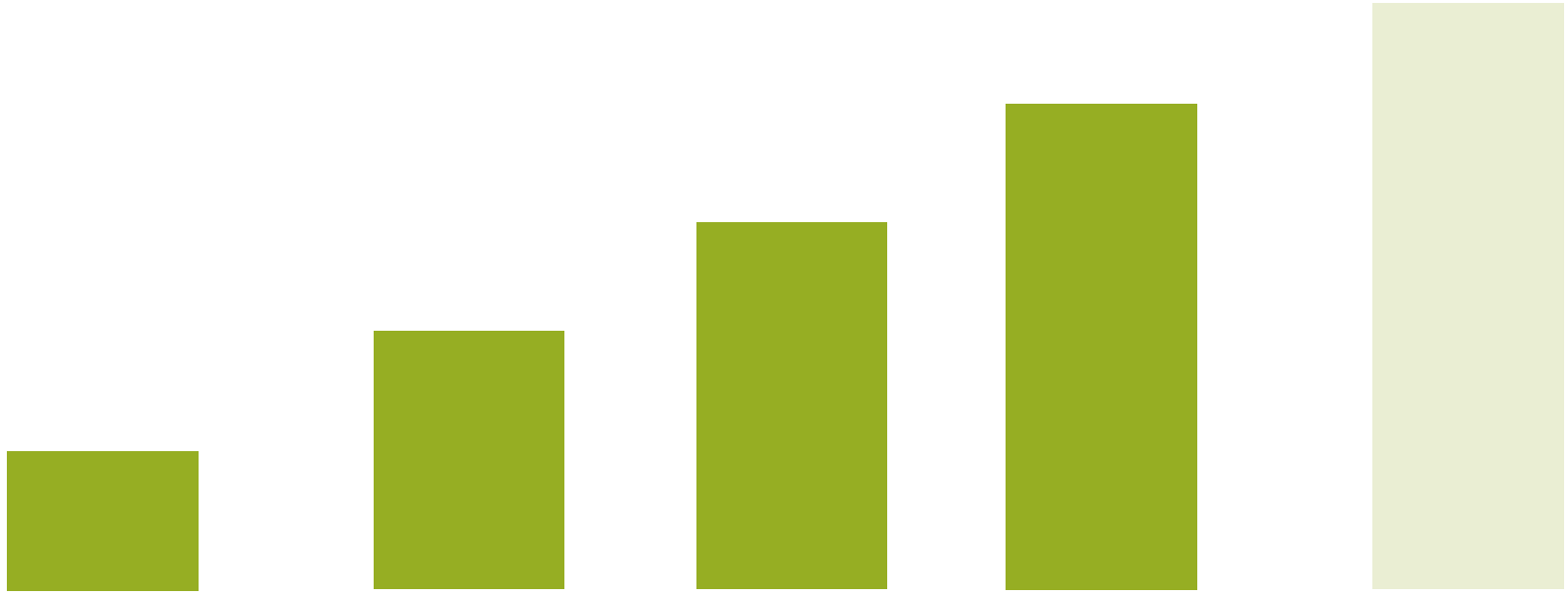
Learnings

- azure.datafactory.tools erlaubt wesentlich bequemeres Deployment

Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

Verschiedene Stufen des Testens



Build erfolgreich

1

Validierungen gegen den Code (Pester)

2

Lösung lässt sich in neuer Umgebung ausführen

3

Testfälle gegen die Daten zeigen korrekte Ausführung (NBi)

4

Lösung lässt sich im Clone des PROD-Systems ausführen

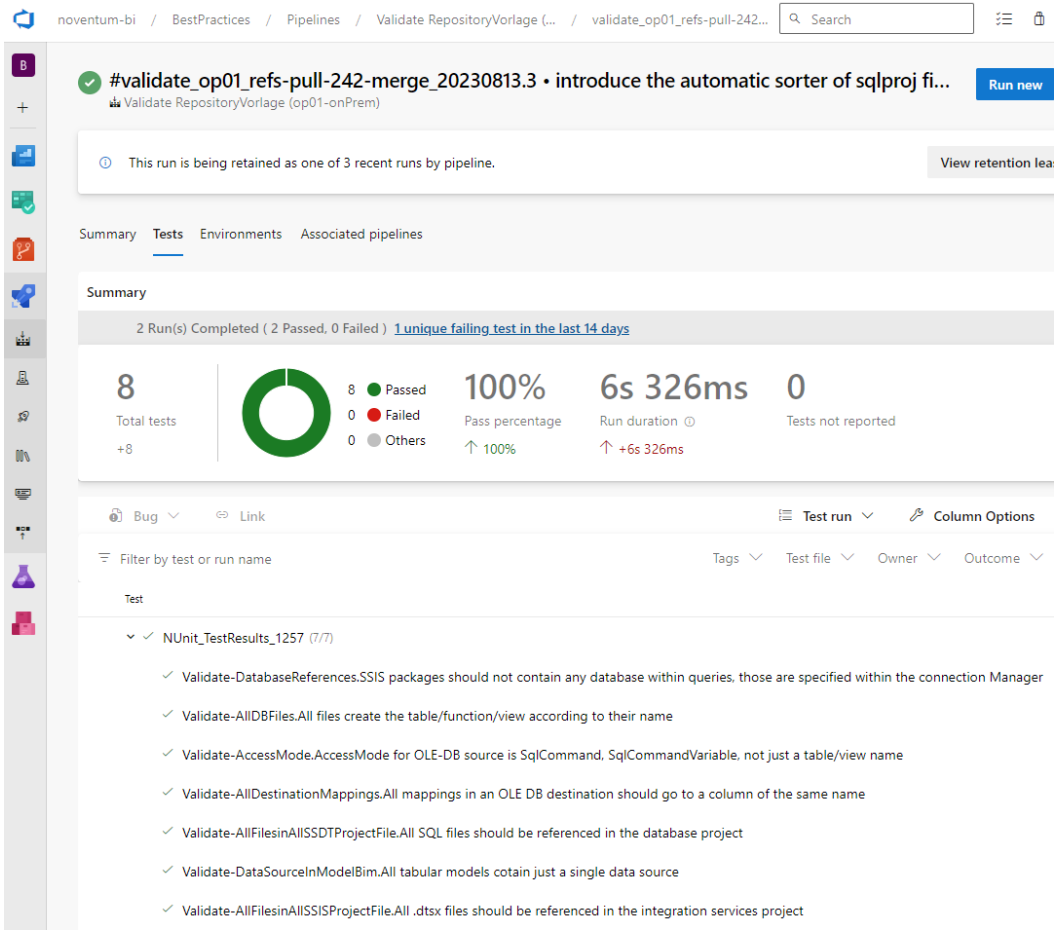
1 Build des Projekts ist ein guter erster Test

- SSIS
  - fehlende Pakete im Projekt
- SSDT
  - Fehlende Abhängigkeiten – referenzierter View oder Tabelle existiert nicht
  - Syntaxfehler (bspw. SELET statt SELECT)

2 Pester das PowerShell Testing Framework

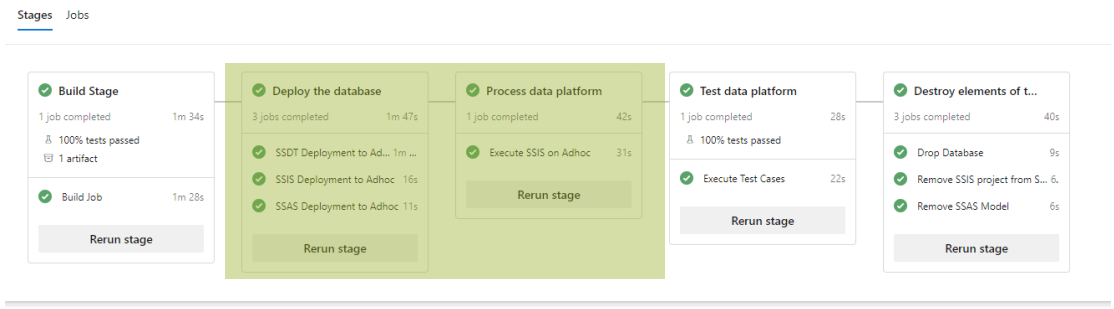
- Pester ist ein Framework um PowerShell Testfälle zu definieren
  - Kann lokal ausgeführt werden
  - Integriert sich mit Azure DevOps

- SSIS
  - Keinerlei Datenbankqualifizierter Queries in Sources
  - Stets ein SELECT-Statement statt „Table or View“
- Datenbankprojekt
  - Dateinamen passen zum zugehörigen SQL Objekt
  - Alle Dateien sind auch Teil der sqlproj-Datei



3 Integrationstest

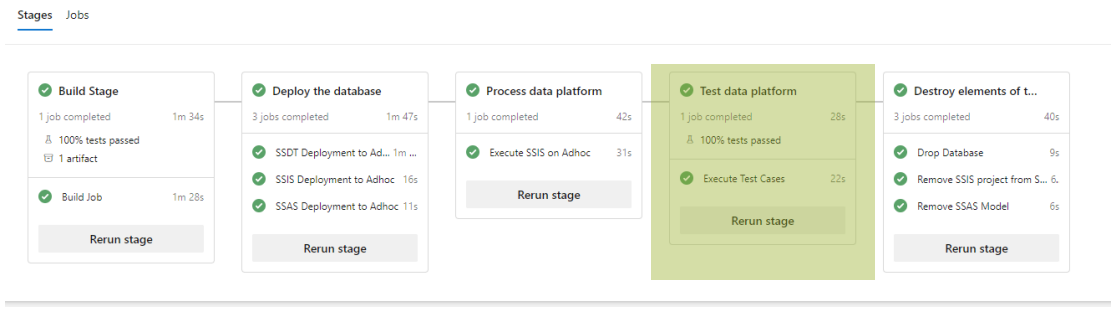
- Viele Fehler verursacht durch Toolbrüche
  - AS greift auf Spalte zu, die es nicht gibt
  - ADF nutzt Stored Procedure, die umbenannt wurde
  - Datentypen zwischen SSIS und der Datenbank weichen voneinander ab
- Umsetzung
  - Deployment in temporäre Umgebung
    - Azure: weitere Resourcen per bicep (vgl. https://github.com/Azure/bicep)
    - onPrem: weitere DBs / Projekte
  - Deaktivieren der Quellsystemanbindung
  - Ausführung des kompletten ETL-Prozesses inkl. Cube-Verarbeitung



4 Datentest

- Einige Fehler brauchen Daten
  - Irgendwo zu viele Daten gefiltert
  - Veränderung der Geschäftslogik

- Umsetzung
  - Samplen von Testdaten aus bestehender Quellsystem-Extraktion
  - Testdaten als Post-Deployment-Skript
  - Validieren von Testfällen nach der Ausführung
  - NBi (vgl. [NBi - Testing framework for Business Intelligence]) integriert sich in Azure DevOps



4 Testen mit NBi

- Seit NBi 1.24 können Connection String Informationen in Umgebungsvariablen übergeben werden
- Die Ergebnisse integrieren sich in Azure DevOps

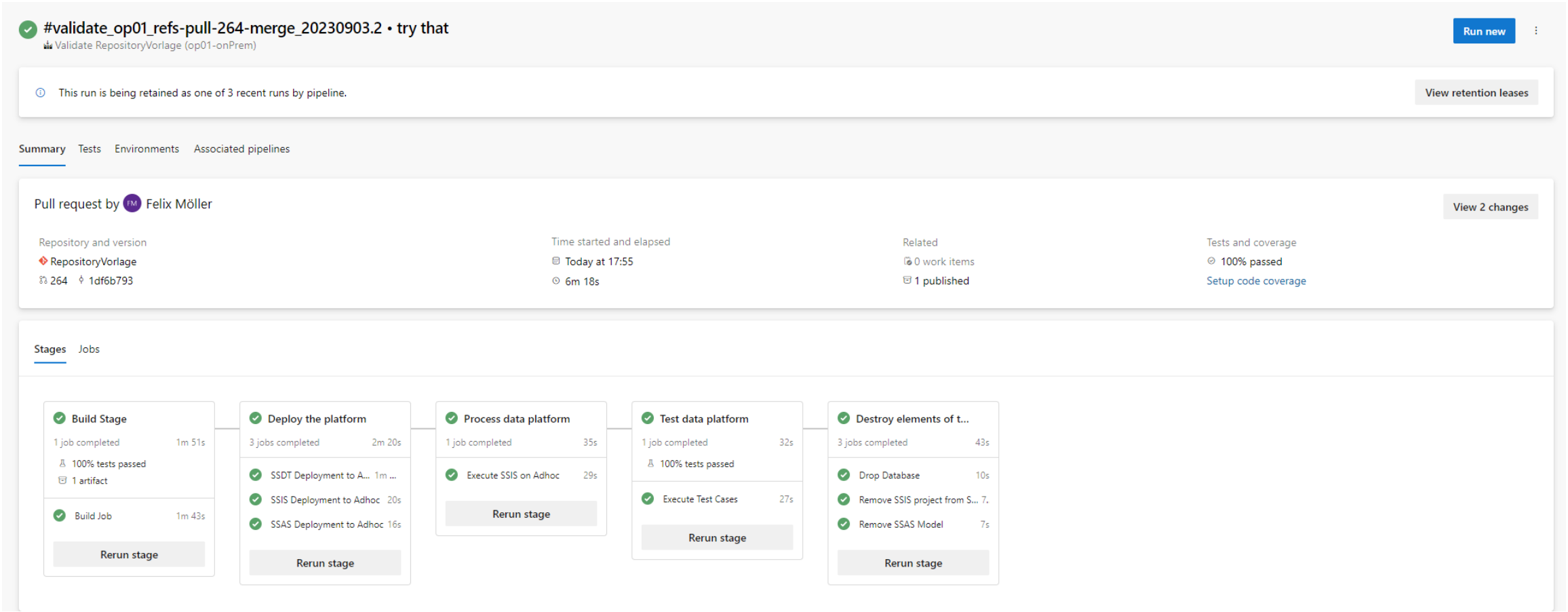
```
- task: PowerShell@2
  displayName: "Download NBi.Framework"
  inputs:
    targetType: 'inline'
    script: |
      $ProgressPreference = 'SilentlyContinue'
      [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12
      $amo = "https://github.com/Seddryck/NBi/releases/download/v1.24/NBi-Framework-1.24.0.zip"
      $amopath = Join-Path $env:TEMP "NBi.Framework.zip"
      Invoke-WebRequest -Uri $amo -OutFile $amopath
      Expand-Archive $amopath -Force -DestinationPath "$(Agent.BuildDirectory)\mybuild\arti1\tests\Framework"
- task: AzurePowerShell@5
  displayName: 'Get Access Token (Azure)'
  inputs:
    azureSubscription: ${{ variables.azureserviceconnectionName }}
    azurePowerShellVersion: LatestVersion
    ScriptType: 'InlineScript'
    Inline: |
      $ato = $(Get-AzAccessToken -Resource "https://database.windows.net/").Token;
      Write-Host "##vso[task.setvariable variable=ato]$ato"
- task: PowerShell@2
  displayName: 'Run NBi test suite'
  inputs:
    targetType: 'inline'
    script: |
      $nunit = "$(Agent.BuildDirectory)\mybuild\arti1\tests\NBiTests.nunit"
      & "C:\Program Files (x86)\NUnit 2.7.1\bin\nunit-console.exe" $nunit
  env:
    DWHConnectionString: "Data Source=$(sqlServerFQDN);Initial
Catalog=$(databaseName);Provider=MSOLEDBSQL;Access Token=$(ato);"
- task: PublishTestResults@2
  displayName: "Publish NBi tests"
  condition: always()
  inputs:
    testResultsFormat: 'NUnit'
    testResultsFiles: '**/TestResult.xml'
    failTaskOnFailedTests: true
```

```
<?xml version="1.0" encoding="UTF-8"?>
<testSuite xmlns="http://NBi/TestSuite" name="NBi Test Suite">
  <settings>
    <reference name="DWH">
      <connection-string><environment name="DWHConnectionString"/></connection-
string>
    </reference>
  </settings>
  <test name="Validate Equality">
    <system-under-test>
      <result-set>
        <query connection-string="@DWH">SELECT 42 AS n</query>
      </result-set>
    </system-under-test>
    <assert>
      <equal-to>
        <result-set>
          <row>
            <cell>42</cell>
          </row>
        </result-set>
      </equal-to>
    </assert>
  </test>
</testSuite>
```

Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

Status der Pipeline



Zusammenfassung

- **Branching** ist essentiell für Zusammenarbeit
- Pre-commit erlaubt Zusammenarbeit mit möglichst wenig Konflikten
- Alle Komponenten ADF / SSIS / SSDT / SSAS erlauben automatisches **Build** und **Deployment**
- Alles gezeigte funktioniert in der Cloud und onPrem
  - Build Agents: **Microsoft-hosted** vs **Self-Hosted Build Agents**
  - Zielumgebung: **onPrem SQL Server** vs. **Azure PaaS Services**
- Testen erlaubt sowohl Qualität im Projekt aufrechtzuerhalten als auch Regressionen zu vermeiden.
- Eine automatische Validation Pipeline findet fast alle technischen Fehler automatisch

Agenda

1. Überblick und Zielsetzung
2. Zusammenarbeit mit Git
3. Build und Deploy
   - 3.1 Integration Services
   - 3.2 SQL Server Database Projects
   - 3.3 Tabular Models
   - 3.4 Azure Data Factory
4. Testing
5. Zusammenfassung
6. Fragen / Diskussion

Start Building your pipline today!

Thank you very much for your attention.

Vielen Dank für Eure Aufmerksamkeit.