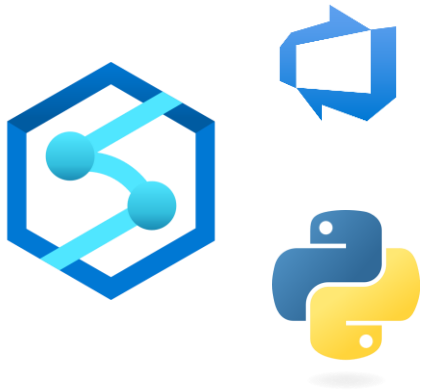
Motion10
Maak(t) uw digitale innovatie succesvol

CICD Synapse Serverless

Incremental deployments with Python and Pandas



Sponsors

Microsoft

POWER BI SENTINEL™
Governance, Disaster Recovery and Auditing for Power BI

dbWatch

redgate

Coeo

ADVANCING ANALYTICS

sqlbits

pyramid

Octopus Deploy

XTEN
CLOUD | DATA | XOPS

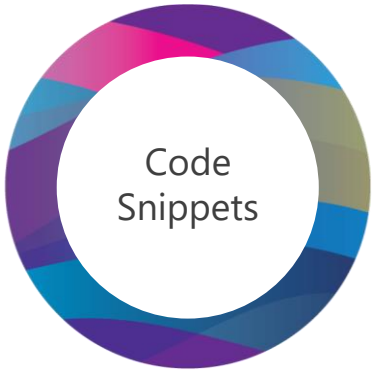
Simpson Associates
The Data Analytics Company

Thank you!
We couldn't do it without you.

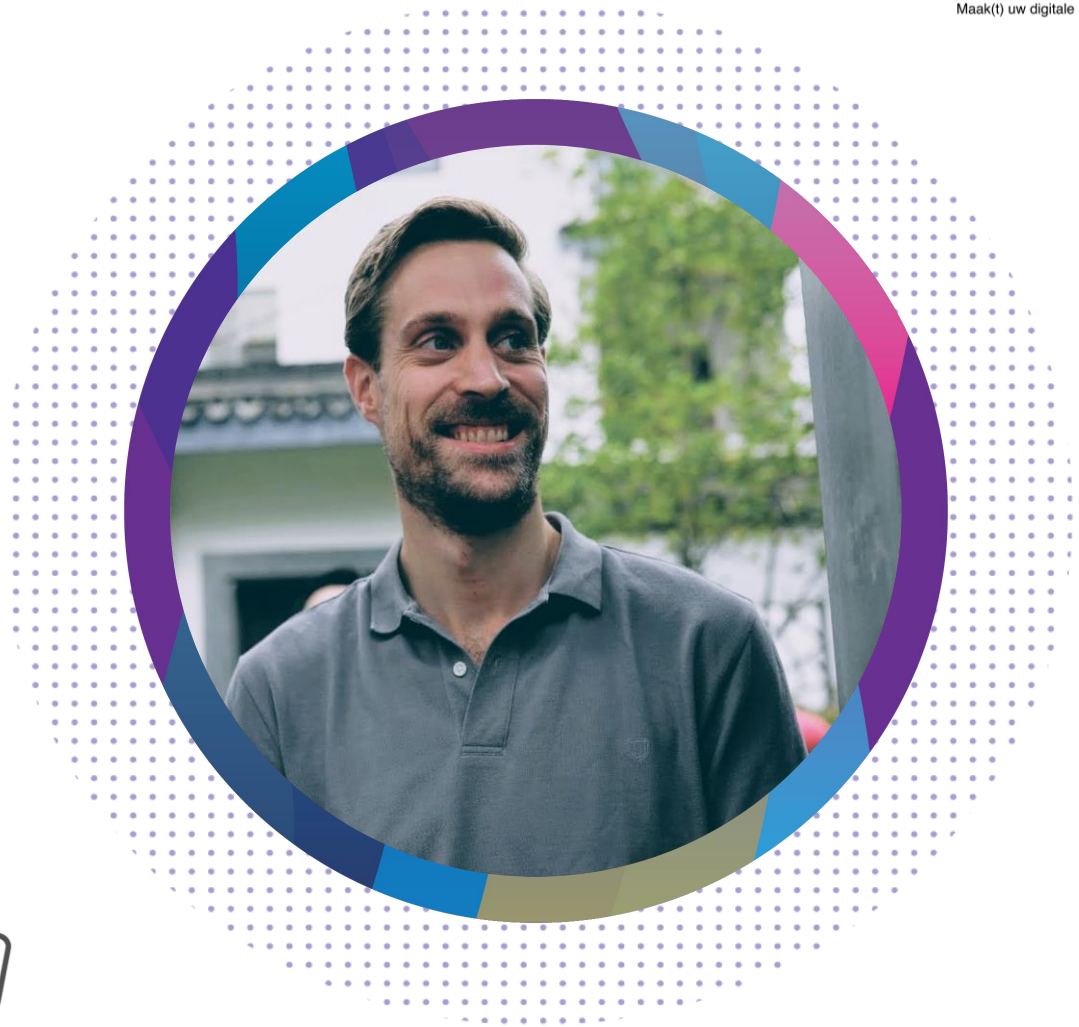
DATA RELAY

Table of Contents

DATA RELAY



Introductions



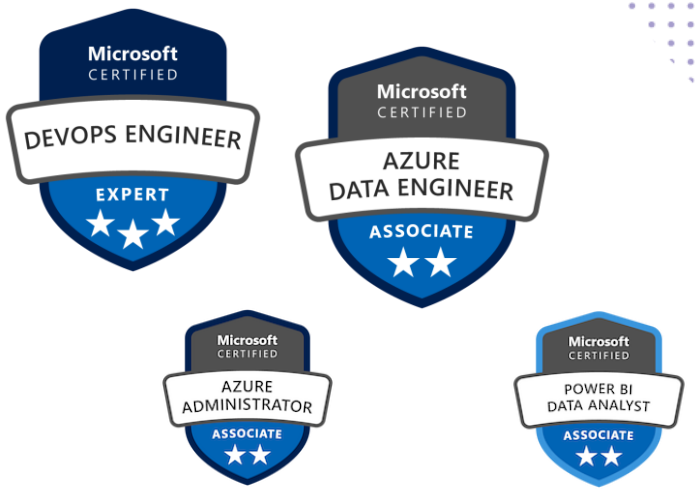
Arthur Steijn

Consultant Data & Analytics at Motion10
(Rotterdam, the Netherlands)

- Started with SSRS somewhere 2014
- Data, Cloud & DevOps Engineer
- Main focus on DevOps for 'BI'

- Father of two
- CrossFit

Blog: **sidequests.blog**



General assumptions of (some) knowledge on

DATA RELAY

General DevOps practices like

- Source Control
- CICD; Continuous Integration, Continuous Deployment

Azure DevOps as a tool for;

- Azure Repos
- Azure Pipelines

Azure Synapse Analytics;

- Source Control for the Synapse Workspace
- Serverless SQL Pool

Sounds nice, but why Serverless SQL Pools

DATA RELAY

- Basic discovery and exploration

- The Logical Data Warehouse (LDW)

- Data Transformation

Sounds nice, but why Serverless SQL Pools

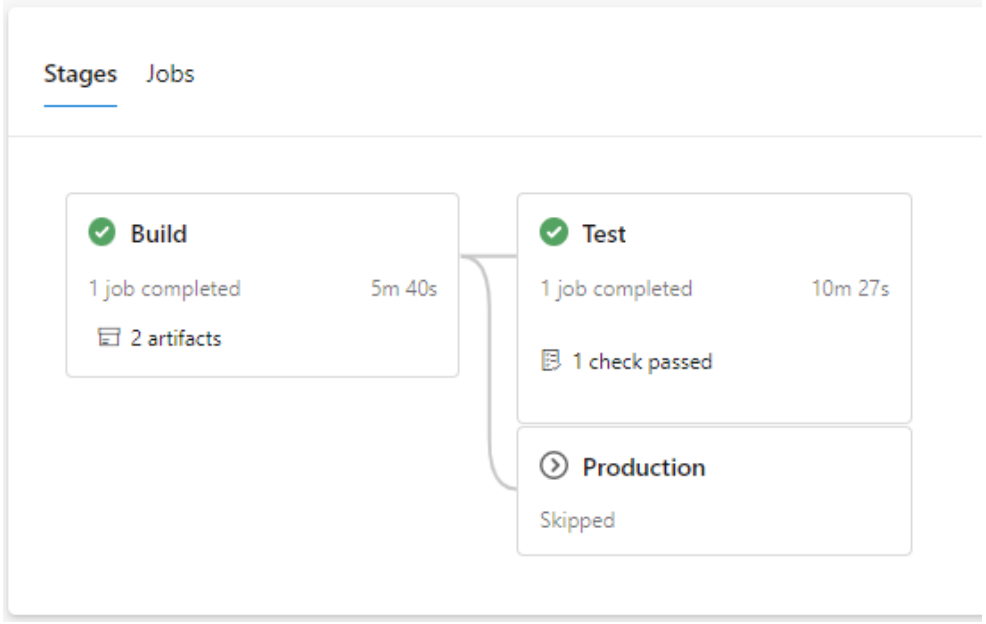
DATA RELAY

- The Logical Data Warehouse (LDW)
  - Separation of storage and compute!
  - Less ETL, hence it is used directly on the Data Lake
  - Works with familiar tools like;
    - SQL Server Management Studio
    - Azure Data Studio
    - Power BI connects like any other SQL source
  - T-SQL Support
    - Databases
    - Schemas
    - Views
    - Logins and Users

Can you guess what comes next?

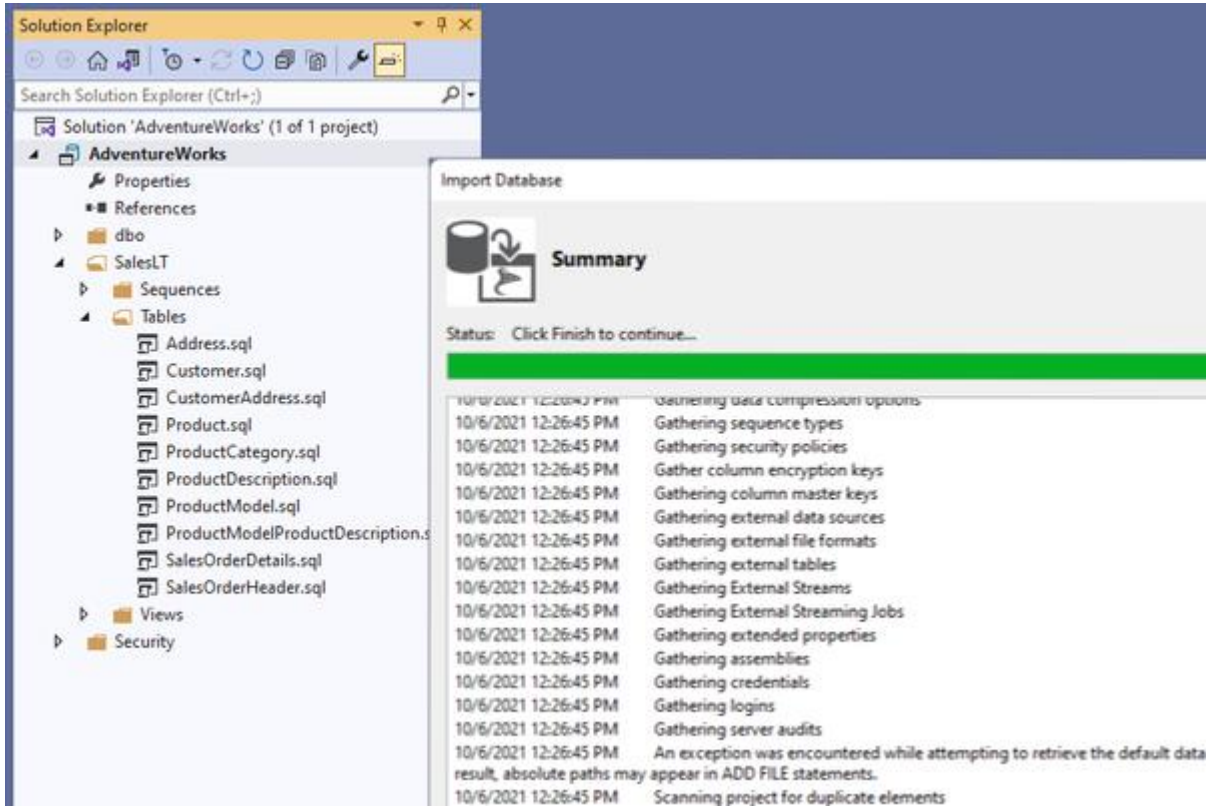
- Separate Environments

- Meaning Source Control and automated Deployments



Why and How

- Familiar with Visual Studio SQL Server Database Projects?
- We need a custom solution for source control and deployment
- What about the Synapse Workspace for source control?
- But why do you need Incremental Deployments for Serverless SQL? **



**If you think you do, think again at the end of this session;)

What makes this even more problematic

DATA RELAY

- Dependencies on other SQL objects like
  - External tables, views, functions, stored procedures

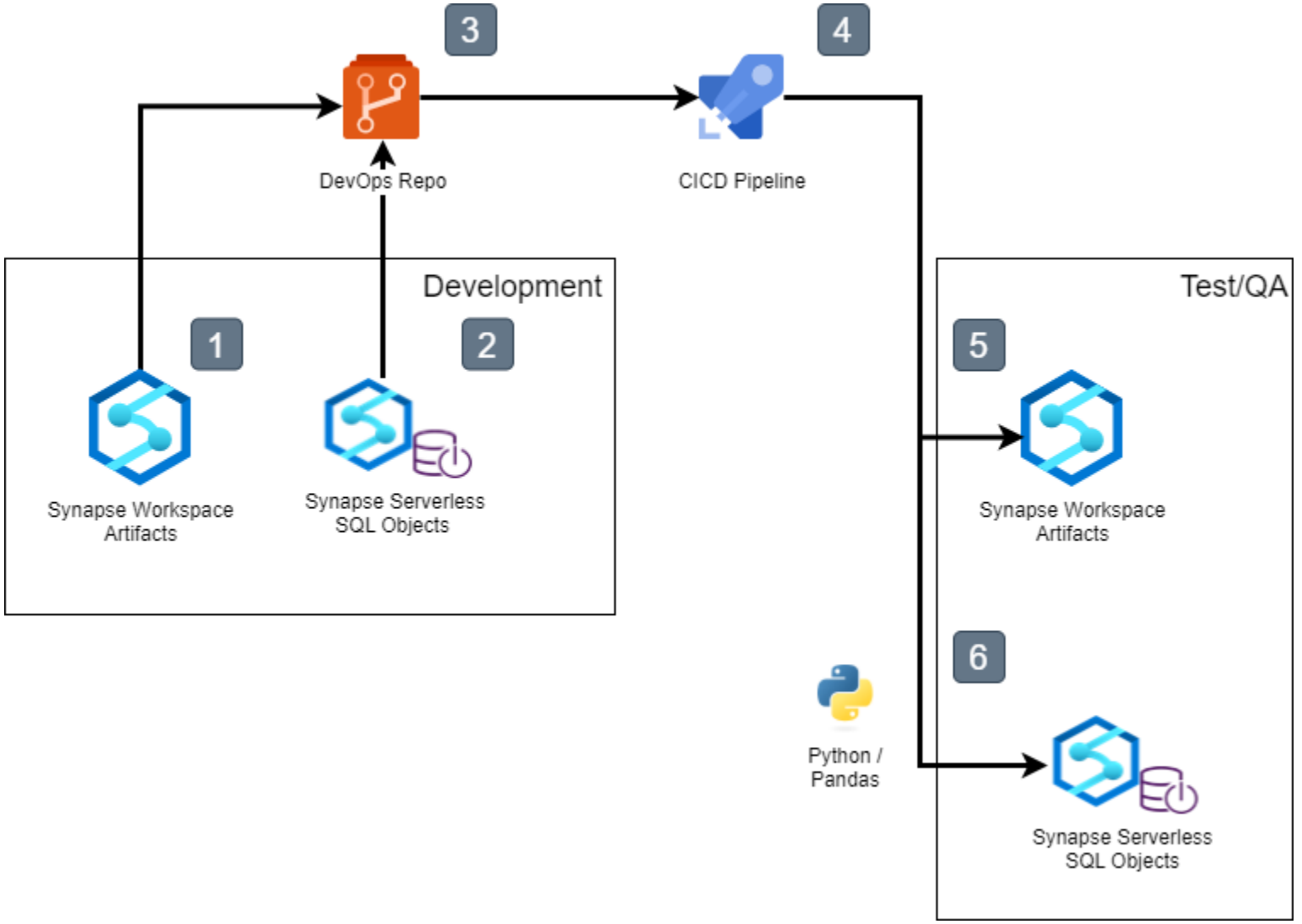
- Dependencies on files in the data lake

  data lake files
  - specific data lake folder paths that change over time

```
SELECT * FROM OPENROWSET(
BULK '/folder/yyyy/MM//dd/*-FULL.parquet',
DATA_SOURCE='storage', --> Root URL is in LOCATION of DATA SOURCE
FORMAT = 'PARQUET') AS [file]
```

- Environment specific references

High Level Architecture - Synapse Deployment

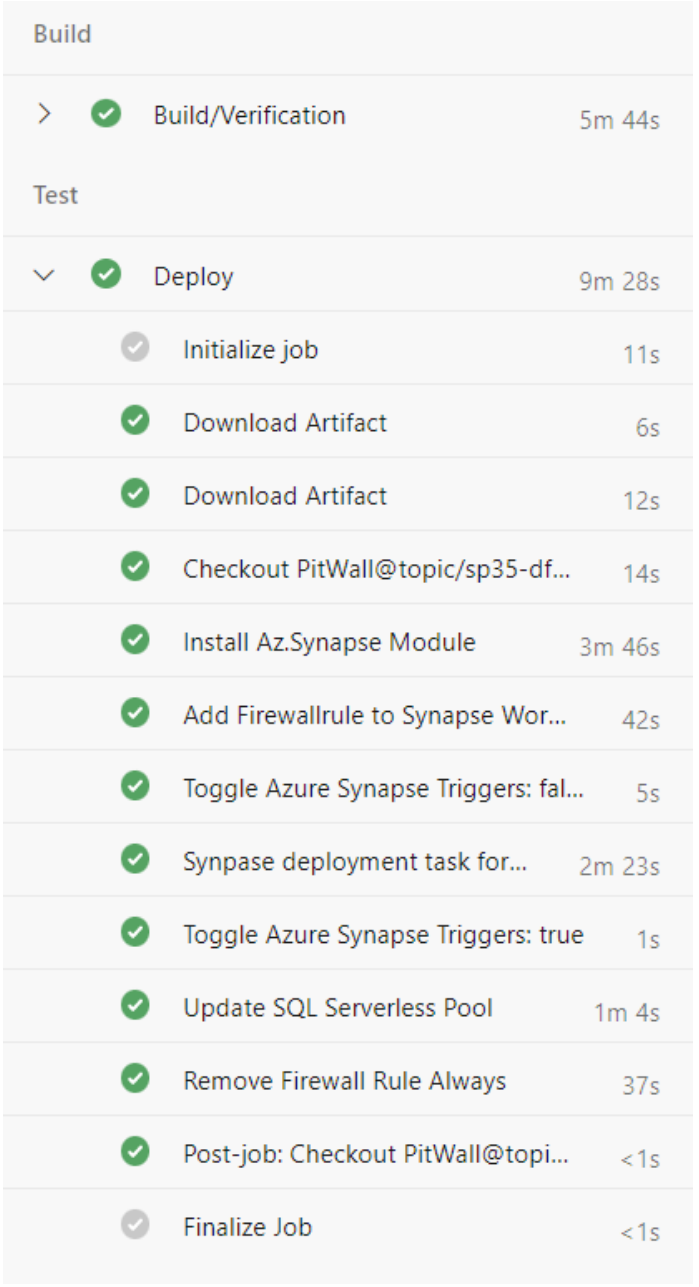


Complete Synapse Pipeline Overview

- Build stage to create needed artifacts and Validation of Synapse workspace object

Task name: *AzureSynapseWorkspace.synapsecicd-deploy.synapse-deploy.Synapse workspace deployment@2*

- Deploy stage steps:
  - Check out the artifacts and repo
  - Install Az.Synapse PowerShell module
  - Add (and remove) a firewall rule for the DevOps Agent
  - Toggle the Triggers off and on again
  - Update the Serverless SQL Pool



Source Control Options

The Synapse Workspace UI SQL Scripts

- Good: if your ws is connected to a repo, you can save directly to your working branch

- Bad: sql scripts are deployed with the Synapse Workspace Deplotment task, but not executed on the Serverless SQL Pool, this is confusing for some
- Bad: the more scripts you have the longer your workspace deployment will take (!!!)
- Bad: SQL Scripts are stored as JSON files and not easily readable in the repository

Directly in Git via Visual Studio Code (or with a similar tool)

- Good: you can save files as .sql and read them in the repo

- Bad: you need to manually add them to your repo (or write an automation script ☺ like I did)
- Bad: you still need to execute the scripts manually on the other environments (or)

Source Control for Serverless SQL Pools

DATA RELAY

1. Initialize database script
2. .sql files for views, stored procedures, functions

- .sql files are exported with the help of a Python script
  - Create a connection with PYODBC
  - Get all databases and objects
  - Store them in your local repository folder

- Git Commit on your working branch

```python
# Run this script to export all database objects from the synapse serverless sql server
import sys
sys.path.insert(0, "scripts\syn-serverless-export\functions\functions.py")
from functions.functions import *

server = 'synapse-server-ondemand.sql.azuresynapse.net'

def main():
    """ Function to connect to an azure synapse ondemand server and exports
    all views, stored procedures and functions to .sql files for all databases """

    # get an azure ad user and repo directory
    username, output_directory = get_personal_settings()

    print('  sql server that is configured is: ' + server)
    print('  username is set to: '+ username)
    print('  output_directory is set to: '+ output_directory)

    # creata a cursor for the connection
    cursor, cnxn =  dbcnxn(server, username)

    # get all databases from the server
    databases = get_databases(cursor)

    # for each database export to sql files
    for database in databases:
        # clean export directory so that deleted objects are removed
        clean_dir(output_directory+'/'+database)
        # export all views, functions and stored procedures
        export_views_to_dotsql(cursor, output_directory, database)
        export_sp_to_dotsql(cursor, output_directory, database)
        export_fn_to_dotsql(cursor, output_directory, database)

    # close connection
    cnxn.commit()
    cursor.close()

    print("All done")

if __name__ == "__main__":
    main()
```

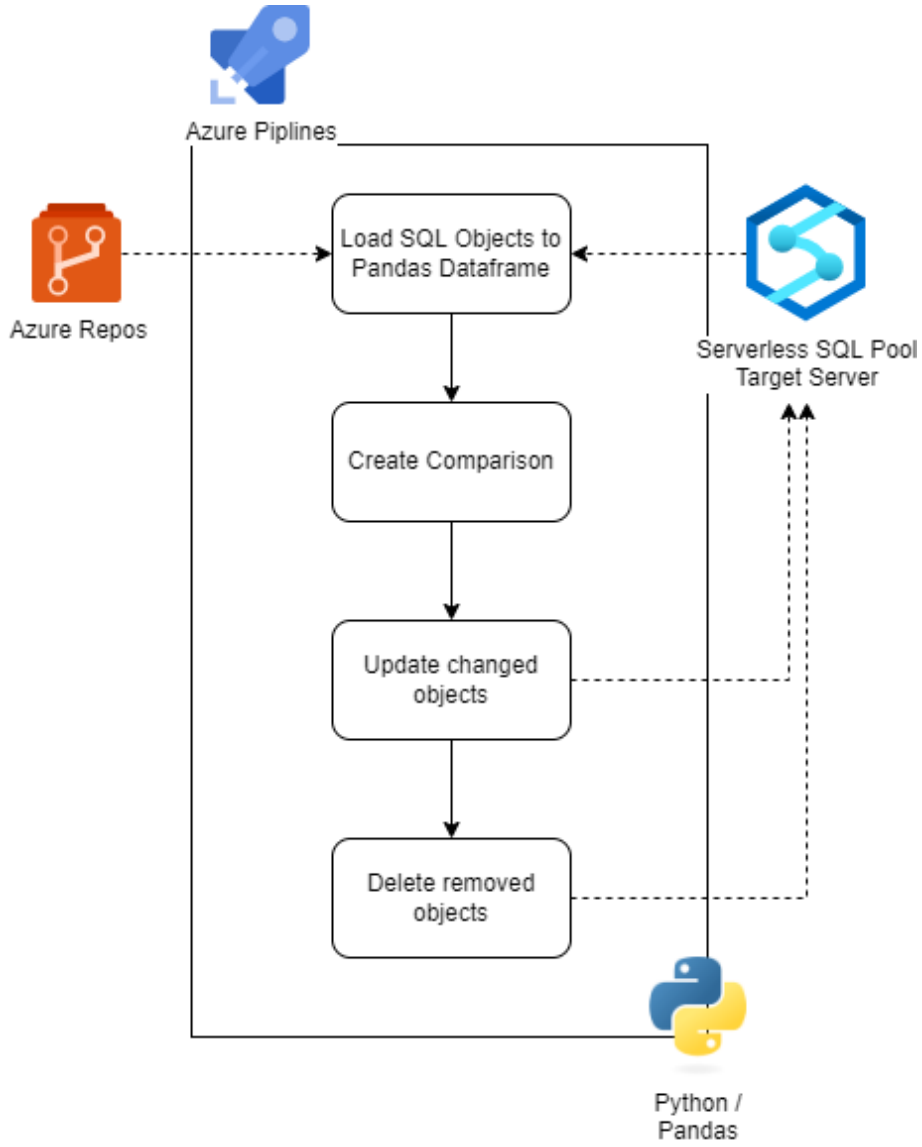
Simple manual deployment

- After database initialization
- Execute all objects from your Repository
- Optional; replace environment specific variables
- Optional; clean your sql statements (at previous source control step)
- Optional; '*Create or Alter …*'

```
# Run this script to execute all database objects on the synapse serverless sql server
import sys
sys.path.insert(0, r"scripts\syn-serverless-export\functions\functions.py")
from functions.functions import *

server = 'target-synapse-server-ondemand.sql.azuresynapse.net'

dataverse_db_name_d = 'development-server'
dataverse_db_name_p = 'production-server'

def main():
    """ Function to connect to an azure synapse ondemand server
        and execute all views, stored procedures and functions on their databases """
    #authenticate to the server with azure ad account
    username, repo_path = get_personal_settings()
    print('Creata a connection and cursor to the synapse server')
    cursor, cnxn =  dbcnxn(server, username)

    #get all the databases in the repository
    databases = os.listdir(repo_path)

    for database in databases:
        for sqlobject in os.listdir(os.path.join(repo_path, database)):
            sqlObjectFile = open(os.path.join(repo_path, database, sqlobject))
            sqlStatementString  = sqlObjectFile.read()
            print('Executing for: '+ database + '.' + sqlobject)

            # replace dataverse database reference
            if 'dp-p'in server:
                sqlStatementString = re.sub(dataverse_db_name_d, dataverse_db_name_p, sqlStatementString)

            #execute the sql statement against the server
            execute_statement(cursor, database, sqlStatementString)

    # close connection
    cnxn.commit()
    cursor.close()

if __name__ == "__main__":
    main()
```

Incremental deployment with Pandas

- Executed from an Azure DevOps agent
- All objects from the Repo are loaded into a Pandas dataframe
- All objects from the Target server are loaded into another dataframe
- The two dataframes are merged and compared on added / updated / removed objects
- The statements are executed



Create Dataframe from repository

DATA RELAY

```
def create_df_from_repo(repo_path):
    # store all the file names in a list
    filelist = []

    for root, dirs, files in os.walk(path):
        for file in files:
        #append the file name to the list
            filelist.append(os.path.join(root,file))

    #initiate pandas dataframe from filelist
    df_repo = pd.DataFrame({'file':filelist})

    # get file contents
    for index, file in df_repo.iterrows():
        f = open(file['file'], "r")
        df_repo.at[index, 'ViewDefinition'] = f.read()
        f.close()

    # clean sql statement
    df_repo['ViewDefinition'] = df_repo['ViewDefinition'].transform(clean_sql_func_df)

    # add database name
    df_repo['DBName'] = df_repo['file'].str.replace(r'.*databasescriptsexports\\', '', regex=True, flags=re.I)
    df_repo['DBName'] = df_repo['DBName'].str.replace(r'\\.*', '', regex=True, flags=re.I)
    # add schema name
    df_repo['SchemaName'] = df_repo['file'].str.replace(r'.*databasescriptsexports\\.*\\' , '', regex=True, flags=re.I)
    df_repo['SchemaName'] = df_repo['SchemaName'].str.replace(r'\..*' , '', regex=True, flags=re.I)
    # add object name
    df_repo['ObjectName'] = df_repo['file'].str.replace(r'.*databasescriptsexports\\.*\\' , '', regex=True, flags=re.I)
    df_repo['ObjectName'] = df_repo['ObjectName'].apply(lambda st: st[st.find(".")+1:st.find(".sql")])

    df_repo['Env'] = "repo"

    df_repo = df_repo.query('DBName != “Demo-Sqldb"')

    return df_repo
```

Create dataframe from Serverless SQL Pool Object

- A lot easier than the previous step

```
def create_target_df(sql, cnxn, cursor, database, objecttype):

    cursor.execute("USE [" + database + "];")

    df = pd.read_sql(sql, cnxn)
    df['objectType'] = objecttype

    return  df
```

- Joining the two dataframes together and compare the object definition

```
df_merged = df_repo.merge(df_target, how='outer’, on=['DBName', 'SchemaName', 'ObjectName’])

df_merged['compare_sql'] = ( df_merged['ViewDefinition_x'] == df_merged['ViewDefinition_y'] )
```

Finaly; Run the solution with the Azure PowerShell task

DATA RELAY

```
#This script is used to deploy the SQL scripts that are saved in the REPO to the SQL Serverless Database
param(
   [string][parameter(Mandatory = $true)] $ServerInstance
)

#Import needed modules
Write-Host("Importing modules")
Import-Module SQLServer
Import-Module Az.Accounts -MinimumVersion 2.2.0

#Get an access token with the Service Pricipal used in the Azure DevOps Pipeline
Write-Host("Get Access Token")
$access_token = (Get-AzAccessToken -ResourceUrl https://database.windows.net).Token

#Install Python Modules
Write-Host("Installing Python Modules")
python -m pip install pyodbc
python -m pip install pandas

#Start Python Script
Write-Host("Starting python script")
& python.exe 'scripts/database-scripts/deploy-database-changes/compare-and-deploy-database-changes.py' -
synserver $ServerInstance -access_token $access_token
if ( $LASTEXITCODE -eq 1)
{
      Write-Error "Script Execution Failed"
}
```

Debugging the Solution

- Just
- Don't
- Give
- Up

Questions?

https://datarelay.co.uk/feedback