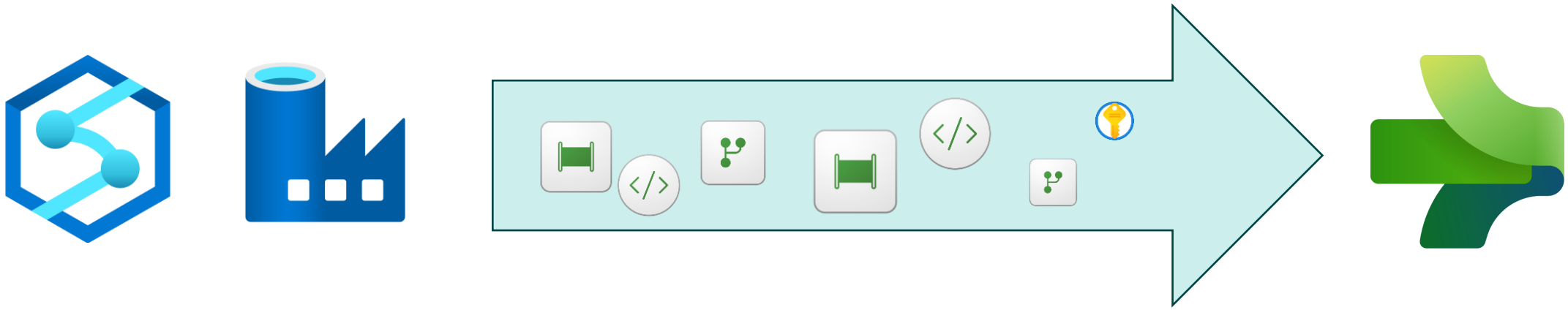
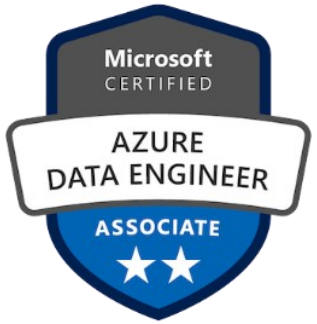
Migration Azure Data Factory / Synapse to Fabric

How to migrate successfully



Stefan Kirner

- > 20 years experience using Microsoft Data Platform
- Data Monster Chapter Lead Karlsruhe & Beirat
- Director Business Intelligence scieneers GmbH
- stefan.kirner@scieneers.de
- LinkedIn: https://www.linkedin.com/in/stefan-kirner/



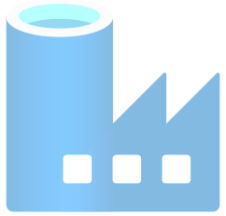
Agenda

- Intro – Data Factory from Azure to Fabric
- Why migrate Data Factory to Fabric
- How to migrate Data Factory to Fabric
- Changes for Application Lifecycle Management
- Summary

Intro –
Data Factory from Azure to Fabric

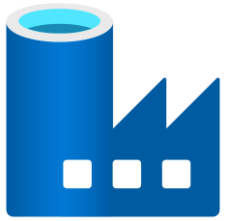
Evolution Azure Data Factory v1 to Fabric Data Factory

ADF v1



Data Ingestion

ADF v2



+ Transform
+ Integration Runtimes

Data Factory in Synapse

+ Spark Notebooks Integration

Data Factory In Fabric

+ SaaS
+ Data Platform Design Integration

Microsoft Fabric in a nutshell

scieneers
DRIVEN BY DATA

Technology

End-to-End-
Analyse- and
Dataplattform

- Data Engineering
- Data Science
- Datawarehouse
- Dashboarding

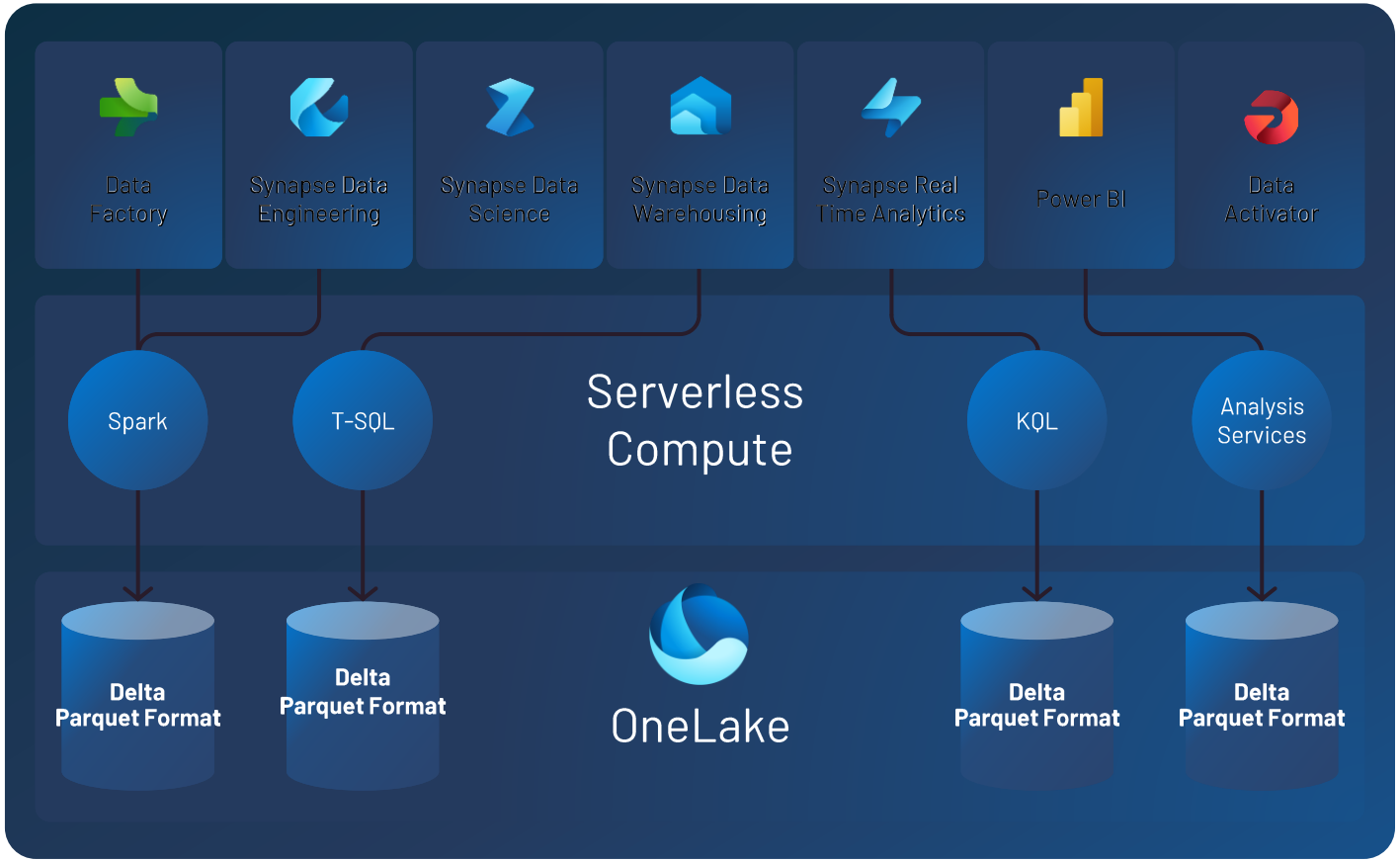
Business

Software as
a Service

- Monthly
Fixprice
- Savings by
Reservations
- Pausable
(off-times)
- Up/Downscale
anytime

Parts of Fabric and how they interact

- Unites any tooling for analytics in one GUI
- One storage for all different compute engines and tools
- Separation of compute and storage
- One central security model possible



Why
migrate Data Factory from Azure to Fabric

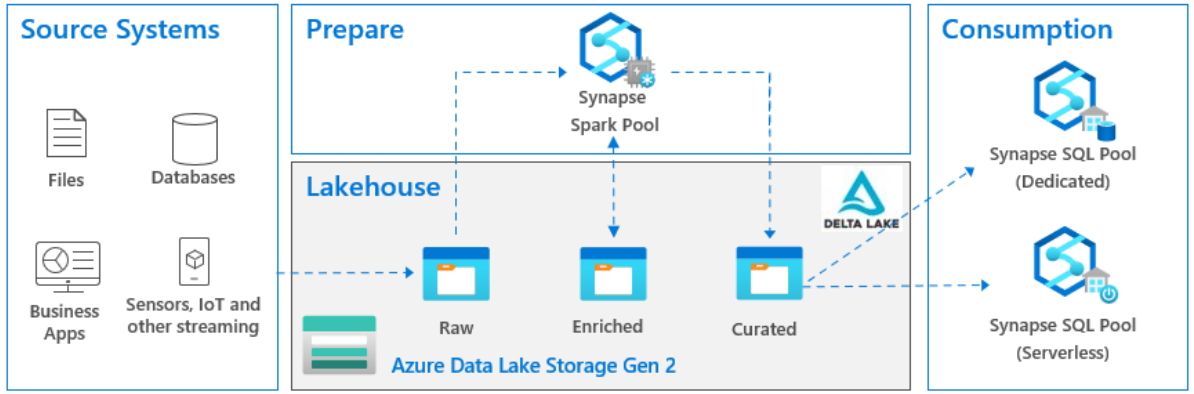
Possible benefits of migration from Azure to Fabric

- Software as a Service
  - Easier to manage regarding IT operations
- Simplicity – one for all
  - Enabling the departments to manage their data using a known GUI
  - Better integration of different functions to „data solution“
- Monthly Fixprice
  - Instead of many different independant services with different pricing models
- New Features
  - Will be first implemented in Fabric
  - OneLake simplifies security & optimizes load and read performance
  - Data Flows Gen2 for GUI based ETL
  - New activities like Outlook and Teams make your life easier

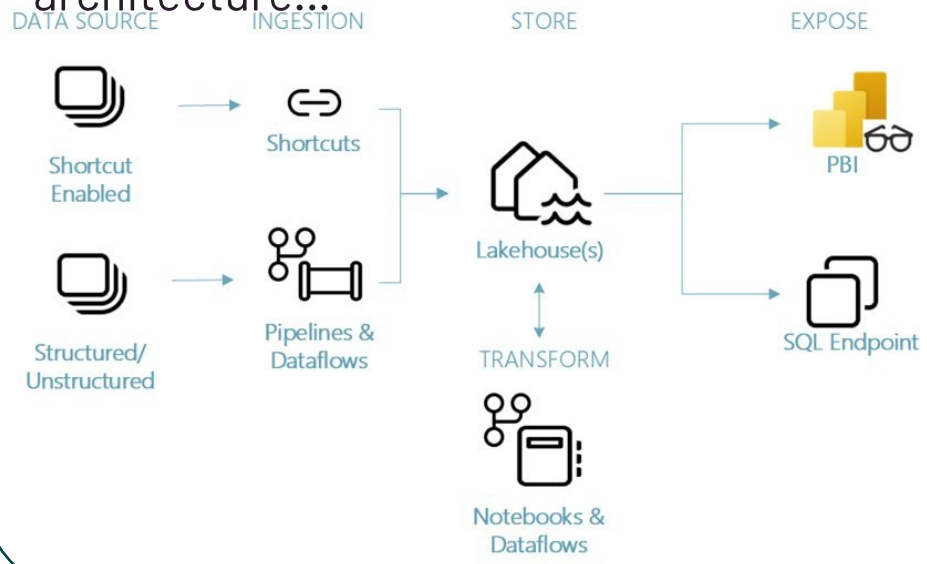
Good starting point is Microsofts official blogpost Fabric explained for existing Synapse users

- Explains conceptually which Fabric and Synapse services correspond to each other, which new services are offered and what the migration path could potentially look like
- No current plans to retire Azure Synapse Analytics (or Azure Data Factory)– continue with full support from Microsoft with no need to change anything

Moving from multi-services Azure PaaS architecture...



...to the consolidated Fabric SaaS architecture...



https://blog.fabric.microsoft.com/de-de/blog/microsoft-fabric-explained-for-existing-synapse-users/

How to migrate Data Factory from Azure to Fabric

- Connectivity
- Data Pipelines
- Dataflows
- Notebooks
- Triggers

Problem: Resources from MS and blogs scattered and incomplete, no one-stop solution

scieneers
DRIVEN BY DATA

- Official Fabric Migration Repository from Microsoft
  - Contains documentation as well as individual scripts that can be used to automate parts of a migration
- Migrate Synapse to Fabric Documentation
  - Contains migration paths for individual Synapse objects such as notebooks and Spark pools
- Microsofts Migration Assistant currently in private-preview
  - And currently focused mainly on T-SQL artefacts on Synapse serverless Pool
- dataroots.io: Migration from Synapse Serverless zu Microsoft Fabric
  - describes the basics of the Polaris Lakehouse Engine, which is used for both Synapse Serverless and the Fabric Lakehouse
- thatbluecloud.com: Moving from Synapse to Fabric
  - describes a step-by-step approach in which Synapse and Fabric exist and are used in parallel

Artefacts not only renamed in Fabric – this will need some effort to migrate

Azure Data Factory / Synapse	Data Factory in Fabric
Pipeline	Data Pipeline
Publish	Run, Save
Linked Service	Connection
Dataset	N.A.
Self-hosted Integration Runtime (SHIR)	On-premises Data Gateway

https://learn.microsoft.com/en-us/fabric/data-factory/compare-fabric-data-factory-and-azure-data-factory

Connectivity solved different - Linked Services on Azure vs. Fabric Connections

- Linked Services
  - Defined per instance of ADF / Synapse
  - Datasets as subitem for further differentation (tables, rest endpoints…)

Recreate manually

- „Power BI“ Connections used
  - Defined on Power BI Tenant Level
  - Shareable on user / group level
  - No datasets objects
  - Privacy levels definable
- governance concept especially for self-service adaption recommended

https://learn.microsoft.com/en-us/fabric/data-factory/migrate-planning-azure-data-factory#linked-services

Connectivity to your **on-premises** source data

Self-hosted Integration Runtime (SHIR)

- Client Agent which establishes encrypted connectivity between on-prem and Azure services
- Used by ADF & Synapse Analytics

On-premises Data Gateway

- Client Agent which establishes encrypted connectivity between on-prem and Microsoft cloud services
- Used by Power BI, Power Apps, Logic Apps, Azure Analysis Services, Power Automate & Fabric

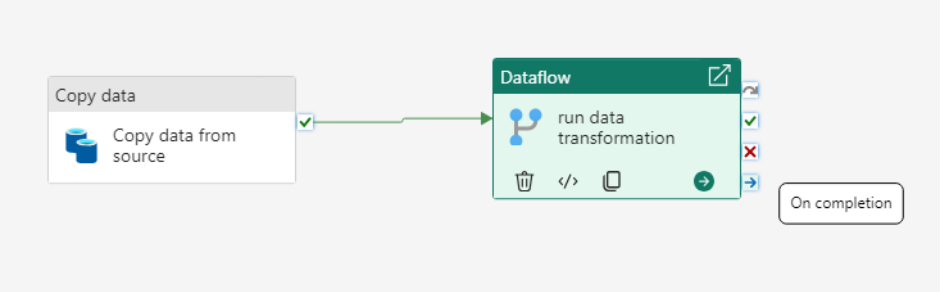
Impact using SHIR vs Data Gateway, just plug & play?

scieneers
DRIVEN BY DATA

- Data Factory used for Ingestion / Copying source data in Fabric
- Any connectivity provider for Fabric DF had to be adapted to use On-premises Data Gateway
- **Not all** connectivity options supported in ADF are also available for Fabric Copy Activity at this point in time, but most for Data Flows Gen2
- Even more confusion: Fabric Pipelines and Data Flows Gen2 each support different sources
- Make sure that your data source is really supported
- Workaround: additional ADF / Synapse Links etc

Pipelines structure data loading activities by running order and constraints

- **Continues** to live in both versions
- **Orchestration** and triggering of ETL/ELT processes
- Differences regardings **activities** see mapping below
- Differences in **support of connectivity**
- Technically, pipeline definitions are saved as a **.json file** in both ADF / Synapse and Fabric
- Microsoft does not yet have a defined migration or upgrade process (see screenshot).
- Interim migration solution could be to run Data Factory pipelines from Fabric (ADF only)



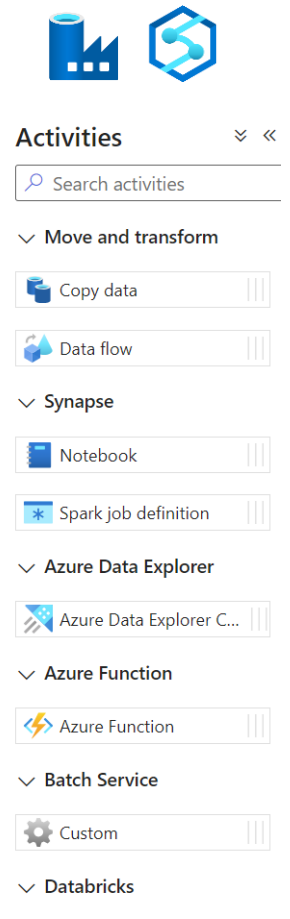
As part of the Data Factory in Microsoft Fabric roadmap, we're working towards the preview of the following by Q2 CY2024:

- **Upgrade from ADF pipelines to Fabric** - We're working with customers and the community to learn how we can best support upgrades of data pipelines from ADF to Fabric. As part of this, we deliver an upgrade experience that empowers you to test your existing data pipelines in Fabric using mounting and upgrading the data pipelines.

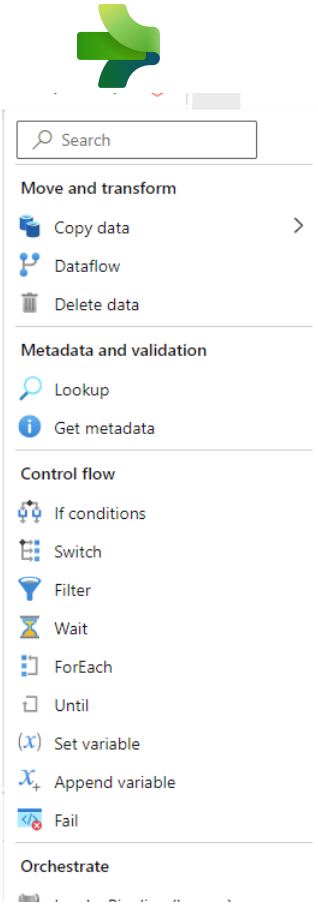
https://learn.microsoft.com/en-us/fabric/data-factory/compare-fabric-data-factory-and-azure-data-factory

Data Pipeline Activities continuity between ADF and Fabric version

- 90% of activities in ADF also supported in Fabric
- Not supported pipeline activities in Fabric version:
  - Validation (workaround)
  - Run **SSIS** package
  - Run **Mapping Data Flow**
  - **HD Insights**
- Different
  - Exec Databricks Notebooks supported in Fabric
    - Missing options for Python / JAR
    - Spark Jobs available as separate component



vs



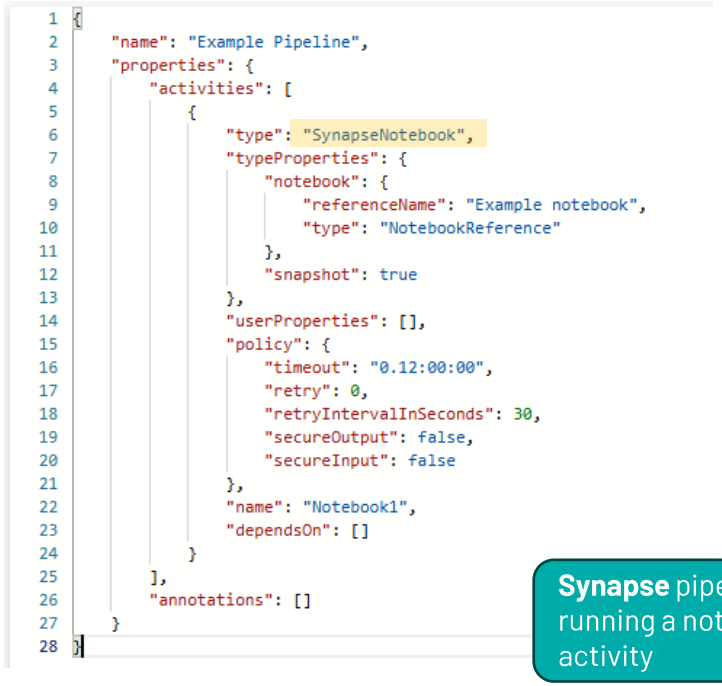
https://learn.microsoft.com/en-us/fabric/data-factory/activity-parity

What about other integration runtimes supported in Synapse / ADF?

- No concept „integration runtimes“ in Fabric
- ADF SSIS integration runtimes?
  - Not supported yet and not on the roadmap for Fabric
- ADF Runtime for airflow?
  - Support in Fabric in preview as „ Apache Airflow Job“

Differences in data pipelines – a simple example

scieneers
DRIVEN BY DATA



```
{
    "name": "Example Pipeline",
    "properties": {
        "activities": [
            {
                "type": "SynapseNotebook",
                "typeProperties": {
                    "notebook": {
                        "referenceName": "Example notebook",
                        "type": "NotebookReference"
                    },
                    "snapshot": true
                },
                "userProperties": [],
                "policy": {
                    "timeout": "0.12:00:00",
                    "retry": 0,
                    "retryIntervalInSeconds": 30,
                    "secureOutput": false,
                    "secureInput": false
                },
                "name": "Notebook1",
                "dependsOn": []
            }
        ],
        "annotations": []
    }
}
```

Synapse pipeline running a notebook activity

```
{
  "properties": {
    "activities": [
      {
        "type": "TridentNotebook",
        "typeProperties": {
          "notebookId": "40b17e35-7705-45ce-8bee-dccfdf8f55cf",
          "workspaceId": "00000000-0000-0000-0000-000000000000"
        },
        "policy": {
          "timeout": "0.12:00:00",
          "retry": 0,
          "retryIntervalInSeconds": 30,
          "secureInput": false,
          "secureOutput": false
        },
        "name": "Notebook1",
        "dependsOn": []
      }
    ]
  }
}
```

Fabric pipeline running a notebook activity

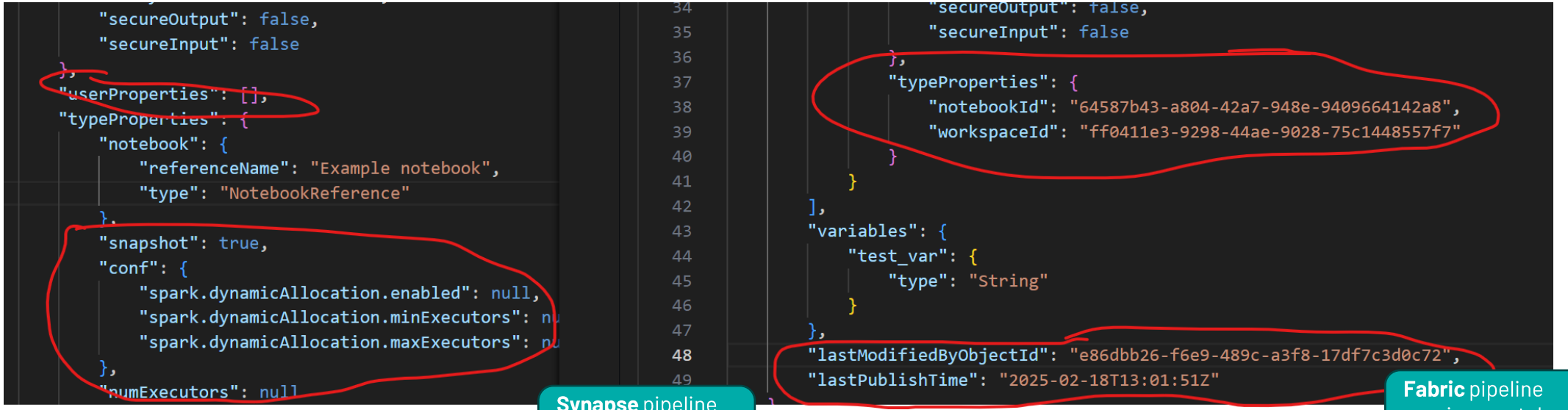
Copying the properties element from the Synapse pipeline to the properties element of the Fabric pipeline works limited . Steps that **do not rely on a linked service** or similar, such as...

- ...define *variables*
- ... *set Variables* steps
- ...*start Notebook* steps

are **adopted correctly**.

- Start Notebook step then naturally refers to a **non-existent notebook**.
- Since the **type = SynapseNotebook** is also set in synapse, a synapse notebook is explicitly expected
- This must first be **adjusted** in the code, then the correct **notebook can be selected** in the GUI.

Further differences in data pipelines activities



```
        "secureOutput": false,
        "secureInput": false
    },
    "userProperties": [],
    "typeProperties": {
        "notebook": {
            "referenceName": "Example notebook",
            "type": "NotebookReference"
        },
        "snapshot": true,
        "conf": {
            "spark.dynamicAllocation.enabled": null,
            "spark.dynamicAllocation.minExecutors": nu
            "spark.dynamicAllocation.maxExecutors": nu
        },
        "numExecutors": null
```

Synapse pipeline running a notebook activity

```
                "secureOutput": false,
                "secureInput": false
            },
            "typeProperties": {
                "notebookId": "64587b43-a804-42a7-948e-9409664142a8",
                "workspaceId": "ff0411e3-9298-44ae-9028-75c1448557f7"
            }
        }
    ],
    "variables": {
        "test_var": {
            "type": "String"
        }
    },
    "lastModifiedByObjectId": "e86dbb26-f6e9-489c-a3f8-17df7c3d0c72",
    "lastPublishTime": "2025-02-18T13:01:51Z"
```

Fabric pipeline running a notebook activity

- Differences in **properties** of the activities have to be considered for migration
- **Annotations** in canvas possible in Synapse

- Fabric using **GUIDs** for references – unknown in ADF / Synapse and cannot be updated in Fabrics JSON Editor which have to be considered
- Some **identifiers and metadata** automatically added by Fabric – not all when copy/pasted
- No **Annotations** in canvas possible in Fabric

Linked services vs connections

scieneers
DRIVEN BY DATA

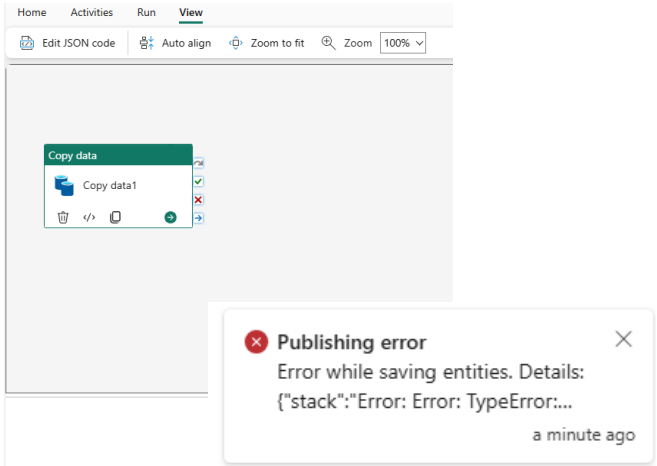
```json
{
    "name": "Zip_data_synapse",
    "properties": {
        "activities": [
            {
                "name": "Copy data1",
                "type": "Copy",
                "dependsOn": [],
                "policy": {
                    "timeout": "0.12:00:00",
                    "retry": 0,
                    "retryIntervalInSeconds": 30,
                    "secureOutput": false,
                    "secureInput": false
                },
                "userProperties": [],
                "typeProperties": {
                    "source": {
                        "type": "RestSource",
                        "httpRequestTimeout": "00:01:40",
                        "requestInterval": "00.00:00:00.010",
                        "requestMethod": "GET",
                        "paginationRules": {
                            "supportRFC5988": "true"
                        }
                    },
                    "sink": {
                        "type": "JsonSink",
                        "storeSettings": {
                            "type": "AzureBlobFSWriteSettings"
                        },
                        "formatSettings": {
                            "type": "JsonWriteSettings"
                        }
                    },
                    "enableStaging": false
                },
                "inputs": [
                    {
                        "referenceName": "RestResource1",
                        "type": "DatasetReference"
                    }
                ],
                "outputs": [
                    {
                        "referenceName": "Json1",
                        "type": "DatasetReference"
                    }
                ]
            }
        ],
        "annotations": []
    }
}
```

Synapse pipeline using linked service

sample pipeline referencing a public API using a linked service

```json
{
    "name": "Zip_data_fabric",
    "objectId": "3c73aab2-034c-4d17-8497-4b5b75d0da19",
    "properties": {
        "activities": [
            {
                "name": "Copy data1",
                "type": "Copy",
                "dependsOn": [],
                "policy": {
                    "timeout": "0.12:00:00",
                    "retry": 0,
                    "retryIntervalInSeconds": 30,
                    "secureOutput": false,
                    "secureInput": false
                },
                "typeProperties": {
                    "source": {
                        "type": "RestSource",
                        "httpRequestTimeout": "00:01:40",
                        "requestInterval": "00.00:00:00.010",
                        "requestMethod": "GET",
                        "paginationRules": {
                            "supportRFC5988": "true"
                        },
                        "datasetSettings": {
                            "annotations": [],
                            "type": "RestResource",
                            "typeProperties": {
                                "relativeUrl": "98121"
                            },
                            "schema": [],
                            "externalReferences": {
                                "connection": "10ead68a-f723-468d-a100-42f7583567d4"
                            }
                        }
                    },
                    "sink": {
                        "type": "JsonSink",
                        "storeSettings": {
                            "type": "AzureBlobFSWriteSettings"
                        },
                        "formatSettings": {
                            "type": "JsonWriteSettings"
                        },
                        "datasetSettings": {
                            "annotations": [],
                            "type": "Json",
                            "typeProperties": {
                                "location": {
                                    "type": "AzureBlobFSLocation",
                                    "folderPath": "zipfiles",
                                    "fileSystem": "test"
                                }
                            },
                            "schema": {},
                            "externalReferences": {
                                "connection": "1f02469d-05d6-48a1-a917-1d48c4085ff0"
                            }
                        }
                    },
                    "enableStaging": false
                }
            }
        ],
        "lastModifiedByObjectId": "3eb696d6-260c-4b95-af2f-cfaab27de692",
        "lastPublishTime": "2025-01-27T16:31:29Z"
    }
}
```

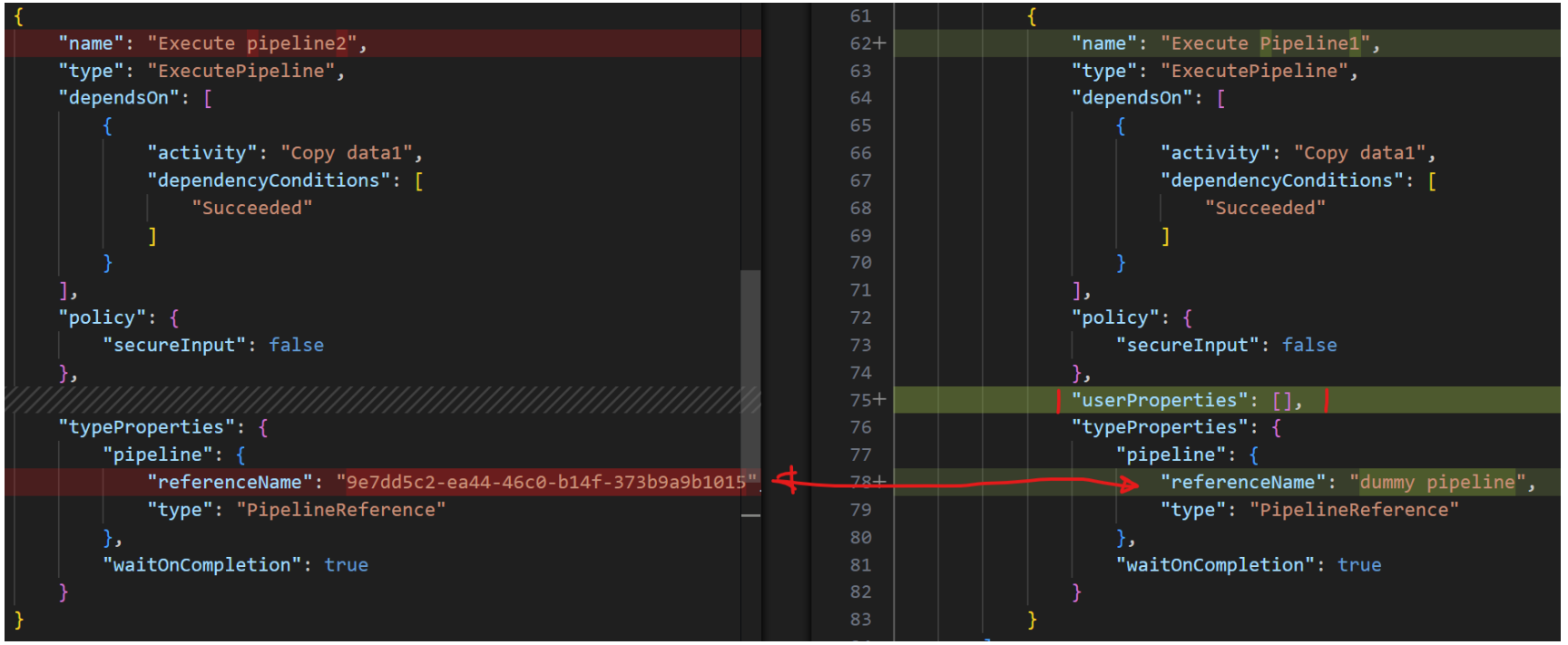
Fabric pipeline using connection



- reference to "**datasets**" implemented differently
- **No Input/outputs** in Fabric → delete to edit pipeline
- After that selection of **connections** manually possible

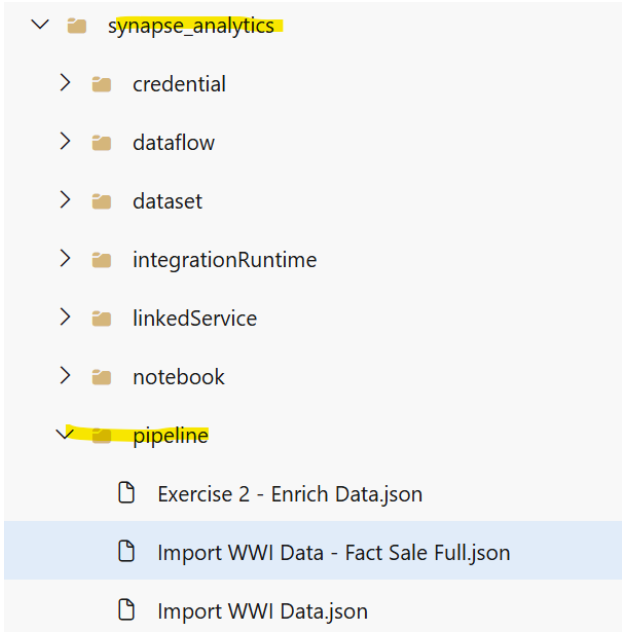
Orchestration elements: execute pipeline

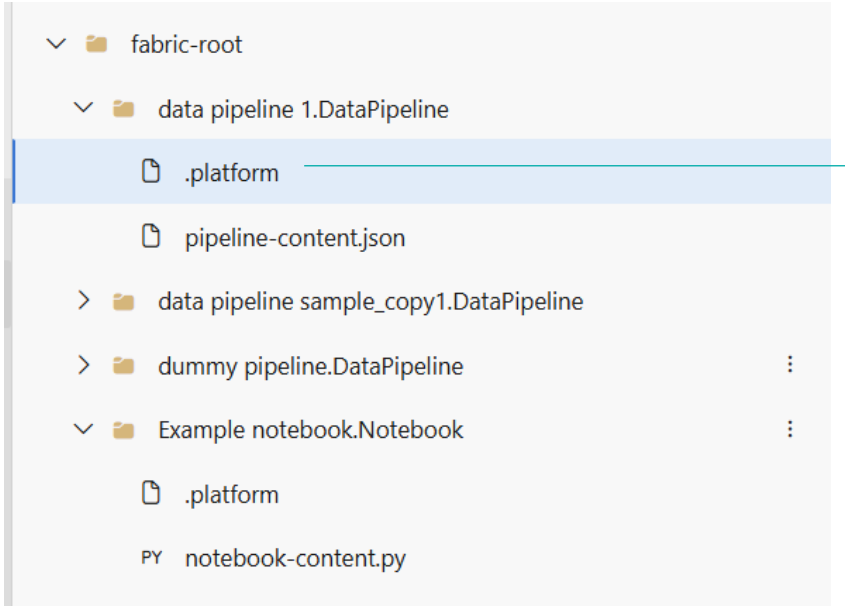
- 2 versions in Fabric
  - Legacy – run pipelines in same workspace
  - Newer in beta: run also pipelines in ADF / Synapse
- Structure is almost the same – good news from a migration perspective because this is often in use

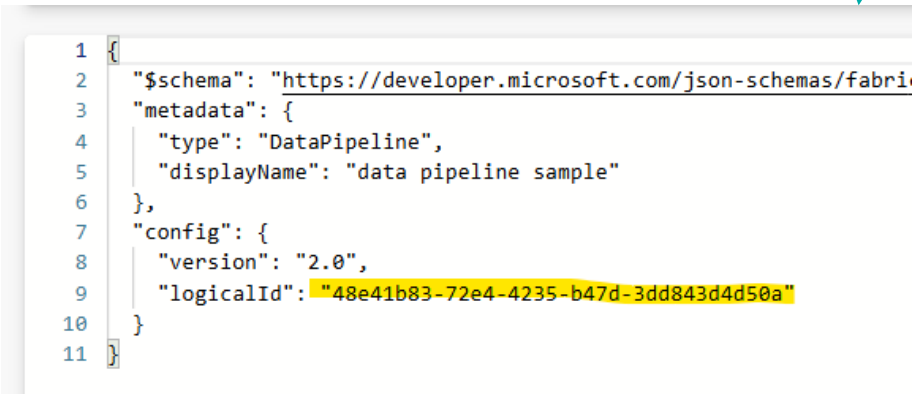


Good idea to push pipeline.json into repo?

- Could work, but be careful
- But some repo-metadata generated by Fabric
- Necessary to generate this items and the folder structure, too







From „Data Flows“ in ADF to „Dataflows Gen2“ in Fabric

- Mapping Data Flows
- Persist data in different sinks
- Spark based **scale out**
- Non-reusable **scala code** generated at runtime
- Pay by runtime, could rather be expensive

- Data Flows Gen2
- Persist data in different Fabric sinks
- No scale out, **scale up** due to In-memory engine
- Reusable **M-Code** as result
- Fixed price, pay by license

https://learn.microsoft.com/en-us/fabric/data-factory/migrate-planning-azure-data-factory#dataflows

Migration Tooling Mapping Data Flows to Python Notebooks

scieneers
DRIVEN BY DATA

- https://github.com/sethiaarun/mapping-data-flow-to-spark
- https://github.com/sethiaarun/mapping-dataflow-to-fabric-with-openai
- Github Repos from a MS CAT employee
- 2 approaches to convert MDF to Fabric Notebook:
  - Using Java and Python based script
  - Using Python & OpenAI
- But last commits 2 years ago
- Limited in supported items
- Rebuild from scratch using notebooks seems to be useful

Notebooks (Synapse)

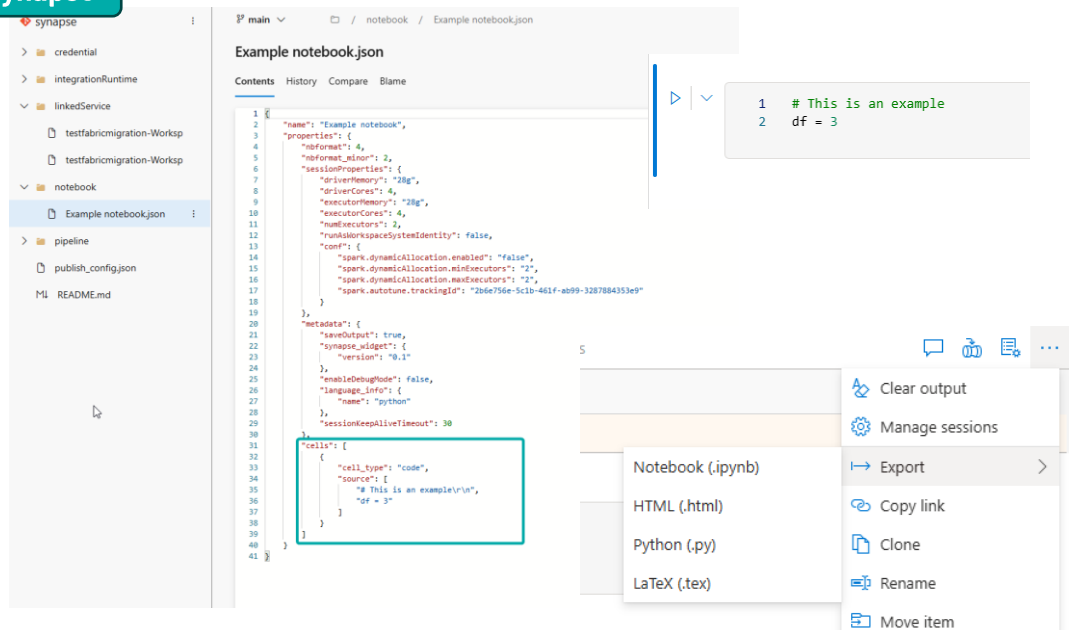
Notebooks in **Synapse**:

- Stored in the repo as **.json** including a lot of metadata
- Export to different formats manually using **GUI**
- Script based approaches could transform this to **.ipynb** file
- Create your own script based export automation using Azure APIs

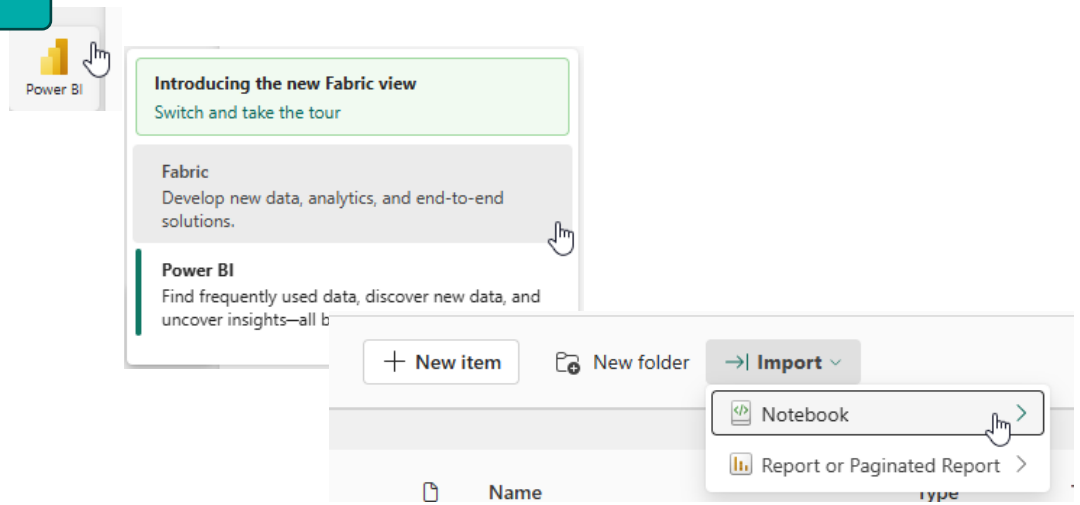
Notebooks in **Fabric**:

- Stored in Repo as **.py** inklusing a separate metadata file
- **Import notebooks in batch** from different formats (.ipynb, .py, .sql, .scala) in batch using **fabric-experience**
- **Import automation via APIs** supported
- **Unique notebook names by workspace** required (Fabric not aware of folders/paths

Synapse



Fabric



Notebooks – Migration support resources by Microsoft

scieneers
DRIVEN BY DATA

- Short documentation
  - for the manual export/import process
  - for a scripted approach via APIs
- Script for automation provided in git Repo
- Necessary **manual adjustment** of referenced data sources, target tables, library paths and referenced files and notebooks
- or develop your own **find-and-replace script**

Python Copy

```
# Azure config
azure_client_id = "<client_id>"
azure_tenant_id = "<tenant_id>"
azure_client_secret = "<client_secret>"

# Azure Synapse workspace config
synapse_workspace_name = "<synapse_workspace_name>"

# Fabric config
workspace_id = "<workspace_id>"
lakehouse_id = "<lakehouse_id>"
export_folder_name = f"export/{synapse_workspace_name}"
prefix = "" # this prefix is used during import {prefix}{notebook_name}

output_folder = f"abfss://{workspace_id}@onelake.dfs.fabric.microsoft.com/{lakehouse_id}/Files/{expor
```

Export notebooks from Azure Synapse

This will export all notebooks from your Azure Synapse workspace to the indicated `output_folder` in ipynb format

```
sc.addPyFile('https://raw.githubusercontent.com/microsoft/fabric-migration/main/data-engineering/utils/util.py')
from util import *

Utils.export_notebooks(azure_client_id, azure_tenant_id, azure_client_secret, synapse_workspace_name, output_folder)
```

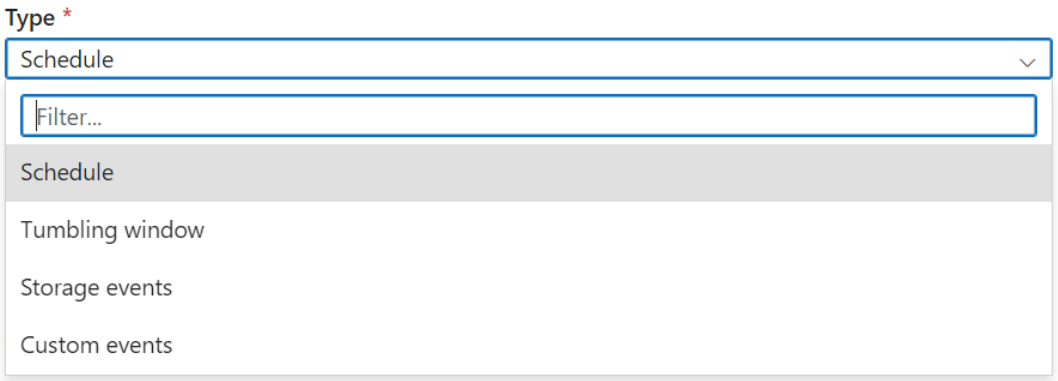
Import notebooks in Fabric

This will import all notebooks from indicated `output_folder` into Fabric

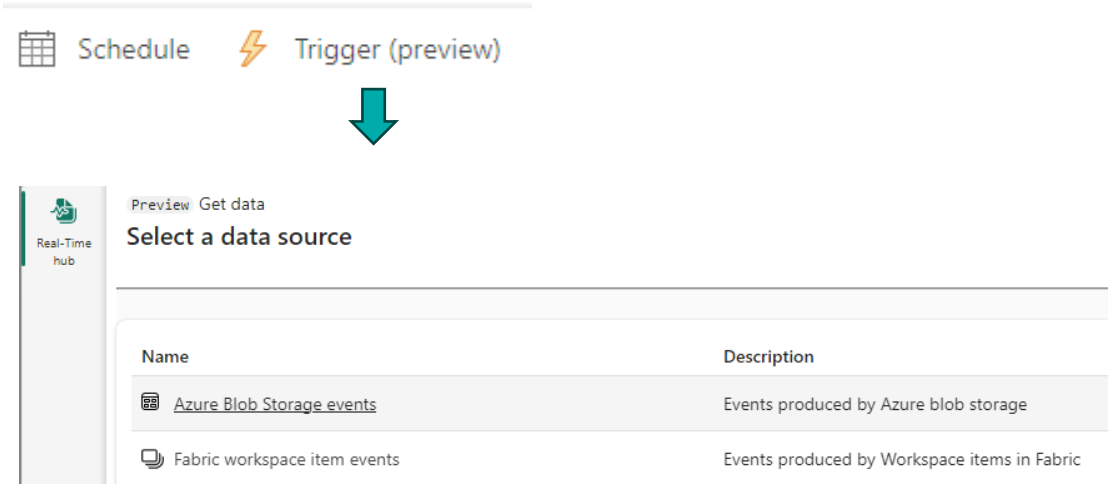
```
Utils.import_notebooks(f"/lakehouse/default/Files/{export_folder_name}", workspace_id, prefix)
```

How to automate execution – triggers ADF vs. Fabric DF

Different types of triggers available – scheduled and events



Scheduled and events in preview



Triggers to execute pipelines based on time or events

- wall-clock time schedule, tumbling window time slices, file-based events, or custom events

- wall-clock time schedule available – part of pipeline artefact
- Event based via Data Activator (file and custom)
- No tumbling window supported

Straightforward approach - rebuild in Fabric

Overview – migration of the different components

Component Azure	Component Fabric	Migration path	Difficulty
Linked Service	Connection	Re-Create	No script support Different scope Not all supported
Pipeline	Data Pipeline	Copy & pPaste of parts	Structural differences in json definition e.g. input/output Different object references GUIDs in Fabric
Mapping Data Flow	Data Flow Gen 2	None Create in Gen2 or convert to notebook	MDF do not exist
Notebook	Notebook	Export / Import	**Works best**
Trigger	Trigger / Data Activitor	Re-Create	Different

Possible migration process

scieneers
DRIVEN BY DATA

1. Record inventory
2. Define target picture
3. Shortcut existing data
4. Create SQL Layer
5. Build transformation processes on shortcut raw data

Testing

6. Load processes for raw data

Testing

7. Switch transformation processes

Testing

8. Final data synchronization between Azure and Fabric

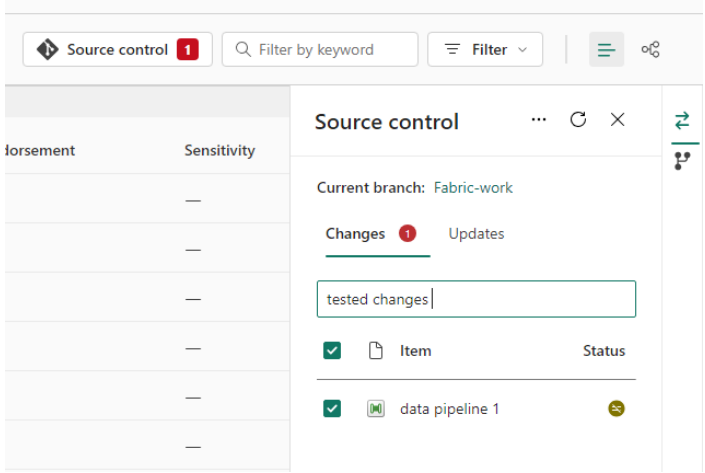
1. Which ADF/ Synapse objects should be migrated?
2. Are there any "legacy objects" that should be removed in the course of the migration?
3. Are there dependencies outside of ADF / Synapse?
   - Business processes pushing data?
   - Mechanisms that trigger synapse processes from outside?

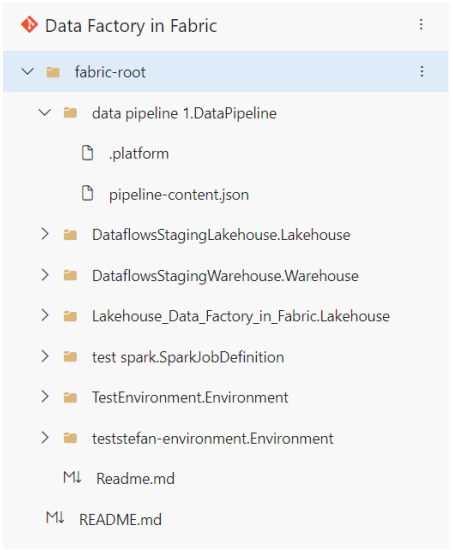
1. Migrating to one or more Fabric Workspaces?
2. Does an access concept need to be revised / adopted?
3. Optimizations to be implemented in the course of the migration?
4. Create architecture diagram
5. Define success criteria / evaluation process

Changes for Application Lifecycle Management?

Teamwork in Fabric Development

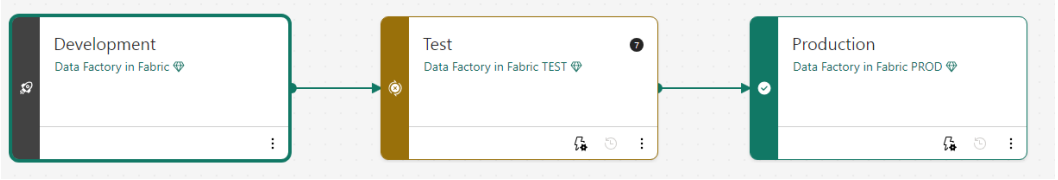
- w/o a repo save commits to the live version (for anyone)
- Nice integration in GUI, making this usable for end users
- Make sure nothing gets lost by using a git repository
  - No auto-commit when saving, go to workspace, source control and commit or get updates

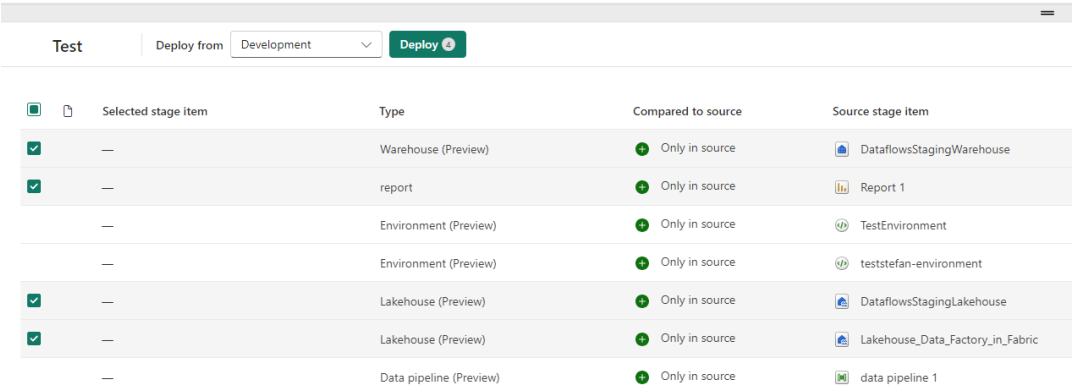


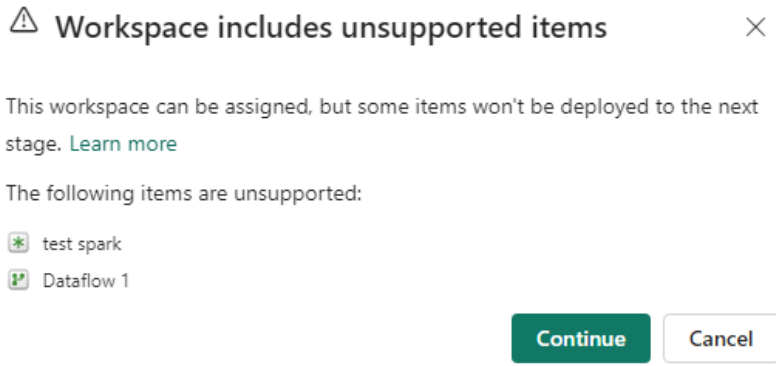


Deployments

- GUI-based Deployment pipelines in Fabric available, comfortable
- 3 workspaces dev, test, prod to be assigned
- Selection of single items possible
- Still not all items supported including **Dataflows Gen2** und **Spark Jobs**
- Difficult to customize e.g. creating folders / copy of config files
- Only deploy from Stage 2 Stage not from a git repo branch
- Just one for all?
- Still buggy…





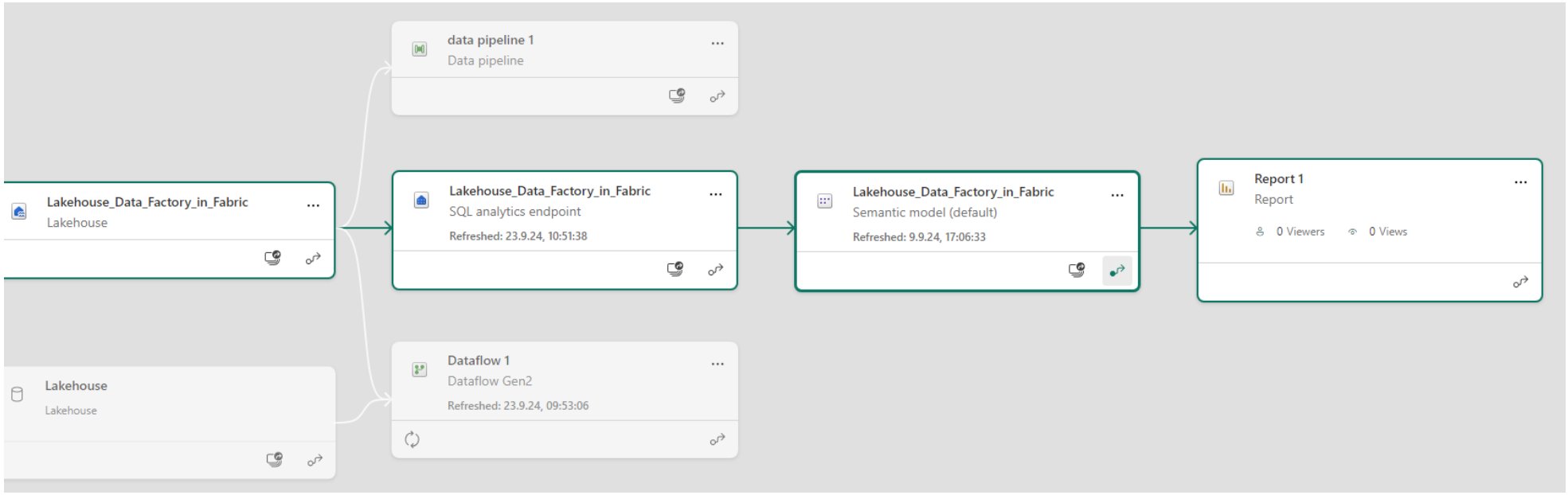


Deployments using Azure Devops / Python

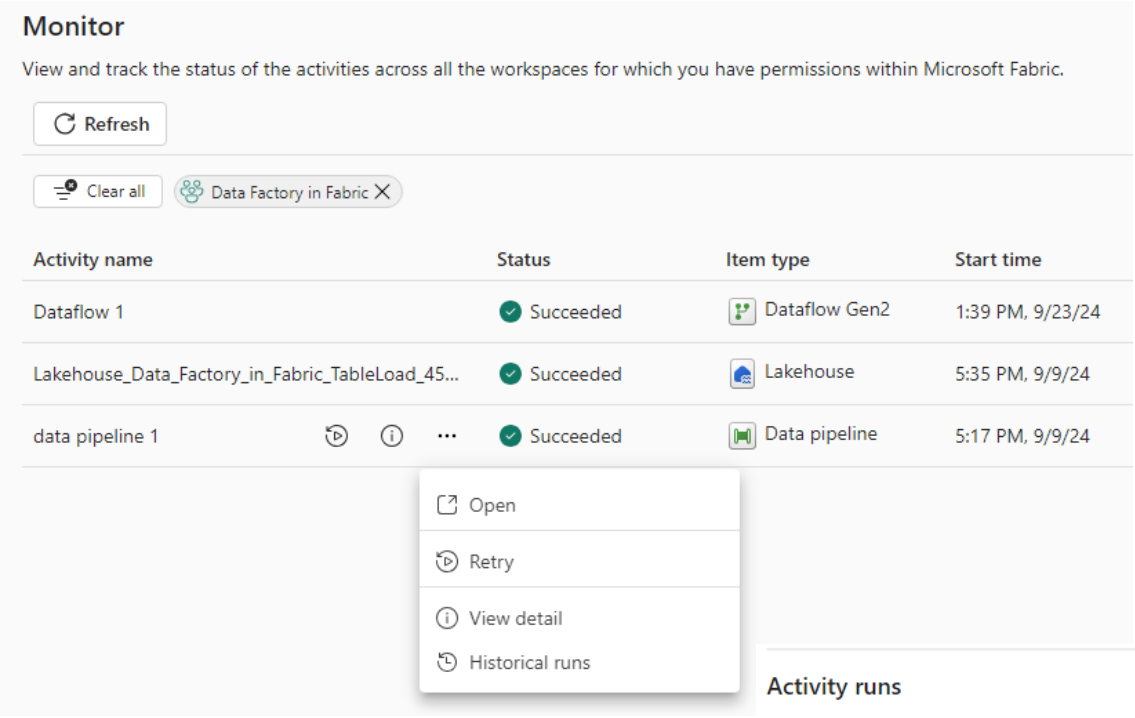
- Use Rest APIs for Fabric with Python, Powershell or whatever in Azure Devops Pipelines
  - https://learn.microsoft.com/en-us/rest/api/fabric/articles/
- Use fabric-cicd
  - Python library designed for use with Microsoft Fabric workspaces
  - code-first Continuous Integration / Continuous Deployment (CI/CD) automations to seamlessly integrate Source Controlled workspaces
  - assist CI/CD developers who prefer not to interact directly with the Microsoft Fabric APIs
  - https://microsoft.github.io/fabric-cicd/latest/

Now as this should be one product – is there also Data Lineage?

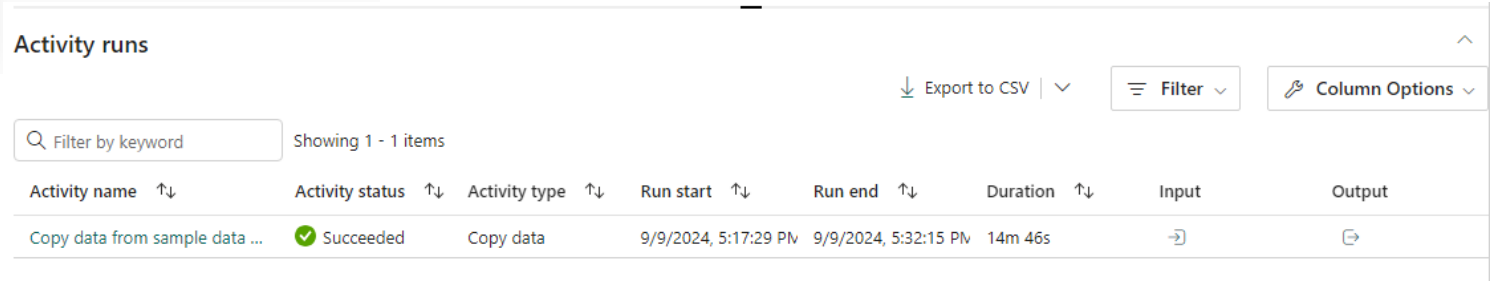
Yes! Maybe more detailed in future



Check state of your processes using the central Monitoring Hub in Fabric



Good overview of the whole solution in a workspace and above different workspaces with dependencies in enterprise context



Summary

Summary

- Currently no automated migration process
- Despite helpful scripts, some manual effort is to be expected
- Migration of Notebooks easiest one, most effort for Mapping Data Flows needed
- Migrating piece by piece is possible and offers advantages
- Running both systems in parallel but independently of each other enables comparison, evaluation and testing
- For complex and big solutions maybe wait for Microsoft to bring up a more sophisticated toolset for migration – or build your own (and share..)