Presentation will start shortly

ilionx
experts in eenvoud

Welcome to the Dutch Fabric User group hosted by ilionx

program

17:00	Walk-in + Dinner
18:00	From Queries to Code: Using Python in Microsoft Fabric By Olivier Sweep
19:00	Short break
19:30	Real-Time Intelligence with Microsoft Fabric By Ramesh Nelluri
20:30- 21:00	Round-up and drinks

ilionx

About your host

ilionx

mission and ambition

mission
creating simplicity in a complex world

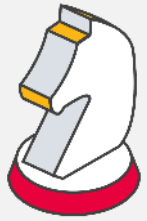
vision
get the best out of IT, move companies and organisations forward, and let people enjoy their work

strategic ambition
the knowledge- and implementation partner in our IT market segment

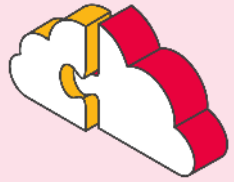
ilionx

your challenges are our expertise

Plan — **Build** — **Run** →



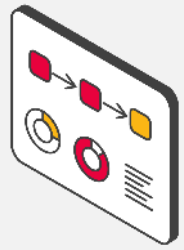
digital strategy



cloud applications

data & AI

hyper-automation



managed services

national coverage

Hoorn

Watergang

Amsterdam

Amersfoort

Woerden

Utrecht

Groningen

Zwolle

Utrecht

Den Bosch

Maastricht

ilionx

From queries to code in Microsoft Fabric

Olivier Sweep

About me

- Working with Power BI and the Azure data stack since 2017
- Loved the Monhtly Power BI updates; every month new shiny features!
- SQL, DAX and (ADF/Synapse) pipelines are well within my comfort zone
- Love helping clients with realising their data ambitions

- osweep@ilionx.com
- +31(0)6 50 03 88 82
- LinkedIn

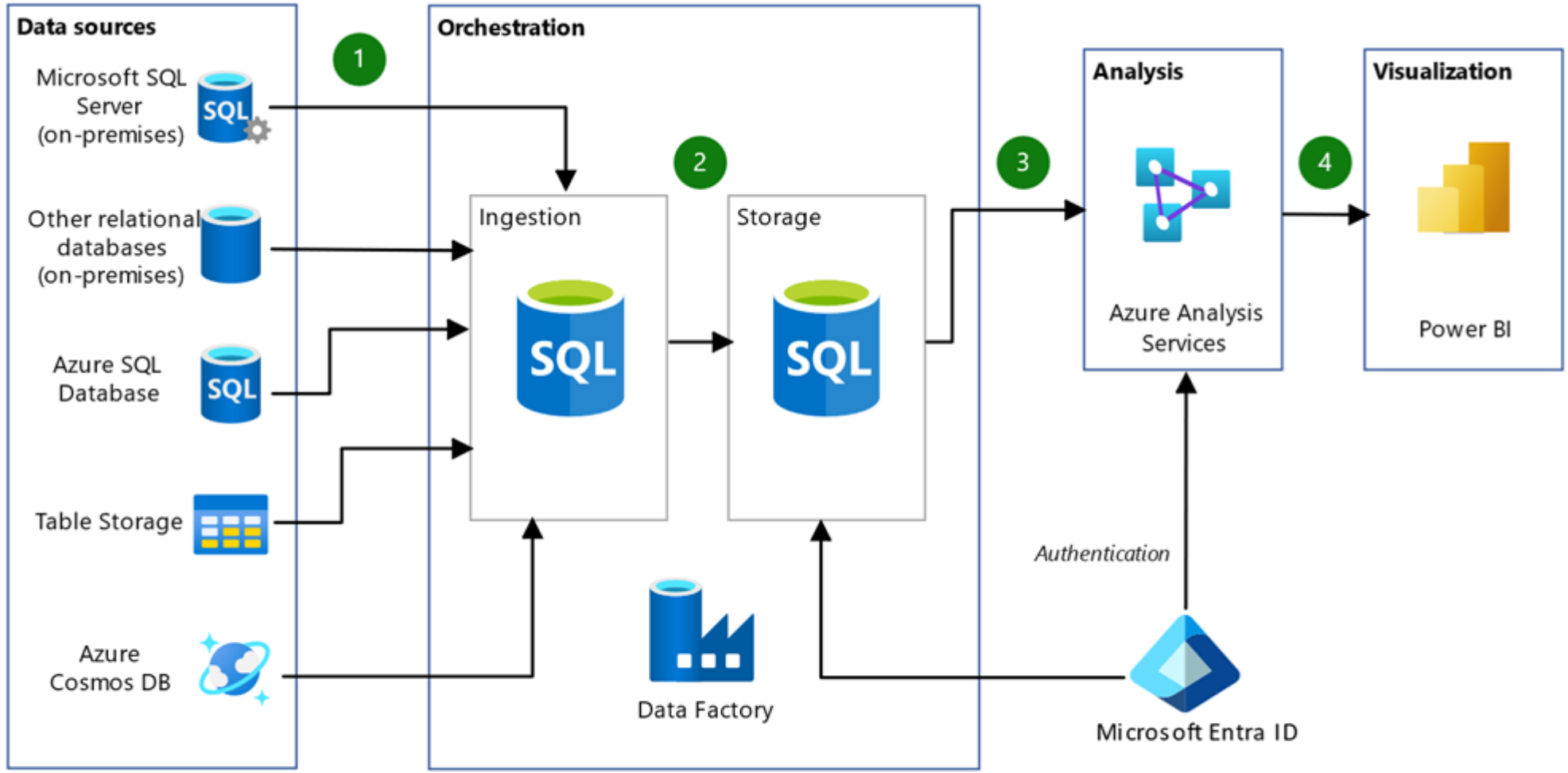
ilionx

Objectives

- What does a programming language bring you?
- Applying some better practices
- Learning some libraries
- Glueing some Fabric

Show of hands

- Most of my experience has been with SQL/DAX/Pipelines (ADF/Synapse/SSIS)
- Am as comfortable working with Python as I am with the ‘traditional’ stack



Microsoft Azure

ilionx

Pros and cons of this traditional approach

- Pros:
  - Tried and proven
  - Works really well for structured (small) data scenario’s
- Cons:
  - Limited to SQL
  - Limited scalability
  - Some limitations in source extraction capabilities, like API’s
  - Always a need for infra as code
  - Situations might ask for extra infrastructure like function apps, spark pools
  - Anyone ever tested their data?

Introducing Microsoft Fabric

- All the tools we know and love plus more
- One license and cost model using Capacity Units (CU)
- No more need for extra Azure infrastructure
- Coding languages available by default (Pyspark, Python, Scala and SparkR)
- Today we will be working in the context of Notebooks

Demo time

Acces to coding languages by default

- Provides a lot of flexibility
- Without headaches about licenses, security and integration
- For every problem there is a library
  - For your own problems, you can build your own!
- Less dependancies on low-code integration tools

The addition of python to your stack brings you

- Programming paradigms like OOP or FP
- Some ‘glue’ inside of Fabric
- More flexibility when working with Rest API’s
- Testing code and semantic models
- PySpark for handling data at scale

The addition of python to your stack brings you

- Programming paradigms like OOP or FP
- Some ‘glue’ inside of Fabric
- More flexibility when working with Rest API’s
- Testing code and semantic models
- PySpark for handling data at scale

Start coding in Python

- Learning PySpark coming from T-SQL is very do-able!
- Python can not only Transform data but also:
  - Extract
  - Load
  - Document
  - Test
  - Add error handling
  - And more!
- Apply the DRY principle (Don’t repeat yourself)

In this demo

- Load, transform and write some data
- Vacuum a delta table

Demo time

Try to avoid..

- Problems:
  - Repeated code
  - Hardcoded values
  - DRY principle?

```
1  from delta.tables import DeltaTable
2  # Welcome to your new notebook
3  # Type here in the cell editor to add code!
4  display(spark.read.format("Delta").load("abfss://f1db2fd4-9837-42de-bbbd-0d47c3a288ed@onelake.dfs.fabric.microsoft.com/bdab4126-fc84-4051-8bf9-995342c8d369/Tables/taxidata"))
```

- Command executed in 1 sec 970 ms by Olivier Sweep on 4:11:26 PM, 3/11/25

```
1  # Create DeltaTable object
2  delta_table = DeltaTable.forPath(spark,"abfss://f1db2fd4-9837-42de-bbbd-0d47c3a288ed@onelake.dfs.fabric.microsoft.com/bdab4126-fc84-4051-8bf9-995342c8d369/Tables/taxidata" )
3  # Run VACUUM with a retention period of 7 days
4  delta_table.vacuum(retentionHours=168)
```

[2] ✓ - Command executed in 11 sec 442 ms by Olivier Sweep on 4:12:04 PM, 3/11/25

ilionx

Functions make your code better

- Better **reusability**
  - Don’t repeat yourself!
- Better **modularity**
  - Easier to update
- Better **readability**

A function deconstructed

1 2 3

```python
def fnc_get_abfss_path(name:str)->str:
    """
    Generates the ABFSS path for a Delta table in the current Microsoft Fabric workspace and lakehouse.

    This function retrieves the workspace ID and the first available lakehouse ID in the environment,
    then constructs the full ABFSS path to access a specified Delta table.
4
    :param name: The name of the Delta table.
    :return: The ABFSS path to the specified Delta table as a string.

    Example:
        >>> fnc_get_abfss_path("sales_data")
        'abfss://<workspace_id>@onelake.dfs.fabric.microsoft.com/<lakehouse_id>/Tables/sales_data'

    Note:
    - The function assumes that at least one lakehouse exists in the workspace.
    - If multiple lakehouses exist, it selects the first one returned by `fabric.list_items("Lakehouse")`.
    """
    workspace_id = fabric.get_workspace_id()
    lakehouse_id = fabric.list_items("Lakehouse")["Id"][0]
5   return f"abfss://{workspace_id}@onelake.dfs.fabric.microsoft.com/{lakehouse_id}/Tables/{name}"
```

1 = Function name

2 = Parameter:input type

3 = Output type

4 = Description

5 = Return value

ilionx

When your project grows

- The number of functions could grow
- How do you keep your code nice and structured?

You can write a Class

- A function within a class is called a method
- You can store data inside of an object
- Makes future extensions easier
- Enables:
  - Abstraction
  - Inheritence
- Storing your Classes and functions inside of another Notebook may help you


Demo time

Demo take aways

- Problem:
  - Messy and redundant code
  - Hard to re-use code
- Solution:
  - Created class with several methods to do several operations on the Delta Table
  - Improved readability and maintainability by moving to seperate Notebook
- Note:
  - Don’t over complicate! YAGNI – "You Ain't Gonna Need It"

Creating a class for everything sounds like a lot of work

- You don’t have to think of every solution yourself
- Import libraries to do the heavy lifting for you

Three kinds of libraries in Fabric

- **Default Libraries**
  - List these libraries using %pip list in a Fabric notebook.
  - Load them with the `import` keyword
  - 400+ in the default runtime
- **Public Libraries**
  - Use `%pip install` package-name to install a library within a notebook session.
  - Installed libraries in a session are temporary and will be lost after restart.
  - Shared libraries are managed through environments
- **Custom Libraries**
  - You can upload and install custom libraries in .whl, .tar.gz, and .jar formats to a custom environment
- **Special mentions**
  - Re-use code by running a Notebook inside another Notebook or using notebook resources
  - Spark job definitions

Some good libraries to start your journey

Library	Use case
Semantic-link-labs or sempy	Fabric ‘glue’: Manage semantic models, Use the Fabric REST-API’s
Notebookutils	Utilities package for: filesystems, notebooks, credentials, session management and more
Pyspark	SQL and dataframes
Delta.tables	Delta table management
Json, jmespath	Basic or more complex json parsing
Requests, Oath2	Basic HTTP requests to get data from API’s, easy authentication
Great expectations	Data validation, testing and documentation

Demo time

Some good libraries to start your journey

- Experienced some notebook utilities
- Worked with sempy
- Got information from the Fabric REST API

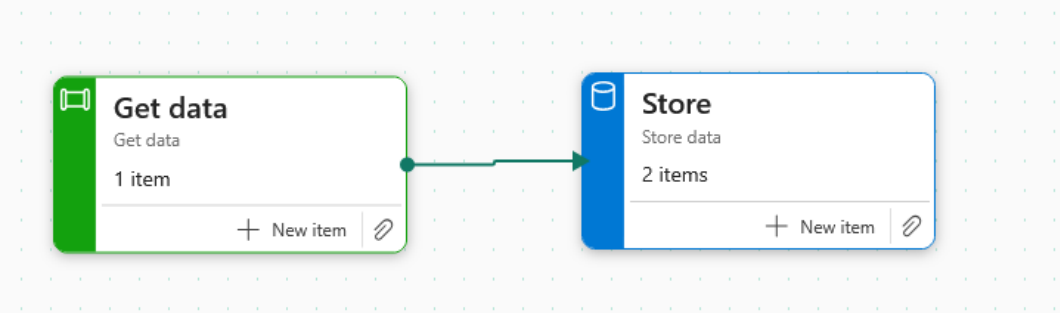
The glue within Fabric

The set up for a “classic approach”:

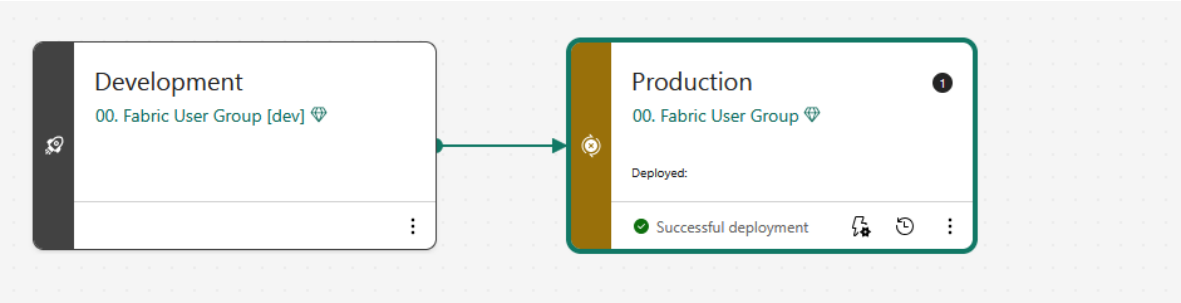
- A Warehouse
- A Data Pipeline
- A Deployment Pipeline

The glue within Fabric

The set up:



Name	Type	Task	Owner
DWH	Warehouse	Store	Olivier Sweep
DWH	Semantic model (default)	Store	00. Fabric User Group [dev]
PL_src_to_dwh	Data pipeline	Get data	Olivier Sweep




Demo time

The glue within Fabric

Problem:

- No deployment rules for pipelines (yet)

Solution:

- Sempy for calling the warehouse REST API
- Extracting the relevant information
- Specifying an output value
- Passing it to the pipelines dynamically

Item	Data source rule	Parameter rule	Default lakehouse rule	Details
Dataflow	✓	✓	✗	Use to determine the values of the data sources or parameters for a specific dataflow.
Semantic model	✓	✓	✗	Use to determine the values of the data sources or parameters for a specific semantic model.
Datamart	✓	✓	✗	Use to determine the values of the data sources or parameters for a specific datamart.
Paginated report	✓	✗	✗	Defined for the data sources of each paginated report. Use to determine the data sources of the paginated report.
Mirrored database	✓	✗	✗	Defined for the data sources of each mirrored database.
Notebook	✗	✗	✓	Use to determine the default lakehouse for a specific notebook.

To summarize

- Learn some good habits when working with Python
- Break up your code in reuseable functions or Classes when needed
- Find a balance between DRY and YAGNI
- Explore libraries, they make life easy
- Have fun learning!

thank you