



Data Saturday #1

February 27th, 2021

Harnessing Azure Synapse Analytics Serverless SQL Pools in Power BI Dataflows

Andy Cutler





Andy Cutler

Independent Consultant & Contractor

Azure Data Platform & Power BI

www.datahai.co.uk

<https://twitter.com/MrAndyCutler>

<https://www.linkedin.com/in/andycutler/>



SCAN ME

 datahai



Session Overview

- **Who** is this session for?
- **Why** are we connecting Power BI to Synapse?
- What are **Power BI Dataflows**?
- What is **Azure Synapse Analytics Serverless SQL Pools**?
- Why use **Serverless SQL Pools with Dataflows**?
- Serverless SQL Pools and Power BI Dataflows **Together**
- Power BI **Query Folding**

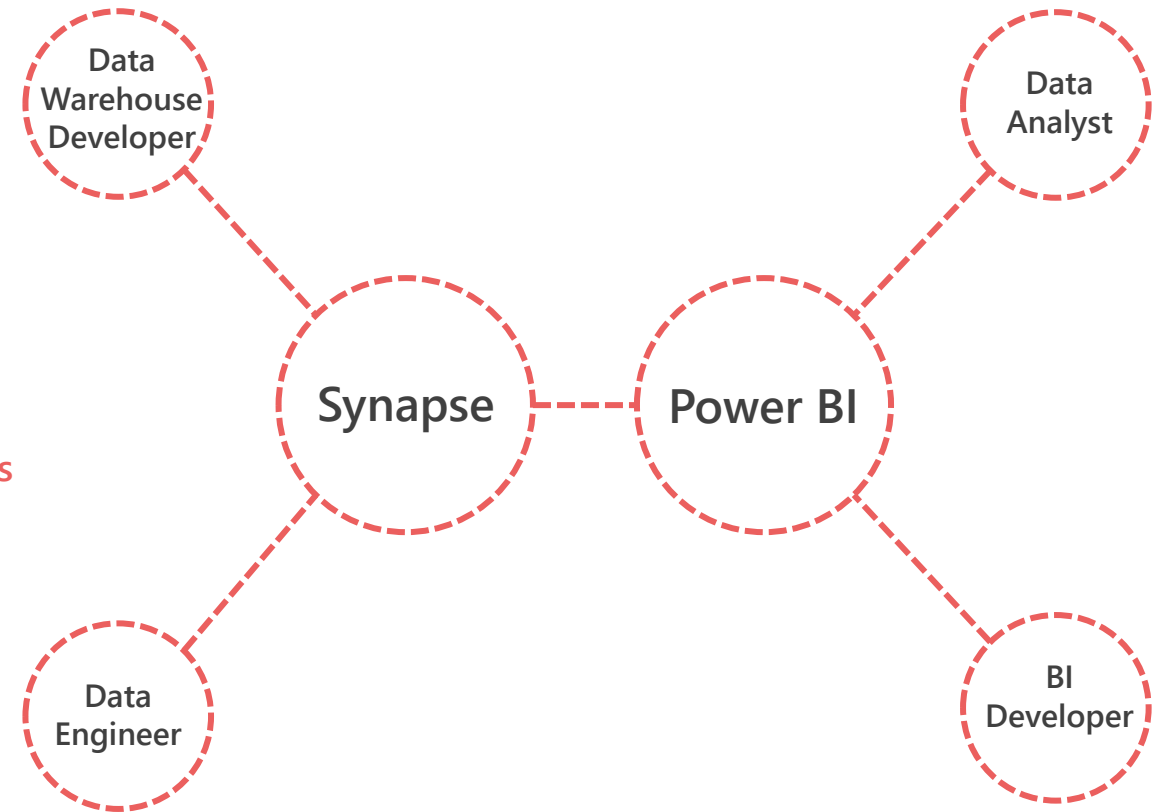


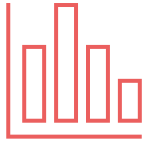


Who is this session for?

Depending on the organisation, the following roles may be carried out by individual teams or a single person.

- Data Warehouse Developer
 - Building Data Warehouse solutions using Synapse Analytics
- Data Engineer
 - Extracting, Loading and Transforming data from many sources
- Business Intelligence Developer
 - Creating centralised data sets and data models
- Data Analyst
 - Creating analyses for an organisation



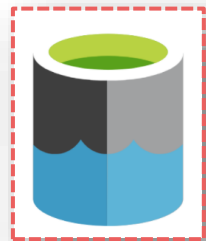
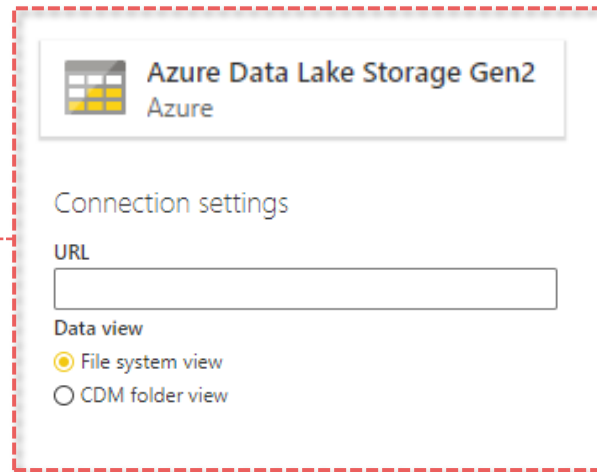


Why Are We Connecting Power BI to Synapse?

Power BI already has a native Azure Data Lake Gen2 connector...why would we want to connect to Synapse Analytics and use the Serverless SQL engine?

Azure Storage

Power BI

- “Out of the box” connector readily available
- Power BI can connect to any level in the Azure Data Lake Gen2 folder hierarchy
- Power BI can recursively load all file data located within the folder hierarchy

However...

- If the size of the data grows or required transformations affect performance...Serverless SQL can take the heavy-lifting away from Power BI.



Power BI Dataflows

A **Power Query** based feature in the **Power BI Service** which enables:

- **Connecting** to a variety of data sources including SQL Databases and Data Lake Storage
- **Cleansing** and **Transforming** the data to suit requirements
- **Mapping** to common business entities using the **Common Data Model**
- Creating a **centralised repository** of data ready for using in Power BI Datasets

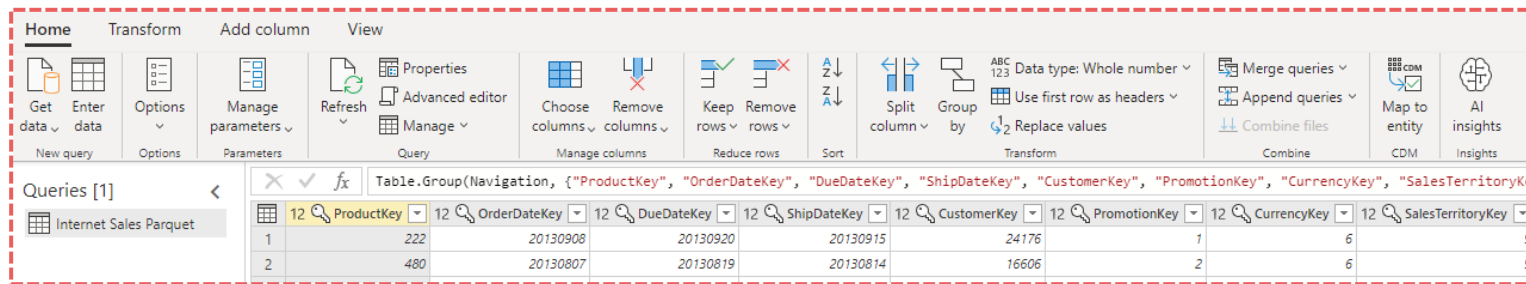
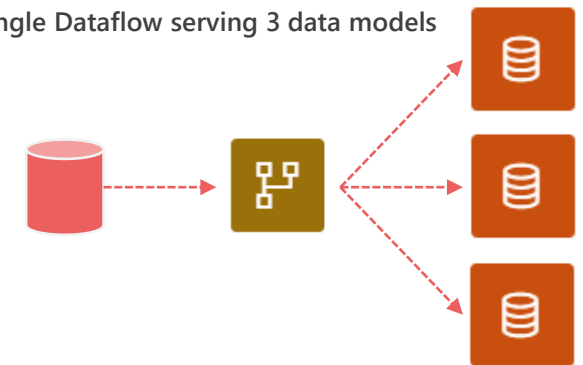


Table: Group(Navigation, {"ProductKey", "OrderDateKey", "DueDateKey", "ShipDateKey", "CustomerKey", "PromotionKey", "CurrencyKey", "SalesTerritoryKey"})

	ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey	PromotionKey	CurrencyKey	SalesTerritoryKey
1	222	20130908	20130920	20130915	24176	1	6	9
2	480	20130807	20130819	20130814	16606	2	6	9

Single Dataflow serving 3 data models





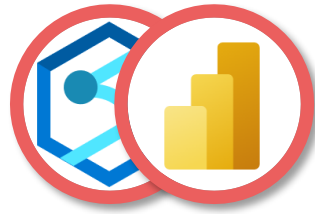
Azure Synapse Analytics Serverless SQL Pools

Cloud Analytics platform to **process data in data lake storage**

- Ability to **query file data "in place"** using T-SQL without copying data to internal storage
- Ability to **write data** to external data lake storage
- Connect **Business Intelligence tools** using SQL endpoint


```
SELECT * FROM
OPENROWSET
(
    BULK 'conformed/factsalesorderheader/**/*.*',
    DATA_SOURCE = 'ExternalDataSourceDataWarehouse',
    FORMAT = 'parquet'
) as fctsl
```

```
SELECT * FROM
OPENROWSET
(
    BULK 'conformed/factsalesorderheader/**/*.*',
    DATA_SOURCE = 'ExternalDataSourceDataWarehouse',
    FORMAT = 'parquet'
) as fctsl
WHERE fctsl.filepath(1) = 2020
AND fctsl.filepath(2) = 7
AND fctsl.filepath(3) = 6
```



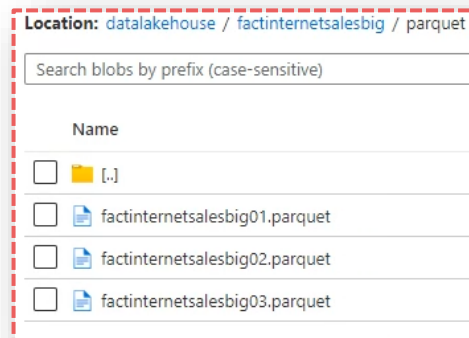
Serverless SQL Pools and Dataflows Together

Power BI can connect to the Serverless SQL endpoint just like any SQL database



Dedicated SQL endpoint	: synapsedemodh.sql.azure.synapse.net
Serverless SQL endpoint	: synapsedemodh-ondemand.sql.azure.synapse.net
Development endpoint	: https://synapsedemodh.dev.azure.synapse.net

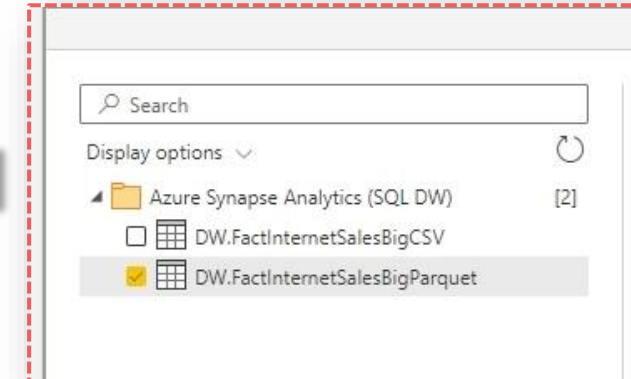
Source data in Data Lake



Create table to read data from Data Lake

```
CREATE EXTERNAL TABLE DW.FactInternetSalesBigParquet (
  ProductKey INT,
  OrderDateKey INT,
  DueDateKey INT,
  ShipDateKey INT,
  CustomerKey INT,
  PromotionKey INT,
  CurrencyKey INT,
  SalesTerritoryKey INT,
  ....(all other columns)
)
WITH (
  LOCATION = '/factinternetsalesbig/parquet',
  DATA_SOURCE = ExternalDataSourceDataLake,
  FILE_FORMAT = SynapseParquetFormat
);
```

Connect to table within Power BI Dataflow



- If you have **CSV, Parquet or JSON** files in Storage, let SQL Serverless do the data processing
- Can create a Dataflow and use **Serverless SQL Pool Database and Tables** as a data source
- SQL is pushed down to SQL Serverless due to Power Query's **Query Folding** feature



Azure Synapse Analytics

Connect to



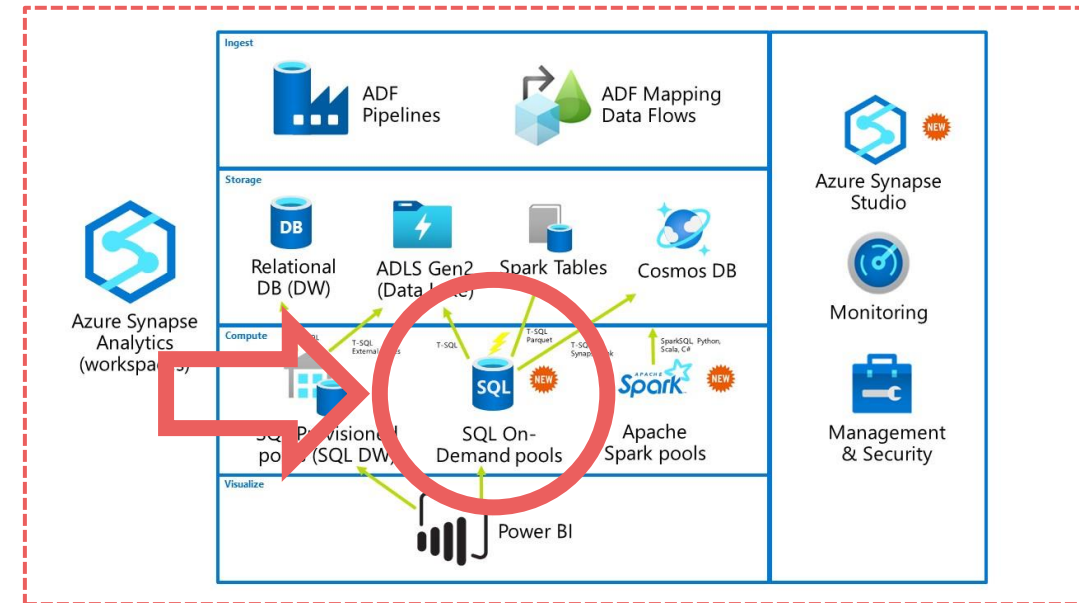
Built-in

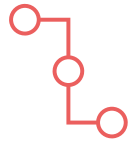


In this scenario, we should think of Synapse Analytics as a **processing engine**.

When an Azure Synapse Analytics resource is created, a **Serverless SQL Pool is created automatically** and is available immediately for use.

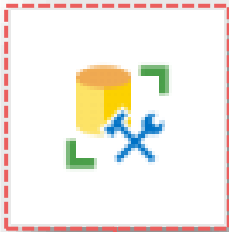
- We **do not need** to provision a Dedicated SQL Pool
- We **do not need** to provision Spark pools
- We **do not need** to integrate Power BI workspaces
- We **do not need** to create pipelines





Connecting to the Serverless SQL Endpoint

SSMS



Azure Data Studio



Power BI



Python / dotNet

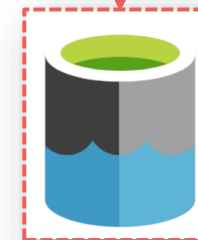
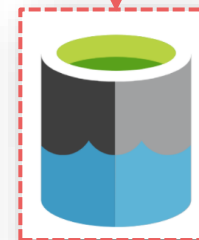
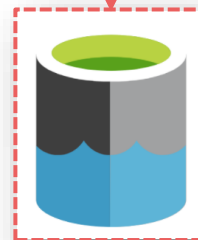
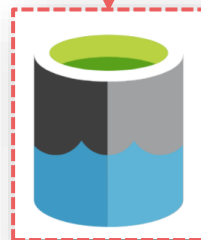
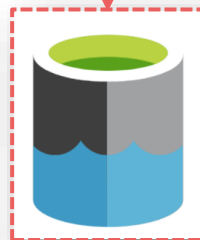


As Serverless SQL Pools exposes a **SQL endpoint**, there are many options in terms of connecting from various software.

In the context of using with Power BI, a Data Analyst/BI Developer could connect to the Serverless SQL Pool via a SQL client tool to perform ad-hoc analysis before creating a Power BI artifact.

Serverless SQL endpoint : `-ondemand.sql.azuresynapse.net`

Serverless SQL



Azure Storage &
Data Lake Gen2

Power BI Query Folding

When transformations are applied to the dataset and can be “folded”, the logic to perform the transformations is passed to the data source. In this scenarios, Serverless SQL Pool will receive a SQL statement.

The GROUP BY transformation in Dataflows

Group by

Specify the column to group by and the desired output.

☐ Basic ☒ Advanced

Group by

ProductKey

OrderDateKey

DueDateKey

ShipDateKey

CustomerKey

PromotionKey

CurrencyKey

SalesTerritoryKey

Add grouping

New column name	Operation	Column	
OrderQuantity	Sum	ProductKey	...
	Count rows		
Add aggregation			

☐ Use fuzzy grouping

> Fuzzy group options

OK

Cancel

The SQL generated by the transformation which will be sent to the Serverless SQL Pool (Query Folding)

fx

Table.Group(Navigation, {"ProductKey", "OrderDateKey", "DueDateKey", "ShipDateKey", "CustomerKey", "PromotionKey", "CurrencyKey", "SalesTerritoryKey"}, {"{"Order

	12	12	12	12	12	12	12	12	12	12	1.2
	OrderDa...	DueDa...	ShipDa...	Custom...	Promoti...	Curren...	SalesTerrito...	OrderQuantity	SalesAmount		
1	182	20121228	20130109	20130104	12132	1	100	7	3	104.97	
2	182	20121228	20130109	20130104	16313	1	100	8	3	104.97	
3	182	20121229	20130110	20130105	11241	1	100	7	3	104.97	
4	182									104.97	
5	182									104.97	
6	182									104.97	
7	182									104.97	
8	182									104.97	
9	182									104.97	
10	182									104.97	
11	182									104.97	
12	182									104.97	
13	182									104.97	
14	182									104.97	
15	182									104.97	
16	182									104.97	
17	182									104.97	
18	182									104.97	
19	182									104.97	
20	182									104.97	
21	182									104.97	
22	182									104.97	
23	182									104.97	
24	182	20130112	20130124	20130119	18218	1	100	1	3	104.97	
25	182	20130112	20130124	20130119	18217	1	6	9	3	104.97	
26	182	20130113	20130124	20130119	21488	1	100	1	3	104.97	

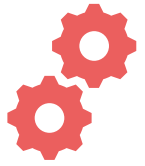
Data source query

```

select [rows].[ProductKey] as [ProductKey],
[rows].[OrderDateKey] as [OrderDateKey],
[rows].[DueDateKey] as [DueDateKey],
[rows].[ShipDateKey] as [ShipDateKey],
[rows].[CustomerKey] as [CustomerKey],
[rows].[PromotionKey] as [PromotionKey],
[rows].[CurrencyKey] as [CurrencyKey],
[rows].[SalesTerritoryKey] as [SalesTerritoryKey],
sum([rows].[OrderQuantity]) as [OrderQuantity],
sum([rows].[SalesAmount]) as [SalesAmount],
sum([rows].[TaxAmt]) as [TaxAmount],
sum([rows].[Freight]) as [FreightAmount]
from
(
  select [ProductKey],
    [OrderDateKey],
    [DueDateKey]

```

OK



Power BI Query Folding

Not all transformations can be “folded” and therefore not pushed back down to the source data engine. In this instance a transformation within the Power BI Dataflow has caused the query to no longer “fold”.

Foldable Query – indicated by the green label

Query settings

Properties

Name

DW FactInternetSalesSmallCSV

Entity type ⓘ

Custom

Applied steps

Source

Navigation

Grouped rows

Non-Foldable Query – indicated by the orange label

Query settings

Properties

Name

DW FactInternetSalesSmallCSV

Entity type ⓘ

Custom

Applied steps

Source

Navigation

Split column by position

ABC 123 Changed column type

Split column by position 1

Lowercased text



Supported File Formats

CSV, Parquet and JSON file formats are supported by Serverless SQL stored in Azure Storage and Azure Data Lake Gen2

CSV

```
SELECT cust.*
FROM OPENROWSET(
    BULK 'sourcedata/*.csv',
    DATA_SOURCE = 'ExternalDataSourceDataWarehouse',
    FORMAT = 'CSV',
    PARSER_VERSION = '2.0',
    HEADER_ROW = TRUE,
    FIELDTERMINATOR = '|'
) WITH
(
    ProductKey INT 1,
    CustomerKey INT 5
) AS cust
```

Support for files delimited by a range of separators and line terminators.

Simple file construct.

Serverless SQL will **read the entire** CSV file.

Ordinal Position 1

Ordinal Position 2

ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey
214	20121228	20130109	20130104	12132
214	20121228	20130109	20130104	16313
214	20121229	20130110	20130105	11241
214	20121229	20130110	20130105	12390
214	20121230	20130111	20130106	11338
214	20121230	20130111	20130106	24604
214	20121231	20130112	20130107	11061

Parquet

```
SELECT *
FROM
    OPENROWSET(
        BULK '/sourcedata/*.parquet',
        DATA_SOURCE = 'ExternalDataSourceDataWarehouse',
        FORMAT = 'PARQUET'
    ) WITH
    (
        ProductKey INT,
        CustomerKey INT
    )
AS fact
```

Compressed file format which contains:

- Data
- Schema
- Statistics

Serverless SQL will **only read the data necessary** to support query

Named Column

Named Column

ProductKey	OrderDateKey	DueDateKey	ShipDateKey	CustomerKey
214	20121228	20130109	20130104	12132
214	20121228	20130109	20130104	16313
214	20121229	20130110	20130105	11241
214	20121229	20130110	20130105	12390
214	20121230	20130111	20130106	11338
214	20121230	20130111	20130106	24604
214	20121231	20130112	20130107	11061



Which SQL Objects Can Power BI Connect To?

The Serverless SQL engine exposes a SQL endpoint which will allow Power BI to connect and interact with the following supported objects:

Object	Notes
Table	An External Table which points to a folder location within Azure Storage / Data Lake Gen 2
View	A View which can point to specific folders locations within a Data Lake and also allow "partition pruning"
Stored Procedure	Conditional logic support but has limitations around "query folding"
Native SQL	A SQL statement can be written and used to retrieve data but has similar limitation to a stored procedure with regards to "query folding"






Query Foldable

Query Not Foldable





Dataflow Incremental Refresh

ENTITY NAME	ENTITY TYPE	ACTIONS
 Internet Sales CSV	Custom	   



Incremental refresh settings

Internet Sales CSV

Incremental refresh updates only data that's changed, to speed refresh, reduce capacity usage, and store historic data. [Learn more](#)

☒ On

Choose a DateTime field to filter by *

Choose a field ▼

Store rows from the past *

Choose a time period ▼

Refresh rows from the past *

Choose a time period ▼

☐ Detect data changes [Learn more](#)

Only refresh data if the maximum value in this field changes

Choose a field ▼

☐ Only refresh complete periods [Learn more](#)

When you save these settings, data from the past [storage period] will be loaded to your dataflow storage the next time this dataflow is refreshed. Subsequent refreshes will update only data that's changed in the past [refresh period].

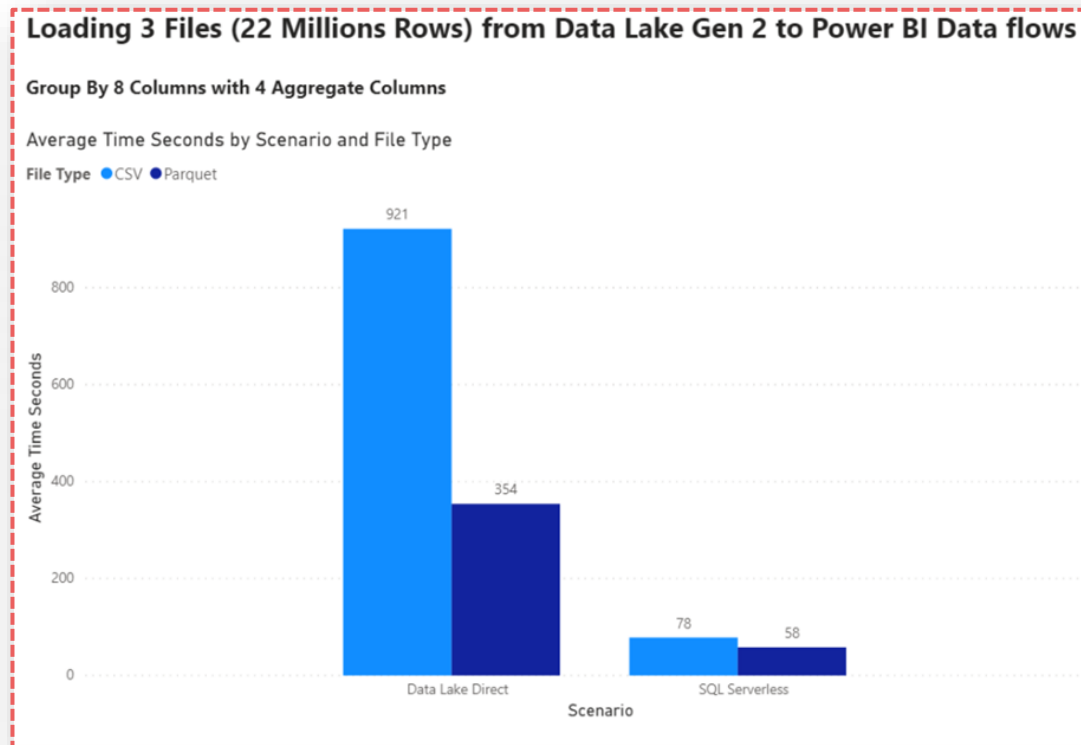
If there is a **DateTime** column within the data then we are able to enable the **Incremental Refresh** (Premium feature in Dataflows) and Power BI Desktop (Pro feature).

- **External Tables:** This will filter the data based on the table location in storage, scans all files.
- **Views:** This will filter the data based on the location in storage with the option of **partition pruning** using the `filepath()` function within Serverless SQL.



Data Loading Performance Analysis

Comparing the performance of Serverless SQL Pools and the Power BI native Data Lake connector



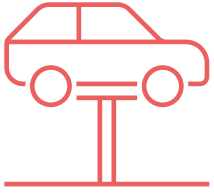
The data loading tests were carried out on **CSV** and **Parquet** data.

Total CSV file size: **4.5 GB**

Total Parquet file size: **1.5 GB**

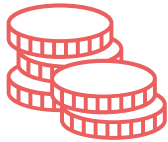
Pushing the data transformations down to the Serverless SQL Pool has reduced the time taken to load the Power BI Dataflow.

Your results may vary!



Considerations

Cost

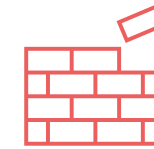


SQL Serverless does cost money!
£4.659 per 1 Terabyte (1TB) of Data Processed

When developing/testing, use a smaller file or set of smaller files

Data at rest does not necessarily translate directly into **data processed**

Infrastructure

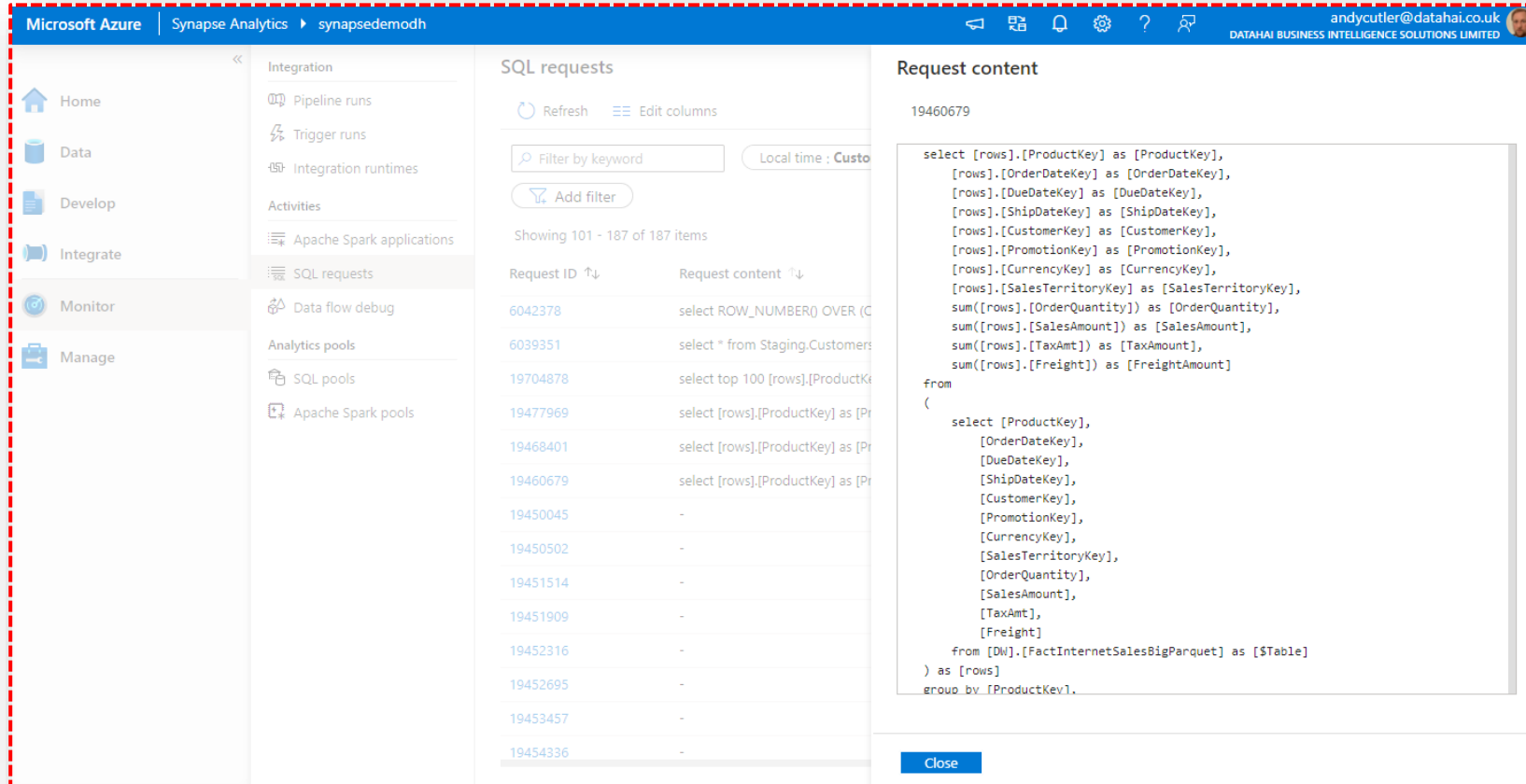


Adds **another service into a data architecture** which will need managing

However, you can use Synapse Analytics SQL Serverless as a **processing engine without any data warehousing**



Serverless SQL Pools Monitoring



The screenshot shows the Microsoft Azure Synapse Analytics interface. The left sidebar contains navigation options: Home, Data, Develop, Integrate, Monitor, and Manage. The 'Monitor' section is selected, showing a list of SQL requests. The 'SQL requests' table displays columns for Request ID and Request content. The 'Request content' pane on the right shows the SQL query for request ID 19460679.

Request ID	Request content
6042378	select ROW_NUMBER() OVER (
6039351	select * from Staging.Customers
19704878	select top 100 [rows].[ProductKey]
19477969	select [rows].[ProductKey] as [Pr
19468401	select [rows].[ProductKey] as [Pr
19460679	select [rows].[ProductKey] as [Pr
19450045	-
19450502	-
19451514	-
19451909	-
19452316	-
19452695	-
19453457	-
19454336	-

```

select [rows].[ProductKey] as [ProductKey],
[rows].[OrderDateKey] as [OrderDateKey],
[rows].[DueDateKey] as [DueDateKey],
[rows].[ShipDateKey] as [ShipDateKey],
[rows].[CustomerKey] as [CustomerKey],
[rows].[PromotionKey] as [PromotionKey],
[rows].[CurrencyKey] as [CurrencyKey],
[rows].[SalesTerritoryKey] as [SalesTerritoryKey],
sum([rows].[OrderQuantity]) as [OrderQuantity],
sum([rows].[SalesAmount]) as [SalesAmount],
sum([rows].[TaxAmt]) as [TaxAmount],
sum([rows].[Freight]) as [FreightAmount]
from
(
select [ProductKey],
[OrderDateKey],
[DueDateKey],
[ShipDateKey],
[CustomerKey],
[PromotionKey],
[CurrencyKey],
[SalesTerritoryKey],
[OrderQuantity],
[SalesAmount],
[TaxAmt],
[Freight]
from [DW].[FactInternetSalesBigParquet] as [$Table]
) as [rows]
group by [ProductKey].
    
```

The SQL generated by Power BI can be seen in the **Monitor** area of Synapse Analytics Studio.

The statistics include the query time and also the **Data Processed** amount, which is a vital statistic.



References

<https://www.datahai.co.uk/power-bi/harnessing-azure-synapse-analytics-sql-serverless-in-power-bi-dataflows/>

